

# Coleção de Exemplos e definições de FUNÇÕES VB 5

**Este trabalho foi feito por:**

**Carlos Alberto Nunes Susviela**

Da Seguinte forma:

Abrido o HELP do VB Selecionando o Texto e Colando no WORD

**POWER Informática – Santana do Livramento – RS**

**Fone: ( 0 5 5 ) 2 4 2 – 5 4 2 7**

**Visite o Site da Power Informática lá você encontra outras **APOSTILAS VB****

### Exemplo da diretiva #Const

Este exemplo utiliza a diretiva **#Const** para declarar constantes de compilação condicional para uso em construções **#If...#Else...#End If**.

```
#Const DebugVersion = 1 ' Avaliará verdadeiro no bloco #If.
```

### Exemplo da construção #If...Then...#Else

Este exemplo faz referência a constantes de compilação condicional em uma construção **#If...Then...#Else** para determinar se certas instruções serão compiladas.

```
' Se Mac avalia como verdadeiro, as instruções que se seguem a #If também o
fazem.
#If Mac Then
    '. Insere aqui exclusivamente instruções Mac.
    ':
    ':
' Caso contrário, se for um programa do Windows de 32 bits, faz o seguinte:
#ElseIf Win32 Then
    '. Insere aqui exclusivamente instruções do Windows de 32 bits.
    ':
    ':
' Caso contrário, se não for uma coisa nem outra, faz o seguinte:
#Else
    '. Insere aqui outras instruções de plataforma.
    ':
    ':
#End If
```

## Diretiva #Const

Utilizada para definir constantes condicionais do compilador no Visual Basic.

### Sintaxe

**#Const** *nomedaconst* = *expressão*

A sintaxe da diretiva de compilador **#Const** possui estas partes:

Parte	Descrição
<i>nomedaconstante</i>	Obrigatória; <b>Variant (String)</b> . Nome da <u>constante</u> ; obedece as convenções padronizadas para nomenclatura de <u>variável</u> .
<i>expressão</i>	Obrigatória. Literal, outra constante condicional do compilador, ou qualquer combinação que inclua algum ou todos os operadores aritméticos ou lógicos, exceto <b>Is</b> .

### Comentários

As constantes condicionais do compilador são sempre **Private** para o módulo no qual aparecem. Não é possível criar constantes do compilador **Public** utilizando a diretiva **#Const**. Constantes do compilador **Public** só podem ser criadas na interface do usuário.

Somente constantes condicionais do compilador e literais podem ser utilizadas em *expressão*. A utilização de uma constante padrão definida com **Const** ou de uma constante indefinida causará a ocorrência de um erro. Contrariamente, as constantes definidas utilizando-se a palavra-chave #Const podem ser utilizadas somente para compilação condicional.

As constantes condicionais do compilador são sempre avaliadas no nível do módulo, independentemente da sua posição no código.

## Diretiva #If...Then...#Else

Compila condicionalmente blocos selecionados do código do Visual Basic.

### Sintaxe

```
#If expressão Then
    instruções
[#Elseif expressão-n Then
    [instruçõeselseif]]
[#Else
    [instruçõeselse]]
#End If
```

A sintaxe da diretiva **#If...Then...#Else** possui estas partes:

Parte	Descrição
<i>expressão</i>	Obrigatória. Qualquer <u>expressão</u> , consistindo exclusivamente de uma ou mais <u>constantes condicionais do compilador</u> , literais e operadores que são avaliados como <b>True</b> ou <b>False</b> .
<i>instruções</i>	Obrigatória. Linhas de programa ou diretivas de compilador do Visual Basic que são avaliadas se a expressão associada for <b>True</b> .
<i>expressão-n</i>	Opcional. Qualquer expressão, consistindo exclusivamente de uma ou mais constantes condicionais do compilador, literais e operadores que são avaliados como <b>True</b> ou <b>False</b> .
<i>instruçõeselseif</i>	Opcional. Uma ou mais linhas de programa ou diretivas de compilador que serão avaliadas se <i>expressão-n</i> for <b>True</b> .
<i>instruçõeselse</i>	Opcional. Uma ou mais linhas de programa ou diretivas de compilador que serão avaliadas se nenhuma <i>expressão</i> ou <i>expressão-n</i> anterior for <b>True</b> .

### Comentários

O comportamento da diretiva **#If...Then...#Else** é igual ao da instrução **If...Then...Else**, exceto que não há um formulário de linha única para as diretivas **#If**, **#Else**, **#Elseif** e **#End If**; isto é, nenhum outro código pode aparecer na mesma linha que qualquer das diretivas. A compilação condicional normalmente é utilizada para compilar o mesmo programa para plataformas diferentes. Também é utilizada para evitar que o código de depuração apareça em um arquivo executável. O código excluído durante a compilação condicional é omitido completamente do arquivo executável final e, desta forma, ele não afeta seu tamanho ou desempenho.

Independentemente do resultado de qualquer avaliação, todas as expressões são avaliadas. Portanto, todas as constantes utilizadas em expressões devem ser definidas; qualquer constante não definida é avaliada como **Empty**.

**Observação** A instrução **Option Compare** não afeta expressões em *instruções #If* e *#Elseif*. Expressões em uma diretiva de compilador condicional são sempre avaliadas com **Option Compare Text**.

## Constantes do Visual Basic

O Visual Basic para Aplicativos define as constantes para simplificar sua programação. As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

## Constantes do calendário

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

Constante	Valor	Descrição
<b>vbCalGreg</b>	<b>0</b>	Indica que o calendário gregoriano é utilizado
<b>vbCalHijri</b>	<b>1</b>	Indica que o calendário Hijri é utilizado

## Constantes do compilador

O Visual Basic para Aplicativos define as constantes para utilização exclusiva com a diretiva **#If...Then...#Else**. Essas constantes são funcionalmente equivalentes às constantes definidas com a diretiva **#If...Then...#Else**, exceto que elas são globais no que se refere ao escopo; isto é, elas se aplicam a todo o projeto.

Em plataformas de desenvolvimento de 16 bits, as constantes do compilador são definidas da seguinte maneira:

Constante	Valor	Descrição
<b>Win16</b>	<b>True</b>	Indica que o ambiente de desenvolvimento é de 16 bits.
<b>Win32</b>	<b>False</b>	Indica que o ambiente de desenvolvimento não é de 32 bits.

Em plataformas de desenvolvimento de 32 bits, as constantes do compilador são definidas da seguinte maneira:

Constante	Valor	Descrição
<b>Win16</b>	<b>False</b>	Indica que o ambiente de desenvolvimento não é de 16 bits.
<b>Win32</b>	<b>True</b>	Indica que o ambiente de desenvolvimento é de 32 bits.

**Observação** Essas constantes são fornecidas pelo Visual Basic. Sendo assim você não pode definir suas próprias constantes com esses mesmos nomes em qualquer nível.

## Constantes Date

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

### Valores do argumento

O argumento *primeirodiadasemana* possui os valores a seguir:

Constante	Valor	Descrição
<b>vbUseSystem</b>	0	Utiliza a definição NLS API.
<b>VbSunday</b>	1	Domingo (padrão)
<b>vbMonday</b>	2	Segunda
<b>vbTuesday</b>	3	Terça
<b>vbWednesday</b>	4	Quarta
<b>vbThursday</b>	5	Quinta
<b>vbFriday</b>	6	Sexta
<b>vbSaturday</b>	7	Sábado

O argumento *primeirodiadoano* possui os valores a seguir:

Constante	Valor	Descrição
<b>vbUseSystem</b>	0	Utiliza a definição NLS API.
<b>VbFirstJan1</b>	1	Inicia com a semana na qual ocorre o dia 1 de janeiro (padrão).
<b>VbFirstFourDays</b>	2	Inicia com a primeira semana que possui pelo menos quatro dias do ano novo.
<b>VbFirstFullWeek</b>	3	Inicia com a primeira semana completa do ano.

### Valores de retorno

Constante	Valor	Descrição
<b>vbSunday</b>	1	Domingo
<b>vbMonday</b>	2	Segunda
<b>vbTuesday</b>	3	Terça
<b>vbWednesday</b>	4	Quarta
<b>vbThursday</b>	5	Quinta
<b>vbFriday</b>	6	Sexta
<b>vbSaturday</b>	7	Sábado

## Constantes Dir, GetAttr e SetAttr

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

Constante	Valor	Descrição
<b>vbNormal</b>	0	Normal (padrão de <b>Dir</b> e <b>SetAttr</b> )
<b>vbReadOnly</b>	1	Somente leitura
<b>vbHidden</b>	2	Oculto
<b>vbSystem</b>	4	Arquivo do sistema
<b>vbVolume</b>	8	Rótulo do volume
<b>vbDirectory</b>	16	Diretório ou pasta
<b>vbArchive</b>	32	O arquivo foi alterado desde o último backup

## Constantes IMEStatus

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

As constantes referentes à localidade japonesa são as seguintes:

Constante	Valor	Descrição
<b>vbIMENoOp</b>	0	Nenhum IME instalado
<b>vbIMEOn</b>	1	IME ligado
<b>vbIMEOff</b>	2	IME desligado
<b>vbIMEDisable</b>	3	IME desativado
<b>vbIMEHiragana</b>	4	Caracteres de duplo byte Hiragana (DBC)
<b>vbIMEKatakanaDbi</b>	5	Katakana DBC
<b>vbIMEKatakanaSng</b>	6	Caracteres de byte simples Katakana (SBC)
<b>vbIMEAlphaDbi</b>	7	DBC alfanumérico
<b>vbIMEAlphaSng</b>	8	SBC alfanumérico

As constantes para a localidade chinesa (em chinês tradicional e simplificado) são as seguintes:

Constante	Valor	Descrição
<b>vbIMENoOp</b>	0	Nenhum IME instalado
<b>vbIMEOn</b>	1	IME ligado
<b>vbIMEOff</b>	2	IME desligado

Para a localidade coreana, os cinco primeiros bits do valor de retorno são os seguintes:

Bit	Valor	Descrição	Valor	Descrição
0	0	Nenhum IME instalado	1	IME instalado
1	0	IME desativado	1	IME ativado
2	0	Modo inglês IME	1	Modo Hangeul
3	0	Modo Banja (SBC)	1	Modo Junja (DBC)
4	0	Modo normal	1	Modo de conversão Hanja

## Constantes Instr e StrComp

As constantes a seguir são definidas na biblioteca de tipos do Visual Basic para Aplicativos e podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

Constante	Valor	Descrição
<b>vbBinaryCompare</b>	0	Realiza a comparação binária
<b>vbTextCompare</b>	1	Realiza a comparação textual
<b>vbDatabaseCompare</b>	2	Para o Microsoft Access, realiza a comparação baseada nas informações contidas em seu banco de dados.

## Constantes diversas

As constantes a seguir são definidas na biblioteca de tipos do Visual Basic para Aplicativos e podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

Constante	Equivalente	Descrição
<b>vbCrLf</b>	<b>Chr(13) + Chr(10)</b>	Combinação de retorno de carro e

		alimentação de linha
<b>vbCr</b>	<b>Chr(13)</b>	Caractere de retorno de carro
<b>vbLf</b>	<b>Chr(10)</b>	Caractere de alimentação de linha
<b>vbNewLine</b>	<b>Chr(13) + Chr(10)</b> ou <b>Chr(13)</b>	Caractere de linha nova específico de uma plataforma; aquele que seja apropriado à plataforma atual
<b>vbNullChar</b>	<b>Chr(0)</b>	Caractere que possui o valor 0
<b>vbNullString</b>	Seqüência contendo o valor 0	Não é o mesmo que a seqüência de comprimento zero (""); utilizada para chamar procedimentos externos
<b>vbTab</b>	<b>Chr(9)</b>	Caractere de tabulação
<b>vbBack</b>	<b>Chr(8)</b>	Caractere de backspace
<b>vbFormFeed</b>	<b>Chr(12)</b>	Não é útil no Microsoft Windows
<b>vbVerticalTab</b>	<b>Chr(11)</b>	Não é útil no Microsoft Windows

## Constantes MsgBox

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

### Argumentos MsgBox

<u>Constante</u>	<u>Valor</u>	<u>Descrição</u>
<b>vbOKOnly</b>	0	Apenas o botão <b>OK</b> (padrão)
<b>vbOKCancel</b>	1	Botões <b>OK</b> e <b>Cancelar</b>
<b>vbAbortRetryIgnore</b>	2	Botões <b>Anular</b> , <b>Repetir</b> e <b>Ignorar</b>
<b>vbYesNoCancel</b>	3	Botões <b>Sim</b> , <b>Não</b> e <b>Cancelar</b>
<b>vbYesNo</b>	4	Botões <b>Sim</b> e <b>Não</b>
<b>vbRetryCancel</b>	5	Botões <b>Repetir</b> e <b>Cancelar</b>
<b>vbCritical</b>	16	Mensagem crítica
<b>vbQuestion</b>	32	Consulta de aviso
<b>vbExclamation</b>	48	Mensagem de aviso
<b>vbInformation</b>	64	Mensagem de informação
<b>vbDefaultButton1</b>	0	Primeiro botão é o padrão (padrão)
<b>vbDefaultButton2</b>	256	Segundo botão é o padrão
<b>vbDefaultButton3</b>	512	Terceiro botão é o padrão
<b>vbDefaultButton4</b>	768	Quarto botão é o padrão
<b>vbApplicationModal</b>	0	Caixa de mensagem relativa ao modo do aplicativo (padrão)
<b>vbSystemModal</b>	4096	Caixa de mensagem relativa ao modo do sistema

### Valores de retorno de MsgBox

<u>Constante</u>	<u>Valor</u>	<u>Descrição</u>
<b>vbOK</b>	1	Botão <b>OK</b> pressionado
<b>vbCancel</b>	2	Botão <b>Cancelar</b> pressionado
<b>vbAbort</b>	3	Botão <b>Anular</b> pressionado
<b>vbRetry</b>	4	Botão <b>Repetir</b> pressionado
<b>vbIgnore</b>	5	Botão <b>Ignorar</b> pressionado
<b>vbYes</b>	6	Botão <b>Sim</b> pressionado
<b>vbNo</b>	7	Botão <b>Não</b> pressionado

## Constantes Shell

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbHide</b>	0	A janela está oculta e o foco é passado para a janela oculta.
<b>vbNormalFocus</b>	1	A janela recebe o foco e é restaurada para seu tamanho e posição originais.
<b>vbMinimizedFocus</b>	2	A janela é exibida como um ícone com o foco sobre ela.
<b>vbMaximizedFocus</b>	3	A janela é maximizada com o foco.
<b>vbNormalNoFocus</b>	4	A janela é restaurada para o tamanho e posição mais recentes. A janela atualmente ativa permanece ativa.
<b>vbMinimizeNoFocus</b>	6	A janela é exibida como um ícone. A janela atualmente ativa permanece ativa.

## Constantes StrConv

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbUpperCase</b>	1	Converte a seqüência em caracteres maiúsculos.
<b>VbLowerCase</b>	2	Converte a seqüência em caracteres minúsculos.
<b>VbProperCase</b>	3	Converte a primeira letra de cada palavra da seqüência em maiúsculas.
<b>VbWide</b>	4	Converte os caracteres estreitos (de byte simples) da seqüência em caracteres largos (de byte duplo). Aplicável às <u>localidades</u> do oriente.
<b>VbNarrow</b>	8	Converte os caracteres largos (de byte duplo) da seqüência em caracteres estreitos (de byte simples). Aplicável às localidades do oriente.
<b>VbKatakana</b>	16	Converte caracteres Hiragana da seqüência em caracteres Katakana. Aplicável apenas ao Japão.
<b>vbHiragana</b>	32	Converte os caracteres Katakana da seqüência em caracteres Hiragana. Aplicável apenas ao Japão.
<b>vbUnicode</b>	64	Converte a seqüência para <u>Unicode</u> utilizando a página de código padrão do sistema.
<b>vbFromUnicode</b>	128	Converte a seqüência de Unicode para a página de códigos padrão do sistema.

## Constantes VarType

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores

reais:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbEmpty</b>	0	Não-inicializada (padrão)
<b>vbNull</b>	1	Não contém dados válidos
<b>vbInteger</b>	2	<b><u>Integer</u></b>
<b>vbLong</b>	3	Inteiro longo
<b>vbSingle</b>	4	Número de vírgula flutuante de precisão única
<b>vbDouble</b>	5	Número de vírgula flutuante de precisão dupla
<b>vbCurrency</b>	6	<b><u>Currency</u></b>
<b>vbDate</b>	7	<b><u>Date</u></b>
<b>vbString</b>	8	<b><u>String</u></b>
<b>vbObject</b>	9	Objeto
<b>vbError</b>	10	Erro
<b>vbBoolean</b>	11	<b><u>Boolean</u></b>
<b>vbVariant</b>	12	<b><u>Variant</u></b> (utilizada apenas para as <u>matrizes</u> de variantes)
<b>vbDataObject</b>	13	Objeto de acesso a dados
<b>vbDecimal</b>	14	<b><u>Decimal</u></b>
<b>vbByte</b>	17	<b><u>Byte</u></b>
<b>vbArray</b>	8192	Matriz

## Constantes de cor

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbBlack</b>	0x0	Preto
<b>vbRed</b>	0xFF	Vermelho
<b>vbGreen</b>	0xFF00	Verde
<b>vbYellow</b>	0xFFFF	Amarelo
<b>vbBlue</b>	0xFF0000	Azul
<b>vbMagenta</b>	0xFF00FF	Magenta
<b>vbCyan</b>	0xFFFF00	Ciano
<b>vbWhite</b>	0xFFFFFFFF	Branco

## Constantes de código de tecla

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbKeyLButton</b>	0x1	Botão esquerdo do mouse
<b>vbKeyRButton</b>	0x2	Botão direito do mouse
<b>vbKeyCancel</b>	0x3	Tecla CANCEL
<b>vbKeyMButton</b>	0x4	Botão do meio do mouse
<b>vbKeyBack</b>	0x8	Tecla BACKSPACE
<b>vbKeyTab</b>	0x9	Tecla TAB
<b>vbKeyClear</b>	0xC	Tecla CLEAR

<b>vbKeyReturn</b>	0xD	Tecla ENTER
<b>vbKeyShift</b>	0x10	Tecla SHIFT
<b>vbKeyControl</b>	0x11	Tecla CTRL
<b>vbKeyMenu</b>	0x12	Tecla MENU
<b>vbKeyPause</b>	0x13	Tecla PAUSE
<b>vbKeyCapital</b>	0x14	Tecla CAPS LOCK
<b>vbKeyEscape</b>	0x1B	Tecla ESC
<b>vbKeySpace</b>	0x20	Tecla SPACEBAR
<b>vbKeyPageUp</b>	0x21	Tecla PAGE UP
<b>vbKeyPageDown</b>	0x22	Tecla PAGE DOWN
<b>vbKeyEnd</b>	0x23	Tecla END
<b>vbKeyHome</b>	0x24	Tecla HOME
<b>vbKeyLeft</b>	0x25	Tecla SETA À ESQUERDA
<b>vbKeyUp</b>	0x26	Tecla SETA ACIMA
<b>vbKeyRight</b>	0x27	Tecla SETA À DIREITA
<b>vbKeyDown</b>	0x28	Tecla SETA ABAIXO
<b>vbKeySelect</b>	0x29	Tecla SELECT
<b>vbKeyPrint</b>	0x2A	Tecla PRINT SCREEN
<b>vbKeyExecute</b>	0x2B	Tecla EXECUTE
<b>vbKeySnapshot</b>	0x2C	Tecla SNAPSHOT
<b>vbKeyInsert</b>	0x2D	Tecla INSERT
<b>vbKeyDelete</b>	0x2E	Tecla DELETE
<b>vbKeyHelp</b>	0x2F	Tecla HELP
<b>vbKeyNumlock</b>	0x90	Tecla NUM LOCK

As teclas de A a Z são as mesmas teclas ASCII de A a Z equivalentes:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbKeyA</b>	65	Tecla A
<b>vbKeyB</b>	66	Tecla B
<b>vbKeyC</b>	67	Tecla C
<b>vbKeyD</b>	68	Tecla D
<b>vbKeyE</b>	69	Tecla E
<b>vbKeyF</b>	70	Tecla F
<b>vbKeyG</b>	71	Tecla G
<b>vbKeyH</b>	72	Tecla H
<b>vbKeyI</b>	73	Tecla I
<b>vbKeyJ</b>	74	Tecla J
<b>vbKeyK</b>	75	Tecla K
<b>vbKeyL</b>	76	Tecla L
<b>vbKeyM</b>	77	Tecla M
<b>vbKeyN</b>	78	Tecla N
<b>vbKeyO</b>	79	Tecla O
<b>vbKeyP</b>	80	Tecla P
<b>vbKeyQ</b>	81	Tecla Q
<b>vbKeyR</b>	82	Tecla R
<b>vbKeyS</b>	83	Tecla S
<b>vbKeyT</b>	84	Tecla T
<b>vbKeyU</b>	85	Tecla U
<b>vbKeyV</b>	86	Tecla V
<b>vbKeyW</b>	87	Tecla W

---

<b>vbKeyX</b>	88	Tecla x
<b>vbKeyY</b>	89	Tecla y
<b>vbKeyZ</b>	90	Tecla z

As teclas de 0 a 9 são as mesmas teclas ASCII de 0 a 9 equivalentes:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbKey0</b>	48	Tecla 0
<b>vbKey1</b>	49	Tecla 1
<b>vbKey2</b>	50	Tecla 2
<b>vbKey3</b>	51	Tecla 3
<b>vbKey4</b>	52	Tecla 4
<b>vbKey5</b>	53	Tecla 5
<b>vbKey6</b>	54	Tecla 6
<b>vbKey7</b>	55	Tecla 7
<b>vbKey8</b>	56	Tecla 8
<b>vbKey9</b>	57	Tecla 9

As constantes a seguir representam teclas no teclado numérico:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbKeyNumpad0</b>	0x60	Tecla 0
<b>vbKeyNumpad1</b>	0x61	Tecla 1
<b>vbKeyNumpad2</b>	0x62	Tecla 2
<b>vbKeyNumpad3</b>	0x63	Tecla 3
<b>vbKeyNumpad4</b>	0x64	Tecla 4
<b>vbKeyNumpad5</b>	0x65	Tecla 5
<b>vbKeyNumpad6</b>	0x66	Tecla 6
<b>vbKeyNumpad7</b>	0x67	Tecla 7
<b>vbKeyNumpad8</b>	0x68	Tecla 8
<b>vbKeyNumpad9</b>	0x69	Tecla 9
<b>vbKeyMultiply</b>	0x6A	Tecla SINAL DE MULTIPLICAÇÃO (*)
<b>vbKeyAdd</b>	0x6B	Tecla SINAL DE ADIÇÃO (+)
<b>vbKeySeparator</b>	0x6C	Tecla ENTER
<b>vbKeySubtract</b>	0x6D	Tecla SINAL DE SUBTRAÇÃO (-)
<b>vbKeyDecimal</b>	0x6E	Tecla PONTO DECIMAL (.)
<b>vbKeyDivide</b>	0x6F	Tecla SINAL DE DIVISÃO (/)

As constantes a seguir representam teclas de função:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbKeyF1</b>	0x70	Tecla F1
<b>vbKeyF2</b>	0x71	Tecla F2
<b>vbKeyF3</b>	0x72	Tecla F3
<b>vbKeyF4</b>	0x73	Tecla F4
<b>vbKeyF5</b>	0x74	Tecla F5
<b>vbKeyF6</b>	0x75	Tecla F6
<b>vbKeyF7</b>	0x76	Tecla F7
<b>vbKeyF8</b>	0x77	Tecla F8
<b>vbKeyF9</b>	0x78	Tecla F9
<b>vbKeyF10</b>	0x79	Tecla F10
<b>vbKeyF11</b>	0x7A	Tecla F11
<b>vbKeyF12</b>	0x7B	Tecla F12
<b>vbKeyF13</b>	0x7C	Tecla F13

<b>vbKeyF14</b>	0x7D	Tecla F14
<b>vbKeyF15</b>	0x7E	Tecla F15
<b>vbKeyF16</b>	0x7F	Tecla F16

## Constantes da cor do sistema

As constantes a seguir podem ser utilizadas em qualquer parte do seu código no lugar dos valores reais:

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbScrollBars</b>	0x80000000	Cor da barra de rolagem
<b>vbDesktop</b>	0x80000001	Cor da Área de trabalho
<b>vbActiveTitleBar</b>	0x80000002	Cor da barra de título da janela ativa
<b>vbInactiveTitleBar</b>	0x80000003	Cor da barra de título da janela inativa
<b>vbMenuBar</b>	0x80000004	Cor de segundo plano do menu
<b>vbWindowBackground</b>	0x80000005	Cor de segundo plano da janela
<b>vbWindowFrame</b>	0x80000006	Cor da moldura da janela
<b>vbMenuText</b>	0x80000007	Cor do texto nos menus
<b>vbWindowText</b>	0x80000008	Cor do texto nas janelas
<b>vbTitleBarText</b>	0x80000009	Cor do texto na legenda, caixa de dimensionamento e seta de rolagem
<b>vbActiveBorder</b>	0x8000000A	Cor da borda da janela ativa
<b>vbInactiveBorder</b>	0x8000000B	Cor da borda da janela inativa
<b>vbApplicationWorkspace</b>	0x8000000C	Cor de segundo plano dos aplicativos de interface de múltiplos documentos (MDI, <i>Multiple-Document Interface</i> )
<b>vbHighlight</b>	0x8000000D	Cor de segundo plano dos itens selecionados em um controle
<b>vbHighlightText</b>	0x8000000E	Cor do texto dos itens selecionados em um controle
<b>vbButtonFace</b>	0x8000000F	Cor de sombreamento da superfície dos botões de comando
<b>vbButtonShadow</b>	0x80000010	Cor de sombreamento da extremidade dos botões de comando
<b>vbGrayText</b>	0x80000011	Texto acinzentado (desativado)
<b>vbButtonText</b>	0x80000012	Cor do texto dos botões de pressionamento
<b>vbInactiveCaptionText</b>	0x80000013	Cor do texto de uma legenda ativa
<b>vb3DHighlight</b>	0x80000014	Cor de realce de elementos de exibição em 3-D
<b>vb3DDKShadow</b>	0x80000015	Cor de sombreamento mais escura dos elementos de

		exibição em 3-D
<b>vb3DLight</b>	0x80000016	Segunda cor 3-D mais clara depois de <b>vb3DHighlight</b>
<b>vbInfoText</b>	0x80000017	Cor do texto das Dicas de ferramentas
<b>vbInfoBackground</b>	0x80000018	Cor de segundo plano das Dicas de ferramentas

### Exemplo da instrução Call

Este exemplo ilustra como a instrução **Call** é utilizada para transferir o controle para um procedimento **Sub**, uma função intrínseca, um procedimento da biblioteca de vinculação dinâmica (DLL) e um procedimento.

```
' Chama um procedimento Sub.
Call PrintToDebugWindow("Alô Mundo")
' A instrução anterior faz com que o controle seja passado ao procedimento
' Sub seguinte.
Sub PrintToDebugWindow(AnyString)
    Debug.Print AnyString ' Imprime na janela Depurar.
End Sub

' Chama uma função intrínseca. O valor de retorno da função é descartado.
Call Shell(AppName, 1) ' AppName contém o caminho do
    arquivo executável.

' Chama um procedimento DLL do Microsoft Windows. A
' instrução Declare deve ser Private em um Módulo de
' classe, mas não em um Módulo padrão.
Private Declare Sub MessageBeep Lib "Usuário" (ByVal N As Integer)
Sub CallMyDll()
    Call MessageBeep(0) ' Chama o procedimento DLL do Windows.
    MessageBeep 0 ' Chama novamente sem a palavra-chave Call.
End Sub
```

### Exemplo da função Choose

Este exemplo utiliza a função **Choose** para exibir um nome em resposta a um índice passado ao procedimento no parâmetro `Ind`.

```
Function GetChoice(Ind As Integer)
    GetChoice = Choose(Ind, "Expresso", "United", "Federal")
End Function
```

### Exemplo da função DoEvents

Este exemplo utiliza a função **DoEvents** para fazer com que se saia para o sistema operacional uma vez a cada 1000 iterações do loop. **DoEvents** retorna o número de formulários abertos do Visual Basic, mas somente quando o aplicativo host for o Visual Basic.

```
' Cria uma variável para conter o número de formulários ' do Visual Basic
carregados e visíveis.
Dim I, AbreFormulários
For I = 1 To 150000 ' Inicia o loop.
    If I Mod 1000 = 0 Then ' Se o loop tiver se repetido 1000 vezes.
        AbreFormulários = DoEvents ' Submete-se ao sistema operacional.
    End If
Next I ' Incrementa o contador de loops.
```

### Exemplo da instrução Do...Loop

Este exemplo mostra como as instruções **Do...Loop** podem ser utilizadas. A instrução **Do...Loop** interna faz o loop 10 vezes, define o valor do sinalizador como **False** e sai prematuramente utilizando a instrução **Exit Do**. O loop externo sai imediatamente após a verificação do valor do sinalizador.

```
Dim Controle, Contador
Controle = True: Contador = 0 ' Inicializa as variáveis.
Do ' Loop externo.
    Do While Contador < 20 ' Loop interno.
        Contador = Contador + 1 ' Incrementa o Contador.
        If Contador = 10 Then ' Se a condição for True.
            Controle = False ' Define o valor do sinalizador como False.
            Exit Do ' Sai do loop interno.
        End If
    Loop
Loop Until Controle = False ' Sai do loop externo imediatamente.
```

### Exemplo da instrução End

Este exemplo utiliza a instrução **End** para encerrar a execução do código se o usuário inserir uma senha inválida.

```
Sub Form_Load
    Dim Senha, ASenha
    Senha = "Swordfish"
    ASenha = InputBox("Digite a sua senha")
    If ASenha <> Senha Then
        MsgBox "Sinto muito, senha incorreta"
    End
End If
End Sub
```

### Exemplo da instrução Exit

Este exemplo utiliza a instrução **Exit** para sair de um loop **For...Next**, **Do...Loop** e de um procedimento **Sub**.

```
Sub ExitStatementDemo()
Dim I, MeuNum
  Do          ' Configura o loop infinito.
    For I = 1 To 1000 ' Faz o loop 1000 vezes.
      MeuNum = Int(Rnd * 1000) ' Gera números aleatórios.
      Select Case MeuNum ' Avalia o número aleatório.
        Case 7: Exit For ' Se 7, sai de For...Next.
        Case 29: Exit Do ' Se 29, sai de Do...Loop.
        Case 54: Exit Sub ' Se 54, sai do procedimento Sub.
      End Select
    Next I
  Loop
End Sub
```

### Exemplo da instrução For Each...Next

Este exemplo utiliza a instrução **For Each...Next** para pesquisar a propriedade **Text** de todos os elementos em uma coleção à procura da seqüência de caracteres "Alô". No exemplo, **MeuObjeto** é um objeto relacionado ao texto e é um elemento da coleção **MinhaColeção**. Ambos são nomes genéricos utilizados somente com finalidades ilustrativas.

```
Dim Encontrado, MeuObjeto, MinhaColeção
Encontrado = False ' Inicializa a variável.
For Each MeuObjeto In MinhaColeção ' Itera a cada elemento.
  If MeuObjeto.Text = "Alô" Then ' Se Text for igual a "Alô".
    Encontrado = True ' Define Encontrado como True.
    Exit For ' Sai do loop.
  End If
Next
```

### Exemplo da instrução For...Next

Este exemplo utiliza a instrução **For...Next** para criar uma seqüência de caracteres que contém 10 instâncias dos números 0 a 9, cada seqüência de caracteres é separada da outra por um único espaço. O loop externo utiliza uma variável de contador de loops que é decrescida toda vez através do loop.

```
Dim Palavras, Caracteres, MinhaSeqüência
For Palavras = 10 To 1 Step -1 ' Configura 10 repetições.
  For Caracteres = 0 To 9 ' Configura 10 repetições.
    MinhaSeqüência = MinhaSeqüência & Caracteres ' Acrescenta o número
    à seqüência de caracteres.
  Next Caracteres ' Incrementa o contador
  MinhaSeqüência = MinhaSeqüência & " " ' Acrescenta um espaço.
Next Palavras
```

### Exemplo da instrução GoSub...Return

Este exemplo utiliza **GoSub** para chamar uma sub-rotina dentro de um procedimento **Sub**. A instrução **Return** faz com que a execução continue imediatamente após a instrução **GoSub**. A instrução **Exit Sub** é utilizada para evitar que o controle flua acidentalmente para a sub-rotina.

```
Sub GosubDemo()
Dim Num
' Solicita um número ao usuário.
  Num = InputBox("Insira um número positivo para ser dividido por 2.")
' Utiliza a rotina somente se o usuário inserir um número positivo.
  If Num > 0 Then GoSub MinhaRotina
  Debug.Print Num
  Exit Sub ' Utiliza Exit para evitar um erro.
MinhaRotina:
  Num = Num/2 ' Executa a divisão.
  Return ' Retorna o controle à instrução
End Sub ' que se segue à instrução GoSub.
```

### Exemplo da instrução GoTo

Este exemplo utiliza a instrução **GoTo** para desviar para rótulos de linha de um procedimento.

```
Sub GotoStatementDemo()
Dim Número, MinhaSeqüência
  Número = 1 ' Inicializa a variável.
  ' Avalia Número e desvia para o rótulo apropriado.
  If Número = 1 Then GoTo Line1 Else GoTo Line2

Line1:
  MinhaSeqüência = "Número igual a 1"
  GoTo LastLine ' Vai para LastLine.
Line2:
  ' A instrução a seguir nunca é executada.
  MinhaSeqüência = "Número é igual a 2"
LastLine:
  Debug.Print MinhaSeqüência ' Imprime "Número é igual a 1" na
  ' janela Depurar.
End Sub
```

### Exemplo da instrução If...Then...Else

Este exemplo mostra as formas de bloco e de linha única da instrução **If...Then...Else**. Ele também ilustra o uso de **If TypeOf...Then...Else**.

```
Dim Número, Dígitos, MinhaSeqüência
Número = 53 ' Inicializa a variável.
If Número < 10 Then
  Dígitos = 1
ElseIf Número < 100 Then
' A condição avalia como True, então a próxima instrução é executada.
  Dígitos = 2
Else
  Dígitos = 3
End If

' Atribui um valor utilizando a forma de sintaxe de linha única.
If Dígitos = 1 Then MinhaSeqüência = "Uma" Else MinhaSeqüência = "Mais de
uma"
```

Utilize a construção **If TypeOf** para determinar se o Controle passado para um procedimento é uma

caixa de texto.

```
Sub ControlProcessor(MeuControle As Control)
    If TypeOf MeuControle Is CommandButton Then
        Debug.Print "Você passou um " & TypeName(MeuControle)
    ElseIf TypeOf MeuControle Is CheckBox Then
        Debug.Print "Você passou um " & TypeName(MeuControle)
    ElseIf TypeOf MeuControle Is TextBox Then
        Debug.Print "Você passou um " & TypeName(MeuControle)
    End If
End Sub
```

### Exemplo da função IIf

Este exemplo utiliza a função **IIf** para avaliar o parâmetro `TestMe` do procedimento `CheckIt` e retorna a palavra "Grande" se a quantidade for maior que 1000; do contrário, retorna a palavra "Pequena".

```
Function CheckIt (TestMe As Integer)
    CheckIt = IIf(TestMe > 1000, "Grande", "Pequena")
End Function
```

### Exemplo das instruções On...GoSub e On...GoTo

Este exemplo utiliza as instruções **On...GoSub** e **On...GoTo** para desviar para sub-rotinas e rótulos de linha, respectivamente.

```
Sub OnGosubGotoDemo()
    Dim Número, MinhaSeqüência
    Número = 2 ' Inicializa a variável.
    ' Desvia para Sub2.
    On Número GoSub Sub1, Sub2 ' A execução continua aqui depois de
    ' On...GoSub.
    On Número GoTo Linha1, Linha2 ' Desvia para Linha2.
    ' A execução não continua aqui depois de On...GoTo.
    Exit Sub
Sub1:
    MinhaSeqüência = "Em Sub1" : Return
Sub2:
    MinhaSeqüência = "Em Sub2" : Return
Linha1:
    MinhaSeqüência = "Em Linha1"
Linha2:
    MinhaSeqüência = "Em Linha2"
End Sub
```

### Exemplo da função Partition

Este exemplo pressupõe que você tem uma tabela **Pedidos** que contém um campo Frete. Ela cria um procedimento de seleção que conta o número de pedidos para os quais o custo se enquadra em cada um dos diversos intervalos. A função **Partition** é utilizada primeiro para estabelecer esses intervalos e, em seguida, a função SQL Count conta o número de pedidos em cada faixa. Neste exemplo, os argumentos da função **Partition** são *início* = 0, *fim* = 500, *intervalo* = 50. O primeiro intervalo, portanto, seria 0:49, e assim por diante até 500.

```
SELECT DISTINCTROW Partition([frete],0, 500, 50) AS Range,
Count(Pedidos.Frete) AS Count
FROM Pedidos
GROUP BY Partition([frete],0,500,50);
```

### Exemplo da instrução Select Case

Este exemplo utiliza a instrução **Select Case** para avaliar o valor de uma variável. A segunda cláusula **Case** contém o valor da variável que está sendo avaliada e, portanto, somente a instrução associada a ela é executada.

```
Dim Número
Número = 8 ' Inicializa a variável.
Select Case Número ' Avalia Número.
Case 1 To 5 ' Número entre 1 e 5.
    Debug.Print "Entre 1 e 5"
' A seguinte é a única cláusula Case que avalia como True.
Case 6, 7, 8 ' Número entre 6 e 8.
    Debug.Print "Entre 6 e 8"
Case Is > 8 And Número < 11 ' Número é 9 ou 10.
    Debug.Print "Maior que 8"
Case Else ' Outros valores.
    Debug.Print "Não entre 1 e 10"
End Select
```

### Exemplo da função Shell

Este exemplo utiliza a função **Shell** para executar um aplicativo especificado pelo usuário.

```
' Especificar 1 como o segundo argumento abre o aplicativo no tamanho
normal e lhe dá o foco.
Dim ValRet
ValRet = Shell("C:\WINDOWS\CALC.EXE", 1) ' Executa Calculator.
```

### Exemplo da instrução Stop

Este exemplo utiliza a instrução **Stop** para suspender a execução de cada iteração através do loop **For...Next**.

```
Dim I
For I = 1 To 10 ' Inicia o loop For...Next.
    Debug.Print I ' Imprime I na janela Depurar.
    Stop ' Pára durante cada iteração.
Next I
```

### Exemplo da função Switch

Este exemplo utiliza a função **Switch** para retornar o nome de um idioma que corresponde ao nome de uma cidade.

```
Function MatchUp (NomeDeCidade As String)
    Matchup = Switch(NomeDeCidade = "Londres", "Inglês", NomeDeCidade = _
        = "Roma", "Italiano", NomeDeCidade = "Paris", "Francês")
End Function
```

### Exemplo da instrução While...Wend

Este exemplo utiliza a instrução **While...Wend** para incrementar uma variável de contador. As instruções no loop são executadas desde que a condição avalie **True**.

```
Dim Contador
Contador = 0 ' Inicializa a variável.
While Contador < 20 ' Testa valor de Contador.
    Contador = Contador + 1 ' Incrementa Contador.
Wend ' Encerra o loop While quando Contador > 19.
Debug.Print Contador ' Imprime 20 na janela Depurar.
```

### Exemplo da instrução With

Este exemplo utiliza a instrução **With** para executar uma série de instruções em um único objeto. O objeto `MeuObjeto` e as suas propriedades são nomes genéricos utilizados somente com finalidades ilustrativas.

```
With MeuObjeto
    .Height = 100 ' Igual a MeuObjeto.Height = 100.
    .Caption = "Alô Mundo" ' Igual a MeuObjeto.Caption = "Alô Mundo".
    With .Font
        .Color = Red ' Igual a MeuObjeto.Font.Color = Red.
        .Bold = True ' Igual a MeuObjeto.Font.Bold = True.
    End With
End With
```

## Instrução Call

Transfere o controle para um procedimento **Sub**, um procedimento **Function** ou procedimento de biblioteca de vínculos dinâmicos (DLL).

### Sintaxe

[**Call**] *nome* [*listadeargumentos*]

A sintaxe da instrução **Call** possui as partes a seguir:

Parte	Descrição
<b>Call</b>	Opcional; <u>palavra-chave</u> . Se for especificada, você deverá colocar <i>listadeargumentos</i> entre parênteses. Por exemplo: <code>Call MyProc(0)</code>
<i>nome</i>	Obrigatória. Nome do procedimento a chamar.
<i>listadeargumentos</i>	Opcional. Lista, delimitada por vírgulas, de <u>variáveis</u> , <u>matrizes</u> ou <u>expressões</u> a serem passadas para o procedimento. Os componentes da <i>listadeargumentos</i> podem incluir as palavras-chave <b>ByVal</b> ou <b>ByRef</b> para descrever como os <u>argumentos</u> são tratados pelo procedimento chamado. Entretanto, <b>ByVal</b> e <b>ByRef</b> podem ser utilizadas com <b>Call</b> somente quando for chamado um procedimento DLL.

### Comentários

Não é necessário que você utilize a palavra-chave **Call** quando chamar um procedimento, entretanto, se você usá-la para chamar um procedimento que requer argumentos, a *listadeargumentos* deve ficar entre parênteses. Se você omitir a palavra-chave **Call**, também deverá omitir os parênteses de *listadeargumentos*. Se utilizar qualquer sintaxe de **Call** para chamar alguma função intrínseca ou definida pelo usuário, o valor de retorno da função será descartado.

Para passar uma matriz inteira para um procedimento, utilize o nome da matriz seguido de parênteses vazios.

## Função Choose

Seleciona e retorna um valor de uma lista de argumentos.

### Sintaxe

**Choose**(*índice*, *escolha-1* [, *escolha-2*, ... [, *escolha-n*]])

A sintaxe da função **Choose** possui as partes a seguir:

Parte	Descrição
<i>índice</i>	Obrigatória. <u>Expressão numérica</u> ou campo que resulta em um valor entre 1 e o número de escolhas disponíveis.
<i>escolha</i>	Obrigatória. <u>Expressão Variant</u> contendo uma das escolhas possíveis.

### Comentários

**Choose** retorna um valor da lista de escolhas com base no valor do *índice*. Se *índice* for 1, **Choose** retorna a primeira escolha da lista, se for 2, retorna a segunda escolha e assim por diante.

Você pode utilizar **Choose** para pesquisar um valor em uma lista de possibilidades. Por exemplo, se *índice* for avaliado como 3 e *escolha-1* = "um", *escolha-2* = "dois" e *escolha-3* = "três", **Choose** retorna "três". Esta capacidade é particularmente útil se *índice* representa o valor em um grupo de opções.

**Choose** avalia todas as escolhas da lista, mesmo que ela retorne somente uma. Em razão disto você

deve observar se existem efeitos colaterais indesejáveis. Por exemplo, se você utilizar a função **MsgBox** como parte de uma expressão em todas as escolhas, será exibida uma caixa de mensagem quando cada escolha for avaliada, mesmo que **Choose** retorne o valor de apenas uma delas.

A função **Choose** retorna um **Null** se *índice* for menor que 1 ou maior que o número de escolhas listadas.

Se *índice* não for um número inteiro, será arredondado para o número inteiro mais próximo antes de ser avaliado.

## Função DoEvents

Cede a execução para que o sistema operacional possa processar outros eventos.

### Sintaxe

**DoEvents( )**

### Comentários

A função **DoEvents** retorna um **Integer** que representa o número de formulários abertos em versões autônomas do Visual Basic, como o Visual Basic, Standard Edition. **DoEvents** retorna zero em todos os outros aplicativos.

**DoEvents** passa o controle para o sistema operacional. O controle é retornado depois de o sistema operacional terminar o processamento de eventos da sua fila e todas as chaves da fila **SendKeys** terem sido enviadas.

**DoEvents** tem uma maior utilidade com ações simples como permitir que um usuário cancele o processo após ele ter sido iniciado, por exemplo, uma procura por um arquivo. Para processos de longa execução, o controle do processador é conseguido com maior facilidade utilizando-se um cronômetro ou delegando-se a tarefa a um componente ActiveX EXE. Nesse último caso, a tarefa pode continuar de forma completamente independente de seu aplicativo e o sistema operacional leva em consideração a multitarefa e a divisão de tempo.

---

**Atenção** Sempre que você cede o controle ao processador dentro de um procedimento de evento, certifique-se de que o procedimento não seja executado novamente a partir de uma outra parte do seu código antes que a primeira chamada retorne, pois isto poderia produzir resultados imprevisíveis. Além disto, não utilize **DoEvents** se houver possibilidade de outros aplicativos interagirem com o seu procedimento de forma imprevisível durante o tempo em que você ceder o controle.

---

## Instrução Do...Loop

Repete um bloco de instruções enquanto uma condição é **True** ou até que ela se torne **True**.

### Sintaxe

```
Do                [{While                |                Until}                condição]
  [instruções]
  [Exit                Do]
  [instruções]
```

### Loop

Ou você pode utilizar esta sintaxe:

```
Do
  [instruções]
  [Exit                Do]
  [instruções]
```

**Loop** [{While | Until} *condição*]

A sintaxe da instrução **Do Loop** possui as partes a seguir:

Parte	Descrição
<i>condição</i>	Opcional. <u>Expressão numérica</u> ou <u>expressão de seqüência</u> que seja <b>True</b> ou <b>False</b> . Se <i>condição</i> for <b>Null</b> , <i>condição</i> é tratada como <b>False</b> .
<i>instruções</i>	Uma ou mais instruções que são repetidas enquanto, ou até que, <i>condição</i> seja <b>True</b> .

### Comentários

Qualquer número de instruções **Exit Do** pode ser colocado em qualquer lugar em **Do...Loop** como meio alternativo para sair de um **Do...Loop**. **Exit Do** é freqüentemente utilizada depois que alguma condição é avaliada, por exemplo, **If...Then**, caso em que a instrução **Exit Do** transfere o controle para a instrução imediatamente seguinte a **Loop**.

Quando utilizada com instruções **Do...Loop** embutidas, **Exit Do** transfere o controle para o loop que está embutido em um nível acima do loop em que ocorre **Exit Do**.

## Instrução End

Finaliza um procedimento ou bloco.

### Sintaxe

```
End
End Function
End If
End Property
End Select
End Sub
End Type
End With
```

A sintaxe da instrução **End** possui as formas a seguir:

Instrução	Descrição
<b>End</b>	Termina a execução – imediatamente. Nunca é obrigatória por si mesma, mas pode ser colocada em qualquer lugar dentro de um procedimento para finalizar a execução do código, fechar arquivos abertos com a instrução <b>Open</b> e limpar <u>variáveis</u> .
<b>End Function</b>	Obrigatória para finalizar uma instrução <b>Function</b> .

<b>End If</b>	Obrigatória para finalizar uma instrução de bloco <b>If...Then...Else</b> .
<b>End Property</b>	Obrigatória para finalizar um procedimento <b>Property Let</b> , <b>Property Get</b> ou <b>Property Set</b> .
<b>End Select</b>	Obrigatória para finalizar uma instrução <b>Select Case</b> .
<b>End Sub</b>	Obrigatória para finalizar uma instrução <b>Sub</b> .
<b>End Type</b>	Obrigatória finalizar uma definição de <u>tipo definido pelo usuário</u> (instrução <b>Type</b> ).
<b>End With</b>	Obrigatória para finalizar uma instrução <b>With</b> .

### Comentários

Quando é executada, a instrução **End** redefine todas as variáveis em nível de módulo e todas as variáveis locais estáticas em todos os módulos. Para preservar o valor dessas variáveis, utilize a instrução **Stop**. Assim, você poderá continuar a execução preservando o valor daquelas variáveis.

**Observação** A instrução **End** interrompe a execução do código abruptamente, sem chamar o evento Unload, QueryUnload ou Terminate ou qualquer outro código do Visual Basic. O código que você colocou nos eventos Unload, QueryUnload e Terminate de formulários e módulos de classe não é executado. Os objetos criados a partir de módulos de classe são destruídos, os arquivos abertos utilizando-se a instrução **Open** são fechados e a memória utilizada pelo seu programa é liberada. As referências de objeto pertencentes a outros programas são invalidadas.

A instrução **End** fornece uma maneira de fazer com que seu programa pare. Para a finalização normal de um programa do Visual Basic, você deve descarregar todos os formulários. O seu programa é finalizado logo que não haja outros programas mantendo referência a objetos criados a partir de seus módulos de classe públicos e nenhum código sendo executado.

## Instrução Exit

Sai de um bloco de código **Do...Loop**, **For...Next**, **Function**, **Sub** ou **Property**.

### Sintaxe

**Exit Do**  
**Exit For**  
**Exit Function**  
**Exit Property**  
**Exit Sub**

A sintaxe da instrução **Exit** possui as formas a seguir:

Instrução	Descrição
<b>Exit Do</b>	Provê um meio de sair de uma instrução <b>Do...Loop</b> . Pode ser utilizada somente dentro de uma instrução <b>Do...Loop</b> . <b>Exit Do</b> transfere o controle para a <u>instrução</u> seguinte à instrução <b>Loop</b> . Quando utilizada com instruções <b>Do...Loop</b> embutidas, <b>Exit Do</b> transfere o controle para o loop que está embutido em um nível acima do loop em que ocorre <b>Exit Do</b> .
<b>Exit For</b>	Provê um meio de sair de um loop <b>For</b> . Pode ser utilizado somente em um loop <b>For...Next</b> ou <b>For Each...Next</b> . <b>Exit For</b> transfere o controle para a instrução seguinte à instrução <b>Next</b> . Quando utilizada com loops <b>For</b> embutidos, <b>Exit For</b> transfere o controle para o loop que está embutido em um nível acima do loop em que ocorre <b>Exit For</b> .
<b>Exit Function</b>	Sai imediatamente do <u>procedimento</u> <b>Function</b> no qual aparece. A execução continua com a instrução seguinte à instrução que chamou o <b>Function</b> .
<b>Exit Property</b>	Sai imediatamente do procedimento <b>Property</b> no qual

**Exit Sub** aparece. A execução continua com a instrução seguinte à instrução que chamou o procedimento **Property**.  
 Sai imediatamente do procedimento **Sub** no qual aparece. A execução continua com a instrução seguinte à instrução que chamou o procedimento **Sub**.

### Comentários

Não confunda instruções **Exit** com instruções **End**. **Exit** não define o final de uma estrutura.

## Instrução For Each...Next

Repete um grupo de instruções para cada elemento de uma matriz ou coleção.

### Sintaxe

```

For          Each          elemento          In          grupo
  [instruções]
  [Exit
  [instruções]
Next [elemento]
  
```

A sintaxe da instrução **For...Each...Next** possui as partes a seguir:

Parte	Descrição
<i>elemento</i>	Obrigatória. <u>Variável</u> utilizada para iterar através dos elementos da coleção ou matriz. Para as coleções, <i>elemento</i> pode ser somente uma variável <b>Variant</b> , uma variável de objeto genérica ou qualquer variável de objeto específica. Para as matrizes, <i>elemento</i> somente pode ser uma variável <b>Variant</b> .
<i>grupo</i>	Obrigatória. Nome de uma coleção ou matriz de objetos(exceto matriz de <u>tipos definidos pelo usuário</u> ).
<i>instruções</i>	Opcional. Uma ou mais instruções que são executadas em cada item de um <i>grupo</i> .

### Comentários

O bloco **For...Each** é inserido se houver pelo menos um elemento em *grupo*. Uma vez que o loop tenha sido inserido, todas as instruções do loop são executadas para o primeiro elemento do *grupo*. Se houver mais elementos em *grupo*, as instruções do loop continuam a ser executadas para cada elemento. Quando não houver mais elementos em *grupo*, o loop sai e a execução continua com a instrução seguinte à instrução **Next**.

Qualquer número de instruções **Exit For** pode ser colocado em qualquer lugar do loop como um meio alternativo para sair. Muitas vezes, **Exit For** é utilizada depois de se avaliar alguma condição, por exemplo, **If...Then**, e transfere o controle para a instrução imediatamente seguinte a **Next**.

Você pode embutir loops **For...Each...Next** colocando um dentro de outro. Entretanto, cada *elemento* do loop deve ser exclusivo.

**Observação** Se você omitir *elemento* em uma instrução **Next** a execução continua como se *elemento* estivesse incluído. Se uma instrução **Next** for encontrada antes de sua instrução **For** correspondente, ocorrerá um erro.

Você não pode utilizar a instrução **For...Each...Next** com uma matriz de tipos definidos pelo usuário porque um **Variant** não pode conter um tipo definido pelo usuário.

## Instrução For...Next

Repete um grupo de instruções um número específico de vezes.

### Sintaxe

```

For      contador      =      início      To      fim      [Step      passo]
  [instruções]
  [Exit
  [instruções]
Next [contador]

```

A sintaxe da instrução **For...Next** possui as partes a seguir:

Parte	Descrição
<i>contador</i>	Obrigatória. <u>Variável</u> numérica utilizada como contador de loops. A variável não pode ser um <b>Boolean</b> ou um elemento de <u>matriz</u> .
<i>início</i>	Obrigatória. Valor inicial do <i>contador</i> .
<i>fim</i>	Obrigatória. Valor final do <i>contador</i> .
<i>passo</i>	Opcional. A quantidade em que o <i>contador</i> é alterado cada vez que passa pelo loop. Se o <i>passo</i> não for especificado, o padrão será um.
<i>instruções</i>	Opcional. Uma ou mais instruções entre <b>For</b> e <b>Next</b> que são executadas conforme o número de vezes especificado.

### Comentários

O argumento do *passo* pode ser positivo ou negativo. O valor do argumento do *passo* determina o processamento do loop da seguinte forma:

Valor	Loop é executado se
Positivo ou 0	$contador \leq fim$
Negativo	$contador \geq fim$

Depois que todas as instruções do loop tiverem sido executadas, *passo* é adicionado a *contador*. Neste momento ou as instruções do loop são executadas novamente (baseadas no mesmo teste que causou a execução inicial do loop) ou o loop sai e a execução continua com a instrução seguinte à instrução **Next**.

**Dica** A alteração do valor de um *contador* estando dentro de um loop pode tornar mais difícil a leitura e depuração do seu código.

Qualquer número de instruções **Exit For** pode ser colocado em qualquer lugar dentro do loop como meio alternativo de sair. Muitas vezes **Exit For** é utilizada depois da avaliação de alguma condição, por exemplo, **If...Then**, e transfere o controle para a instrução imediatamente seguinte a **Next**.

Você pode embutir loops **For...Next** colocando um dentro do outro. Dê a cada loop um nome de variável exclusivo como seu *contador*. A construção a seguir é correta:

```

For I = 1 To 10
  For J = 1 To 10
    For K = 1 To 10
      ...
    Next K
  Next J
Next I

```

**Observação** Se você omitir *contador* em uma instrução **Next**, a execução continua como se *contador* estivesse incluído. Se uma instrução **Next** for encontrada antes de sua instrução **For** correspondente, ocorrerá um erro.

## Instrução GoSub...Return

Desvia para, e retorna de uma sub-rotina dentro de um procedimento.

### Sintaxe

#### GoSub

...  
*linha*

...

#### Return

O argumento *linha* pode ser qualquer rótulo de linha ou número de linha.

*linha*

### Comentários

Você pode utilizar **GoSub** e **Return** em qualquer posição dentro de um procedimento, mas **GoSub** e a instrução correspondente **Return** devem estar no mesmo procedimento. Uma sub-rotina pode conter mais de uma instrução **Return**, mas a primeira encontrada faz com que o fluxo da execução se desvie de volta para a instrução imediatamente seguinte à instrução **GoSub** mais recentemente executada.

**Observação** Você não pode entrar ou sair de procedimentos **Sub** com **GoSub...Return**.

**Dica** Criar procedimentos separados que você possa chamar pode proporcionar uma alternativa mais estruturada para a utilização de **GoSub...Return**.

## Instrução GoTo

Desvia incondicionalmente para uma linha especificada dentro de um procedimento.

### Sintaxe

#### GoTo *linha*

O argumento *linha* requerido pode ser qualquer rótulo de linha ou número de linha.

### Comentários

**GoTo** pode desviar somente para linhas de dentro do procedimento em que aparece.

**Observação** Excesso de instruções **GoTo** pode tornar difícil a leitura e depuração do código. Sempre que possível, utilize instruções de controle estruturado (**Do...Loop**, **For...Next**, **If...Then...Else**, **Select Case**).

## Instrução If...Then...Else

Executa condicionalmente um grupo de instruções, dependendo do valor de uma expressão.

### Sintaxe

**If** *condição* **Then** [*instruções*] [**Else** *instruçõeselse*]

Ou você pode utilizar a sintaxe de formato de bloco:

<b>If</b>	<i>condição</i>	<b>Then</b>
[ <i>instruções</i> ]		
<b>[Elseif]</b>	<i>condição-n</i>	<b>Then</b>
[ <i>instruçõeselseif</i> ] . . .		
<b>[Else</b>		
[ <i>instruçõeselse</i> ]]		
<b>End If</b>		

A instrução **If...Then...Else** possui as partes a seguir:

Parte	Descrição
<i>condição</i>	Obrigatória. Um ou mais dos dois tipos de expressão seguintes: Uma <u>expressão numérica</u> ou <u>expressão de seqüência de caracteres</u> que é avaliada como <b>True</b> ou <b>False</b> . Se <i>condição</i> for <b>Null</b> , será tratada como <b>False</b> . Uma expressão do formato <b>TypeOf nomedoobjeto Is tipodoobjeto</b> . O <i>nomedoobjeto</i> é qualquer referência a objeto e <i>tipodoobjeto</i> é qualquer tipo de objeto válido. A expressão será <b>True</b> se <i>nomedoobjeto</i> for o <u>tipo de objeto</u> especificado por <i>tipodoobjeto</i> ; caso contrário, será <b>False</b> .
<i>instruções</i>	Opcional em formato de bloco; requerida em formato de uma só linha que não possui cláusula <b>Else</b> . Uma ou mais instruções separadas por dois-pontos; executada se <i>condição</i> for <b>True</b> .
<i>condição-n</i>	Opcional. Igual a <i>condição</i> .
<i>instruçõeselseif</i>	Opcional. Uma ou mais instruções executadas se a <i>condição-n</i> associada for <b>True</b> .
<i>instruçõeselse</i>	Opcional. Uma ou mais instruções executadas se nenhuma expressão <i>condição</i> ou <i>condição-n</i> anterior for <b>True</b> .

### Comentários

Você pode utilizar o formato de uma só linha (primeira sintaxe) para testes simples e curtos. Entretanto, o formato de bloco (segunda sintaxe) proporciona mais estrutura e flexibilidade do que o formato de uma só linha e normalmente é mais fácil de ler, manter e depurar.

**Observação** Com o formato de uma só linha é possível ter múltiplas instruções executadas como resultado de uma decisão **If...Then**. Todas as instruções devem estar na mesma linha e separadas por dois-pontos, como na seguinte instrução:

```
If A > 10 Then A = A + 1 : B = B + A : C = C + B
```

Uma instrução **If** em formato de bloco deve ser a primeira em uma linha. As partes **Else**, **Elseif**, e **End If** da instrução podem ter somente um número de linha ou rótulo de linha precedendo-as. O **If** em bloco deve terminar com uma instrução **End If**.

Para determinar se uma instrução é ou não **If** em bloco, examine o que vem em seguida à palavra-chave **Then**. Se qualquer coisa exceto um comentário aparecer depois de **Then** na mesma linha, a instrução será tratada como instrução **If** de uma só linha.

A cláusulas **Else** e **Elseif** são opcionais. Você pode ter tantas cláusulas **Elseif** em um bloco **If** quantas desejar, mas nenhuma pode aparecer depois de uma cláusula **Else**. Instruções **If** em bloco podem ser embutidas, isto é, contidas uma dentro da outra.

Executando um **If** em bloco (segunda sintaxe), *condição* é testada. Se *condição* for **True**, as instruções seguintes a **Then** são executadas. Se *condição* for **False**, cada condição **Elseif** (se houver) será por sua vez avaliada. Quando uma condição **True** for encontrada, as instruções imediatamente subseqüentes à **Then** associada são executadas. Se nenhuma das condições **Elseif** forem **True** (ou se não houver cláusulas **Elseif**), as instruções subseqüentes a **Else** são executadas. Depois da execução das instruções subseqüentes a **Then** ou **Else**, a execução continua com a instrução subseqüente a **End If**.

**Dica** **Select Case** pode ser mais útil na avaliação de uma expressão única com diversas ações possíveis. Entretanto, a cláusula **TypeOf nomedoobjeto Is tipodoobjeto** não pode ser utilizada com a instrução **Select Case**.

## Função If

Retorna uma das duas partes, dependendo da avaliação de uma expressão.

### Sintaxe

**If(*expr*, *truepart*, *falsepart*)**

A sintaxe da função **If** possui os argumentos nomeados a seguir:

Parte	Descrição
<b><i>expr</i></b>	Obrigatória. Expressão que você deseja avaliar.
<b><i>truepart</i></b>	Obrigatória. Valor ou expressão retornada se <b><i>expr</i></b> for <b>True</b> .
<b><i>falsepart</i></b>	Obrigatória. Valor ou expressão retornada se <b><i>expr</i></b> for <b>False</b> .

### Comentários

**If** sempre avalia ***truepart*** e ***falsepart***, mesmo que retorne somente uma delas. Em razão disto, você deve observar se ocorrem efeitos colaterais indesejáveis. Por exemplo, se a avaliação de ***falsepart*** resulta em um erro de divisão por zero, ocorrerá um erro mesmo se ***expr*** for **True**.

## Instruções On...GoSub, On...GoTo

Desviam para uma das diversas linhas especificadas, dependendo do valor de uma expressão.

### Sintaxe

**On expressão GoSub listadedestino**

**On expressão GoTo listadedestino**

A sintaxe das instruções **On...GoSub** e **On...GoTo** possui as partes a seguir:

Parte	Descrição
<i>expressão</i>	Obrigatória. Qualquer <u>expressão numérica</u> avaliada como número inteiro entre 0 e 255, inclusive. Se <i>expressão</i> for qualquer número não inteiro, ele será arredondado antes da avaliação.
<i>listadedestino</i>	Obrigatória. Lista de <u>números de linha</u> ou <u>rótulos de linha</u> separados por vírgulas.

### Comentários

O valor da *expressão* determina a linha para a qual será desviada na *listadedestino*. Se o valor for menor que 1 ou maior que o número de itens da lista, ocorrerá um dos seguintes resultados:

Se <i>expressão</i> for	Então
Igual a 0	O controle cai para a <u>instrução</u> subsequente a <b>On...GoSub</b> ou <b>On...GoTo</b> .
Maior que o número de itens de uma lista	O controle cai para a instrução subsequente a <b>On...GoSub</b> ou <b>On...GoTo</b> .
Negativo	Ocorre um erro.
Maior que 255	Ocorre um erro.

Você pode misturar números de linha e rótulos de linha na mesma lista. Pode utilizar quantos rótulos de linha e números de linha desejar com **On...GoSub** e **On...GoTo**. Entretanto, se você utilizar mais rótulos ou números do que cabem em uma só linha, deverá utilizar o caractere de continuação de linha para continuar a linha lógica até a linha física seguinte.

**Dica** **Select Case** proporciona um meio mais estruturado e mais flexível de executar múltiplo desvio.

## Função Partition

Retorna um **Variant (String)** que indica onde ocorre um número dentro de uma série calculada de intervalos.

### Sintaxe

**Partition(number, start, stop, interval)**

A sintaxe da função **Partition** possui os argumentos nomeados a seguir:

Parte	Descrição
<b>number</b>	Obrigatório. Número inteiro que você deseja avaliar em relação aos intervalos.
<b>start</b>	Obrigatório. Número inteiro que é o início do intervalo geral de números. Não pode ser menor que 0.
<b>stop</b>	Obrigatório. Número inteiro que é o final do intervalo geral de números. Não pode ser igual nem menor que <b>start</b> .
<b>interval</b>	Obrigatório. Número inteiro que é a faixa coberta por cada segmento da série de <b>start</b> a <b>stop</b> . Não pode ser menor que 1.

### Comentários

A função **Partition** identifica o intervalo específico no qual **number** está e retorna um **Variant (String)** que descreve esse intervalo. A função **Partition** é mais útil em consultas. Você pode criar uma consulta seleção que mostra quantas ordens estão dentro de diversos intervalos, por exemplo, valores de ordem de 1 a 1000, 1001 a 2000, e assim por diante.

A tabela a seguir mostra como os intervalos são determinados utilizando-se três conjuntos de partes **start**, **stop**, e **interval**. As colunas Primeiro Intervalo e Último Intervalo mostram o que **Partition** retorna. Os intervalos são representados por *valormínimo:valormáximo*, onde o extremo inferior (*valormínimo*) do intervalo é separado do extremo superior (*valormáximo*) por dois-pontos (:).

<b>start</b>	<b>stop</b>	<b>interval</b>	Antes do primeiro	Primeiro intervalo	Último intervalo	Depois do último
0	99	5	" :-1"	" 0: 4"	" 95: 99"	" 100: "
20	199	10	" : 19"	" 20: 29"	" 190: 199"	" 200: "
100	1010	20	" : 99"	" 100: 119"	" 1000: 1010"	" 1011: "

Na tabela acima, a terceira linha mostra o resultado de quando **start** e **stop** definem um conjunto de números que não podem ser divididos exatamente por **interval**. O último intervalo estende-se até **stop** (11 números) embora **interval** seja 20.

Se for necessário, **Partition** retorna um intervalo com espaços suficientes para que haja o mesmo número de caracteres à esquerda e à direita de dois-pontos que o número de caracteres em **stop** mais um. Isto garante que, se você utilizar **Partition** com outros números, o texto resultante será manipulado adequadamente durante qualquer operação de classificação subsequente.

Se **interval** for 1, o intervalo será **number:number**, independente dos argumentos de **start** e **stop**. Por exemplo, se **interval** for 1, **number** for 100 e **stop** for 1000, **Partition** retornará " 100: 100".

Se qualquer das partes for **Null**, **Partition** retornará um **Null**.

## Instrução Select Case

Executa um dos diversos grupos de instruções, dependendo do valor de uma expressão.

### Sintaxe

```

Select
  [Case
    [instruções-n]]
  [Case
    [instruçõeselse]]
End Select

```

*expressãodeteste*  
*listadeexpressões-n*  
...  
**Else**

A sintaxe da instrução **Select Case** possui as partes a seguir:

Parte	Descrição
<i>expressãodeteste</i> <i>e</i>	Obrigatória. Qualquer <u>expressão numérica</u> ou <u>expressão de seqüência</u> .
<i>listadeexpressões-n</i>	Obrigatória se aparecer uma <b>Case</b> . Lista delimitada de uma ou mais das seguintes formas: <i>expressão</i> , <i>expressão To expressão</i> , <i>Is operadordecomparação expressão</i> . A <u>palavra-chave To</u> especifica um intervalo de valores. Se você utilizar essa palavra-chave, o valor menor deve aparecer antes de <b>To</b> . Utilize a palavra-chave <b>Is</b> com <u>operadores de comparação</u> (exceto <b>Is</b> e <b>Like</b> ) para especificar um intervalo de valores. Se não for fornecida, a palavra-chave <b>Is</b> será inserida automaticamente.
<i>Instruções-n</i>	Opcional. Uma ou mais instruções são executadas se <i>expressãodeteste</i> coincidir com qualquer parte de <i>listadeexpressões-n</i> .
<i>instruçõeselse</i>	Opcional. Uma ou mais instruções são executadas se <i>expressãodeteste</i> não coincidir com qualquer das cláusulas <b>Case</b> .

### Comentários

Se *expressãodeteste* coincidir com qualquer expressão **Case** *listadeexpressões*, as *instruções* subseqüentes àquela cláusula **Case** serão executadas até e próxima cláusula **Case**, ou, para a última cláusula, até **End Select**. Então o controle passa para a instrução seguinte a **End Select**. Se *expressãodeteste* coincidir com uma expressão *listadeexpressões* em mais de uma cláusula **Case**, somente as instruções subseqüentes à primeira coincidência serão executadas.

A cláusula **Case Else** é utilizada para indicar a *instruçõeselse* a ser executada se não for encontrada coincidência entre *expressãodeteste* e uma *listadeexpressões* em qualquer das outras seleções de **Case**. Embora não seja necessário, é uma boa idéia ter uma instrução **CaseElse** no seu bloco **Select Case** para manipular valores não previstos de *expressãodeteste*. Se nenhuma **Case** *listadeexpressões* coincidir com *expressãodeteste* e não houver instrução **Case Else**, a execução continua na instrução subseqüente a **End Select**.

Você pode utilizar múltiplas expressões ou intervalos em cada cláusula **Case**. Por exemplo, a linha a seguir é válida:

```
Case 1 To 4, 7 To 9, 11, 13, Is > MaxNumber
```

**Observação** O operador de comparação **Is** não é igual à palavra-chave **Is** utilizada na instrução **Select Case**.

Você pode também especificar intervalos e várias expressões para seqüências de caracteres. No exemplo a seguir, **Case** coincide com seqüências que são exatamente iguais a `tudo`, seqüências que ficam entre `nozes` e `sopa` em ordem alfabética e o valor atual de `TestItem`:

```
Case "tudo", "nozes" To "sopa", TestItem
```

Instruções **Select Case** podem ser embutidas. Cada instrução **Select Case** embutida deve ter uma declaração **End Select** correspondente.

## Função Shell

Executa um programa executável e retorna um **Variant (Double)** que representa o código da tarefa do programa, se tiver sucesso, caso contrário, retorna zero.

### Sintaxe

**Shell**(*pathname*[,*windowstyle*])

A sintaxe da função **Shell** possui os argumentos nomeados a seguir:

Parte	Descrição
<b>pathname</b>	Obrigatório; <b>Variant (String)</b> . Nome do programa a ser executado e qualquer opção de <u>argumento</u> ou <u>linha de comando</u> obrigatória; pode incluir diretório ou pasta e unidade de disco.
<b>windowstyle</b>	Opcional. <b>Variant (Integer)</b> correspondente ao estilo da janela na qual o programa deve ser executado. Se <b>windowstyle</b> for omitido, o programa iniciará minimizado com foco.

O argumento nomeado **windowstyle** possui os valores a seguir:

Constante	Valor	Descrição
<b>vbHide</b>	0	A janela é ocultada e o foco é passado para a janela oculta.
<b>VbNormalFocus</b>	1	A janela tem foco e é restaurada para seu tamanho e posição originais.
<b>vbMinimizedFocus</b>	2	A janela é exibida como um ícone com foco.
<b>vbMaximizedFocus</b>	3	A janela é maximizada com foco.
<b>vbNormalNoFocus</b>	4	A janela é restaurada para seu tamanho e posição mais recentes. A janela ativa atual permanece ativa.
<b>vbMinimizedNoFocus</b>	6	A janela é exibida como um ícone. A janela atualmente ativa permanece ativa.

### Comentários

Se a função **Shell** executar o arquivo nomeado com sucesso, retornará o código da tarefa do programa iniciado. A identificação da tarefa é um número exclusivo que identifica o programa em execução. Se a função **Shell** não puder iniciar o programa nomeado, ocorrerá um erro. Se você utilizar a função **MacID** com **Shell** no Microsoft Windows, ocorrerá um erro.

**Observação** A função **Shell** executa outros programas de forma assíncrona. Isto significa que um programa iniciado com **Shell** poderá não completar a execução antes que as instruções subseqüentes à função **Shell** sejam executadas.

## Instrução Stop

Suspende a execução.

### Sintaxe

#### Stop

### Comentários

Você pode colocar instruções **Stop** em qualquer lugar dos procedimentos para suspender a execução. A utilização da instrução **Stop** assemelha-se a definir um ponto de interrupção no código.

A instrução **Stop** suspende a execução, mas, ao contrário de **End**, não fecha qualquer arquivo nem limpa variáveis, a menos que esteja em um arquivo executável (.exe) compilado.

## Função Switch

Avalia uma lista de expressões e retorna um valor de **Variant** ou uma expressão associada com a primeira expressão da lista que seja **True**.

### Sintaxe

**Switch**(*expr-1*, *valor-1* [, *expr-2*, *valor-2* ... [, *expr-n*, *valor-n*]])

A sintaxe da função **Switch** possui as partes a seguir:

Parte	Descrição
<i>expr</i>	Obrigatória. <u>expressão <b>Variant</b></u> que você deseja avaliar.
<i>valor</i>	Obrigatória. Valor ou expressão a ser retornada se a expressão correspondente for <b>True</b> .

### Comentários

A lista de argumentos da função **Switch** consiste em pares de expressões e valores. As expressões são avaliadas da esquerda para a direita e o valor associado à primeira expressão avaliada como **True** é retornado. Se as partes não estiverem adequadamente dispostas em par, ocorrerá um erro em tempo de execução. Por exemplo, se *expr-1* for **True**, **Switch** retornará *valor-1*. Se *expr-1* for **False**, mas *expr-2* for **True**, **Switch** retornará *valor-2*, e assim por diante.

**Switch** retorna um valor **Null** se:

- Nenhuma das expressões for **True**.
- A primeira expressão **True** tiver um valor correspondente que seja **Null**.

**Switch** avalia todas as expressões, mesmo que retorne apenas uma delas. Por isso você deve observar se ocorrem efeitos colaterais indesejáveis. Por exemplo, se a avaliação de qualquer expressão resultar em um erro de divisão por zero, ocorrerá um erro.

## Instrução While...Wend

Executa uma série de instruções desde que uma dada condição seja **True**.

### Sintaxe

#### While

[*instruções*]

*condição*

#### Wend

A sintaxe da instrução **While...Wend** possui as partes a seguir:

Parte	Descrição
<i>condição</i>	Obrigatória. <u>Expressão numérica</u> ou <u>expressão de seqüência</u> que é avaliada como <b>True</b> ou <b>False</b> . Se <i>condição</i>

for **Null**, será tratada como **False**.  
*instruções* Opcional. Uma ou mais instruções que são executadas enquanto a condição for **True**.

### Comentários

Se *condição* for **True**, todas as *instruções* serão executadas até que a instrução **Wend** seja encontrada. Então o controle retorna para **While** e *condição* é novamente verificada. Se *condição* ainda for **True**, o processo é repetido. Se não for **True**, a execução continua com a instrução subsequente à instrução **Wend**.

Os loops **While...Wend** podem ser embutidos até qualquer nível. Cada **Wend** corresponde ao mais recente **While**.

**Dica** A instrução **Do...Loop** proporciona um meio mais estruturado e flexível para executar um loop.

## Instrução With

Executa uma série de instruções em um único objeto ou um tipo definido pelo usuário.

### Sintaxe

#### With

[*instruções*]

*objeto*

#### End With

A instrução **With** possui as partes a seguir:

Parte	Descrição
<i>objeto</i>	Obrigatória. Nome de um objeto ou de um tipo definido pelo usuário.
<i>instruções</i>	Opcional. Uma ou mais instruções a serem executadas com <i>objeto</i> .

### Comentários

A instrução **With** permite executar uma série de instruções com um objeto específico sem requalificar o nome do objeto. Por exemplo, para alterar diversas propriedades diferentes de um único objeto, coloque as instruções de atribuição de propriedades dentro da estrutura de controle de **With**, referindo o objeto uma só vez em lugar de fazê-lo em todas as atribuições de propriedade. O exemplo a seguir ilustra a utilização da instrução **With** para atribuir valores a diversas propriedades do mesmo objeto.

```
With MyLabel
    .Height = 2000
    .Width = 2000
    .Caption = "Este é MyLabel"
End With
```

**Observação** Uma vez que um bloco **With** seja inserido, *objeto* não pode ser modificado. Como resultado, você não pode utilizar uma só instrução **With** para afetar diversos objetos diferentes.

Você pode embutir instruções **With** colocando blocos **With** um dentro do outro. Entretanto, em razão de os membros dos blocos externos de **With** ficarem mascarados dentro dos blocos internos, você deve proporcionar uma referência a objeto totalmente qualificada em um bloco **With** interno para qualquer membro de um objeto existente em um bloco externo.

**Importante** Não salte para dentro e para fora de blocos **With**. Se as instruções em um bloco **With** forem executadas mas a instrução **With** ou **End With** não for, você poderá provocar erros ou comportamento imprevisível.

### Exemplo da função Asc

Este exemplo utiliza a função **Asc** para retornar um código de caractere que corresponde à primeira

letra na seqüência de caracteres.

```
Dim MeuNúmero
MeuNúmero = Asc("A") ' Retorna 65.
MeuNúmero = Asc("a") ' Retorna 97.
MeuNúmero = Asc("Amora") ' Retorna 65.
```

### Exemplo da função CBool

Este exemplo utiliza a função **CBool** para converter uma expressão em um **Boolean**. Se a expressão avaliar como um valor diferente de zero, **CBool** retornará **True**; caso contrário, retornará **False**.

```
Dim A, B, Controle
A = 5: B = 5 ' Inicializa variáveis.
Controle = CBool(A = B) ' Controle contém True.

A = 0 ' Define a variável.
Controle = CBool(A) ' Controle contém False.
```

### Exemplo da função CByte

Este exemplo utiliza a função **CByte** para converter uma expressão em um **Byte**.

```
Dim MeuDouble, MeuByte
MeuDouble = 125.5678 ' MeuDouble é um Double.
MeuByte = CByte(MeuDouble) ' MeuByte contém 126.
```

### Exemplo da função CDate

Este exemplo utiliza a função **CDate** para converter uma seqüência de caracteres em um **Date**. Em geral, datas e horas definidas no programa como seqüências de caracteres (conforme mostrado neste exemplo) não são recomendadas. Em vez disso, utilize literais de data e literais de hora, como #2/12/1969# e #4:45:23 PM#.

```
Dim MinhaData, MinhaDataAbreviada, MinhaHora, MinhaHoraAbreviada
MinhaData = "12 de Fevereiro de 1969" ' Define a data.
MinhaDataAbreviada = CDate(MinhaData) ' Converte no tipo de dados Date.

MinhaHora = "4:35:47 PM" ' Define a hora.
MinhaHoraAbreviada = CDate(MinhaHora) ' Converte no tipo de dados Date.
```

### Exemplo da função CCur

Este exemplo utiliza a função **CCur** para converter uma expressão em um **Currency**.

```
Dim MeuDouble, MeuCurr
MeuDouble = 543.214588 ' MeuDouble é um Double.
MeuCurr = CCur(MeuDouble * 2) ' Converte o resultado de MeuDouble * 2
    ' (1086.429176) em um
    ' Currency (1086.4292).
```

### Exemplo da função CDb1

Este exemplo utiliza a função **CDb1** para converter uma expressão em um **Double**.

```
Dim MeuCurr, MeuDouble
MeuCurr = CCur(234.456784) ' MeuCurr é um Currency.
MeuDouble = CDb1(MeuCurr * 8.2 * 0.01) ' Converte o resultado em um
Double.
```

### Exemplo da função CInt

Este exemplo utiliza a função **CInt** para converter um valor em um **Integer**.

```
Dim MeuDouble, MeuInt
MeuDouble = 2345.5678 ' MeuDouble é um Double.
MeuInt = CInt(MeuDouble) ' MeuInt contém 2346.
```

### Exemplo da função CLng

Este exemplo utiliza a função **CLng** para converter um valor em um **Long**.

```
Dim MeuVal1, MeuVal2, MeuLong1, MeuLong2
MeuVal1 = 25427.45: MeuVal2 = 25427.55 ' MeuVal1 e MeuVal2 são Doubles.
MeuLong1 = CLng(MeuVal1) ' MeuLong1 contém 25427.
MeuLong2 = CLng(MeuVal2) ' MeuLong2 contém 25428.
```

### Exemplo da função CSng

Este exemplo utiliza a função **CSng** para converter um valor em um **Single**.

```
Dim MeuDouble1, MeuDouble2, MeuSingle1, MeuSingle2
' MeuDouble1, MeuDouble2 são Doubles.
MeuDouble1 = 75.3421115: MeuDouble2 = 75.3421555
MeuSingle1 = CSng(MeuDouble1) ' MeuSingle1 contém 75,34211.
MeuSingle2 = CSng(MyDouble2) ' MeuSingle2 contém 75,34216.
```

### Exemplo da função CStr

Este exemplo utiliza a função **CStr** para converter um valor numérico em uma **String**.

```
Dim MeuDouble, MinhaSeqüência
MeuDouble = 437.324 ' MeuDouble é um Double.
MinhaSeqüência = CStr(MeuDouble) ' MinhaSeqüência contém "437,324".
```

### Exemplo da função CVar

Este exemplo utiliza a função **CVar** para converter uma expressão em uma **Variant**.

```
Dim MeuInt, MinhaVar
MeuInt = 4534 ' MeuInt é um Integer.
MinhaVar = CVar(MeuInt & "000") ' MinhaVar contém a seqüência de
caracteres
' 4534000.
```

### Exemplo da função CVErr

Este exemplo utiliza a função **CVErr** para retornar uma **Variant** cujo **VarType** é **vbError** (10). A função definida pelo usuário `CalculaDouble` retorna um erro se o argumento passado a ela não for um número. Você pode utilizar **CVErr** para retornar erros definidos pelo usuário de procedimentos definidos pelo usuário ou para adiar o tratamento de um erro em tempo de execução. Utilize a função **IsError** para testar se o valor representa um erro.

```
' Chama CalculaDouble com um argumento de produção de erro.
Sub Test()
    Debug.Print CalculaDouble("345.45robert")
End Sub
' Define o procedimento da função CalculaDouble.
Function CalculaDouble(Número)
    If IsNumeric(Número) Then
        CalculaDouble = Número * 2 ' Retorna o resultado.
    Else
        CalculaDouble = CVErr(2001) ' Retorna um número de erro
    End If ' definido pelo usuário.
End Function
```

### Exemplo da função Val

Este exemplo utiliza a função **Val** para retornar os números contidos em uma seqüência de caracteres.

```
Dim MeuValor
MeuValor = Val("2457") ' Retorna 2457.
MeuValor = Val(" 2 45 7") ' Retorna 2457.
MeuValor = Val("24 com 57") ' Retorna 24.
```

## Funções de conversão

### Função Asc

Retorna um **Integer** que representa o código de caractere correspondente à primeira letra de uma seqüência.

#### Sintaxe

**Asc**(seqüência)

O argumento obrigatório seqüência pode ser qualquer expressão de seqüência válida. Se a seqüência não contiver caracteres, um erro em tempo de execução será gerado.

#### Comentários

O intervalo para os retornos é de 0 a 255 em sistemas não-DBCS, mas de -32768 a 32767 em sistemas DBCS.

**Observação** A função **AscB** é utilizada com os dados de byte contidos em uma seqüência. Em vez de retornar o código de caractere para o primeiro caractere, **AscB** retorna o primeiro byte. A função **AscW** retorna o código de caractere Unicode, exceto em plataformas nas quais o Unicode não é suportado, caso em que o comportamento dessa função é idêntico ao da função **Asc**.

## Função CVer

Retorna um **Variant** de subtipo **Error** contendo um número de erro especificado pelo usuário.

### Sintaxe

**CVer**(*númerodoerro*)

O argumento obrigatório *númerodoerro* pode ser qualquer número de erro válido.

### Comentários

Utilize a função **CVer** para criar erros definidos pelo usuário nos procedimentos criados pelo usuário. Por exemplo, se você criar uma função que aceite vários argumentos e normalmente retorne uma seqüência, você poderá fazer com que sua função avalie os argumentos de entrada para assegurar de que eles se encontram dentro do intervalo aceitável. Se não se encontrarem, é provável que sua função não retorne o esperado. Nesse caso, **CVer** permite retornar um número de erro que informa que atitude tomar.

Observe que a conversão implícita de um **Error** não é permitida. Por exemplo, você não pode atribuir diretamente o valor de retorno de **CVer** a uma variável que não é **Variant**. Contudo, pode efetuar uma conversão explícita (utilizando **CInt**, **CDBl** e assim por diante) do valor retornado por **CVer** e atribuí-la a uma variável do tipo de dados apropriado.

## Função Val

Retorna os números contidos em uma seqüência como um valor numérico do tipo apropriado.

### Sintaxe

**Val**(*seqüência*)

O argumento obrigatório *seqüência* pode ser qualquer expressão de seqüência válida.

### Comentários

A função **Val** interrompe a leitura da seqüência no primeiro caractere que não puder reconhecer como parte de um número. Os símbolos e os caracteres que são geralmente considerados partes de valores numéricos, como o cifrão e as vírgulas, não são reconhecidos. Contudo, a função reconhece os prefixos de raiz &O (para octal) e &H (para hexadecimal). Os caracteres em branco, as tabulações e a alimentação de linha são retirados do argumento.

O exemplo a seguir retorna o valor 1615198:

```
Val(" 1615 198th Street N.E.")
```

No código abaixo, **Val** retorna o valor decimal -1 para o valor hexadecimal apresentado:

```
Val("&HFFFF")
```

**Observação** A função **Val** reconhece somente o ponto (.) como um separador decimal válido. Quando separadores decimais diferentes puderem ser utilizados, por exemplo, em aplicativos que não estejam em inglês, utilize **CDBl** ao invés de converter uma seqüência em um número.

## Funções de conversão de tipo

Cada função força a conversão de uma expressão em um tipo de dados específico.

### Sintaxe

**CBool**(*expressão*)

**CByte**(*expressão*)

**CCur**(*expressão*)

**CDate**(*expressão*)

**CDBl**(*expressão*)

**CDec**(expressão)**CInt**(expressão)**CLng**(expressão)**CSng**(expressão)**CVar**(expressão)**CStr**(expressão)

O argumento obrigatório *expressão* pode ser qualquer expressão de seqüência ou expressão numérica.

## Tipos de retorno

O nome da função determina o tipo de retorno como mostrado a seguir:

Função	Tipo retorno	de Intervalo do argumento <i>expressão</i>
<b>CBool</b>	<b><u>Boolean</u></b>	Qualquer seqüência ou expressão numérica válida.
<b>CByte</b>	<b><u>Byte</u></b>	de 0 a 255.
<b>CCur</b>	<b><u>Currency</u></b>	de -922.337.203.685.477,5808 a 922.337.203.685.477,5807.
<b>CDate</b>	<b><u>Date</u></b>	Qualquer <u>expressão de data</u> válida.
<b>CDbl</b>	<b><u>Double</u></b>	de -1,79769313486232E308 a -4,94065645841247E-324 para valores negativos; de 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos.
<b>CDec</b>	<b><u>Decimal</u></b>	+/-79.228.162,514.264.337.593.543.950.335 para números inteiros, isto é, números sem casas decimais. Para números com 28 casas decimais, o intervalo é +/-7,9228162514264337593543950335. O menor número possível diferente de zero é 0,00000000000000000000000001.
<b>CInt</b>	<b><u>Integer</u></b>	de -32.768 a 32.767; as frações são arredondadas.
<b>CLng</b>	<b><u>Long</u></b>	de -2.147.483.648 a 2.147.483.647; as frações são arredondadas.
<b>CSng</b>	<b><u>Single</u></b>	de -3,402823E38 a -1,401298E-45 para valores negativos; de 1,401298E-45 a 3,402823E38 para valores positivos.
<b>CVar</b>	<b><u>Variant</u></b>	Mesmo intervalo de <b>Double</b> para valores numéricos. Mesmo intervalo de <b>String</b> para valores não-numéricos.
<b>CStr</b>	<b><u>String</u></b>	<u>Os retornos para CStr</u> dependem do argumento da <i>expressão</i> .

## Comentários

Se a *expressão* passada para a função estiver fora do intervalo para o qual o tipo de dados está sendo convertido, um erro será gerado.

Em geral, você pode documentar seu código utilizando as funções de conversão de tipo de dados para mostrar que o resultado de algumas operações deve ser expressado como um tipo de dados em particular ao invés do tipo de dados padrão. Por exemplo, utilize **CCur** para forçar o cálculo monetário nos casos onde normalmente ocorreriam cálculos de precisão única, precisão dupla ou de número inteiro.

Você deve utilizar as funções de conversão de tipo de dados em vez de **Val** para fornecer conversões

de dados internacionais de um tipo de dados para outro. Por exemplo, quando você utiliza **CCur**, separadores decimais diferentes, separadores de milhares diferentes e várias opções de moeda são reconhecidas apropriadamente dependendo da definição da localidade em seu computador.

Quando uma parte fracionária equivaler exatamente a 0,5, **CInt** e **CLng** sempre a arredondarão para o número par mais próximo. Por exemplo, 0,5 é arredondado para 0 e 1,5 para 2. **CInt** e **CLng** diferem das funções **Fix** e **Int**, que truncam, ao invés de arredondar, a parte fracionária de um número. Além disso, **Fix** e **Int** sempre retornam um valor do mesmo tipo do valor que é passado para eles.

Utilize **IsDate** para determinar se a *data* pode ser convertida para uma data ou hora. **CDate** reconhece os literals de data e de hora assim como alguns números contidos no intervalo de datas aceitáveis. Ao converter um número em data, todo o número é convertido em uma data. Qualquer parte fracionária do número é convertida em uma hora do dia, começando à meia-noite.

**CDate** reconhece os formatos de data de acordo com a definição de localidade de seu sistema. A ordem correta de dia, mês e ano pode não ser determinada se for fornecida em um formato diferente das definições de data reconhecidas. Além disso, o formato de data não é reconhecido se também contiver o nome do dia da semana.

Uma função **CVDate** também é fornecida para se obter compatibilidade com as versões anteriores do Visual Basic. A sintaxe da função **CVDate** é idêntica à da função **Cdate**. Entretanto, **CVDate** retorna um **Variant** cujo subtipo é **Date** em vez de um tipo **Date** real. Como já existe um tipo de data **Date** intrínseca, não haverá mais necessidade de **CVDate**. O mesmo efeito pode ser alcançado convertendo uma expressão em **Date**, e, em seguida, atribuindo-lhe um **Variant**. Essa técnica é consistente com a conversão de todos os outros tipos intrínsecos a seus subtipos de **Variant** equivalentes.

**Observação** A função **CDec** não retorna um tipo de dados discreto; ao contrário, ela sempre retorna um **Variant** cujo valor tenha sido convertido em um subtipo **Decimal**.

#### Retornos de CStr

Se a expressão for	CStr retorna
<b>Boolean</b>	Uma seqüência contendo <b>True</b> ou <b>False</b>
<b>Date</b>	Uma seqüência contendo uma data no formato curto de seu sistema
<b>Null</b>	Um erro em tempo de execução
<b>Empty</b>	Uma seqüência de comprimento zero ("")
<b>Error</b>	Uma seqüência contendo a palavra <b>Error</b> seguida pelo <u>número do erro</u>
Outros valores numéricos	Uma seqüência contendo o número

## Tipo de dados Boolean

As variáveis Boolean são armazenadas como números de 16 bits (2 bytes), mas podem conter somente os valores **True** ou **False**. As variáveis **Boolean** são exibidas como `True` ou `False` (quando é usado **Print**) ou `#TRUE#` ou `#FALSE#` (quando é usado **Write #**). Use as palavras-chave True e False para atribuir um dos dois estados a variáveis **Boolean**.

Quando outros tipos numéricos são convertidos em valores **Boolean**, 0 torna-se **False** e todos os outros valores tornam-se **True**. Quando os valores **Boolean** são convertidos para outros tipos de dados, **False** torna-se 0 e **True** torna-se -1.

## Tipo de dados Byte

As variáveis Byte são armazenadas como números de 8 bits (1 byte), sem sinal, únicos, que variam em valor de 0 a 255.

O tipo de dados Byte é útil para conter dados binários.

## Tipo de dados Currency

As variáveis Currency são armazenadas como números de 64 bits (8 bytes) no formato de número inteiro, com escala de 10.000 para fornecer um número de vírgula fixa com 15 dígitos à esquerda do ponto decimal e 4 dígitos à direita. Essa representação proporciona um intervalo de -922.337.203.685.477,5808 a 922.337.203.685.477,5807. O caractere de declaração de tipo referente a **Currency** é o sinal (@).

O tipo de dados Currency é útil para cálculos que envolvem dinheiro e cálculos de vírgula fixa, nos quais a precisão é especialmente importante.

## Resumo dos tipos de dados

A tabela a seguir mostra os tipos de dados suportados, incluindo tamanhos e intervalos de armazenamento.

<b>Tipo de dados</b>	<b>Tamanho de armazenamento</b>	<b>Intervalo</b>
<b>Byte</b>	1 byte	0 a 255
<b>Boolean</b>	2 bytes	<b>True</b> ou <b>False</b>
<b>Integer</b>	2 bytes	-32.768 a 32.767
<b>Long</b> (inteiro longo)	4 bytes	-2.147.483.648 a 2.147.483.647
<b>Single</b> (vírgula flutuante de precisão simples)	4 bytes	-3,402823E38 a -1,401298E-45 para valores negativos; 1,401298E-45 a 3,402823E38 para valores positivos
<b>Double</b> (vírgula flutuante de precisão dupla)	8 bytes	-1,79769313486232E308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos
<b>Currency</b> (inteiro escalado)	8 bytes	-922.337.203.685.477,5808 a 922.337.203.685.477,5807
<b>Decimal</b>	14 bytes	+/-79.228.162.514.264.337.593.543.950.335 sem ponto decimal; +/-7,9228162514264337593543950335 com 28 casas decimais à direita; o menor número diferente de zero é

			+/-0,00000000000000000000000000000001.
<b>Date</b>	8 bytes		De 1º de Janeiro de 100 até 31 de Dezembro de 9999
<b>Object</b>	4 bytes		Qualquer referência a <b>Object</b>
<b>String</b> (comprimento da variável)	10 bytes + comprimento da seqüência de caracteres		De 0 até aproximadamente 2 bilhões
<b>String</b> (comprimento fixo)	Comprimento da seqüência de caracteres		De 1 até aproximadamente 65.400
<b>Variant</b> (com números)	16 bytes		Qualquer valor numérico até o intervalo de um <b>Double</b>
<b>Variant</b> (com caracteres)	22 bytes + comprimento da seqüência de caracteres		O mesmo intervalo que <b>String</b> de comprimento variável
Definido pelo usuário (usando <b>Type</b> )	Número requerido por elementos		O intervalo de cada elemento é o mesmo que o intervalo do seu tipo de dados.

**Observação** Matrizes de qualquer tipo de dados requerem 20 bytes de memória, mais 4 bytes para cada dimensão da matriz, mais o número de bytes ocupados pelos próprios dados. A memória ocupada pelos dados pode ser calculada multiplicando-se o número de elementos de dados pelo tamanho de cada elemento. Por exemplo, os dados em uma matriz de dimensão única consistindo de 4 elementos de dados **Integer** de 2 bytes cada, ocupa 8 bytes. Os 8 bytes exigidos para os dados, mais os 24 bytes fixos, fazem com que o requisito de memória para a matriz seja de 32 bytes.

Um **Variant** contendo uma matriz requer 12 bytes a mais do que uma matriz sozinha.

## Tipo de dados Date

As variáveis **Date** são armazenadas como números IEEE de vírgula flutuante de 64 bits (8 bytes) que representam as datas que variam de 1 de janeiro do ano 100 até 31 de dezembro de 9999 e as horas que variam de 0:00:00 até 23:59:59. Qualquer valor literal de data reconhecível pode ser atribuído a variáveis **Date**. Os literais **Date** devem estar entre sinais (#), por exemplo, #January 1, 1993# ou #1 Jan 93#.

As variáveis **Date** exibem as datas de acordo com o formato abreviado de data reconhecido pelo seu computador. As horas são exibidas de acordo com o formato de hora (de 12 ou de 24 horas) reconhecido pelo seu computador.

Quando outros tipos numéricos são convertidos em **Date**, os valores à esquerda do decimal representam as informações de data, enquanto os valores à direita representam a hora. Meia-noite é 0 e meio-dia é 0,5. Os números negativos inteiros representam datas anteriores a 30 de dezembro de 1899.

## Tipo de dados Decimal

As variáveis **Decimal** são armazenadas como números inteiros, sem sinal, de 96 bits (12 bytes), escalados por uma potência de 10 variável. O fator de escala potência de 10 especifica o número de dígitos à direita da vírgula decimal e varia de 0 a 28. Com uma escala de 0 (sem casas decimais), o maior valor possível é +/-79.228.162.514.264.337.593.543.950.335. Com 28 casas decimais, o maior valor é +/-7,9228162514264337593543950335 e o menor valor, diferente de zero, é +/-0,00000000000000000000000000000001.

**Observação** Por ora, o tipo de dados **Decimal** pode ser usado somente dentro de um Variant, ou seja, você não pode declarar uma variável como sendo do tipo **Decimal**. Você pode, entretanto, criar um **Variant** cujo subtipo seja **Decimal** usando a função **CDec**.

## Tipo de dados Double

As variáveis **Double** (vírgula flutuante de precisão dupla) são armazenadas como números IEEE de vírgula flutuante de 64 bits (8 bytes), com valor no intervalo -1,79769313486232E308 a -4,94065645841247E-324 para valores negativos, e de 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos. O caractere de declaração de tipo para **Double** é o símbolo (#).

## Tipo de dados Integer

As variáveis **Integer** são armazenadas como números de 16 bits (2 bytes) com valor no intervalo de -32.768 a 32.767. O caractere de declaração de tipo para **Integer** é o símbolo de porcentagem (%).

As variáveis **Integer** também podem ser usadas para representar valores enumerados. Um valor enumerado pode conter um conjunto finito de números inteiros, cada qual possuindo um significado especial no contexto em que é usado. Os valores enumerados oferecem uma forma conveniente de selecionar entre um número de opções conhecidas, por exemplo, preto = 0, branco = 1, e assim por diante. É boa prática de programação definir constantes usando a instrução **Const** para cada valor enumerado.

## Tipo de dados Long

As variáveis **Long** (inteiro longo) são armazenadas como números de 32 bits (4 bytes) assinalados no intervalo de -2.147.483.648 a 2.147.483.647. O caractere de declaração de tipo referente a **Long** é o ampersand (&).

## Tipo de dados Object

As variáveis **Object** são armazenadas como endereços de 32 bits (4 bytes) que se referem a objetos. Usando a instrução **Set**, uma variável declarada como um **Object** pode ter qualquer referência de objeto atribuída a ela.

**Observação** Embora uma variável declarada com o tipo **Object** seja flexível o suficiente para conter uma referência a qualquer objeto, a ligação ao objeto referido por aquela variável é sempre atrasada (ligação por tempo de execução). Para forçar a ligação adiantada (ligação por tempo de compilação), atribua a referência do objeto a uma variável declarada com um nome de classe específico.

## Tipo de dados Single

As variáveis **Single** (de vírgula flutuante de precisão única) são armazenadas como números IEEE de vírgula flutuante de 32 bits (4 bytes), com valor no intervalo de -3,402823E38 a -1,401298E-45 para valores negativos e de 1,401298E-45 a 3,402823E38 para valores positivos. O caractere de declaração de tipo para **Single** é o ponto de exclamação (!).

## Tipo de dados String

Existem dois tipos de seqüências de caracteres: de comprimento variável e de comprimento fixo.

- Uma seqüência de comprimento variável pode conter até aproximadamente 2 bilhões ( $2^{31}$ ) de caracteres.
- Uma seqüência de comprimento fixo pode conter de 1 até aproximadamente 64K ( $2^{16}$ ) caracteres.

**Observação** Uma seqüência de caracteres de comprimento fixo **Public** não pode ser usada em um módulo de classe.

Os códigos para caracteres **String** variam de 0 a 255. Os primeiros 128 caracteres (0–127) do conjunto de caracteres correspondem às letras e símbolos de um teclado padrão americano. Esses 128 primeiros caracteres são os mesmos definidos pelo conjunto de caracteres ASCII. Os 128 caracteres restantes (128–255) representam caracteres especiais, como letras de alfabetos internacionais, acentos, símbolos monetários e frações. O caractere de declaração de tipo para **String** é o cifrão (\$).

## Tipo de dados definido pelo usuário

É qualquer tipo de dados que você define usando a instrução **Type**. Os tipos de dados definidos pelo usuário podem conter um ou mais elementos de um tipo de dados, uma matriz ou um tipo previamente definido pelo usuário. Por exemplo:

```
Type MeuTipo
    MeuNome As String ' A variável de seqüência de caracteres armazena um
nome.
    MinhaDataNasc As Date ' Variável de data armazena a data do
nascimento.
    MeuSexo As Integer ' A variável de inteiro armazena o sexo (0 para
End Type ' feminino, 1 para masculino).
```

## Tipo de dados Variant

O tipo de dados **Variant** é o tipo de dados para todas as variáveis que não foram explicitamente declaradas como algum outro tipo (usando instruções como **Dim**, **Private**, **Public** ou **Static**). O tipo de dados **Variant** não possui caractere de declaração de tipo.

Um **Variant** é um tipo de dados especial que contém qualquer tipo de dados com exceção de dados **String** de comprimento fixo e os tipos definidos pelo usuário. O **Variant** também pode conter valores especiais **Empty**, **Error**, **Nothing** e **Null**. Você pode determinar como os dados em uma **Variant** são tratados usando as funções **VarType** ou **TypeName**.

Os dados numéricos podem ter qualquer valor de número inteiro ou real no intervalo de -1,797693134862315E308 a -4,94066E-324 para valores negativos e de 4,94066E-324 a 1,797693134862315E308 para valores positivos. Geralmente os dados numéricos **Variant** são mantidos em seu tipo de dados original dentro de **Variant**. Por exemplo, se você atribuir um **Integer** a uma **Variant**, as operações subsequentes tratarão a **Variant** como um **Integer**. Entretanto, se uma operação aritmética for realizada em uma **Variant** contendo um **Byte**, um **Integer**, um **Long** ou um **Single**, e o resultado exceder o intervalo normal para o tipo de dados original, o resultado será promovido dentro da **Variant** para o próximo tipo de dados maior. Um **Byte** é promovido a **Integer**, um **Integer** é promovido a **Long** e um **Long** e um **Single** são promovidos a **Double**. Ocorrerá um erro quando as variáveis de **Variant** contendo os valores **Currency**, **Decimal** e **Double** excederem seus respectivos intervalos.

Você pode usar o tipo de dados **Variant** no lugar de qualquer tipo de dados para trabalhar com dados de uma forma mais flexível. Se o conteúdo de uma variável **Variant** for dígitos, eles podem ser a representação em seqüência de caracteres dos dígitos ou seu valor real, dependendo do contexto. Por exemplo:

```
Dim MinhaVar As Variant
```

```
MinhaVar = 98052
```

No exemplo anterior, `MinhaVar` contém uma representação numérica—o valor real 98052. Os operadores aritméticos funcionam como esperado em variáveis **Variant** que contêm valores numéricos ou dados de seqüência de caracteres que podem ser interpretados como números. Se você usar o operador `+` para adicionar `MinhaVar` a uma outra **Variant** contendo um número ou a uma variável de um tipo numérico, o resultado será uma soma aritmética.

O valor **Empty** denota uma variável **Variant** que não foi inicializada (atribuído um valor inicial). Uma **Variant** contendo **Empty** será 0 se for usada em um contexto numérico, e uma seqüência de caracteres de comprimento zero ("" ) se for usada em um contexto de seqüência de caracteres.

Não confunda **Empty** com **Null**. **Null** indica que a variável **Variant** propositadamente não contém dados válidos.

Em uma **Variant**, **Error** representa um valor especial usado para indicar que ocorreu uma condição de erro em um procedimento. Entretanto, ao contrário de outros tipos de erro, não ocorre a manipulação normal de erros no nível do aplicativo. Isso permite que você ou o próprio aplicativo, realize alguma ação alternativa com base no valor do erro. Os valores de **Error** são criados pela conversão de números reais em valores de erro usando a função **CVErr**.

### Exemplo da função Date

Este exemplo utiliza a função **Date** para retornar a data atual do sistema.

```
Dim MinhaData
MinhaData = Date ' MinhaData contém a data atual do sistema.
```

### Exemplo da instrução Date

Este exemplo utiliza a instrução **Date** para definir a data do sistema do computador. No ambiente de desenvolvimento, a literal de data é exibida em formato de data abreviada utilizando as definições de localidade do seu código.

```
Dim MinhaData
MinhaData = #Fevereiro 12, 1985# ' Atribui uma data.
Date = MinhaData ' Altera a data do sistema.
```

### Exemplo da função DateAdd

Este exemplo toma uma data e, utilizando a função **DateAdd**, exibe uma data correspondente em um número especificado de meses no futuro.

```
Dim PrimeiraData As Date ' Declara variáveis.
Dim TipoDoIntervalo As String
Dim Número As Integer
Dim Msg
TipoDoIntervalo = "m" ' "m" especifica meses como intervalo.
PrimeiraData = InputBox("Insira uma data")
Número = InputBox("Insira o número de meses a adicionar")
Msg = "Nova data: " & DateAdd(TipoDoIntervalo, Número, PrimeiraData)
MsgBox Msg
```

### Exemplo da função DateDiff

Este exemplo utiliza a função **DateDiff** para exibir o número de dias entre uma determinada data e hoje.

```
Dim AData As Date ' Declara as variáveis.
Dim Msg
AData = InputBox("Insira uma data")
Msg = "Dias a contar de hoje: " & DateDiff("d", Agora, AData)
MsgBox Msg
```

### Exemplo da função DatePart

Este exemplo toma uma data e, utilizando a função **DatePart**, exibe o trimestre do ano no qual ela ocorre.

```
Dim AData As Date ' Declara as variáveis.
Dim Msg
AData = InputBox("Insira uma data:")
Msg = "Trimestre: " & DatePart("q", AData)
MsgBox Msg
```

### Exemplo da função DateSerial

Este exemplo utiliza a função **DateSerial** para retornar a data do ano, mês e dia especificados.

```
Dim MinhaData
' MinhaData contém a data de 12 de fevereiro de 1969.
MinhaData = DateSerial(1969, 2, 12) ' Retorna uma data.
```

### Exemplo da função DateValue

Este exemplo utiliza a função **DateValue** para converter uma seqüência de caracteres em uma data. Você pode também utilizar literais de data para atribuir diretamente uma data a uma variável **Variant** ou **Date**, por exemplo, MinhaData = #2/12/69#).

```
Dim MinhaData
MinhaData = DateValue("12 Fevereiro 1969") ' Retorna uma data.
```

### Exemplo da função Day

Este exemplo utiliza a função **Day** para obter o dia do mês de uma data especificada. No ambiente de desenvolvimento, a literal de data é exibida em formato abreviado utilizando as definições de localidade do seu código.

```
Dim MinhaData, MeuDia
MinhaData = #Fevereiro 12, 1969# ' Atribui uma data.
MeuDia = Day(MinhaData) ' MeuDia contém 12.
```

### Exemplo da função Hour

Este exemplo utiliza a função **Hour** para obter a hora de um horário especificado. No ambiente de desenvolvimento, a literal de hora é exibida em formato de hora abreviada utilizando as definições de localidade do seu código.

```
Dim MeuHorário, MinhaHora
MeuHorário = #4:35:17 PM# ' Atribui um horário.
MinhaHora = Hour(MeuHorário) ' MinhaHora contém 16.
```

### Exemplo da função Minute

Este exemplo utiliza a função **Minute** para obter o minuto da hora de um horário especificado. No ambiente de desenvolvimento, a literal de hora é exibida em formato de hora abreviada utilizando as definições de localidade do seu código.

```
Dim MeuHorário, MeuMinuto
MeuHorário = #4:35:17 PM# ' Atribui uma hora.
MeuMinuto = Minute(MeuHorário) ' MeuMinuto contém 35.
```

### Exemplo da função Month

Este exemplo utiliza a função **Month** para obter o mês de uma data especificada. No ambiente de desenvolvimento, a literal de data é exibida em formato de data abreviada utilizando as definições de localidade do seu código.

```
Dim MinhaData, MeuMês
MinhaData = #Fevereiro 12, 1969# ' Atribui uma data.
MeuMês = Month(MinhaData) ' MeuMês contém 2.
```

### Exemplo da função Now

Este exemplo utiliza a função **Now** para retornar a data e a hora do sistema atual.

```
Dim Hoje
Hoje = Now ' Atribui a data e a hora atuais do sistema.
```

### Exemplo da função Second

Este exemplo utiliza a função **Second** para obter o segundo do minuto de um horário especificado. No ambiente de desenvolvimento, a literal de hora é exibida em formato de hora abreviada utilizando as definições de localidade do seu código.

```
Dim MeuHorário, MeuSegundo
MeuHorário = #4:35:17 PM# ' Atribui uma hora.
MeuSegundo = Second(MeuHorário) ' MeuSegundo contém 17.
```

### Exemplo da função Time

Este exemplo utiliza a função **Time** para retornar o horário atual do sistema.

```
Dim MeuHorário
MeuHorário = Time ' Retorna o horário atual do sistema.
```

### Exemplo da instrução Time

Este exemplo utiliza a instrução **Time** para definir o horário do sistema do computador como um horário definido pelo usuário.

```
Dim MeuHorário
MeuHorário = #4:35:17 PM# ' Atribui um horário.
Time = MeuHorário ' Define o horário do sistema como MeuHorário.
```

### Exemplo da função Timer

Este exemplo utiliza a função **Timer** para pausar o aplicativo. O exemplo também utiliza **DoEvents** para submeter-se a outros processos durante a pausa.

```
Dim TempoDePausa, Início, Fim, TempoTotal
If (MsgBox("Pressione Sim para pausar por 5 segundos", 4)) = vbYes Then
    TempoDePausa = 5 ' Define a duração.
    Início = Timer ' Define a hora inicial.
    Do While Timer < Início + TempoDePausa
        DoEvents ' Submete-se a outros processos.
    Loop
    Fim = Timer ' Define a hora final.
    TempoTotal = Fim - Início ' Calcula o tempo total.
    MsgBox "Pausou por " & TempoTotal & " segundos"
Else
    End
End If
```

### Exemplo da função TimeSerial

Este exemplo utiliza a função **TimeSerial** para retornar um horário para a hora, minuto e segundo especificados.

```
Dim MeuHorário
MeuHorário = TimeSerial(16, 35, 17) ' MeuHorário contém a representação
    ' serial de 4:35:17 PM.
```

### Exemplo da função TimeValue

Este exemplo utiliza a função **TimeValue** para converter uma seqüência de caracteres em um horário. Você pode também utilizar literais de data para atribuir diretamente uma hora a uma variável **Variant** ou **Date**, por exemplo, MeuHorário = #4:35:17 PM#.

```
Dim MeuHorário
MeuHorário = TimeValue("4:35:17 PM") ' Retorna uma hora.
```

### Exemplo da função Weekday

Este exemplo utiliza a função **Weekday** para obter o dia da semana de uma data especificada.

```
Dim MinhaData, MeuDiaDaSemana
MinhaData = #Fevereiro 12, 1969# ' Atribui uma data.
MeuDiaDaSemana = Weekday(MinhaData) ' MeuDiaDaSemana contém 4 porque
    ' MinhaData representa uma quarta-feira.
```

### Exemplo da função Year

Este exemplo utiliza a função **Year** para obter o ano de uma data especificada. No ambiente de desenvolvimento, a literal de data é exibida em formato de data abreviada utilizando as definições de localidade do seu código.

```
Dim MinhaData, MeuAno
MinhaData = #Fevereiro 12, 1969# ' Atribui uma data.
MeuAno = Year (MinhaData) ' MeuAno contém 1969.
```

## Função Date

Retorna um **Variant (Date)** contendo a data atual do sistema.

### Sintaxe

**Date**

### Comentários

Para definir a data do sistema, utilize a instrução **Date**.

## Instrução Date

Define a data atual do sistema.

### Sintaxe

**Date = data**

Para sistemas com o Microsoft Windows 95, a especificação *data* exigida deve ser uma data de 1º de janeiro de 1980 a 31 de dezembro de 2099. Para sistemas que executam o Microsoft Windows NT, *data* deve ser uma data de 1º de janeiro de 1980 a 31 de dezembro de 2079.

## Função DateAdd

Retorna um **Variant (Date)** contendo uma data à qual foi adicionado um segmento de tempo específico.

### Sintaxe

**DateAdd(interval, number, date)**

A sintaxe da função **DateAdd** possui estes argumentos nomeados:

Parte	Descrição
<b>interval</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que é o segmento de tempo que você deseja adicionar.
<b>number</b>	Obrigatório. <u>Expressão numérica</u> que é o número de segmentos que você deseja adicionar. Pode ser positivo (para obter datas futuras) ou negativo (para obter datas passadas).
<b>date</b>	Obrigatório. <b>Variant (Date)</b> ou literal que representa a data na qual o segmento é adicionado.

### Definições

O argumento **interval** possui estas definições:

Definição	Descrição
o	
yyyy	Ano

q	Trimestre
m	Mês
y	Dia do ano
d	Dia
w	Dia da semana
ww	Semana
h	Hora
n	Minuto
s	Segundo

### Comentários

Você pode utilizar a função **DateAdd** para adicionar ou subtrair um segmento de tempo específico de uma data. Por exemplo, você pode utilizar **DateAdd** para calcular uma data a 30 dias de hoje ou uma hora a 45 minutos de agora.

Para adicionar dias a **date**, você pode utilizar Dia do ano ("y"), Dia ("d"), ou Dia da semana ("w").

A função **DateAdd** não retorna uma data inválida. O exemplo a seguir adiciona um mês a 31 de janeiro:

```
DateAdd("m", 1, "31-Jan-95")
```

Neste caso, **DateAdd** retorna 28-fev-95 e não 31-fev-95. Se **date** fosse 31-jan-95, retornaria 29-fev-96 porque 1996 é um ano bissexto.

Se a data calculada fosse anterior ao ano 100 (isto é, você subtrairia mais anos do que estão em **date**), ocorreria um erro.

Se **number** não for um valor **Long**, será arredondado para o número inteiro mais próximo antes de ser avaliado.

## Função DateDiff

Retorna um **Variant (Long)** que especifica o número de segmentos de tempo entre duas datas especificadas.

### Sintaxe

**DateDiff(interval, date1, date2[, firstdayofweek[, firstweekofyear]])**

A sintaxe da função **DateDiff** possui estes argumentos nomeados:

Parte	Descrição
<b>interval</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que é o segmento que você utiliza para calcular a diferença entre <b>date1</b> e <b>date2</b> .
<b>date1, date2</b>	Obrigatório; <b>Variant (Date)</b> . Duas datas que você deseja utilizar no cálculo.
<b>firstdayofweek</b>	Opcional. Uma <u>constante</u> que especifica o primeiro dia da semana. Se não for especificada, é assumido o domingo.
<b>firstweekofyear</b>	Opcional. Uma constante que especifica a primeira semana do ano. Se não for especificada, é assumida como a semana em que ocorre o dia 1 de janeiro.

### Definições

O argumento **interval** possui estas definições:

Definição	Descrição
-----------	-----------

yyyy	Ano
q	Trimestre
m	Mês
y	Dia do ano
d	Dia
w	Dia da semana
ww	Semana
h	Hora
n	Minuto
s	Segundo

O argumento **firstdayofweek** possui estas definições:

Constante	Valor	Descrição
<b>vbUseSystem</b>	0	Utiliza a definição NLS API.
<b>VbSunday</b>	1	Domingo (padrão)
<b>vbMonday</b>	2	Segunda-feira
<b>vbTuesday</b>	3	Terça-feira
<b>vbWednesday</b>	4	Quarta-feira
<b>vbThursday</b>	5	Quinta-feira
<b>vbFriday</b>	6	Sexta-feira
<b>vbSaturday</b>	7	Sábado

O argumento **firstweekofyear** possui estas definições:

Constante	Valor	Descrição
<b>vbUseSystem</b>	0	Utiliza a definição NLS API.
<b>vbFirstJan1</b>	1	Inicia com a semana na qual ocorre o dia 1 de janeiro (padrão).
<b>vbFirstFourDays</b>	2	Inicia com a primeira semana que tem pelo menos quatro dias no ano novo.
<b>vbFirstFullWeek</b>	3	Inicia com a primeira semana cheia do ano.

### Comentários

Você pode utilizar a função **DateDiff** para determinar quantos segmentos de tempo especificados existem entre duas datas. Por exemplo, você pode utilizar **DateDiff** para calcular o número de dias entre duas datas, ou o número de semanas entre hoje e o final do ano.

Para calcular o número de dias entre **date1** e **date2**, você pode utilizar Dia do ano ("y") ou Dia ("d"). Quando **interval** for um Dia da semana ("w"), **DateDiff** retorna o número de semanas entre as duas datas. Se **date1** cair em uma segunda-feira, **DateDiff** contará o número de segundas-feiras até **date2**. É contada **date2** mas não **date1**. Entretanto, se **interval** for Semana ("ww") a função **DateDiff** retorna o número de semanas de calendário entre as duas datas. Ela conta o número de segundas-feiras entre **date1** e **date2**. **DateDiff** conta **date2** se cair em um domingo, mas não conta **date1**, mesmo que caia em um domingo.

Se **date1** se referir a um ponto no tempo além de **date2**, a função **DateDiff** retornará um número negativo.

O argumento **firstdayofweek** afeta os cálculos que utilizam os símbolos de intervalo "w" e "ww".

Se **data1** ou **data2** for uma literal de data, o ano especificado se torna uma parte permanente daquela data. Entretanto, se **data1** ou **data2** for colocada entre aspas duplas (" ") e você omitir o ano, o ano atual será inserido no seu código cada vez que a expressão **data1** ou **data2** for avaliada. Isto torna possível escrever o código que pode ser utilizado em anos diferentes.

Quando compara 31 de dezembro com 1º de janeiro do ano imediatamente seguinte, **DateDiff** para Ano ("yyyy") retorna 1, mesmo que tenha se passado apenas um dia.

## Função DatePart

Retorna um **Variant (Integer)** que contém a parte especificada de uma data dada.

### Sintaxe

**DatePart(interval, date[, firstdayofweek[, firstweekofyear]])**

A sintaxe da função **DatePart** possui estes argumentos nomeados:

Parte	Descrição
<b>interval</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que é o intervalo de tempo que você deseja retornar.
<b>date</b>	Obrigatório. Valor de <b>Variant (Date)</b> que você deseja avaliar.
<b>firstdayofweek</b>	Opcional. Uma <u>constante</u> que especifica o primeiro dia da semana. Se não for especificada, é assumido o domingo.
<b>firstweekofyear</b>	Opcional. Uma constante que especifica a primeira semana do ano. Se não for especificada, é assumida como aquela em que ocorre o dia 1º de janeiro.

### Definições

O argumento interval possui estas definições:

Definição	Descrição
o	
yyyy	Ano
q	Trimestre
m	Mês
y	Dia do ano
d	Dia
w	Dia da semana
ww	Semana
h	Hora
n	Minuto
s	Segundo

O argumento **firstdayofweek** possui estas definições:

Constante	Valor	Descrição
<b>vbUseSystem</b>	0	Utiliza a definição NLS API.
<b>vbSunday</b>	1	Domingo (padrão)
<b>vbMonday</b>	2	Segunda-feira
<b>vbTuesday</b>	3	Terça-feira
<b>vbWednesday</b>	4	Quarta-feira
<b>vbThursday</b>	5	Quinta-feira
<b>vbFriday</b>	6	Sexta-feira
<b>vbSaturday</b>	7	Sábado

O argumento **firstweekofyear** possui estas definições:

Constante	Valor	Descrição
<b>vbUseSystem</b>	0	Utiliza a definição NLS API.
<b>vbFirstJan1</b>	1	Inicia com a semana na qual ocorre o dia 1 de janeiro (padrão).
<b>vbFirstFourDays</b>	2	Inicia com a semana que tem pelo menos quatro dias no ano novo.

### Comentários

Você pode utilizar a função **DatePart** para avaliar uma data e retornar um segmento de tempo específico. Por exemplo, você poderia utilizar **DataPart** para calcular o dia da semana ou a hora atual.

O argumento **firstdayofweek** afeta cálculos que utilizam os símbolos de intervalo "w" e "ww".

Se *data* for uma literal de data, o ano especificado torna-se uma parte permanente dessa data. Entretanto, se *data* estiver entre aspas duplas (" ") e você omitir o ano, o ano atual será inserido no seu código toda vez que a expressão *data* for avaliada. Isto torna possível escrever o código que pode ser utilizado em anos diferentes.

## Função DateSerial

Retorna um **Variant (Date)** para um ano, mês e dia específicos.

### Sintaxe

#### **DateSerial(year, month, day)**

A sintaxe da função **DateSerial** possui estes argumentos nomeados:

Parte	Descrição
<b>year</b>	Obrigatório; <b>Integer</b> . Número entre 100 e 9999, inclusive, ou uma <u>expressão numérica</u> .
<b>month</b>	Obrigatório; <b>Integer</b> . Qualquer expressão numérica.
<b>day</b>	Obrigatório; <b>Integer</b> . Qualquer expressão numérica.

### Comentários

Para especificar uma data, como 31 de dezembro de 1991, o intervalo de números para cada argumento de **DateSerial** deve estar no intervalo aceito para a unidade, isto é, 1–31 para dias e 1–12 para meses. Entretanto, você pode também especificar datas relativas para cada argumento utilizando qualquer expressão numérica que represente algum número de dias, meses ou anos, antes ou depois de uma certa data.

O exemplo a seguir utiliza expressões numéricas em vez de números absolutos de data. Aqui a função **DateSerial** retorna uma data que é o dia anterior ao primeiro dia (1 - 1), dois meses antes de agosto (8 - 2), 10 anos antes de 1990 (1990 - 10); em outras palavras, 31 de maio de 1980.

```
DateSerial(1990 - 10, 8 - 2, 1 - 1)
```

Para o argumento **year**, os valores entre 0 e 99, inclusive, são interpretados como os anos 1900–1999. Para todos os outros argumentos **year**, utilize o ano de quatro dígitos (por exemplo, 1800).

Quando qualquer argumento exceder o intervalo aceito para esse argumento, ele é incrementado até a próxima unidade maior apropriada. Por exemplo, se você especificar 35 dias, será avaliado como um mês e alguns dias, dependendo em que época do ano seja aplicado. Se qualquer argumento único estiver fora do intervalo -32.768 a 32.767, ocorrerá um erro. Se a data especificada pelos três argumentos ficar fora do intervalo de datas aceitável, ocorrerá um erro.

## Função DateValue

Retorna um **Variant (Date)**.

### Sintaxe

#### **DateValue**(*data*)

O argumento *data* requerido é normalmente uma expressão de seqüência de caracteres que representa uma data de 1º de janeiro de 100 a 31 de dezembro de 9999. Entretanto, *data* também pode ser qualquer expressão que possa representar uma data, uma hora ou ambos, naquele intervalo.

### Comentários

Se *data* for uma seqüência de caracteres que inclui somente números separados por separadores de data válidos, **DateValue** reconhecerá a ordem para mês, dia e ano de acordo com o formato Short Date que você especificou para o seu sistema. **DateValue** também reconhece datas não ambíguas que contenham nomes de meses, na forma longa ou abreviada. Por exemplo, além de reconhecer 30/12/1991 e 30/12/91, **DateValue** reconhece também 30 de dezembro de 1991 e 30-dez-1991.

Se a parte de ano de *data* for omitida, **DateValue** utiliza o ano atual da data do seu sistema de computador.

Se o argumento *data* incluir informações de hora, **DateValue** não a retorna. Entretanto, se *data* incluir informação de hora inválida (como "89:98"), ocorrerá um erro.

## Função Day

Retorna um **Variant (Integer)** que especifica um número inteiro entre 1 e 31, inclusive, representando o dia do mês.

### Sintaxe

#### **Day**(*data*)

O argumento *data* requerido é qualquer **Variant**, expressão numérica, expressão de seqüência de caracteres, ou qualquer combinação que possa representar uma data. Se *data* contiver **Null**, será retornado **Null**.

## Função Hour

Retorna um **Variant (Integer)** que especifica um número inteiro entre 0 e 23, inclusive, representando a hora do dia.

### Sintaxe

#### **Hour**(*hora*)

O argumento *hora* requerido é qualquer **Variant**, expressão numérica, expressão de seqüência de caracteres, ou qualquer combinação que possa representar a hora. Se *hora* contiver **Null**, será retornado **Null**.

## Função Minute

Retorna um **Variant (Integer)** que especifica um número inteiro entre 0 e 59, inclusive, representando os minutos da hora.

### Sintaxe

#### Minute(*hora*)

O argumento *hora* requerido é qualquer **Variant**, expressão numérica, expressão de seqüência de caracteres ou qualquer combinação que possa representar a hora. Se *hora* contiver **Null**, será retornado **Null**.

## Função Month

Retorna um **Variant (Integer)** que especifica um número inteiro entre 1 e 12, inclusive, representando o mês do ano.

### Sintaxe

#### Month(*data*)

O argumento *data* requerido é qualquer **Variant**, expressão numérica, expressão de seqüência de caracteres, ou qualquer combinação que possa representar uma data. Se *data* contiver **Null**, será retornado **Null**.

## Função Now

Retorna um **Variant (Date)** que especifica a data e hora atual de acordo com a data e hora do sistema do seu computador.

### Sintaxe

#### Now

## Função Second

Retorna um **Variant (Integer)** que especifica um número inteiro entre 0 e 59, inclusive, representando os segundos do minuto.

### Sintaxe

#### Second(*hora*)

O argumento *hora* requerido é qualquer **Variant**, expressão numérica, expressão de seqüência de caracteres, ou qualquer combinação que possa representar a hora. Se *hora* contiver **Null**, será retornado **Null**.

## Função Time

Retorna um **Variant (Date)** que indica a hora atual do sistema.

### Sintaxe

#### Time

#### Comentários

Para definir a hora do sistema utilize a instrução **Time**.

## Instrução Time

Define a hora do sistema.

### Sintaxe

**Time** = *hora*

O argumento *hora* requerido é qualquer expressão numérica, expressão de seqüência de caracteres, ou qualquer combinação que possa representar uma hora.

### Comentários

Se *hora* for uma seqüência de caracteres, **Time** tentará convertê-la para horas utilizando os separadores de hora que você especificou para o seu sistema. Se não puder ser convertida para uma hora válida, ocorrerá um erro.

## Função Timer

Retorna um **Single** que representa o número de segundos decorridos desde a meia-noite.

### Sintaxe

**Timer**

## Função TimeSerial

Retorna um **VARIANT (Date)** contendo uma indicação de hora para a hora, minuto e segundo específicos.

### Sintaxe

**TimeSerial(hour, minute, second)**

A sintaxe da função **TimeSerial** possui estes argumentos nomeados:

Parte	Descrição
<i>hour</i>	Obrigatório; <b>VARIANT (Integer)</b> . Número entre 0 (0:00 h) e 23 (23:00 h), inclusive, ou uma <u>expressão numérica</u> .
<i>minute</i>	Obrigatório; <b>VARIANT (Integer)</b> . Qualquer expressão numérica.
<i>second</i>	Obrigatório; <b>VARIANT (Integer)</b> . Qualquer expressão numérica.

### Comentários

Para especificar uma hora, como 11:59:59, o intervalo de números para cada argumento de **TimeSerial** deve estar no intervalo normal da unidade, isto é, 0–23 para horas e 0–59 para minutos e segundos. Entretanto, você pode também especificar horas relativas para cada argumento utilizando qualquer expressão numérica que represente alguma quantidade de horas, minutos ou segundos antes e depois de uma determinada hora. O exemplo a seguir utiliza expressões em vez de números de horas absolutos. A função **TimeSerial** retorna a hora de 15 minutos antes (-15) de seis horas antes do meio-dia (12 - 6), ou 5:45:00 h.

```
TimeSerial(12 - 6, -15, 0)
```

Quando qualquer argumento excede seu intervalo normal, ele é incrementado até a próxima unidade maior apropriada. Por exemplo, se você especificar 75 minutos, isto será avaliado como uma hora e 15 minutos. Se qualquer argumento único estiver fora do intervalo -32.768 a 32.768, ocorrerá um erro. Se a hora especificada pelos três argumentos fizer com que a data fique fora do intervalo de datas aceitável, ocorrerá um erro.

## Função TimeValue

Retorna um **Variant (Date)** contendo a hora.

### Sintaxe

#### TimeValue(*hora*)

O argumento *hora* requerido normalmente é uma expressão de seqüência de caracteres representando uma hora entre 0:00:00 (0:00:00 h) e 23:59:59 (23:59:59 h), inclusive. Entretanto, *hora* pode ser também qualquer expressão que represente uma hora naquele intervalo. Se *hora* contiver **Null**, será retornado **Null**.

### Comentários

Você pode inserir horas válidas utilizando um relógio de 12 ou de 24 horas. Por exemplo, tanto "2:24PM" como "14:24" são argumentos de *hora* válidos.

Se o argumento *hora* contiver informação de data, **TimeValue** não o retorna. Entretanto, se *hora* incluir informação de data inválida, ocorrerá um erro.

## Função Weekday

Retorna um **Variant (Integer)** contendo um número inteiro que representa o dia da semana.

### Sintaxe

#### Weekday(*date*, [*firstdayofweek*])

A sintaxe da função **Weekday** possui estes argumentos nomeados:

Parte	Descrição
<i>date</i>	Obrigatório. <b>Variant</b> , <u>expressão numérica</u> , <u>expressão de seqüência de caracteres</u> , ou qualquer combinação que possa representar uma data. Se <b>date</b> contiver <b>Null</b> , será retornado <b>Null</b> .
<i>firstdayofweek</i>	Opcional. Uma <u>constante</u> que especifica o primeiro dia da semana. Se não for especificada, será assumida <b>vbSunday</b> .

### Definições

O argumento *firstdayofweek* possui estas definições:

Constante	Valor	Descrição
<b>vbUseSystem</b>	0	Utiliza a definição NLS API.
<b>vbSunday</b>	1	Domingo (padrão)
<b>vbMonday</b>	2	Segunda-feira
<b>vbTuesday</b>	3	Terça-feira
<b>vbWednesday</b>	4	Quarta-feira
<b>vbThursday</b>	5	Quinta-feira
<b>vbFriday</b>	6	Sexta-feira
<b>vbSaturday</b>	7	Sábado

### Valores de Retorno

A função **Weekday** pode retornar qualquer um destes valores:

Constante	Valor	Descrição
<b>vbSunday</b>	1	Domingo
<b>vbMonday</b>	2	Segunda-feira
<b>vbTuesday</b>	3	Terça-feira

---

<b>vbWednesday</b>	4	Quarta-feira
<b>vbThursday</b>	5	Quinta-feira
<b>vbFriday</b>	6	Sexta-feira
<b>vbSaturday</b>	7	Sábado

## Função Year

Retorna um **Variant (Integer)** contendo um número inteiro que representa o ano.

### Sintaxe

#### **Year**(*data*)

O argumento *data* requerido é qualquer **Variant**, expressão numérica, expressão de seqüência de caracteres, ou qualquer combinação que possa representar uma data. Se *data* contiver **Null**, será retornado **Null**.

## Função Array

Retorna uma **Variant** que contém uma matriz.

### Sintaxe

#### **Array**(*arglist*)

O argumento obrigatório *arglist* é uma lista de valores delimitados por vírgulas que são atribuídos aos elementos da matriz contidos dentro de **Variant**. Se nenhum argumento for especificado, uma matriz de tamanho zero será criada.

### Comentários

A notação utilizada para referir-se a um elemento de uma matriz consiste no nome da variável seguido por um número de índice entre parênteses indicando o elemento desejado. No exemplo abaixo, a primeira instrução cria uma variável denominada **A** como um **Variant**. A segunda instrução atribui uma matriz à variável **A**. A última instrução atribui o valor contido no segundo elemento de matriz a outra variável.

```
Dim A As Variant
A = Array(10, 20, 30)
B = A(2)
```

O limite inferior de uma matriz criada com a função **Array** será sempre zero. Ao contrário de outros tipos de matrizes, ela não é afetada pelo limite inferior especificado pela instrução **Option Base**.

**Observação:** Uma **Variant** que não é declarada como matriz pode, ainda assim, conter uma matriz. Uma variável **Variant** pode conter uma matriz de qualquer tipo, com exceção de seqüências de caracteres de comprimento zero e tipos definidos pelo usuário. Embora um **Variant** que contém uma matriz seja conceitualmente diferente de uma matriz cujos elementos sejam de tipo **Variant**, os elementos da matriz são acessados da mesma forma.

## Instrução Const

Declara as constantes a serem utilizadas no lugar de valores literais.

### Sintaxe

[**Public** | **Private**] **Const** *constname*[**As** *type*] = *expression*

A sintaxe da instrução **Const** tem estas partes:

Parte	Descrição
<b>Public</b>	Opcional. <u>Palavra-chave</u> utilizada no <u>nível de módulo</u> para declarar constantes que estão disponíveis para todos os <u>procedimentos</u> em todos os <u>módulos</u> . Não é permitida em procedimentos.
<b>Private</b>	Opcional. Palavra-chave utilizada no nível de módulo para declarar constantes disponíveis apenas dentro do módulo no qual a <u>declaração</u> é feita. Não é permitida em procedimentos.
<i>constname</i>	Obrigatório. Nome da constante; segue a <u>variável padrão</u> que dá nome às convenções.
<i>type</i>	Opcional. <u>Tipo de dados</u> da constante; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (atualmente não é suportado), <b>Date</b> , <b>String</b> ou <b>Variant</b> . Utilize uma cláusula <i>type</i> <b>As</b> separada para cada constante que for declarada.
<i>expression</i>	Obrigatório. Um literal, outra constante ou qualquer combinação que inclua todos os operadores aritméticos ou lógicos com exceção de <b>Is</b> .

### Comentários

As constantes são privadas como padrão. Dentro de procedimentos, as constantes sempre serão particulares; sua visibilidade não pode ser alterada. Em módulos padrão, a visibilidade padrão de constantes no nível de módulo pode ser alterada através da palavra-chave **Public**. Nos módulos de classe, contudo, as constantes só podem ser privadas e sua visibilidade não pode ser alterada através da palavra-chave **Public**.

Para combinar diversas declarações de constantes na mesma linha, separe cada atribuição de constante com uma vírgula. Quando as declarações de constante são combinadas dessa forma, a palavra-chave **Public** ou **Private**, se utilizada, aplica-se a todas elas.

Você não pode utilizar variáveis, funções definidas pelo usuário ou funções intrínsecas ao Visual Basic (como **Chr**) em expressões atribuídas a constantes.

**Observação:** As constantes podem tornar seus programas automaticamente documentáveis e fáceis de serem modificados. Ao contrário das variáveis, as constantes não podem ser alteradas inadvertidamente enquanto seu programa está sendo executado.

Se você não declarar explicitamente o tipo de constante utilizando **As type**, a constante receberá o tipo de dados mais apropriado à *expression*.

As constantes declaradas em um procedimento **Sub**, **Function** ou **Property** são locais a esse procedimento. Uma constante declarada fora de um procedimento é definida em todo o módulo no qual é declarada. Você pode utilizar constantes em qualquer lugar onde for possível utilizar uma expressão.

## Função CreateObject

Cria e retorna uma referência a um objeto ActiveX.

### Sintaxe

#### CreateObject(class)

O argumento class utiliza a sintaxe *apname.objecttype* e tem as partes abaixo:

Parte	Descrição
<i>apname</i>	Obrigatória; <b>Variant (String)</b> . O nome do aplicativo que fornece o objeto.
<i>objecttype</i>	Obrigatória; <b>Variant (String)</b> . O tipo ou <u>classe</u> do objeto a ser criado.

### Comentários

Qualquer aplicativo que suporte Automação fornece pelo menos um tipo de objeto. Por exemplo, um aplicativo de processamento de texto pode fornecer um objeto **Application**, um objeto **Document** e um objeto **Toolbar**.

Para criar um objeto ActiveX, atribua o objeto retornado por **CreateObject** a uma variável de objeto:

```
' Declara uma variável de objeto para conter a referência do objeto
' Dim as Object provoca ligação tardia.
Dim ExcelPlan As Object
Set ExcelPlan = CreateObject("Excel.Plan")
```

Esse código inicia o aplicativo que cria o objeto, nesse caso uma planilha do Microsoft Excel. Após um objeto ter sido criado, ele pode ser referido no código através da variável de objeto que você definiu. No exemplo abaixo, você acessa as propriedades e os métodos do novo objeto utilizando a variável de objeto, ExcelPlan e outros objetos do Microsoft Excel, incluindo o objeto Application e a coleção Cells.

```
' Torna o Excel visível através do objeto Application
ExcelPlan.Application.Visible = True
' Coloca um texto na primeira célula da planilha
ExcelPlan.Cells(1, 1).Value = "Esta é a coluna A, fileira 1"
' Salva a planilha no diretório C:\test.doc
ExcelPlan.SaveAs "C:\ TEST.DOC"
' Fecha o Excel com o método Quit no objeto Application
ExcelPlan.Application.Quit
' Libera a variável de objeto
Set ExcelPlan = Nothing
```

Declarar uma variável de objeto com a cláusula *As Object* cria uma variável que pode conter uma referência a qualquer tipo de objeto. No entanto, o acesso ao objeto através dessa variável dá-se com uma ligação tardia, ou seja, a ligação ocorre quando seu programa já está sendo executado. Para criar uma variável de objeto que resulte em ligação inicial, ou seja, quando o programa é compilado, declare a variável do objeto com um código de classe específico. Por exemplo, você pode declarar e criar as referências abaixo do Microsoft Excel:

```
Dim xlApp As Excel.Application
Dim xlBook As Excel.Workbook
Dim xlSheet As Excel.WorkSheet
Set xlApp = CreateObject("Excel.Application")
Set xlBook = xlApp.Workbooks.Add
Set xlSheet = xlBook.Worksheets(1)
```

A referência através de uma variável de vinculação inicial pode proporcionar melhor desempenho, mas só pode conter uma referência à classe especificada na declaração.

Você pode passar um objeto retornado pela função **CreateObject** para uma função que espere um

objeto como argumento. Por exemplo, o código abaixo cria e passa uma referência a um objeto Excel.Application:

```
Call MinhaSub (CreateObject("Excel.Application"))
```

**Observação:** Utilize **CreateObject** quando não houver nenhuma ocorrência atual do objeto. Se uma ocorrência do objeto já estiver sendo executada, uma nova ocorrência é iniciada e um objeto do tipo especificado é criado. Para utilizar a ocorrência atual ou para iniciar o aplicativo e fazê-lo carregar um arquivo, utilize a função **GetObject**.

Se um objeto tiver registrado a si próprio como um objeto de ocorrência única, apenas uma ocorrência do objeto será criada, independente de quantas vezes **CreateObject** for executado.

## Instrução Declare

Utilizada no nível de módulo para declarar referências a procedimentos externos em uma biblioteca de vínculo dinâmico (DLL).

### Sintaxe 1

[Public | Private] **Declare Sub** *name* **Lib** "*libname*" [**Alias** "*aliasname*"] [(*arglist*)]

### Sintaxe 2

[Public | Private] **Declare Function** *name* **Lib** "*libname*" [**Alias** "*aliasname*"] [(*arglist*)] [**As** *type*]

A sintaxe da instrução **Declare** tem estas partes:

Parte	Descrição
<b>Public</b>	Opcional. Utilizada para declarar procedimentos que estejam disponíveis a todos os outros procedimentos em todos os <u>módulos</u> .
<b>Private</b>	Opcional. Utilizada para declarar procedimentos que estejam disponíveis apenas dentro do módulo no qual a <u>declaração</u> é feita.
<b>Sub</b>	Opcional (tanto <b>Sub</b> quanto <b>Function</b> devem aparecer). Indica que o procedimento não retorna um valor.
<b>Function</b>	Opcional ( <b>Sub</b> ou <b>Function</b> deve aparecer). Indica que o procedimento retorna um valor que pode ser utilizado em uma <u>expressão</u> .
<i>name</i>	Obrigatória. Qualquer nome de procedimento válido. Observe que os pontos de entrada da DLL coincidem maiúsculas e minúsculas.
<b>Lib</b>	Obrigatória. Indica que uma DLL ou recurso de código contém o procedimento que está sendo declarado. A cláusula <b>Lib</b> é exigida para todas as declarações.
<i>Libname</i>	Obrigatória. Nome da DLL ou recurso de código que contém o procedimento declarado.
<b>Alias</b>	Opcional. Indica que o procedimento que está sendo chamado possui outro nome na DLL. Isso é útil quando o nome do procedimento externo é o mesmo de uma palavra-chave. Você também pode utilizar <b>Alias</b> quando um procedimento de DLL tem o mesmo nome de uma <u>variável pública</u> , <u>constante</u> ou qualquer outro procedimento no mesmo <u>escopo</u> . <b>Alias</b> também é útil se qualquer caractere do nome do procedimento da DLL não for permitido pela convenção de nomenclatura da DLL.
<i>Aliasname</i>	Opcional. Nome do procedimento da DLL ou recurso de código. Se o primeiro caractere não for o sinal numérico (#), <i>aliasname</i> será o nome do ponto de entrada do procedimento na DLL. Se (#) for o primeiro caractere, todos os caracteres seguintes deverão indicar o número ordinal do ponto de entrada do procedimento.
<i>Arglist</i>	Opcional. Lista de variáveis que representam <u>argumentos</u>

*Type* que são passados ao procedimento quando ele é chamado. Opcional. Tipo de dados do valor retornado por um procedimento **Function**; pode ser **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (atualmente não-suportado), **Date**, **String** (apenas de comprimento variável) ou **Variant**, um tipo definido pelo usuário ou um tipo de objeto.

O argumento *arglist* apresenta a sintaxe e as partes abaixo:

[Optional] [ByVal | ByRef] [ParamArray] *varname* [( )] [As *type*]

Parte	Descrição
<b>Optional</b>	Opcional. Indica que um argumento não é obrigatório. Se utilizado, todos os argumentos subseqüentes em <i>arglist</i> deverão ser opcionais e declarados através da palavra-chave <b>Optional</b> . <b>Optional</b> não pode ser utilizado para qualquer argumento se <b>ParamArray</b> for utilizado.
<b>ByVal</b>	Opcional. Indica que o argumento é passado <u>por valor</u> .
<b>ByRef</b>	Indica que o argumento é passado <u>por referência</u> . <b>ByRef</b> é o padrão no Visual Basic.
<b>ParamArray</b>	Opcional. Utilizado apenas como o último argumento em <i>arglist</i> para indicar que o argumento final é uma <u>matriz</u> <b>Optional</b> de elementos <b>Variant</b> . A palavra-chave <b>ParamArray</b> permite que você forneça um número arbitrário de argumentos. A palavra-chave <b>ParamArray</b> não pode ser utilizada com <b>ByVal</b> , <b>ByRef</b> ou <b>Optional</b> .
<i>varname</i>	Obrigatória. Nome da variável que representa o argumento passado ao procedimento; segue as convenções de nomenclatura padrão de variáveis.
( )	Obrigatória para variáveis de matriz. Indica que <i>varname</i> é uma matriz.
<i>type</i>	Opcional. Tipo de dados do argumento passado ao procedimento; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (atualmente não-suportado), <b>Date</b> , <b>String</b> (apenas de comprimento variável), <b>Object</b> , <b>Variant</b> , um tipo definido pelo usuário ou um tipo de objeto.

### Comentários

Para procedimentos **Function**, o tipo de dados do procedimento determina o tipo de dados retornado. Você pode utilizar uma cláusula **As** seguindo *arglist* para especificar o tipo de retorno da função. Dentro de *arglist*, você pode utilizar uma cláusula **As** para especificar o tipo de dados de qualquer um dos argumentos passados ao procedimento. Além de especificar qualquer tipo de dados padrão, você pode especificar **As Any** em *arglist* para inibir a verificação de tipo e permitir que qualquer tipo de dados seja passado ao procedimento.

Parênteses vazios indicam que o procedimento **Sub** ou **Function** não possui argumentos e que o Visual Basic deve garantir que nenhum argumento será passado. No exemplo abaixo, *First* não possui argumentos. Se você utilizar argumentos em um chamada para *First*, um erro será gerado:

```
Declare Sub First Lib "MinhaBib" ()
```

Se você incluir uma lista de argumentos, o número e tipo de argumentos são verificados a cada vez que o procedimento for chamado. No exemplo abaixo, *First* aceita um argumento **Long**:

```
Declare Sub First Lib "MinhaLib" (X As Long)
```

**Observação:** A presença de seqüências de caracteres de comprimento fixo na lista de argumentos de uma instrução **Declare** não é permitida; apenas seqüências de caracteres de comprimento variável podem ser passadas a procedimentos. Seqüências de caracteres de comprimento fixo podem aparecer como argumentos de procedimentos, mas são convertidas em seqüências de caracteres de comprimento variável antes de serem passadas.

**Observação:** A constante **vbNullString** é utilizada ao chamar-se procedimentos externos, quando os procedimentos externos exigem uma seqüência de caracteres cujo valor seja zero. Isso é diferente de uma seqüência de caracteres de comprimento zero ("").

## Instruções Deftype

Utilizadas a nível de módulo para definir o tipo de dados padrão para variáveis, argumentos passados a procedimentos e o tipo de retorno para os procedimentos **Function** e **Property Get** cujos nomes se iniciem pelos caracteres especificados.

### Sintaxe

**DefBool** *letterrange*[, *letterrange*] ...  
**DefByte** *letterrange*[, *letterrange*] ...  
**DefInt** *letterrange*[, *letterrange*] ...  
**DefLng** *letterrange*[, *letterrange*] ...  
**DefCur** *letterrange*[, *letterrange*] ...  
**DefSng** *letterrange*[, *letterrange*] ...  
**DefDbl** *letterrange*[, *letterrange*] ...  
**DefDec** *letterrange*[, *letterrange*] ...  
**DefDate** *letterrange*[, *letterrange*] ...  
**DefStr** *letterrange*[, *letterrange*] ...  
**DefObj** *letterrange*[, *letterrange*] ...  
**DefVar** *letterrange*[, *letterrange*] ...

O argumento obrigatório *letterrange* apresenta a sintaxe abaixo:

*letter1*[-*letter2*]

Os argumentos *letter1* e *letter2* especificam o intervalo de nomes para o qual você pode definir um tipo de dados padrão. Cada argumento representa a primeira letra da variável, argumento, procedimento **Function** ou nome de procedimento **Property Get** e pode ser qualquer letra do alfabeto. A caixa das letras em *letterrange* não é relevante.

### Comentários

O nome da instrução determina o tipo de dados:

Instrução	Tipo de dados
<b>DefBool</b>	<b><u>Boolean</u></b>
<b>DefByte</b>	<b><u>Byte</u></b>
<b>DefInt</b>	<b><u>Integer</u></b>
<b>DefLng</b>	<b><u>Long</u></b>
<b>DefCur</b>	<b><u>Currency</u></b>
<b>DefSng</b>	<b><u>Single</u></b>
<b>DefDbl</b>	<b><u>Double</u></b>
<b>DefDec</b>	<b>Decimal</b> (atualmente não-suportado)
<b>DefDate</b>	<b><u>Date</u></b>
<b>DefStr</b>	<b><u>String</u></b>
<b>DefObj</b>	<b><u>Object</u></b>
<b>DefVar</b>	<b><u>Variant</u></b>

Por exemplo, no fragmento de programa abaixo, *Message* é uma variável de seqüência de caracteres:

```
DefStr A-Q
. . .
Message = "Sem espaço em pilha."
```

Uma instrução **Deftype** afeta somente o módulo onde ela é utilizada. Por exemplo, uma instrução **DefInt** em um módulo afeta somente o tipo de dados padrão de variáveis, argumentos passados a procedimentos e o tipo de retorno para procedimentos **Function** e **Property Get** declarados nesse

módulo; o tipo de dados padrão de variáveis, argumentos e os tipos de retorno em outros módulos não são afetados. Caso não sejam explicitamente declarados através de uma instrução **Deftype**, o tipo de dados padrão para todas as variáveis, todos os argumentos, todos os procedimentos **Function** e todos os procedimentos **Property Get** será **Variant**.

Quando você especifica um intervalo de letras, ele geralmente define o tipo de dados para variáveis iniciadas com letras entre os primeiros 128 caracteres do conjunto de caracteres. No entanto, quando você especifica o intervalo de letras A – Z, você define o padrão para o tipo de dados especificado para todas as variáveis, incluindo as iniciadas por caracteres internacionais da parte estendida do conjunto de caracteres (128 – 255).

Depois que o intervalo A – Z tiver sido especificado, você não poderá redefinir qualquer subintervalo de variável utilizando instruções **Deftype**. Uma vez que um intervalo tenha sido especificado, se você incluir uma letra predefinida em uma outra instrução **Deftype**, um erro será gerado. No entanto, você pode especificar explicitamente o tipo de dados de qualquer variável, definida ou não, utilizando uma instrução **Dim** com uma cláusula **As type**. Por exemplo, você pode utilizar o código abaixo no nível do módulo para definir uma variável como um **Double**, embora o tipo de dados padrão seja **Integer**:

```
DefInt A-Z
Dim PercTaxa As Double
```

As instruções **Deftype** não afetam os elementos de tipos definidos pelo usuário porque eles devem ser explicitamente declarados.

## Instrução Dim

Declara variáveis e aloca espaço de armazenamento.

### Sintaxe

**Dim** [**WithEvents**] *varname*[(*subscripts*)] [**As** [**New**] *type*] [, [**WithEvents**] *varname*[(*subscripts*)] [**As** [**New**] *type*]] . . .

A sintaxe da instrução **Dim** tem estas partes:

Parte	Descrição
<b>WithEvents</b>	Opcional. Palavra-chave que especifica que <i>varname</i> é uma <u>variável de objeto</u> utilizada para responder a eventos acionados por um <u>objeto ActiveX</u> . É válida somente em <u>módulos de classe</u> . Você pode declarar quantas variáveis individuais desejar utilizando <b>WithEvents</b> , mas não pode criar <u>matrizes</u> com <b>WithEvents</b> . Você não pode utilizar <b>New</b> com <b>WithEvents</b> .
<i>varname</i>	Obrigatória. Nome da variável; segue as convenções de nomenclatura padrão de variáveis.
<i>subscripts</i>	Opcional. Dimensões de uma variável de matriz; até 60 dimensões múltiplas podem ser declaradas. O argumento <i>subscripts</i> utiliza a sintaxe abaixo: <p>[<i>inferior To</i>] <i>superior</i> [, [<i>inferior To</i>] <i>superior</i>] . . .</p> Quando não enunciado explicitamente em <i>inferior</i> , o limite inferior de uma matriz é controlado pela instrução <b>Option Base</b> . O limite inferior é zero se nenhuma instrução <b>Option Base</b> estiver presente.
<b>New</b>	Opcional. Palavra-chave que permite a criação implícita de um objeto. Se você utilizar <b>New</b> ao declarar a variável de objeto, uma nova ocorrência do objeto será criada na primeira referência a ele. Dessa forma, você não precisa utilizar a instrução <b>Set</b> para atribuir a referência do objeto. A palavra-chave <b>New</b> não pode ser utilizada para declarar variáveis de qualquer <u>tipo de dados</u> intrínseco, não pode ser utilizada para declarar instâncias de objetos dependentes e nem ser utilizada com <b>WithEvents</b> .
<i>type</i>	Opcional. Tipo de dados da variável; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (atualmente não-suportado), <b>Date</b> , <b>String</b> (para seqüências de caracteres de

comprimento variável), **String** \* *comprimento* (para seqüências de caracteres de comprimento fixo), **Object**, **Variant**, um tipo definido pelo usuário ou um tipo de objeto. Utilize uma cláusula **As type** separada para cada variável que você declarar.

### Comentários

As variáveis declaradas com **Dim** no nível de módulo estão disponíveis para todos os procedimentos dentro do módulo. No nível de procedimento, as variáveis estão disponíveis somente dentro do procedimento.

Utilize a instrução **Dim** no nível de módulo ou procedimento para declarar o tipo de dados de uma variável. Por exemplo, a instrução abaixo declara uma variável como **Integer**.

```
Dim NúmeroDeEmpregados As Integer
```

Também é possível utilizar uma instrução **Dim** para declarar o tipo de objeto de uma variável. O exemplo abaixo declara uma variável para uma nova ocorrência de uma planilha.

```
Dim X As New Planilha
```

Se a palavra-chave **New** não for utilizada para declarar uma variável de objeto, um objeto já existente deverá ser atribuído a uma variável referente ao objeto através da instrução **Set** antes que ele possa ser utilizado. Até ter um objeto atribuído a ela, a variável de objeto declarada apresentará o valor especial **Nothing**, que indica que ela não se refere a nenhuma ocorrência específica de um objeto.

Você também pode utilizar a instrução **Dim** com parênteses vazios para declarar uma matriz dinâmica. Após declará-la, utilize a instrução **ReDim** dentro de um procedimento para definir o número de dimensões e elementos na matriz. Se você tentar declarar novamente uma dimensão para uma variável de matriz cujo tamanho tenha sido especificado explicitamente em uma instrução **Private**, **Public** ou **Dim**, um erro será gerado.

Se você não especificar um tipo de dados ou de objeto e se não houver nenhuma instrução **DefType** no módulo, a variável será **Variant** como padrão.

Quando as variáveis são inicializadas, uma variável numérica é inicializada como 0, uma seqüência de caracteres de comprimento variável é inicializada como uma seqüência de caracteres de comprimento zero ("") e uma seqüência de caracteres de comprimento fixo é preenchida com zeros. As variáveis **Variant** são inicializadas como **Empty**. Cada elemento de uma variável de tipo definida pelo usuário é inicializado como se fosse uma variável independente.

**Observação:** Quando você utiliza a instrução **Dim** em um procedimento, ela geralmente é colocada no início do procedimento.

## Instrução Enum

Declara um tipo para uma enumeração.

### Sintaxe

```
[Public | Private] Enum name
    membername [= constantexpression]
    membername [= constantexpression]
    ...
```

### End Enum

A instrução **Enum** tem estas partes:

Parte	Descrição
<b>Public</b>	Opcional. Especifica que o tipo <b>Enum</b> é visível através de todo o <u>projeto</u> . Os tipos <b>Enum</b> são <b>Public</b> como padrão.
<b>Private</b>	Opcional. Especifica que o tipo <b>Enum</b> somente é visível dentro do <u>módulo</u> em que ele aparece.
<i>name</i>	Obrigatório. O nome do tipo <b>Enum</b> . O <i>name</i> deve

	ser um identificador do Visual Basic válido e é especificado como o tipo ao declarar <u>variáveis</u> ou <u>parâmetros</u> do tipo <b>Enum</b> .
<i>membername</i>	Obrigatório. Um identificador do Visual Basic válido especificando o nome pelo qual um elemento componente do tipo <b>Enum</b> será conhecido.
<i>constantexpression</i>	Opcional. O valor do elemento (avaliado como <b>Long</b> ). Pode ser outro tipo <b>Enum</b> . Se nenhuma <i>constantexpression</i> for especificada, o valor atribuído será ou zero (se for o primeiro <i>membername</i> ) ou 1 maior que o valor do <i>membername</i> imediatamente anterior.

### Comentários

As variáveis de enumeração são variáveis declaradas com um tipo **Enum**. Ambas as variáveis e parâmetros podem ser declarados com um tipo **Enum**. Os elementos do tipo **Enum** são inicializados como valores constantes dentro da instrução **Enum**. Os valores atribuídos não podem ser modificados em tempo de execução e podem incluir números negativos ou positivos. Por exemplo:

```
Enum SecurityLevel
    IllegalEntry = -1
    SecurityLevel1 = 0
    SecurityLevel2 = 1
End Enum
```

Uma instrução **Enum** somente pode aparecer em nível de módulo. Uma vez que o tipo **Enum** é definido, ele pode ser usado para declarar variáveis, parâmetros ou procedimentos retornando seu tipo. Você não pode qualificar um nome de tipo **Enum** com um nome de módulo. Tipos **Public Enum** em um módulo de classe não são membros da classe; entretanto, eles são gravados na biblioteca de tipos. Tipos **Enum** definidos em módulos padrão não são gravados em bibliotecas de tipos. Os tipos **Public Enum** de mesmo nome não podem ser definidos em ambos os módulos padrão e módulos de classe, pois eles compartilham o mesmo espaço de nome. Quando dois tipos **Enum** em diferentes tipos de biblioteca têm o mesmo nome, mas elementos diferentes, uma referência a uma variável do tipo depende de qual biblioteca de tipos tem prioridade mais alta nas **References**.

Você não pode usar um tipo **Enum** como destino de um bloco **With**.

## Instrução Erase

Reinicializa os elementos de matrizes de tamanho fixo e libera espaço de armazenamento de matrizes dinâmicas.

### Sintaxe

**Erase** *arraylist*

O argumento obrigatório *arraylist* é uma ou mais variáveis de matrizes delimitadas por vírgulas a serem apagadas.

### Comentários

**Erase** comporta-se de formas diferentes dependendo da matriz ser de tamanho fixo (normal) ou dinâmica. **Erase** não recupera memória para matrizes de tamanho fixo. **Erase** define os elementos de uma matriz fixa da seguinte forma:

<u>Tipo de matriz</u>	<u>Efeito de Erase em elementos de matriz fixa</u>
Matriz numérica fixa	Define cada elemento como zero.
Matriz de seqüência de caracteres fixa (tamanho variável)	Define cada elemento como uma seqüência de caracteres de comprimento zero ("").
Matriz de seqüência de caracteres fixa (tamanho fixo)	Define cada elemento como zero.
Matriz <u>Variant</u> fixa	Define cada elemento como <u>Empty</u> .
Matriz de <u>tipos definidos pelo usuário</u>	Define cada elemento como se fosse uma variável independente.
Matriz de objetos	Define cada elemento com o valor especial <b>Nothing</b> .

**Erase** libera a memória utilizada por matrizes dinâmicas. Antes que seu programa possa se referir novamente à matriz dinâmica, ele deverá declarar as dimensões da variável de matriz novamente, através da instrução **ReDim**.

## Instrução Event

Declara um evento definido pelo usuário.

### Sintaxe

**[Public] Event** *procedurename* [(*arglist*)]

A instrução **Event** tem estas partes:

<b>Parte</b>	<b>Descrição</b>
<b>Public</b>	Opcional. Especifica que o <b>Event</b> seja visível em todo o <u>projeto</u> . Os tipos de <b>Events</b> são <b>Public</b> como padrão. Observe que os eventos somente podem ser provocados no <u>módulo</u> em que são declarados.
<i>procedurename</i>	Obrigatório. Nomeie o evento; siga as convenções padrão de nomenclatura de variáveis.

O argumento *arglist* tem a seguinte sintaxe e partes:

**[ByVal | ByRef]** *varname*[( )] [**As** *type*]

Parte	Descrição
<b>ByVal</b>	Opcional. Indica que o <u>argumento</u> foi passado <u>por valor</u> .
<b>ByRef</b>	Opcional. Indica que o argumento é passado <u>por referência</u> . <b>ByRef</b> é o padrão no Visual Basic.
<i>varname</i>	Obrigatório. Nome da variável representando o argumento que está sendo passado ao <u>procedimento</u> ; segue as convenções padrão de nomenclatura de variáveis.
<i>type</i>	Opcional. <u>Tipo de dados</u> do argumento passado ao procedimento; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não suportado atualmente), <b>Date</b> , <b>String</b> (somente comprimento variável), <b>Object</b> , <b>Variant</b> , um <u>tipo definido pelo usuário</u> , ou um tipo de objeto.

### Comentários

Uma vez declarado o evento, use a instrução **RaiseEvent** para dispará-lo. Ocorrerá um erro de sintaxe se uma declaração **Event** aparecer em um módulo padrão. Um evento não pode ser declarado para retornar um valor. Um evento típico deve ser declarado e provocado como mostram os fragmentos abaixo:

```
' Declarar um evento a nível de módulo de um módulo de classe

Event LogonCompleted (UserName as String)

Sub
    RaiseEvent LogonCompleted("AntoineJan")
End Sub
```

## Instrução Function

Declara o nome, os argumentos e o código que formam o corpo de um procedimento Function.

### Sintaxe

```
[Public | Private] [Static] Function name [(arglist)] [As type]
    [statements]
    [name = expression]
[Exit]
[statements]
[name = expression]
End Function
```

### End Function

A sintaxe da instrução **Function** tem estas partes:

Parte	Descrição
<b>Public</b>	Opcional. Indica que o procedimento <b>Function</b> está acessível a todos os outros procedimentos em todos os <u>módulos</u> . Se utilizado em um módulo que contenha <b>Option Private</b> , o procedimento não estará disponível fora do <u>projeto</u> .
<b>Private</b>	Opcional. Indica que o procedimento <b>Function</b> está acessível apenas a outros procedimentos no módulo onde ele é declarado.
<b>Static</b>	Opcional. Indica que as <u>variáveis</u> locais do procedimento <b>Function</b> estão preservadas entre as chamadas. O atributo <b>Static</b> não afeta as variáveis que são declaradas fora de <b>Function</b> , mesmo que elas sejam utilizadas no procedimento.
<i>name</i>	Obrigatória. Nome de <b>Function</b> ; segue as convenções de

	nomenclatura padrão de variáveis.
<i>arglist</i>	Opcional. Lista de variáveis que representam argumentos passados ao procedimento <b>Function</b> quando ele é chamado. Variáveis múltiplas são separadas por vírgulas.
<i>type</i>	Opcional. O <u>tipo de dados</u> do valor retornado pelo procedimento <b>Function</b> ; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (atualmente não-suportado), <b>Date</b> , <b>String</b> (exceto seqüências de caracteres de comprimento fixo), <b>Object</b> , <b>Variant</b> ou qualquer <u>tipo</u> definido pelo usuário. Matrizes de qualquer tipo não podem ser retornadas, mas um <b>Variant</b> que contenha uma matriz pode.
<i>statements</i>	Opcional. Qualquer grupo de instruções a serem executadas dentro do procedimento <b>Function</b> .
<i>expression</i>	Opcional. Valor de retorno de <b>Function</b> .

O argumento *arglist* apresenta a sintaxe e as partes abaixo:

[Optional] [ByVal | ByRef] [ParamArray] *varname*( ) [As *type*] [= *defaultvalue*]

Parte	Descrição
<b>Optional</b>	Opcional. Indica que um argumento não é obrigatório. Se utilizada, todos os argumentos subseqüentes em <i>arglist</i> deverão ser opcionais e declarados através da palavra-chave <b>Optional</b> . <b>Optional</b> não pode ser utilizado para qualquer argumento se <b>ParamArray</b> for utilizado.
<b>ByVal</b>	Opcional. Indica que o argumento é passado <u>por valor</u> .
<b>ByRef</b>	Opcional. Indica que o argumento é passado <u>por referência</u> . <b>ByRef</b> é o padrão no Visual Basic.
<b>ParamArray</b>	Opcional. Utilizada apenas como o último argumento em <i>arglist</i> para indicar que o argumento final é uma matriz <b>Optional</b> de elementos <b>Variant</b> . A palavra-chave <b>ParamArray</b> permite que você forneça um número arbitrário de argumentos. Ela não pode ser utilizada com <b>ByVal</b> , <b>ByRef</b> ou <b>Optional</b> .
<i>varname</i>	Obrigatória. Nome da variável que representa o argumento; segue as convenções de nomenclatura padrão de variáveis.
<i>type</i>	Opcional. Tipo de dados do argumento passado ao procedimento; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (atualmente não-suportado) <b>Date</b> , <b>String</b> (somente de comprimento variável), <b>Object</b> , <b>Variant</b> . Se o parâmetro não for <b>Optional</b> , um tipo definido pelo usuário ou um <u>tipo de objeto</u> também pode ser especificado.
<i>defaultvalue</i>	Opcional. Qualquer <u>constante</u> ou expressão de constante. Válido somente para parâmetros <b>Optional</b> . Se o tipo for um <b>Object</b> , um valor padrão explícito só poderá ser <b>Nothing</b> .

### Comentários

Se não forem especificados explicitamente através de **Public** ou **Private**, os procedimentos **Function** serão públicos como padrão. Se **Static** não for utilizado, o valor de variáveis locais não será preservado entre as chamadas. A palavra-chave **Friend** somente pode ser usada em módulos de classe. Entretanto, procedimentos **Friend** podem ser acessados por procedimentos em qualquer módulo de um projeto. Um procedimento **Friend** não aparece na biblioteca de tipos de sua classe pai, nem pode um procedimento **Friend** ser acoplado posteriormente.

**Atenção** Os procedimentos **Function** podem ser recursivos; ou seja, eles podem chamar a si próprios para efetuar uma determinada tarefa. No entanto, a recursão pode levar ao estouro da pilha. A palavra-chave **Static** em geral não é utilizada com procedimentos recursivos **Function**.

Todo o código executável deve estar em procedimentos. Você não pode definir um procedimento **Function** dentro de outro procedimento **Function**, **Sub** ou **Property**.

A instrução **Exit Function** causa a saída imediata de um procedimento **Function**. A execução do programa continua com a instrução seguinte àquela que chamou o procedimento **Function**. Qualquer número de instruções **Exit Function** pode aparecer em qualquer lugar de um procedimento **Function**.

Assim como um procedimento **Sub**, um procedimento **Function** é um procedimento separado que pode aceitar argumentos, executar uma série de instruções e alterar os valores de seus argumentos. No entanto, ao contrário de um procedimento **Sub**, você pode utilizar um procedimento **Function** à direita de uma expressão da mesma forma que você utiliza qualquer função intrínseca, como **Sqr**, **Cos** ou **Chr**, quando você deseja utilizar o valor retornado pela função.

Você chama um procedimento **Function** utilizando o nome da função, seguido pela lista de argumentos entre parênteses, em uma expressão. Consulte a instrução **Call** para obter Especificidades sobre como chamar procedimentos **Function**.

Para retornar um valor de uma função, atribua o valor ao nome da função. Qualquer quantidade dessas atribuições pode aparecer em qualquer lugar do procedimento. Se não for atribuído um valor a *name*, o procedimento retornará um valor padrão: uma função numérica retornará 0, uma função de seqüência de caracteres retornará uma seqüência de caracteres de comprimento zero ("") e uma função **Variant** retornará **Empty**. Uma função que retorna uma referência de objeto retornará **Nothing** se nenhuma referência de objeto for atribuída a *name* (utilizando **Set**) dentro de **Function**.

O exemplo abaixo mostra como atribuir um valor de retorno a uma função denominada PesquisaBinária. Neste caso, **False** é atribuído ao nome para indicar que um certo valor não foi encontrado.

```
Function PesquisaBinária(. . .) As Boolean
. . .
    ' Valor não encontrado. Retorna um valor False.
    If inferior > superior Then
        PesquisaBinária = False
        Exit Function
    End If
. . .
End Function
```

As variáveis utilizadas em procedimentos **Function** dividem-se em duas categorias: aquelas explicitamente declaradas dentro do procedimento e aquelas que não são declaradas dessa forma. As variáveis que são explicitamente declaradas dentro de um procedimento (utilizando **Dim** ou equivalente) são sempre locais em relação ao procedimento. As variáveis utilizadas mas não explicitamente declaradas em um procedimento também são locais, a menos que explicitamente declaradas em algum nível superior fora do procedimento.

---

**Atenção** Um procedimento pode utilizar uma variável que não é explicitamente declarada no procedimento, mas poderá ocorrer um conflito de nomenclatura se algo definido no nível do módulo apresentar o mesmo nome. Se o seu procedimento referir-se a uma variável não declarada que tenha o mesmo nome de outro procedimento, constante ou variável, pressupõe-se que seu procedimento se refere àquele nome do nível do módulo. Para evitar esse tipo de conflito, convém declarar variáveis de forma explícita. Você pode utilizar uma instrução **Option Explicit** para forçar a declaração explícita de variáveis.

---

**Atenção** O Visual Basic pode reorganizar expressões aritméticas para aumentar a eficácia interna. Evite utilizar um procedimento **Function** em uma expressão aritmética quando a função alterar o valor das variáveis na mesma expressão.

---

## Função GetObject

Retorna uma referência a um objeto ActiveX de um arquivo.

### Sintaxe

**GetObject**([*pathname*] [, *class*])

A sintaxe da função **GetObject** apresenta os argumentos nomeados abaixo:

Parte	Descrição
<i>pathname</i>	Opcional; <b>Variant (String)</b> . Indica o caminho completo e nome do arquivo que contém o objeto a ser recuperado. Se <i>pathname</i> for omitido, <i>class</i> passa a ser obrigatório.
<i>class</i>	Opcional; <b>Variant (String)</b> . Uma seqüência de caracteres que representa a <u>classe</u> do objeto.

O argumento class utiliza a sintaxe *appname.objecttype* e possui partes abaixo:

Parte	Descrição
<i>appname</i>	Obrigatória; <b>Variant (String)</b> . Nome do aplicativo que fornece o objeto.
<i>objecttype</i>	Obrigatória; <b>Variant (String)</b> . Tipo ou classe do objeto a ser criado.

### Comentários

Utilize a função **GetObject** para acessar um objeto ActiveX de um arquivo e atribuir o objeto a uma variável de objeto. Utilize a instrução **Set** para atribuir o objeto retornado por **GetObject** à variável do objeto. Por exemplo:

```
Dim ObjetoCAD As Object
Set ObjetoCAD = GetObject("C:\CAD\SCHEMA.CAD")
```

Quando esse código é executado, o aplicativo associado ao *pathname* especificado é iniciado e o objeto no arquivo especificado é ativado.

Se *pathname* for uma seqüência de caracteres de comprimento zero (""), **GetObject** retornará uma nova ocorrência de objeto do tipo especificado. Se o argumento do *pathname* for omitido, **GetObject** retornará um objeto atualmente ativo do tipo especificado. Se não houver nenhum objeto do tipo especificado, um erro será gerado.

Alguns aplicativos permitem que você ative parte de um arquivo. Adicione um ponto de exclamação (!) ao final do nome do arquivo e acrescente uma seqüência de caracteres que identifique a parte do arquivo que você deseja ativar. Para obter informações sobre como criar essa seqüência de caracteres, consulte a documentação referente ao aplicativo que criou o objeto.

Por exemplo, em um aplicativo de desenho, você pode ter diversas camadas de um desenho armazenado em um arquivo. Você poderia utilizar o código abaixo para ativar uma camada em um desenho denominado SCHEMA.CAD:

```
Set CamadaObjeto = GetObject("C:\CAD\SCHEMA.CAD!Camada3")
```

Se você não especificar o *class* do objeto, a Automação determinará o aplicativo a ser iniciado e o objeto a ser ativado, de acordo com o nome de arquivo fornecido. Alguns arquivos, contudo, podem suportar mais do que uma classe de objeto. Por exemplo, um desenho poderia suportar três diferentes tipos de objetos: um objeto **Application**, um objeto **Drawing** e um objeto **Toolbar**, os quais fazem parte do mesmo arquivo. Para especificar o objeto de um arquivo que você deseja ativar, utilize o argumento de *class* opcional. Por exemplo:

```
Dim MeuObjeto As Object
Set MeuObjeto = GetObject("C:\DRAWINGS\SAMPLE.DRW", "FIGMENT.DRAWING")
```

No exemplo acima, FIGMENT é o nome de um aplicativo de desenho e DRAWING é um dos tipos de objeto que ele suporta.

Depois que o objeto tiver sido ativado, você pode referenciá-lo em código utilizando a variável de objeto que você definiu. No exemplo anterior, você acessa as propriedades e os métodos do novo

objeto utilizando a variável de objeto `MeuObjeto`. Por exemplo:

```
MeuObjeto.Line 9, 90
MeuObjeto.InsertText 9, 100, "Olá, mundo."
MeuObjeto.SaveAs "C:\DRAWINGS\SAMPLE.DRW"
```

**Observação:** Utilize a função **GetObject** quando houver uma ocorrência atual do objeto ou quando você desejar criar o objeto com um arquivo já carregado. Se não houver uma ocorrência atual e se você não quiser iniciar o objeto com um arquivo carregado, utilize a função **CreateObject**.

Se um objeto registrou a si próprio como um objeto de ocorrência única, apenas uma ocorrência desse objeto será criada, independente de quantas vezes **CreateObject** for executado. Com um objeto de ocorrência única, **GetObject** sempre retornará a mesma ocorrência quando chamado com a sintaxe da seqüência de caracteres de comprimento zero (""), e se o argumento do **pathname** for omitido, um erro será gerado. Você não pode utilizar **GetObject** para obter uma referência a uma classe criada com o Visual Basic.

## Instrução Implements

Especifica uma interface ou classe que será implementada no módulo de classe onde ele aparece.

### Sintaxe

**Implements** [*InterfaceName* | *Class*]

O *InterfaceName* ou *Class* obrigatório é o nome de uma interface ou classe na biblioteca de tipos, cujos métodos serão implementados pelos métodos correspondentes na classe Visual Basic.

### Comentários

Uma interface é uma coleção de protótipos representando os membros (métodos e propriedades) encapsulados na interface; isto é, contém apenas as declarações para os procedimentos membros. Uma classe oferece uma implementação de todos os métodos e propriedades de uma ou mais interfaces. As classes oferecem o código usado quando cada função é chamada por um controlador da classe. Todas as classes implementam pelo menos uma interface, que é considerada a interface padrão da classe. No Visual Basic, todo membro que não é explicitamente um membro de uma interface implementada é implicitamente um membro da interface padrão.

Quando uma classe do Visual Basic implementa uma interface, a classe do Visual Basic oferece sua própria versão de todos os procedimentos Public especificados na biblioteca de tipos da interface. Além de oferecer um mapeamento entre protótipos de interface e seus procedimentos, a instrução **Implements** provoca a aceitação de chamada COM QueryInterfaces pela classe para a identificação da interface especificada.

Quando você implementa uma interface ou classe, você deve incluir todos os procedimentos **Public** envolvidos. Um membro faltando em uma implementação de interface ou classe provoca um erro. Se você não colocar código em um dos procedimentos de uma classe que estiver implementando, você pode provocar o erro adequado (**Const E\_NOTIMPL = &H80004001**) de modo que o usuário da implementação compreenda que o membro não está implementado.

A instrução **Implements** não pode aparecer em um módulo padrão.

## Função LBound

Retorna um **Long** que contém o menor subscrito disponível para a dimensão indicada de uma matriz.

### Sintaxe

**LBound**(*arrayname* [, *dimension*])

A sintaxe da função **LBound** apresenta as partes abaixo:

Parte	Descrição
<i>arrayname</i>	Obrigatória. Nome da <u>variável</u> de matriz, de acordo com as convenções de nomenclatura padrão de variáveis.
<i>dimension</i>	Opcional; <b>Variant (Long)</b> . Número inteiro que indica qual limite inferior da dimensão é retornado. Utilize 1 para a primeira dimensão, 2 para a segunda e assim por diante. Se <i>dimension</i> for omitida, será utilizado 1.

### Comentários

A função **LBound** é utilizada com a função **UBound** para determinar o tamanho de uma matriz. Utilize a função **UBound** para calcular o limite superior de uma dimensão de matriz.

**LBound** retorna os valores apresentados na tabela abaixo para uma matriz com as dimensões abaixo:

Dim A(1 To 100, 0 To 3, -3 To 4)

Instrução	Valor de retorno
LBound (A, 1)	1
LBound (A, 2)	0
LBound (A, 3)	-3

O limite inferior padrão para qualquer dimensão é 0 ou 1, de acordo com a definição da instrução **Option Base**. A base de uma matriz criada com a função **Array** é zero; ela não é afetada por **Option Base**.

As matrizes para as quais são definidas dimensões utilizando-se a cláusula **To** em uma instrução **Dim**, **Private**, **Public**, **ReDim** ou **Static** podem apresentar qualquer valor inteiro como um limite inferior.

## Instrução Let

Atribui o valor de uma expressão a uma variável ou propriedade.

### Sintaxe

[**Let**] *varname* = *expression*

A sintaxe da instrução **Let** tem estas partes:

Parte	Descrição
<b>Let</b>	Opcional. A utilização explícita da <u>palavra-chave</u> <b>Let</b> é uma questão de estilo, mas ela é em geral omitida.
<i>varname</i>	Obrigatória. Nome da variável ou propriedade; segue as convenções de nomenclatura padrão de variáveis.
<i>expression</i>	Obrigatória. Valor atribuído à variável ou propriedade.

### Comentários

Uma expressão de valor pode ser atribuída a uma variável ou propriedade somente se for de um tipo de dados compatível com a variável. Você não pode atribuir expressões de seqüências de caracteres a variáveis numéricas e não pode atribuir expressões numéricas a variáveis de seqüências de caracteres. Se o fizer, ocorrerá um erro em tempo de compilação.

A variáveis **Variant** podem ser atribuídas tanto expressões de seqüência de caracteres quanto expressões numéricas. No entanto, o contrário nem sempre é verdadeiro. Qualquer **Variant** exceto **Null** pode ser atribuído a uma variável de seqüência de caracteres, mas apenas um **Variant** cujo valor possa ser interpretado como um número pode ser atribuído a uma variável numérica. Utilize a função **IsNumeric** para determinar se o **Variant** pode ser convertido em um número.

---

**Atenção** Atribuir uma expressão de um tipo numérico a uma variável de um tipo numérico diferente converterá o valor da expressão no tipo numérico da variável resultante.

---

As instruções **Let** podem ser utilizadas para atribuir uma variável de registro a outra apenas quando ambas as variáveis são do mesmo tipo definido pelo usuário. Utilize a instrução **LSet** para atribuir variáveis de registro a diferentes tipos definidos pelo usuário. Utilize a instrução **Set** para atribuir referências de objetos às variáveis.

## Instrução Option Base

Utilizada no nível de módulo para declarar o limite inferior padrão para os subscripts de matriz.

### Sintaxe

**Option Base** {0 | 1}

### Comentários

Uma vez que a base padrão é **0**, a instrução **Option Base** nunca é exigida. Se utilizada, a instrução deve aparecer em um módulo antes de qualquer procedimento. **Option Base** só pode aparecer uma vez em um módulo e deve anteceder as declarações de matrizes que incluam dimensões.

**Observação:** A cláusula **To** nas instruções **Dim**, **Private**, **Public**, **ReDim** e **Static** fornecem uma maneira mais flexível de controlar o intervalo de subscritos de uma matriz. No entanto, se você não definir explicitamente o limite mínimo com uma cláusula **To**, poderá utilizar **Option Base** para alterar o limite inferior padrão para 1. A base de uma matriz criada com a função **Array** ou com a palavra-chave **ParamArray** é zero; **Option Base** não afeta **Array** ou **ParamArray**.

A instrução **Option Base** afeta somente o limite inferior das matrizes no módulo no qual se localiza a instrução.

## Instrução Option Compare

Utilizada no nível de módulo para declarar o método de comparação padrão a ser utilizado quando dados de seqüência de caracteres forem comparados.

### Sintaxe

**Option Compare** {Binary | Text | Database}

### Comentários

Se utilizada, a instrução **Option Compare** deve aparecer em um módulo antes dos procedimentos.

A instrução **Option Compare** especifica o método de comparação de seqüências de caracteres (**Binary**, **Text** ou **Database**) de um módulo. Se um módulo não incluir uma instrução **Option Compare**, o método de comparação de texto padrão será **Binary**.

**Option Compare Binary** resulta em comparações de seqüências de caracteres baseadas em uma ordem de classificação derivada das representações binárias internas dos caracteres. No Microsoft Windows, a ordem de classificação é determinada pela página de código. Uma ordem de classificação binária típica é mostrada no exemplo abaixo:

A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø

**Option Compare Text** resulta em comparações de seqüências de caracteres baseadas em uma ordem de classificação de texto que não coincide maiúsculas e minúsculas determinada pela localidade do seu sistema. Quando alguns caracteres são classificados utilizando-se **Option Compare Text**, a ordem de classificação de texto abaixo é produzida:

(A=a) < (À=à) < (B=b) < (E=e) < (Ê=ê) < (Z=z) < (Ø=ø)

**Option Compare Database** somente pode ser utilizado dentro do Microsoft Access. Isso resulta em comparações de seqüências de caracteres baseadas na ordem de classificação determinada pelo código de localidade do banco de dados em que ocorrem as comparações de seqüências de caracteres.

## Instrução Option Explicit

Utilizada no nível de módulo para forçar a declaração explícita de todas as variáveis desse módulo.

### Sintaxe

#### Option Explicit

#### Comentários

Se utilizada, a instrução **Option Explicit** deve aparecer em um módulo antes de qualquer procedimento.

Quando **Option Explicit** aparece em um módulo, você deve declarar explicitamente todas as variáveis utilizando as instruções **Dim**, **Private**, **Public**, **ReDim** ou **Static**. Se você tentar utilizar um nome de variável não-declarado, ocorrerá um erro em tempo de compilação.

Se você não utilizar a instrução **Option Explicit**, todas as variáveis não-declaradas serão do tipo **Variant**, a menos que o tipo padrão seja especificado com uma instrução **Deftype**.

**Observação:** Utilize **Option Explicit** para evitar a digitação incorreta do nome de uma variável existente ou para evitar confusão no código em que o escopo da variável não esteja claro.

## Instrução Option Private

Quando utilizada em aplicativos host que permitem referências em múltiplos projetos, **Option Private Module** impede que o conteúdo de um módulo seja referenciado fora do seu projeto. Em aplicativos host que não permitem essas referências, por exemplo, em versões independentes do Visual Basic, **Option Private** não terá efeito.

### Sintaxe

#### Option Private Module

#### Comentários

Se utilizada, a instrução **Option Private** deve aparecer no nível de módulo antes de qualquer procedimento.

Quando um módulo contém **Option Private Module**, as partes públicas, por exemplo, variáveis, objetos e tipos definidos pelo usuário declarados no nível de módulo, continuam disponíveis dentro do projeto que contém o módulo, mas não ficam disponíveis para outros aplicativos ou projetos.

**Observação:** **Option Private** é somente útil para aplicativos host que suportam carregamento simultâneo de vários projetos e permitem referências entre os projetos carregados. Por exemplo, o Microsoft Excel permite carregamento de vários projetos e **Option Private Module** pode ser utilizado para restringir a visibilidade do projeto cruzado. Ainda que o Visual Basic permita o carregamento de vários projetos, referências entre projetos nunca são permitidas no Visual Basic.

## Instrução Private

Utilizada no nível de módulo para declarar variáveis privadas e alocar espaço de armazenamento.

### Sintaxe

**Private** [**WithEvents**] *varname*[[*(subscripts)*]] [**As** [**New**] *type*] [, [**WithEvents**] *varname*[[*(subscripts)*]] [**As** [**New**] *type*]] . . .

A sintaxe da instrução **Private** tem estas partes:

Parte	Descrição
<b>WithEvents</b>	Opcional. <u>Palavra-chave</u> que especifica que <i>varname</i> é uma <u>variável de objeto</u> utilizada para responder a eventos acionados por um <u>objeto ActiveX</u> . Válida somente em <u>módulos de classe</u> . Você pode declarar quantas variáveis individuais quiser utilizando <b>WithEvents</b> , mas não pode criar <u>matrizes</u> com <b>WithEvents</b> . Você não pode utilizar <b>New</b> com <b>WithEvents</b> .
<i>varname</i>	Obrigatória. Nome da variável; segue as convenções de nomenclatura padrão de variáveis.
<i>subscripts</i>	Opcional. Dimensões de uma variável de matriz; até 60 dimensões múltiplas podem ser declaradas. O <u>argumento subscripts</u> utiliza a sintaxe abaixo: [ <i>lower To</i> ] <i>upper</i> [, [ <i>lower To</i> ] <i>upper</i> ] . . . Quando não enunciado explicitamente em <i>lower</i> , o limite inferior de uma matriz é controlado pela instrução <b>Option Base</b> . O limite inferior será zero se não estiver presente uma instrução <b>Option Base</b> .
<b>New</b>	Opcional. Palavra-chave que permite a criação implícita de um objeto. Se você utilizar <b>New</b> ao declarar a variável de objeto, uma nova ocorrência do objeto será criada na primeira referência a ele; portanto, você não precisa utilizar a instrução <b>Set</b> para atribuir a referência do objeto. A palavra-chave <b>New</b> não pode ser utilizada para declarar variáveis de <u>tipo de dados</u> intrínseco, não pode ser utilizada para declarar instâncias de objetos dependentes e não pode ser utilizada com <b>WithEvents</b> .
<i>type</i>	Opcional. Tipo de dados da variável; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não-suportado atualmente), <b>Date</b> , <b>String</b> (para seqüências de caracteres de comprimento variável), <b>String * length</b> (para seqüências de caracteres de comprimento fixo), <b>Object</b> , <b>Variant</b> , um <u>tipo definido pelo usuário</u> ou um <u>tipo de objeto</u> . Utilize uma cláusula <b>As type</b> para cada variável que estiver sendo definida.

### Comentários

As variáveis **Private** encontram-se disponíveis somente para o módulo em que são declaradas.

Utilize a instrução **Private** para declarar o tipo de dados de uma variável. Por exemplo, a instrução abaixo declara uma variável como um **Integer**:

```
Private NúmeroDeEmpregados As Integer
```

Você também pode utilizar uma instrução **Private** para declarar o tipo de objeto de uma variável. A instrução abaixo declara uma variável para uma nova ocorrência de uma planilha.

```
Private X As New Worksheet
```

Se a palavra-chave **New** não for utilizada quando declarar uma variável de objeto, a variável que se refere ao objeto deverá ser atribuída a um objeto existente utilizando-se a instrução **Set** para que

possa ser utilizada. Até que seja atribuído um objeto, a variável de objeto declarada terá o valor especial **Nothing**, que indica que não se refere a uma ocorrência específica de um objeto.

Se você não especificar um tipo de dados ou tipo de objeto e não houver instrução **Defype** no módulo, a variável será **Variant** como padrão.

Você também pode utilizar a instrução **Private** com parênteses vazios para declarar uma matriz dinâmica. Depois de declarar uma matriz dinâmica, utilize a instrução **ReDim** dentro de um procedimento para definir o número de dimensões e elementos na matriz. Se você tentar declarar novamente uma dimensão de uma variável de matriz cujo tamanho tenha sido explicitamente especificado em uma instrução **Private**, **Public** ou **Dim**, um erro será gerado.

Quando variáveis são inicializadas, uma variável numérica é inicializada como 0, uma seqüência de caracteres de comprimento variável é inicializada como uma seqüência de caracteres de comprimento zero ("") e uma seqüência de caracteres de comprimento fixo é preenchida com zeros. As variáveis **Variant** são inicializadas como **Empty**. Cada elemento de uma variável de tipo definido pelo usuário é inicializado como se fosse uma variável separada.

**Observação:** Quando você utiliza a instrução **Private** em um procedimento, geralmente coloca a instrução **Private** no início dele.

## Instrução Property Get

Declara o nome, os argumentos e o código que formam o corpo de um procedimento **Property**, que obtém o valor de uma propriedade.

### Sintaxe

```
[Public | Private] [Static] Property Get name [(arglist)] [As type]
    [statements]
    [name = expression]
[Exit Property]
[statements]
[name = expression]
```

### End Property

A sintaxe da instrução **Property Get** tem estas partes:

Parte	Descrição
<b>Public</b>	Opcional. Indica que o procedimento <b>Property Get</b> é acessível a todos os outros procedimentos de todos os módulos. Se utilizada em um módulo que contém uma instrução <b>Option Private</b> , o procedimento não se encontrará disponível fora do <u>projeto</u> .
<b>Private</b>	Opcional. Indica que o procedimento <b>Property Get</b> é acessível somente a outros procedimentos do módulo em que é declarado.
<b>Static</b>	Opcional. Indica que as <u>variáveis</u> locais do procedimento <b>Property Get</b> são preservadas entre chamadas. O atributo <b>Static</b> não afeta variáveis que sejam declaradas fora do procedimento <b>Property Get</b> , mesmo que elas sejam utilizadas no procedimento.
<i>name</i>	Obrigatória. Nome do procedimento <b>Property Get</b> ; segue as convenções de nomenclatura padrão de variáveis, exceto que o nome pode ser o mesmo de um procedimento <b>Property Let</b> ou <b>Property Set</b> do mesmo módulo.
<i>arglist</i>	Opcional. Lista de variáveis que representam argumentos que são passados ao procedimento <b>Property Get</b> quando ele é chamado. Múltiplos argumentos são separados por vírgulas. O nome e o <u>tipo de dados</u> de cada argumento de um procedimento <b>Property Get</b> devem ser iguais ao argumento correspondente de um procedimento <b>Property</b>

<i>type</i>	<b>Let</b> (se houver um). Opcional. Tipo de dados do valor retornado pelo procedimento <b>Property Get</b> ; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não-suportado atualmente), <b>Date</b> , <b>String</b> (exceto comprimento fixo), <b>Object</b> , <b>Variant</b> ou tipo definido pelo usuário. Não podem ser retornadas <u>matrizes</u> de tipo algum, mas um <b>Variant</b> que contenha uma matriz pode. O <i>type</i> de retorno de um procedimento <b>Property Get</b> deve ser do mesmo tipo de dados do último (ou algumas vezes do único) argumento de um procedimento <b>Property Let</b> correspondente (se houver um) que define o valor atribuído à propriedade do lado direito de uma <u>expressão</u> .
<i>statements</i>	Opcional. Qualquer grupo de instruções a serem executadas dentro do corpo do procedimento <b>Property Get</b> .
<i>expression</i>	Opcional. Valor da propriedade retornado pelo procedimento definido pela instrução <b>Property Get</b> .

O argumento *arglist* possui a sintaxe e as partes abaixo:

[Optional] [ByVal | ByRef] *varname*[( )] [As *type*] [= *defaultvalue*]

Parte	Descrição
<b>Optional</b>	Opcional. Indica que um argumento não é obrigatório. Se utilizada, todos os argumentos subseqüentes em <i>arglist</i> deverão também ser opcionais e declarados utilizando-se a palavra-chave <b>Optional</b> .
<b>ByVal</b>	Opcional. Indica que o argumento é passado <u>por valor</u> .
<b>ByRef</b>	Opcional. Indica que o argumento é passado <u>por referência</u> . <b>ByRef</b> é o padrão do Visual Basic.
<b>ParamArray</b>	Opcional. Utilizada apenas como o último argumento em <i>arglist</i> para indicar que o argumento final é uma matriz <b>Optional</b> de elementos <b>Variant</b> . A palavra-chave <b>ParamArray</b> permite que você forneça um número arbitrário de argumentos. Ela não deve ser utilizada com <b>ByVal</b> , <b>ByRef</b> ou <b>Optional</b> .
<i>varname</i>	Obrigatória. Nome da variável que representa o argumento; segue as convenções de nomenclatura padrão de variáveis.
<i>type</i>	Opcional. Tipo de dados do argumento passado ao procedimento; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não-suportado atualmente), <b>Date</b> , <b>String</b> (somente de comprimento variável), <b>Object</b> , <b>Variant</b> . Se o parâmetro não for <b>Optional</b> , um tipo definido pelo usuário ou um <u>tipo de objeto</u> também poderá ser especificado.
<i>defaultvalue</i>	Opcional. Qualquer <u>constante</u> ou expressão de constante. Válido somente para parâmetros <b>Optional</b> . Se o tipo for um <b>Object</b> , um valor padrão explícito somente poderá ser <b>Nothing</b> .

### Comentários

Se não especificado explicitamente utilizando-se **Public** ou **Private**, os procedimentos **Property** serão públicos como padrão. Se **Static** não for utilizado, o valor de variáveis locais não será preservado entre chamadas.

Todo o código executável deve estar nos procedimentos. Você não pode definir um procedimento **Property Get** dentro de outro procedimento **Property**, **Sub** ou **Function**.

A instrução **Exit Property** causa uma saída imediata de um procedimento **Property Get**. A execução do programa continua com a instrução que segue a instrução que chamou o procedimento **Property Get**. Pode aparecer qualquer número de instruções **Exit Property** em qualquer lugar de um

procedimento **Property Get**.

Como um procedimento **Sub** e **Property Let**, um procedimento **Property Get** é um procedimento separado que pode tomar argumentos, efetuar uma série de instruções e alterar os valores de seus argumentos. Entretanto, ao contrário de um procedimento **Sub** ou **Property Let**, você pode utilizar um procedimento **Property Get** do lado direito de uma expressão da mesma forma que utiliza um **Function** ou um nome de propriedade quando deseja retornar o valor de uma propriedade.

## Instrução Property Let

Declara o nome, os argumentos e o código que formam o corpo de um procedimento Property Let, que atribui um valor a uma propriedade.

### Sintaxe

```
[Public | Private] [Static] Property Let name ([arglist,] value)
    [statements]
[Exit Property]
    [statements]
End Property
```

A sintaxe da instrução **Property Let** tem estas partes:

Parte	Descrição
<b>Public</b>	Opcional. Indica que o procedimento <b>Property Let</b> é acessível a todos os outros procedimentos de todos os <u>módulos</u> . Se utilizada em um módulo que contém uma instrução <b>Option Private</b> , o procedimento não estará disponível fora do <u>projeto</u> .
<b>Private</b>	Opcional. Indica que o procedimento <b>Property Let</b> é acessível somente para outros procedimentos do módulo em que é declarado.
<b>Static</b>	Opcional. Indica que as <u>variáveis</u> locais do procedimento <b>Property Let</b> são preservadas entre chamadas. O atributo <b>Static</b> não afeta variáveis que sejam declaradas fora do procedimento <b>Property Let</b> , mesmo que elas sejam utilizadas no procedimento.
<i>nome</i>	Obrigatória. Nome do procedimento <b>Property Let</b> ; segue as convenções de nomenclatura padrão de variáveis, exceto que o nome pode ser o mesmo de um procedimento <b>Property Get</b> ou <b>Property Set</b> do mesmo módulo.
<i>arglist</i>	Obrigatória. Lista de variáveis que representam argumentos que são passados ao procedimento <b>Property Let</b> quando ele é chamado. Múltiplos argumentos são separados por vírgulas. O nome e o <u>tipo de dados</u> de cada argumento de um procedimento <b>Property Let</b> (exceto o último) devem ser iguais ao argumento correspondente de um procedimento <b>Property Get</b> .
<i>value</i>	Obrigatória. Variável que contém o valor a ser atribuído à propriedade. Quando o procedimento é chamado, este argumento aparece no lado direito da <u>expressão</u> de chamada. O tipo de dados de <i>value</i> deve ser o mesmo que o tipo de retorno do procedimento <b>Property Get</b> correspondente.
<i>statements</i>	Opcional. Qualquer grupo de <u>instruções</u> a ser executado dentro do procedimento <b>Property Let</b> .

O argumento *arglist* possui a sintaxe e as partes abaixo:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type] [= defaultvalue]
```

Parte	Descrição
<b>Optional</b>	Opcional. Indica que um argumento não é obrigatório. Se

	utilizada, todos os argumentos subseqüentes em <i>arglist</i> devem também ser opcionais e declarados utilizando-se a palavra-chave <b>Optional</b> . Observe que o lado direito de uma expressão <b>Property Let</b> não pode ser <b>Optional</b> .
<b>ByVal</b>	Opcional. Indica que o argumento é passado <u>por valor</u> .
<b>ByRef</b>	Opcional. Indica que o argumento é passado <u>por referência</u> . <b>ByRef</b> é o padrão no Visual Basic.
<b>ParamArray</b>	Opcional. Utilizada apenas como o último argumento em <i>arglist</i> para indicar que o argumento final é uma matriz <b>Optional</b> de elementos <b>Variant</b> . A palavra-chave <b>ParamArray</b> permite que você forneça um número arbitrário de argumentos. Ela não pode ser utilizada com <b>ByVal</b> , <b>ByRef</b> ou <b>Optional</b> .
<i>varname</i>	Obrigatória. Nome da variável que representa o argumento; segue as convenções de nomenclatura padrão de variáveis.
<i>type</i>	Opcional. Tipo de dados do argumento passado ao procedimento; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não-suportado atualmente), <b>Date</b> , <b>String</b> (somente de comprimento variável), <b>Object</b> , <b>Variant</b> . Se o parâmetro não for <b>Optional</b> , também pode ser especificado um <u>tipo definido pelo usuário</u> ou um <u>tipo de objeto</u> .
<i>defaultvalue</i>	Opcional. Qualquer <u>constante</u> ou expressão de constante. Válido somente para parâmetros <b>Optional</b> . Se o tipo for um <b>Object</b> , um valor padrão explícito somente poderá ser <b>Nothing</b> .

**Observação:** Toda instrução **Property Let** deve definir pelo menos um argumento para o procedimento que ela define. Esse argumento (ou o último argumento se houver mais de um) contém o valor real a ser atribuído à propriedade quando o procedimento definido pela instrução **Property Let** for chamado. Tal argumento é referido como *value* na sintaxe anterior.

### Comentários

Se não especificados explicitamente utilizando-se **Public** ou **Private**, os procedimentos **Property** serão públicos como padrão. Se **Static** não for utilizado, o valor de variáveis locais não será preservado entre chamadas.

Todo o código executável deve estar nos procedimentos. Você não pode definir um procedimento **Property Let** dentro de outro procedimento **Property**, **Sub** ou **Function**.

A instrução **Exit Property** causa uma saída imediata de um procedimento **Property Let**. A execução do programa continua com a instrução que segue a instrução que chamou o procedimento **Property Let**. Pode aparecer qualquer número de instruções **Exit Property** em qualquer lugar de um procedimento **Property Let**.

Da mesma forma que um procedimento **Function** e **Property Get**, um procedimento **Property Let** é um procedimento separado que pode tomar argumentos, efetuar uma série de instruções e alterar o valor de seus argumentos. Entretanto, ao contrário de um procedimento **Function** e **Property Get**, os quais retornam um valor, você somente pode utilizar um procedimento **Property Let** do lado esquerdo de uma expressão de atribuição de propriedade ou instrução **Let**.

## Instrução Property Set

Declara o nome, os argumentos e o código que formam o corpo de um procedimento **Property**, o qual define uma referência a um objeto.

### Sintaxe

```
[Public | Private] [Static] Property Set name ([arglist,] reference)
    [statements]
[Exit
    [statements]
End Property
```

A sintaxe da instrução **Property Set** tem estas partes:

Parte	Descrição
<b>Optional</b>	Opcional. Indica que o argumento pode ou não ser fornecido pelo chamador.
<b>Public</b>	Opcional. Indica que o procedimento <b>Property Set</b> é acessível a todos os outros procedimentos em todos os <u>módulos</u> . Se utilizada em um módulo que contém uma instrução <b>Option Private</b> , o procedimento não estará disponível fora do <u>projeto</u> .
<b>Private</b>	Opcional. Indica que o procedimento <b>Property Set</b> é acessível somente a outros procedimentos do módulo em que é declarado.
<b>Static</b>	Opcional. Indica que as <u>variáveis</u> locais do procedimento <b>Property Set</b> são preservadas entre chamadas. O atributo <b>Static</b> não afeta variáveis que sejam declaradas fora do procedimento <b>Property Set</b> , mesmo se forem utilizadas no procedimento.
<i>name</i>	Obrigatória. Nome do procedimento <b>Property Set</b> ; segue as convenções de nomenclatura padrão de variáveis, exceto que o nome pode ser o mesmo de um procedimento <b>Property Get</b> ou <b>Property Let</b> do mesmo módulo.
<i>arglist</i>	Obrigatória. Lista de variáveis que representam argumentos que são passados ao procedimento <b>Property Set</b> quando ele é chamado. Múltiplos argumentos são separados por vírgulas.
<i>reference</i>	Obrigatória. Variável que contém a referência ao objeto utilizada no lado direito da atribuição de referência ao objeto.
<i>statements</i>	Opcional. Qualquer grupo de instruções a ser executado dentro do corpo do procedimento <b>Property</b> .

O argumento *arglist* possui a sintaxe e as partes abaixo:

```
[Optional] [ByVal | ByRef] varname( ) [As type] [= defaultvalue]
```

Parte	Descrição
<b>Optional</b>	Opcional. Indica que um argumento não é obrigatório. Se utilizada, todos os argumentos subseqüentes em <i>arglist</i> também deverão ser opcionais e declarados utilizando-se a palavra-chave <b>Optional</b> . Observe que o lado direito de uma <u>expressão</u> <b>Property Set</b> não pode ser <b>Optional</b> .
<b>ByVal</b>	Opcional. Indica que o argumento é passado <u>por valor</u> .
<b>ByRef</b>	Opcional. Indica que o argumento é passado <u>por referência</u> . <b>ByRef</b> é o padrão do Visual Basic.
<b>ParamArray</b>	Opcional. Utilizada apenas como o último argumento em <i>arglist</i> para indicar que o argumento final é uma matriz <b>Optional</b> de elementos <b>Variant</b> . A palavra-chave <b>ParamArray</b> permite que você forneça um número arbitrário de argumentos. Ela não pode ser utilizada com <b>ByVal</b> , <b>ByRef</b> ou <b>Optional</b> .
<i>varname</i>	Obrigatória. Nome da variável que representa o argumento; segue as convenções de nomenclatura padrão de variáveis.
<i>type</i>	Opcional. <u>Tipo de dados</u> do argumento passado ao procedimento;

pode ser **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (não-suportado atualmente), **Date**, **String** (somente de comprimento variável), **Object**, **Variant**. Se o parâmetro não for **Optional**, poderá ser especificado um tipo definido pelo usuário ou um tipo de objeto.

*defaultvalue*

Opcional. Qualquer constante ou expressão de constante. Válido somente para parâmetros **Optional**. Se o tipo for um **Object**, um valor padrão explícito somente poderá ser **Nothing**.

**Observação:** Toda instrução **Property Set** deve definir pelo menos um argumento do procedimento que ela define. Esse argumento (ou o último argumento se houver mais de um) contém a referência real ao objeto da propriedade quando o procedimento definido pela instrução **Property Set** é chamado. Ele é referido como *reference* na sintaxe anterior. Não pode ser **Optional**.

### Comentários

Se não forem especificados explicitamente utilizando-se **Public** ou **Private**, os procedimentos **Property** serão públicos como padrão. Se **Static** não for utilizado, o valor de variáveis locais não será preservado entre chamadas.

Todo o código executável deve estar nos procedimentos. Você não pode definir um procedimento **Property Set** dentro de outro procedimento **Property**, **Sub** ou **Function**.

A instrução **Exit Property** causa uma saída imediata de um procedimento **Property Set**. A execução do programa continua com a instrução que segue a instrução que chamou o procedimento **Property Set**. Pode aparecer qualquer número de instruções **Exit Property** em qualquer lugar de um procedimento **Property Set**.

Da mesma forma que um procedimento **Function** e **Property Get**, um procedimento **Property Set** é um procedimento separado que pode tomar argumentos, efetuar uma série de instruções e alterar o valor de seus argumentos. Entretanto, ao contrário de um procedimento **Function** e **Property Get**, os quais retornam um valor, você pode utilizar somente um procedimento **Property Set** do lado esquerdo de uma atribuição de referência de objeto (instrução **Set**).

## Instrução Public

Utilizada no nível de módulo para declarar variáveis públicas e alocar espaço de armazenamento.

### Sintaxe

**Public** [**WithEvents**] *varname*[(*subscripts*)] [**As** [**New**] *type*] [, [**WithEvents**] *varname*[(*subscripts*)] [**As** [**New**] *type*]] . . .

A sintaxe da instrução **Public** tem estas partes:

Parte	Descrição
<b>WithEvents</b>	Opcional. <u>Palavra-chave</u> que especifica que <i>varname</i> é uma <u>variável de objeto</u> utilizada para responder a eventos acionados por um <u>objeto ActiveX</u> . Válida somente em <u>módulos de classe</u> . Você pode declarar quantas variáveis individuais desejar utilizando <b>WithEvents</b> , mas não pode criar <u>matrizes</u> com <b>WithEvents</b> . Você não pode utilizar <b>New</b> com <b>WithEvents</b> .
<i>varname</i>	Obrigatória. Nome da variável; segue as convenções de nomenclatura padrão de variáveis.
<i>subscripts</i>	Opcional. Dimensões de uma variável de matriz; até 60 dimensões múltiplas podem ser declaradas. O <u>argumento subscripts</u> utiliza a sintaxe abaixo: [ <i>lower To</i> ] <i>upper</i> [, [ <i>lower To</i> ] <i>upper</i> ] . . . Quando não enunciado explicitamente em <i>lower</i> , o limite inferior de uma matriz é controlado pela instrução <b>Option Base</b> . O limite inferior será zero se não estiver presente uma

	instrução <b>Option Base</b> .
<b>New</b>	Opcional. Palavra-chave que permite a criação implícita de um objeto. Se você utilizar <b>New</b> quando declarar a variável de objeto, uma nova ocorrência do objeto será criada na primeira referência a ele; portanto, você não precisa utilizar a instrução <b>Set</b> para atribuir a referência ao objeto. A palavra-chave <b>New</b> não pode ser utilizada para declarar variáveis de <u>tipo de dados</u> intrínseco, não pode ser utilizada para declarar instâncias de objetos dependentes e não pode ser utilizada com <b>WithEvents</b> .
<i>type</i>	Opcional. Tipo de dados da variável; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não-suportado atualmente), <b>Date</b> , <b>String</b> , (para seqüências de caracteres de comprimento variável), <b>String * length</b> (para seqüências de caracteres de comprimento fixo), <b>Object</b> , <b>Variant</b> , um <u>tipo definido pelo usuário</u> ou um <u>tipo de objeto</u> . Utilize uma cláusula <b>As type</b> separada para cada variável que estiver sendo definida.

### Comentários

Variáveis declaradas utilizando-se a instrução **Public** estão disponíveis para todos os procedimentos de todos os módulos de todos os aplicativos, a menos que **Option Private Module** esteja em vigor, caso em que as variáveis serão públicas somente dentro do projeto no qual residem.

---

**Atenção** A instrução **Public** não pode ser utilizada em um módulo de classe para declarar uma variável de seqüência de caracteres de comprimento fixo.

---

Utilize a instrução **Public** para declarar o tipo de dados de uma variável. Por exemplo, a instrução abaixo declara uma variável como um **Integer**:

```
Public NúmeroDeEmpregados As Integer
```

Utilize também uma instrução **Public** para declarar o tipo de objeto de uma variável. A instrução abaixo declara uma variável para uma nova ocorrência de uma planilha.

```
Public X As New Worksheet
```

Se a palavra-chave **New** não for utilizada ao declarar uma variável de objeto, a variável que se refere ao objeto deverá ser atribuída a um objeto existente utilizando-se a instrução **Set** para que ela possa ser utilizada. Até que seja atribuído um objeto, a variável de objeto declarada terá o valor especial **Nothing**, que indica que não se refere a uma ocorrência específica de um objeto.

Você também pode utilizar a instrução **Public** com parênteses vazios para declarar uma matriz dinâmica. Depois de declarar uma matriz dinâmica, utilize a instrução **ReDim** dentro de um procedimento para definir o número de dimensões e elementos na matriz. Se você tentar declarar novamente uma dimensão de uma variável de matriz cujo tamanho tenha sido especificado explicitamente em uma instrução **Private**, **Public** ou **Dim**, um erro será gerado.

Se você não especificar um tipo de dados ou um tipo de objeto e não houver uma instrução **Default** no módulo, a variável será **Variant** como padrão.

Quando variáveis são inicializadas, uma variável numérica é inicializada como 0, uma seqüência de caracteres de comprimento variável é inicializada como uma seqüência de caracteres de comprimento zero ("") e uma seqüência de caracteres de comprimento fixo é preenchida com zeros. As variáveis **Variant** são inicializadas como **Empty**. Cada elemento de uma variável de tipo definido pelo usuário é inicializado como se fosse uma variável separada.

## Instrução ReDim

Utilizado no nível de procedimento para realocar o espaço de armazenamento para as variáveis de matriz dinâmica.

### Sintaxe

**ReDim** [**Preserve**] *varname(subscripts)* [**As type**] [, *varname(subscripts)* [**As type**]] . . .

A sintaxe da instrução **ReDim** tem estas partes:

Parte	Descrição
<b>Preserve</b>	Opcional. <u>Palavra-chave</u> utilizada para preservar os dados em uma <u>matriz</u> existente quando você altera o tamanho da última dimensão.
<i>varname</i>	Obrigatória. Nome da variável; segue as convenções de nomenclatura padrão de variável.
<i>subscripts</i>	Obrigatória. Dimensões de uma variável de matriz; até 60 dimensões múltiplas podem ser declaradas. O <u>argumento subscripts</u> utiliza a sintaxe abaixo: [ <i>lower To</i> ] <i>upper</i> [, [ <i>lower To</i> ] <i>upper</i> ] . . . Quando não declarado de forma explícita em <i>lower</i> , o limite inferior de uma matriz é controlado pela instrução <b>Option Base</b> . O limite inferior será zero se nenhuma instrução <b>Option Base</b> estiver presente.
<i>type</i>	Opcional. <u>Tipo de dados</u> da variável; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não-suportado atualmente), <b>Date</b> , <b>String</b> (para seqüências de caracteres de comprimento variável), <b>String</b> * <i>length</i> (para seqüências de caracteres de comprimento fixo), <b>Object</b> , <b>Variant</b> , um <u>tipo definido pelo usuário</u> ou um <u>tipo de objeto</u> . Utilize uma cláusula <b>As type</b> separada para cada variável sendo definida. Para um <b>Variant</b> contendo uma matriz, <i>type</i> descreve o tipo de cada elemento da matriz, mas não altera o <b>Variant</b> para algum outro tipo.

### Comentários

A instrução ReDim é utilizada para dimensionar ou redimensionar uma matriz dinâmica que já tenha sido declarada formalmente utilizando-se uma instrução **Private**, **Public** ou **Dim** com os parênteses vazios (sem subscritos de dimensão).

Você pode utilizar a instrução **ReDim** repetidamente para alterar o número de elementos e dimensões em uma matriz. Contudo, você não pode declarar uma matriz de um tipo dados e utilizar posteriormente **ReDim** para alterar a matriz para um outro tipo de dados, a menos que a matriz esteja contida em um **Variant**. Se a matriz estiver contida em um **Variant**, o tipo dos elementos podem ser alterados utilizando-se uma cláusula **As type**, a menos que você esteja utilizando a palavra-chave **Preserve**, que no caso, não permite alterações no tipo de dados.

Se você utilizar a palavra-chave **Preserve**, poderá redimensionar somente a última dimensão da matriz e não poderá alterar o número de dimensões. Por exemplo, se sua matriz possuir somente uma dimensão, você poderá redimensionar essa dimensão porque ela é a última e a única dimensão. Contudo, se sua matriz possuir duas ou mais dimensões, você poderá alterar somente o tamanho da última dimensão e ainda preservar o conteúdo da matriz. O exemplo abaixo mostra como você pode aumentar o tamanho da última dimensão de uma matriz dinâmica sem apagar os dados nela contidos.

```
ReDim X(10, 10, 10)
. . .
ReDim Preserve X(10, 10, 15)
```

De forma semelhante, quando você utiliza **Preserve**, pode alterar o tamanho da matriz somente alterando o limite superior; alterar o limite inferior gera um erro.

Se você tornar uma matriz menor do que seu tamanho original, os dados contidos nos elementos

eliminados serão perdidos. Se você passar uma matriz para um procedimento por referência, não poderá redimensionar a matriz dentro do procedimento.

Quando as variáveis forem inicializadas, uma variável numérica será inicializada para 0, uma seqüência de caracteres de comprimento variável será inicializada para uma seqüência de caracteres de comprimento zero ("") e uma seqüência de caracteres de comprimento fixo será preenchida com zeros. As variáveis **Variant** são inicializadas para **Empty**. Cada elemento de uma variável de tipo definido pelo usuário é inicializada como se fosse uma variável separada. Uma variável que faz referência a um objeto deve ser atribuída a um objeto existente utilizando-se a instrução **Set** antes que possa ser utilizada. Até que um objeto seja atribuído a ela, a variável de objeto declarada terá o valor especial **Nothing**, o que indica que ela não faz referência a uma ocorrência específica de um objeto.

---

**Atenção** A instrução **ReDim** age como uma instrução declarativa se a variável que ela declara não existir no nível de módulo ou no nível de procedimento. Se uma outra variável com o mesmo nome for criada posteriormente, mesmo em um escopo mais amplo, **ReDim** fará referência à variável posterior e não causará necessariamente um erro de compilação, mesmo que **Option Explicit** esteja em vigor. Para evitar tais conflitos, **ReDim** não deve ser utilizada como uma instrução declarativa, mas simplesmente para o redimensionamento de matrizes.

---

**Observação:** Para redimensionar uma matriz contida em um **Variant**, você deve declarar de forma explícita a variável **Variant** antes de tentar redimensionar sua matriz.

## Instrução Rem

Utilizada para incluir comentários explicativos em um programa.

### Sintaxe

**Rem** *comment*

Você também pode utilizar a sintaxe abaixo:

```
' comment
```

O argumento opcional *comment* é o texto de qualquer comentário que você deseje incluir. É necessário um espaço entre a palavra-chave **Rem** e o *comment*.

### Comentários

Se você utilizar números de linha ou rótulos de linha, poderá desviar de uma instrução GoTo ou GoSub para uma linha contendo uma instrução **Rem**. A execução continua com a primeira instrução executável que segue a instrução **Rem**. Se a palavra-chave **Rem** seguir outras instruções em uma linha, ela deve ser separada das instruções por dois-pontos (:).

Você pode utilizar um apóstrofo (') ao invés da palavra-chave **Rem**. Quando você utiliza um apóstrofo, os dois-pontos não são obrigatórios após outras instruções.

## Instrução Set

Atribui uma referência de objeto a uma variável ou propriedade.

### Sintaxe

**Set** *objectvar* = {[**New**] *objectexpression* | **Nothing**}

A sintaxe da instrução **Set** tem estas partes:

Parte	Descrição
<i>objectvar</i>	Obrigatória. Nome da variável ou propriedade; segue as convenções de nomenclatura padrão de variável.
<b>New</b>	Opcional. <b>New</b> é geralmente utilizado durante a declaração para possibilitar a criação implícita de objeto. Quando <b>New</b> for utilizada com <b>Set</b> , ela criará uma nova ocorrência da <u>classe</u> . Se <i>objectvar</i> contiver uma referência a um objeto,

	essa referência será liberada quando for atribuída uma nova. A <u>palavra-chave</u> <b>New</b> não pode ser utilizada para criar novas instâncias de qualquer <u>tipo de dados</u> intrínseco e não pode ser utilizada para criar objetos dependentes.
<i>objectexpression</i>	Obrigatória. <u>Expressão</u> que consiste no nome de um objeto, uma outra variável declarada do mesmo <u>tipo de objeto</u> ou uma função ou <u>método</u> que retorna um objeto do mesmo tipo.
<b>Nothing</b>	Opcional. Suspende a associação de <i>objectvar</i> com qualquer objeto específico. Atribuir <b>Nothing</b> a <i>objectvar</i> libera todos os recursos de sistema e memória associados com o objeto referenciado anteriormente quando nenhuma outra variável fizer referência a ele.

### Comentários

Para ser válido, *objectvar* deve ser um tipo de objeto consistente com o objeto sendo atribuído ao mesmo.

As instruções **Dim**, **Private**, **Public**, **ReDim** e **Static** declaram somente uma variável que faz referência a um objeto. Não é feita nenhuma referência a um objeto real até que você utilize a instrução **Set** para atribuir um objeto específico.

O exemplo abaixo ilustra como **Dim** é utilizado para declarar uma matriz com o tipo Form1. Nenhuma ocorrência de Form1 existe realmente. **Set** então atribui referências às novas instâncias de Form1 à variável FormuláriosFilho. Tal código pode ser utilizado para criar formulários filho em um aplicativo MDI.

```
Dim FormuláriosFilho(1 to 4) As Form1
Set FormuláriosFilho(1) = New Form1
Set FormuláriosFilho(2) = New Form1
Set FormuláriosFilho(3) = New Form1
Set FormuláriosFilho(4) = New Form1
```

Geralmente, quando você utiliza **Set** para atribuir uma referência de objeto a uma variável, nenhuma cópia do objeto é criada para essa variável. Em vez disso, uma referência ao objeto é criada. É possível fazer com que mais do que uma variável de objeto faça referência ao mesmo objeto. Como tais variáveis são referências ao objeto em vez de cópias do objeto, qualquer alteração no objeto é refletida em todas as variáveis que fazem referência a ele. Contudo, quando você utiliza a palavra-chave **New** na instrução **Set**, estará realmente criando uma ocorrência do objeto.

## Instrução Static

Utilizada no nível de procedimento para declarar variáveis e alocar espaço de armazenamento. As variáveis declaradas com a instrução **Static** retêm seus valores desde que o código esteja sendo executado.

### Sintaxe

**Static** *varname*[(*subscripts*)] [**As** [**New**] *type*] [, *varname*[(*subscripts*)] [**As** [**New**] *type*]] .  
..

A sintaxe da instrução **Static** tem estas partes:

Parte	Descrição
<i>varname</i>	Obrigatória. Nome da variável; segue as convenções de nomenclatura padrão de variável.
<i>subscripts</i>	Opcional. As dimensões de uma variável de <u>matriz</u> ; podem ser declaradas até 60 dimensões variadas. O <u>argumento</u> <i>subscripts</i> utiliza a sintaxe abaixo: [ <i>lower To</i> ] <i>upper</i> [, [ <i>lower To</i> ] <i>upper</i> ] . . . Quando não enunciado de forma explícita em <i>lower</i> , o limite

	inferior de uma matriz será controlado pela instrução <b>Option Base</b> . O limite inferior será zero se nenhuma instrução <b>Option Base</b> estiver presente.
<b>New</b>	Opcional. <u>Palavra-chave</u> que possibilita a criação implícita de um objeto. Se você utilizar <b>New</b> ao declarar a <u>variável de objeto</u> , uma nova ocorrência do objeto será criada na primeira referência a ele, assim não será necessário utilizar a instrução <b>Set</b> para atribuir a referência ao objeto. A palavra-chave <b>New</b> não pode ser utilizada para declarar variáveis de qualquer <u>tipo de dados</u> intrínseco e não pode ser utilizada para declarar instâncias de objetos dependentes.
<i>type</i>	Opcional. Tipo de dados da variável; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não-suportado atualmente), <b>Date</b> , <b>String</b> , (para seqüências de caracteres de comprimento variável), <b>String * length</b> (para seqüências de caracteres de comprimento fixo), <b>Object</b> , <b>Variant</b> , um <u>tipo definido pelo usuário</u> ou um <u>tipo de objeto</u> . Utilize uma cláusula <b>As type</b> separada para cada variável sendo definida.

### Comentários

Assim que o código do módulo estiver sendo executado, as variáveis declaradas com a instrução Static reterão seus valores até que o módulo seja redefinido ou reinicializado. Utilize a instrução **Static** em um procedimento não-estático para declarar as variáveis explicitamente, as quais são visíveis somente dentro do procedimento, mas cujo tempo de vida seja o mesmo que o módulo no qual o procedimento é definido.

Utilize uma instrução **Static** dentro de um procedimento para declarar o tipo de dados de uma variável que retenha seu valor entre as chamadas de procedimento. Por exemplo, a instrução abaixo declara uma matriz de tamanho fixo de números inteiros:

```
Static NúmeroDoEmpregado(200) As Integer
```

A instrução abaixo declara uma variável para uma nova ocorrência de uma planilha:

```
Static X As New Worksheet
```

Se a palavra-chave **New** não for utilizada ao declarar uma variável de objeto, deverá ser atribuído um objeto existente à variável que se refere ao objeto utilizando-se a instrução **Set** antes que ela possa ser utilizada. Até que um objeto seja atribuído a ela, a variável de objeto declarada conterá o valor especial **Nothing**, indicando que ela não se refere a uma ocorrência em particular de um objeto. Quando você utiliza a palavra-chave **New** na declaração, uma ocorrência do objeto é criada na primeira referência a ele.

Se você não especificar um tipo de dados ou um tipo objeto e se não houver uma instrução **Deftype** no módulo, a variável será **Variant** como padrão.

**Observação:** A instrução **Static** e a palavra-chave **Static** são semelhantes, mas são utilizadas para efeitos diferentes. Se você declarar um procedimento utilizando a palavra-chave **Static** (como em `Static Sub ContabVendas ()`), o espaço de armazenamento para todas as variáveis locais dentro de um procedimento será alocado uma vez e os valores das variáveis serão preservados durante todo o tempo em que o programa encontrar-se em execução. Para procedimentos não-estáticos, o espaço de armazenamento referente às variáveis é alocado cada vez que o procedimento é chamado e liberado quando o procedimento é finalizado. A instrução **Static** é utilizada para declarar variáveis específicas dentro de procedimentos não-estáticos para preservar seus valores durante o tempo em que o programa estiver sendo executado.

Quando as variáveis são inicializadas, uma variável numérica é inicializada para o valor 0, uma seqüência de caracteres de comprimento variável é inicializada para uma seqüência de caracteres de comprimento (""), e uma seqüência de caracteres de comprimento fixo é preenchida com zeros. As variáveis **Variant** são inicializadas para **Empty**. Cada elemento de uma variável de tipo definida pelo usuário é inicializado como se fosse uma variável separada.

**Observação:** Quando você utilizar as instruções **Static** dentro de um procedimento, coloque-as no início do procedimento com outras instruções declarativas como **Dim**.

## Instrução Sub

Declara o nome, os argumentos e o código que formam o corpo de um procedimento **Sub**.

### Sintaxe

```
[Private | Public] [Static] Sub name [(arglist)]
    [statements]
[Exit]
[statements]
End Sub
```

A sintaxe da instrução **Sub** tem estas partes:

Parte	Descrição
<b>Public</b>	Opcional. Indica que o procedimento <b>Sub</b> é acessível a todos os outros procedimentos em todos os <u>módulos</u> . Se utilizado em um módulo que contiver uma instrução <b>Option Private</b> , o procedimento não estará disponível fora do <u>projeto</u> .
<b>Private</b>	Opcional. Indica que o procedimento <b>Sub</b> é acessível somente a outros procedimentos no módulo onde ele é declarado.
<b>Static</b>	Opcional. Indica que as <u>variáveis</u> locais do procedimento <b>Sub</b> são preservadas entre as chamadas. O atributo <b>Static</b> não afeta as variáveis que são declaradas fora de <b>Sub</b> , mesmo que elas sejam utilizadas no procedimento.
<i>name</i>	Obrigatória. Nome de <b>Sub</b> ; segue as convenções de nomenclatura padrão de variável.
<i>arglist</i>	Opcional. Lista de variáveis representando os argumentos que são passados para o procedimento <b>Sub</b> quando ele é chamado. As variáveis são separadas por vírgulas.
<i>statements</i>	Opcional. Qualquer grupo de <u>instruções</u> a serem executadas dentro do procedimento <b>Sub</b> .

O argumento *arglist* possui a sintaxe e as partes abaixo:

```
[Optional] [ByVal | ByRef] [ParamArray] varname( ) [As type] [= defaultvalue]
```

Parte	Descrição
<b>Optional</b>	Opcional. <u>Palavra-chave</u> indicando que um argumento não é obrigatório. Se utilizada, todos os argumentos subseqüentes em <i>arglist</i> deverão também ser opcionais e declarados utilizando-se a palavra-chave <b>Optional</b> . <b>Optional</b> não poderá ser utilizado para qualquer argumento se <b>ParamArray</b> for utilizado.
<b>ByVal</b>	Opcional. Indica que o argumento é passado <u>por valor</u> .
<b>ByRef</b>	Opcional. Indica que o argumento é passado <u>por referência</u> . <b>ByRef</b> representa o padrão no Visual Basic.
<b>ParamArray</b>	Opcional. Utilizado somente como último argumento em <i>arglist</i> para indicar que o argumento final é uma <u>matriz</u> <b>Optional</b> de elementos <b>Variant</b> . A palavra-chave <b>ParamArray</b> permite-lhe fornecer um número arbitrário de argumentos. <b>ParamArray</b> não pode ser utilizado com <b>ByVal</b> , <b>ByRef</b> ou <b>Optional</b> .
<i>varname</i>	Obrigatória. Nome da variável representando o argumento; segue as convenções de nomenclatura padrão de variável.
<i>type</i>	Opcional. <u>Tipo de dados</u> do argumento passado ao procedimento; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não-suportado atualmente), <b>Date</b> , <b>String</b> (somente de comprimento variável), <b>Object</b> , <b>Variant</b> . Se o parâmetro não for <b>Optional</b> , um <u>tipo</u>

*defaultvalue* definido pelo usuário ou um tipo de objeto também pode ser especificado.  
Opcional. Qualquer constante ou expressão de constante. Válida somente para parâmetros **Optional**. Se o tipo for um **Object**, o valor padrão explícito poderá ser apenas **Nothing**.

### Comentários

Se não forem especificados de forma explícita utilizando **Public** ou **Private**, os procedimentos **Sub** serão públicos como padrão. Se não for utilizado **Static**, o valor das variáveis locais não será preservado entre as chamadas.

---

**Atenção** Os procedimentos **Sub** podem ser recursivos; ou seja, eles podem chamar a si mesmos para efetuar uma certa tarefa. Contudo, a recursão pode levar a um estouro de pilha. A palavra-chave **Static** geralmente não é utilizada com os procedimentos **Sub** recursivos.

---

Todos os códigos executáveis devem estar contidos em procedimentos. Você não pode definir um procedimento **Sub** dentro de um outro procedimento **Sub**, **Function** ou **Property**.

As palavras-chave **Exit Sub** causam uma saída imediata de um procedimento **Sub**. A execução do programa continua com a instrução que segue a instrução que chamou esse procedimento. É possível aparecer um número variado de instruções **Exit Sub** em qualquer lugar de um procedimento **Sub**.

Como um procedimento **Function**, um procedimento **Sub** é um procedimento separado que pode tomar os argumentos, efetuar uma série de instruções e alterar o valor de seus argumentos. Contudo, ao contrário do procedimento **Function**, que retorna um valor, um procedimento **Sub** não pode ser utilizado em uma expressão.

Você chama um procedimento **Sub** utilizando o nome do procedimento seguido pela lista de argumentos. Consulte a instrução **Call** para obter Especificidades sobre como chamar os procedimentos **Sub**.

As variáveis utilizadas nos procedimentos **Sub** incorrem em duas categorias: aquelas que são declaradas de forma explícita dentro de um procedimento e aquelas que não o são. As variáveis que são declaradas de forma explícita em um procedimento (utilizando-se **Dim** ou equivalente) são sempre locais no que se refere ao procedimento. As variáveis que são utilizadas mas não são declaradas de forma explícita em um procedimento também são locais, a menos que elas sejam declaradas de forma explícita em algum nível superior fora do procedimento.

---

**Atenção** Um procedimento pode utilizar uma variável que não é declarada de forma explícita no procedimento, mas poderá ocorrer um conflito de nomeação se algo que você tenha definido no nível de módulo possuir o mesmo nome. Se seu procedimento fizer referência a uma variável não-declarada que possui o mesmo nome que um outro procedimento, constante ou variável, será assumido que seu procedimento está se referindo àquele nome no nível de módulo. Para evitar esse tipo de conflito, declare as variáveis de forma explícita. Você pode utilizar uma instrução **Option Explicit** para forçar a declaração explícita das variáveis.

---

**Observação:** Você não pode utilizar **GoSub**, **GoTo** ou **Return** para entrar ou sair de um procedimento **Sub**.

## Instrução Type

Utilizada no nível de módulo para definir um tipo de dados definido pelo usuário contendo um ou mais elementos.

### Sintaxe

```
[Private | Public] Type varname
    elementname([subscripts]) type
    [elementname([subscripts]) type]
    ...
End Type
```

A sintaxe da instrução **Type** tem estas partes:

Parte	Descrição
<b>Public</b>	Opcional. Utilizada para declarar <u>tipos definidos pelo usuário</u> que estão disponíveis para todos os <u>procedimentos</u> em todos os <u>módulos</u> e em todos os <u>projetos</u> .
<b>Private</b>	Opcional. Utilizada para declarar tipos definidos pelo usuário que estão disponíveis somente dentro do módulo onde ocorre a <u>declaração</u> .
<i>varname</i>	Obrigatória. Nome do tipo definido pelo usuário; segue as convenções de nomenclatura padrão de <u>variáveis</u> .
<i>element name</i>	Obrigatória. Nome de um elemento de tipo definido pelo usuário. Os nomes de elemento também seguem as convenções de nomenclatura padrão de variável, exceto que <u>palavras-chave</u> podem ser utilizadas.
<i>subscripts</i>	Opcional. Dimensões de um elemento de <u>matriz</u> . Utilize somente parênteses quando declarar uma matriz cujo tamanho pode ser alterado. O <u>argumento subscripts</u> utiliza a sintaxe abaixo: [ <i>lower To</i> ] <i>upper</i> [, [ <i>lower To</i> ] <i>upper</i> ] . . . Quando não for declarado de forma explícita em <i>lower</i> , o limite inferior de uma matriz será controlado pela instrução <b>Option Base</b> . O limite inferior será zero se nenhuma instrução <b>Option Base</b> estiver presente.
<i>type</i>	Obrigatória. Tipo de dados do elemento; pode ser <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Decimal</b> (não-suportado atualmente), <b>Date</b> , <b>String</b> (para seqüências de caracteres de comprimento variável), <b>String</b> * <i>comprimento</i> (para seqüências de caracteres de comprimento fixo), <b>Object</b> , <b>Variant</b> , um outro tipo definido pelo usuário ou um <u>tipo de objeto</u> .

### Comentários

A instrução **Type** pode ser utilizada somente no nível de módulo. Assim que você tiver declarado um tipo definido pelo usuário utilizando a instrução **Type**, será possível declarar uma variável daquele tipo em qualquer lugar dentro do escopo da declaração. Utilize **Dim**, **Private**, **Public**, **ReDim** ou **Static** para declarar uma variável do tipo definido pelo usuário.

Nos módulos padrão, os tipos definidos pelo usuário são públicos como padrão. Essa visibilidade pode ser alterada utilizando-se a palavra-chave **Private**. Nos módulos de classe, contudo, os tipos definidos pelo usuário podem somente ser privados e a visibilidade não pode ser alterada utilizando-se a palavra-chave **Public**.

Não são permitidos números de linha e rótulos de linha nos blocos **Type...End Type**.

Os tipos definidos pelo usuário são geralmente utilizados com registros de dados, que freqüentemente consistem em um número de elementos relacionados de diferentes tipos de dados.

O exemplo abaixo mostra a utilização de matrizes de tamanho fixo em um tipo definido pelo usuário:

```
Type StateData
    CódigoDaCidade (1 To 100) As Integer ' Declara uma matriz estática.
```

```
Município As String * 30
End Type
```

```
Dim SãoPaulo(1 To 100) As StateData
```

No exemplo anterior, `StateData` inclui a matriz estática `CódigoDaCidade` e o registro `SãoPaulo` possui a mesma estrutura que `StateData`.

Quando você declara uma matriz de tamanho fixo dentro de um tipo definido pelo usuário, suas dimensões devem ser declaradas com literais numéricos ou constantes em vez de variáveis.

A definição da instrução **Option Base** determina o limite inferior para as matrizes dentro dos tipos definidos pelo usuário.

## Função UBound

Retorna um **Long** contendo o maior subscrito disponível para a dimensão indicada de uma matriz.

### Sintaxe

**UBound**(*arrayname*[, *dimension*])

A sintaxe da função **UBound** tem estas partes:

Parte	Descrição
<i>arrayname</i>	Obrigatória. Nome da <u>variável</u> de matriz; segue as convenções de nomenclatura padrão de variável.
<i>dimension</i>	Opcional; <b>Variant (Long)</b> . Número inteiro indicando o limite superior da dimensão que é retornado. Utilize 1 para a primeira dimensão, 2 para a segunda e assim por diante. Se <i>dimension</i> for omitida, assume-se o valor 1.

### Comentários

A função **UBound** é utilizada com a função **LBound** para determinar o tamanho de uma matriz. Utilize a função **LBound** para encontrar o limite inferior da dimensão de uma matriz.

**UBound** retorna os seguintes valores para uma matriz com as dimensões abaixo:

```
Dim A(1 To 100, 0 To 3, -3 To 4)
```

Instrução	Valor de retorno
UBound(A, 1)	100
UBound(A, 2)	3
UBound(A, 3)	4

### Exemplo da função Array

Este exemplo utiliza a função **Array** para retornar uma **Variant** que contém uma matriz.

```
Dim MyWeek, MyDay
MyWeek = Array("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
' Os valores de retorno assumem o limite inferior definido como 1
' (utilizando a instrução Option Base).
MyDay = MyWeek(2) ' MyDay contém "Tue".
MyDay = MyWeek(4) ' MyDay contém "Thu".
```

### Exemplo da instrução Const

Este exemplo utiliza a instrução **Const** para declarar constantes para uso em lugar de valores literais. As constantes **Public** ocorrem em uma seção Geral de um módulo padrão, em vez de em um módulo de classe. As constantes **Private** podem aparecer na seção Geral de qualquer tipo de módulo.

```
' As constantes são Private como padrão.
Const MyVar = 459

' Declara a constante Public.
Public Const MyString = "HELP"

' Declara a constante Integer Private.
Private Const MyInt As Integer = 5

' Declara múltiplas constantes na mesma linha.
Const MyStr = "Hello", MyDouble As Double = 3.4567
```

### Exemplo da função CreateObject

Este exemplo utiliza a função **CreateObject** para definir uma referência (xlApp) ao Microsoft Excel. Ele utiliza a referência para acessar a propriedade **Visible** do Microsoft Excel e, em seguida, utiliza o método **Quit** do Microsoft Excel para fechá-lo. Por fim, a própria referência é liberada.

```
Dim xlApp As Object ' Declara a variável para conter a referência.

Set xlApp = CreateObject("excel.application")
' Você pode ter que definir a propriedade Visible como True
' se quiser ver o aplicativo.
xlApp.Visible = True
' Usa xlApp para acessar outros objetos do
' Microsoft Excel.
xlApp.Quit ' Quando terminar, use o método Quit para fechar
Set xlApp = Nothing ' o aplicativo e, em seguida, libere a referência.
```

### Exemplo da instrução Declare

Este exemplo mostra como a instrução **Declare** é utilizada em nível de módulo de um módulo padrão para declarar uma referência a um procedimento externo em uma biblioteca de vinculação dinâmica (DLL) ou no recurso de código do Macintosh. Você pode inserir as instruções **Declare** em módulos de classe se as instruções **Declare** forem **Private**.

```
' No Microsoft Windows (de 16 bits):
Declare Sub MessageBeep Lib "User" (ByVal N As Integer)
' Suponha que SomeBeep é um alias para o nome do procedimento.
Declare Sub MessageBeep Lib "User" Alias "SomeBeep" (ByVal N As Integer)
' Usa um ordinal na cláusula Alias para chamar GetWinFlags.
Declare Function GetWinFlags Lib "Kernel" Alias "#132" () As Long

' Nos sistemas Microsoft Windows de 32 bits, especifique a biblioteca
USER32.DLL,
' em vez de USER.DLL. Você pode utilizar a compilação
' condicional para gravar um código que possa ser
' executado tanto no Win32 como no Win16.
#If Win32 Then
    Declare Sub MessageBeep Lib "User32" (ByVal N As Long)
#Else
    Declare Sub MessageBeep Lib "User" (ByVal N As Integer)
#End If

' No Macintosh:
Declare Sub MessageAlert Lib "MyHd:MyAlert" Alias "MyAlert" (ByVal N _
As Integer)
```

```
' Utiliza um recurso de código na cláusula Alias.
Declare Sub MessageAlert Lib "MyHd:MyAlert" Alias "XTST$MyAlert" _
(ByVal N As Integer)

' Se o especificador do tipo de recurso de código tiver
' somente 3 caracteres, certifique-se de deixar um
' espaço em branco onde apareceria normalmente o
' caractere final.
Declare Sub MessageAlert Lib "MyHd:AnAlert" Alias "COD $AnAlert" _
(ByVal N As Integer)
```

### Exemplo de instruções Deftype

Este exemplo mostra vários usos das instruções **Deftype** para definir os tipos de dados padrão de variáveis e procedimentos de função cujos nomes iniciam com caracteres especificados. O tipo de dados padrão pode ser sobrescrito somente por atribuição explícita utilizando-se a instrução **Dim**. As instruções **Deftype** somente podem ser utilizadas em nível de módulo (ou seja, fora dos procedimentos).

```
' Os nomes de variável que começam com as letras entre A e K têm como
padrão Integer.
DefInt A-K
' Os nomes de variável que começam com as letras entre L e Z têm como
padrão String.
DefStr L-Z
CalcVar = 4 ' Inicializa Integer.
StringVar = "Hello, there" ' Inicializa String.
AnyVar = "Hello" ' Provoca o erro "Type Mismatch".
Dim Calc As Double ' Define explicitamente o tipo como Double.
Calc = 2.3455 ' Atribui um Double.

' As instruções Deftype também se aplicam a procedimentos de função.
CalcNum = ATestFunction(4) ' Chama a função definida pelo usuário.
' Definição do procedimento de função ATestFunction.
Function ATestFunction(INumber)
    ATestFunction = INumber * 2 ' O valor de retorno é um inteiro.
End Function
```

### Exemplo da instrução Dim

Este exemplo mostra vários usos da instrução **Dim** para declarar variáveis. Ele também mostra a instrução **Dim** sendo utilizada para declarar matrizes. O limite inferior padrão dos subscritos de matriz é 0 e pode ser sobrescrito em nível de módulo utilizando-se a instrução **Option Base**.

```
' AnyValue e MyValue são declaradas como Variant como padrão com o valores
definidos como Empty.
Dim AnyValue, MyValue

' Declara explicitamente uma variável do tipo Integer.
Dim Número As Integer

' Múltiplas declarações em uma única linha. AnotherVar é do tipo Variant
' porque o seu tipo está omitido.
Dim AnotherVar, Option As Boolean, BirthDate As Date

' DayArray é uma matriz de variantes com 51 elementos indexados, de
' 0 a 50, pressupondo-se que Option Base esteja definida como 0 (padrão)
para o
' módulo atual.
Dim DayArray(50)

' Matrix é uma matriz bidimensional de inteiros.
```

```
Dim Matrix(3, 4) As Integer
' MyMatrix é uma matriz tridimensional de duplos com limites explícitos.
Dim MyMatrix(1 To 5, 4 To 9, 3 To 5) As Double
' BirthDay é uma matriz de datas com índices de 1 a 10.
Dim BirthDay(1 To 10) As Date
' MyArray é uma matriz dinâmica de variantes.
Dim MyArray()
```

### Exemplo da instrução Enum

O exemplo abaixo mostra a instrução **Enum** usada para definir uma coleção de constantes nomeadas. Neste caso, as constantes são cores que você poderia escolher para designar formulários de entrada de dados para um banco de dados.

```
Public Enum InterfaceColors
    icMistyRose = &HE1E4FF&
    icSlateGray = &H908070&
    icDodgerBlue = &HFF901E&
    icDeepSkyBlue = &HFFBF00&
    icSpringGreen = &H7FFF00&
    icForestGreen = &H228B22&
    icGoldenrod = &H20A5DA&
    icFirebrick = &H2222B2&
End Enum
```

### Exemplo da instrução Erase

Este exemplo utiliza a instrução **Erase** para reinicializar os elementos de matrizes de tamanho fixo e desalocar espaço de armazenamento de matrizes dinâmicas.

```
' Declara variáveis de matriz.
Dim NumArray(10) As Integer ' Matriz Integer.
Dim StrVarArray(10) As String ' Matriz de seqüência de caracteres variável.
Dim Str|FixArray(10) As String * 10 ' Matriz de seqüência de caracteres fixa.
Dim VarArray(10) As Variant ' Matriz de variante.
Dim DynamicArray() As Integer ' Matriz Dinâmica.
ReDim DynamicArray(10) ' Alocar espaço de armazenamento.
Erase NumArray ' Cada elemento definido como 0.
Erase StrVarArray ' Cada elemento definido como seqüência de caracteres de Length
    ' zero ("").
Erase Str|FixArray ' Cada elemento definido como 0.
Erase VarArray ' Cada elemento definido como Empty.
Erase DynamicArray ' Liberar a memória utilizada pela matriz.
```

### Exemplo da instrução Event

O exemplo abaixo utiliza eventos para contar segundos durante uma demonstração da mais rápida corrida de 100 metros rasos. O código ilustra todos os métodos relacionados com eventos, propriedades e instruções, incluindo a instrução **Event**.

A classe que produz um evento da origem do evento e a classe que implementa o evento são os coletores. Uma origem de evento pode ter múltiplos coletores para os eventos que ela gera. Quando a classe provoca o evento, ele é disparado em todas as classes que escolheram eventos coletores para aquela ocorrência do objeto.

O exemplo também utiliza um formulário (Form1) com um botão (Command1), um rótulo (Label1) e duas caixas de texto (Text1 e Text2). Quando você clica no botão, a primeira caixa de texto exibe "A partir de agora" e a segunda inicia a contagem de segundos. Quando o tempo completo (9,84 segundos) decorre, a primeira caixa de texto exibe "Até agora" e a segunda exibe "9,84"

O código para Form1 especifica os estados inicial e final do formulário. Ele também contém o código executado quando os eventos são provocados.

```
Option Explicit

Private WithEvents mText As TimerState

Private Sub Command1_Click()
    Text1.Text = "Frow now"
    Text1.Refresh
    Text2.Text = "0"
    Text2.Refresh
    Call mText.TimerTask(9.84)
End Sub

Private Sub Form_Load()
    Command1.Caption = "Click to Start Timer"
    Text1.Text = ""
    Text2.Text = ""
    Label1.Caption = "The fastest 100 meter run took this long:"
    Set mText = New TimerState
End Sub

Private Sub mText_ChangeText()
    Text1.Text = "Até agora"
    Text2.Text = "9.84"
End Sub

Private Sub mText_UpdateTime(ByVal dblJump As Double)
    Text2.Text = Str(Format(dblJump, "0"))
    DoEvents
End Sub
```

O código restante está em um módulo de classe chamado TimerState. As instruções **Event** declaram os procedimentos iniciados quando eventos são provocados.

```
Option Explicit
Public Event UpdateTime(ByVal dblJump As Double)
Public Event ChangeText()

Public Sub TimerTask(ByVal Duration As Double)
    Dim dblStart As Double
    Dim dblSecond As Double
    Dim dblSoFar As Double
    dblStart = Timer
    dblSoFar = dblStart

    Do While Timer < dblStart + Duration
```

```

    If Timer - dblSoFar >= 1 Then
        dblSoFar = dblSoFar + 1
        RaiseEvent UpdateTime (Timer - dblStart)
    End If
Loop

RaiseEvent ChangeText

End Sub

```

### Exemplo da instrução Function

Este exemplo utiliza a instrução **Function** para declarar o nome, argumentos e código que formam o corpo de um procedimento **Function**. O último exemplo utiliza argumentos **Optional** inicializados.

```

' A seguinte função definida pelo usuário retorna a raiz quadrada do
' argumento passado a ela.
Function CalculateSquareRoot (NumberArg As Double) As Double
    If NumberArg < 0 Then ' Avalia o argumento.
        Exit Function ' Sai para chamar o procedimento.
    Else
        CalculateSquareRoot = Sqr(NumberArg) ' Retorna a raiz quadrada.
    End If
End Function

```

O uso da palavra-chave **ParamArray** possibilita que uma função aceite um número variável de argumentos. Na definição a seguir, **FirstArg** é passado pelo valor.

```

Function CalcSum (ByVal FirstArg As Integer, ParamArray OtherArgs ())
Dim ReturnValue
' Se a função for chamada da seguinte maneira:
ReturnValue = CalcSum(4, 3, 2, 1)
' As variáveis locais são atribuídas aos seguintes valores: FirstArg = 4,
' OtherArgs(1) = 3, OtherArgs(2) = 2, e assim por diante, pressupondo-se o
limite inferior padrão
' para matrizes = 1.

```

Os argumentos **Optional** agora podem ter valores e tipos padrão diferentes de **Variant**.

```

' Se os argumentos de uma função forem definidos da seguinte maneira:
Function MyFunc(MyStr As String, Optional MyArg1 As Integer = 5, Optional
MyArg2 = "Dolly")
Dim RetVal
' A função pode ser chamada da seguinte maneira:
RetVal = MyFunc("Hello", 2, "World") ' Todos os 3 argumentos
fornecidos.
RetVal = MyFunc("Test", , 5) ' Segundo argumento omitido.
' Argumentos um e três utilizando argumentos nomeados.
RetVal = MyFunc(MyStr:="Hello ", MyArg1:=7)

```

**Exemplo da função GetObject**

Este exemplo utiliza a função **GetObject** para obter uma referência a uma planilha específica do Microsoft Excel (*MyXL*). Ele utiliza a propriedade **Application** da planilha para tornar o Microsoft Excel visível, fechá-lo etc. A primeira chamada de **GetObject** provoca um erro se o Microsoft Excel ainda não estiver sendo executado. No exemplo, o erro faz com que o sinalizador *ExcelNãoEstavaSendoExecutado* seja definido como True. A segunda chamada de **GetObject** especifica um arquivo a ser aberto. Se o Microsoft Excel ainda não estiver sendo executado, a segunda chamada o iniciará e retornará uma referência à planilha representada pelo arquivo especificado. O arquivo, *meuteste.xls* no exemplo, deve existir no local especificado; caso contrário, será gerado o erro de automação do Visual Basic. Depois, o código de exemplo torna visíveis tanto o Microsoft Excel como a janela que contém a planilha especificada. Por fim, se não havia versão anterior do Microsoft Excel sendo executada, o código utiliza o método **Quit** do objeto **Application** para fechar o Microsoft Excel. Se o aplicativo já estava sendo executado, não é feita nenhuma tentativa de fechá-lo. A própria referência é liberada sendo definida como **Nothing**.

```
' Declara necessário as rotinas API :
Declare Function FindWindow Lib "user32" Alias _
"FindWindowA" (ByVal lpClassName as String, _
                ByVal lpWindowName As Long) As Long

Declare Function SendMessage Lib "user32" Alias _
"SendMessageA" (ByVal hWnd as Long, ByVal wParam as Long _
                ByVal lParam As Long) As Long

Sub GetExcel()
Dim MyXL As Object ' Variável para conter a referência ao Microsoft Excel.
Dim ExcelWasNotRunning As Boolean ' Sinalizador para liberação final.

' Testa para ver se já há uma cópia do Microsoft Excel sendo executada.
On Error Resume Next ' Adia a interceptação do erro.
' Função GetObject chamada sem que o primeiro argumento
' retorne uma referência a uma instância do aplicativo. ' Se o aplicativo
não estiver sendo executado, ocorrerá ' um erro. Observe a vírgula
utilizada como marcador para o primeiro argumento
Set MyXL = Getobject(, "Excel.Application")
If Err.Number <> 0 Then ExcelWasNotRunning = True
Err.Clear ' Limpa o objeto Err se tiver ocorrido o erro.
' Verifique para o Microsoft Excel. Se o Microsoft Excel está sendo
executado,
' insira dentro da tabela Running Object.
    DetectExcel

' Define a variável de objeto para fazer referência ao arquivo que você
deseja ver.
Set MyXL = Getobject("c:\vb4\MYTEST.XLS")

' Mostra o Microsoft Excel através da sua propriedade Application. Em
seguida mostra a janela real que contém o arquivo que utiliza a coleção
Windows
' da referência do objeto MyXL.
MyXL.Application.Visible = True
MyXL.Parent.Windows(1).Visible = True
    ' Faz a manipulação do seu
    ' arquivo aqui.
    ' ...
' Se esta cópia do Microsoft Excel não estava sendo executada quando você
iniciou, feche-a utilizando o método Quit da propriedade Application.
' Observe que, quando você tentar sair do Microsoft Excel, a barra de
título do Microsoft Excel
```

```

' piscará e o Microsoft Excel exibirá uma mensagem perguntando-lhe se
deseja
' salvar os arquivos carregados.
If ExcelWasNotRunning = True Then MyXL.Application.Quit
End If
Set MyXL = Nothing' Libera a referência para o
' aplicativo e planilha.
End Sub

Sub DetectExcel()
' O procedimento detecta que o Excel está sendo executado e registra-o.
Const WM_USER = 1024
Dim hWnd As Long
' Se o Excel está sendo executado esta chamada API retorna seu
identificador.
hWnd = FindWindow("XLMAIN", 0)
If hWnd = 0 Then ' 0 significa que o Excel não está executando.
Exit Sub
Else
' O Excel está sendo executado, então use o SendMessage API
' função para inserir a tabela Running Object .
SendMessage hWnd, WM_USER + 18, 0, 0
End If
End Sub

```

### Exemplo da instrução Implements

O exemplo abaixo mostra como usar a instrução **Implements** para fazer um conjunto de declarações disponíveis para múltiplas classes. Compartilhando as declarações através da instrução **Implements**, nenhuma classe precisa fazer qualquer declaração.

Suponhamos que existam dois formulários. O formulário Selector tem dois botões, Customer Data e Supplier Data. Para inserir informações de nome e endereço para um cliente ou um fornecedor, o usuário clica no botão Customer ou no botão Supplier no formulário Selector e, em seguida, insere o nome e o endereço usando o formulário Data Entry. Este formulário tem dois campos, Name e Address.

O código abaixo para as declarações compartilhadas está em uma classe chamada PersonalData:

```

Public Name As String
Public Address As String

```

O código que suporta os dados do cliente está em um módulo de classe chamado Customer:

```

Implements PersonalData
Private Property Get PersonalData_Address() As String
PersonalData_Address = "CustomerAddress"
End Property

Private Property Let PersonalData_Address(ByVal RHS As String)
'
End Property

Private Property Let PersonalData_Name(ByVal RHS As String)
'
End Property

Private Property Get PersonalData_Name() As String
PersonalData_Name = "CustomerName"
End Property

```

O código que suporta os dados do fornecedor está em um módulo de classe chamado Supplier:

**Implements** PersonalData

```
Private Property Get PersonalData_Address() As String
PersonalData_Address = "SupplierAddress"
End Property

Private Property Let PersonalData_Address(ByVal RHS As String)
'
End Property

Private Property Let PersonalData_Name(ByVal RHS As String)
'
End Property

Private Property Get PersonalData_Name() As String
PersonalData_Name = "SupplierName"
End Property
```

O código abaixo suporta o formulário Selector:

```
Private cust As New Customer
Private sup As New Supplier

Private Sub Command1_Click()
Dim frm2 As New Form2
    Set frm2.PD = cust
    frm2.Show 1
End Sub

Private Sub Command2_Click()
Dim frm2 As New Form2
    Set frm2.PD = sup
    frm2.Show 1
End Sub
```

O código abaixo suporta o formulário Data Entry:

```
Private m_pd As PersonalData
Private Sub Form_Load()
    With m_pd
        Text1 = .Name
        Text2 = .Address
    End With
End Sub
Public Property Set PD(Data As PersonalData)
    Set m_pd = Data
End Property
```

### Exemplo da função LBound

Este exemplo utiliza a função **LBound** para determinar o menor subscrito disponível para a dimensão indicada de uma matriz. Utiliza a instrução **Option Base** para sobrescrever o valor 0 do subscrito da matriz base padrão.

```
Dim Lower
Dim MyArray(1 To 10, 5 To 15, 10 To 20) ' Declara as variáveis de matriz.
Dim AnyArray(10)
Lower = Lbound(MyMatrix, 1) ' Retorna 1.
Lower = Lbound(MyMatrix, 3) ' Retorna 10.
Lower = Lbound(AnyArray) ' Retorna 0 ou 1, dependendo da
    ' definição de Option Base.
```

### Exemplo da instrução Let

Este exemplo atribui os valores de expressões a variáveis utilizando a instrução **Let** explícita e implícita.

```
Dim MyStr, MyInt
' As atribuições de variável a seguir utilizam a instrução Let.
Let MyStr = "Hello World"
Let MyInt = 5
```

As instruções a seguir são as mesmas, sem a instrução **Let**.

```
Dim MyStr, MyInt
MyStr = "Hello World"
MyInt = 5
```

### Exemplo da instrução Option Base

Este exemplo utiliza a instrução **Option Base** para sobrescrever o valor 0 do subscrito da matriz base padrão. A função **LBound** retorna o menor subscrito disponível para a dimensão indicada de uma matriz. A instrução **Option Base** é utilizada somente em nível de módulo.

```
Option base 1 ' Define os subscritos da matriz padrão como 1.

Dim Lower
Dim MyArray(20), TwoDArray(3, 4) ' Declara as variáveis de matriz.
Dim ZeroArray(0 To 5) ' Sobrescreve o subscrito base padrão.
' Utiliza a função LBound para testar os limites Inferiores da matrizes.
Lower = LBound(MyArray) ' Retorna 1.
Lower = LBound(TwoDArray, 2) ' Retorna 1.
Lower = LBound(ZeroArray) ' Retorna 0.
```

### Exemplo da instrução Option Compare

Este exemplo utiliza a instrução **Option Compare** para definir o método de comparação de seqüências padrão. A instrução **Option Compare** é utilizada somente no nível de módulo.

```
' Define o método de comparação de seqüências de caracteres como Binary.
Option compare Binary ' Ou seja, "AAA" é menos que "aaa"
' Define o método de comparação de seqüências como Text.
Option compare Text ' Ou seja, "AAA" é igual a "aaa".
```

**Exemplo da instrução Option Explicit**

Este exemplo utiliza a instrução **Option Explicit** para forçá-lo a declarar explicitamente todas as variáveis. A tentativa de utilizar uma variável não declarada provoca um erro em tempo de compilação. A instrução **Option Explicit** é utilizada somente no nível de módulo.

```
Option explicit ' Força a declaração de variável explícita.
Dim MyVar ' Declara a variável.
MyInt = 10 ' Variável não declarada gera erro.
MyVar = 10 ' Variável declarada não gera erro.
```

**Exemplo da instrução Option Private**

Este exemplo demonstra a instrução **Option Private**, que é utilizada no nível de módulo para indicar que o módulo inteiro é privado. Com **Option Private Module**, os símbolos de nível de módulo não declarados **Private** continuam disponíveis para outros módulos no projeto, mas não para outros projetos ou aplicativos.

```
Option private Module ' Indica que o módulo é privado.
```

**Exemplo da instrução Private**

Este exemplo mostra a instrução **Private** sendo utilizada no nível de módulo para declarar variáveis como privadas, ou seja, elas estão disponíveis somente para o módulo em que são declaradas.

```
Private Número As Integer ' Variável Integer privada.
Private NameArray(1 To 5) As String ' Variável de matriz privada.
' Múltiplas declarações, duas Variants e um Integer, todos Private.
Private MyVar, YourVar, ThisVar As Integer
```

**Exemplo da instrução Property Get**

Este exemplo utiliza a instrução **Property Get** para definir um procedimento de propriedade que obtém o valor de uma propriedade. A propriedade identifica, como uma seqüência de caracteres, a cor atual de uma caneta em um pacote de desenho.

```
Dim CurrentColor As Integer
Const BLACK = 0, RED = 1, GREEN = 2, BLUE = 3

' Retorna a cor atual da caneta como uma seqüência de caracteres.
Property Get PenColor() As String
    Select Case CurrentColor
        Case RED
            PenColor = "Red"
        Case GREEN
            PenColor = "Green"
        Case BLUE
            PenColor = "Blue"
    End Select
End Property

' O código a seguir obtém a cor da caneta
' que chama o procedimento Property Get.
ColorName = PenColor
```

### Exemplo da instrução Property Let

Este exemplo utiliza a instrução **Property Let** para definir um procedimento que atribui um valor a uma propriedade. A propriedade identifica a cor da caneta para um pacote de desenho.

```
Dim CurrentColor As Integer
Const BLACK = 0, RED = 1, GREEN = 2, BLUE = 3

' Define a propriedade da cor da caneta de um pacote de desenho.
' A variável em nível de módulo CurrentColor é definida como um valor
numérico que identifica a cor utilizada no desenho.
Property Let PenColor(ColorName As String)
    Select Case ColorName ' Verifica a seqüência de caracteres do nome da
cor.
        Case "Red"
            CurrentColor = RED ' Atribui o valor a Red.
        Case "Green"
            CurrentColor = GREEN ' Atribui o valor a Green.
        Case "Blue"
            CurrentColor = BLUE ' Atribui o valor a Blue.
        Case Else
            CurrentColor = BLACK ' Atribui valor padrão.
    End Select
End Property

' O código a seguir define a propriedade PenColor de um pacote de desenho
' chamando o procedimento Property Let.
PenColor = "Red"
```

### Exemplo da instrução Property Set

Este exemplo utiliza a instrução **Property Set** para definir um procedimento de propriedade que define uma referência a um objeto.

```
' A propriedade Pen pode ser definida como diferentes implementações Pen.
Property Set Pen(P As Object)
    Set CurrentPen = P ' Atribui Pen ao objeto.
End Property
```

### Exemplo da instrução Public

Este exemplo utiliza a instrução **Public** no nível de módulo (seção Geral) de um módulo padrão para declarar explicitamente variáveis como públicas, ou seja, elas estão disponíveis para todos os procedimentos em todos os aplicativos, a menos que **Option Private Module** esteja ativa.

```
Public Número As Integer ' Variável Integer pública.
Public NameArray(1 To 5) As String ' Variável de matriz pública.
' Múltiplas declarações, duas Variants e um Integer, todos Public.
Public MyVar, YourVar, ThisVar As Integer
```

**Exemplo da instrução ReDim**

Este exemplo utiliza a instrução **ReDim** para alocar e realocar o espaço de armazenamento para variáveis de matriz dinâmica. Ele pressupõe que a **Option Base** seja 1.

```
Dim MyArray() As Integer ' Declara a matriz dinâmica.
Redim MyArray(5) ' Aloca 5 elementos.
For I = 1 To 5 ' Faz o loop 5 vezes.
    MyArray(I) = I ' Inicializa a matriz.
Next I
```

A próxima instrução redimensiona a matriz e apaga os elementos.

```
Redim MyArray(10) ' Redimensiona para 10 elementos.
For I = 1 To 10 ' Faz o loop 10 vezes.
    MyArray(I) = I ' Inicializa a matriz.
Next I
```

A instrução a seguir redimensiona a matriz, mas não apaga os elementos:

```
Redim Preserve MyArray(15) ' Redimensiona para 15 elementos.
```

**Exemplo da instrução Rem**

Este exemplo ilustra as várias formas da instrução **Rem**, que é utilizada para incluir comentários explicativos em um programa.

```
Rem Esta é a primeira forma da sintaxe.
```

A segunda forma da sintaxe é a seguinte:

```
Dim MyStr1, MyStr2
MyStr1 = "Hello": Rem Comentário após uma instrução separada por dois-
pontos.
MyStr2 = "Goodbye" ' Isto também é um comentário; não há necessidade de
dois-pontos.
```

**Exemplo da instrução Set**

Este exemplo utiliza a instrução **Set** para atribuir referências de objeto às variáveis. YourObject é considerado um objeto válido com uma propriedade **Text**.

```
Dim YourObject, MyObject, MyStr
Set MyObject = YourObject ' Atribui referência a objeto.
' MyObject e YourObject se referem ao mesmo objeto.
YourObject.Text = "Hello World" ' Inicializa a propriedade.
MyStr = MyObject.Text ' Retorna "Hello World".

' Descontinua a associação. MyObject não se refere mais a YourObject.
Set MyObject = Nothing ' Solta o objeto.
```

**Exemplo da instrução Static**

Este exemplo utiliza a instrução **Static** para conter o valor de uma variável desde que o código do módulo esteja sendo executado.

```
' Definição de função.
Function KeepTotal(Number)
    ' Somente a variável Accumulate preserva o seu valor entre chamadas.
    Static Accumulate
    Accumulate = Accumulate + Number
    KeepTotal = Accumulate
```

```

End Function
' Definição de função Static.
Static Function MyFunction(Arg1, Arg2, Arg3)
    ' Todas as variáveis locais preservam o valor entre chamadas de função.
    Accumulate = Arg1 + Arg2 + Arg3
    Half = Accumulate / 2
    MyFunction = Half
End Function

```

### Exemplo da instrução Sub

Este exemplo utiliza a instrução **Sub** para definir o nome, argumentos e o código que formam o corpo de um procedimento **Sub**.

```

' Definição do procedimento Sub.
' Procedimento Sub com dois argumentos.
Sub SubComputeArea(Length, TheWidth)
    Dim Area As Double ' Declara a variável local.
    If Length = 0 Or TheWidth = 0 Then
        ' Se ambos os argumentos = 0.
        Exit Sub ' Sai de Sub imediatamente.
    End If
    Area = Length * TheWidth ' Calcula a área do retângulo.
    Debug.Print Area ' Imprime a área na janela Depurar.
End Sub

```

### Exemplo da instrução Type

Este exemplo utiliza a instrução **Type** para definir um tipo de dados definido pelo usuário. A instrução **Type** é utilizada somente no nível de módulo. Se ela aparecer em um módulo de classe, uma instrução **Type** deverá ser precedida pela palavra-chave **Private**.

```

Type EmployeeRecord ' Cria o tipo definido pelo usuário.
    ID As Integer ' Define elementos de um tipo de dados.
    Name As String * 20
    Adress As String * 30
    Phone As Long
    HireDate As Date
End Type
Sub CreateRecord()
    Dim MyRecord As EmployeeRecord ' Declara a variável.
    ' A atribuição à variável EmployeeRecord deve ocorrer em um
    procedimento.
    MyRecord.ID = 12003 ' Atribui um valor a um elemento.
End Sub

```

### Exemplo da função UBound

Este exemplo utiliza a função **UBound** para determinar o maior subscrito disponível para a dimensão indicada de uma matriz.

```
Dim Upper
Dim MyArray(1 To 10, 5 To 15, 10 To 20) ' Declara as variáveis de matriz.
Dim AnyArray(10)
Upper = UBound(MyArray, 1) ' Retorna 10.
Upper = UBound(MyArray, 3) ' Retorna 20.
Upper = UBound(AnyArray) ' Retorna 10.
```

## Convenções da documentação

A documentação do Visual Basic utiliza as seguintes convenções tipográficas.

Convenções	Descrição
<b>Sub, If, ChDir, Print, True, Debug</b>	Palavras em negrito cuja primeira letra é maiúscula indicam palavras-chaves específicas da linguagem.
<b>Setup</b>	Palavras que você é instruído a digitar aparecem em negrito.
<i>Objeto, nomevar, lista_arg</i>	Letras minúsculas em itálico indicam lugares reservados para informações que você fornece.
<b><i>nomecaminho, númeroarquivo</i></b>	Letras minúsculas em itálico e negrito indicam lugares reservados para argumentos onde você pode usar sintaxe posicional ou de <u>argumento nomeado</u> .
[ <i>listaexpressão</i> ]	Na sintaxe, os itens entre de colchetes são opcionais.
{ <b>While</b>   <b>Until</b> }	Na sintaxe, chaves e uma barra vertical indicam uma escolha obrigatória entre dois ou mais itens. Você deve escolher um dos itens, a não ser que todos os itens também estejam entre colchetes. Por exemplo: [ <b>{Isto</b>   <b>OuAquilo</b> }]
ESC, ENTER	Palavras em maiúsculas pequenas indicam nomes de teclas e seqüências de teclas.
ALT+F1, CTRL+R	Um sinal de adição (+) entre nomes de teclas indica uma combinação de teclas. Por exemplo, ALT+F1 significa que você deve manter a tecla ALT pressionada enquanto pressiona a tecla F1.

### Convenções de código

São usadas as seguintes convenções de código:

Sample Code	Descrição
MinhaSeqüência = "Alô, mundo!"	Esta fonte é usada para código, variáveis e texto de mensagens de erro.
' Isto é um comentário.	Uma apóstrofo (') introduz comentários sobre o código.
MinhaVar = "Isto é um " _ & "exemplo"	Um espaço e uma sublinha ( _ ) dão continuidade a uma linha de código.

& " de como continuar o código."

### Exemplo da instrução ChDir

Este exemplo utiliza a instrução **ChDir** para alterar o diretório ou pasta atual.

```
' Altera o diretório ou pasta atual para "MEUDIR".
ChDir "MEUDIR"

' Pressupõe que "C:" é a unidade de disco atual. A instrução a seguir
altera
' o diretório padrão na unidade de disco "D:". "C:" permanece a unidade de
disco atual.
ChDir "D:\WINDOWS\SYSTEM"
```

### Exemplo da instrução ChDrive

Este exemplo utiliza a instrução **ChDrive** para alterar a unidade de disco atual.

```
ChDrive "D" ' Torna "D" a unidade de disco atual.
```

### Exemplo da função CurDir

Este exemplo utiliza a função **CurDir** para retornar o caminho atual.

```
' Pressupõe que o caminho atual na unidade de disco C é
"C:\WINDOWS\SYSTEM".
' Pressupõe que o caminho atual na unidade de disco D é "D:\EXCEL".
' Pressupõe que C é a unidade de disco atual.
Dim MeuCaminho
MeuCaminho = CurDir ' Retorna "C:\WINDOWS\SYSTEM".
MeuCaminho = CurDir("C") ' Retorna "C:\WINDOWS\SYSTEM".
MeuCaminho = CurDir("D") ' Retorna "D:\EXCEL".
```

### Exemplo da função Dir

Este exemplo utiliza a função **Dir** para verificar se determinados arquivos e diretórios existem.

```
Dim MeuArquivo, MeuCaminho, MeuNome
' Retorna "WIN.INI" se existir.
MeuArquivo = Dir("C:\WINDOWS\WIN.INI")

' Retorna o nome do arquivo com a extensão especificada. Se existir mais de
um arquivo *.ini,
' o primeiro arquivo encontrado será retornado.
MeuArquivo = Dir("C:\WINDOWS\*.INI")

' Chama Dir novamente sem argumentos para retornar o próximo arquivo *.INI
no
' mesmo diretório.
MeuArquivo = Dir

' Retorna primeiro o arquivo *.TXT com um atributo oculto definido.
MeuArquivo = Dir("*.TXT", vbHidden)

' Exibe os nomes em C:\ que representam diretórios.
MeuCaminho = "c:\" ' Define o caminho.
MeuNome = Dir(MeuCaminho, vbDirectory) ' Recupera a primeira entrada.
Do While MeuNome <> "" ' Inicia o loop.
' Ignora o diretório atual e o diretório abrangente.
If MeuNome <> "." And MeuNome <> ".." Then
```

```

    ' Utiliza a comparação bit a bit para se certificar de que MeuNome é
    um diretório.
    If (GetAttr(MeuCaminho & MeuNome) And vbDirectory) = vbDirectory Then
        Debug.Print MeuNome ' Exibe a entrada somente se
        End If ' representar um diretório.
    End If
    MeuNome = Dir ' Obtém a próxima entrada.
Loop

```

### Exemplo da função Environ

Este exemplo utiliza a função **Environ** para fornecer o número e comprimento de entrada da instrução PATH da tabela de seqüências de caracteres de ambiente.

```

Dim SeqAmb, Ind, Msg, CompCaminho ' Declara as variáveis.
Ind = 1 ' Inicializa o índice como 1.
Do
    SeqAmb = Environ(Ind) ' Obtém a variável
        ' de ambiente.
    If Left(SeqAmb, 5) = "PATH=" Then ' Verifica a entrada PATH.
        CompCaminho = Len(Environ("PATH")) ' Obtém o comprimento.
        Msg = "entrada PATH = " & Ind & " e comprimento = " & CompCaminho
        Exit Do
    Else
        Ind = Ind + 1 ' Não há entrada PATH,
    End If ' então incrementa.
Loop Until SeqAmb = ""
If CompCaminho > 0 Then
    MsgBox Msg ' Exibe a mensagem.
Else
    MsgBox "Não existe a variável de ambiente PATH."
End If

```

### Exemplo da instrução FileCopy

Este exemplo utiliza a instrução **FileCopy** para copiar um arquivo em outro. Para este exemplo, pressuponha que ARQORIGEM é um arquivo que contém alguns dados.

```

Dim ArquivoDeOrigem, ArquivoDeDestino
ArquivoDeOrigem = "ARQORIGEM" ' Define o nome do arquivo de origem.
ArquivoDeDestino = "ARQDESTINO" ' Define o nome do arquivo de destino.
FileCopy ArquivoDeOrigem, ArquivoDeDestino ' Copia a origem no destino.

```

### Exemplo da função `FileDateTime`

Este exemplo utiliza a função `FileDateTime` para determinar a data e a hora em que um arquivo foi criado ou modificado pela última vez. O formato da data e hora exibido é baseado nas definições de localidade do seu sistema.

```
Dim MinhaMarcação
' Pressupõe que ARQUIVODETESTE foi modificado pela última vez em 12 de
fevereiro de 1993, às 4:35:47 PM.
' Pressupõe as definições de localidade Inglês/EUA.
MinhaMarcação = FileDateTime("ARQUIVODETESTE") ' Retorna "12/2/93 4:35:47
PM".
```

### Exemplo da função `FileLen`

Este exemplo utiliza a função `FileLen` para retornar o comprimento de um arquivo em bytes. Para este exemplo, suponha que `ARQUIVODETESTE` é um arquivo que contém alguns dados.

```
Dim MeuTamanho
MeuTamanho = FileLen("ARQUIVODETESTE") ' Retorna o comprimento do arquivo
(bytes).
```

### Exemplo da função `GetAttr`

Este exemplo utiliza a função `GetAttr` para determinar os atributos de um arquivo e diretório ou pasta.

```
Dim MeuAtributo
' Pressupõe que o arquivo ARQUIVODETESTE tem o atributo de oculto definido.
MeuAtributo = GetAttr("ARQUIVODETESTE") ' Retorna 2.
' Retorna um valor diferente de zero se o atributo de oculto estiver
definido em ARQUIVODETESTE.
Debug.Print MeuAtributo And vbHidden
' Pressupõe que o arquivo ARQUIVODETESTE tem os atributos de oculto e
somente leitura definidos.
MeuAtributo = GetAttr("ARQUIVODETESTE") ' Retorna 3.
' Retorna um valor diferente de zero se o atributo de oculto estiver
definido em ARQUIVODETESTE.
Debug.Print MeuAtributo And (vbHidden + vbReadOnly)
' Pressupõe que MEUDIR é um diretório ou pasta.
MeuAtributo = GetAttr("MEUDIR") ' Retorna 16.
```

### Exemplo da instrução `Kill`

Este exemplo utiliza a instrução `Kill` para excluir um arquivo de um disco.

```
' Pressupõe que ARQUIVODETESTE é um arquivo que contém alguns dados.
Kill "ArquivoDeTeste" ' Exclui o arquivo.
' Exclui todos os arquivos *.TXT do diretório atual.
Kill "*.TXT"
```

### Exemplo da instrução MkDir

Este exemplo utiliza a instrução **MkDir** para criar um diretório ou pasta. Se a unidade de disco não for especificada, o novo diretório ou pasta será criado na unidade de disco atual.

```
MkDir "MEUDIR" ' Cria um novo diretório ou pasta.
```

### Exemplo da instrução Name

Este exemplo utiliza a instrução **Name** para renomear um arquivo. Para este exemplo, suponha que os diretórios ou pastas especificados já existem.

```
Dim NomeAntigo, NovoNome
```

```
NomeAntigo = "NOMEANTIGO": NovoNome = "NOVOARQUIVO" ' Define os nomes de arquivo.
```

```
Name NomeAntigo As NovoNome ' Renomeia o arquivo.
```

```
NomeAntigo = "C:\MEUDIR\ARQUIVOANTIGO": NovoNome = "C:\SEUDIR\NOVOARQUIVO"
```

```
Name NomeAntigo As NovoNome ' Move e renomeia o arquivo.
```

### Exemplo da função QBColor

Este exemplo utiliza a função **QBColor** para alterar a propriedade **BackColor** do formulário passado como `MeuFormulário` para a cor indicada por `CódigoDaCor`. **QBColor** aceita valores inteiros entre 0 e 15.

```
Sub ChangeBackColor (CódigoDaCor As Integer, MeuFormulário As Form)
```

```
    MeuFormulário.BackColor = QBColor (CódigoDaCor)
```

```
End Sub
```

### Exemplo da função RGB

Este exemplo mostra como a função **RGB** é utilizada para retornar um número inteiro que representa um valor de cor **RGB**. Ela é utilizada em métodos e propriedades de aplicativo que aceitam uma especificação de cor. O objeto `MeuObjeto` e a sua propriedade são utilizados somente com finalidades ilustrativas. Se `MeuObjeto` não existir, ou se não tiver uma propriedade `Color`, ocorrerá um erro.

```
Dim VERMELHO, I, ValorRGB, MeuObjeto
```

```
Vermelho = RGB (255, 0, 0) ' Retorna o valor para Vermelho.
```

```
I = 75 ' Inicializa o deslocamento.
```

```
ValorRGB = RGB (I, 64 + I, 128 + I) ' Igual a RGB (75, 139, 203).
```

```
MeuObjeto.Color = RGB (255, 0, 0) ' Define a propriedade Color de
```

```
    MeuObjeto como Vermelho.
```

### Exemplo da instrução Rmdir

Este exemplo utiliza a instrução **Rmdir** para remover um diretório ou pasta existente.

```
' Pressupõe que MEUDIR é um diretório ou pasta vazia.
```

```
Rmdir "MEUDIR" ' Remove MEUDIR.
```

### Exemplo da instrução SetAttr

Este exemplo utiliza a instrução **SetAttr** para definir atributos para um arquivo.

```
SetAttr "ARQUIVODETESTE ", vbHidden ' Define o atributo de oculto.  
SetAttr "ARQUIVODETESTE", vbHidden + vbReadOnly' Define os atributos de  
' oculto e somente leitura.
```

## Instrução ChDir

Altera o diretório ou pasta atual.

### Sintaxe

#### **ChDir** *caminho*

O argumento *caminho* obrigatório é uma expressão de seqüência que identifica qual diretório ou pasta se torna o novo diretório ou pasta padrão. O *caminho* pode incluir a unidade de disco. Se nenhuma unidade for especificada, **ChDir** altera o diretório ou pasta padrão na unidade atual.

### Comentários

A instrução **ChDir** altera o diretório padrão, mas não a unidade de disco padrão. Por exemplo, se a unidade padrão for C, a instrução a seguir altera o diretório padrão na unidade D, mas C permanece como unidade padrão:

```
ChDir "D:\TMP"
```

## Instrução ChDrive

Altera a unidade de disco atual.

### Sintaxe

#### **ChDrive** *unidade*

O argumento *unidade* obrigatório é uma expressão de seqüência que especifica uma unidade existente. Se você fornecer uma seqüência de comprimento zero (""), a unidade atual não será alterada. Se o argumento *unidade* for uma seqüência de múltiplos caracteres, **ChDrive** utilizará somente a primeira letra.

## Função CurDir

Retorna um **Variant (String)** representando o caminho atual.

### Sintaxe

#### **CurDir**[(*unidade*)]

O argumento opcional *unidade* é uma expressão de seqüência que especifica uma unidade existente. Se não for especificada uma unidade ou se *unidade* for uma seqüência de comprimento zero (""), **CurDir** retorna o caminho para a unidade atual.

## Função Dir

Retorna um **String** que representa o nome de um arquivo, diretório ou pasta que corresponde a um padrão especificado ou atributo de arquivo, ou o rótulo de volume de uma unidade de disco.

### Sintaxe

**Dir**[(*nomedocaminho*[, *atributos*])]

A sintaxe da função **Dir** possui as partes a seguir:

Parte	Descrição
<i>nomedocaminho</i>	Opcional. <u>Expressão de seqüência</u> que especifica um nome de arquivo – pode incluir diretório ou pasta e unidade de disco. Retorna uma seqüência de comprimento zero (""), se <i>nomedocaminho</i> não for encontrado.
<i>atributos</i>	Opcional. <u>Constante</u> ou <u>expressão numérica</u> cuja soma especifica os atributos de arquivo. Se for omitido, todos os arquivos que correspondem a <i>nomedocaminho</i> são retornados.

### Definições

As definições do argumento *atributos* são:

Constante	Valor	Descrição
<b>vbNormal</b>	0	Normal
<b>vbHidden</b>	2	Oculto
<b>vbSystem</b>	4	Arquivo do sistema
<b>vbVolume</b>	8	Rótulo do volume. Se for especificado, todos os outros atributos serão ignorados
<b>vbDirectory</b>	16	Diretório ou pasta

**Observação** Estas constantes são especificadas pelo Visual Basic para Aplicativos e podem ser utilizadas em qualquer parte do seu código no lugar de valores reais.

### Comentários

**Dir** suporta a utilização de curingas de múltiplos caracteres (\*) e de caractere único (?) para especificar vários arquivos.

Você deve especificar *nomedocaminho* na primeira vez que chamar a função **Dir** ou um erro será gerado. Se especificar também atributos de arquivo, será necessário incluir *nomedocaminho*.

**Dir** retorna o primeiro nome de arquivo que corresponda a *nomedocaminho*. Para obter qualquer nome de arquivo adicional que combine com *nomedocaminho*, chame **Dir** novamente sem argumentos. Quando não houver mais nenhum nome de arquivo correspondente, **Dir** retornará uma seqüência de comprimento zero (""), Uma vez que essa seqüência seja retornada, você deverá especificar *nomedocaminho* em chamadas subseqüentes ou um erro será gerado. Você pode mudar para um novo *nomedocaminho* sem recuperar todos os nomes de arquivo que correspondam ao *nomedocaminho* atual. Entretanto, você não pode chamar a função **Dir** recorrentemente. Chamar **Dir** com o atributo **vbDirectory** não retorna subdiretórios continuamente.

**Dica** Em razão de os nomes de arquivo serem recuperados sem nenhuma ordem específica, pode ser que você deseje armazenar nomes de arquivo retornados em uma matriz e depois classificá-la.

## Função Environ

Retorna o **String** associado a uma variável de ambiente do sistema operacional.

### Sintaxe

**Environ**({*envstring* | *number*})

A sintaxe da função **Environ** possui os argumentos nomeados a seguir:

Parte	Descrição
<i>envstring</i>	Opcional. <u>Expressão de seqüência</u> que contém o nome de uma variável de ambiente.
<i>number</i>	Opcional. <u>Expressão numérica</u> que corresponde à ordem numérica da seqüência de ambiente na tabela de seqüência de ambiente. O <u>argumento</u> <i>number</i> pode ser qualquer expressão numérica, mas será arredondado para um número inteiro antes de ser avaliado.

### Comentários

Se *envstring* não puder ser encontrado na tabela de seqüência de ambiente, uma seqüência de comprimento zero ("") será retornada. Caso contrário, **Environ** retornará o texto atribuído ao *envstring* especificado, isto é, o texto subsequente ao sinal de igual (=) na tabela de seqüência de ambiente daquela variável de ambiente.

Se você especificar *number*, a seqüência que ocupa aquela posição numérica na tabela de seqüência de ambiente será retornada. Neste caso, **Environ** retornará todo o texto, incluindo *envstring*. Se não houver seqüência de ambiente na posição especificada, **Environ** retornará uma seqüência de comprimento zero.

## Instrução FileCopy

Copia um arquivo.

### Sintaxe

**FileCopy** *source*, *destination*

A sintaxe da instrução **FileCopy** possui os argumentos nomeados a seguir:

Parte	Descrição
<i>source</i>	Obrigatório. <u>Expressão de seqüência</u> que especifica o nome do arquivo a ser copiado. <b>Source</b> pode incluir diretório ou pasta e unidade de disco.
<i>destination</i>	Obrigatório. <u>Expressão de seqüência</u> que especifica o nome do arquivo de destino. <b>Destination</b> pode incluir diretório ou pasta e unidade de disco.

### Comentários

Se você tentar utilizar a instrução **FileCopy** em um arquivo atualmente aberto, um erro será gerado.

## Função FileDateTime

Retorna um **Variant (Date)** que indica a data e hora em que um arquivo foi criado ou modificado pela última vez.

### Sintaxe

**FileDateTime**(*nomedocaminho*)

O argumento obrigatório *nomedocaminho* é uma expressão de seqüência que especifica um nome de arquivo. O *nomedocaminho* pode incluir o diretório ou pasta e a unidade de disco.

## Função FileLen

Retorna um **Long** que especifica o tamanho de um arquivo em bytes.

### Sintaxe

**FileLen**(*nomedocaminho*)

O argumento obrigatório *nomedocaminho* é uma expressão de seqüência que especifica um arquivo. O *nomedocaminho* pode incluir o diretório ou pasta e a unidade de disco.

### Comentários

Se o arquivo especificado estiver aberto quando a função **FileLen** for chamada, o valor retornado representa o tamanho do arquivo imediatamente antes de ele ser aberto.

**Observação** Para obter o tamanho de um arquivo aberto, utilize a função **LOF**.

## Função GetAttr

Retorna um **Integer** que representa os atributos de um arquivo, diretório ou pasta.

### Sintaxe

**GetAttr**(*nomedocaminho*)

O argumento obrigatório *nomedocaminho* é uma expressão de seqüência que especifica um nome de arquivo. O *nomedocaminho* pode incluir o diretório ou pasta e a unidade de disco.

### Valores de retorno

O valor retornado por **GetAttr** é a soma dos valores de atributos a seguir:

Constante	Valor	Descrição
<b>vbNormal</b>	0	Normal
<b>vbReadOnly</b>	1	Somente leitura
<b>vbHidden</b>	2	Oculto
<b>vbSystem</b>	4	Sistema
<b>vbDirectory</b>	16	Diretório ou pasta
<b>vbArchive</b>	32	O arquivo foi alterado desde o último backup

**Observação** Estas constantes são especificadas pelo Visual Basic para Aplicativos. Os nomes podem ser utilizados em qualquer parte do seu código no lugar dos valores reais.

### Comentários

Para determinar quais atributos estão definidos, utilize o operador **And** para efetuar uma comparação bit a bit do valor retornado pela função **GetAttr** e do valor do atributo individual de arquivo que você deseja. Se o resultado não for zero, o atributo será definido para o arquivo nomeado. Por exemplo, o valor de retorno da expressão **And** a seguir será zero se o atributo Archive não estiver definido:

```
Result = GetAttr(FName) And vbArchive
```

Um valor diferente de zero será retornado se o atributo Archive estiver definido.

## Instrução Kill

Exclui arquivos de um disco.

### Sintaxe

**Kill** *nomedocaminho*

O argumento obrigatório *nomedocaminho* é uma expressão de seqüência que especifica um ou mais nomes de arquivo a serem excluídos. O *nomedocaminho* pode incluir o diretório ou pasta e a unidade de disco.

### Comentários

**Kill** suporta a utilização de curingas de múltiplos caracteres (\*) ou de caractere único (?) para especificar vários arquivos.

Se você tentar utilizar **Kill** para excluir um arquivo aberto, um erro será gerado.

**Observação** Para excluir diretórios, utilize a instrução **Rmdir**.

## Instrução Mkdir

Cria um novo diretório ou pasta.

### Sintaxe

**Mkdir** *caminho*

O argumento obrigatório *caminho* é uma expressão de seqüência que identifica o diretório ou pasta a ser criado. O *caminho* pode incluir a unidade de disco. Se uma unidade não for especificada, **Mkdir** cria o novo diretório ou pasta na unidade atual.

## Instrução Name

Renomeia um arquivo, diretório ou pasta no disco.

### Sintaxe

**Name** *nomeantigodocaminho* **As** *nomenovodocaminho*

A sintaxe da instrução **Name** possui as partes a seguir:

Parte	Descrição
<i>nomeantigodocaminho</i>	Obrigatório. <u>Expressão de seqüência</u> que especifica o nome de arquivo existente e sua localização – pode incluir diretório ou pasta e unidade de disco.
<i>Nomenovodocaminho</i>	Obrigatório. Expressão de seqüência que especifica o novo nome de arquivo e sua localização – pode incluir diretório ou pasta e unidade de disco. O nome de arquivo especificado por <i>nomenovodocaminho</i> não pode já existir.

### Comentários

Tanto *nomenovodocaminho* quanto *nomeantigodocaminho* devem estar na mesma unidade. Se o

caminho em *nomenovodocaminho* existir e for diferente do caminho em *nomeantigodocaminho*, a instrução **Name** moverá o arquivo para o novo diretório ou pasta e, se necessário, renomeará o arquivo. Se *nomenovodocaminho* e *nomeantigodocaminho* tiverem caminhos diferentes e nomes de arquivo iguais, **Name** moverá o arquivo para a nova localização e não alterará o nome do arquivo. Utilizando **Name**, você poderá mover um arquivo de um diretório ou pasta para outro, mas não poderá mover um diretório ou pasta.

A utilização de **Name** em um arquivo aberto gera um erro. Você deve fechar um arquivo que esteja aberto antes de renomeá-lo. Os argumentos de **Name** não podem conter curingas de múltiplos caracteres (\*) nem de caractere único (?).

## Função QBColor

Retorna um **Long** que representa o código de cores RGB correspondente ao número de cor especificado.

### Sintaxe

**QBColor**(*cor*)

O argumento obrigatório *cor* é um número inteiro no intervalo de 0 a 15.

### Definições

O argumento *cor* possui as definições a seguir:

Número	Cor	Número	Cor
0	Preto	8	Cinza
1	Azul	9	Azul-claro
2	Verde	10	Verde-claro
3	Ciano	11	Ciano-claro
4	Vermelho	12	Vermelho-claro
5	Magenta	13	Magenta-claro
6	Amarelo	14	Amarelo-claro
7	Branco	15	Branco-brilhante

### Comentários

O argumento *cor* representa valores de cor utilizados por versões anteriores do Basic (como o Microsoft Visual Basic para MS-DOS e o Basic Compiler). Começando pelo byte menos significativo, o valor retornado especifica os valores de vermelho, verde e azul utilizados para definir a cor apropriada no sistema RGB utilizado pelo Visual Basic para Aplicativos.

## Função RGB

Retorna um número inteiro **Long** que representa um valor de cor RGB.

### Sintaxe

**RGB**(*red, green, blue*)

A sintaxe da função **RGB** possui os argumentos nomeados a seguir:

Parte	Descrição
<i>red</i>	Obrigatório; <b>Variant (Integer)</b> . Número no intervalo de 0 a 255, inclusive, que representa o componente vermelho da cor.
<i>green</i>	Obrigatório; <b>Variant (Integer)</b> . Número no intervalo de 0 a.255, inclusive, que representa o componente verde da cor.
<i>blue</i>	Obrigatório; <b>Variant (Integer)</b> . Número no intervalo de 0 a 255, inclusive, que representa o componente azul da cor.

### Comentários

Os métodos e as propriedades de aplicativos que aceitam uma especificação de cor esperam que essa especificação seja um número que represente um valor de cor RGB. Este valor especifica a intensidade relativa de vermelho, verde e azul que causa a exibição de uma cor específica.

O valor de qualquer argumento de **RGB** que exceda 255 é assumido como sendo 255.

A tabela a seguir lista algumas cores padrão e os valores de vermelho, verde, e azul que elas contêm:

Cor	Valor do vermelho	Valor do verde	Valor do azul
Preto	0	0	0
Azul	0	0	255
Verde	0	255	0
Ciano	0	255	255
Vermelho	255	0	0
Magenta	255	0	255
Amarelo	255	255	0
Branco	255	255	255

## Instrução Rmdir

Remove um diretório ou pasta existente.

### Sintaxe

**Rmdir** *caminho*

O argumento obrigatório *caminho* é uma expressão de seqüência que identifica o diretório ou pasta a ser removida. O *caminho* pode incluir a unidade de disco. Se uma unidade não for especificada, **Rmdir** remove o diretório ou pasta da unidade atual.

### Comentários

Se você tentar utilizar **Rmdir** em um diretório ou pasta que contenha arquivos, um erro será gerado. Utilize a instrução **Kill** para excluir todos os arquivos antes de tentar a remoção de um diretório ou pasta.

## Instrução SetAttr

Define as informações de atributo de um arquivo.

### Sintaxe

**SetAttr** *pathname, attributes*

A sintaxe da instrução **SetAttr** possui os argumentos nomeados a seguir:

Parte	Descrição
<b><i>pathname</i></b>	Obrigatório. <u>Expressão de seqüência</u> que especifica um nome de arquivo – pode incluir diretório ou pasta e unidade de disco.
<b><i>Attributes</i></b>	Obrigatório. <u>Constante</u> ou <u>expressão numérica</u> cuja soma especifica os atributos do arquivo.

### Definições

As definições do argumento ***attributes*** são:

Constante	Valor	Descrição
-----------	-------	-----------

<b>vbNormal</b>	0	Normal (padrão)
<b>vbReadOnly</b>	1	Somente leitura
<b>vbHidden</b>	2	Oculto
<b>vbSystem</b>	4	Arquivo do sistema
<b>vbArchive</b>	32	O arquivo foi alterado desde o último backup

**Observação** Estas constantes são especificadas pelo Visual Basic para Aplicativos. Os nomes podem ser utilizados em qualquer parte do seu código no lugar dos valores reais.

### Comentários

Um erro em tempo de execução ocorrerá se você tentar definir os atributos de um arquivo que esteja aberto.

### Exemplo da instrução Close

Este exemplo utiliza a instrução **Close** para fechar os três arquivos abertos para **Output**.

```
Dim I, NomeDoArquivo
For I = 1 To 3 ' Faz o loop 3 vezes.
    NomeDoArquivo = "TESTE" & I      ' Cria o nome de arquivo.
    Open NomeDoArquivo For Output As #I ' Abre o arquivo.
    Print #I, "Isto é um teste."      ' Grava a seqüência de caracteres no
arquivo.
Next I
Close ' Fecha os 3 arquivos abertos.
```

### Exemplo da função EOF

Este exemplo utiliza a função **EOF** para detectar o final de um arquivo. Este exemplo pressupõe que MEUARQUIVO seja um arquivo de texto com algumas linhas de texto.

```
Dim DadosDeEntrada
Open "MEUARQUIVO" For Input As #1 ' Abre o arquivo para entrada.
Do While Not EOF(1) ' Procura o fim do arquivo.
    Line Input #1, DadosDeEntrada ' Lê a linha de dados.
    Debug.Print DadosDeEntrada    ' Imprime para a janela Depurar.
Loop
Close #1 ' Fecha o arquivo.
```

### Exemplo da função FileAttr

Este exemplo utiliza a função **FileAttr** para retornar o modo de arquivo e o identificador de arquivo de um arquivo aberto.

```
Dim NumArquivo, Modo, Identificador
NumArquivo = 1 ' Atribui o número de arquivo.
Open "ARQUIVODETESTE" For Append As NumArquivo ' Abre o arquivo.
Modo = FileAttr(NumArquivo, 1) ' Retorna 8 (modo de arquivo Acréscimo).
Identificador = FileAttr(NumArquivo, 2) ' Retorna o identificador do
arquivo.
Close NumArquivo ' Fecha o arquivo.
```

### Exemplo da função FreeFile

Este exemplo utiliza a função **FreeFile** para retornar o próximo número de arquivo disponível. Cinco arquivos são abertos para saída dentro do loop e alguns dados de exemplo são gravados em cada.

```
Dim MeuÍndice, NúmeroDoArquivo
For MeuÍndice = 1 To 5 ' Faz o loop 5 vezes.
    NúmeroDoArquivo = FreeFile ' Obtém o número de arquivo
    ' não utilizado.
    Open "TESTE" & MeuÍndice For Output As #NúmeroDoArquivo ' Cria o nome do
arquivo.
    Write #NúmeroDoArquivo, "Isto é um exemplo." ' Dá saída ao texto.
    Close #NúmeroDoArquivo ' Fecha o arquivo.
Next MeuÍndice
```

### Exemplo da instrução Get

Este exemplo utiliza a instrução **Get** para ler dados de um arquivo para uma variável. Este exemplo pressupõe que ARQUIVODETESTE seja um arquivo que contenha cinco registros do tipo Record definido pelo usuário.

```
Type Record ' Define o tipo definido pelo usuário.
    ID As Integer
    Name As String * 20
End Type

Dim MeuRegistro As Record, Posição ' Declara as variáveis.
' Abre o arquivo de exemplo para acesso aleatório.
Open "ARQUIVODETESTE" For Random As #1 Len = Len(MeuRegistro)
' Lê o arquivo de exemplo utilizando a instrução Get.
Posição = 3 ' Define o número do registro.
Get #1, Posição, MeuRegistro ' Lê o terceiro registro.
Close #1 ' Fecha o arquivo.
```

### Exemplo da instrução Input #

Este exemplo utiliza a instrução **Input #** para ler dados de um arquivo em duas variáveis. Este exemplo pressupõe que ARQUIVODETESTE seja um arquivo com algumas linhas de dados gravadas nele utilizando a instrução **Write #**, ou seja, cada linha contém uma seqüência de caracteres entre aspas e um número separado por uma vírgula; por exemplo, ("Alô", 234).

```
Dim MinhaSeqüência, MeuNúmero
Open "ARQUIVODETESTE" For Input As #1 ' Abre o arquivo para entrada.
Do While Not EOF(1) ' Faz o loop até o fim do arquivo.
    Input #1, MinhaSeqüência, MeuNúmero ' Lê os dados para duas variáveis.
    Debug.Print MinhaSeqüência, MeuNúmero ' Imprime os dados para a janela
Depurar.
Loop
Close #1 ' Fecha o arquivo.
```

**Exemplo da função Input**

Este exemplo utiliza a função **Input** para ler um caractere de cada vez de um arquivo e o imprime para a janela **Depurar**. Este exemplo pressupõe que ARQUIVODETESTE seja um arquivo com algumas linhas de dados de exemplo.

```
Dim MeuCar
Open "ARQUIVODETESTE" For Input As #1 ' Abre o arquivo.
Do While Not EOF(1) ' Faz o loop até o fim do arquivo.
    MeuCar = Input(1, #1) ' Obtém um caractere.
    Debug.Print MeuCar ' Imprime para a janela Depurar.
Loop
Close #1 ' Fecha o arquivo.
```

**Exemplo da instrução Line Input #**

Este exemplo utiliza a instrução **Line Input #** para ler uma linha de um arquivo seqüencial e a atribui a uma variável. Este exemplo pressupõe que ARQUIVODETESTE seja um arquivo de texto com algumas linhas de dados de exemplo.

```
Dim LinhaDeTexto
Open "ARQUIVODETESTE" For Input As #1 ' Abre o arquivo.
Do While Not EOF(1) ' Faz o loop até o fim do arquivo.
    Line Input #1, LinhaDeTexto ' Lê a linha para a variável.
    Debug.Print LinhaDeTexto ' Imprime para a janela Depurar.
Loop
Close #1 ' Fecha o arquivo.
```

**Exemplo da função Loc**

Este exemplo utiliza a função **Loc** para retornar a posição atual de leitura/gravação dentro de um arquivo aberto. Este exemplo pressupõe que ARQUIVODETESTE seja um arquivo de texto com algumas linhas de dados de exemplo.

```
Dim MeuLocal, MinhaLinha
Open "ARQUIVODETESTE" For Binary As #1 ' Abre o arquivo recém-criado.
Do While MeuLocal < LOF(1) ' Faz o loop até o fim do arquivo.
    MinhaLinha = MinhaLinha & Input(1, #1) ' Lê o caractere para a
variável.
    MeuLocal = Loc(1) ' Obtém a posição atual dentro do arquivo.
' Imprime para a janela Depurar.
    Debug.Print MinhaLinha; Tab; MeuLocal
Loop
Close #1 ' Fecha o arquivo.
```

**Exemplo das instruções Lock e Unlock**

Este exemplo ilustra o uso das instruções **Lock** e **Unlock**. Enquanto um registro está sendo modificado, o acesso por outros processos é negado. Este exemplo pressupõe que ARQUIVODETESTE seja um arquivo que contenha cinco registros do tipo Record definido pelo usuário.

```
Type Record ' Define o tipo definido pelo usuário.
    ID As Integer
    Name As String * 20
End Type

Dim MeuRegistro As Record, NúmeroDoRegistro ' Declara as variáveis.
' Abre o arquivo de exemplo para acesso aleatório.
Open "ARQUIVODETESTE" For Random Shared As #1 Len = Len(MeuRegistro)
```

```
NúmeroDoRegistro = 4 ' Define o número do registro.  
Lock #1, NúmeroDoRegistro ' Protege o registro.  
Get #1, NúmeroDoRegistro, MeuRegistro ' Lê o registro.  
MeuRegistro.ID = 234 ' Modifica o registro.  
MeuRegistro.Name = "João da Silva"  
Put #1, NúmeroDoRegistro, MeuRegistro ' Grava o arquivo modificado.  
Unlock #1, NúmeroDoRegistro ' Desprotege o registro atual.  
Close #1 ' Fecha o arquivo.
```

### Exemplo da função LOF

Este exemplo utiliza a função **LOF** para determinar o tamanho de um arquivo aberto. Este exemplo pressupõe que ARQUIVODETESTE seja um arquivo de texto que contenha dados de exemplo.

```
Dim ComprimentoDoArquivo  
Open "ARQUIVODETESTE" For Input As #1 ' Abre o arquivo.  
ComprimentoDoArquivo = LOF(1) ' Obtém o comprimento do arquivo.  
Close #1 ' Fecha o arquivo.
```

### Exemplo da instrução Open

Este exemplo ilustra vários usos da instrução **Open** para permitir a entrada e saída em um arquivo.

O código a seguir abre o arquivo ARQUIVODETESTE no modo de entrada seqüencial.

```
Open "ARQUIVODETESTE" For Input As #1  
' Fecha antes de reabrir em outro modo.  
Close #1
```

Este exemplo abre o arquivo no modo Binário somente para gravar operações.

```
Open "ARQUIVODETESTE" For Binary Access Write As #1  
' Fecha antes de reabrir em outro modo.  
Close #1
```

O exemplo a seguir abre o arquivo no modo Aleatório. O arquivo contém registros do tipo Record definido pelo usuário.

```
Type Record ' Define o tipo definido pelo usuário.  
    ID As Integer  
    Name As String * 20  
End Type
```

```
Dim MeuRegistro As Record ' Declara a variável.  
Open "ARQUIVODETESTE" For Random As #1 Len = Len(MeuRegistro)  
' Fecha antes de reabrir em outro modo.  
Close #1
```

Este exemplo de código abre o arquivo para saída seqüencial; todo processo pode ler ou gravar no arquivo.

```
Open "ARQUIVODETESTE" For Output Shared As #1  
' Fecha antes de reabrir em outro modo.  
Close #1
```

Este exemplo de código abre o arquivo no modo Binário para leitura; outros processos não conseguem ler o arquivo.

```
Open "ARQUIVODETESTE" For Binary Access Read Lock Read As #1
```

**Exemplo da instrução Print #**

Este exemplo utiliza a instrução **Print #** para gravar dados em um arquivo.

```
Open "ARQUIVODETESTE" For Output As #1 ' Abre o arquivo para saída.
Print #1, "Isto é um teste" ' Imprime o texto para o arquivo.
Print #1, ' Imprime a linha no arquivo.
Print #1, "Zona 1"; Tab ; "Zona 2" ' Imprime em duas zonas de impressão.
Print #1, "Alô" ; " " ; "Mundo" ' Seqüências de caracteres separadas por
espaço.
Print #1, Spc(5) ; "5 espaços à esquerda " ' Imprime cinco espaços à
esquerda.
Print #1, Tab(10) ; "Alô" ' Imprime a palavra na coluna 10.

' Atribui valores Boolean, Date, Null e Error.
Dim MeuBool, MinhaData, MeuNulo, MeuErro
MeuBool = False : MinhaData = #Fevereiro 12, 1969# : MeuNulo = Null
MeuErro = CVErr(32767)
' True, False, Null e Error são traduzidos utilizando-se as configurações
de localidade do seu
' sistema. As literais de data são gravadas utilizando-se a data abreviada
padrão.

Print #1, MeuBool ; " é um valor Booleano"
Print #1, MinhaData ; " é uma data"
Print #1, MeuNulo ; " é um valor nulo"
Print #1, MeuErro ; " é um valor de erro"
Close #1 ' Fecha o arquivo.
```

**Exemplo da instrução Put**

Este exemplo utiliza a instrução **Put** para gravar dados em um arquivo. Cinco registros do tipo Record definido pelo usuário são gravados no arquivo.

```
Type Record ' Define o tipo definido pelo usuário.
    ID As Integer
    Name As String * 20
End Type

Dim MeuRegistro As Record, NúmeroDoRegistro ' Declara as variáveis.
' Abre o arquivo para acesso aleatório.
Open "ARQUIVODETESTE" For Random As #1 Len = Len(MeuRegistro)
For NúmeroDoRegistro = 1 To 5 ' Faz o loop 5 vezes.
    MeuRegistro.ID = NúmeroDoRegistro ' Define o código.
    MeuRegistro.Name = "Meu Nome" & NúmeroDoRegistro ' Cria uma seqüência
de caracteres.
    Put #1, NúmeroDoRegistro, MeuRegistro ' Grava o registro no arquivo.
Next NúmeroDoRegistro
Close #1 ' Fecha o arquivo.
```

### Exemplo da instrução Reset

Este exemplo utiliza a instrução **Reset** para fechar todos os arquivos abertos e gravar o conteúdo de todos os buffers de arquivo em disco. Observe o uso da variável **Variant** NúmeroDoArquivo tanto como a seqüência de caracteres e o número.

```
Dim NúmeroDoArquivo
For NúmeroDoArquivo = 1 To 5 ' Faz o loop 5 vezes.
  ' Abre o arquivo para saída. NúmeroDoArquivo é concatenado na seqüência
  de caracteres
  ' TESTE para o nome do arquivo, mas é um número que se segue a #.
  Open "TESTE" & NúmeroDoArquivo For Output As #NúmeroDoArquivo
  Write #NúmeroDoArquivo, "Alô Mundo" ' Grava os dados no arquivo.
Next NúmeroDoArquivo
Reset ' Fecha os arquivos e grava o conteúdo
      ' em disco.
```

### Exemplo da função Seek

Este exemplo utiliza a função **Seek** para retornar a posição de arquivo atual. O exemplo pressupõe que ARQUIVODETESTE seja um arquivo que contenha registros do tipo Record definido pelo usuário.

```
Type Record ' Define o tipo definido pelo usuário.
  ID As Integer
  Name As String * 20
End Type
```

Para arquivos abertos no modo Aleatório, **Seek** retorna o número do próximo registro.

```
Dim MeuRegistro As Record ' Declara a variável.
Open "ARQUIVODETESTE" For Random As #1 Len = Len(MeuRegistro)
Do While Not EOF(1) ' Faz o loop até o final do arquivo.
  Get #1, , MeuRegistro ' Lê o próximo registro.
  Debug.Print Seek(1) ' Imprime o número do registro para a janela
  ' Depurar.
Loop
Close #1 ' Fecha o arquivo.
```

Para arquivos abertos em modo diferentes do modo **Aleatório**, **Seek** retorna a posição do byte em que ocorre a próxima operação. Suponha que ARQUIVODETESTE seja um arquivo que contenha algumas linhas de texto.

```
Dim MeuCar
Open "ARQUIVODETESTE" For Input As #1 ' Abre o arquivo para leitura.
Do While Not EOF(1) ' Faz o loop até o fim do arquivo.
  MeuCar = Input(1, #1) ' Lê o próximo caractere de dados.
  Debug.Print Seek(1) ' Imprime a posição do byte para a janela
  ' Depurar.
Loop
Close #1 ' Fecha o arquivo.
```

### Exemplo da instrução Seek

Este exemplo utiliza a instrução **Seek** para definir a posição da próxima leitura ou gravação dentro de um arquivo. Este exemplo pressupõe que ARQUIVODETESTE seja um arquivo que contenha registros do tipo Record definido pelo usuário.

```
Type Record ' Define o tipo definido pelo usuário.
    ID As Integer
    Name As String * 20
End Type
```

Para arquivos abertos no modo Aleatório, **Seek** define o próximo registro.

```
Dim MeuRegistro As Record, TamanhoMax, NúmeroDoRegistro ' Declara as
variáveis.
' Abre o arquivo no modo de arquivo aleatório.
Open "ARQUIVODETESTE" For Random As #1 Len = Len(MeuRegistro)
TamanhoMax = LOF(1) \ Len(MeuRegistro) ' Obtém número de registros no
arquivo.
' O loop lê todos os registros a começar do último.
For NúmeroDoRegistro = TamanhoMax To 1 Step -1
    Seek #1, NúmeroDoRegistro ' Define a posição.
    Get #1, , MeuRegistro ' Lê o registro.
Next NúmeroDoRegistro
Close #1 ' Fecha o arquivo.
```

Para arquivos abertos em modos diferentes do modo **Aleatório**, **Seek** define a posição do byte em que a próxima operação ocorre. Suponha que ARQUIVODETESTE seja um arquivo que contenha algumas linhas de texto.

```
Dim TamanhoMax, PróximoCar, MeuCar
Open "ARQUIVODETESTE" For Input As #1 ' Abre o arquivo para entrada.
TamanhoMax = LOF(1) ' Obtém o tamanho do arquivo em bytes.
' O loop lê todos os caracteres a começar do último.
For PróximoCar = TamanhoMax To 1 Step -1
    Seek #1, PróximoCar ' Define a posição.
    MeuCar = Input(1, #1) ' Lê os caracteres.
Next PróximoCar
Close #1 ' Fecha o arquivo.
```

### Exemplo da função Spc

Este exemplo utiliza a função **Spc** para posicionar a saída em um arquivo e na janela **Depurar**.

```
' A função Spc pode ser utilizada com a instrução Print #.
Open "ARQUIVODETESTE" For Output As #1 ' Abre o arquivo para saída.
Print #1, "10 espaços entre aqui "; Spc(10); "e aqui."
Close #1 ' Fecha o arquivo.
```

A instrução a seguir faz com que o texto seja impresso na janela **Depurar** (utilizando o método Print), precedido por 30 espaços.

```
Debug.Print Spc(30); "Trinta espaços depois..."
```

**Exemplo da função Tab**

Este exemplo utiliza a função **Tab** para posicionar a saída em um arquivo e na janela **Depurar**.

```
' A função Tab pode ser utilizada com a instrução Print #.
Open "ARQUIVODETESTE" For Output As #1 ' Abre o arquivo para saída.
' A segunda palavra é impressa na coluna 20.
Print #1, "Alô"; Tab(20); "Mundo."
' Se o argumento for omitido, o cursor será movido para a próxima zona de
impressão.
Print #1, "Alô"; Tab; "Mundo"
Close #1 ' Fecha o arquivo.
```

A função **Tab** também pode ser utilizada com o método **Print**. A instrução a seguir imprime o texto começando na coluna 10.

```
Debug.Print Tab(10); "10 colunas depois do início."
```

**Exemplo da instrução Width #**

Este exemplo utiliza a instrução **Width #** para definir a largura da linha de saída de um arquivo.

```
Dim I
Open "ARQUIVODETESTE" For Output As #1 ' Abre o arquivo para saída.
Width #1, 5 ' Define a largura da linha de saída como 5.
For I = 0 To 9 ' Faz o loop 10 vezes.
    Print #1, Chr(48 + I); ' Imprime cinco caracteres por linha.
Next I
Close #1 ' Fecha o arquivo.
```

**Exemplo da instrução Write #**

Este exemplo utiliza a instrução **Write #** para gravar dados brutos em um arquivo seqüencial.

```
Open "ARQUIVODETESTE" For Output As #1 ' Abre o arquivo para saída.
Write #1, "Alô Mundo", 234 ' Grava os dados delimitados por vírgula.
Write #1, ' Grava linha em branco.

Dim MeuBool, MinhaData, MeuNulo, MeuErro
' Atribui os valores Boolean, Date, Null e Error.
MeuBool = False : MinhaData = #Fevereiro 12, 1969# : MeuNulo = Null
MeuErro = CVErr(32767)
' Os dados Booleanos são gravados como #TRUE# ou #FALSE#. As literais de
data são
' gravadas em formato de data universal; por exemplo, #1994-07-13#
' representa 13 de julho de 1994. Os dados nulos são gravados como #NULL#.
' Os dados Error são gravados como #ERROR errorcode#.
Write #1, MeuBool ; " é um valor Booleano"
Write #1, MinhaData ; " é uma data"
Write #1, MeuNulo ; " é um valor nulo"
Write #1, MeuErro ; " é um valor de erro"
Close #1 ' Fecha o arquivo.
```

## Instrução Close

Conclui a entrada/saída (E/S) para um arquivo aberto utilizando-se a instrução **Open**.

### Sintaxe

**Close** [*listadenúmero doarquivo*]

O argumento opcional *listadenúmero doarquivo* pode ser um ou mais números de arquivo que utilizam a sintaxe a seguir, onde *número doarquivo* é qualquer número de arquivo válido:

[[#]*número doarquivo*] [, [#]*número doarquivo*] . . .

### Comentários

Se você omitir *listadenúmero doarquivo*, todos os arquivos ativos abertos pela instrução **Open** serão fechados.

Quando você fecha os arquivos que foram abertos para **Output** ou **Append**, o buffer final de saída é gravado no buffer do sistema operacional referente ao arquivo. Todo o espaço de buffer associado ao arquivo fechado é liberado.

Quando a instrução **Close** é executada, encerra-se a associação de um arquivo com o seu número de arquivo.

## Função EOF

Retorna um **Integer** contendo o valor **Boolean True** quando se chega ao final de um arquivo aberto para **Input** seqüencial ou **Random**.

### Sintaxe

**EOF**(*número doarquivo*)

O argumento obrigatório *número doarquivo* é um **Integer** contendo qualquer número de arquivo válido.

### Comentários

Utilize **EOF** para evitar o erro gerado pela tentativa de inserir dados depois do final de um arquivo.

A função **EOF** retorna **False** enquanto não o final do arquivo não é alcançado. Com os arquivos abertos para o acesso **Random** ou **Binary**, **EOF** retorna **False** até que a última instrução **Get** executada não possa ler um registro inteiro.

Com os arquivos abertos para acesso **Binary**, uma tentativa de leitura do arquivo utilizando-se a função **Input** até que **EOF** retorne **True** gera um erro. Utilize as funções **LOF** e **Loc** em vez de **EOF** ao ler os arquivos binários com **Input** ou utilize **Get** ao utilizar a função **EOF**. Com os arquivos abertos para **Output**, **EOF** sempre retorna **True**.

## Função FileAttr

Retorna um **Long** representando o modo de arquivo para arquivos abertos com a instrução **Open**.

### Sintaxe

**FileAttr**(*filenumber, returntype*)

A sintaxe da função **FileAttr** possui os argumentos nomeados a seguir:

<u>Parte</u>	<u>Descrição</u>
<i>filenumber</i>	Obrigatório; <b>Integer</b> . Qualquer <u>número de arquivo</u> válido.
<i>returntype</i>	Obrigatório; <b>Integer</b> . Número indicando os tipos de informações a serem retornadas. Especifique 1 para retornar um valor indicando o modo de arquivo. Somente em sistemas que sejam de 16 bits, especifique 2 para recuperar

um identificador de arquivo de sistema operacional.  
**Returntype** 2 não é suportado em sistemas de 32 bits e gera um erro.

### Valores de retorno

Quando o argumento **returntype** é 1, os valores de retorno a seguir indicam o modo de acesso ao arquivo:

Modo	Valor
<b>Input</b>	1
<b>Output</b>	2
<b>Random</b>	4
<b>Append</b>	8
<b>Binary</b>	32

## Função FreeFile

Retorna um **Integer** representando o próximo número de arquivo disponível para utilização pela instrução **Open**.

### Sintaxe

**FreeFile**[(*númerodointervalo*)]

O argumento opcional *númerodointervalo* é um **Variant** que especifica o intervalo a partir do qual o próximo número de arquivo disponível deve ser retornado. Especifique 0 (padrão) para retornar um número de arquivo no intervalo de 1 a 255, inclusive. Especifique 1 para retornar um número de arquivo no intervalo de 256 a 511.

### Comentários

Utilize **FreeFile** para fornecer um número de arquivo que não esteja já sendo utilizado.

## Instrução Get

Lê os dados a partir de um arquivo de disco aberto e os coloca em uma variável.

### Sintaxe

**Get** [#]númerodoarquivo, [númerodoreg], nomedavar

A sintaxe da instrução **Get** possui as partes a seguir:

Parte	Descrição
<i>númerodoarquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.
<i>númerodoreg</i>	Opcional. <b>Variant (Long)</b> . Número do registro (arquivos no modo <b>Random</b> ) ou número de byte (arquivos no modo <b>Binary</b> ) no qual é iniciada a leitura.
<i>nomedavar</i>	Obrigatório. Nome de variável válido no qual os dados são lidos.

### Comentários

A leitura de dados com **Get** é normalmente gravada em um arquivo com **Put**.

O primeiro registro ou byte em um arquivo encontra-se na posição 1, o segundo registro ou byte encontra-se na posição 2 e assim por diante. Se você omitir *númerodoreg*, o próximo registro ou byte após a última instrução **Get** ou **Put** (ou apontado pela última função **Seek**) será lido. Você deve incluir as vírgulas delimitadoras, por exemplo:

```
Get #4, ,FileBuffer
```

Para os arquivos abertos no modo **Random**, são aplicadas as regras a seguir:

- Se o comprimento dos dados que estão sendo lidos for menor que o comprimento especificado na cláusula **Len** da instrução **Open**, **Get** lê os registros subseqüentes nos limites de comprimento dos registros. O espaço entre o final de um registro e o começo do próximo é preenchido com o conteúdo existente do buffer do arquivo. Como a quantidade dos dados de preenchimento não pode ser determinada com segurança, é aconselhável fazer com que o tamanho do registro corresponda ao comprimento dos dados que estão sendo lidos.
- Se a variável que está sendo lida for uma seqüência de caracteres de comprimento variável, **Get** lê um descritor de 2 bytes contendo o comprimento da seqüência de caracteres e, em seguida, lê os dados da variável. Entretanto, o comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser pelo menos 2 bytes maior do que o comprimento real da seqüência de caracteres.
- Se a variável que está sendo lida for um **Variant** de tipo numérico, **Get** lerá 2 bytes que identificam o **VarType** de **Variant** e, em seguida, os dados da variável. Por exemplo, ao ler um **Variant** de **VarType** 3, **Get** lê 6 bytes: 2 bytes que identificam o **Variant** como **VarType** 3 (**Long**) e 4 bytes que contêm os dados **Long**. O comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser pelo menos 2 bytes maior do que o número de bytes real necessário para armazenar a variável.

**Observação** Você pode utilizar a instrução **Get** para ler uma matriz Variant a partir do disco, mas não pode utilizar **Get** para ler um **Variant** escalar contendo uma matriz. Você não pode também utilizar **Get** para ler os objetos a partir do disco.

- Se a variável que está sendo lida for um **Variant** de **VarType** 8 (**String**), **Get** lê 2 bytes que identificam o **VarType**, 2 bytes que indicam o comprimento da seqüência de caracteres, e, em seguida, lê os dados da seqüência de caracteres. O comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser pelo menos 4 bytes maior do que o comprimento real da seqüência de caracteres.
- Se a variável que está sendo lida for uma matriz dinâmica, **Get** lê um descritor cujo comprimento é igual a 2 mais 8 vezes o número de dimensões, ou seja,  $2 + 8 * \text{NúmeroDeDimensões}$ . O comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser maior ou igual à soma de todos os bytes necessários para ler os dados da matriz e o descritor da matriz. Por exemplo, a declaração de matriz a seguir exige 118 bytes quando a matriz é gravada no disco:

```
Dim MinhaMatriz(1 To 5,1 To 10) As Integer
```

Os 118 bytes são distribuídos da seguinte maneira: 18 bytes para o descritor ( $2 + 8 * 2$ ) e 100 bytes para os dados ( $5 * 10 * 2$ ).

- Se a variável que está sendo lida for uma matriz de tamanho fixo, o **Get** lerá somente os dados. Nenhum descritor é lido.
- Se a variável que está sendo lida for de qualquer outro tipo (não for uma seqüência de caracteres de comprimento variável ou uma **Variant**), o **Get** lê somente os dados da variável. O comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser maior ou igual ao comprimento dos dados que estão sendo lidos.
- **Get** lê os elementos dos tipos definidos pelo usuário como se cada um estivesse sendo lido individualmente, com exceção de que não há preenchimento algum entre os elementos. No disco, uma matriz dinâmica em um tipo definido pelo usuário (gravado com **Put**) é prefixada por um descritor cujo comprimento é igual a 2 mais 8 vezes o número de dimensões, ou seja,  $2 + 8 * \text{NúmeroDeDimensões}$ . O comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser maior ou igual à soma de todos os bytes necessários para ler os elementos individuais, incluindo as matrizes e seus descritores.

Para arquivos abertos no modo **Binary**, todas as regras de **Random** são aplicadas, exceto:

- A cláusula **Len** na instrução **Open** não possui efeito. **Get** lê todas as variáveis a partir do disco de forma contígua; ou seja, sem preenchimento entre os registros.
- Para qualquer matriz que não tenha sido definida pelo usuário, **Get** lê somente os dados. Nenhum descritor é lido.
- **Get** lê as seqüências de caracteres de comprimento variável que não sejam elementos de tipos definidos pelo usuário sem esperar pelo descritor de 2 bytes de comprimento. O número de bytes lidos é igual ao número de caracteres que já se encontram na seqüência de caracteres. Por exemplo, as instruções a seguir lêem 10 bytes do número de arquivo 1:

```
SeqVar = String(10, " ")
Get #1,,SeqVar
```

## Instrução Input #

Lê os dados de um arquivo seqüencial aberto e atribui os dados às variáveis.

### Sintaxe

**Input #** *número do arquivo*, *lista de var*

A sintaxe da instrução **Input #** possui as partes a seguir:

Parte	Descrição
<i>número do arquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.
<i>lista de var</i>	Obrigatório. Lista de variáveis delimitada por vírgulas às quais são atribuídos os valores lidos do arquivo – não pode ser uma <u>matriz</u> ou <u>variável de objeto</u> . Contudo, as variáveis que descrevem um elemento de uma matriz ou um <u>tipo definido pelo usuário</u> podem ser utilizadas.

### Comentários

Os dados lidos com **Input #** são geralmente gravados em um arquivo com **Write #**. Utilize essa instrução somente com arquivos abertos no modo **Input** ou **Binary**.

Quando lidos, seqüências de caracteres padrão ou dados numéricos são atribuídos às variáveis sem modificação. A tabela a seguir ilustra como outros dados de saída são tratados:

Dados	Valor atribuído à variável
Vírgula delimitadora ou linha em branco	<b>Empty</b>
#NULL#	<b>Null</b>
#TRUE# ou #FALSE#	<b>True</b> ou <b>False</b>
#dd-mm-aaaa hh:mm:ss#	A data e/ou hora representada pela <u>expressão</u>
#ERROR número de erro#	<u>número de erro</u> (variável é um <b>Variant</b> marcado como um erro)

As aspas duplas (" ") dentro dos dados de entrada são ignoradas.

Os itens dos dados em um arquivo devem aparecer na mesma ordem que as variáveis em *lista de var* e corresponderem às variáveis do mesmo tipo de dados. Se uma variável for numérica e os dados não forem, um valor zero é atribuído à variável.

Se o fim do arquivo é atingido enquanto um item de dados estiver sendo inserido, a inserção é interrompida e um erro é gerado.

**Observação** Para possibilitar a leitura correta dos dados em variáveis a partir de um arquivo com **Input #**, utilize a instrução **Write #** em vez da instrução **Print #** para gravar os dados nos arquivos. A utilização de **Write #** assegura que cada campo de dados separado seja delimitado de forma apropriada.

## Função Input

Retorna **String** contendo os caracteres de um arquivo aberto no modo **Input** ou **Binary**.

### Sintaxe

**Input**(*número*, [#]*númerodoarquivo*)

A sintaxe da função **Input** possui as partes a seguir:

Parte	Descrição
<i>número</i>	Obrigatório. Qualquer <u>expressão numérica</u> válida especificando o número de caracteres a serem retornados.
<i>númerodoarquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.

### Comentários

Os dados lidos com a função **Input** são normalmente gravados em um arquivo com **Print #** ou **Put**. Utilize essa função apenas com arquivos abertos no modo **Input** ou **Binary**.

Ao contrário da instrução **Input #**, a função **Input** retorna todos os caracteres que lê, incluindo vírgulas, retornos de carro, alimentações de linha, aspas e espaços à esquerda.

Com arquivos abertos para acesso **Binary**, uma tentativa de se ler o arquivo utilizando a função **Input** até que **EOF** retorne **True** gera um erro. Utilize as funções **LOF** e **Loc** em vez de **EOF** quando estiver lendo arquivos binários com **Input** ou utilize **Get** quando utilizar a função **EOF**.

**Observação** Utilize a função **InputB** para dados de bytes contidos em arquivos de texto. Com **InputB**, *número* especifica o número de bytes a serem retornados em vez do número de caracteres a serem retornados.

## Instrução Line Input #

Lê uma única linha a partir de um arquivo seqüencial aberto e a atribui a uma variável **String**.

### Sintaxe

**Line Input #***númerodoarquivo*, *nomedavar*

A sintaxe da instrução **Line Input #** possui as seguintes partes:

Parte	Descrição
<i>númerodoarquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.
<i>nomedavar</i>	Obrigatório. Nome da variável <b>Variant</b> ou <b>String</b> válido.

### Comentários

Os dados lidos com **Line Input #** são geralmente gravados a partir de um arquivo com **Print #**.

A instrução **Line Input #** lê um caractere do arquivo de cada vez, até encontrar um retorno de carro (**Chr(13)**) ou uma seqüência de retorno de carro – alimentação de linha (**Chr(13)** + **Chr(10)**). As seqüências de retorno de carro – alimentação de linha são ignoradas em vez de serem anexadas à seqüência de caracteres.

## Função Loc

Retorna um **Long** especificando a posição de leitura/gravação atual dentro de um arquivo aberto.

### Sintaxe

**Loc**(*númerodoarquivo*)

O argumento obrigatório *númerodoarquivo* pode ser qualquer número de arquivo **Integer** válido.

### Comentários

A tabela abaixo descreve o valor de retorno para cada modo de acesso a arquivo:

<b>Modo</b>	<b>Valor de retorno</b>
<b>Random</b>	Número do último registro lido de ou gravado no arquivo.
<b>Sequential</b>	Posição atual do byte no arquivo dividido por 128. Entretanto, as informações retornadas por <b>Loc</b> para os arquivos seqüenciais não são utilizadas nem exigidas.
<b>Binary</b>	Posição do último byte lido ou gravado.

## Instruções Lock e Unlock

Controla o acesso através de outros processos para todo ou parte de um arquivo aberto com a instrução **Open**.

### Sintaxe

**Lock** [*#*]númerodoarquivo[, *intervalodoregistro*]

**Unlock** [*#*]númerodoarquivo[, *intervalodoregistro*]

A sintaxe das instruções **Lock** e **Unlock** possui as partes a seguir:

<b>Parte</b>	<b>Descrição</b>
<i>númerodoarquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.
<i>intervalodoregistro</i>	Opcional. O intervalo de registros a serem bloqueados ou desbloqueados.

### Definições

As definições do argumento *intervalodoregistro* são:

*númerodoreg* | [*início*] **To** *fim*

<b>Definição</b>	<b>Descrição</b>
<i>númerodoreg</i>	Número do registro (arquivos no modo <b>Random</b> ) ou número de byte (arquivos no modo <b>Binary</b> ) no qual o bloqueio ou desbloqueio é iniciado.
<i>início</i>	Número do primeiro registro ou byte a ser bloqueado ou desbloqueado.
<i>fim</i>	Número do último registro ou byte a ser bloqueado ou desbloqueado.

### Comentários

As instruções **Lock** e **Unlock** são utilizadas em ambientes onde vários processos podem precisar ter acesso ao mesmo arquivo.

As instruções **Lock** e **Unlock** são sempre utilizadas aos pares. Os argumentos para **Lock** e **Unlock** devem ter correspondência exata.

O primeiro registro ou byte em um arquivo encontra-se na posição 1, o segundo registro ou byte na

posição 2 e assim por diante. Se você especificar apenas um registro, somente esse registro será bloqueado ou desbloqueado. Se você especificar um intervalo de registros e omitir um registro inicial (*início*), todos os registros do primeiro até o último (*fim*) do intervalo serão bloqueados ou desbloqueados. A utilização de **Lock** sem *númerodereg* bloqueia todo o arquivo; a utilização de **Unlock** sem *númerodereg* desbloqueia todo o arquivo.

Se o arquivo tiver sido aberto para entrada ou saída seqüencial, **Lock** e **Unlock** afetarão todo o arquivo, independente do intervalo especificado por *início* e *fim*.

---

**Atenção** Certifique-se de remover todos os bloqueios com uma instrução **Unlock** antes de fechar um arquivo ou sair de seu programa. Caso contrário, resultados imprevisíveis poderão ser provocados.

---

## Função LOF

Retorna um **Long** representando o tamanho, em bytes, de um arquivo aberto com a instrução **Open**.

### Sintaxe

**LOF**(*númerodoarquivo*)

O argumento obrigatório *númerodoarquivo* é um **Integer** que contém um número de arquivo válido.

**Observação** Utilize a função **FileLen** para obter o comprimento de um arquivo que não está aberto.

## Instrução Open

Possibilita a entrada/saída (E/S) para um arquivo.

### Sintaxe

**Open** *nomedocaminho* **For** *modo* [**Access** *acesso*] [*bloqueio*] **As** [#]*númerodoarquivo*  
 [**Len**=*comprimentodereg*]

A sintaxe da instrução **Open** possui as partes a seguir:

<u>Parte</u>	<u>Descrição</u>
<i>nomedocaminho</i>	Obrigatório. <u>Expressão de seqüência</u> que especifica um nome de arquivo—pode incluir um diretório ou pasta e unidade de disco.
<i>modo</i>	Obrigatório. <u>Palavra-chave</u> especificando o modo de arquivo: <b>Append</b> , <b>Binary</b> , <b>Input</b> , <b>Output</b> ou <b>Random</b> . Se não for especificado, o arquivo é aberto para acesso <b>Random</b> .
<i>acesso</i>	Opcional. Palavra-chave especificando as operações permitidas no arquivo aberto: <b>Read</b> , <b>Write</b> ou <b>Read Write</b> .
<i>bloqueio</i>	Opcional. Palavra-chave especificando as operações permitidas no arquivo aberto por outros processos: <b>Shared</b> , <b>Lock Read</b> , <b>Lock Write</b> e <b>Lock Read Write</b> .
<i>númerodoarquivo</i>	Obrigatório. Um <u>número de arquivo</u> válido no intervalo de 1 a 511, inclusive. Utilize a função <b>FreeFile</b> para obter o próximo número de arquivo disponível.
<i>comprimentodereg</i>	Opcional. Número menor ou igual a 32.767 (bytes). Para arquivos abertos para acesso aleatório, esse valor é o comprimento do registro. Para arquivos seqüenciais, esse valor é o número de caracteres presentes no buffer.

## Comentários

Você deve abrir um arquivo antes que nele seja realizada qualquer operação de E/S. **Open** aloca um buffer para E/S para o arquivo e determina o modo de acesso a ser utilizado com o buffer.

Se o arquivo especificado por *nomedocaminho* não existir, ele será criado quando um arquivo for aberto para os modos **Append**, **Binary**, **Output** ou **Random**.

Se o arquivo já se encontra aberto por um outro processo e o tipo de acesso especificado não for permitido, a operação **Open** falhará e um erro será gerado.

A cláusula **Len** é ignorada se *modo* for **Binary**.

**Importante** Nos modos **Binary**, **Input** e **Random**, você pode abrir um arquivo utilizando um número de arquivo diferente sem ter que antes fechar o arquivo. Nos modos **Append** e **Output**, você deve fechar um arquivo antes de abri-lo com um número de arquivo diferente.

## Instrução Print #

Grava os dados formatados para tela em um arquivo seqüencial.

### Sintaxe

**Print #***número do arquivo*, [*listadesaída*]

A sintaxe da instrução **Print #** possui as partes a seguir:

Parte	Descrição
<i>número do arquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.
<i>listadesaída</i>	Opcional. <u>Expressão</u> ou lista de expressões a ser impressa.

### Definições

As definições do argumento listadesaída são:

[**Spc**(*n*) | **Tab**(*n*)] [*expressão*] [*posdocarac*]

Definição	Descrição
<b>Spc</b> ( <i>n</i> )	Utilizado para inserir caracteres de espaço na saída, onde <i>n</i> é o número de caracteres de espaço a serem inseridos.
<b>Tab</b> ( <i>n</i> )	Utilizado para posicionar o ponto de inserção em um número absoluto de coluna, onde <i>n</i> é o número da coluna. Utilize <b>Tab</b> sem argumento para posicionar o ponto de inserção no começo da <u>próxima zona de impressão</u> .
<i>expressão</i>	<u>Expressões numéricas</u> ou <u>expressões de seqüência de caracteres</u> a serem impressas.
<i>posdocarac</i>	Especifica o ponto de inserção para o próximo caractere. Utilize um ponto-e-vírgula para posicionar o ponto de inserção imediatamente após o último caractere exibido. Utilize <b>Tab</b> ( <i>n</i> ) para posicionar o ponto de inserção em um número absoluto de coluna. Utilize <b>Tab</b> sem argumento para posicionar o ponto de inserção no início da próxima zona de impressão. Se <i>posdocarac</i> for omitido, o próximo caractere será impresso na próxima linha.

### Comentários

Os dados gravados com **Print #** são geralmente lidos a partir de um arquivo com **Line Input #** ou **Input**.

Se você omitir *listadesaída* e incluir apenas um separador de lista após *número do arquivo*, uma linha

em branco será impressa no arquivo. Várias expressões podem ser separadas com um espaço ou um ponto-e-vírgula. Um espaço possui o mesmo efeito de um ponto-e-vírgula.

Para dados **Boolean**, `True` ou `False` são impressos. As palavras-chave **True** e **False** não são traduzidas, independente da localidade.

Os dados **Date** são gravados no arquivo utilizando o formato padrão curto de datas reconhecido pelo seu sistema. Quando o componente data ou hora não estiver presente ou for zero, somente a parte fornecida é gravada no arquivo.

Se os dados de *listadesaída* forem **Empty**., nada será gravado no arquivo. Contudo, se *listadesaída* for **Null**, **Null** é gravado no arquivo.

Para os dados de **Error**, a saída aparece como `Error códigoerro`. A palavra-chave **Error** não é traduzida independente da localidade.

Todos os dados gravados no arquivo com **Print #** são reconhecidos internacionalmente; ou seja, os dados são formatados de forma correta com o separador decimal apropriado.

Como **Print #** grava uma imagem dos dados ao arquivo, você deve delimitar os dados para que eles sejam impressos corretamente. Se você utilizar **Tab** sem argumentos para mover a posição da impressão para a próxima zona de impressão, **Print #** também gravará os espaços entre os campos de impressão no arquivo.

**Observação** Se, no futuro, você desejar ler os dados de um arquivo utilizando a instrução **Input #**, utilize a instrução **Write #** ao invés da instrução **Print #** para gravar os dados no arquivo. A utilização de **Write #** garante a integridade de cada campo de dados em particular, delimitando-os do modo adequado, para que possam ser lidos de volta utilizando **Input #**. A utilização de **Write #** também assegura que os dados possam ser lidos corretamente em qualquer localidade.

## Instrução Put

Grava os dados a partir de uma variável para um arquivo em disco.

### Sintaxe

**Put** [#]númerodoarquivo, [númerodoreg], nomedavar

A sintaxe da instrução **Put** possui as partes a seguir:

Parte	Descrição
<i>númerodoarquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.
<i>númerodoreg</i>	Opcional. <b>Variant (Long)</b> . Número do registro (arquivos no modo <b>Random</b> ) ou número do byte (arquivos no modo <b>Binary</b> ) no qual a gravação é iniciada.
<i>nomedavar</i>	Obrigatório. Nome da variável contendo os dados a serem gravados no disco.

### Comentários

Os dados gravados com **Put** são geralmente lidos de um arquivo com **Get**.

O primeiro registro ou byte em um arquivo encontra-se na posição 1, o segundo registro ou byte na posição 2 e assim por diante. Se *númerodoreg* for omitido, o próximo registro ou byte após a última instrução **Get** ou **Put** ou aquele apontado pela última função **Seek** será gravado. Você deve incluir as vírgulas de delimitação, por exemplo:

```
Put #4,,FileBuffer
```

Para os arquivos abertos no modo **Random**, são aplicadas as regras a seguir:

- Se o comprimento dos dados que estão sendo gravados for menor do que o comprimento especificado na cláusula **Len** da instrução **Open**, **Put** grava os registros subseqüentes nos limites do comprimento do registro. O espaço entre o final de um registro e o início do próximo registro é preenchido com o conteúdo existente do buffer do arquivo. Como a quantidade dos dados de

preenchimento não pode ser determinada com segurança, convém fazer com que o comprimento do registro corresponda ao dos dados que estão sendo gravados. Se o comprimento dos dados que estão sendo gravados for maior do que o comprimento especificado na cláusula **Len** da instrução **Open**, um erro será gerado.

- Se a variável que está sendo gravada for uma seqüência de caracteres de comprimento variável, **Put** grava um descritor de 2 bytes contendo o comprimento da seqüência e, em seguida, a variável. O comprimento do registro sendo especificado pela cláusula **Len** na instrução **Open** deve ser pelo menos 2 bytes maior do que o comprimento real da seqüência de caracteres.
- Se a variável que está sendo gravada for um **Variant** de um tipo numérico, **Put** grava 2 bytes identificando o **VarType** do **Variant** e, em seguida, grava a variável. Por exemplo, ao gravar um **Variant** de **VarType 3**, **Put** grava 6 bytes: 2 bytes identificando o **Variant** como **VarType 3 (Long)** e 4 bytes contendo os dados **Long**. O comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser pelo menos 2 bytes maior do que o número real de bytes necessários para armazenar a variável.

**Observação** Você pode utilizar a instrução **Put** para gravar uma matriz Variant em disco, mas não pode utilizar **Put** para gravar em disco um **Variant** escalar contendo uma matriz. **Put** também não pode ser utilizado para gravar objetos em disco.

- Se a variável que está sendo gravada for um **Variant** de **VarType 8 (String)**, **Put** grava 2 bytes identificando **VarType**, 2 bytes indicando o comprimento da seqüência de caracteres e, em seguida, grava os dados da seqüência. O comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser pelo menos 4 bytes maior do que o comprimento real da seqüência de caracteres.

Se a variável que está sendo gravada for uma matriz dinâmica, **Put** grava um descritor cujo comprimento é igual a 2 mais 8 vezes o número de dimensões, ou seja,  $2 + 8 * \text{NúmeroDeDimensões}$ . O comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser maior ou igual à soma de todos os bytes necessários para gravar os dados da matriz e o descritor da matriz. Por exemplo, a declaração da matriz a seguir exige 118 bytes quando a matriz é gravada em disco.

```
Dim MinhaMatriz (1 To 5,1 To 10) As Integer
```

- Os 118 bytes são distribuídos da seguinte forma: 18 bytes para o descritor ( $2 + 8 * 2$ ) e 100 bytes para os dados ( $5 * 10 * 2$ ).
- Se a variável que está sendo gravada for uma matriz de tamanho fixo, **Put** grava somente os dados. Nenhum descritor é gravado no disco.
- Se a variável que está sendo gravada for qualquer outro tipo de variável (não uma seqüência de caracteres de comprimento variável ou um **Variant**), **Put** grava somente os dados da variável. O comprimento do registro especificado pela cláusula **Len** na instrução **Open** deve ser maior ou igual ao comprimento dos dados que estão sendo gravados.
- **Put** grava os elementos dos tipos definidos pelo usuário como se cada um fosse gravado individualmente, com exceção de que não há preenchimento entre os elementos. No disco, uma matriz dinâmica de um tipo definido pelo usuário gravada com **Put** é prefixada por um descritor cujo comprimento é igual a 2 mais 8 vezes o número de dimensões, ou seja,  $2 + 8 * \text{NúmeroDeDimensões}$ . O comprimento do registro especificado pela cláusula **Len** na instrução **Open** pode ser maior ou igual à soma de todos os bytes necessários para gravar os elementos individuais, incluindo as matrizes e seus descritores.

Para os arquivos abertos no modo **Binary**, são aplicadas todas as regras de **Random**, exceto:

- A cláusula **Len** na instrução **Open** não possui efeito. **Put** grava todas as variáveis no disco de forma contígua; ou seja, sem preenchimento entre os registros.
- Para qualquer outra matriz diferente daquela de tipo definido pelo usuário, **Put** grava somente os dados. Nenhum descritor é gravado.
- **Put** grava as seqüências de caracteres de comprimento variável que não sejam elementos dos tipos definidos pelo usuário sem o descritor de comprimento de 2 bytes. O número de bytes gravados é igual ao número de caracteres da seqüência. Por exemplo, as instruções a seguir gravam 10 bytes em um arquivo de número 1:

```
VarString$ = String$(10, " ")
Put #1, ,VarString$
```

## Instrução Reset

Fecha todos os arquivos em disco abertos com a instrução **Open**.

### Sintaxe

#### Reset

### Comentários

A instrução **Reset** fecha todos os arquivos ativos abertos pela instrução **Open** e grava em disco o conteúdo de todos os buffers de arquivo.

## Função Seek

Retorna um **Long** especificando a posição atual de leitura/gravação dentro de um arquivo aberto utilizando-se a instrução **Open**.

### Sintaxe

**Seek**(*número do arquivo*)

O argumento obrigatório *número do arquivo* é um **Integer** contendo um número de arquivo válido.

### Comentários

**Seek** retorna um valor entre 1 e 2.147.483.647 (equivalente a  $2^{31} - 1$ ), inclusive.

A tabela a seguir descreve os valores de retorno para cada modo de acesso a arquivos.

<b>Modo</b>	<b>Valor de retorno</b>
<b>Random</b>	Número do próximo registro lido ou gravado
<b>Binary Output, Append e Input</b>	Posição do byte na qual ocorre a próxima operação. O primeiro byte em um arquivo encontra-se na posição 1, o segundo byte na posição 2 e assim por diante.

## Instrução Seek

Define a posição da próxima operação de leitura/gravação dentro de um arquivo aberto com a instrução **Open**.

### Sintaxe

**Seek** [#]*número do arquivo, posição*

A sintaxe da instrução **Seek** possui as partes a seguir:

<b>Parte</b>	<b>Descrição</b>
<i>número do arquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.
<i>posição</i>	Obrigatório. Número no intervalo de 1 – 2.147.483.647, inclusive, que indica onde a próxima operação de leitura/gravação deve ocorrer.

### Comentários

Os números de registro especificados nas instruções **Get** e **Put** sobrepõem-se à posição de arquivo efetuada por **Seek**.

Realizar uma operação de gravação de arquivo após uma operação **Seek** além do final de um arquivo faz com que o mesmo aumente de tamanho. Se você tentar uma operação **Seek** em uma posição negativa ou zero, um erro será gerado.

## Função Spc

Utilizada com a instrução **Print #** ou o método **Print** para posicionar a saída.

### Sintaxe

#### Spc(*n*)

O argumento obrigatório *n* é o número de espaços a serem inseridos antes da exibição ou impressão da próxima expressão em uma lista.

### Comentários

Se *n* for menor que a largura da linha de saída, a próxima posição de impressão seguirá imediatamente o número de espaços impressos. Se *n* for maior que a largura da linha de saída, **Spc** calculará a próxima posição de impressão utilizando a fórmula:

$$\text{posiçãoatualdaimpressão} + (n \text{ Mod } \text{largura})$$

Por exemplo, se a posição atual da impressão for 24, a largura da linha de saída for 80 e você especificar **Spc(90)**, a próxima impressão começará na posição 34 (a posição atual de impressão + o resto de 90/80). Se a diferença entre a posição atual da impressão e a largura da linha de saída for menor do que *n* (ou *n Mod largura*), a função **Spc** passará para o início da próxima linha e gerará espaços iguais a *n - (largura - posiçãoatualdaimpressão)*.

**Observação** Certifique-se de que suas colunas tabulares sejam largas o suficiente para acomodarem letras largas.

Quando você utiliza o método **Print** com uma fonte proporcionalmente espaçada, a largura dos caracteres de espaço impressos com a função **Spc** é sempre uma média da largura de todos os caracteres em tamanho de pontos para a fonte escolhida. Contudo, não há correlação entre o número de caracteres impressos e o número de colunas de largura fixa que esses caracteres ocupam. Por exemplo, a letra maiúscula W ocupa mais do que uma coluna de largura fixa e a letra minúscula i ocupa menos.

## Função Tab

Utilizada com a instrução **Print #** ou o método **Print** para posicionar a saída.

### Sintaxe

#### Tab[(*n*)]

O argumento opcional *n* representa o número da coluna que é movido antes da exibição ou impressão da próxima expressão em uma lista. Se for omitido, **Tab** move o ponto de inserção para o início da próxima zona de impressão. Isso permite que **Tab** seja utilizado em vez de uma vírgula nas localidades onde a vírgula é utilizada como um separador decimal.

### Comentários

Se a posição atual de impressão na linha atual for maior do que *n*, **Tab** pula para a coluna de número *n* da próxima linha de saída. Se *n* for menor do que 1, **Tab** move a posição de impressão para a coluna 1. Se *n* for maior do que a largura da linha de saída, **Tab** calcula a próxima posição de impressão utilizando a fórmula:

$$n \text{ Mod } \text{largura}$$

Por exemplo, se a *largura* for 80 e você especificar **Tab(90)**, a próxima impressão começará na coluna 10 (o resto de 90/80). Se *n* for menor do que a posição atual de impressão, a impressão começará na próxima linha na posição de impressão calculada. Se a posição de impressão calculada for maior do que a posição atual de impressão, a impressão começará na posição de impressão calculada na mesma linha.

A posição de impressão mais à esquerda em uma linha de saída é sempre 1. Quando você utiliza a instrução **Print #** para imprimir em arquivos, a posição de impressão mais à direita é a largura atual da linha de saída, que pode ser definida utilizando-se a instrução **Width #**.

**Observação** Certifique-se de que suas colunas tabulares sejam largas o suficiente para acomodarem letras largas.

Quando você utiliza a função **Tab** com o método **Print**, a superfície de impressão é dividida em colunas de largura fixa e uniformes. A largura de cada coluna é uma média da largura de todos os caracteres em tamanho de pontos para a fonte escolhida. Contudo, não há correlação entre o número de caracteres impressos e o número de colunas de largura fixa que esses caracteres ocupam. Por exemplo, a letra maiúscula **W** ocupa mais do que uma coluna de largura fixa e a letra minúscula **i** ocupa menos.

## Instrução Width #

Atribui uma largura de linha de saída a um arquivo aberto com a instrução **Open**.

### Sintaxe

**Width #***númerodoarquivo*, *largura*

A sintaxe da instrução **Width #** possui as partes a seguir:

Parte	Descrição
<i>númerodoarquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.
<i>largura</i>	Obrigatório. <u>Expressão numérica</u> na faixa de 0 a 255, inclusive, que indica a quantidade de caracteres que aparece em uma linha antes que uma nova linha seja iniciada. Se a <i>largura</i> for igual a 0, não haverá limite para o comprimento de uma linha. O valor padrão para a <i>largura</i> é 0.

## Instrução Write #

Grava os dados em um arquivo seqüencial.

### Sintaxe

**Write #***númerodoarquivo*, [*listadesaída*]

A sintaxe da instrução **Write #** possui as partes a seguir:

Parte	Descrição
<i>númerodoarquivo</i>	Obrigatório. Qualquer <u>número de arquivo</u> válido.
<i>listadesaída</i>	Opcional. Uma ou mais <u>expressões numéricas</u> delimitadas por vírgula ou <u>expressões de seqüência de caracteres</u> a serem gravadas em um arquivo.

### Comentários

Os dados gravados com **Write #** são geralmente lidos a partir de um arquivo com **Input #**.

Se você omitir *listadesaída* e incluir uma vírgula após *númerodoarquivo*, uma linha em branco é impressa no arquivo. Várias expressões podem ser separadas com um espaço, um ponto-e-vírgula ou uma vírgula. Um espaço possui o mesmo efeito que um ponto-e-vírgula.

Quando **Write #** for utilizado para gravar dados em um arquivo, várias hipóteses universais são seguidas para que os dados possam sempre ser lidos e interpretados corretamente utilizando **Input #**, independente da localidade:

- Os dados numéricos são sempre gravados utilizando o ponto como separador decimal.

- Para dados **Boolean**, #TRUE# ou #FALSE# é impresso. As palavras-chave True e False não são traduzidas, independente da localidade.
- Os dados **Date** são gravados no arquivo utilizando o formato de data universal. Quando o componente de data ou hora estiver faltando, ou for igual a zero, somente a parte fornecida é gravada no arquivo.
- Se os dados de *listadesaída* forem **Empty**, nada será gravado no arquivo. Contudo, para dados **Null**, #NULL# é gravado.
- Se os dados de *listadesaída* forem **Null**, #NULL# é gravado no arquivo.
- Para os dados de **Error**, a saída aparece como #ERROR códigoerro#. A palavra-chave **Error** não é traduzida, independente da localidade.

Ao contrário da instrução **Print #**, a instrução **Write #** insere vírgulas entre itens e aspas ao redor de seqüências de caracteres assim que são gravadas no arquivo. Você não precisa colocar delimitadores explícitos na lista. **Write #** insere um caractere de linha nova, ou seja, um retorno de carro–alimentação de linha (**Chr(13) + Chr(10)**), após ter gravado o último caractere de *listadesaída* no arquivo.

### Exemplo da função DDB

Este exemplo utiliza a função **DDB** para retornar a depreciação de um bem por um período especificado dado o custo inicial (*CustoInicial*), o valor do salvamento ao final da vida útil do bem (*ValorDoSalvamento*), a vida total do bem em anos (*VidaÚtil*) e o período em anos para o qual a depreciação é calculada (*Depr*).

```
Dim Fmt, CustoInicial, ValorDoSalvamento, VidaMensal, VidaÚtil, AnoDep,
Depr
Const YRMOS = 12 ' Número de meses em um ano.
Fmt = "###,##0.00"
CustoInicial = InputBox("Qual é o custo inicial do bem?")
ValorDoSalvamento = InputBox("Insira o valor do bem ao fim da sua vida.")
VidaMensal = InputBox("Qual é a vida útil do bem em meses?")
Do While VidaMensal < YRMOS ' Certifique-se de que o período é >= 1 ano.
    MsgBox "A vida do bem deve ser de um ano ou mais".
    VidaMensal = InputBox("Qual é a vida útil do bem em meses?")
Loop
VidaÚtil = VidaMensal / YRMOS ' Converte meses em anos.
If VidaÚtil <> Int(VidaMensal / YRMOS) Then
    VidaÚtil = Int(VidaÚtil + 1) ' Arredonda para o ano mais próximo.
End If
AnoDep = CInt(InputBox("Insira o ano para o cálculo da depreciação."))
Do While AnoDep < 1 Or AnoDep > VidaÚtil
    MsgBox "Você deve inserir no mínimo 1 mas não mais de " & VidaÚtil
    AnoDep = InputBox("Insira o ano para o cálculo da depreciação.")
Loop
Depr = DDB(CustoInicial, ValorDoSalvamento, VidaÚtil, AnoDep)
MsgBox "A depreciação do ano " & AnoDep & " é " & _
Format(Depr, Fmt) & "."
```

### Exemplo da função FV

Este exemplo utiliza a função **FV** para retornar o valor futuro de um investimento dada a taxa percentual que se acumula por período (*APR / 12*), o número total de pagamentos (*TotPgto*s), o pagamento (*Pagamento*), o valor atual do investimento (*ValP*) e um número que indica se o pagamento é feito no início ou no fim do período de pagamento (*TipoPag*). Observe que, como *Pagamento* representa dinheiro pago, é um número negativo.

```
Dim Fmt, Pagamento, TPA, TotPgto, TipoPag, ValP, ValF
```

```

Const PERÍODOFINAL = 0, PERÍODOINICIAL = 1      ' Quando os pagamentos são
feitos.
Fmt = "###,###,##0.00"      ' Define o formato monetário.
Pagamento = InputBox("Quanto você pretende economizar por mês?")
TPA = InputBox("Insira a taxa de percentagem anual de juros esperados.")
If TPA > 1 Then TPA = TPA / 100      ' Certifica-se de que se trata do devido
formulário.
TotPgtos = InputBox("Durante quantos meses espera fazer economia?")
TipoPag = MsgBox("Você faz os pagamentos no fim do mês?", vbYesNo)
If TipoPag = vbNo Then TipoPag = PERÍODOINICIAL Else TipoPag = PERÍODOFINAL
ValP = InputBox("Quanto há na conta de economias agora?")
ValF = FV(TPA / 12, TotPgtos, -Pagamento, -ValP, TipoPag)
MsgBox "Suas economias serão valiosos " & Format(ValF, Fmt) & "."

```

### Exemplo da função IPmt

Este exemplo utiliza a função **IPmt** para calcular quanto de um pagamento é juro quando todos os pagamentos são de igual valor. Fornecidos a taxa percentual dos juros por período ( $TPA / 12$ ), o período de pagamento para o qual se deseja a parte dos juros (Período), o número total de pagamentos (TotPgtos), o valor presente ou principal do empréstimo (ValP), o valor futuro do empréstimo (ValF) e um número que indica se o pagamento vence no início ou no final do período de pagamento (TipoPag).

```

Dim ValF, Fmt, ValP, TPA, TotPgtos, TipoPag, Período, PgtoJuros, TotJuros,
Msg
Const PERÍODOFINAL = 0, PERÍODOINICIAL = 1      ' Quando os pagamentos são
feitos.
ValF = 0      ' Normalmente 0 para um empréstimo.
Fmt = "###,###,##0.00"      ' Define o formato monetário.
ValP = InputBox("Quanto você quer pegar emprestado?")
TPA = InputBox("Qual a taxa de percentagem anual do seu empréstimo?")
If TPA > 1 Then TPA = TPA / 100      ' Certifica-se de que se trata do
formulário apropriado.
TotPgtos = InputBox("Quantos pagamentos mensais?")
TipoPag = MsgBox("Você faz os pagamentos no fim do mês?", vbYesNo)
If TipoPag = vbNo Then TipoPag = PERÍODOINICIAL Else TipoPag = PERÍODOFINAL
For Período = 1 To TotPgtos      ' Totaliza os juros.
    PgtoJuros = IPmt(TPA / 12, Período, TotPgtos, -ValP, ValF, TipoPag)
    TotJuros = TotJuros + PgtoJuros
Next Período
Msg = "Você vai pagar um total de " & Format(TotJuros, Fmt)
Msg = Msg & " em juros por este empréstimo."
MsgBox Msg      ' Exibe os resultados.

```

**Exemplo da função IRR**

Neste exemplo, a função **IRR** retorna a taxa interna de retorno para uma série de 5 fluxos de caixa contidos na matriz `Valores()`. O primeiro elemento da matriz é um fluxo de caixa negativo que representa os custos iniciais do negócio. Os 4 fluxos de caixa restantes representam fluxos de caixas positivos para os quatro anos subseqüentes. `Estimativa` é a taxa interna de retorno estimada.

```
Dim Estimativa, Fmt, TaxaRet, Msg
Static Valores(5) As Double ' Configura a matriz.
Estimativa = .1 ' Inicia a estimativa em 10 por cento.
Fmt = "#0.00" ' Define o formato da percentagem.
Valores(0) = -70000 ' Custos iniciais do negócio.
' Fluxo de caixa positivo refletindo a receita por quatro anos sucessivos.
Valores(1) = 22000: Valores(2) = 25000
Valores(3) = 28000: Valores(4) = 31000
TaxaRet = IRR(Valores(), Estimativa) * 100 ' Calcula a taxa interna.
Msg = "A taxa interna de retorno para estes cinco fluxos de caixa é"
Msg = Msg & Format(TaxaRet, Fmt) & " por cento."
MsgBox Msg ' Exibe a taxa interna de retorno.
```

**Exemplo da função MIRR**

Este exemplo utiliza a função **MIRR** para retornar a taxa interna de retorno modificada de uma série de fluxos de caixa contidos na matriz `Valores()`. `TPAEmpréstimo` representa os juros financeiros e `TPAInv` representa a taxa de juros recebida no reinvestimento.

```
Dim TPAEmpréstimo, TPAInv, Fmt, TaxaRet, Msg
Static Valores(5) As Double ' Configura a matriz.
TPAEmpréstimo = .1 ' Taxa de empréstimo.
TPAInv = .12 ' Taxa de reinvestimento.
Fmt = "#0.00" ' Define o formato monetário.
Valores(0) = -70000 ' Custo inicial do negócio.
' Fluxo de caixa positivo refletindo a receita por quatro anos sucessivos.
Valores(1) = 22000: Valores(2) = 25000
Valores(3) = 28000: Valores(4) = 31000
TaxaRet = MIRR(Valores(), TPAEmpréstimo, TPAInv) ' Calcula a taxa
interna.
Msg = "A taxa interna de retorno modificada para estes cinco fluxos de
caixa é"
Msg = Msg & Format(Abs(TaxaRet) * 100, Fmt) & "%."
MsgBox Msg ' Exibe a taxa de retorno
' interna.
```

**Exemplo da função NPer**

Este exemplo utiliza a função **NPer** para retornar o número de períodos durante os quais os pagamentos devem ser feitos para liquidar um empréstimo cujo valor está contido em `ValP`. Também são fornecidos a taxa percentual de juros por período (`TPA / 12`), o pagamento (`Pagamento`), o valor futuro do empréstimo (`ValF`) e o número que indica se o pagamento vence no início ou no final do período de pagamento (`TipoPag`).

```
Dim ValF, ValP, TPA, Pagamento, TipoPag, TotPgts
Const PERÍODOFINAL = 0, PERÍODOINICIAL = 1 ' Quando os pagamentos são
feitos.
ValF = 0 ' Normalmente 0 para um empréstimo.
ValP = InputBox("Quanto você quer pegar emprestado?")
TPA = InputBox("Qual é a taxa de percentagem anual do seu empréstimo?")
If TPA > 1 Then TPA = TPA / 100 ' Certifica-se de que se trata do
formulário apropriado.
```

```
Pagamento = InputBox("Quanto você deseja pagar por mês?")
TipoPag = MsgBox("Você faz os pagamentos no fim do mês?", vbYesNo)
If TipoPag = vbNo Then TipoPag = PERÍODOINICIAL Else TipoPag = PERÍODOFINAL
TotPgts = NPer(TPA / 12, -Pagamento, ValP, ValF, TipoPag)
If Int(TotPgts) <> TotPgts Then TotPgts = Int(TotPgts) + 1
MsgBox "Você levará " & TotPgts & " meses para liquidar o seu empréstimo."
```

### Exemplo da função NPV

Este exemplo utiliza a função **NPV** para retornar o valor presente líquido de uma série de fluxos de caixa contidos na matriz `Valores()`. `TaxaRet` representa a taxa interna fixa de retorno.

```
Dim Fmt, Estimativa, TaxaRet, ValPLíquido, Msg
Static Valores(5) As Double ' Configura a matriz.
Fmt = "###,##0.00" ' Define o formato monetário.
Estimativa = .1 ' Inicia a estimativa em 10 por cento.
TaxaRet = .0625 ' Define a taxa interna fixa.
Valores(0) = -70000 ' Custo inicial do negócio.
' Fluxos de caixa positivos refletindo a receita por quatro anos
sucessivos.
Valores(1) = 22000: Valores(2) = 25000
Valores(3) = 28000: Valores(4) = 31000
ValPLíquido = NPV(TaxaRet, Valores()) ' Calcula o valor presente
líquido.
Msg = "O valor presente líquido destes fluxos de caixa é "
Msg = Msg & Format(ValPLíquido, Fmt) & "."
MsgBox Msg ' Exibe o valor presente líquido.
```

### Exemplo da função Pmt

Este exemplo utiliza a função **Pmt** para retornar o pagamento mensalmente de um empréstimo durante um período fixo. Fornecida a taxa percentual de juros por período (`TPA / 12`), o número total de pagamentos (`TotPgts`), o valor presente ou principal do empréstimo (`ValP`), o valor futuro do empréstimo (`ValF`) e um número que indica se o pagamento vence no início ou final do período de pagamento (`TipoPag`).

```
Dim Fmt, ValF, ValP, TPA, TotPgts, TipoPag, Pagamento
Const PERÍODOFINAL = 0, PERÍODOINICIAL = 1 ' Quando os pagamentos são
feitos.
Fmt = "###,###,##0.00" ' Define o formato monetário.
ValF = 0 ' Normalmente 0 para um empréstimo.
ValP = InputBox("Quanto você quer pegar emprestado?")
TPA = InputBox("Qual é a taxa percentual anual do seu empréstimo?")
If TPA > 1 Then TPA = TPA / 100 ' Certifica-se de que se trata do
formulário apropriado.
TotPgts = InputBox("Quantos pagamentos mensais você vai fazer?")
TipoPag = MsgBox("Você faz os pagamentos no fim do mês?", vbYesNo)
If TipoPag = vbNo Then TipoPag = PERÍODOINICIAL Else TipoPag = PERÍODOFINAL
Pagamento = Pmt(TPA / 12, TotPgts, -ValP, ValF, TipoPag)
MsgBox "O seu pagamento será " & Format(Pagamento, Fmt) & " por mês."
```

**Exemplo da função PPmt**

Este exemplo utiliza a função **PPmt** para calcular quanto de um pagamento por um período específico é principal quando todos os pagamentos são de igual valor. Fornecidos a taxa percentual de juros por período (TPA / 12), o período de pagamento para o qual se deseja a parte principal (Período), o número total de pagamentos (TotPgts), o valor presente ou principal do empréstimo (ValP), o valor futuro do empréstimo (ValF) e um número que indica se o pagamento vence no início ou no final do período de pagamento (TipoPag).

```
Dim NL, TB, Fmt, ValF, ValP, TPA, TotPgts, TipoPag, Pagamento, Msg,
  CriaGráfico, Período, P, I
Const PERÍODOFINAL = 0, PERÍODOINICIAL = 1      ' Quando os pagamentos são
  feitos.
NL = Chr(13) & Chr(10)  ' Define a nova linha.
TB = Chr(9)  ' Define a tabulação.
Fmt = "###,###,##0.00"  ' Define o formato monetário.
ValF = 0  ' Normalmente 0 para um empréstimo.
ValP = InputBox("Quanto você quer pegar emprestado?")
TPA = InputBox("Qual é a taxa de percentagem anual do seu empréstimo?")
If TPA > 1 Then TPA = TPA / 100  ' Certifica-se de que se trata do
  formulário apropriado.
TotPgts = InputBox("Quantos pagamentos mensais você precisa fazer?")
TipoPag = MsgBox("Você faz os pagamentos no fim do mês?", vbYesNo)
If TipoPag = vbNo Then TipoPag = PERÍODOINICIAL Else TipoPag = PERÍODOFINAL
Pagamento = Abs(-Pmt(TPA / 12, TotPgts, ValP, ValF, TipoPag))
Msg = "O seu pagamento mensal é " & Format(Pagamento, Fmt) & ". "
Msg = Msg & "Você gostaria de desdobrar o seu principal e
  Msg = Msg & "os juros por período?"
CriaGráfico = MsgBox(Msg, vbYesNo)  ' Veja se deseja um gráfico.
If CriaGráfico <> vbNo Then
  If TotPgts > 12 Then MsgBox "Somente o primeiro ano será mostrado."
  Msg = "Mês  Pagamento  Principal  Juros" & NL
  For Período = 1 To TotPgts
    If Período > 12 Then Exit For      ' Mostra somente os primeiros 12.
    P = PPmt(TPA / 12, Período, TotPgts, -ValP, ValF, TipoPag)
    P = (Int((P + .005) * 100) / 100)  ' Arredonda o principal.
    I = Pagamento - P
    I = (Int((I + .005) * 100) / 100)  ' Arredonda os juros
    Msg = Msg & Período & TB & Format(Pagamento, Fmt)
    Msg = Msg & TB & Format(P, Fmt) & TB & Format(I, Fmt) & NL
  Next Período
  MsgBox Msg  ' Exibe a tabela de amortização.
End If
```

**Exemplo da função PV**

Neste exemplo, a função **PV** retorna o valor presente de uma anuidade de \$1.000.000 que fornecerá \$50.000 por ano nos próximos 20 anos. São fornecidos a taxa de percentagem anual esperada (TPA), o número total de pagamentos (TotPgts), o valor de cada pagamento (SuaReceita), o valor total do investimento (ValF) e o número que indica se cada pagamento é feito no início ou no final do período de pagamento (TipoPag). Observe que SuaReceita é um número negativo porque representa o dinheiro pago da anuidade a cada ano.

```
Dim Fmt, TPA, TotPgts, SuaReceita, ValF, TipoPag, ValP
Const PERÍODOFINAL = 0, PERÍODOINICIAL = 1      ' Quando os pagamentos são
  feitos.
Fmt = "###,##0.00"  ' Define o formato monetário.
TPA = .0825  ' Taxa de percentagem anual.
TotPgts = 20  ' Número total de pagamentos.
SuaReceita = 50000  ' Receita anual.
```

```

ValF = 1000000 ' Valor futuro.
TipoPag = PERÍODOINICIAL ' Pagamento no início do mês.
ValP = PV(TPA, TotPgts, -SuaReceita, ValF, TipoPag)
MsgBox "O valor presente é " & Format(ValP, Fmt) & "."

```

### Exemplo da função Rate

Este exemplo utiliza a função **Rate** para calcular a taxa de juros de um empréstimo, fornecidos o número total de pagamentos (TotPgts), o valor de pagamento do empréstimo (Pagamento), o valor presente ou principal do empréstimo (ValP), o valor futuro do empréstimo (ValF), um número que indica se o pagamento vence no início ou no final do período de pagamento (TipoPag) e uma estimativa da taxa de juros esperada (Estimativa).

```

Dim Fmt, ValF, Estimativa, ValP, Pagamento, TotPgts, TipoPag, TPA
Const PERÍODOFINAL = 0, PERÍODOINICIAL = 1 ' Quando os pagamentos são
feitos.
Fmt = "##0.00" ' Define o formato percentual.
ValF = 0 ' Normalmente 0 para um empréstimo.
Estimativa = .1 ' Estimativa de 10 por cento.
ValP = InputBox("Quanto você pegou emprestado?")
Pagamento = InputBox("Qual é o seu pagamento mensal?")
TotPgts = InputBox("Quantos pagamentos mensais você precisa fazer?")
TipoPag = MsgBox("Você faz os pagamentos no fim do mês?", _
vbYesNo)
If TipoPag = vbNo Then TipoPag = PERÍODOINICIAL Else TipoPag = PERÍODOFINAL
TPA = (Rate(TotPgts, -Pagamento, ValP, ValF, TipoPag, Estimativa) * 12) *
100
MsgBox "A sua taxa de juros é " & Format(CInt(TPA), Fmt) & " por cento."

```

### Exemplo da função SLN

Este exemplo utiliza a função **SLN** para retornar a depreciação linear de um bem por um único período, fornecido o custo inicial do bem (CustoInicial), o valor do salvamento ao final da vida útil do bem (ValorDoSalvamento) e a vida total do bem em anos (VidaÚtil).

```

Dim Fmt, CustoInicial, ValorDoSalvamento, VidaMensal, VidaÚtil, DeprP
Const ANOMESES = 12 ' Número de meses em um ano.
Fmt = "###,##0.00" ' Define o formato monetário.
CustoInicial = InputBox("Qual é o custo inicial do bem?")
ValorDoSalvamento = InputBox("Qual é o valor do bem ao fim da sua vida
útil?")
VidaMensal = InputBox("Qual é a vida útil do bem em meses?")
Do While VidaMensal < ANOMESES ' Certifica-se de que o período é >= 1
ano.
MsgBox "A vida do bem deve ser de um ano ou mais."
VidaMensal = InputBox("Qual é a vida útil do bem em meses?")
Loop
VidaÚtil = VidaMensal / ANOMESES ' Converte meses em anos.
If VidaÚtil <> Int(VidaMensal / ANOMESES) Then
VidaÚtil = Int(VidaÚtil + 1) ' Arredonda para o ano mais próximo.
End If
DeprP = SLN(CustoInicial, ValorDoSalvamento, VidaÚtil)
MsgBox "A depreciação é " & Format(DeprP, Fmt) & " por ano."

```

### Exemplo da função SYD

Este exemplo utiliza a função **SYD** para retornar a depreciação de um bem por um período especificado, fornecidos o custo inicial do bem (*CustoInicial*), o valor do salvamento ao fim da vida útil do bem (*ValorDoSalvamento*) e a vida total do bem em anos (*VidaÚtil*). O período em anos pelo qual a depreciação é calculada é *DeprP*.

```
Dim Fmt, CustoInicial, ValorDoSalvamento, VidaMensal, VidaÚtil, AnoDep,
DeprP
Const ANOMESES = 12 ' Número de meses em um ano.
Fmt = "###,##0.00" ' Define o formato monetário.
CustoInicial = InputBox("Qual é o custo inicial do bem?")
ValorDoSalvamento = InputBox("Qual é o valor do bem no fim da sua vida?")
VidaMensal = InputBox("Qual é a vida útil do bem em meses?")
Do While VidaMensal < ANOMESES ' Certifica-se de que o período é >= 1
ano.
    MsgBox "A vida do bem deve ser de um ano ou mais."
    VidaMensal = InputBox("Qual é a vida útil do bem em meses?")
Loop
VidaÚtil = VidaMensal / ANOMESES ' Converte os meses em anos.
If VidaÚtil <> Int(VidaMensal / ANOMESES) Then
    VidaÚtil = Int(VidaÚtil + 1) ' Arredonda para o anos mais próximo.
End If
AnoDep = CInt(InputBox("Para qual ano você deseja a depreciação?"))
Do While AnoDep < 1 Or AnoDep > VidaÚtil
    MsgBox "Você deve inserir no mínimo 1 mas não mais de " & VidaÚtil
    AnoDep = CInt(InputBox("Para qual ano você deseja a depreciação?"))
Loop
DeprP = SYD(CustoInicial, ValorDoSalvamento, VidaÚtil, AnoDep)
MsgBox "A depreciação por ano " & AnoDep & " é " & Format(DeprP, Fmt) & "."
```

### Função DDB

Retorna um **Double** que especifica a depreciação de um ativo para um período de tempo específico utilizando o método de balanço de declinação dupla ou algum outro método que você especifique.

#### Sintaxe

**DDB(*cost, salvage, life, period[, factor]*)**

A função **DDB** possui estes argumentos nomeados:

Parte	Descrição
<b>cost</b>	Obrigatório. <b>Double</b> que especifica o custo inicial do ativo.
<b>salvage</b>	Obrigatório. <b>Double</b> que especifica o valor do ativo ao final de sua vida útil.
<b>life</b>	Obrigatório. <b>Double</b> que especifica a duração da vida útil do ativo.
<b>period</b>	Obrigatório. <b>Double</b> que especifica o período em que a depreciação do ativo é calculada.
<b>factor</b>	Opcional. <b>Variant</b> que especifica a taxa na qual o balanço declina. Se for omitido, supõe-se 2 (método de declinação dupla).

#### Comentários

O método de balanço de declinação dupla calcula a depreciação a uma taxa acelerada. A depreciação é mais alta no primeiro período e diminui nos períodos sucessivos.

Os argumentos **life** e **period** devem ser expressos nas mesmas unidades. Por exemplo, se **life** for dado em meses, **period** também deve ser dado em meses. Todos os argumentos devem ser números positivos.

A função **DDB** utiliza a seguinte fórmula para calcular a depreciação para um dado período:

$$\text{Depreciação} / \text{period} = ((\text{cost} - \text{salvage}) * \text{factor}) / \text{life}$$

## Função FV

Retorna um **Double** que especifica o valor futuro de uma anuidade com base em pagamentos fixos periódicos e uma taxa de juros fixa.

### Sintaxe

**FV**(rate, nper, pmt[, pv[, type]])

A função **FV** tem os seguintes argumentos nomeados:

Parte	Descrição
<b>rate</b>	Obrigatório. <b>Double</b> que especifica a taxa de juros por período. Por exemplo, se você obtiver um financiamento para um carro com uma taxa de porcentagem anual (TPA) de 10 por cento e efetuar pagamentos mensais, a taxa por período será de 0,1/12 ou 0,0083.
<b>nper</b>	Obrigatório. <b>Integer</b> que especifica o número total de períodos de pagamento na anuidade. Por exemplo, se você pagar prestações mensais de um financiamento de quatro anos para um carro, seu empréstimo terá um total de 4 * 12 (ou 48) períodos de pagamento.
<b>pmt</b>	Obrigatório. <b>Double</b> que especifica o pagamento a ser efetuado em cada período. Os pagamentos geralmente contêm o principal e a taxa de juros que não se altera durante a duração da anuidade.
<b>pv</b>	Opcional. <b>Variant</b> que especifica o valor presente (ou quantia global) de uma série de pagamentos futuros. Por exemplo, quando você obtém um financiamento para comprar um carro, a quantia do empréstimo é o valor presente para o emprestador dos pagamentos mensais do carro que você efetuará. Se for omitido, será assumido o valor 0.
<b>type</b>	Opcional. <b>Variant</b> que especifica quando vencem os pagamentos. Use 0 se os pagamentos vencerem no final do período de pagamento ou 1 se os pagamentos vencerem no início do período. Se for omitido, será assumido o valor 0.

### Comentários

Uma anuidade é uma série de pagamentos monetários fixos efetuados durante um período de tempo. A anuidade pode ser um empréstimo (como um financiamento imobiliário) ou um investimento (como uma poupança mensal).

Os argumentos **rate** e **nper** devem ser calculados usando-se períodos de pagamento expressos nas mesmas unidades. Por exemplo, se a **rate** for calculada usando meses, **nper** também deve ser calculado usando meses.

Para todos os argumentos, a quantia paga (como depósitos em poupança) é representada por valores negativos; a quantia recebida (como cheques de dividendos) é representada por números positivos.

## Função IPmt

Retorna um **Double** que especifica o pagamento de juros em um dado período de uma anuidade com base em pagamentos fixos, periódicos e uma taxa de juros fixa.

### Sintaxe

**IPmt(rate, per, nper, pv[, fv[, type]])**

A função **IPmt** tem os seguintes argumentos nomeados:

Parte	Descrição
<b>rate</b>	Obrigatório. <b>Double</b> que especifica a taxa de juros por período. Por exemplo, se você obtiver um financiamento para um carro a uma taxa de porcentagem anual (TPA) de 10 por cento e fizer pagamentos mensais, a taxa por período será de 0,1/12 ou 0,0083.
<b>per</b>	Obrigatório. <b>Double</b> que especifica o período de pagamento no intervalo de 1 até <b>nper</b> .
<b>nper</b>	Obrigatório. <b>Double</b> que especifica o número total de períodos de pagamento na anuidade. Por exemplo, se você fizer pagamentos mensais por um financiamento de carro de quatro anos, seu empréstimo terá um total de 4 * 12 (ou 48) períodos de pagamento.
<b>pv</b>	Obrigatório. <b>Double</b> que especifica o valor presente, ou valor de hoje, de uma série de pagamentos ou recebimentos futuros. Por exemplo, quando você obtém uma quantia emprestada para comprar um carro, essa quantia é o valor presente para o prestador dos pagamentos mensais do carro que você efetuará.
<b>fv</b>	Opcional. <b>Variant</b> que especifica o valor futuro ou o saldo desejado após ter efetuado o pagamento final. Por exemplo, o valor futuro de um empréstimo é de R\$ 0,00 porque esse é o valor após o pagamento final. Contudo, se você quiser poupar R\$ 50.000 em um período de 18 anos para garantir a educação de seu filho, então o valor futuro é R\$ 50.000. Se for omitido, será assumido 0.
<b>type</b>	Opcional. <b>Variant</b> que especifica quando vencerão os pagamentos. Use 0 se os pagamentos vencerem no final do período, ou 1 se vencerem no início do período. Se for omitido, será assumido 0.

### Comentários

Uma anuidade é uma série de pagamentos monetários fixos efetuados em um período. A anuidade pode ser um empréstimo (como um financiamento imobiliário) ou um investimento (como uma poupança mensal).

Os argumentos **rate** e **nper** devem ser calculados usando-se períodos de pagamento expressos na mesma unidade. Por exemplo, se **rate** for calculada usando meses, **nper** também deve ser calculado usando meses.

Para todos os argumentos, o valor monetário pago (como depósitos em poupança) é representado por números negativos; o valor monetário recebido (como cheques de dividendos) é representado por números positivos.

## Função IRR

Retorna um **Double** que especifica uma taxa interna de retorno para uma série de fluxos de caixa periódicos (pagamentos e recebimentos).

### Sintaxe

**IRR(values)[, guess]**

A função **IRR** tem os seguintes argumentos nomeados:

Parte	Descrição
-------	-----------

<b>values()</b>	Obrigatório. <u>Matriz</u> de <b>Double</b> que especifica valores de fluxo de caixa. A matriz deve conter pelo menos um valor negativo (um pagamento) e um valor positivo (um recebimento).
<b>guess</b>	Opcional. <u>Variant</u> que especifica o valor por você estimado que <b>IRR</b> retornará. Se for omitido, <b>guess</b> será 0.1 (10 por cento).

### Comentários

A taxa de retorno interna é a taxa de juros recebida para um investimento consistindo em pagamentos e recebimentos que ocorrem a intervalos regulares.

A função **IRR** utiliza a ordem dos valores dentro da matriz para interpretar a ordem de pagamentos e recebimentos. Certifique-se de inserir os valores dos seus pagamentos e recebimentos na seqüência correta. O fluxo de caixa de cada período não precisa ser fixo, como é com a anuidade.

**IRR** é calculado por iteração. Começando com o valor de **guess**, **IRR** calcula várias vezes até que o resultado seja exato dentro de 0,00001 por cento. Se **IRR** não puder encontrar um resultado após 20 tentativas, ele falhará.

## Função MIRR

Retorna um **Double** que especifica a taxa interna de retorno modificada de uma série de fluxos de caixa periódicos (pagamentos e recebimentos).

### Sintaxe

**MIRR(values(), finance\_rate, reinvest\_rate)**

A função **MIRR** tem os seguintes argumentos nomeados:

<b>Parte</b>	<b>Descrição</b>
<b>values()</b>	Obrigatório. <u>Matriz</u> de <b>Double</b> que especifica os valores de fluxo de caixa. A matriz deve conter pelo menos um valor negativo (um pagamento) e um valor positivo (um recebimento).
<b>finance_rate</b>	Obrigatório. <b>Double</b> que especifica a taxa de juros paga como custo do financiamento.
<b>reinvest_rate</b>	Obrigatório. <b>Double</b> que especifica a taxa de juros recebida em ganhos do reinvestimento monetário.

### Comentários

A taxa interna de retorno modificada é a taxa interna de retorno quando os pagamentos e recebimentos são financiados com taxas diferentes. A função **MIRR** leva em conta o custo do investimento (**finance\_rate**) e a taxa de juros recebida no reinvestimento do dinheiro (**reinvest\_rate**).

Os argumentos **finance\_rate** e **reinvest\_rate** são porcentagens expressas em valores decimais. Por exemplo, 12 por cento é expresso como 0,12.

A função **MIRR** usa a ordem dos valores dentro da matriz para interpretar a ordem dos pagamentos e recebimentos. Certifique-se de inserir os valores dos seus pagamentos e recebimentos na seqüência correta.

## Função NPer

Retorna um **Double** que especifica o número de períodos para uma anuidade com base em pagamentos fixos e periódicos e uma taxa de juros fixa.

### Sintaxe

**NPer**(rate, pmt, pv [, fv [, type]])

A função **NPer** tem os seguintes argumentos nomeados:

Parte	Descrição
<b>rate</b>	Obrigatório. <b>Double</b> que especifica a taxa de juros por período. Por exemplo, se você obtiver um financiamento para um carro a uma taxa de porcentagem anual (TPA) de 10 por cento e efetuar pagamentos mensais, a taxa por período será de 0,1/12 ou 0,0083.
<b>pmt</b>	Obrigatório. <b>Double</b> que especifica o pagamento a ser efetuado a cada período. Os pagamentos geralmente contêm o principal e os juros que não se alteram durante a duração da anuidade.
<b>pv</b>	Obrigatório. <b>Double</b> que especifica o valor presente, ou valor de hoje, de uma série de pagamentos ou recebimentos futuros. Por exemplo, quando você obtém um financiamento para comprar um carro, a quantia do empréstimo é o valor presente para o prestador dos pagamentos mensais do carro que você efetuará.
<b>fv</b>	Opcional. <b>Variant</b> que especifica o valor futuro ou o saldo que você deseja após ter efetuado o pagamento final. Por exemplo, o valor futuro de um empréstimo é de R\$ 0,00 porque esse é o valor após o pagamento final. Contudo, se você quiser poupar R\$ 50.000 em 18 anos para garantir a educação de seu filho, então R\$ 50.000 será o valor futuro. Se for omitido, será assumido o valor 0.
<b>type</b>	Opcional. <b>Variant</b> que especifica o vencimento dos pagamentos. Use 0 se os pagamentos vencerem ao final do período de pagamento, ou 1 se vencerem no início desse período. Se for omitido, será assumido o valor 0.

### Comentários

Uma anuidade é uma série de pagamentos monetários fixos efetuados em um período de tempo. A anuidade pode ser um empréstimo (como um financiamento imobiliário) ou um investimento (como uma poupança mensal).

Para todos os argumentos, o valor pago (como depósitos em poupança) é representado por números negativos; o valor recebido (como cheques de dividendos) é representado por números positivos.

## Função NPV

Retorna um **Double** que especifica o valor presente líquido de um investimento com base em uma série de fluxos de caixa periódicos (pagamentos e recebimentos) e a taxa de desconto.

### Sintaxe

**NPV**(rate, values())

A função **NPV** tem os seguintes argumentos nomeados:

Parte	Descrição
<b>rate</b>	Obrigatório. <b>Double</b> que especifica a taxa de desconto em um período, expresso na forma de valor decimal.
<b>values()</b>	Obrigatório. <b>Matriz</b> de <b>Double</b> que especifica os valores de fluxo de caixa. A matriz deve conter pelo menos um valor negativo (um pagamento) e um valor positivo (um

recebimento).

### Comentários

O valor presente líquido de um investimento é o valor atual de uma futura série de pagamentos ou recebimentos.

A função **NPV** usa a ordem dos valores dentro da matriz para interpretar a ordem dos pagamentos ou recebimentos. Certifique-se de fornecer os valores de seu pagamento e recebimento na seqüência correta.

O investimento **NPV** começa um período antes da data do primeiro valor de fluxo de caixa e termina com o último valor de fluxo de caixa da matriz.

O cálculo do valor presente líquido é baseado nos fluxos de caixa futuros. Se o seu primeiro fluxo de caixa ocorrer no início do primeiro período, o primeiro valor deverá ser adicionado ao valor retornado por **NPV** e não deve ser incluído nos valores de fluxo de caixa **values()**.

A função **NPV** é semelhante à função **PV** (valor presente) exceto que a função **PV** permite que os fluxos de caixa comecem no final ou no início do período. Ao contrário dos valores variáveis de fluxo de caixa **NPV**, os fluxos de caixa **PV** devem ser fixos durante o investimento.

## Função Pmt

Retorna um **Double** que especifica o pagamento de uma anuidade com base em pagamentos fixos, periódicos e uma taxa de juros fixa.

### Sintaxe

**Pmt(rate, nper, pv [, fv [, type]])**

A função **Pmt** tem os seguintes argumentos nomeados:

Parte	Descrição
<b>rate</b>	Obrigatório. <b>Double</b> que especifica a taxa de juros por período. Por exemplo, se você obtiver um financiamento para um carro a uma taxa percentual anual (TPA) de 10 por cento e efetuar pagamentos mensais, a taxa do período será de 0,1/12, ou 0,0083.
<b>nper</b>	Obrigatório. <b>Integer</b> que especifica o número total de períodos de pagamento na anuidade. Por exemplo, se você efetuar pagamentos mensais sobre um financiamento de quatro anos para um carro, seu financiamento terá um total de 4 * 12 (ou 48) períodos de pagamento.
<b>pv</b>	Obrigatório. <b>Double</b> que especifica o valor presente (ou a quantia global) que uma série de pagamentos a serem feitos no futuro valem agora. Por exemplo, quando você obtém um financiamento para comprar um carro, a quantia do financiamento é o valor presente para o empréstador dos pagamentos mensais do carro que você efetuará.
<b>fv</b>	Opcional. <b>Variant</b> que especifica o valor futuro ou o saldo desejado após ter efetuado o pagamento final. Por exemplo, o valor futuro de um empréstimo é de R\$ 0,00 porque esse é seu valor após o pagamento final. Contudo, se você quiser poupar R\$ 50.000 em um período de 18 anos para garantir a educação de seu filho, então R\$ 50.000 será o valor futuro. Se for omitido, será assumido o valor 0.
<b>type</b>	Opcional. <b>Variant</b> que especifica o vencimento dos pagamentos. Use 0 se os pagamentos vencerem ao final do período de pagamento, ou 1 se vencerem no início do período. Se for omitido, será assumido o valor 0.

### Comentários

Uma anuidade é uma série de pagamentos fixos a serem efetuados em um período. A anuidade pode ser um empréstimo (como um financiamento imobiliário) ou um investimento (como uma poupança

mensal).

Os argumentos **rate** e **nper** devem ser calculados usando-se períodos de pagamento expressos nas mesmas unidades. Por exemplo, se **rate** for calculada usando meses, **nper** também deve ser calculado usando meses.

Para todos os argumentos, o valor pago (como depósitos em uma poupança) é representado por números negativos; o valor recebido (como cheques de dividendos) é representado por números positivos.

## Função PPmt

Retorna um **Double** que especifica o pagamento principal por um dado período de uma anuidade com base em pagamentos fixos e uma taxa de juros fixa.

### Sintaxe

**PPmt**(*rate*, *per*, *nper*, *pv* [, *fv* [, *type*]])

A função **PPmt** tem os seguintes argumentos nomeados:

Parte	Descrição
<b>rate</b>	Obrigatório. <b>Double</b> que especifica a taxa de juros por período. Por exemplo, se você obtiver um financiamento para um carro a uma taxa de porcentagem anual (TPA) de 10 por cento e efetuar pagamentos mensais, a taxa por período será de 0,1/12, ou 0,0083.
<b>per</b>	Obrigatório. <b>Integer</b> que especifica o período de pagamento no intervalo de 1 até <b>nper</b> .
<b>nper</b>	Obrigatório. <b>Integer</b> que especifica o número total de períodos de pagamento na anuidade. Por exemplo, se você efetuar pagamentos mensais sobre um financiamento de quatro anos para um carro, seu empréstimo terá um total de 4 * 12 (ou 48) períodos de pagamento.
<b>pv</b>	Obrigatório. <b>Double</b> que especifica o valor presente, ou valor de hoje, de uma série de pagamentos ou recebimentos futuros. Por exemplo, quando você obtém um financiamento para comprar um carro, a quantia do financiamento é o valor presente para o prestador dos pagamentos mensais do carro que você efetuará.
<b>fv</b>	Opcional. <b>Variant</b> que especifica o valor futuro ou o saldo do valor monetário desejado após ter efetuado o pagamento final. Por exemplo, o valor futuro de um empréstimo é de R\$ 0,00 porque esse é o valor após o pagamento final. Contudo, se você quiser poupar R\$ 50.000 em 18 anos para garantir a educação de seu filho, então R\$ 50.000 será o valor futuro. Se for omitido, será assumido o valor 0.
<b>type</b>	Opcional. <b>Variant</b> que especifica o vencimento dos pagamentos. Use 0 se os pagamentos vencerem no final do período de pagamento, ou 1 se vencerem no início desse período. Se for omitido, será assumido o valor 0.

### Comentários

Uma anuidade é uma série de pagamentos fixos efetuados em um período. Ela pode ser um empréstimo (como um financiamento imobiliário) ou um investimento (como uma poupança mensal).

Os argumentos **rate** e **nper** devem ser calculados usando-se períodos de pagamento expressos nas mesmas unidades. Por exemplo, se a **rate** for calculada usando meses, **nper** deve também ser calculado usando meses.

Para todos os argumentos, o valor pago (como depósitos em poupança) é representado por números negativos; o valor recebido (como cheque de dividendo) é representado por números positivos.

## Função PV

Retorna um **Double** que especifica o valor presente de uma anuidade com base em pagamentos fixos periódicos a serem pagos no futuro e taxas de juros fixas.

### Sintaxe

**PV**(*rate*, *nper*, *pmt*[, *fv*, *type*])

A função **PV** tem os seguintes argumentos nomeados:

Parte	Descrição
<b>rate</b>	Obrigatório. <b>Double</b> que especifica a taxa de juros por período. Por exemplo, se você obtiver um financiamento para um carro a uma taxa de porcentagem anual (TPA) de 10 por cento e efetuar pagamentos anuais, a taxa por período é de 0,1/12 ou 0,0083.
<b>nper</b>	Obrigatório. <b>Integer</b> que especifica o número total de períodos de pagamento da anuidade. Por exemplo, se você efetuar pagamentos mensais sobre um financiamento de quatro anos para um carro, seu empréstimo terá um total de 4 * 12 (ou 48) períodos de pagamento.
<b>pmt</b>	Obrigatório. <b>Double</b> que especifica o pagamento a ser efetuado a cada período. Os pagamentos normalmente contêm o principal e os juros que não se alteram durante o período de duração da anuidade.
<b>fv</b>	Opcional. <b>Variant</b> que especifica o valor futuro ou o saldo do valor monetário desejado após ter efetuado o pagamento final. Por exemplo, o valor futuro de um empréstimo é de R\$ 0,00 porque esse é o valor após o pagamento final. Contudo, se você quiser poupar R\$ 50.000 em 18 anos para garantir a educação de seu filho, então R\$ 50.000 será o valor futuro. Se for omitido, será assumido o valor 0.
<b>type</b>	Opcional. <b>Variant</b> que especifica o vencimento dos pagamentos. Use 0 se os pagamentos vencerem ao final do período de pagamento, ou 1 se vencerem no início desse período. Se for omitido, será assumido o valor 0.

### Comentários

Uma anuidade é uma série de pagamentos monetários fixos efetuados em um período de tempo. A anuidade pode ser um empréstimo (como um financiamento imobiliário) ou um investimento (como uma poupança mensal).

Os argumentos **rate** e **nper** devem ser calculados usando-se períodos de pagamento expressos nas mesmas unidades. Por exemplo, se a **rate** for calculada usando meses, **nper** também deve ser calculado usando meses.

Para todos os argumentos, o valor pago (como depósitos em poupança) é representado por números negativos; o valor recebido (como cheques de dividendos) é representado por números positivos.

## Função Rate

Retorna um **Double** que especifica a taxa de juros por período da anuidade.

### Sintaxe

**Rate**(*nper*, *pmt*, *pv* [, *fv*, *type*, *guess*])

A função **Rate** tem os seguintes argumentos nomeados:

Parte	Descrição
<b>nper</b>	Obrigatório. <b>Double</b> que especifica o número total de períodos de pagamento da anuidade. Por exemplo, se você efetuar pagamentos mensais sobre um financiamento de quatro anos para um carro, seu financiamento possuirá um total de 4 * 12 (ou 48) períodos de pagamento.
<b>pmt</b>	Obrigatório. <b>Double</b> que especifica o pagamento a ser efetuado a cada período. Os pagamentos geralmente contêm o principal e os juros que não se alteram durante a duração da anuidade.
<b>pv</b>	Obrigatório. <b>Double</b> que especifica o valor presente, ou o valor de hoje, de uma série de pagamentos ou recebimentos futuros. Por exemplo, quando você obtiver um financiamento para comprar um carro, a quantia do financiamento é o valor presente para o prestador dos pagamentos mensais do carro que você efetuará.
<b>fv</b>	Opcional. <b>Variant</b> que especifica o valor futuro ou o saldo desejado após ter efetuado o pagamento final. Por exemplo, o valor futuro de um empréstimo é de R\$ 0,00 porque esse é o valor após o pagamento final. Contudo, se você quiser poupar R\$ 50.000 em 18 anos para garantir a educação de seu filho, então R\$ 50.000 será o valor futuro. Se for omitido, será assumido o valor 0.
<b>type</b>	Opcional. <b>Variant</b> que especifica um número indicando o vencimento dos pagamentos. Use 0 se os pagamentos vencerem ao final do período de pagamento, ou 1 se vencerem no início desse período. Se for omitido, será assumido o valor 0.
<b>guess</b>	Opcional. <b>Variant</b> que especifica o valor por você estimado que será retornado por <b>Rate</b> . Se for omitido, <b>guess</b> terá o valor de 0,1 (10 por cento).

### Comentários

Uma anuidade é uma série de pagamentos monetários fixos efetuados em um período de tempo. A anuidade pode ser um empréstimo (como um financiamento imobiliário) ou um investimento (como uma poupança mensal).

Para todos os argumentos, o valor pago (como depósitos em poupança) é representado por números negativos; o valor recebido (como cheques de dividendos) é representado por números positivos.

**Rate** é calculado por iteração. Começando com o valor **guess**, **Rate** calcula várias vezes até que o resultado seja exato dentro de 0,00001 por cento. Se **Rate** não puder encontrar um resultado após 20 tentativas, ele falhará. Se sua estimativa for de 10 por cento e **Rate** falhar, tente um valor diferente para **guess**.

## Função SLN

Retorna um **Double** que especifica uma depreciação em linha reta de um ativo para um período único.

### Sintaxe

**SLN(cost, salvage, life)**

A função **SLN** tem os seguintes argumentos nomeados:

<b>Parte</b>	<b>Descrição</b>
<b>cost</b>	Obrigatório. <b>Double</b> que especifica o custo inicial do ativo.
<b>salvage</b>	Obrigatório. <b>Double</b> que especifica o valor do ativo ao final de sua vida útil.
<b>life</b>	Obrigatório. <b>Double</b> que especifica a duração da vida útil do ativo.

### Comentários

O período de depreciação deve ser expresso na mesma unidade que o argumento **life**. Todos os argumentos devem ser números positivos.

## Função SYD

Retorna um **Double** que especifica a depreciação dos dígitos da soma dos anos de um ativo para um período especificado.

### Sintaxe

**SYD(cost, salvage, life, period)**

A função **SYD** tem os seguintes argumentos nomeados:

<b>Parte</b>	<b>Descrição</b>
<b>cost</b>	Obrigatório. <b>Double</b> que especifica o custo inicial do ativo.
<b>salvage</b>	Obrigatório. <b>Double</b> que especifica o valor do ativo ao final de sua vida útil.
<b>life</b>	Obrigatório. <b>Double</b> que especifica a duração da vida útil do ativo.
<b>period</b>	Obrigatório. <b>Double</b> que especifica o período no qual a depreciação do ativo é calculada.

### Comentários

Os argumentos **life** e **period** devem ser expressos nas mesmas unidades. Por exemplo, se **life** for dado em meses, **period** também deve ser dado em meses. Todos os argumentos devem ser números positivos.

### Exemplo da função IsArray

Este exemplo utiliza a função **IsArray** para verificar se uma variável é uma matriz.

```
Dim MinhaMatriz(1 To 5) As Integer, SuaMatriz, MeuControle ' Declara as
variáveis de matriz.
SuaMatriz = Array(1, 2, 3) ' Utiliza a função Array.
MeuControle = IsArray(MinhaMatriz) ' Retorna True.
MeuControle = IsArray(SuaMatriz) ' Retorna True.
```

### Exemplo da função IsDate

Este exemplo utiliza a função **IsDate** para determinar se uma expressão pode ser convertida em uma data.

```
Dim MinhaData, SuaData, SemData, MeuControle
MinhaData = "12 de Fevereiro de 1969": SuaData = #2/12/69#: SemData = "Alô"
MeuControle = IsDate(MinhaData) ' Retorna True.
MeuControle = IsDate(SuaData) ' Retorna True.
MeuControle = IsDate(SemData) ' Retorna False.
```

### Exemplo da função IsEmpty

Este exemplo utiliza a função **IsEmpty** para determinar se uma variável foi inicializada.

```
Dim MinhaVar, MeuControle
MeuControle = IsEmpty(MinhaVar) ' Retorna True.

MinhaVar = Null ' Atribui Null.
MeuControle = IsEmpty(MinhaVar) ' Retorna False.

MinhaVar = Empty ' Atribui Empty.
MeuControle = IsEmpty(MinhaVar) ' Retorna True.
```

### Exemplo da função IsError

Este exemplo utiliza a função **IsError** para verificar se uma expressão numérica é um valor de erro. A função **CVErr** é utilizada para retornar uma **Error Variant** de uma função definida pelo usuário. Suponha que **FunçãoDoUsuário** é um procedimento de função definido pelo usuário que retorna um valor de erro; por exemplo, um valor de retorno atribuído com a instrução **FunçãoDoUsuário = CVErr(32767)**, onde 32767 é um número definido pelo usuário.

```
Dim ValorDeRetorno, MeuControle
ValorDeRetorno = FunçãoDoUsuário()
MeuControle = IsError(ValorDeRetorno) ' Retorna True.
```

**Exemplo da função IsMissing**

Este exemplo utiliza a função **IsMissing** para verificar se um argumento opcional foi passado a um procedimento definido pelo usuário. Observe que os argumentos **Optional** agora podem ter valores e tipos padrão diferentes de **VARIANT**.

```
Dim ValorDeRetorno
' As instruções a seguir chamam o procedimento de função definido pelo
usuário.
ValorDeRetorno = RetornaDobrado() ' Retorna Null.
ValorDeRetorno = RetornaDobrado(2) ' Retorna 4.

' Definição do procedimento de função.
Function RetornaDobrado(Optional A)
    If IsMissing(A) Then
        ' Se o argumento estiver ausente, retorna um Null.
        RetornaDobrado = Null
    Else
        ' Se o argumento estiver presente, retorna duas vezes o valor.
        RetornaDobrado = A * 2
    End If
End Function
```

**Exemplo da função IsNull**

Este exemplo utiliza a função **IsNull** para determinar se uma variável contém um **Null**.

```
Dim MinhaVar, MeuControle
MeuControle = IsNull(MinhaVar) ' Retorna False.
MinhaVar = ""
MeuControle = IsNull(MinhaVar) ' Retorna False.
MinhaVar = Null
MeuControle = IsNull(MinhaVar) ' Retorna True.
```

**Exemplo da função IsNumeric**

Este exemplo utiliza a função **IsNumeric** para determinar se uma variável pode ser avaliada como um número.

```
Dim MinhaVar, MeuControle
MinhaVar = "53" ' Atribui o valor.
MeuControle = IsNumeric(MinhaVar) ' Retorna True.
MinhaVar = "459,95" ' Atribui o valor.
MeuControle = IsNumeric(MinhaVar) ' Retorna True.
MinhaVar = "Ajuda 45" ' Atribui o valor.
MeuControle = IsNumeric(MinhaVar) ' Retorna False.
```

### Exemplo da função IsObject

Este exemplo utiliza a função **IsObject** para determinar se um identificador representa uma variável de objeto. `MeuObjeto` and `SeuObjeto` são variáveis de objeto do mesmo tipo. São nomes genéricos utilizados somente com fins ilustrativos.

```
Dim MeuInt As Integer, SeuObjeto, MeuControle ' Declara as variáveis.
Dim MeuObjeto As Object
Set SeuObjeto = MeuObjeto ' Atribui uma referência a objeto.
MeuControle = IsObject (SeuObjeto) ' Retorna True.
MeuControle = IsObject (MeuInt) ' Retorna False.
```

### Exemplo da função TypeName

Este exemplo utiliza a função **TypeName** para retornar informações sobre uma variável.

```
' Declara as variáveis.
Dim VarNula, MeuTipo, VarSeq As String, VarInt As Integer, VarAtual As
Currency
Dim VarMatriz (1 To 5) As Integer
VarNula = Null ' Atribui o valor Null.
MeuTipo = TypeName (VarSeq) ' Retorna "String".
MeuTipo = TypeName (VarInt) ' Retorna "Integer".
MeuTipo = TypeName (VarAtual) ' Retorna "Currency".
MeuTipo = TypeName (VarNula) ' Retorna "Null".
MeuTipo = TypeName (VarMatriz) ' Retorna "Integer()".
```

### Exemplo da função VarType

Este exemplo utiliza a função **VarType** para determinar o subtipo de uma variável.

```
Dim VarInt, VarSeq, VarData, MeuControle
' Inicializa as variáveis.
VarInt = 459: VarSeq = "Alô Mundo": VarData = #2/12/69#
MeuControle = VarType (VarInt) ' Retorna 2.
MeuControle = VarType (VarData) ' Retorna 7.
MeuControle = VarType (VarSeq) ' Retorna 8.
```

## Função IsArray

Retorna um valor **Boolean** que indica se uma variável é uma matriz.

### Sintaxe

**IsArray**(*nomedavar*)

O argumento obrigatório *nomedavar* é um identificador que especifica uma variável.

### Comentários

**IsArray** retornará **True** se a variável for uma matriz; caso contrário, retornará **False**. **IsArray** é especialmente útil com variantes que contêm matrizes.

## Função IsDate

Retorna um valor **Boolean** que indica se uma expressão pode ser convertida em uma data.

### Sintaxe

**IsDate**(*expressão*)

O argumento obrigatório *expressão* é um **Variant** que contém uma expressão de data ou expressão de seqüência que pode ser reconhecida como uma data ou hora.

### Comentários

**IsDate** retornará **True** se a expressão for uma data ou puder ser convertida em uma data válida; caso contrário, retornará **False**. No Microsoft Windows, o intervalo de datas válidas é 1 de janeiro de 100 A.D. a 31 de dezembro de 9999 A.D. Os intervalos podem variar de um sistema operacional para outro.

## Função IsEmpty

Retorna um valor **Boolean** que indica se uma variável foi inicializada.

### Sintaxe

**IsEmpty**(*expressão*)

O argumento obrigatório *expressão* é um **Variant** que contém uma expressão numérica ou expressão de seqüência. Entretanto, por ser **IsEmpty** utilizado para determinar se variáveis individuais são inicializadas, o argumento *expressão* é geralmente um nome de variável única.

### Comentários

**IsEmpty** retornará **True** se a variável não tiver sido inicializada ou se tiver sido explicitamente definida como **Empty**; caso contrário, retornará **False**. Se *expressão* contiver mais de uma variável, **False** sempre será retornado. **IsEmpty** retorna somente informações significativas para variantes.

## Função IsError

Retorna um valor **Boolean** que indica se uma expressão é um valor de erro.

### Sintaxe

**IsError**(*expressão*)

O argumento obrigatório *expressão* deve ser uma **Variant** do tipo **VarType vbError**.

### Comentários

Os valores de erro são criados pela conversão de números reais em valores de erro utilizando a função **CVErr**. A função **IsError** é utilizada para determinar se uma expressão numérica representa um erro. **IsError** retornará **True** se o argumento *expressão* indicar um erro. Caso contrário, retornará **False**. **IsError** retorna somente informações significativas para variantes de **VarType vbError**.

## Função IsMissing

Retorna um valor **Boolean** que indica se um argumento **Variant** opcional foi passado para um procedimento.

### Sintaxe

**IsMissing**(*nomedoarg*)

O argumento obrigatório *nomedoarg* contém o nome de um argumento de procedimento **Variant** opcional.

### Comentários

Utilize a função **IsMissing** para detectar se argumentos **Variant** opcionais foram fornecidos ou não na chamada de um procedimento. **IsMissing** retornará **True** se nenhum valor tiver sido passado para o argumento especificado. Caso contrário, retornará **False**. Se **IsMissing** retornar **True** para um argumento, a utilização do argumento ausente em outro código pode gerar um erro definido pelo usuário. Se **IsMissing** for utilizado em um argumento **ParamArray**, ele sempre retornará **False**. Para detectar um **ParamArray** vazio, faça um teste para ver se o limite superior da matriz é menor que seu limite inferior.

## Função IsNull

Retorna um valor **Boolean** que indica se uma expressão contém dados que não são válidos (**Null**).

### Sintaxe

**IsNull**(*expressão*)

O argumento obrigatório *expressão* é um **Variant** que contém uma expressão numérica ou expressão de seqüência.

### Comentários

**IsNull** retornará **True** se *expressão* for **Null**; caso contrário, **IsNull** retornará **False**. Se *expressão* consistir em mais de uma variável, **Null** em qualquer variável constituinte fará com que **True** seja retornado para a expressão inteira.

O valor **Null** indica que **Variant** contém dados que não são válidos. **Null** não é igual a **Empty**, o que indica que uma variável ainda não foi inicializada. Ele também não é igual a uma seqüência de comprimento zero (""), que, algumas vezes, é referida como seqüência nula.

**Importante** Utilize a função **IsNull** para determinar se uma expressão contém um valor **Null**. Expressões que você esperaria serem avaliadas como **True** em algumas circunstâncias, como `If Var = Null and If Var <> Null`, são sempre **False**. Isto acontece porque qualquer expressão que contenha um **Null** é por si mesma **Null** e, portanto, **False**.

## Função IsNumeric

Retorna um valor **Boolean** que indica se uma expressão pode ser avaliada como um número.

### Sintaxe

**IsNumeric**(*expressão*)

O argumento obrigatório *expressão* é um **Variant** que contém uma expressão numérica ou expressão de seqüência.

### Comentários

**IsNumeric** retornará **True** se a *expressão* inteira for reconhecida como um número; caso contrário, retornará **False**.

**IsNumeric** retornará **False** se *expressão* for uma expressão de data.

## Função IsObject

Retorna um valor **Boolean** que indica se um identificador representa uma variável de objeto.

### Sintaxe

**IsObject**(*identificador*)

O argumento obrigatório *identificador* é um nome de variável.

### Comentários

**IsObject** é útil somente para determinar se um **Variant** é do **VarType vbObject**. Isto poderia ocorrer se **Variant** realmente se referisse (ou se já se referiu) a um objeto ou se contiver **Nothing**.

**IsObject** retornará **True** se *identificador* for uma variável declarada com o tipo **Object** ou qualquer tipo de classe válido, ou se *identificador* for um **Variant** de **VarType vbObject**, ou um objeto definido pelo usuário; caso contrário, retornará **False**. **IsObject** retornará **True** mesmo que a variável tenha sido definida como **Nothing**.

Utilize a interceptação de erro para se certificar de que a referência a um objeto é válida.

## Função TypeName

Retorna uma **String** que fornece informações sobre uma variável.

### Sintaxe

**TypeName**(*nomedavar*)

O argumento obrigatório *nomedavar* é uma **Variant** que contém qualquer variável, exceto a de tipo definido pelo usuário.

### Comentários

A seqüência de caracteres retornada por **TypeName** pode ser qualquer uma das seguintes:

Seqüência de caracteres retornada	de Variável
<u>tipo de objeto</u>	Um objeto cujo tipo é <i>tipodeobjeto</i>
<b>Byte</b>	Valor em bytes
<b>Integer</b>	Número inteiro
<b>Long</b>	Inteiro longo
<b>Single</b>	Número de vírgula flutuante de precisão simples
<b>Double</b>	Número de vírgula flutuante de precisão dupla
<b>Currency</b>	Valor de moeda
<b>Decimal</b>	Valor decimal
<b>Date</b>	Valor de data
<b>String</b>	Seqüência de caracteres
<b>Boolean</b>	Valor booleano
<b>Error</b>	Um valor de erro
<b>Empty</b>	Não-inicializada
<b>Null</b>	Dados não-válidos
<b>Object</b>	Um objeto
Desconhecido	Um objeto cujo tipo é desconhecido

**Nothing** Variável de objeto que não se refere a um objeto

Se *nomedavar* for uma matriz, a seqüência de caracteres retornada pode ser qualquer uma das seqüências retornadas possíveis (ou **Variant**) com parênteses vazios anexados. Por exemplo, se *nomedavar* for uma matriz de números inteiros, **TypeName** retornará "Integer ()".

## Função VarType

Retorna um **Integer** que indica o subtipo de uma variável.

### Sintaxe

**VarType**(*nomedavar*)

O argumento obrigatório *nomedavar* é um **Variant** que contém qualquer variável, exceto uma do tipo definido pelo usuário.

### Valores de retorno

Constante	Valor	Descrição
<b>vbEmpty</b>	0	<b>Empty</b> (não-inicializada)
<b>vbNull</b>	1	<b>Null</b> (dados não-válidos)
<b>vbInteger</b>	2	Número inteiro
<b>vbLong</b>	3	Inteiro longo
<b>vbSingle</b>	4	Número de vírgula flutuante de precisão simples
<b>vbDouble</b>	5	Número de vírgula flutuante de precisão dupla
<b>vbCurrency</b>	6	Valor de moeda
<b>vbDate</b>	7	Valor da data
<b>vbString</b>	8	Seqüência de caracteres
<b>vbObject</b>	9	Objeto
<b>vbError</b>	10	Valor do erro
<b>vbBoolean</b>	11	Valor booleano
<b>vbVariant</b>	12	<b>Variant</b> (utilizada somente com <u>matrizes</u> de variantes)
<b>vbDataObject</b>	13	Um objeto de acesso de dados
<b>vbDecimal</b>	14	Valor decimal
<b>vbByte</b>	17	Valor em bytes
<b>vbArray</b>	8192	Matriz

**Observação** Estas constantes são especificadas pelo Visual Basic para Aplicativos. Os nomes podem ser utilizados em qualquer parte do seu código no lugar dos valores reais.

### Comentários

A função **VarType** nunca retorna o valor de **vbArray** por si mesma. Ele é sempre adicionado a algum outro valor para indicar uma matriz de um tipo específico. A constante **vbVariant** somente é retornada em conjunto com **vbArray** para indicar que o argumento para a função **VarType** é uma matriz do tipo **Variant**. Por exemplo, o valor retornado para uma matriz de números inteiros é calculado como **vbInteger** + **vbArray** ou 8194. Se um objeto possui uma propriedade padrão, **VarType** (*objeto*) retorna o tipo da propriedade padrão do objeto.

## Empty

A palavra-chave **Empty** é utilizada como um subtipo de **Variant**. Ela indica um valor de variável não inicializada.

## False

A palavra-chave **False** tem valor igual a 0.

## Me

A palavra-chave **Me** comporta-se como uma variável declarada implicitamente. Ela é automaticamente disponível para cada procedimento em um módulo de classe. Quando uma classe pode ter mais de uma instância, **Me** proporciona uma forma de referir a instância específica da classe onde o código está sendo executado. O uso de **Me** é particularmente útil para passar informações sobre a instância de uma classe que está sendo atualmente executada para um procedimento em um outro módulo. Por exemplo, suponha que você possua o seguinte procedimento em um módulo:

```
Sub AlterarCorForm(NomeForm As Form)
    NomeForm.BackColor = RGB(Rnd * 256, Rnd * 256, Rnd * 256)
End Sub
```

Você pode chamar esse procedimento e passar a instância atual da classe do Formulário como um argumento utilizando a seguinte instrução:

```
AlterarCorForm Me
```

## Nothing

A palavra-chave **Nothing** é utilizada para desassociar uma variável de objeto de um objeto real. Use a instrução **Set** para atribuir **Nothing** a uma variável de objeto. Por exemplo:

```
Set MeuObjeto = Nothing
```

Diversas variáveis de objeto podem se referir ao mesmo objeto real. Quando **Nothing** é atribuído a uma variável de objeto, essa variável não fará mais referência ao objeto real. Quando várias variáveis de objeto se referirem ao mesmo objeto, os recursos de memória e sistema associados ao objeto ao qual as variáveis se referem são liberados somente após todos eles terem sido definidos como **Nothing**, usando **Set** de forma explícita ou implícita após a última variável de objeto definida como **Nothing** sair do escopo.

## Null

A palavra-chave **Null** é utilizada como um subtipo de **Variant**. Ela indica que uma variável não contém dados válidos.

## True

A palavra-chave **True** tem valor igual a -1.

### Exemplo da função Abs

Este exemplo utiliza a função **Abs** para computar o valor absoluto de um número.

```
Dim MeuNúmero
MeuNúmero = Abs(50.3) ' Retorna 50.3.
MeuNúmero = Abs(-50.3) ' Retorna 50.3.
```

### Exemplo da função Atn

Este exemplo utiliza a função **Atn** para calcular o valor de pi.

```
Dim pi
pi = 4 * Atn(1) ' Calcula o valor de pi.
```

### Exemplo da função Cos

Este exemplo utiliza a função **Cos** para retornar o co-seno de um ângulo.

```
Dim MeuÂngulo, MinhaSecante
MeuÂngulo = 1.3 ' Define o ângulo em radianos.
MinhaSecante = 1 / Cos (MeuÂngulo) ' Calcula a secante.
```

### Exemplo da função Exp

Este exemplo utiliza a função **Exp** para retornar e elevado a uma potência.

```
Dim MeuÂngulo, MeuSenoH
' Define o ângulo em radianos.
MeuÂngulo = 1.3
' Calcula o seno hiperbólico.
MeuSenoH = (Exp (MeuÂngulo) - Exp (-1 * MeuÂngulo)) / 2
```

### Exemplo da função Int e da função Fix

Este exemplo ilustra como as funções **Int** e **Fix** retornam porções inteiras de números. No caso de um argumento de número negativo, a função **Int** retorna o primeiro inteiro negativo menor que ou igual ao número; a função **Fix** retorna o primeiro inteiro negativo maior que ou igual ao número.

```
Dim MeuNúmero
MeuNúmero = Int (99.8) ' Retorna 99.
MeuNúmero = Fix (99.2) ' Retorna 99.

MeuNúmero = Int (-99.8) ' Retorna -100.
MeuNúmero = Fix (-99.8) ' Retorna -99.

MeuNúmero = Int (-99.2) ' Retorna -100.
MeuNúmero = Fix (-99.2) ' Retorna -99.
```

### Exemplo da função Log

Este exemplo utiliza a função **Log** para retornar o logaritmo natural de um número.

```
Dim MeuÂngulo, MeuLog
' Define o ângulo em radianos.
MeuÂngulo = 1.3
' Calcula o seno hiperbólico inverso.
MeuLog = Log (MeuÂngulo + Sqr (MeuÂngulo * MeuÂngulo + 1))
```

### Exemplo da instrução Randomize

Este exemplo utiliza a instrução **Randomize** para inicializar o gerador de números aleatórios. Como o argumento de número foi omitido, **Randomize** utiliza o valor de retorno da função **Timer** como o novo valor semente.

```
Dim MeuValor
Randomize ' Inicializa o gerador de números aleatórios.
MeuValor = Int((6 * Rnd) + 1) ' Gera um valor aleatório entre 1 e 6.
```

### Exemplo da função Rnd

Este exemplo utiliza a função **Rnd** para gerar um valor inteiro aleatório de 1 a 6.

```
Dim MeuValor
MeuValor = Int((6 * Rnd) + 1) ' Gera um valor aleatório entre 1 e 6.
```

### Exemplo da função Sgn

Este exemplo utiliza a função **Sgn** para determinar o sinal de um número.

```
Dim MinhaVar1, MinhaVar2, MinhaVar3, MeuSinal
MinhaVar1 = 12: MinhaVar2 = -2.4: MinhaVar3 = 0
MeuSinal = Sgn(MinhaVar1) ' Retorna 1.
MeuSinal = Sgn(MinhaVar2) ' Retorna -1.
MeuSinal = Sgn(MinhaVar3) ' Retorna 0.
```

### Exemplo da função Sin

Este exemplo utiliza a função **Sin** para retornar o seno de um ângulo.

```
Dim MeuÂngulo, MinhaCossecante
MeuÂngulo = 1.3 ' Define o ângulo em radianos.
MinhaCossecante = 1 / Sin(MeuÂngulo) ' Calcula a cossecante.
```

### Exemplo da função Sqr

Este exemplo utiliza a função **Sqr** para calcular a raiz quadrada de um número.

```
Dim MinhaRaizQuad
MinhaRaizQuad = Sqr(4) ' Retorna 2.
MinhaRaizQuad = Sqr(23) ' Retorna 4.79583152331272.
MinhaRaizQuad = Sqr(0) ' Retorna 0.
MinhaRaizQuad = Sqr(-4) ' Gera um erro em tempo de execução.
```

### Exemplo da função Tan

Este exemplo utiliza a função **Tan** para retornar a tangente de um ângulo.

```
Dim MeuÂngulo, MinhaCotangente
MeuÂngulo = 1.3 ' Define o ângulo em radianos.
MinhaCotangente = 1 / Tan(MeuÂngulo) ' Calcula a cotangente.
```

## Funções matemáticas

### Função Abs

Retorna um valor do mesmo tipo que é passado para ele especificando o valor absoluto de um número.

#### Sintaxe

**Abs**(*número*)

O argumento obrigatório *número* pode ser qualquer expressão numérica válida. Se *número* contiver **Null**, será retornado **Null**; se for uma variável não inicializada, será retornado zero.

#### Comentários

O valor absoluto de um número é sua magnitude sem sinal. Por exemplo, tanto `ABS (-1)` como `ABS (1)` retornam 1.

### Função Atn

Retorna um **Double** que especifica o arco tangente de um número.

#### Sintaxe

**Atn**(*número*)

O argumento obrigatório *número* é um **Double** ou qualquer expressão numérica válida.

#### Comentários

A função **Atn** toma a razão entre dois lados de um triângulo retângulo (*número*) e retorna o ângulo correspondente em radianos. A razão é o comprimento do lado oposto ao ângulo dividido pelo comprimento do lado adjacente a ele.

O intervalo do resultado é  $-\pi/2$  a  $\pi/2$  radianos.

Para converter graus em radianos, multiplique graus por  $\pi/180$ . Para converter radianos em graus, multiplique radianos por  $180/\pi$ .

**Observação** **Atn** é a função trigonométrica inversa de **Tan**, que toma um ângulo como seu argumento e retorna a razão entre dois lados de um triângulo retângulo. Não confunda **Atn** com a co-tangente, que é o inverso simples da tangente ( $1/\text{tangente}$ ).

### Função Cos

Retorna um **Double** que especifica o co-seno de um ângulo.

#### Sintaxe

**Cos**(*número*)

O argumento obrigatório *número* é um **Double** ou qualquer expressão numérica válida que expressa um ângulo em radianos.

#### Comentários

A função **Cos** toma um ângulo e retorna a razão entre dois lados de um triângulo retângulo. A razão é o comprimento do lado adjacente ao ângulo dividido pelo comprimento da hipotenusa.

O resultado situa-se no intervalo -1 a 1.

Para converter graus em radianos, multiplique graus por  $\pi/180$ . Para converter radianos em graus, multiplique radianos por  $180/\pi$ .

## Função Exp

Retorna um **Double** que especifica *e* (a base dos logaritmos naturais) elevada a uma potência.

### Sintaxe

**Exp**(*número*)

O argumento obrigatório *número* é um **Double** ou qualquer expressão numérica válida.

### Comentários

Se o valor de *número* exceder 709,782712893, ocorrerá um erro. A constante *e* vale aproximadamente 2,718282.

**Observação** A função **Exp** complementa a ação da função **Log** e às vezes é chamada de antilogaritmo.

## Funções Int, Fix

Retornam um valor do tipo passado para ele contendo a parte inteira de um número.

### Sintaxe

**Int**(*número*)

**Fix**(*número*)

- argumento obrigatório *número* é um **Double** ou qualquer expressão numérica válida. Se *número* contiver **Null**, é retornado **Null**.

### Comentários

Tanto **Int** como **Fix** removem a parte fracionária de *número* e retornam o valor inteiro resultante.

A diferença entre **Int** e **Fix** é que, se *número* for negativo, **Int** retorna o primeiro inteiro negativo que seja menor ou igual a *número*, enquanto **Fix** retorna o primeiro inteiro negativo maior ou igual a *número*. Por exemplo, **Int** converte -8.4 para -9, e **Fix** converte -8.4 para -8.

**Fix**(*número*) é equivalente a:

`Sgn(número) * Int(Abs(número))`

## Função Log

Retorna um **Double** que especifica o logaritmo natural de um número.

### Sintaxe

**Log**(*número*)

O argumento obrigatório *número* é um **Double** ou qualquer expressão numérica válida maior que zero.

### Comentários

O logaritmo natural é o logaritmo com base *e*. A constante *e* vale aproximadamente 2,718282.

Você pode calcular logaritmos de base *n* para qualquer número *x* dividindo o logaritmo natural de *x* pelo logaritmo natural de *n*, como a seguir:

$\text{Log}_n(x) = \text{Log}(x) / \text{Log}(n)$

O exemplo seguinte ilustra um **Function** personalizada que calcula logaritmos de base 10:

```
Static Function Log10(X)
    Log10 = Log(X) / Log(10#)
```

---

End Function

## Instrução Randomize

Inicializa o gerador de números aleatórios.

### Sintaxe

**Randomize** [*número*]

O argumento opcional *número* é um **Variant** ou qualquer expressão numérica válida.

### Comentários

**Randomize** utiliza *número* para inicializar o gerador de números aleatórios da função **Rnd**, dando-lhe um novo valor de semente. Se você omitir *número*, o valor retornado pelo cronômetro do sistema será utilizado como o novo valor de semente.

Se **Randomize** não for utilizado, a função **Rnd** (sem argumentos) utilizará o mesmo número como uma semente a primeira vez é chamada, e daí em diante utilizará o último número gerado como valor semente.

**Observação** Para repetir seqüências de números aleatórios, chame **Rnd** com um argumento negativo imediatamente antes de utilizar **Randomize** com um argumento numérico. Utilizar **Randomize** com o mesmo valor de *número* não repete a seqüência anterior.

## Função Rnd

Retorna um **Single** que contém um número aleatório.

### Sintaxe

**Rnd**[(*número*)]

O argumento opcional *número* é um **Single** ou qualquer expressão numérica válida.

### Valores de Retorno

<b>Se <i>número</i> for</b>	<b>Rnd gera</b>
Menor que zero	O mesmo número todas as vezes, utilizando <i>número</i> como <u>semente</u> .
Maior que zero	O próximo número aleatório na seqüência.
Igual a zero	O número gerado mais recentemente.
Não fornecido	O próximo número aleatório na seqüência.

### Comentários

A função **Rnd** retorna um valor menor que 1, mas maior ou igual a zero.

O valor de *número* determina como **Rnd** gera um número aleatório:

Para qualquer semente inicial dada, a mesma seqüência de números é gerada porque cada chamada que sucede a função **Rnd** utiliza o número anterior como semente para o próximo número da seqüência.

Antes de chamar **Rnd**, utilize a instrução **Randomize** sem argumento para inicializar o gerador de números aleatórios com uma semente baseada no cronômetro do sistema.

Para produzir números inteiros aleatórios, utilize esta fórmula:

```
Int((limitesuperior - limiteinferior + 1) * Rnd + limiteinferior)
```

Aqui, *limitesuperior* é o número mais alto do intervalo e *limiteinferior* é o número mais baixo.

**Observação** Para repetir seqüências de números aleatórios, chame **Rnd** com um argumento

negativo imediatamente antes de utilizar **Randomize** com um argumento numérico. A utilização de **Randomize** com o mesmo valor de *número* não repete a seqüência anterior.

## Função Sgn

Retorna um **Variant (Integer)** que indica o sinal de um número.

### Sintaxe

**Sgn**(*número*)

O argumento obrigatório *número* pode ser qualquer expressão numérica válida.

### Valores de retorno

Se <i>número</i> for	Sgn retorna
Maior que zero	1
Igual a zero	0
Menor que zero	-1

### Comentários

O sinal do argumento *número* determina o valor de retorno da função **Sgn**.

## Função Sin

Retorna um **Double** que especifica o seno de um ângulo.

### Sintaxe

**Sin**(*número*)

O argumento obrigatório *número* é um **Double** ou qualquer expressão numérica válida que exprima um ângulo em radianos.

### Comentários

A função **Sin** toma um ângulo e retorna a razão entre dois lados de um triângulo retângulo. A razão é o comprimento do lado oposto ao ângulo dividido pelo comprimento da hipotenusa.

O resultado situa-se no intervalo -1 a 1.

Para converter graus em radianos, multiplique graus por  $\pi/180$ . Para converter radianos em graus, multiplique radianos por  $180/\pi$ .

## Função Sqr

Retorna um **Double** que especifica a raiz quadrada de um número.

### Sintaxe

**Sqr**(*número*)

O argumento obrigatório *número* é um **Double** ou qualquer expressão numérica válida maior ou igual a zero.

## Função Tan

Retorna um **Double** que especifica a tangente de um ângulo.

### Sintaxe

#### Tan(*número*)

O argumento obrigatório *número* é um **Double** ou qualquer expressão numérica válida que exprima um ângulo em radianos.

### Comentários

**Tan** toma um ângulo e retorna a razão entre dois lados de um triângulo retângulo. A razão é o comprimento do lado oposto ao ângulo dividido pelo comprimento do lado adjacente.

Para converter graus em radianos, multiplique graus por  $\pi/180$ . Para converter radianos em graus, multiplique radianos por  $180/\pi$ .

## Funções matemáticas derivadas

A lista abaixo é uma lista de funções matemáticas não intrínsecas que podem ser derivadas a partir das funções matemáticas intrínsecas:

<b>Função</b>	<b>Equivalentes derivadas</b>
Secante	$\text{Sec}(X) = 1 / \text{Cos}(X)$
Co-secante	$\text{Cosec}(X) = 1 / \text{Sin}(X)$
Co-tangente	$\text{Cotan}(X) = 1 / \text{Tan}(X)$
Seno inverso	$\text{Arcsin}(X) = \text{Atn}(X / \text{Sqr}(-X * X + 1))$
Co-seno inverso	$\text{Arccos}(X) = \text{Atn}(-X / \text{Sqr}(-X * X + 1)) + 2 * \text{Atn}(1)$
Secante inversa	$\text{Arcsec}(X) = \text{Atn}(X / \text{Sqr}(X * X - 1)) + \text{Sgn}((X) - 1) * (2 * \text{Atn}(1))$
Co-secante inversa	$\text{Arccosec}(X) = \text{Atn}(X / \text{Sqr}(X * X - 1)) + (\text{Sgn}(X) - 1) * (2 * \text{Atn}(1))$
Co-tangente inversa	$\text{Arccotan}(X) = \text{Atn}(X) + 2 * \text{Atn}(1)$
Seno hiperbólico	$\text{Sinh}(X) = (\text{Exp}(X) - \text{Exp}(-X)) / 2$
Co-seno hiperbólico	$\text{Cosh}(X) = (\text{Exp}(X) + \text{Exp}(-X)) / 2$
Tangente hiperbólica	$\text{Tanh}(X) = (\text{Exp}(X) - \text{Exp}(-X)) / (\text{Exp}(X) + \text{Exp}(-X))$
Secante hiperbólica	$\text{Sech}(X) = 2 / (\text{Exp}(X) + \text{Exp}(-X))$
Co-secante hiperbólica	$\text{Cosech}(X) = 2 / (\text{Exp}(X) - \text{Exp}(-X))$
Co-tangente hiperbólica	$\text{Cotanh}(X) = (\text{Exp}(X) + \text{Exp}(-X)) / (\text{Exp}(X) - \text{Exp}(-X))$
Seno hiperbólico inverso	$\text{Arcsinh}(X) = \text{Log}(X + \text{Sqr}(X * X + 1))$
Co-seno hiperbólico inverso	$\text{Arccosh}(X) = \text{Log}(X + \text{Sqr}(X * X - 1))$
Tangente hiperbólica inversa	$\text{Arctanh}(X) = \text{Log}((1 + X) / (1 - X)) / 2$
Secante hiperbólica inversa	$\text{Arcsech}(X) = \text{Log}((\text{Sqr}(-X * X + 1) + 1) / X)$
Co-secante hiperbólica inversa	$\text{Arccosech}(X) = \text{Log}((\text{Sgn}(X) * \text{Sqr}(X * X + 1) + 1) / X)$
Co-tangente hiperbólica inversa	$\text{Arccotanh}(X) = \text{Log}((X + 1) / (X - 1)) / 2$
Logaritmo de base N	$\text{Logn}(X) = \text{Log}(X) / \text{Log}(N)$

**Exemplo do método Add**

Este exemplo utiliza o método **Add** para adicionar objetos `Inst` (instâncias de uma classe denominada `Class1` que contém uma variável **Public** `NomeDaInstância`) a uma coleção denominada `MinhasClasses`. Para ver como isto funciona, insira um módulo de classe e declare uma variável pública denominada `NomeDaInstância` em nível de módulo da `Class1` (digite **Public** `NomeDaInstância`) para conter os nomes de cada instância. Deixe o nome padrão como `Class1`. Copie e cole o código a seguir no procedimento de evento `Form_Load` de um módulo de formulário.

```
Dim MinhasClasses As New Collection ' Cria um objeto Collection.
Dim Num As Integer ' Contador para individualizar chaves.
Dim Msg
Dim ONome ' Contém os nomes que o usuário insere.
Do
    Dim Inst As New Class1 ' Cria uma nova instância da Class1.
    Num = Num + 1 ' Incrementa Num e, em seguida, obtém um nome.
    Msg = "Insira um nome para este objeto". & Chr(13) _
        & "Pressione Cancelar para ver os nomes na coleção".
    ONome = InputBox(Msg, "Nomeie a coleção de Items")
    Inst.NomeDaInstância = ONome ' Coloca o nome na instância de objeto.
    ' Se o usuário inseriu o nome, adiciona-o à coleção.
    If Inst.NomeDaInstância <> "" Then
        ' Adiciona o objeto nomeado à coleção.
        MinhasClasses.Add item := Inst, key := CStr(Num)
    End If
    ' Limpa a referência atual em preparação para a seguinte.
    Set Inst = Nothing
Loop Until ONome = ""
For Each x In MinhasClasses
    MsgBox x.NomeDaInstância, , "Nome da instância"
Next
```

**Exemplo do método Clear**

Este exemplo utiliza o método **Clear** do objeto **Err** para redefinir as propriedades numéricas do objeto **Err** como zero e as suas propriedades de seqüência de caracteres como seqüências de caracteres de comprimento zero. Se **Clear** fosse omitido do código a seguir, a caixa de diálogo de mensagem de erro seria exibida a cada iteração do loop (depois que ocorresse um erro), independentemente de um cálculo sucessivo gerar ou não um erro. Você pode simplesmente passar pelo código para ver o resultado.

```
Dim Resultado(10) As Integer ' Declara a matriz cujos elementos
    ' serão facilmente sobrecarregados.
Dim ind
On Error Resume Next ' Adia a interceptação do erro.
Do Until ind = 10
    ' Gera um erro ocasional ou armazena o resultado se não houver erro.
    Resultado(ind) = Rnd * ind * 20000
    If Err.Number <> 0 Then
        MsgBox Err, , "Erro gerado: ", Err.HelpFile, Err.HelpContext
        Err.Clear ' Limpa as propriedades do objeto Err.
    End If
    ind = ind + 1
Loop
```

### Exemplo do método Item

Este exemplo utiliza o método **Item** para recuperar uma referência a um objeto em uma coleção. Supondo que `Aniversários` é um objeto **Collection**, o código a seguir recupera referências aos objetos da coleção que representam a data do aniversário de Bill Smith e do aniversário de Adam Smith, utilizando as chaves "SmithBill" e "SmithAdam" como argumentos de *índice*. Observe que a primeira chamada especifica explicitamente o método **Item**, mas a segunda não. Ambas as chamadas funcionam porque o método **Item** é o padrão de um objeto **Collection**. As referências, atribuídas a `SmithBillBD` e `SmithAdamBD` utilizando **Set**, podem ser utilizadas para acessar as propriedades e métodos dos objetos especificados. Para executar este código, crie a coleção e preencha-a com pelo menos dois membros referenciados.

```
Dim SmithBillBD As Object
Dim SmithAdamBD As Object
Dim Aniversários
Set SmithBillBD = Aniversários.Item("SmithBill")
Set SmithAdamBD = Aniversários("SmithAdam")
```

### Exemplo do método Print

Utilizando o método **Print**, este exemplo exibe o valor da variável `MinhaVar` no painel **Imediato** da janela **Depurar**. Observe que o método **Print** se aplica somente a objetos que podem exibir texto.

```
Dim MinhaVar
MinhaVar = "Venha me ver no painel Imediato"
Debug.Print MinhaVar
```

### Exemplo do método Raise

Este exemplo utiliza o método **Raise** do objeto **Err** para gerar um erro com um objeto de Automação gravado no Visual Basic. Ele tem o código programático `MeuProj.MeuObjeto`.

```
Const MeuCódigoDeContexto = 1010407 ' Define uma constante para o código do contexto.
Function TestName(NomeAtual, NovoNome)
    If Instr(NovoNome, "bob") Then ' Testa a validade de NovoNome.
        ' Provoca a exceção.
        Err.Raise vbObjectError + 27, "MeuProj.MeuObjeto", _
            "Nenhum ""bob"" permitido no seu nome", "c:\MeuProj\MinhAjuda.Hlp", _
            MeuCódigoDeContexto
    End If
End Function
```

### Exemplo do método Remove

Este exemplo ilustra o uso do método **Remove** para remover objetos de um objeto **Collection**, `MinhasClasses`. Este código remove o objeto cujo índice é 1 em cada iteração do loop.

```
Dim Num, MinhasClasses
For Num = 1 To MinhasClasses.Count
    MinhasClasses.Remove 1 ' Remove o primeiro objeto cada vez
    ' através do loop até que não haja mais
    ' objetos na coleção.
Next Num
```

## Método Add

Adiciona um membro a um objeto **Collection**.

**Sintaxe***objeto*.**Add item, key, before, after**A sintaxe do método **Add** tem o qualificador de objeto e os argumentos nomeados a seguir:

<b>Parte</b>	<b>Descrição</b>
<i>objeto</i>	Obrigatório. Uma <u>expressão de objeto</u> que é avaliado como um objeto da lista Relativo a.
<i>item</i>	Obrigatório. Uma <u>expressão</u> de qualquer tipo que especifica o membro a ser adicionado à <u>coleção</u> .
<i>key</i>	Opcional. Uma <u>expressão de seqüência de caracteres</u> exclusiva que especifica uma seqüência-chave que pode ser utilizada, em vez de um índice posicional, para acessar um membro da coleção.
<i>before</i>	Opcional. Uma expressão que especifica uma posição relativa na coleção. O membro a ser adicionado é colocado na coleção antes do membro identificado pelo <u>argumento before</u> . Se for uma <u>expressão numérica</u> , <b>before</b> deve ser um número de 1 ao valor da propriedade <b>Count</b> da coleção. Se for uma expressão de seqüência de caracteres, <b>before</b> deve corresponder à <b>key</b> especificada quando o membro sendo referido foi adicionado à coleção. Você pode especificar uma posição <b>before</b> ou uma posição <b>after</b> , mas não ambas.
<i>after</i>	Opcional. Uma expressão que especifica uma posição relativa na coleção. O membro a ser adicionado é colocado na coleção depois do membro identificado pelo argumento <b>after</b> . Se for numérico, <b>after</b> deve ser um número de 1 ao valor da propriedade <b>Count</b> da coleção. Se for uma seqüência de caracteres, <b>after</b> deve corresponder à <b>key</b> especificada quando o membro referido foi adicionado à coleção. Você pode especificar uma posição <b>before</b> ou uma posição <b>after</b> , mas não ambas.

**Comentários**

Se o argumento **before** ou **after** for uma expressão de seqüência de caracteres ou uma expressão numérica, ele deve fazer referência a um membro existente da coleção, ou ocorrerá um erro.

Também ocorrerá um erro se uma **key** especificada duplicar a **key** de um membro existente da coleção.

**Método Clear**

Limpa todas as configurações de propriedade do objeto **Err**.

**Sintaxe***objeto*.**Clear**O *objeto* é sempre o objeto **Err**.**Comentários**

Utilize **Clear** para limpar explicitamente o objeto **Err** após um erro ter sido tratado, por exemplo, quando você utiliza a manipulação de erro diferida com **On Error Resume Next**. O método **Clear** é chamado automaticamente sempre que qualquer uma das instruções a seguir for executada:

- Qualquer tipo de instrução **Resume**
- **Exit Sub**, **Exit Function**, **Exit Property**
- Qualquer instrução **On Error**

**Observação** A construção **On Error Resume Next** pode ser preferível a **On Error GoTo** ao tratar erros gerados durante o acesso a outros objetos. Verificar **Err** após cada interação com um objeto

remove a ambigüidade sobre qual objeto foi acessado pelo código. Você pode ter certeza de qual objeto colocou o código de erro em **Err.Number**, assim como de qual objeto gerou originalmente o erro (o objeto especificado em **Err.Source**).

## Método Item

Retorna um membro específico de um objeto **Collection**, seja por posição ou por chave.

### Sintaxe

*objeto*.Item(*índice*)

A sintaxe do método **Item** tem o qualificador e parte do objeto a seguir:

Parte	Descrição
<i>objeto</i>	Obrigatório. Uma <u>expressão de objeto</u> que é avaliada como um objeto na lista Relativo a.
<i>índice</i>	Obrigatório. Uma <u>expressão</u> que especifica a posição de um membro da <u>coleção</u> . Se for uma <u>expressão numérica</u> , <i>índice</i> deve ser um número de 1 ao valor da propriedade <b>Count</b> da coleção. Se for uma <u>expressão de seqüência de caracteres</u> , <i>índice</i> deve corresponder ao <u>argumento</u> <b>key</b> especificado quando o membro referido tiver sido adicionado à coleção.

### Comentários

Se o valor fornecido como *índice* não corresponder a nenhum membro existente da coleção, ocorrerá um erro.

O método **Item** é o método padrão de uma coleção. Por essa razão, as linhas de código a seguir são equivalentes:

```
Print MinhaColeção(1)
Print MinhaColeção.Item(1)
```

## Método Print

Imprime o texto no painel **Imediato** da janela **Depurar**.

### Sintaxe

*objeto*.Print [*listadesaída*]

A sintaxe do método **Print** tem o seguinte qualificador e parte do objeto:

Parte	Descrição
<i>objeto</i>	Opcional. Uma <u>expressão de objeto</u> que é avaliada como um objeto da lista Relativo a.
<i>listadesaída</i>	Opcional. <u>Expressão</u> ou lista de expressões a ser impressa. Se for omitida, será impressa uma linha em branco.

O argumento *listadesaída* tem as seguintes sintaxe e partes:

{**Spc**(*n*) | **Tab**(*n*)} *expressão poscarac*

Parte	Descrição
<b>Spc</b> ( <i>n</i> )	Opcional. Utilizado para inserir caracteres espaço na saída, onde <i>n</i> é o número de caracteres espaço a ser inseridos.
<b>Tab</b> ( <i>n</i> )	Opcional. Utilizado para posicionar o ponto de inserção em um número de coluna absoluto onde <i>n</i> é o número de coluna. Use <b>Tab</b> sem argumento para posicionar o ponto de inserção

	no início da próxima <u>área de impressão</u> .
<i>expressão</i>	Opcional. <u>Expressão numérica</u> ou <u>expressão de seqüência de caracteres</u> a ser impressa.
<i>poscarac</i>	Opcional. Especifica o ponto de inserção para o próximo caractere. Use ponto-e-vírgula (;) para posicionar o ponto de inserção imediatamente após o último caractere exibido. Use <b>Tab(n)</b> para posicionar o ponto de inserção em um número de coluna absoluto. Use <b>Tab</b> sem argumento para posicionar o ponto de inserção no início da área de impressão. Se <i>poscarac</i> for omitido, o próximo caractere será impresso na próxima linha.

### Comentários

Várias expressões podem ser separadas com um espaço ou um ponto-e-vírgula.

Todos os dados impressos na janela **Imediato** são formatados de forma apropriada usando o separador decimal para as configurações de localidade especificadas para seu sistema. As palavras-chave são colocadas no idioma apropriado do aplicativo host.

Para os dados **Boolean**, é impresso `True` ou `False`. As palavras-chave **True** e **False** são convertidas de acordo com a definição de localidade para o aplicativo host.

Os dados **Date** são gravados usando o formato de data padrão reconhecido pelo seu sistema. Quando faltam os componentes de data ou hora ou eles equivalem a zero, somente os dados fornecidos são gravados.

Nada é gravado se os dados de *listadesaída* estiverem **Empty**. Contudo, se esses dados forem **Null**, a saída será `Null`. A palavra-chave **Null** é convertida de forma apropriada na saída.

Para os dados com erro, a saída é gravada como `Error códigoerro`. A palavra-chave **Error** é convertida de forma apropriada na saída.

O *objeto* será requerido se o método for utilizado fora de um módulo contendo um espaço de exibição padrão. Por exemplo, ocorrerá um erro se o método for chamado em um módulo padrão sem especificar um *objeto*, mas, se for chamado em um módulo de formulário, *listadesaída* será exibida no formulário.

**Observação** Como o método **Print** tipicamente imprime com caracteres com espaço proporcional, não há correlação entre o número de caracteres impressos e o número de colunas de largura fixa que esses caracteres ocupam. Por exemplo, uma letra larga, como o "W", ocupa mais do que uma coluna de largura fixa, e uma letra estreita, como o "i", ocupa menos do que essa coluna. Para permitir os casos em que os caracteres maiores do que a média sejam usados, suas colunas tabulares devem ser posicionadas longe o suficiente umas das outras. De forma alternativa, você pode imprimir usando uma fonte de espaçamento fixo (como Courier) para assegurar que cada caractere ocupe somente uma coluna.

## Método Raise

Gera um erro de tempo de execução.

### Sintaxe

*objeto*.**Raise** *number, source, description, helpfile, helpcontext*

O método **Raise** tem o qualificador de objeto e os argumentos nomeados a seguir:

<b>Argumento</b>	<b>Descrição</b>
<i>objeto</i>	Obrigatório. É sempre o objeto <b>Err</b> .
<i>number</i>	Obrigatório. Inteiro <b>Long</b> que identifica a natureza do erro. Os erros do Visual Basic (erros definidos pelo Visual Basic ou definidos pelo usuário) encontram-se no intervalo de 0 a 65535. Ao definir a propriedade <b>Número</b> para seu próprio código de erro em um módulo de classe, você adiciona seu

	número de código de erro à <u>constante</u> . <b>vbObjectError</b> . Por exemplo, para gerar o <u>número de erro</u> 1050, atribua <b>vbObjectError</b> + 1050 para a propriedade <b>Number</b> .
<b>source</b>	Opcional. <u>Expressão de seqüência de caracteres</u> que dá nome ao objeto ou aplicativo que gerou o erro. Ao definir essa <u>propriedade</u> para um objeto, use a forma <i>projeto.classe</i> . Se <b>source</b> não for especificada, é utilizada a identificação programática do <u>projeto</u> atual do Visual Basic.
<b>description</b>	Opcional. A expressão de seqüência de caracteres que descreve o erro. Se não for especificada, o valor em <b>Number</b> será examinado. Se ele puder ser mapeado para um código de erro de tempo de execução do Visual Basic, a seqüência de caracteres que iria ser retornada pela função <b>Error</b> será utilizada como <b>Description</b> . Se não houver erro no Visual Basic que corresponda a <b>Number</b> , será usada a mensagem "Erro de definição de aplicativo ou de definição de objeto".
<b>helpfile</b>	Opcional. O caminho completo para o arquivo de Ajuda do Microsoft Windows no qual pode ser encontrada ajuda sobre esse erro. Se não for especificado, o Visual Basic utilizará a unidade, o caminho e o nome do arquivo completo do arquivo de Ajuda do Visual Basic.
<b>helpcontext</b>	Opcional. A identificação de contexto que identifica um tópico dentro do <b>helpfile</b> que fornece ajuda para o erro. Se for omitido, será usada a identificação de contexto do arquivo de Ajuda do Visual Basic referente ao erro correspondente à propriedade <b>Number</b> , caso ele exista.

### Comentários

Todos os argumentos são opcionais com exceção de **number**. Se você usar **Raise** sem especificar alguns argumentos, e as configurações de propriedade do objeto **Err** contiverem valores que tenham sido removidos, esses valores servirão como os valores para o seu erro.

**Raise** é utilizado para gerar erros de tempo de execução e pode ser usado no lugar da instrução **Error**. **Raise** é útil para gerar erros ao gravar módulos de classe, pois o objeto **Err** fornece informações mais completas do que seria possível se você gerasse erros com a instrução **Error**. Por exemplo, com o método **Raise**, a fonte que gerou o erro pode ser especificada na propriedade **Source**, a Ajuda on-line para o erro pode ser referida e assim por diante.

## Método Remove

Remove um membro de um objeto **Collection**.

### Sintaxe

*objeto.Remove índice*

A sintaxe do método **Remove** tem o qualificador e parte do objeto a seguir:

<b>Parte</b>	<b>Descrição</b>
<i>objeto</i>	Obrigatório. Uma <u>expressão de objeto</u> que é avaliada como um objeto da lista Relativo a.
<i>índice</i>	Obrigatório. Uma <u>expressão</u> que especifica a posição de um membro da <u>coleção</u> . Se for uma <u>expressão numérica</u> , <i>índice</i> deve ser um número de 1 ao valor da <u>propriedade Count</u> da coleção. Se for uma <u>expressão de seqüência de caracteres</u> , <i>índice</i> deve corresponder ao <u>argumento key</u> especificado quando o membro referido tiver sido adicionado à coleção.

## Comentários

Se o valor fornecido como *índice* não corresponder a um membro existente da coleção, ocorrerá um erro.

### Exemplo da instrução AppActivate

Este exemplo ilustra vários usos da instrução **AppActivate** para ativar uma janela de aplicativo. As instruções **Shell** supõem que os aplicativos estão nos caminhos especificados.

```
Dim MeuCódigoDoAplicativo, ValorDeRetorno
' No Microsoft Windows:
AppActivate "Microsoft Word" ' Ativa o Microsoft
' Word.

' AppActivate também pode utilizar o valor de retorno da função Shell.
MeuCódigoDoAplicativo = Shell("C:\WORD\WINWORD.EXE", 1) ' Executa o
Microsoft Word.
AppActivate MeuCódigoDoAplicativo ' Ativa o Microsoft
' Word.

' Você pode também utilizar o valor de retorno da função Shell.
ValorDeRetorno = Shell("c:\EXCEL\EXCEL.EXE",1) ' Executa o Microsoft Excel.
AppActivate ValorDeRetorno ' Ativa o Microsoft
' Excel.
```

### Exemplo da instrução Beep

Este exemplo utiliza a instrução **Beep** para emitir três avisos sonoros consecutivos através do alto-falante do computador.

```
Dim I
For I = 1 To 3 ' Faz o loop 3 vezes.
    Beep ' Emite um aviso sonoro.
Next I
```

### Exemplo da função Command

Este exemplo utiliza a função **Command** para obter os argumentos da linha de comando em uma função que os retorna em uma **Variant** que contém uma matriz.

```
Function GetCommandLine(Optional ArgsMax)
'Declara as variáveis.
Dim C, LinhaDeComando, CompLinhaDeComando, ArgIn, I, ArgsNum
'Vê se ArgsMax foi fornecido.
If IsMissing(ArgsMax) Then ArgsMax = 10
'Torna a matriz do tamanho correto.
ReDim MatrizArg(ArgsMax)
ArgsNum = 0: ArgIn = False
'Obtém argumentos da linha de comando.
LinhaDeComando = Command()
CompLinhaDeComando = Len(LinhaDeComando)
'Percorre a linha de comando um caractere
'de cada vez.
For I = 1 To CompLinhaDeComando
    C = Mid(LinhaDeComando, I, 1)
    'Testa quanto a espaço ou tabulação.
    If (C <> " " And C <> vbTab) Then
```

```

'Nem espaço nem tabulação.
'Testa se já existe no argumento.
If Not ArgIn Then
'O novo argumento é iniciado.
'Testa quanto ao número excessivo de argumentos.
    If ArgsNum = ArgsMax Then Exit For
    ArgsNum = ArgsNum + 1
    ArgIn = True
End If
'Adiciona caractere ao argumento atual.
MatrizArg(ArgsNum) = MatrizArg(ArgsNum) + C
Else
'Encontrado um espaço ou tabulação.
'Define o sinalizador ArgIn como False.
ArgIn = False
End If
Next I
'Redimensiona a matriz o suficiente para conter argumentos.
ReDim Preserve MatrizArg(ArgsNum)
'Retorna a Matriz no nome da função.
GetCommandLine = MatrizArg()
End Function

```

### Exemplo da função InputBox

Este exemplo mostra várias maneiras de utilizar a função **InputBox** para solicitar ao usuário que insira um valor. Se as posições x e y forem omitidas, a caixa de diálogo será automaticamente centralizada em relação aos respectivos eixos. A variável `MeuValor` contém o valor inserido pelo usuário se o usuário clicar em **OK** ou pressionar a tecla ENTER. Se o usuário clicar em **Cancelar**, será retornada uma seqüência de comprimento zero.

```

Dim Mensagem, Título, Padrão, MeuValor
Mensagem = "Insira um valor entre 1 e 3" ' Define o aviso.
Título = "Demonstração da CaixaDeEntrada" ' Define o título.
Padrão = "1" ' Define o padrão.
' Exibe a mensagem, o título e o valor padrão.
MeuValor = InputBox(Mensagem, Título, Padrão)

' Utiliza o arquivo de Ajuda e o contexto. O botão Ajuda é adicionado
automaticamente.
MeuValor = InputBox(Mensagem, Título, , , , "DEMO.HLP", 10)

' Exibe a caixa de diálogo na posição 100, 100.
MeuValor = InputBox(Mensagem, Título, Padrão, 100, 100)

```

### Exemplo da função MsgBox

Este exemplo utiliza a função **MsgBox** para exibir uma mensagem de erro crítico em uma caixa de diálogo com os botões **Sim** e **Não**. O botão **Não** é especificado como a resposta padrão. O valor retornado pela função **MsgBox** depende do botão escolhido pelo usuário. Este exemplo supõe que DEMO.HLP é um arquivo de Ajuda que contém um tópico com um número de contexto da Ajuda igual a 1000.

```
Dim Msg, Estilo, Título, Ajuda, Ctxt, Resposta, MinhaSeqüência
Msg = "Deseja continuar?" ' Define a mensagem.
Estilo = vbYesNo + vbCritical + vbDefaultButton2 ' Define os botões.
Título = "Demonstração de MsgBox" ' Define o título.
Ajuda = "DEMO.HLP" ' Define o arquivo de Ajuda.
Ctxt = 1000 ' Define o contexto do
        ' tópico.
        ' Exibe a mensagem.
Resposta = MsgBox(Msg, Estilo, Título, Ajuda, Ctxt)
If Resposta = vbYes Then ' O usuário escolheu Sim.
    MinhaSeqüência = "Sim" ' Executa alguma ação.
Else ' O usuário escolheu Não.
    MinhaSeqüência = "Não" ' Executa alguma ação.
End If
```

### Exemplo da instrução SendKeys

Este exemplo utiliza a função **Shell** para executar o aplicativo Calculadora incluído no Microsoft Windows. Utiliza a instrução **SendKeys** para enviar pressionamentos de teclas para adicionar alguns números e, então, sair da Calculadora. (Para ver o exemplo, cole-o em um procedimento e, em seguida, execute o procedimento. Como AppActivate altera o foco para o aplicativo Calculadora, não é possível simplesmente percorrer o código.)

```
Dim ValorDeRetorno, I
ValorDeRetorno = Shell("CALC.EXE", 1) ' Executa a Calculadora.
AppActivate ValorDeRetorno ' Ativa a Calculadora.
For I = 1 To 100 ' Configura o loop de contagem.
    SendKeys I & "{+}", True ' Envia pressionamentos de teclas para
    Calculadora
Next I ' para adicionar cada valor de I.
SendKeys "=", True ' Obtém o total geral.
SendKeys "%{F4}", True ' Envia ALT+F4 para fechar a Calculadora.
```

## Instrução AppActivate

Ativa uma janela do aplicativo.

### Sintaxe

**AppActivate** *title* [, *wait*]

A sintaxe da instrução **AppActivate** possui os argumentos nomeados a seguir:

Parte	Descrição
<b>title</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que especifica o título na barra de título da janela do aplicativo que você deseja ativar. O código de tarefa retornado pela função <b>Shell</b> pode ser utilizado em vez de <b>title</b> para ativar um aplicativo.
<b>wait</b>	Opcional. Valor <b>Boolean</b> que especifica se o aplicativo de chamada possui o foco antes de ativar outro. Se for <b>False</b> (padrão), o aplicativo especificado é ativado imediatamente, mesmo se o aplicativo de chamada não possuir o foco. Se for

**True**, o aplicativo de chamada aguarda até possuir o foco e depois ativa o aplicativo especificado.

### Comentários

A instrução **AppActivate** muda o foco para o aplicativo ou a janela especificados, mas sem afetar seu estado maximizado ou minimizado. O foco se move da janela de aplicativo ativada quando o usuário realiza alguma ação para mudar o foco ou fechar a janela. Utilize a função **Shell** para iniciar um aplicativo e definir o estilo de janela.

Na determinação de qual aplicativo deve ser ativado, **title** é comparado à seqüência de caracteres de título de cada aplicativo que está em execução. Se não houver correspondência exata, todo aplicativo cuja seqüência de caracteres de título comece com **title** será ativado. Se houver mais de uma ocorrência do aplicativo especificado por **title**, uma ocorrência será arbitrariamente ativada.

## Instrução Beep

Emita um som pelo alto-falante do computador.

### Sintaxe

#### Beep

### Comentários

A freqüência e a duração do som dependem do hardware e do software do sistema e variam de um computador para outro.

## Função Command

Retorna a parte argumento da linha de comando utilizada para executar o Microsoft Visual Basic ou um programa executável desenvolvido com o Visual Basic.

### Sintaxe

#### Command

### Comentários

Quando o Visual Basic é executado a partir da linha de comando, qualquer parte da linha de comando que esteja após `/cmd` é passada ao programa como o argumento de linha de comando. No exemplo a seguir, `cmdlineargs` representa as informações de argumento retornadas pela função **Command**.

```
VB /cmd cmdlineargs
```

Para aplicativos desenvolvidos com o Visual Basic e compilados para um arquivo `.exe`, **Command** retorna qualquer argumento que apareça depois do nome do aplicativo na linha de comando. Por exemplo:

```
MyApp cmdlineargs
```

Para descobrir de que forma os argumentos de linha de comando podem ser alterados na interface do usuário do aplicativo que você está utilizando, procure **Argumentos de linha de comando** na **Ajuda**.

## Função InputBox

Exibe um aviso em uma caixa de diálogo, aguarda até que o usuário insira texto ou clique em um botão e retorna um **String** com o conteúdo da caixa de texto.

### Sintaxe

**InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])**

A sintaxe da função **InputBox** possui os argumentos nomeados a seguir:

Parte	Descrição
<b>prompt</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> exibida como mensagem na caixa de diálogo. O comprimento máximo de <b>prompt</b> é de aproximadamente 1024 caracteres, dependendo da largura dos caracteres utilizados. Se <b>prompt</b> consistir em mais de uma linha, você poderá separar as linhas utilizando um caractere de retorno de carro ( <b>Chr(13)</b> ), um caractere de alimentação de linha ( <b>Chr(10)</b> ) ou uma combinação de caracteres de retorno de carro e alimentação de linha ( <b>Chr(13) &amp; Chr(10)</b> ) entre cada linha.
<b>title</b>	Opcional. Expressão de seqüência de caracteres exibida na barra de título da caixa de diálogo. Se você omitir <b>title</b> , o nome do aplicativo será inserido na barra de título.
<b>default</b>	Opcional. Expressão de seqüência de caracteres exibida na caixa de texto como resposta padrão se nenhuma entrada for fornecida. Se você omitir <b>default</b> , a caixa de texto será exibida vazia.
<b>xpos</b>	Opcional. <u>Expressão numérica</u> que especifica, em twips, a distância horizontal da extremidade esquerda da caixa de diálogo em relação à extremidade esquerda da tela. Se <b>xpos</b> for omitido, a caixa de diálogo será centralizada horizontalmente.
<b>ypos</b>	Opcional. Expressão numérica que especifica, em twips, a distância vertical da extremidade superior da caixa de diálogo em relação ao topo da tela. Se <b>ypos</b> for omitido, a caixa de diálogo será posicionada na vertical a aproximadamente um terço da extremidade inferior da tela.
<b>helpfile</b>	Opcional. Expressão de seqüência de caracteres que identifica o arquivo de <b>Ajuda</b> a ser utilizado para fornecer ajuda sensível ao contexto relativa à caixa de diálogo. Se <b>helpfile</b> for fornecido, <b>context</b> também deverá ser fornecido.
<b>context</b>	Opcional. Expressão numérica que é o número de contexto da <b>Ajuda</b> atribuído ao tópico da <b>Ajuda</b> apropriado pelo autor da <b>Ajuda</b> . Se <b>context</b> for fornecido, <b>helpfile</b> também deverá ser fornecido.

### Comentários

Quando **helpfile** e **context** são fornecidos, o usuário pode pressionar F1 para exibir o tópico de **Ajuda** que corresponde ao **context**. Alguns aplicativos host, por exemplo, o Microsoft Excel, também adicionam automaticamente um botão **Ajuda** à caixa de diálogo. Se o usuário clicar em **OK** ou pressionar ENTER, a função **InputBox** retornará o que houver na caixa de texto. Se o usuário clicar em **Cancelar**, a função retornará uma seqüência de caracteres de comprimento zero ("").

**Observação** Para especificar mais que o primeiro argumento nomeado, você deve utilizar **InputBox** em uma expressão. Para omitir alguns argumentos posicionais, você deve incluir o delimitador de vírgula correspondente.

## Função MsgBox

Exibe uma mensagem em uma caixa de diálogo, aguarda que o usuário clique em um botão e retorna um **Integer** que indica qual botão o usuário clicou.

### Sintaxe

**MsgBox(prompt[, buttons] [, title] [, helpfile, context])**

A sintaxe da função **MsgBox** possui os argumentos nomeados a seguir:

Parte	Descrição
<b>prompt</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> exibida como mensagem na caixa de diálogo. O comprimento máximo de <b>prompt</b> é de aproximadamente 1024 caracteres, dependendo da largura dos caracteres utilizados. Se <b>prompt</b> consistir em mais de uma linha, você poderá separar as linhas utilizando um caractere de retorno de carro ( <b>Chr(13)</b> ), um caractere de alimentação de linha ( <b>Chr(10)</b> ) ou uma combinação de caracteres de retorno de carro e alimentação de linha ( <b>Chr(13) &amp; Chr(10)</b> ) entre cada linha.
<b>buttons</b>	Opcional. <u>Expressão numérica</u> que é a soma de valores que especifica o número e o tipo de botões a exibir, o estilo de ícone a utilizar, a identidade do botão padrão e a modalidade da caixa de mensagem. Se omitido, o valor padrão para <b>buttons</b> é 0.
<b>title</b>	Opcional. Expressão de seqüência de caracteres exibida na barra de título da caixa de diálogo. Se você omitir <b>title</b> , o nome do aplicativo será inserido na barra de título.
<b>helpfile</b>	Opcional. Expressão de seqüência de caracteres que identifica o arquivo de <b>Ajuda</b> a ser utilizado para fornecer a ajuda sensível ao contexto relativa à caixa de diálogo. Se <b>helpfile</b> for fornecido, <b>context</b> também deverá ser fornecido.
<b>context</b>	Opcional. Expressão numérica que é o número de contexto da <b>Ajuda</b> atribuído ao tópico de <b>Ajuda</b> apropriado pelo autor da <b>Ajuda</b> . Se <b>context</b> for fornecido, <b>helpfile</b> também deverá ser fornecido.

### Definições

As definições do argumento **buttons** são:

Constante	Valor	Descrição
<b>vbOKOnly</b>	0	Exibe somente o botão <b>OK</b> .
<b>VbOKCancel</b>	1	Exibe os botões <b>OK</b> e <b>Cancelar</b> .
<b>VbAbortRetryIgnore</b>	2	Exibe os botões <b>Abortar</b> , <b>Repetir</b> e <b>Ignorar</b> .
<b>VbYesNoCancel</b>	3	Exibe os botões <b>Sim</b> , <b>Não</b> e <b>Cancelar</b> .
<b>VbYesNo</b>	4	Exibe os botões <b>Sim</b> e <b>Não</b> .
<b>VbRetryCancel</b>	5	Exibe os botões <b>Repetir</b> e <b>Cancelar</b> .
<b>VbCritical</b>	16	Exibe o ícone <b>Mensagem crítica</b> .
<b>VbQuestion</b>	32	Exibe o ícone <b>Consulta de aviso</b> .
<b>VbExclamation</b>	48	Exibe o ícone <b>Mensagem de aviso</b> .
<b>VbInformation</b>	64	Exibe o ícone <b>Mensagem de informação</b> .
<b>VbDefaultButton1</b>	0	O primeiro botão é o padrão.
<b>VbDefaultButton2</b>	256	O segundo botão é o padrão.
<b>VbDefaultButton3</b>	512	O terceiro botão é o padrão.
<b>VbDefaultButton4</b>	768	O quarto botão é o padrão.

<b>VbApplicationModal</b>	0	Janela restrita do aplicativo; o usuário deve responder à caixa de mensagem antes de continuar o trabalho no aplicativo atual.
<b>VbSystemModal</b>	4096	Janela restrita do sistema; todos os aplicativos são suspensos até que o usuário responda à caixa de mensagem.

O primeiro grupo de valores (0–5) descreve o número e o tipo de botões exibidos na caixa de diálogo; o segundo grupo (16, 32, 48, 64) descreve o estilo de ícone; o terceiro grupo (0, 256, 512) determina qual botão é o padrão e o quarto grupo (0, 4096) determina a modalidade da caixa de mensagem. Quando estiver somando números para criar um valor final para o argumento **buttons**, utilize somente um número de cada grupo.

**Observação** Estas constantes são especificadas pelo Visual Basic para Aplicativos. Como resultado, os nomes podem ser utilizados em qualquer lugar do seu código em vez dos valores reais.

#### Valores de retorno

<b>Constante</b>	<b>Valor</b>	<b>Descrição</b>
<b>vbOK</b>	1	<b>OK</b>
<b>vbCancel</b>	2	<b>Cancelar</b>
<b>vbAbort</b>	3	<b>Abortar</b>
<b>vbRetry</b>	4	<b>Repetir</b>
<b>vbIgnore</b>	5	<b>Ignorar</b>
<b>vbYes</b>	6	<b>Sim</b>
<b>vbNo</b>	7	<b>Não</b>

#### Comentários

Quando **helpfile** e **context** são fornecidos, o usuário pode pressionar F1 para exibir o tópico da **Ajuda** correspondente ao **context**. Alguns aplicativos host, por exemplo, o Microsoft Excel, também adicionam automaticamente um botão **Ajuda** à caixa de diálogo.

Se a caixa de diálogo exibir um botão **Cancelar**, pressionar a tecla ESC terá o mesmo efeito que clicar em **Cancelar**. Se a caixa de diálogo contiver um botão **Ajuda**, será fornecida a ajuda sensível ao contexto relativa à caixa de diálogo. Entretanto, nenhum valor será retornado até que um dos outros botões seja clicado.

**Observação** Para especificar mais do que o primeiro argumento nomeado, você deve utilizar **MsgBox** em uma expressão. Para omitir algum argumento posicional, você deve incluir o delimitador de vírgula correspondente.

## Instrução SendKeys

Envia um ou mais pressionamentos de teclas para a janela ativa como se tivessem sido digitados no teclado.

#### Sintaxe

**SendKeys** *string* [, *wait*]

A sintaxe da instrução **SendKeys** possui os argumentos nomeados a seguir:

<b>Parte</b>	<b>Descrição</b>
<b>string</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que especifica os pressionamentos de teclas a enviar.
<b>wait</b>	Opcional. Valor <b>Boolean</b> que especifica o modo de espera. Se for <b>False</b> (padrão), o controle será retornado ao <u>procedimento</u> imediatamente após as teclas serem enviadas. Se for <b>True</b> , os pressionamentos de teclas devem ser processados antes que o controle seja retornado ao

procedimento.

### Comentários

Cada tecla é representada por um ou mais caracteres. Para especificar um único caractere de teclado, utilize o próprio caractere. Por exemplo, para representar a letra A, utilize "A" para **string**. Para representar mais de um caractere, anexe cada caractere adicional ao que o antecede. Para representar as letras A, B e C, utilize "ABC" para **string**.

O sinal de adição (+), circunflexo (^), sinal de porcentagem (%), til (~) e parênteses ( ) possuem significados especiais para **SendKeys**. Para especificar um desses caracteres, coloque-os entre chaves ({}). Por exemplo, para especificar o sinal de adição, utilize {+}. Os colchetes ([ ]) não têm significado especial para **SendKeys**, mas você deve colocá-los entre chaves. Em outros aplicativos, os colchetes têm um significado especial que pode ser importante quando ocorrer intercâmbio dinâmico de dados (DDE, *Dynamic Data Exchange*). Para especificar caracteres de chaves, utilize {-} e { }.

Para especificar caracteres que não são exibidos quando você pressiona uma tecla, como ENTER ou TAB, e teclas que representam ações em vez de caracteres, utilize os códigos a seguir.

Tecla	Código
BACKSPACE	{BACKSPACE}, {BS} ou {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL ou DELETE	{DELETE} ou {DEL}
SETA ABAIXO	{DOWN}
END	{END}
ENTER	{ENTER} ou ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS ou INSERT	{INSERT} ou {INS}
SETA À ESQUERDA	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
SETA À DIREITA	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
SETA ACIMA	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}

---

F15	{F15}
F16	{F16}

Para especificar teclas combinadas com qualquer combinação das teclas SHIFT, CTRL e ALT, preceda o código de teclas de um ou mais dos códigos a seguir:

<b>Tecla</b>	<b>Código</b>
SHIFT	+
CTRL	^
ALT	%

Para especificar que qualquer combinação de SHIFT, CTRL e ALT deve ser mantida pressionada enquanto várias outras teclas são pressionadas, coloque o código dessas teclas entre parênteses. Por exemplo, para especificar manter pressionada SHIFT enquanto E e C são pressionadas, utilize "+(EC)". Para especificar manter pressionada SHIFT enquanto E é pressionada, seguida de C sem SHIFT, utilize "+EC".

Para especificar a repetição de teclas, utilize a forma {tecla número}. Você deve colocar um espaço entre tecla e número. Por exemplo, {ESQUERDA 42} significa pressionar a tecla SETA À ESQUERDA 42 vezes; {h 10} significa pressionar H 10 vezes.

**Observação** Você não pode utilizar **SendKeys** para enviar pressionamentos de teclas para um aplicativo que não tenha sido projetado para ser executado no Microsoft Windows. **Sendkeys** também não pode enviar a tecla PRINT SCREEN {PRTSC} a nenhum aplicativo.

## Conjunto de caracteres (0–127)

0	•	32	[espaço]	64	@	96	`
1	•	33	!	65	A	97	a
2	•	34	"	66	B	98	b
3	•	35	#	67	C	99	c
4	•	36	\$	68	D	100	d
5	•	37	%	69	E	101	e
6	•	38	&	70	F	102	f
7	•	39	'	71	G	103	g
8	**	40	(	72	H	104	h
9	**	41	)	73	I	105	i
10	**	42	*	74	J	106	j
11	•	43	+	75	K	107	k
12	•	44	,	76	L	108	l
13	**	45	-	77	M	109	m
14	•	46	.	78	N	110	n
15	•	47	/	79	O	111	o
16	•	48	0	80	P	112	p
17	•	49	1	81	Q	113	q
18	•	50	2	82	R	114	r
19	•	51	3	83	S	115	s
20	•	52	4	84	T	116	t
21	•	53	5	85	U	117	u
22	•	54	6	86	V	118	v
23	•	55	7	87	W	119	w
24	•	56	8	88	X	120	x
25	•	57	9	89	Y	121	y
26	•	58	:	90	Z	122	z
27	•	59	;	91	[	123	{
28	•	60	<	92	\	124	
29	•	61	=	93	]	125	}
30	•	62	>	94	^	126	~
31	•	63	?	95	_	127	•

• Esses caracteres não são suportados pelo Microsoft Windows.

\*\* Os valores 8, 9, 10 e 13 são convertidos para os caracteres backspace, tab, alimentação de linha e retorno de carro, respectivamente. Eles não têm representação gráfica mas, dependendo do aplicativo, podem afetar a exibição visual do texto.

## Função IMEStatus

Retorna um **Integer** especificando o modo Editor de Método de Entrada (IME, *Input Method Editor*) atual do Microsoft Windows; disponível somente nas versões para o Extremo Oriente.

### Sintaxe

#### IMEStatus

#### Valores de retorno

Os valores de retorno referentes à localidade Japão são os seguintes:

Constante	Valor	Descrição
<b>vbIMENoOP</b>	0	Nenhum IME instalado
<b>vbIMEOn</b>	1	IME ligado
<b>vbIMEOff</b>	2	IME desligado
<b>vbIMEDisable</b>	3	IME desativado
<b>vbIMEHiragana</b>	4	Caracteres de byte duplo Hiragana (DBC)
<b>vbIMEKatakanaDbi</b>	5	Katakana DBC
<b>vbIMEKatakanaSng</b>	6	Caracteres de byte simples Katakana (SBC)
<b>vbIMEAlphaDbi</b>	7	DBC Alfanumérico
<b>vbIMEAlphaSng</b>	8	SBC Alfanumérico

Os valores de retorno referentes à localidade China (chinês tradicional e simplificado) são os seguintes:

Constante	Valor	Descrição
<b>vbIMENoOP</b>	0	Nenhum IME instalado
<b>vbIMEOn</b>	1	IME ligado
<b>vbIMEOff</b>	2	IME desligado

Para a localidade coreana, os primeiros cinco bits de retorno são definidos da seguinte forma:

Bit	Valor	Descrição	Valor	Descrição
0	0	Nenhum instalado	IME 1	IME instalado
1	0	IME desativado	1	IME ativado
2	0	Modo Inglês IME	1	Modo Hangeul
3	0	Modo Banja (SB)	1	Modo Junja (DB)
4	0	Modo Normal	1	Modo de Conversão Hanja

## Collection (Pesquisador de objeto)

O módulo **Collection** contém procedimentos utilizados para realizar operações no objeto **Collection**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## ColorConstants (Pesquisador de objeto)

O módulo **Color Constants** contém constantes de cores predefinidas. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## Constants (Pesquisador de objeto)

O módulo **Constants** contém várias constantes. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## Conversion (Pesquisador de objeto)

O módulo **Conversion** contém os procedimentos utilizados para executar várias operações de conversão. Estas constantes podem ser utilizadas em qualquer parte do seu código.

**Observação** Quando você utiliza as variáveis **Variant**, as conversões de tipo de dados explícitas não desnecessárias.

## DateTime (Pesquisador de objeto)

O módulo **DateTime** contém os procedimentos e propriedades utilizados em operações de data e hora. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## ErrObject (Pesquisador de objeto)

O módulo **ErrObject** contém propriedades e procedimentos utilizados para identificar e manipular erros de tempo de execução utilizando o objeto **Err**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## FileSystem (Pesquisador de objeto)

O módulo **FileSystem** contém os procedimentos utilizados para realizar operações de arquivo, diretório ou pasta e sistema. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## Financial (Pesquisador de objeto)

O módulo **Financial** contém procedimentos utilizados para realizar operações financeiras. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## Global (Pesquisador de objeto)

O módulo **Global** contém procedimentos e propriedades utilizadas para realizar operações no objeto **UserForm**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## Information (Pesquisador de objeto)

O módulo **Information** contém os procedimentos utilizados para retornar, testar ou verificar informações. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## Interaction (Pesquisador de objeto)

O módulo **Interaction** contém procedimentos utilizados para interagir com objetos, aplicativos e sistemas. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## eycodeConstants (Pesquisador de objeto)

O módulo **KeyCodeConstants** contém constantes de código-chave predefinidas que podem ser utilizadas em qualquer parte do seu código.

## Math (Pesquisador de objeto)

O módulo **Math** contém procedimentos utilizados para realizar operações matemáticas. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## String (Pesquisador de objeto)

O módulo **String** contém procedimentos utilizados para realizar operações de seqüências de caracteres. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## SystemColorConstants (Pesquisador de objeto)

O módulo **System ColorConstants** contém constantes que identificam várias partes da interface de usuário gráfica. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## VbAppWinStyle (Pesquisador de objeto)

A enumeração **VbAppWinStyle** contém constantes utilizadas pela função **Shell** para controlar o estilo de uma janela de aplicativo. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## VbCalendar (Pesquisador de objeto)

A enumeração **VbCalendar** contém constantes utilizadas para determinar o tipo de calendário utilizado pelo Visual Basic. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## VbCompareMethod (Pesquisador de objeto)

A enumeração **VbCompareMethod** contém constantes utilizadas para determinar a forma como as seqüências de caracteres são comparadas ao utilizar as funções **Instr** e **StrComp**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

## VbDayOfWeek (Pesquisador de objeto)

A enumeração **VbDayOfWeek** contém constantes utilizadas para identificar dias da semana específicos ao utilizar as funções **DateDiff**, **DatePart** e **Weekday**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

### VbFileAttribute (Pesquisador de objeto)

A enumeração **VbFileAttribute** contém constantes utilizadas para identificar atributos de arquivos utilizados nas funções **Dir**, **GetAttr** e **SetAttr**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

### VbFirstWeekOfYear (Pesquisador de objeto)

A enumeração **VbFirstWeekOfYear** contém constantes utilizadas para identificar como a primeira semana de um ano é determinada quando estiver utilizando as funções **DateDiff** e **DatePart**. Estas constantes podem ser utilizados em qualquer outra parte do código.

### VbIMEStatus (Pesquisador de objeto)

Disponível apenas nas versões do Oriente Médio, a enumeração **VbIMEStatus** contém constantes utilizadas para identificar o Input Method Editor (IME) quando estiver utilizando a função **IMEStatus**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

### VbMsgBoxResult (Pesquisador de objeto)

A enumeração **VbMsgBoxResult** contém constantes utilizadas para identificar que botão foi pressionado em uma caixa de mensagem exibida utilizando a função **MsgBox**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

### VbMsgBoxStyle (Pesquisador de objeto)

A enumeração **VbMsgBoxStyle** contém constantes utilizadas para especificar o comportamento de uma caixa de mensagem, junto com símbolos e botões que aparecem nela, quando exibida utilizando a função **MsgBox**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

### VbQueryClose (Pesquisador de objeto)

A enumeração **VbQueryClose** contém constantes utilizadas para identificar o que causou o evento **QueryClose**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

### VbStrConv (Pesquisador de objeto)

A enumeração **VbStrConv** contém constantes utilizadas para identificar o tipo de conversão de seqüência de caracteres a ser realizada pela função **StrConv**. Estas constantes podem ser utilizadas em qualquer parte do seu código.

### VbVarType (Pesquisador de objeto)

A enumeração **VbVarType** contém constantes utilizadas para identificar os vários tipos de dados que podem estar contidos em uma **Variante**. Estas constantes correspondem aos valores de retorno da função **VarType** e pode ser utilizado em qualquer parte do seu código.

### Exemplo do objeto Collection

Este exemplo cria um objeto **Collection** (MinhasClasses) e, em seguida, cria uma caixa de diálogo na qual os usuários podem adicionar objetos à coleção. Para ver como isto funciona, escolha o comando **Módulo de classe** no menu **Inserir** e declare uma variável pública denominada NomeDaInstância em nível de módulo de Class 1 (digite **Public** NomeDaInstância) para conter o nome de cada instância. Deixe o nome padrão como Class1. Copie e cole o código a seguir na seção Geral de outro módulo e, em seguida, inicie-o com a instrução ClassNamer em outro procedimento. (Este exemplo funciona somente com aplicativos host que suportam classes.)

```
Sub ClassNamer()
    Dim MinhasClasses As New Collection ' Cria um objeto Collection.
    Dim Num ' Contador para chaves de individualização.
    Dim Msg As String ' Variável para conter seqüência de caracteres de
    aviso.
    Dim ONome, MeuObjeto, ListaDeNomes ' Variants para conter
    informações.
    Do
        Dim Inst As New Class1 ' Cria uma nova instância de Class1.
        Num = Num + 1 ' Incrementa Num e, em seguida, obtém um nome.
        Msg = "Insira um nome para este objeto." & Chr(13) _
            & "Pressione Cancelar para ver os nomes na coleção.
        ONome = InputBox(Msg, "Nomeie a coleção de Items")
        Inst.NomeDaInstância = ONome ' Coloca o nome na instância do objeto.
        ' Se o usuário inseriu um nome, adiciona-o à coleção.
        If Inst.NomeDaInstância <> "" Then
            ' Adiciona o objeto nomeado à coleção.
            MinhasClasses.Add item := Inst, key := CStr(Num)
        End If
        ' Limpa a referência atual em preparação para a próxima.
        Set Inst = Nothing
    Loop Until ONome = ""
    For Each MeuObjeto In MinhasClasses ' Cria a lista de nomes.
        ListaDeNomes = ListaDeNomes & MeuObjeto.NomeDaInstância & Chr(13)
    Next MeuObjeto
    ' Exibe a lista de nomes em uma caixa de mensagem.
    MsgBox ListaDeNomes, , "Nomes de instância na coleção MinhasClasses"

    For Num = 1 To MinhasClasses.Count ' Remove o nome da coleção.
        MinhasClasses.Remove 1 ' Como as coleções são reindexadas
            ' automaticamente, remove o primeiro
    Next ' membro de cada iteração.
End Sub
```

### Exemplo do objeto Err

Este exemplo utiliza as propriedades do objeto **Err** na construção de uma caixa de diálogo de mensagem de erro. Observe que se você utilizar primeiro o método **Clear**, ao gerar um erro do Visual Basic com o método **Raise**, os valores padrão do Visual Basic se tornarão as propriedades do objeto **Err**.

```
Dim Msg
' Se ocorrer um erro, construa uma mensagem de erro
On Error Resume Next ' Adia o tratamento do erro.
Err.Clear
Err.Raise 6 ' Gera um erro "Sobrecarga".
' Procura pelo erro e, em seguida, exibe a mensagem.
If Err.Number <> 0 Then
    Msg = "Erro # " & Str(Err.Number) & " foi gerado por " _
        & Err.Source & Chr(13) & Err.Description
    MsgBox Msg, , "Erro", Err.Helpfile, Err.HelpContext
End If
```

## Objeto Collection

Um objeto **Collection** é um conjunto de itens ordenados que podem ser referidos como uma unidade.

### Comentários

O objeto **Collection** proporciona uma forma conveniente de se fazer referência a um grupo relacionado de itens como um objeto único. Os itens, ou membros, de uma coleção precisam ser relacionados somente pelo fato de existirem em uma coleção. Os membros de uma coleção não precisam compartilhar o mesmo tipo de dados.

Uma coleção pode ser criada da mesma forma que os outros objetos são criados. Por exemplo:

```
Dim X As Nova Collection
```

Depois de uma coleção ser criada, os membros podem ser adicionados usando-se o método **Add** e removidos usando-se o método **Remove**. Os membros específicos podem ser retornados da coleção usando-se o método **Item**, enquanto toda a coleção pode ser iterada utilizando-se a instrução **For Each...Next**.

## Objeto Debug

O objeto **Debug** envia a saída para a janela **Imediato** durante o tempo de execução.

## Objeto Err

Contém informações sobre erros em tempo de execução.

### Comentários

As propriedades do objeto **Err** são definidas pelo gerador de um erro — o Visual Basic, um objeto ou um programador de Visual Basic.

A propriedade padrão do objeto **Err** é **Number**. Como a propriedade padrão pode ser representada pelo nome do objeto **Err**, o código escrito anteriormente utilizando a função **Err** ou a instrução **Err** não precisa ser modificado.

Quando ocorre um erro em tempo de execução as propriedades do objeto **Err** são preenchidas com informações que identificam o erro de forma exclusiva e informações que podem ser utilizadas para manipular esse erro. Para gerar um erro em tempo de execução em seu código, use o método **Raise**.

As propriedades do objeto **Err** são redefinidas como zero ou seqüências de comprimento zero (""), após qualquer forma da instrução **Resume** ou **On Error** e após uma instrução **Exit Sub**, **Exit Function** ou **Exit Property** dentro de uma rotina de manipulação de erro. O método **Clear** pode ser usado para redefinir **Err** de forma explícita.

Use o método **Raise**, em vez da instrução **Error**, para gerar erros de tempo de execução para um módulo de classe. Usar o método **Raise** em outro código depende da riqueza das informações que você quer retornar. Em códigos que utilizam instruções **Error** em vez do método **Raise** para gerar erros, as propriedades do objeto **Err** são atribuídas aos seguintes valores padrão quando **Error** é executado:

Propriedade	Valor
<b>Number</b>	Valor especificado como <u>argumento</u> para a instrução <b>Error</b> . Pode ser qualquer <u>número de erro</u> válido.
<b>Source</b>	Nome do <u>projeto</u> atual do Visual Basic.
<b>Description</b>	Uma seqüência de caracteres correspondente ao retorno da função <b>Error</b> para o <b>Number</b> especificado, se essa seqüência existir. Se não existir, a <b>Description</b> conterá "Erro de definição de aplicativo ou de definição de objeto".

<b>HelpFile</b>	Unidade de disco, caminho e nome de arquivo completos do arquivo de Ajuda do Visual Basic.
<b>HelpContext</b>	Identificação do contexto do arquivo de Ajuda do Visual Basic referente ao erro correspondente à propriedade <b>Number</b> .
<b>LastDLLError</b>	Somente em sistemas operacionais Microsoft Windows de 32 bits, contém o código de erro de sistema referente à última chamada a uma <u>biblioteca de vínculo dinâmico</u> (DLL). A propriedade <b>LastDLLError</b> é somente leitura.

Você não precisa alterar o código existente que utiliza o objeto **Err** e a instrução **Error**. Contudo, utilizar o objeto **Err** e a instrução **Error** pode resultar em conseqüências não desejadas. Por exemplo, mesmo que você preencha as propriedades referentes ao objeto **Err**, elas serão redefinidas com os valores padrão indicados na tabela anterior assim que a instrução **Error** for executada. Embora você possa utilizar a instrução **Error** para gerar erros em tempo de execução do Visual Basic, ela é retida principalmente para haver compatibilidade com o código existente. Utilize o objeto **Err**, o método **Raise** e o método **Clear** para os erros do sistema e em código novo, especialmente para módulos de classe.

O objeto **Err** é um objeto intrínseco com escopo global. Não há necessidade de se criar uma instância dele em seu código.

### Exemplo do operador ^

Este exemplo utiliza o operador ^ para elevar um número à potência de um expoente.

```
Dim MeuValor
MeuValor = 2 ^ 2 ' Retorna 4.
MeuValor = 3 ^ 3 ^ 3 ' Retorna 19683.
MeuValor = (-5) ^ 3 ' Retorna -125.
```

### Exemplo do operador +

Este exemplo utiliza o operador + para somar números. O operador + também pode ser utilizado para concatenar seqüências de caracteres. Entretanto, para eliminar a ambigüidade, você deve utilizar o operador & em seu lugar. Se os componentes de uma expressão criada com o operador + incluírem tanto seqüências de caracteres como numéricas, será atribuído o resultado aritmético. Se os componentes forem exclusivamente seqüências de caracteres, as seqüências de caracteres serão concatenadas.

```
Dim MeuNúmero, Var1, Var2
MeuNúmero = 2 + 2 ' Retorna 4.
MeuNúmero = 4257.04 + 98112 ' Retorna 102369,04.

Var1 = "34": Var2 = 6 ' Inicializa variáveis mistas.
MeuNúmero = Var1 + Var2 ' Retorna 40.

Var1 = "34": Var2 = "6" ' Inicializa variáveis com seqüências de
caracteres.
MeuNúmero = Var1 + Var2 ' Retorna "346" (concatenação de seqüências de
caracteres).
```

### Exemplo do operador -

Este exemplo utiliza o operador - para calcular a diferença entre dois números.

```
Dim MeuResultado
MeuResultado = 4 - 2 ' Retorna 2.
MeuResultado = 459.35 - 334.90 ' Retorna 124,45.
```

### Exemplo do operador \*

Este exemplo utiliza o operador \* para multiplicar dois números.

```
Dim MeuValor
MeuValor = 2 * 2 ' Retorna 4.
MeuValor = 459.35 * 334.90 ' Retorna 153836,315.
```

### Exemplo do operador /

Este exemplo utiliza o operador / para executar divisão de vírgula flutuante.

```
Dim MeuValor
MeuValor = 10 / 4 ' Retorna 2,5.
MeuValor = 10 / 3 ' Retorna 3,333333.
```

### Exemplo do operador \

Este exemplo utiliza o operador \ para executar a divisão de inteiros.

```
Dim MeuValor
MeuValor = 11 \ 4 ' Retorna 2.
MeuValor = 9 \ 3 ' Retorna 3.
MeuValor = 100 \ 3 ' Retorna 33.
```

### Exemplo do operador Mod

Este exemplo utiliza o operador **Mod** para dividir dois números e retornar somente o resto. Se algum dos números for de vírgula flutuante, ele será primeiro arredondado para inteiro.

```
Dim MeuResultado
MeuResultado = 10 Mod 5 ' Retorna 0.
MeuResultado = 10 Mod 3 ' Retorna 1.
MeuResultado = 12 Mod 4.3 ' Retorna 0.
MeuResultado = 12.6 Mod 5 ' Retorna 3.
```

### Exemplo do operador &

Este exemplo utiliza o operador & para forçar a concatenação de seqüências de caracteres.

```
Dim MinhaSeq
MinhaSeq = "Alô" & " Mundo" ' Retorna "Alô Mundo".
MinhaSeq = "Controle " & 123 & " Controle" ' Retorna "Controle 123
Controle".
```

**Exemplo de operadores de comparação**

Este exemplo mostra vários usos dos operadores de comparação, que você utiliza para comparar expressões.

```
Dim MeuResultado, Var1, Var2
MeuResultado = (45 < 35) ' Retorna False.
MeuResultado = (45 = 45) ' Retorna True.
MeuResultado = (4 <> 3) ' Retorna True.
MeuResultado = ("5" > "4") ' Retorna True.

Var1 = "5": Var2 = 4 ' Inicializa as variáveis.
MeuResultado = (Var1 > Var2) ' Retorna True.

Var1 = 5: Var2 = Empty
MeuResultado = (Var1 > Var2) ' Retorna True.

Var1 = 0: Var2 = Empty
MeuResultado = (Var1 = Var2) ' Retorna True.
```

**Exemplo do operador And**

Este exemplo utiliza o operador **And** para executar uma conjunção lógica em duas expressões.

```
Dim A, B, C, D, MeuControle
A = 10: B = 8: C = 6: D = Null ' Inicializa as variáveis.
MeuControle = A > B And B > C ' Retorna True.
MeuControle = B > A And B > C ' Retorna False.
MeuControle = A > B And B > D ' Retorna Null.
MeuControle = A And B ' Retorna 8 (comparação bit a bit).
```

**Exemplo do operador Eqv**

Este exemplo utiliza o operador **Eqv** para executar a equivalência lógica em duas expressões.

```
Dim A, B, C, D, MeuControle
A = 10: B = 8: C = 6: D = Null ' Inicializa as variáveis.
MeuControle = A > B Eqv B > C ' Retorna True.
MeuControle = B > A Eqv B > C ' Retorna False.
MeuControle = A > B Eqv B > D ' Retorna Null.
MeuControle = A Eqv B ' Retorna -3 (comparação bit a bit).
```

**Exemplo do operador Imp**

Este exemplo utiliza o operador **Imp** para executar a implicação lógica em duas expressões.

```
Dim A, B, C, D, MeuControle
A = 10: B = 8: C = 6: D = Null ' Inicializa as variáveis.
MeuControle = A > B Imp B > C ' Retorna True.
MeuControle = A > B Imp C > B ' Retorna False.
MeuControle = B > A Imp C > B ' Retorna True.
MeuControle = B > A Imp C > D ' Retorna True.
MeuControle = C > D Imp B > A ' Retorna Null.
MeuControle = B Imp A ' Retorna -1 (comparação bit a bit).
```

**Exemplo do operador Not**

Este exemplo utiliza o operador **Not** para executar a negação lógica em uma expressão.

```
Dim A, B, C, D, MeuControle
A = 10: B = 8: C = 6: D = Null      ' Inicializa as variáveis.
MeuControle = Not (A > B) ' Retorna False.
MeuControle = Not (B > A) ' Retorna True.
MeuControle = Not (C > D) ' Retorna Null.
MeuControle = Not A      ' Retorna -11 (comparação bit a bit).
```

**Exemplo do operador Or**

Este exemplo utiliza o operador **Or** para executar a disjunção lógica em duas expressões.

```
Dim A, B, C, D, MeuControle
A = 10: B = 8: C = 6: D = Null      ' Inicializa as variáveis
MeuControle = A > B Or B > C ' Retorna True.
MeuControle = B > A Or B > C ' Retorna True.
MeuControle = A > B Or B > D ' Retorna True.
MeuControle = B > D Or B > A ' Retorna Null.
MeuControle = A Or B      ' Retorna 10 (comparação bit a bit).
```

**Exemplo do operador Xor**

Este exemplo utiliza o operador **Xor** para executar a exclusão lógica em duas expressões.

```
Dim A, B, C, D, MeuControle
A = 10: B = 8: C = 6: D = Null      ' Inicializa as variáveis.
MeuControle = A > B Xor B > C ' Retorna False.
MeuControle = B > A Xor B > C ' Retorna True.
MeuControle = B > A Xor C > B ' Retorna False.
MeuControle = B > D Xor A > B ' Retorna Null.
MeuControle = A Xor B      ' Retorna 2 (comparação bit a bit).
```

**Exemplo do operador Like**

Este exemplo utiliza o operador **Like** para comparar uma seqüência de caracteres com um padrão.

```
Dim MeuControle
MeuControle = "aBBa" Like "a*a"      ' Retorna True.
MeuControle = "F" Like "[A-Z]"      ' Retorna True.
MeuControle = "F" Like "[!A-Z]"     ' Retorna False.
MeuControle = "a2a" Like "a#a"     ' Retorna True.
MeuControle = "aM5b" Like "a[L-P]#[!c-e]" ' Retorna True.
MeuControle = "BAT123khg" Like "B?T*" ' Retorna True.
MeuControle = "CAT123khg" Like "B?T*" ' Retorna False.
```

**Exemplo do operador Is**

Este exemplo utiliza o operador **Is** para comparar duas referências a objeto. Os nomes de variáveis de objeto são genéricos e usados somente com finalidade ilustrativa.

```
Dim MeuObjeto, SeuObjeto, EsteObjeto, OutroObjeto, AqueleObjeto,
MeuControle
Set SeuObjeto = MeuObjeto ' Atribui referências a objeto.
Set EsteObjeto = MeuObjeto
Set AqueleObjeto = OutroObjeto
```

```

MeuControle = SeuObjeto Is EsteObjeto      ' Retorna True.
MeuControle = AqueleObjeto Is EsteObjeto  ' Retorna False.
' Pressupõe que MeuObjeto <> OutroObjeto
MeuControle = MeuObjeto Is AqueleObjeto   ' Retorna False.

```

## Resumo de operadores

<b>Operadores</b>	<b>Descrição</b>
<u>Operadores aritméticos</u>	Operadores utilizados para efetuar cálculos matemáticos.
<u>Operadores de comparação</u>	Operadores utilizados para efetuar comparações.
<u>Operadores de concatenação</u>	Operadores utilizados para combinar seqüências de caracteres.
<u>Operadores lógicos</u>	Operadores utilizados para efetuar operações lógicas.

## Precedência de operadores

Quando diversas operações ocorrem em uma expressão, cada parte é avaliada e resolvida em uma ordem predeterminada chamada precedência de operadores.

Quando as expressões contêm operadores de mais de uma categoria, os operadores aritméticos são avaliados em primeiro lugar, seguidos pelos operadores de comparação e, finalmente, pelos operadores lógicos. Os operadores de comparação apresentam todos a mesma precedência, ou seja, são avaliados da esquerda para a direita na ordem em que aparecem. Os operadores lógicos e os aritméticos são avaliados na ordem de precedência a seguir:

<b>Aritméticos</b>	<b>De comparação</b>	<b>Lógicos</b>
Exponenciação (^)	Igualdade (=)	<b>Not</b>
Negação (-)	Desigualdade (<>)	<b>And</b>
Multiplicação e divisão (*, /)	Menor que (<)	<b>Or</b>
Divisão de inteiros (\)	Maior que (>)	<b>Xor</b>
Módulo aritmético ( <b>Mod</b> )	Menor que ou igual a (<=)	<b>Eqv</b>
Adição e subtração (+, -)	Maior que ou igual a (>=)	<b>Imp</b>
Concatenação de seqüências de caracteres (&)	<b>Like</b>	
	<b>Is</b>	

Quando a multiplicação e a divisão ocorrem ao mesmo tempo em uma expressão, cada operação é avaliada à medida que ela ocorre da esquerda para a direita. Quando a adição e a subtração ocorrem ao mesmo tempo em uma expressão, cada operação é avaliada na ordem em que aparece da esquerda para a direita. Parênteses podem ser utilizados para ignorar a ordem de precedência e fazer com que algumas partes da expressão sejam avaliadas antes de outras. As operações entre parênteses são sempre executadas antes daquelas que se encontram fora deles. Dentro dos parênteses, contudo, a precedência de operadores é mantida.

O operador de concatenação de seqüências de caracteres (&) não é um operador aritmético, mas segue todos os operadores aritméticos e precede todos os operadores de comparação.

O operador **Like** apresenta a mesma precedência de todos os operadores de comparação, mas é, na verdade, um operador de correspondência de padrão.

O operador **Is** é um operador de comparação de referência de objeto. Ele não compara objetos ou seus valores; somente verifica e determina se duas referências de objeto referem-se a um mesmo objeto.

## Operador ^

Utilizado para elevar um número à potência de um expoente.

### Sintaxe

*resultado* = *número*^*expoente*

A sintaxe do operador ^ possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>número</i>	Obrigatório; qualquer <u>expressão numérica</u> .
<i>expoente</i>	Obrigatório; qualquer expressão numérica.

### Comentários

Um *número* pode ser negativo somente se *expoente* for um valor inteiro. Quando mais de uma exponenciação é efetuada em uma única expressão, o operador ^ é avaliado à medida que é encontrado da esquerda para a direita.

Em geral, o tipo de dados de *resultado* é um **Double** ou um **Variant** contendo um **Double**. No entanto, se *número* ou *expoente* for uma expressão **Null**, *resultado* será **Null**.

## Operador +

Utilizado para somar dois números.

### Sintaxe

*resultado* = *expressão1*+*expressão2*

A sintaxe do operador + possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>expressão1</i>	Obrigatório; qualquer <u>expressão</u> .
<i>expressão2</i>	Obrigatório; qualquer expressão.

### Comentários

Quando você utiliza o operador +, pode não ser capaz de determinar se ocorrerá a adição ou a concatenação de seqüências de caracteres. Utilize o operador & para a concatenação de forma a eliminar a ambigüidade e fornecer código autodocumentável.

Se pelo menos uma expressão não for **Variant**, as regras a seguir se aplicam:

Se	Então
Ambas as expressões são <u>tipos de dados numéricos</u> ( <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>Currency</b> ou <b>Decimal</b> )	Adiciona.
Ambas as expressões são <b>String</b>	Concatena.
Uma expressão for um tipo de dados numérico e a outra for qualquer <b>Variant</b> exceto <b>Null</b>	Adiciona.
Uma expressão for um <b>String</b> e a outra for qualquer <b>Variant</b> exceto <b>Null</b>	Concatena.
Uma expressão for um <b>Variant Empty</b>	Retorna a outra expressão inalterada como <i>resultado</i> .
Uma expressão for um tipo de dados numérico e a outra for um <b>String</b>	Um erro <code>Type mismatch</code> ocorre.
Qualquer uma das expressões for <b>Null</b>	<i>resultado</i> é <b>Null</b> .

Se ambas as expressões forem **Variant**, as regras a seguir se aplicam:

Se	Então
----	-------

Ambas as expressões <b>Variant</b> forem numéricas	Adiciona.
Ambas as expressões <b>Variant</b> forem seqüências de caracteres	Concatena.
Uma expressão <b>Variant</b> for numérica e a outra for uma seqüência de caracteres	Adiciona.

Para uma adição aritmética simples envolvendo somente expressões de tipos de dados numéricos, o tipo de dados de *resultado* é geralmente o mesmo da expressão mais precisa. A ordem de precisão, da menos para a mais precisa, é **Byte**, **Integer**, **Long**, **Single**, **Double**, **Currency** e **Decimal**. A seguir são apresentadas exceções a essa ordem:

<b>Se</b>	<b>Então resultado será</b>
Um <b>Single</b> e um <b>Long</b> são adicionados,	um <b>Double</b> .
O tipo de dados de <i>resultado</i> for uma variante de <b>Long</b> , <b>Single</b> ou <b>Date</b> que ultrapasse o intervalo permitido,	convertido a uma variante de <b>Double</b> .
O tipo de dados de <i>resultado</i> for uma variante de <b>Byte</b> que ultrapasse o intervalo permitido,	convertido a uma variante de <b>Integer</b> .
O tipo de dados de <i>resultado</i> for uma variante de <b>Integer</b> que ultrapasse o intervalo permitido,	convertido a uma variante de <b>Long</b> .
Uma <b>Date</b> for adicionada a qualquer tipo de dados,	um <b>Date</b> .

Se uma ou ambas as expressões forem **Null**, *resultado* será **Null**. Se ambas as expressões forem **Empty**, *resultado* será um **Integer**. No entanto, se somente uma expressão for **Empty**, a outra será retornada inalterada como *resultado*.

**Observação** A ordem de precisão utilizada para adição e subtração não é a mesma utilizada pela multiplicação.

## Operador –

Utilizado para encontrar a diferença entre dois números ou para indicar o valor negativo em uma expressão numérica.

### Sintaxe 1

*resultado* = número1–número2

### Sintaxe 2

–número

A sintaxe do operador – possui as partes a seguir:

<b>Parte</b>	<b>Descrição</b>
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>número</i>	Obrigatório; qualquer expressão numérica.
<i>número1</i>	Obrigatório; qualquer expressão numérica.
<i>número2</i>	Obrigatório; qualquer expressão numérica.

### Comentários

Em Sintaxe 1, o operador – é o operador aritmético de subtração utilizado para encontrar a diferença entre dois números. Em Sintaxe 2, o operador – é utilizado como operador de negação unário para indicar o valor negativo de uma expressão.

O tipo de dados de *resultado* é geralmente o mesmo da expressão mais precisa. A ordem de precisão, da menos para a mais precisa, é **Byte**, **Integer**, **Long**, **Single**, **Double**, **Currency** e

**Decimal.** A seguir estão algumas exceções a essa ordem:

<b>Se</b>	<b>Então resultado será</b>
A subtração envolve um <b>Single</b> e um <b>Long</b> ,	convertido para <b>Double</b> .
O tipo de dados de <i>resultado</i> for um variante de <b>Long</b> , <b>Single</b> ou <b>Date</b> que ultrapasse o intervalo permitido,	convertido para um <b>Variant</b> que contém um <b>Double</b> .
O tipo de dados de <i>resultado</i> for uma variante de <b>Byte</b> que ultrapasse o intervalo permitido,	convertido para uma variante de <b>Integer</b> .
O tipo de dados de <i>resultado</i> for uma variante de <b>Integer</b> que ultrapasse o intervalo permitido,	convertido para uma variante de <b>Long</b> .
A subtração envolve um <b>Date</b> e qualquer outro tipo de dados,	um <b>Date</b> .
A subtração envolve duas expressões <b>Date</b> ,	um <b>Double</b> .

Se uma ou ambas as expressões forem **Null**, *resultado* será **Null**. Se uma expressão for **Empty**, ela será tratada como 0.

**Observação** A ordem de precisão utilizada pela adição e subtração não é a mesma utilizada pela multiplicação.

## Operador \*

Utilizado para multiplicar dois números.

### Sintaxe

*resultado* = número1\*número2

A sintaxe do operador \* possui as partes a seguir:

<b>Parte</b>	<b>Descrição</b>
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>número1</i>	Obrigatório; qualquer <u>expressão numérica</u> .
<i>número2</i>	Obrigatório; qualquer expressão numérica.

### Comentários

O tipo de dados de *resultado* é geralmente o mesmo da expressão mais precisa. A ordem de precisão, da menos para a mais precisa, é **Byte**, **Integer**, **Long**, **Single**, **Currency**, **Double** e **Decimal**. A seguir estão algumas exceções a essa ordem:

<b>Se</b>	<b>Então resultado será</b>
A multiplicação envolve um <b>Single</b> e um <b>Long</b> ,	convertido para um <b>Double</b> .
O tipo de dados de <i>resultado</i> for uma variante de <b>Long</b> , <b>Single</b> ou <b>Date</b> que ultrapasse o intervalo permitido,	convertido para um <b>Variant</b> contendo um <b>Double</b> .
O tipo de dados de <i>resultado</i> for uma variante de <b>Byte</b> que ultrapasse o intervalo permitido,	convertido para uma variante de <b>Integer</b> .
O tipo de dados de <i>resultado</i> for uma variante de <b>Integer</b> que ultrapasse o intervalo permitido,	convertido para uma variante de <b>Long</b> .

Se uma ou ambas as expressões forem **Null**, *resultado* será **Null**. Se uma expressão for **Empty**, ela será tratada como 0.

**Observação** A ordem de precisão utilizada pela multiplicação não é a mesma utilizada pela adição e subtração.

## Operador /

Utilizado para dividir dois números e retornar um resultado de vírgula flutuante.

### Sintaxe

*resultado* = *número1/número2*

A sintaxe do operador / possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>número1</i>	Obrigatório; qualquer <u>expressão numérica</u> .
<i>número2</i>	Obrigatório; qualquer expressão numérica.

### Comentários

O tipo de dados de *resultado* é geralmente um **Double** ou uma variante de **Double**. A seguir são apresentadas exceções a essa regra:

Se	Então <i>resultado</i> será
Ambas as <u>expressões</u> forem expressões <b>Byte</b> , <b>Integer</b> ou <b>Single</b> ,	<b>Single</b> , a menos que ele ultrapasse o intervalo permitido; nesse caso, um erro será gerado.
Ambas as expressões forem variantes de <b>Byte</b> , <b>Integer</b> ou <b>Single</b> ,	uma variante de <b>Single</b> , a menos que ela ultrapasse o intervalo permitido; nesse caso, <i>resultado</i> será um <b>Variant</b> contendo um <b>Double</b> .
A divisão envolver um <b>Decimal</b> e qualquer outro tipo de dados,	um tipo de dados <b>Decimal</b> .

Se uma ou ambas as expressões forem **Null**, *resultado* será **Null**. Qualquer expressão que seja **Empty** será tratada como 0.

## Operador \

Utilizado para dividir dois números e retornar um resultado inteiro.

### Sintaxe

*resultado* = *número1\número2*

A sintaxe do operador \ possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>número1</i>	Obrigatório; qualquer <u>expressão numérica</u> .
<i>número2</i>	Obrigatório; qualquer expressão numérica.

### Comentários

Antes de a divisão ser realizada, as expressões numéricas são arredondadas para expressões **Byte**, **Integer** ou **Long**.

Em geral, o tipo de dados de *resultado* é um **Byte**, uma variante de **Byte**, um **Integer**, uma variante de **Integer**, um **Long** ou uma variante de **Long**, independente de *resultado* ser um número inteiro. Qualquer parte fracionária é truncada. No entanto, se qualquer expressão for **Null**, *resultado* será **Null**. Qualquer expressão que seja **Empty** será tratada como 0.

## Operador Mod

Utilizado para dividir dois números e retornar somente o resto.

### Sintaxe

*resultado* = *número1* **Mod** *número2*

A sintaxe do operador **Mod** possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>número1</i>	Obrigatório; qualquer <u>expressão numérica</u> .
<i>número2</i>	Obrigatório; qualquer expressão numérica.

### Comentários

O operador de módulo, ou resto, divide *número1* por *número2* (arredondando números de vírgula flutuante para inteiros) e retorna somente o resto como *resultado*. Por exemplo, na expressão a seguir, A (*resultado*) é igual a 5.

```
A = 19 Mod 6.7
```

Geralmente, o tipo de dados de *resultado* é um **Byte**, uma variante de **Byte**, um **Integer**, uma variante de **Integer**, um **Long** ou uma **Variant** contendo um **Long**, independente de *resultado* ser um número inteiro ou não. Qualquer parte fracionária é truncada. No entanto, se qualquer expressão for **Null**, *resultado* será **Null**. Qualquer expressão que seja **Empty** será tratada como 0.

## Operador &

Utilizado para forçar a concatenação de seqüências de caracteres de duas expressões.

### Sintaxe

*resultado* = *expressão1* & *expressão2*

A sintaxe do operador **&** possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> <b>String</b> ou <b>Variant</b> .
<i>expressão1</i>	Obrigatório; qualquer expressão.
<i>expressão2</i>	Obrigatório; qualquer expressão.

### Comentários

Se uma expressão não for uma seqüência de caracteres, ela será convertida para uma variante de **String**. O tipo de dados de *resultado* é **String** se ambas as expressões forem expressões de seqüência de caracteres; caso contrário, *resultado* será uma variante de **String**. Se ambas as expressões forem **Null**, *resultado* será **Null**. No entanto, se somente uma expressão for **Null**, essa expressão será tratada como uma seqüência de caracteres de comprimento zero (""), quando concatenada com outra expressão. Qualquer expressão que seja **Empty** é tratada também como uma seqüência de caracteres de comprimento zero.

## Operadores de comparação

Utilizados para comparar expressões.

### Sintaxe

*resultado* = *expressão1* *operadordecomparação* *expressão2*

*resultado* = *objeto1* **Is** *objeto2*

*resultado* = *seqüenciadecaracteres* **Like** *padrão*

Os operadores de comparação possuem as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>expressão</i>	Obrigatório; qualquer expressão.
<i>operadordecomparação</i>	Obrigatório; qualquer operador de comparação.
<i>objeto</i>	Obrigatório; qualquer nome de objeto.
<i>seqüenciadecaracteres</i>	Obrigatório; qualquer <u>expressão de seqüência de caracteres</u> .
<i>padrão</i>	Obrigatório; qualquer expressão de seqüência de caracteres ou intervalo de caracteres.

### Comentários

A tabela a seguir contém uma lista dos operadores de comparação e as condições que determinam se *resultado* é **True**, **False** ou **Null**:

Operador	True se	False se	Null se
< (Menor que)	<i>expressão1</i> < <i>expressão2</i>	<i>expressão1</i> >= <i>expressão2</i>	<i>expressão1</i> ou <i>expressão2</i> = <b>Null</b>
<= (Menor que ou igual a)	<i>expressão1</i> <= <i>expressão2</i>	<i>expressão1</i> > <i>expressão2</i>	<i>expressão1</i> ou <i>expressão2</i> = <b>Null</b>
> (Maior que)	<i>expressão1</i> > <i>expressão2</i>	<i>expressão1</i> <= <i>expressão2</i>	<i>expressão1</i> ou <i>expressão2</i> = <b>Null</b>
>= (Maior que ou igual a)	<i>expressão1</i> >= <i>expressão2</i>	<i>expressão1</i> < <i>expressão2</i>	<i>expressão1</i> ou <i>expressão2</i> = <b>Null</b>
= (Igual a)	<i>expressão1</i> = <i>expressão2</i>	<i>expressão1</i> <> <i>expressão2</i>	<i>expressão1</i> ou <i>expressão2</i> = <b>Null</b>
<> (Diferente de)	<i>expressão1</i> <> <i>expressão2</i>	<i>expressão1</i> = <i>expressão2</i>	<i>expressão1</i> ou <i>expressão2</i> = <b>Null</b>

**Observação** Os operadores **Is** e **Like** apresentam uma funcionalidade de comparação específica que os diferencia dos operadores dessa tabela.

Ao comparar duas expressões, você pode não ser capaz de determinar com facilidade se as expressões estão sendo comparadas como números ou como seqüências de caracteres. A tabela a seguir mostra como as expressões são comparadas ou o resultado quando uma das expressões não é um **Variante**:

Se	Então
Ambas as expressões são <u>tipos de dados numéricos</u> ( <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>Currency</b> ou <b>Decimal</b> )	Efetuem uma comparação numérica.
Ambas as expressões são <b>String</b>	Efetuem uma <u>comparação de seqüências de caracteres</u> .
Uma expressão é um tipo de dados numérico e a outra é um <b>Variante</b> que é, ou pode ser, um número	Efetuem uma comparação numérica.
Uma expressão é um tipo de dados	Um erro <code>Type Mismatch</code> ocorre.

numérico e a outra é um **Variant** de seqüência de caracteres que não pode ser convertido para um número

Uma expressão é um **String** e a outra é qualquer **Variant** exceto **Null**

Uma expressão é **Empty** e a outra é um tipo de dados numérico

Uma expressão é **Empty** e a outra é um **String**

Efetuem uma comparação de seqüências de caracteres.

Efetuem uma comparação numérica, utilizando 0 como a expressão **Empty**.

Efetuem uma comparação de seqüências de caracteres, utilizando uma seqüência de caracteres de comprimento zero ("") como a expressão **Empty**.

Se tanto *expressão1* quanto *expressão2* são expressões **Variant**, seus tipos básicos determinam como elas serão comparadas. A tabela a seguir mostra como as expressões são comparadas ou o resultado da comparação, dependendo do tipo básico da **Variant**:

<b>Se</b>	<b>Então</b>
Ambas as expressões <b>Variant</b> são numéricas	Efetuem uma comparação numérica.
Ambas as expressões <b>Variant</b> são seqüências de caracteres	Efetuem uma comparação de seqüências de caracteres.
Uma expressão <b>Variant</b> é numérica e a outra é uma seqüência de caracteres	A expressão numérica é menor que a expressão de seqüência de caracteres.
Uma expressão <b>Variant</b> é <b>Empty</b> e a outra é numérica	Efetuem uma comparação numérica, utilizando 0 como a expressão <b>Empty</b> .
Uma expressão <b>Variant</b> é <b>Empty</b> e a outra é uma seqüência de caracteres	Efetuem uma comparação de seqüência de caracteres, utilizando uma seqüência de caracteres de comprimento zero ("") como a expressão <b>Empty</b> .
Ambas as expressões <b>Variant</b> são <b>Empty</b>	As expressões são iguais.

Quando um **Single** é comparado a um **Double**, o **Double** é arredondado à precisão de **Single**.

Se um **Currency** é comparado com um **Single** ou **Double**, **Single** ou **Double** é convertido para um **Currency**. Da mesma forma, quando um **Decimal** é comparado com um **Single** ou **Double**, o **Single** ou **Double** é convertido para um **Decimal**. Para **Currency**, qualquer valor fracionário menor que 0,0001 pode ser perdido; para **Decimal**, qualquer valor fracionário menor que 1E-28 pode ser perdido ou um erro de sobrecarga pode ocorrer. Essa perda de valores fracionais pode fazer com que dois valores sejam considerados iguais quando na verdade não o são.

## Operador And

Utilizado para efetuar uma conjunção lógica em duas expressões.

### Sintaxe

*resultado* = *expressão1* **And** *expressão2*

A sintaxe do operador **And** possui as partes a seguir:

<b>Parte</b>	<b>Descrição</b>
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>expressão1</i>	Obrigatório; qualquer expressão.

*expressão2* Obrigatório; qualquer expressão.

### Comentários

Se ambas as expressões forem avaliadas como **True**, *resultado* será **True**. Se uma das expressões for avaliada como **False**, *resultado* será **False**. A tabela a seguir mostra como *resultado* é determinado:

<b>Se expressão1 for</b>	<b>E expressão2 for</b>	<b>O resultado será</b>
True	True	True
True	False	False
True	<u>Null</u>	Null
False	True	False
False	False	False
False	Null	False
Null	True	Null
Null	False	False
Null	Null	Null

O operador **And** também efetua uma comparação bit-a-bit dos bits posicionados da mesma forma em duas expressões numéricas e define o bit correspondente em *resultado* de acordo com a tabela a seguir:

<b>Se o bit na expressão1 for</b>	<b>E o bit na expressão2 for</b>	<b>O resultado será</b>
0	0	0
0	1	0
1	0	0
1	1	1

## Operador Eqv

Utilizado para efetuar uma equivalência lógica em duas expressões.

### Sintaxe

*resultado* = *expressão1* **Eqv** *expressão2*

A sintaxe do operador **Eqv** possui as partes a seguir:

<b>Parte</b>	<b>Descrição</b>
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica
<i>expressão1</i>	Obrigatório; qualquer expressão.
<i>expressão2</i>	Obrigatório; qualquer expressão.

### Comentários

Se uma das expressões for Null, *resultado* também será **Null**. Se nenhuma das expressões for **Null**, *resultado* será determinado de acordo com a tabela a seguir:

<b>Se expressão1 for</b>	<b>E expressão2 for</b>	<b>O resultado será</b>
True	True	True
True	False	False
False	True	False
False	False	True

O operador **Eqv** executa uma comparação bit-a-bit dos bits posicionados da mesma forma em duas expressões numéricas e define o bit correspondente em *resultado* de acordo com a tabela a seguir:

<b>Se o bit na expressão1 for</b>	<b>E o bit na expressão2 for</b>	<b>O resultado será</b>
-----------------------------------	----------------------------------	-------------------------

---

0	0	1
0	1	0
1	0	0
1	1	1

## Operador Imp

Utilizado para efetuar uma implicação lógica em duas expressões.

### Sintaxe

*resultado* = *expressão1* **Imp** *expressão2*

A sintaxe do operador **Imp** possui as partes a seguir:

<b>Parte</b>	<b>Descrição</b>
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica
<i>expressão1</i>	Obrigatório; qualquer expressão.
<i>expressão2</i>	Obrigatório; qualquer expressão.

### Comentários

A tabela a seguir mostra como *resultado* é determinado:

<b>Se expressão1 for</b>	<b>E expressão2 for</b>	<b>O resultado será</b>
<b>True</b>	<b>True</b>	<b>True</b>
<b>True</b>	<b>False</b>	<b>False</b>
<b>True</b>	<b>Null</b>	<b>Null</b>
<b>False</b>	<b>True</b>	<b>True</b>
<b>False</b>	<b>False</b>	<b>True</b>
<b>False</b>	<b>Null</b>	<b>True</b>
<b>Null</b>	<b>True</b>	<b>True</b>
<b>Null</b>	<b>False</b>	<b>Null</b>
<b>Null</b>	<b>Null</b>	<b>Null</b>

O operador **Imp** efetua uma comparação bit-a-bit dos bits posicionados da mesma forma em duas expressões numéricas e define o bit correspondente em *resultado* de acordo com a tabela a seguir:

<b>Se o bit na expressão1 for</b>	<b>E o bit na expressão2 for</b>	<b>O resultado será</b>
0	0	1
0	1	1
1	0	0
1	1	1

## Operador Not

Utilizado para efetuar uma negação lógica em uma expressão.

### Sintaxe

*resultado* = **Not** *expressão*

A sintaxe do operador **Not** possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>expressão1</i>	Obrigatório; qualquer expressão.

### Comentários

A tabela a seguir mostra como *resultado* é determinado:

Se <i>expressão</i> for	Então <i>resultado</i> será
<b>True</b>	<b>False</b>
<b>False</b>	<b>True</b>
<b>Null</b>	<b>Null</b>

Além disso, o operador **Not** inverte os valores de bit de qualquer variável e define o bit correspondente em *resultado* de acordo com a tabela a seguir:

Se o bit na <i>expressão</i> for	Então o bit em <i>resultado</i> será
0	1
1	0

## Operador Or

Utilizado para efetuar uma disjunção lógica em duas expressões.

### Sintaxe

*resultado* = *expressão1* **Or** *expressão2*

A sintaxe do operador **Or** possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>expressão1</i>	Obrigatório; qualquer expressão.
<i>expressão2</i>	Obrigatório; qualquer expressão.

### Comentários

Se uma ou ambas as expressões forem **True**, *resultado* será **True**. A tabela a seguir mostra como *resultado* é determinado:

Se <i>expressão1</i> for	E <i>expressão2</i> for	Então <i>resultado</i> será
<b>True</b>	<b>True</b>	<b>True</b>
<b>True</b>	<b>False</b>	<b>True</b>
<b>True</b>	<b>Null</b>	<b>True</b>
<b>False</b>	<b>True</b>	<b>True</b>
<b>False</b>	<b>False</b>	<b>False</b>
<b>False</b>	<b>Null</b>	<b>Null</b>
<b>Null</b>	<b>True</b>	<b>True</b>
<b>Null</b>	<b>False</b>	<b>Null</b>



## Operador Is

Utilizado para comparar duas variáveis de referência de objeto.

### Sintaxe

*resultado* = *objeto1* **Is** *objeto2*

A sintaxe do operador **Is** possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer variável numérica.
<i>objeto1</i>	Obrigatório; qualquer nome de objeto.
<i>objeto2</i>	Obrigatório; qualquer nome de objeto.

### Comentários

Se tanto *objeto1* quanto *objeto2* referem-se ao mesmo objeto, *resultado* será **True**; caso contrário, *resultado* será **False**. Duas variáveis podem se referir ao mesmo objeto de diversas formas.

No exemplo a seguir, A foi definido de forma a se referir ao mesmo objeto que B:

```
Set A = B
```

O exemplo a seguir faz com que A e B se refiram ao mesmo objeto que C:

```
Set A = C
Set B = C
```

## Operador Like

Utilizado para comparar duas seqüências de caracteres.

### Sintaxe

*resultado* = *seqüenciadecaracteres* **Like** *padrão*

A sintaxe do operador **Like** possui as partes a seguir:

Parte	Descrição
<i>resultado</i>	Obrigatório; qualquer <u>variável</u> numérica.
<i>seqüenciadecaracteres</i>	Obrigatório; qualquer <u>expressão de seqüência de caracteres</u> .
<i>padrão</i>	Obrigatório; qualquer expressão de seqüência de caracteres que esteja de acordo com as convenções de correspondência de padrão descritas em "Comentários".

### Comentários

Se *seqüenciadecaracteres* corresponder a *padrão*, *resultado* será **True**; caso contrário, *resultado* será **False**. Se *seqüenciadecaracteres* ou *padrão* for **Null**, *resultado* será **Null**.

O comportamento do operador **Like** depende da instrução **Option Compare**. O método padrão de comparação de seqüências de caracteres para cada módulo é **Option Compare Binary**.

**Option Compare Binary** resulta em comparações de seqüências de caracteres baseadas em uma ordem de classificação derivada das representações binárias internas dos caracteres. No Microsoft Windows, a ordem de classificação é determinada pela página de código. No exemplo a seguir, uma ordem de classificação binária típica é apresentada:

```
A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø
```

**Option Compare Text** resulta em comparações de seqüências de caracteres baseadas em uma ordem de classificação textual, sem coincidir maiúsculas ou minúsculas, determinada pela localidade de seu sistema. Quando você classifica os mesmos caracteres utilizando **Option Compare Text**, o

seguinte texto de ordem de classificação é produzido:

(A=a) < (Â=â) < (B=b) < (E=e) < (Ê=ê) < (Z=z) < (Ø=ø)

A correspondência de padrão interna representa uma ferramenta versátil para comparações de seqüências de caracteres. Os recursos de correspondência de padrão permitem que você utilize caracteres curinga, listas de caracteres ou intervalos de caracteres, em qualquer combinação, para fazer a correspondência de seqüências de caracteres. A tabela a seguir mostra os caracteres permitidos em *padrão* e a o que eles correspondem:

Caracteres em <i>padrão</i>	Correspondem na <i>seqüênciadecaracteres a</i>
?	Qualquer caractere único.
*	Zero ou mais caracteres.
#	Qualquer dígito único (0–9).
[ <i>listadecarac</i> ]	Qualquer caractere único em <i>listadecarac</i> .
[! <i>listadecarac</i> ]	Qualquer caractere único que não esteja em <i>listadecarac</i> .

Um grupo de um ou mais caracteres (*listadecarac*) entre colchetes ([ ]) pode ser utilizado para corresponder a qualquer caractere único em *seqüênciadecaracteres* e pode incluir quase qualquer código de caractere, incluindo dígitos.

**Observação** Para fazer a correspondência dos caracteres especiais colchete esquerdo ([), ponto de interrogação (?), número (#) e asterisco (\*), coloque-os entre colchetes. O colchete direito (]) não pode ser utilizado dentro de um grupo para corresponder a si próprio, mas pode ser utilizado fora de um grupo como um caractere individual.

Ao utilizar um hífen (–) para separar os limites superior e inferior do intervalo, *listadecarac* pode especificar um intervalo de caracteres. Por exemplo, [A–Z] resulta em uma correspondência se a posição do caractere correspondente em *seqüênciadecaracteres* contiver qualquer letra em caixa alta no intervalo de A–Z. Intervalos múltiplos são incluídos entre os colchetes sem delimitadores.

O significado de um intervalo especificado depende da ordem dos caracteres válida em tempo de execução (como determinado por **Option Compare** e pela definição da localidade do sistema no qual o código está sendo executado). Utilizando o exemplo de **Option Compare Binary**, o intervalo de [A–E] corresponde a A, B e E. Com **Option Compare Text**, [A–E] corresponde a A, a, Â, à, B, b, E, e. O intervalo não corresponde a Ê ou ê porque caracteres acentuados situam-se após caracteres não acentuados na ordem de classificação.

Outras regras importantes para a correspondência de padrão incluem:

- Um ponto de exclamação (!) no início de *listadecarac* significa que uma correspondência é realizada se qualquer caractere, com exceção daqueles em *listadecarac*, for encontrado em *seqüênciadecaracteres*. Quando utilizado fora de colchetes, o ponto de exclamação corresponde a si mesmo.
- Um hífen (–) pode aparecer tanto no início (após um ponto de exclamação, se um for utilizado) quanto no final de *listadecarac* para corresponder a si mesmo. Em qualquer outra localização, o hífen é utilizado para identificar um intervalo de caracteres.
- Quando um intervalo de caracteres é especificado, estes devem aparecer em uma ordem de classificação crescente (do menor para o maior). [A–Z] é um padrão válido, mas [Z–A] não o é.
- A seqüência de caracteres [] é considerada uma seqüência de caracteres de comprimento zero ("").

Em alguns idiomas, existem caracteres especiais no alfabeto que representam dois caracteres diferentes. Por exemplo, diversos idiomas utilizam o caractere "æ" para representar os caracteres "a" e "e" quando eles aparecem juntos. O operador **Like** reconhece a equivalência entre o caractere especial único e os dois caracteres individuais.

Quando um idioma que utiliza um caractere especial é especificado nas definições de localidade do sistema, uma ocorrência do caractere especial único em *padrão* ou *seqüênciadecaracteres* corresponde à seqüência de dois caracteres equivalente na outra seqüência de caracteres. Da mesma forma, um caractere especial único em *padrão* entre colchetes (sozinho, em uma lista ou em um intervalo) corresponde à seqüência de dois caracteres equivalente em *seqüênciadecaracteres*.

### Exemplo da propriedade Count

Este exemplo utiliza a propriedade **Count** do objeto **Collection** para especificar quantas iterações são requeridas para remover todos os elementos da coleção `MinhasClasses`. Quando as coleções são indexadas numericamente, a base é 1 como padrão. Como as coleções são reindexadas automaticamente quando é feita uma remoção, o código a seguir remove o primeiro membro em cada iteração.

```
Dim Num, MinhasClasses
For Num = 1 To MinhasClasses.Count ' Remove o nome da coleção.
    MinhasClasses.Remove 1 ' Os índices numéricos da coleção padrão
Next ' começam em 1.
```

### Exemplo da propriedade Description

Este exemplo atribui uma mensagem definida pelo usuário à propriedade **Description** do objeto **Err**.

```
Err.Description = "Não foi possível acessar um objeto necessário " _
& "para esta operação".
```

### Exemplo da propriedade HelpContext

Este exemplo utiliza a propriedade **HelpContext** do objeto **Err** para mostrar o tópico da Ajuda do Visual Basic relativo ao erro `Sobrecarga`.

```
Dim Msg
Err.Clear
On Error Resume Next
Err.Raise 6 ' Gera o erro "Sobrecarga".
If Err.Number <> 0 Then
    Msg = "Pressione F1 ou Ajuda para ver " & Err.HelpFile & " o tópico
relativo ao" & _
        " seguinte ContextoDaAjuda: " & Err.HelpContext
    MsgBox Msg, , "Erro: " & Err.Description, Err.HelpFile, _
Err.HelpContext
End If
```

### Exemplo da propriedade HelpFile

Este exemplo utiliza a propriedade **HelpFile** do objeto **Err** para iniciar o sistema de Ajuda do Microsoft Windows. Como padrão, a propriedade **HelpFile** contém o nome do arquivo de Ajuda do Visual Basic.

```
Dim Msg
Err.Clear
On Error Resume Next ' Suprime os erros por finalidade demonstrativa.
Err.Raise 6 ' Gera o erro "Sobrecarga".
Msg = "Pressione F1 ou Ajuda para ver " & Err.HelpFile & _
" o tópico relativo a este erro"
MsgBox Msg, , "Erro: " & Err.Description, Err.HelpFile, Err.HelpContext
```

**Exemplo da propriedade LastDLLError**

Quando colado em um módulo **UserForm**, o código a seguir ocasiona a tentativa de chamar uma função DLL. A chamada falha porque o argumento que é passado (um ponteiro nulo) gera um erro e, em qualquer evento, a SQL não pode ser cancelada se não estiver sendo executada. O código seguinte à chamada verifica o retorno da chamada e, em seguida, é impresso na propriedade **LastDLLError** do objeto **Err** para revelar o código do erro.

```
Private Declare Function SQLCancel Lib "ODBC32.dll" _
    (ByVal hstmt As Long) As Integer

Private Sub UserForm_Click()
    Dim ValRet
    ' Chama com argumento inválido.
    ValRet = SQLCancel(myhandle&)
    ' Procura pelo código de erro SQL.
    If ValRet = -2 Then
        'Exibe o código de informações.
        MsgBox "O código do erro é :" & Err.LastDllError
    End If
End Sub
```

**Exemplo da propriedade Number**

O primeiro exemplo ilustra um uso típico da propriedade **Number** em uma rotina de tratamento de erros. O segundo exemplo examina a propriedade **Number** do objeto **Err** para determinar se um erro retornado por um objeto de Automação foi definido pelo objeto ou se foi mapeado para um erro definido pelo Visual Basic. Observe que a constante **vbObjectError** é um número negativo muito grande que um objeto adiciona ao seu próprio código de erro para indicar que o erro é definido pelo servidor. Portanto, subtraí-lo de **Err.Number** o exclui do resultado. Se o erro for definido pelo objeto, o número base será deixado em **MeuErro**, que é exibido em uma caixa de mensagem junto com a fonte original do erro. Se **Err.Number** representa um erro do Visual Basic, então o número do erro do Visual Basic é exibido na caixa de mensagem.

```
' Uso típico da propriedade Number
Sub test()
    On Error GoTo out

    Dim x, y
    x = 1 / y ' Cria o erro de divisão por zero
    Exit Sub
out:
    MsgBox Err.Number
    MsgBox Err.Description
    ' Procura pelo erro de divisão por zero
    If Err.Number = 11 Then
        y = y + 1
    End If
    Resume
End Sub
```

```
' Utilizando a propriedade Number com um erro de um
' objeto de Automação
Dim MeuErro, Msg
' Primeiro exclua a constante adicionada pelo objeto para indicar um dos
' seus próprios erros.
MeuErro = Err.Number - vbObjectError
' Se você subtrair a constante vbObjectError e o número continuar
```

```
' no intervalo 0-65,535, trata-se de um código de erro definido pelo
objeto.
If MeuErro > 0 And MeuErro < 65535 Then
    Msg = "O objeto que você acessou atribuiu este número ao erro: " _
        & MeuErro & ". Quem deu origem ao erro foi: " _
        & Err.Source & ". Pressione F1 para ver o tópico da Ajuda que deu
origem".
' Caso contrário, é o número de erro do Visual Basic.
Else
    Msg = "Este erro (# " & Err.Number & ") é um número de erro do Visual
Basic" & _
        " Pressione o botão Ajuda ou F1 para obter o" _
        & " tópico da Ajuda do Visual Basic sobre este erro.
End If
MsgBox Msg, , "Erro de objeto", Err.HelpFile, Err.HelpContext
```

### Exemplo da propriedade Source

Este exemplo atribui a identificação Programática de um objeto de Automação criado no Visual Basic à variável `MeuCódigoDeObjeto` e, em seguida, o atribui à propriedade **Source** do objeto **Err** quando gera um erro com o método **Raise**. Quando estiver tratando de erros, você não deve usar a propriedade **Source** (nem nenhuma propriedade **Err** diferente de **Number**) programaticamente. O único uso válido de propriedades diferentes de **Number** é a exibição de informações ricas para um usuário final nos casos em que você não conseguir tratar um erro. O exemplo assume que `Aplicativo` e `MinhaClasse` são referências válidas.

```
Dim MinhaClasse, MeuCódigoDeObjeto, MeuArquivoDeAjuda, MeuContextoDeAjuda
' Um objeto do tipo MinhaClasse gera um erro e preenche todas as
propriedades do objeto Err,
' incluindo Source, que recebe MeuCódigoDeObjeto, que é uma
' combinação da propriedade Title do objeto App e da
' propriedade do objeto MinhaClasse.
MeuCódigoDeObjeto = App.Title & "." & MinhaClasse.Name
Err.RaiseNumber := vbObjectError + 894, Source := MeuCódigoDeObjeto, _
    Description := "Não foi possível completar a sua tarefa", _
    HelpFile := MeuArquivoDeAjuda, HelpContext :=
MeuContextoDeAjuda
```

### Propriedade Calendar

Retorna ou configura um valor especificando o tipo de calendário a ser usado com seu projeto.

Você pode usar uma das duas configurações abaixo para **Calendar**:

Configuração	Valor	Descrição
<b>vbCalGreg</b>	0	Use o calendário Gregoriano (padrão).
<b>vbCalHijri</b>	1	Use o calendário Hijri.

### Observações

Você somente pode configurar a propriedade **Calendar** através de programa. Por exemplo, para usar o calendário Hijri, use:

```
Calendar = vbCalHijri
```

## Propriedade Count

Retorna um **Long** (número inteiro longo) que contém o número de objetos de uma coleção. Somente leitura.

## Propriedade Description

Retorna ou define uma expressão de seqüência de caracteres que contém uma seqüência de caracteres descritiva associada a um objeto. Leitura/gravação.

Para o objeto **Err**, retorna ou define uma seqüência de caracteres descritiva associada a um erro.

### Comentários

A definição da propriedade **Description** consiste em uma breve descrição do erro. Utilize essa propriedade para alertar o usuário sobre um erro que você não pode ou não deseja manipular. Quando gerar um erro definido pelo usuário, atribua uma breve descrição do erro à propriedade **Description**. Se **Description** não estiver preenchida e o valor de **Number** corresponder a um erro em tempo de execução do Visual Basic, a seqüência de caracteres retornada pela função **Error** será localizada em **Description** quando o erro for gerado.

## Propriedade HelpContext

Retorna ou define uma expressão de seqüência de caracteres que contém a identificação de contexto de um tópico em um arquivo de Ajuda do Microsoft Windows. Leitura/gravação.

### Comentários

A propriedade HelpContext é utilizada para exibir automaticamente o tópico de Ajuda especificado na propriedade **HelpFile**. Se tanto **HelpFile** quanto **HelpContext** estiverem vazios, o valor de **Number** será selecionado. Se **Number** corresponder a um valor de erro em tempo de execução do Visual Basic, a identificação de contexto da Ajuda do Visual Basic para o erro será utilizada. Se o valor de **Number** não corresponder a um erro do Visual Basic, a tela de conteúdo do arquivo de Ajuda do Visual Basic será exibida.

**Observação** Você deve colocar rotinas de manipulação de erros típicos no seu aplicativo. Ao programar com um objeto, você pode utilizar o arquivo de Ajuda do objeto para melhorar a qualidade da manipulação de erros ou para exibir uma mensagem significativa para seu usuário se o erro não for recuperável.

## Propriedade HelpFile

Retorna ou define em uma expressão de seqüência de caracteres o caminho completo para um arquivo de Ajuda do Microsoft Windows. Leitura/gravação.

### Comentários

Se um arquivo de Ajuda for especificado em **HelpFile**, ele será automaticamente chamado quando o usuário pressionar o botão **Help** (ou a tecla F1) na caixa de diálogo de mensagem de erro. Se a propriedade **HelpContext** contiver uma identificação de contexto válida para o arquivo especificado, aquele tópico será automaticamente exibido. Se nenhum **HelpFile** for especificado, o arquivo de Ajuda do Visual Basic será exibido.

**Observação** Você deve colocar rotinas de manipulação de erros típicos no seu aplicativo. Ao programar com um objeto, você pode utilizar o arquivo de Ajuda do objeto para melhorar a qualidade da manipulação de erros ou para exibir uma mensagem significativa para seu usuário se o erro não for recuperável.

## Propriedade LastDLLError

Retorna um código de erro de sistema produzido por uma chamada a uma biblioteca de vínculo dinâmico (DLL). Somente-leitura.

### Comentários

A propriedade LastDLLError aplica-se somente a chamadas a DLL feitas a partir de código do Visual Basic. Quando tal chamada é feita, a função chamada retorna um código indicando êxito ou falha, e a propriedade **LastDLLError** é preenchida. Verifique a documentação das funções da DLL para determinar os valores de retorno que indicam êxito ou falha. Sempre que o código de falha é retornado, o aplicativo do Visual Basic deve verificar imediatamente a propriedade **LastDLLError**. Nenhuma exceção é levantada quando a propriedade **LastDLLError** está definida.

## Propriedade Number

Retorna ou define um valor numérico que especifica um erro. **Number** é a propriedade padrão do objeto **Err**. Leitura/gravação.

### Comentários

Quando retornar um erro definido pelo usuário de um objeto, defina **Err.Number** adicionando o número que você selecionou como código de erro à constante **vbObjectError**. Por exemplo, para retornar o número 1051 como um código de erro você usaria o código a seguir:

```
Err.Raise Number := vbObjectError + 1051, Source:= "AlgumaClasse"
```

## Propriedade Source

Retorna ou define uma expressão de seqüência de caracteres que especifica o nome do objeto ou aplicativo que originalmente gerou o erro. Leitura/gravação.

### Comentários

A propriedade Source especifica uma expressão de seqüência de caracteres que representa o objeto que gerou o erro; a expressão é normalmente o nome de classe do objeto ou uma identificação programática. Utilize **Source** para fornecer informações quando seu código não for capaz de manipular um erro gerado em um objeto acessado. Por exemplo, se você acessar o Microsoft Excel e ele gerar um erro *Division by zero*, o Microsoft Excel define **Err.Number** para seu código de erro correspondente e define **Source** como *Excel.Application*.

Quando gerar um erro a partir do código, **Source** será a identificação programática do seu aplicativo. Para módulos de classe, **Source** deve conter um nome com a forma *projeto.classe*. Quando um erro inesperado ocorre em seu código, a propriedade **Source** é automaticamente preenchida. Para erros em um módulo padrão, **Source** contém o nome do projeto. Para erros em um módulo de classe, **Source** contém um nome com a forma *projeto.classe*.

### Exemplo da instrução DeleteSetting

O exemplo a seguir primeiro utiliza a instrução **SaveSetting** para criar entradas no Registro do Windows (ou no arquivo .ini nas plataformas Windows de 16 bits) para o aplicativo `MeuAplicativo` e, em seguida, utiliza a instrução **DeleteSetting** para removê-las. Como não há argumento **key** especificado, toda a seção é excluída, incluindo o nome da seção e todas as suas chaves.

```
' Insere algumas definições no Registro.
SaveSetting appname := "MeuAplicativo", section := "Inicialização", _
    key := "Superior", setting := 75
SaveSetting "MeuAplicativo","Inicialização", "Esquerda", 50
' Remove a seção e todas as suas definições do Registro.
DeleteSetting "MeuAplicativo", "Inicialização"
```

### Exemplo da função GetAllSettings

Este exemplo primeiro utiliza a instrução **SaveSetting** para criar entradas no Registro do Windows (ou no arquivo .ini nas plataformas Windows de 16 bits) para o aplicativo especificado como **appname** e, em seguida, utiliza a função **GetAllSettings** para exibir as configurações. Observe que os nomes de aplicativo e os nomes de **section** não podem ser recuperados com **GetAllSettings**. Por fim, a instrução **DeleteSetting** remove as entradas do aplicativo.

```
' Variant para conter uma matriz bidimensional retornada por GetAllSettings
' Integer para conter o contador.
Dim MinhasDefinições As Variant, DefiniçõesInt As Integer
' Insere algumas definições no Registro.
SaveSetting appname := "MeuAplicativo", section := "Inicialização", _
key := "Superior", setting := 75
SaveSetting "MeuAplicativo","Inicialização", "Esquerda", 50
' Recupera as definições.
MinhasDefinições = GetAllSettings(appname := "MeuAplicativo", section :=
"Inicialização")
For DefiniçõesInt = LBound(MinhasDefinições, 1) To
UBound(MinhasDefinições, 1)
    Debug.Print MinhasDefinições(DefiniçõesInt, 0),
MinhasDefinições(DefiniçõesInt, 1)
Next DefiniçõesInt
DeleteSetting "MeuAplicativo", "Inicialização"
```

### Exemplo da função GetSetting

Este exemplo primeiro utiliza a instrução **SaveSetting** para criar entradas no Registro do Windows (ou no arquivo .ini nas plataformas Windows de 16 bits) para o aplicativo especificado como **appname** e, em seguida, utiliza a função **GetSetting** para exibir uma das definições. Como o argumento **default** é especificado, algum valor certamente será retornado. Observe que os nomes de **section** não podem ser recuperados com **GetSetting**. Por fim, a instrução **DeleteSetting** remove todas as entradas do aplicativo.

```
' Variant para conter uma matriz bidimensional retornada por GetSetting.
Dim MinhasDefinições As Variant
' Insira algumas definições no Registro.
SaveSetting "MeuAplicativo","Inicialização", "Superior", 75
SaveSetting "MeuAplicativo","Inicialização", "Esquerda", 50

Debug.Print GetSetting(appname := "MeuAplicativo", section :=
"Inicialização", _
    key := "Esquerda", default := "25")

DeleteSetting "MeuAplicativo", "Inicialização"
```

### Exemplo da instrução SaveSetting

O exemplo a seguir primeiro utiliza a instrução **SaveSetting** para criar entradas no Registro do Windows (ou no arquivo .ini nas plataformas Windows de 16 bits) para o aplicativo `MeuAplicativo` e, em seguida, utiliza a instrução **DeleteSetting** para removê-las.

```
' Insere algumas definições no Registro.
SaveSetting appname := "MeuAplicativo", section := "Inicialização", _
    key := "Superior", setting := 75
SaveSetting "MeuAplicativo","Inicialização", "Esquerda", 50
' Remove a seção e todas as suas definições do Registro.
DeleteSetting "MeuAplicativo", "Inicialização"
```

### Instrução DeleteSetting

Exclui uma seção ou definição de chave existente na entrada de um aplicativo no Registro do Windows.

#### Sintaxe

**DeleteSetting** *appname*, *section*[,*key*]

A sintaxe da instrução **DeleteSetting** possui estes argumentos nomeados:

Parte	Descrição
<b>appname</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que contém o nome do aplicativo ou <u>projeto</u> ao qual se aplica a seção ou a definição de chave.
<b>section</b>	Obrigatório. Expressão de seqüência de caracteres que contém o nome da seção da qual a definição da chave está sendo excluída. Se forem fornecidos somente <b>appname</b> e <b>section</b> , a seção especificada será excluída juntamente com todas as definições de chave relacionadas.
<b>key</b>	Opcional. Expressão de seqüência de caracteres que contém o nome da definição da chave sendo excluída.

#### Comentários

Se todos os argumentos forem fornecidos, a definição da chave especificada será excluída. Entretanto, a instrução **DeleteSetting** não fará nada se a seção ou a definição de chave não existir.

### Função GetAllSettings

Retorna uma lista de definições de chave e seus respectivos valores (originariamente criados com **SaveSetting**) a partir da entrada de um aplicativo no Registro do Windows.

#### Sintaxe

**GetAllSettings**(*appname*, *section*)

A sintaxe da função **GetAllSettings** possui estes argumentos nomeados:

Parte	Descrição
<b>appname</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que contém o nome do aplicativo ou <u>projeto</u> cujas definições de chave são solicitadas.
<b>section</b>	Obrigatório. Expressão de seqüência de caracteres que contém o nome da seção cujas definições de chave são solicitadas. <b>GetAllSettings</b> retorna um <b>Variant</b> cujo conteúdo é uma <u>matriz</u> bidimensional de seqüências de caracteres

contendo todas as definições de chave da seção especificada e seus valores correspondentes.

#### Comentários

**GetAllSettings** retorna um **Variant** não inicializada se **appname** ou **section** não existir.

## Função GetSetting

Retorna um valor de definição da chave para uma entrada de aplicativo no Registro do Windows.

#### Sintaxe

**GetSetting**(*appname*, *section*, *key*[, *default*])

A sintaxe da função **GetSetting** possui estes argumentos nomeados:

Parte	Descrição
<b>appname</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que contém o nome do aplicativo ou projeto cuja definição de chave é solicitada.
<b>section</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que contém o nome da seção onde se encontra a definição de chave.
<b>key</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que contém o nome da definição de chave a ser retornada.
<b>default</b>	Opcional. <u>Expressão</u> que contém o valor a retornar se nenhum valor for definido na definição da chave. Se for omitido, <b>default</b> será assumido como uma seqüência de caracteres de tamanho nulo ("").

#### Comentários

Se qualquer dos itens nomeados nos argumentos de **GetSetting** não existirem, **GetSetting** retornará o valor do **default**.

## Instrução SaveSetting

Salva ou cria uma entrada de aplicativo no Registro do Windows.

#### Sintaxe

**SaveSetting** *appname*, *section*, *key*, *setting*

A sintaxe da instrução **SaveSetting** possui estes argumentos nomeados:

Parte	Descrição
<b>appname</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que contém o nome do aplicativo ou <u>projeto</u> ao qual se aplica a definição.
<b>section</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que contém o nome da seção em que a definição da chave está sendo salva.
<b>key</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que contém o nome da definição de chave sendo salva.
<b>setting</b>	Obrigatório. <u>Expressão</u> que contém o valor com que <b>key</b> está sendo definido.

#### Comentários

Ocorrerá um erro se, por qualquer motivo, a definição da chave não puder ser salva.

**Exemplo da função Chr**

Este exemplo utiliza a função **Chr** para retornar o caractere associado ao código de caractere especificado.

```
Dim MeuCar
MeuCar = Chr(65) ' Retorna A.
MeuCar = Chr(97) ' Retorna a.
MeuCar = Chr(62) ' Retorna >.
MeuCar = Chr(37) ' Retorna %.
```

**Exemplo da função Format**

Este exemplo mostra vários usos da função **Format** para formatar valores utilizando tanto formatos nomeados quanto formatos definidos pelo usuário. Para o separador de data (/), separador de hora (:) e literal AM/PM, a saída formatada real exibida pelo seu sistema depende das definições da localidade em que o código está sendo executado. Quando as horas e datas são exibidas no ambiente de desenvolvimento, são usados o formato de hora abreviada e o formato de data abreviada da localidade do código. Quando exibidas pela execução do código, são usados o formato de hora abreviada e o formato de data abreviada da localidade do sistema, que podem diferir da localidade do código. Para este exemplo, pressupõe-se Inglês/E.U.A.

MinhaHora e MinhaData são exibidas no ambiente de desenvolvimento usando-se a definição de hora abreviada e a definição de data abreviada.

```
Dim MinhaHora, MinhaData, MinhaSeq
MinhaHora = #17:04:23#
MinhaData = #Janeiro 27, 1993#

' Retorna a hora atual do sistema no formato de hora completa definido pelo
usuário.
MinhaSeq = Format(Time, "Long Time")

' Retorna a data atual do sistema no formato de data completa definido pelo
usuário.
MinhaSeq = Format(Date, "Long Date")

MinhaSeq = Format(MinhaHora, "h:m:s") ' Retorna "17:4:23".
MinhaSeq = Format(MinhaHora, "hh:mm:ss AM/PM") ' Retorna "05:04:23 PM".
MinhaSeq = Format(MinhaData, "dddd, mmm d yyyy") ' Retorna "Quarta-
feira,
' 27 jan 1993".
' Se o formato não for fornecido, é retornada uma seqüência de caracteres.
MinhaSeq = Format(23) ' Retorna "23".

' Formatos definidos pelo usuário.
MinhaSeq = Format(5459.4, "##,##0.00") ' Retorna "5,459.40".
MinhaSeq = Format(334.9, "###0.00") ' Retorna "334.90".
MinhaSeq = Format(5, "0.00%") ' Retorna "500.00%".
MinhaSeq = Format("ALÔ", "<") ' Retorna "alô".
MinhaSeq = Format("É isto", ">") ' Retorna "É ISTO".
```

**Exemplo da função Hex**

Este exemplo utiliza a função **Hex** para retornar o valor hexadecimal de um número.

```
Dim MeuHex
MeuHex = Hex(5) ' Retorna 5.
MeuHex = Hex(10) ' Retorna A.
MeuHex = Hex(459) ' Retorna 1CB.
```

**Exemplo da função InStr**

Este exemplo utiliza a função **InStr** para retornar a posição da primeira ocorrência de uma seqüência de caracteres dentro de outra.

```
Dim SeqüênciaDePesquisa, CarDePesquisa, MinhaPos
SeqüênciaDePesquisa = "XXpXXpXXpXXP" ' Seqüência de caracteres a pesquisar.
CarDePesquisa = "P" ' Procurar por "P".

' Uma comparação textual iniciando na posição 4. Retorna 6.
MinhaPos = InStr(4, SeqüênciaDePesquisa, CarDePesquisa, 1)

' Uma comparação de binários iniciando na posição 1. Retorna 9.
MinhaPos = InStr(1, SeqüênciaDePesquisa, CarDePesquisa, 0)

' A comparação é binária como padrão (o último argumento é omitido).
MinhaPos = InStr(SeqüênciaDePesquisa, CarDePesquisa) ' Retorna 9.

MinhaPos = InStr(1, SeqüênciaDePesquisa, "W") ' Retorna 0.
```

**Exemplo da função LCase**

Este exemplo utiliza a função **LCase** para retornar uma versão em minúsculas de uma seqüência de caracteres.

```
Dim Maiúsculas, Minúsculas
Maiúsculas = "Alô Mundo 1234" ' Seqüência de caracteres a converter.
Minúsculas = LCase(Maiúsculas) ' Retorna "alô mundo 1234".
```

**Exemplo da função Left**

Este exemplo utiliza a função **Left** para retornar um número especificado de caracteres do lado esquerdo de uma seqüência de caracteres.

```
Dim QualquerSeqüência, MinhaSeq
QualquerSeqüência = "Alô Mundo" ' Define a seqüência de caracteres.
MinhaSeq = Left(QualquerSeqüência, 1) ' Retorna "A".
MinhaSeq = Left(QualquerSeqüência, 5) ' Retorna "Alô M".
MinhaSeq = Left(QualquerSeqüência, 20) ' Retorna "Alô Mundo".
```

**Exemplo da função Len**

Este exemplo utiliza a função **Len** para retornar o número de caracteres de uma seqüência de caracteres ou o número de bytes necessários para armazenar uma variável. O bloco **Type...End Type** que define RegistroDoCliente deve ser precedido pela palavra-chave **Private** se aparecer em um módulo de classe. Em um módulo padrão, uma instrução **Type** pode ser **Public**.

```
Type RegistroDoCliente ' Define o tipo definido pelo usuário.
    ID As Integer ' Coloca esta definição em um
    Name As String * 10 ' módulo padrão.
    Address As String * 30
```

```
End Type
```

```
Dim Cliente As RegistroDoCliente ' Declara as variáveis.
Dim MeuInt As Integer, MeuCur As Currency
Dim MinhaSeqüência, MeuComp
MinhaSeqüência = "Alô Mundo" ' Inicializa a variável.
MeuComp = Len(MeuInt) ' Retorna 2.
MeuComp = Len(Cliente) ' Retorna 42.
MeuComp = Len(MinhaSeqüência) ' Retorna 9.
MeuComp = Len(MeuCur) ' Retorna 8.
```

### Exemplo da instrução LSet

Este exemplo utiliza a instrução **LSet** para alinhar à esquerda uma seqüência de caracteres dentro de uma variável de seqüência de caracteres. Ainda que **LSet** possa ser utilizada para copiar uma variável de um tipo definido pelo usuário em outra variável de um tipo definido pelo usuário diferente mas compatível, esta prática não é recomendada. Devido à variedade de implementações de estruturas de dados entre plataformas, não se pode garantir a portabilidade desse uso de **LSet**.

```
Dim MinhaSeqüência
MinhaSeqüência = "0123456789" ' Inicializa a seqüência de caracteres.
Lset MinhaSeqüência = "<-Left" ' MinhaSeqüência contém "<-Left".
```

### Exemplo das funções LTrim, RTrim e Trim

Este exemplo utiliza a função **LTrim** para retirar os espaços à esquerda e a função **RTrim** para retirar os espaços à direita de uma variável de seqüência de caracteres. Ele utiliza a função **Trim** para retirar ambos os tipos de espaços.

```
Dim MinhaSeqüência, SuprimirSeqüência
MinhaSeqüência = " <-Trim-> " ' Inicializa a seqüência de caracteres.
SuprimirSeqüência = LTrim(MinhaSeqüência) ' SuprimirSeqüência = "<-Trim->".
SuprimirSeqüência = RTrim(MinhaSeqüência) ' SuprimirSeqüência = " <-Trim->".
SuprimirSeqüência = LTrim(RTrim(MinhaSeqüência)) ' SuprimirSeqüência = "<-Trim->".
' O uso da função Trim sozinha implica o mesmo resultado.
SuprimirSeqüência = Trim(MinhaSeqüência) ' SuprimirSeqüência = "<-Trim->".
```

### Exemplo da função Mid

Este exemplo utiliza a função **Mid** para retornar um número especificado de caracteres de uma seqüência de caracteres.

```
Dim MinhaSeqüência, PrimeiraPalavra, ÚltimaPalavra, PalavrasDoMeio
MinhaSeqüência = "Demonstração da função Mid" ' Cria a seqüência de caracteres de texto.
PrimeiraPalavra = Mid(MinhaSeqüência, 1, 12) ' Retorna "Demonstração".
ÚltimaPalavra = Mid(MinhaSeqüência, 24, 3) ' Retorna "Mid".
PalavrasDoMeio = Mid(MinhaSeqüência, 14) ' Retorna "da função Mid".
```

**Exemplo da instrução Mid**

Este exemplo utiliza a instrução **Mid** para substituir um número especificado de caracteres de uma variável de seqüência de caracteres por caracteres de outra seqüência de caracteres.

```
Dim MinhaSeqüência
MinhaSeqüência = "O gato salta"      ' Inicializa a seqüência de caracteres.
Mid(MinhaSeqüência, 3, 4) = "sapo"   ' MinhaSeqüência = "O sapo salta".
Mid(MinhaSeqüência, 3) = "rato"      ' MinhaSeqüência = "O rato salta".
Mid(MinhaSeqüência, 3) = "rato saltou sobre" ' MinhaSeqüência = "O rato salto".
Mid(MinhaSeqüência, 3, 4) = "ganso"  ' MinhaSeqüência = "O gans salto".
```

**Exemplo da função Oct**

Este exemplo utiliza a função **Oct** para retornar o valor octal de um número.

```
Dim MeuOctal
MeuOctal = Oct(4) ' Retorna 4.
MeuOctal = Oct(8) ' Retorna 10.
MeuOctal = Oct(459) ' Retorna 713.
```

**Exemplo da função Right**

Este exemplo utiliza a função **Right** para retornar um número especificado de caracteres do lado direito de uma seqüência de caracteres.

```
Dim QualquerSeqüência, MinhaSeq
QualquerSeqüência = "Alô Mundo"      ' Define a seqüência de caracteres.
MinhaSeq = Right(QualquerSeqüência, 1) ' Retorna "o".
MinhaSeq = Right(QualquerSeqüência, 6) ' Retorna " Mundo".
MinhaSeq = Right(QualquerSeqüência, 20) ' Retorna "Alô Mundo".
```

**Exemplo da instrução RSet**

Este exemplo utiliza a instrução **RSet** para alinhar à direita uma seqüência de caracteres dentro de uma variável de seqüência de caracteres.

```
Dim MinhaSeqüência
MinhaSeqüência = "0123456789" ' Inicializa a seqüência de caracteres.
Rset MinhaSeqüência = "Right->" ' MinhaSeqüência contém " Right->".
```

**Exemplo da função Space**

Este exemplo utiliza a função **Space** para retornar uma seqüência de caracteres que consiste em um número especificado de espaços.

```
Dim MinhaSeqüência
' Retorna a seqüência de caracteres com 10 espaços.
MinhaSeqüência = Space(10)

' Insere 10 espaços entre duas seqüências de caracteres.
MinhaSeqüência = "Alô" & Space(10) & "Mundo"
```

**Exemplo da função Str**

Este exemplo utiliza a função **Str** para retornar uma representação em seqüência de caracteres de um número. Quando um número é convertido em uma seqüência de caracteres, um espaço à esquerda é sempre reservado para o seu sinal.

```
Dim MinhaSeqüência
```

```
MinhaSeqüência = Str(459) ' Retorna " 459".
MinhaSeqüência = Str(-459.65) ' Retorna "-459.65".
MinhaSeqüência = Str(459.001) ' Retorna " 459.001".
```

### Exemplo da função StrComp

Este exemplo utiliza a função **StrComp** para retornar os resultados de uma comparação de seqüências de caracteres. Se o terceiro argumento for 1, será executada uma comparação textual; se for 0 ou omitido, será executada uma comparação binária.

```
Dim MinhaSeq1, MinhaSeq2, MinhaComp
MinhaSeq1 = "ABCD": MinhaSeq2 = "abcd" ' Define as variáveis.
MinhaComp = StrComp(MinhaSeq1, MinhaSeq2, 1) ' Retorna 0.
MinhaComp = StrComp(MinhaSeq1, MinhaSeq2, 0) ' Retorna -1.
MinhaComp = StrComp(MinhaSeq2, MinhaSeq1) ' Retorna 1.
```

### Exemplo da função String

Este exemplo utiliza a função **String** para retornar as seqüências de caracteres repetidos com o comprimento especificado.

```
Dim MinhaSeqüência
MinhaSeqüência = String(5, "*") ' Retorna "*****".
MinhaSeqüência = String(5, 42) ' Retorna "*****".
MinhaSeqüência = String(10, "ABC") ' Retorna "AAAAAAAAAA".
```

### Exemplo da função UCase

Este exemplo utiliza a função **UCase** para retornar uma versão em maiúsculas de uma seqüência de caracteres.

```
Dim Minúsculas, Maiúsculas
Minúsculas = "Alô Mundo 1234" ' Seqüência de caracteres a converter.
Maiúsculas = UCase(Minúsculas) ' Retorna "ALÔ MUNDO 1234".
```

## Função Chr

Retorna um **String** que contém o caractere associado ao código de caractere especificado.

### Sintaxe

**Chr**(*códigodecarac*)

O argumento obrigatório *códigodecarac* é um **Long** que identifica um caractere.

### Comentários

Os números de 0 a 31 são os mesmos que os códigos padrão não-imprimíveis ASCII. Por exemplo, **Chr**(10) retorna um caractere de alimentação de linha. O intervalo normal para *códigodecarac* é 0 – 255. No entanto, em sistemas DBCS, o intervalo real para *códigodecarac* é de -32768 a 65536.

**Observação** A função **ChrB** é utilizada com dados de byte contidos em um **String**. Em vez de retornar um caractere, que pode ser de um ou dois bytes, **ChrB** sempre retorna um único byte. A função **ChrW** retorna um **String** contendo o caractere Unicode, com exceção de plataformas nas quais Unicode não é suportado, caso em que o comportamento dessa função é idêntico ao da função **Chr**.

## Função Format

Retorna um **Variant (String)** que contém uma expressão formatada de acordo com as instruções contidas em uma expressão de formato.

### Sintaxe

**Format**(*expressão*[, *formato*[, *primeirodiadasemana*[, *primeirasemanadoano*]]])

A sintaxe da função **Format** possui as partes a seguir:

Parte	Descrição
<i>expressão</i>	Obrigatória. Qualquer expressão válida.
<i>formato</i>	Opcional. Uma expressão válida de formato nomeado ou definido pelo usuário.
<i>primeirodiadasemana</i>	Opcional. Uma <u>constante</u> que especifica o primeiro dia da semana.
<i>primeirasemanadoano</i>	Opcional. Uma constante que especifica a primeira semana do ano.

### Definições

O argumento *primeirodiadasemana* possui as definições a seguir:

Constante	Valor	Descrição
<b>vbUseSystem</b>	0	Utilizar definição de API NLS.
<b>VbSunday</b>	1	Domingo (padrão)
<b>vbMonday</b>	2	Segunda-feira
<b>vbTuesday</b>	3	Terça-feira
<b>vbWednesday</b>	4	Quarta-feira
<b>vbThursday</b>	5	Quinta-feira
<b>vbFriday</b>	6	Sexta-feira
<b>vbSaturday</b>	7	Sábado

O argumento *primeirasemanadoano* possui as definições a seguir:

Constante	Valor	Descrição
<b>vbUseSystem</b>	0	Utilizar definição de API NLS.
<b>VbFirstJan1</b>	1	Iniciar com a semana de 1o de janeiro (padrão).
<b>VbFirstFourDays</b>	2	Iniciar com a primeira semana do ano de pelo menos quatro dias.
<b>vbFirstFullWeek</b>	3	Iniciar com a primeira semana completa do ano.

### Comentários

Para formatar	Faça isto
Números	Utilize formatos numéricos predefinidos nomeados ou crie formatos numéricos definidos pelo usuário.
Data e hora	Utilize formatos de data/hora predefinidos nomeados ou crie formatos de data/hora definidos pelo usuário.
Números de série de data e hora	Utilize formatos numéricos ou formatos de data e hora.
Seqüências de caracteres	Crie seus próprios formatos de seqüência de caracteres definidos pelo usuário.

Se você tentar formatar um número sem especificar *formato*, **Format** oferece uma funcionalidade semelhante à da função **Str**, embora reconheça os padrões internacionais. No entanto, números positivos formatados como seqüências de caracteres utilizando-se **Format** não incluem um espaço à esquerda reservado para o sinal do valor; aqueles convertidos utilizando-se **Str** mantêm o espaço à

esquerda.

## Formatos numéricos nomeados (função Format)

A tabela a seguir identifica os nomes de formatos numéricos predefinidos:

<b>Nome do formato</b>	<b>Descrição</b>
<b>General Number</b>	Exibe número sem separador de milhar.
<b>Currency</b>	Exibe número com separador de milhar, se for o caso; exibe dois dígitos à direita do separador decimal. A saída é baseada nas definições de <u>localidade</u> do sistema.
<b>Fixed</b>	Exibe pelo menos um dígito à esquerda e dois à direita do separador decimal.
<b>Standard</b>	Exibe o número com separador de milhar, pelo menos um dígito à esquerda e dois à direita do separador decimal.
<b>Percent</b>	Exibe o número multiplicado por 100 com um sinal de porcentagem (%) à direita; exibe sempre dois dígitos à direita do separador decimal.
<b>Scientific</b>	Utiliza notação científica padrão.
<b>Yes/No</b>	Exibe Não se o número for 0; caso contrário, exibe Sim.
<b>True/False</b>	Exibe <b>False</b> se o número for 0; caso contrário, exibe <b>True</b> .
<b>On/Off</b>	Exibe Desativado se o número for 0; caso contrário, exibe Ativado.

## Formatos numéricos definidos pelo usuário (função Format)

A tabela a seguir identifica caracteres que você pode utilizar para criar formatos numéricos definidos pelo usuário:

<b>Caractere</b>	<b>Descrição</b>
Nenhum	Exibe o número sem formatação.
(0)	<p>Espaço reservado para dígito. Exibe um dígito ou um zero. Se a <u>expressão</u> apresenta um dígito na posição onde o 0 aparece na seqüência de caracteres de formato, exibe-o; caso contrário, exibe um zero nessa posição.</p> <p>Se o número apresenta menos dígitos do que os zeros existentes (de qualquer lado do decimal) na expressão de formato, exibe zeros à esquerda ou à direita. Se o número apresenta mais dígitos à direita do separador decimal do que zeros à direita do separador decimal na expressão de formato, arredonda o número em tantas casas decimais quanto for o número de zeros. Se o número apresenta mais dígitos à esquerda do separador decimal do que zeros à esquerda do separador decimal na expressão de formato, exibe os dígitos extras sem modificação.</p>
(#)	<p>Espaço reservado para dígito. Exibe um dígito ou nada. Se a expressão apresenta um dígito onde o # aparece na seqüência de caracteres de formato, exibe-o; caso contrário, não exibe nada nessa posição.</p> <p>Esse símbolo funciona como o espaço reservado para o dígito 0, exceto que os zeros à direita e à esquerda não são</p>

- exibidos se o número apresentar o mesmo número de dígitos ou menos do que o número de caracteres # de cada lado do separador decimal na expressão de formato.
- (.) Espaço reservado decimal. Em algumas localidades, uma vírgula é utilizada como separador decimal. O espaço reservado decimal determina quantos dígitos são exibidos à esquerda e à direita do separador decimal. Se a expressão de formato contém somente sinais numéricos à esquerda desse símbolo, números menores do que 1 começam com um separador decimal. Para exibir um zero à esquerda exibido com números fracionários, utilize 0 como o primeiro espaço reservado de dígito à esquerda do separador decimal. O caractere realmente utilizado como um espaço reservado decimal na saída formatada depende do Formato numérico reconhecido pelo seu sistema.
- (%) Espaço reservado de porcentagem. A expressão é multiplicada por 100. O caractere de porcentagem (%) é inserido na mesma posição em que aparece na seqüência de caracteres de formato.
- (,) Separador de milhar. Em algumas localidades, um ponto é utilizado como separador de milhar. O separador de milhar separa as centenas dos milhares dentro de um número com quatro ou mais posições à esquerda do separador decimal. A utilização padrão do separador de milhar é especificada se o formato contiver um separador de milhar cercado por espaços reservados para dígitos (0 ou #). Dois separadores de milhar adjacentes ou um separador de milhar imediatamente à esquerda do separador decimal (independente de o decimal ser ou não especificado) significa "escalonar o número dividindo-o por 1000 e arredondando se necessário." Por exemplo, você pode utilizar a seqüência de caracteres de formato "##0,," para representar 100 milhões como 100. Números menores do que 1 milhão são exibidos como 0. Dois separadores de milhar adjacentes em qualquer posição diferente de imediatamente à esquerda do separador decimal são tratados simplesmente como especificação da utilização de um separador de milhar. O caractere realmente utilizado como separador de milhar na saída formatada depende do Formato numérico reconhecido pelo seu sistema.
- (:) Separador de tempo. Em algumas localidades, outros caracteres podem ser utilizados para representar o separador de tempo. O separador de tempo separa horas, minutos e segundos quando os valores de tempo são formatados. O caractere realmente utilizado como separador de tempo na saída formatada depende das definições de seu sistema.
- (/) Separador de data. Em algumas localidades, outros caracteres podem ser utilizados para representar o separador de data. O separador de data separa o dia, o mês e o ano quando os valores de data são formatados. O caractere realmente utilizado como separador de data na saída formatada depende das definições de seu sistema.
- (E- E+ e- e+) Formato científico. Se a expressão de formato contém pelo menos um espaço reservado para dígito (0 ou #) à direita de E-, E+, e-, ou e+, o número é exibido em formato científico e E ou e é inserido entre o número e seu expoente. O número de espaços reservados para dígitos à direita determina o número de dígitos no expoente. Utilize E- ou e- para colocar um sinal de menos ao lado de expoentes negativos. Utilize E+ ou e+ para colocar um sinal de menos ao lado de expoentes negativos e um sinal de mais ao lado de expoentes positivos.

- + \$ ( )	Exibe um caractere literal. Para exibir um caractere diferente daqueles listados, anteceda-o com uma barra invertida (\) ou coloque-o entre aspas duplas (" ").
(\)	Exibe o próximo caractere na seqüência de caracteres de formato. Para exibir um caractere que apresenta um significado especial como um caractere literal, anteceda-o com uma barra invertida (\). A barra invertida, na verdade, não é exibida. Utilizá-la é a mesma coisa que colocar o próximo caractere entre aspas duplas. Para exibir uma barra invertida, utilize duas barras invertidas (\\).  Exemplos de caracteres que não podem ser exibidos como caracteres literais são os caracteres de formatação de data e hora (a, c, d, h, m, n, p, q, s, t, w, y, / e :), os caracteres de formatação numérica (#, 0, %, E, e, vírgula e ponto) e os caracteres de formatação da seqüência de caracteres (@, &, <, > e !).
("ABC")	Exibe a seqüência de caracteres dentro de aspas duplas (" "). Para incluir uma seqüência de caracteres em <b>format</b> a partir do código, você deve utilizar <b>Chr(34)</b> para envolver o texto (34 é o <u>código de caractere</u> para uma aspa ("")).

## Formatos diferentes para valores numéricos diferentes (função Format)

Uma expressão de formato para números definida pelo usuário pode apresentar de uma a quatro seções separadas por ponto-e-vírgula. Se o argumento *format* contiver um dos quatro formatos numéricos nomeados, somente uma seção será permitida.

<b>Se você utilizar</b>	<b>O resultado será</b>
Somente uma seção	A expressão de formato aplica-se a todos os valores.
Duas seções	A primeira seção aplica-se a valores positivos e zeros e a segunda a valores negativos.
Três seções	A primeira seção aplica-se a valores positivos, a segunda a valores negativos e a terceira a zeros.
Quatro seções	A primeira seção aplica-se a valores positivos, a segunda a valores negativos, a terceira a zeros e a quarta a valores <b>Null</b> .

O exemplo a seguir apresenta duas seções: a primeira define o formato para valores positivos e zeros; a segunda define o formato para valores negativos.

```
"$,##0;($#,##0)"
```

Se você incluir pontos-e-vírgulas sem nada entre eles, a seção ausente será impressa utilizando o formato do valor positivo. Por exemplo, o formato a seguir exibe valores positivos e negativos utilizando o formato da primeira seção e exibindo "Zero" se o valor for zero.

```
"$,##0;;\Z\e\r\o"
```

## Exemplo de expressões de formato numérico definidas pelo usuário

A tabela a seguir contém algumas expressões de exemplo de formato para números. (Estes exemplos pressupõem que a definição de localidade do seu sistema seja Português (Brasil) A primeira coluna contém as seqüências de caracteres de formato; as outras contêm a saída resultante se os dados formatados apresentarem o valor fornecido nos cabeçalhos das colunas.

<b>Formato (format)</b>	<b>Positivo 5</b>	<b>Negativo 5</b>	<b>Decimal .5</b>	<b>Null</b>
Seqüência de caracteres de comprimento Zero ("")	5	-5	0,5	
0	5	-5	1	
0,00	5,00	-5,00	0,50	
#.##0	5	-5	1	
#.##0,00;;; Nil	5,00	-5,00	0,50	Nil
R\$#.##0; (R\$#.##0)	R\$5	(R\$5)	R\$1	
R\$#.##0,00; (R\$#.##0,00)	R\$5,00	(R\$5,00)	R\$0,50	
0%	500%	-500%	50%	
0,00%	500,00%	-500,00%	50,00%	
0,00E+00	5,00E+00	-5,00E+00	5,00E-01	
0,00E-00	5,00E00	-5,00E00	5,00E-01	

## Formatos nomeados de data/hora (função Format)

A tabela a seguir identifica os nomes de formato de data e hora predefinidos:

<b>Nome do formato</b>	<b>Descrição</b>
<b>General Date</b>	Exibe uma data e/ou hora. Para números reais, exibe uma data e hora, por exemplo, 3/4/93 17:34. Se não existir uma parte fracionária, exibe somente uma data, por exemplo, 3/4/93. Se não houver uma parte inteira, exibe somente a hora, por exemplo, 17:34. A exibição da data é determinada pelas definições do seu sistema.
<b>Long Date</b>	Exibe uma data de acordo com o formato por extenso de data de seu sistema.
<b>Medium Date</b>	Exibe uma data utilizando o formato médio de data apropriado à versão do idioma do <u>aplicativo host</u> .
<b>Short Date</b>	Exibe uma data utilizando o formato abreviado de data de seu sistema.
<b>Long Time</b>	Exibe uma hora utilizando o formato de hora por extenso de seu sistema, incluindo horas, minutos e segundos.
<b>Medium Time</b>	Exibe hora em formato de 12 horas utilizando horas e minutos e a designação AM/PM.
<b>Short Time</b>	Exibe uma hora utilizando o formato de 24 horas, por exemplo, 17:45.

## Formatos de data/hora definidos pelo usuário (função Format)

A tabela a seguir identifica os caracteres que você pode utilizar para criar formatos de data/hora definidos pelo usuário:

<b>Caractere</b>	<b>Descrição</b>
(:)	Separador de hora. Em algumas <u>localidades</u> , outros caracteres podem ser utilizados para representar o separador de hora. O separador de hora separa horas, minutos e segundos quando os valores de hora são formatados. O caractere real utilizado como separador de hora na saída formatada é determinado pelas definições do seu sistema.
(/)	<u>Separador de data</u> . Em algumas localidades, outros caracteres podem ser utilizados para representar o separador de data. O separador de data separa o dia, o mês e o ano quando os valores de data são formatados. O caractere realmente utilizado como separador de data na saída formatada depende das definições do seu sistema.
c	Exibe a data como dddd e exibe a hora como hhhh, nessa ordem. Exibe somente informações de data se não houver parte fracionária para o número de série da data; exibe somente informações de hora se não houver parte de número inteiro.
d	Exibe o dia como um número sem um zero à esquerda (1 – 31).
dd	Exibe o dia como um número com um zero à esquerda(01 – 31).
ddd	Exibe o dia de forma abreviada (Dom – Sáb).
dddd	Exibe o nome completo do dia (Domingo – Sábado).
ddddd	Exibe a data completa (incluindo dia, mês e ano), formatada de acordo com a definição de formato de data abreviada do seu sistema. Para o Microsoft Windows, o formato padrão de data abreviada é d/m/aa.
dddddd	Exibe um número de série de data como uma data completa (incluindo dia, mês e ano) formatada de acordo com a definição de data por extenso reconhecida pelo seu sistema. Para o Microsoft Windows, o formato padrão de data por extenso é dddd mm, aaaa.
w	Exibe o dia da semana como um número (1 para Domingo até 7 até para Sábado).
ww	Exibe a semana do ano como um número (1 – 54).
m	Exibe o mês como um número sem um zero à esquerda (1 – 12). Se m seguir imediatamente h ou hh, o minuto é exibido ao invés do mês.
mm	Exibe o mês como um número com um zero à esquerda (01 – 12). Se m seguir imediatamente h ou hh, o minuto é exibido ao invés do mês.
mmm	Exibe o mês de forma abreviada (Jan – Dez).
mmmm	Exibe o nome completo do mês (Janeiro – Dezembro).
t	Exibe o trimestre do ano como um número (1 – 4).
a	Exibe o dia do ano como um número (1 – 366).
aa	Exibe o ano como um número de dois dígitos (00 – 99).
aaaa	Exibe o ano como um número de quatro dígitos (100 – 9999).
h	Exibe a hora como um número sem zeros à esquerda (0 – 23).

hh	Exibe a hora como um número com zeros à esquerda (00 – 23).
n	Exibe o minuto como um número sem zeros à esquerda (0 – 59).
nn	Exibe o minuto como um número com zeros à esquerda(00 – 59).
s	Exibe o segundo como um número sem zeros à esquerda (0 – 59).
ss	Exibe o segundo como um número com zeros à esquerda (00 – 59).
h h h h h	Exibe a hora completa (incluindo hora, minuto e segundo), formatada utilizando o separador de hora definido pelo formato de hora reconhecido pelo seu sistema. Um zero à esquerda é exibido se a opção zero à esquerda estiver selecionada e se a hora for anterior a 10:00 ou 22:00. Para o Microsoft Windows, o formato padrão de hora é h:mm:ss.
AM/PM	Utiliza o relógio de 12 horas e exibe AM em letras maiúsculas sem qualquer hora antes do meio-dia; exibe PM em letras maiúsculas com qualquer hora entre meio-dia e 23:59.
am/pm	Utiliza o relógio de 12 horas e exibe AM em letras minúsculas sem qualquer hora antes do meio-dia; exibe PM em letras minúsculas com qualquer hora entre meio-dia e 23:59.
A/P	Utiliza o relógio de 12 horas e exibe um A maiúsculo sem qualquer hora antes do meio-dia; exibe um P maiúsculo com qualquer hora entre meio-dia e 23:59.
a/p	Utiliza o relógio de 12 horas e exibe um A minúsculo sem qualquer hora antes do meio-dia; exibe um P minúsculo com qualquer hora entre meio-dia e 23:59.
AMPM	Utiliza o relógio de 12 horas e exibe uma <u>literal de seqüência de caracteres</u> AM como definido pelo seu sistema com qualquer hora antes de meio-dia; exibe a literal de seqüência de caracteres PM como definido pelo seu sistema com qualquer hora entre meio-dia e 23:59. AMPM podem estar em letras maiúsculas ou minúsculas, mas o tamanho da letra da seqüência de caracteres exibida deve corresponder àquele definido na definição do seu sistema. Para o Microsoft Windows, o formato padrão é AM/PM.

## Exemplo de formatos de data/hora definidos pelo usuário

A seguir são apresentados exemplos de formatos de data e hora definidos pelo usuário para 7 de Dezembro de 1958:

<b>Formato</b>	<b>Exibe</b>
d/m/aa	7/12/58
d-mmm	7-Dez
d-mmmm-aa	7-Dezembro-58
d mmmm	7 Dezembro
mmmm aa	Dezembro 58
hh:mm AM/PM	08:50 PM
h:mm:ss a/p	8:50:35 p
h:mm	20:50
h:mm:ss	20:50:35
d/m/aa h:mm	7/12/58 20:50

## Formatos de seqüência de caracteres definidos pelo usuário (função Format)

Você pode utilizar qualquer um dos caracteres a seguir para criar uma expressão de formato para seqüências de caracteres:

<b>Caractere</b>	<b>Descrição</b>
@	Espaço reservado para caractere. Exibe um caractere ou um espaço. Se a seqüência de caracteres apresenta um caractere onde o símbolo (@) aparece na seqüência de caracteres de formato, exibe-o; caso contrário, exibe um espaço nessa posição. Espaços reservados são preenchidos da direita para a esquerda a menos que exista um ponto de exclamação (!) na seqüência de caracteres de formato.
&	Espaço reservado para caractere. Exibe um caractere ou nada. Se a seqüência de caracteres apresenta um caractere na posição onde o E comercial (&) aparece, exibe-o; caso contrário, não exibe nada. Espaços reservados são preenchidos da direita para a esquerda a menos que exista um ponto de exclamação (!) na seqüência de caracteres de formato.
<	Força letras minúsculas. Exibe todos os caracteres em formato de minúsculas.
>	Força letras maiúsculas. Exibe todos os caracteres em formato de maiúsculas.
!	Força preenchimento de espaços reservados da esquerda para a direita. O padrão é o preenchimento da direita para a esquerda.

## Formatos diferentes para valores de seqüência de caracteres diferentes (função Format)

Uma expressão de formato para seqüências de caracteres pode apresentar uma seção ou duas seções separadas por um ponto-e-vírgula (;).

<b>Se você utilizar</b>	<b>O resultado será</b>
Somente uma seção	O formato se aplicará a todos os dados da seqüência de caracteres.
Duas seções	A primeira seção se aplicará aos dados da seqüência de caracteres e a segunda a valores <b>Null</b> e seqüências de caracteres de comprimento zero ("").

## Função Hex

Retorna um **String** que representa o valor hexadecimal de um número.

### Sintaxe

**Hex**(*número*)

O argumento obrigatório *número* é qualquer expressão numérica válida ou expressão de seqüência de caracteres.

### Comentários

Se *número* não for um número inteiro, deve ser arredondado para o número mais próximo antes de ser avaliado.

Se <i>número</i> for	Hex retornará
<b>Null</b>	Nulo
<b>Empty</b>	Zero (0)
Qualquer outro número	Até oito caracteres hexadecimais

Você pode representar números hexadecimais diretamente ao anteceder números no intervalo apropriado por &H. Por exemplo, &H10 representa 16 decimal em notação hexadecimal.

## Função InStr

Retorna um **Variant (Long)** especificando a posição da primeira ocorrência de uma seqüência de caracteres dentro de outra.

### Sintaxe

**InStr**([*start*, ]*string1*, *string2*[, *compare*])

A sintaxe da função **InStr** apresenta os argumentos a seguir:

Parte	Descrição
<b>start</b>	Opcional. <u>Expressão numérica</u> que define a posição inicial para cada pesquisa. Se omitida, a pesquisa tem início na primeira posição de caractere. Se <b>start</b> contiver <b>Null</b> , um erro será gerado. O argumento <b>start</b> é obrigatório se <b>compare</b> for especificado.
<b>string1</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> sendo pesquisada.
<b>string2</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> sendo pesquisada.
<b>compare</b>	Opcional. Especifica o tipo de <u>comparação de seqüência de caracteres</u> . O argumento <b>compare</b> pode ser omitido, ou pode ser 0, 1 ou 2. Especifique <b>0</b> (padrão) para executar uma comparação binária. Especifique <b>1</b> para executar uma comparação textual, que não coincida maiúsculas e minúsculas. Para o Microsoft Access somente, especifique <b>2</b> para executar uma comparação baseada nas informações contidas em seu banco de dados. Se <b>compare</b> for Nulo, um erro ocorrerá. Se <b>compare</b> for omitido, a configuração <b>Option Compare</b> determinará o tipo de comparação.

### Valores de retorno

Se	InStr retorna
<b>string1</b> tiver comprimento zero	0

**string1** for **Null** Nulo  
**string2** tiver **start**  
 comprimento  
 zero  
**string2** for **Null** Nulo  
**string2** não for 0  
 encontrado  
**string2** for Posição na qual a correspondência é encontrada  
 encontrado dentro de  
**string1**  
**start > string2** 0

### Comentários

A função **InStrB** é utilizada com os dados de byte contidos em uma seqüência de caracteres. Em vez de retornar a posição do caractere da primeira ocorrência de uma seqüência de caracteres dentro de outra, **InStrB** retorna a posição do byte.

## Função LCase

Retorna um **String** que foi convertido para letra minúscula.

### Sintaxe

**LCase**(seqüênciadecaracteres)

O argumento obrigatório seqüênciadecaracteres é qualquer expressão de seqüência de caracteres.válida. Se seqüênciadecaracteres contiver **Null**, será retornado Nulo.

### Comentários

Somente as letras maiúsculas são convertidas para minúsculas; todas as letras minúsculas e caracteres que não sejam letras permanecem inalterados.

## Função Left

Retorna um **Variant (String)** contendo um número especificado de caracteres do lado esquerdo de uma seqüência de caracteres.

### Sintaxe

**Left**(string, length)

A sintaxe da função **Left** apresenta os argumentos nomeados a seguir:

Parte	Descrição
<b>string</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> a partir da qual os caracteres mais à esquerda são retornados. Se <b>string</b> contiver <b>Null</b> , Nulo será retornado.
<b>length</b>	Obrigatório. <b>Variant (Long)</b> . <u>Expressão numérica</u> que indica quantos caracteres devem ser retornados. Se 0, uma seqüência de caracteres de comprimento zero ("") é retornada. Se maior ou igual ao número de caracteres de <b>string</b> , toda a seqüência de caracteres é retornada.

### Comentários

Para determinar o número de caracteres em **string**, utilize a função **Len**.

**Observação** Utilize a função **LeftB** com dados de byte contidos em uma seqüência de caracteres. Ao invés de especificar o número de caracteres a serem retornados, **length** especifica o número de bytes.

## Função Len

Retorna um **Long** que contém o número de caracteres em uma seqüência de caracteres ou o número de bytes necessários para armazenar uma variável.

### Sintaxe

**Len**(*seqüenciadecaracteres* | *nomedavar*)

A sintaxe da função **Len** apresenta as partes a seguir:

Parte	Descrição
<i>string</i>	Qualquer <u>expressão de seqüência de caracteres</u> válida. Se <i>seqüenciadecaracteres</i> contiver <b>Null</b> , Nulo é retornado.
<i>varname</i>	Qualquer nome de <u>variável</u> válido. Se <i>nomedavar</i> contiver Nulo, Nulo será retornado. Se <i>nomedavar</i> for <b>Variant</b> , <b>Len</b> o tratará como se fosse um <b>String</b> e sempre retornará o número de caracteres que ele contiver.

### Comentários

Um (e somente um) dos dois argumentos possíveis deve ser especificado. Com tipos definidos pelo usuário, **Len** retorna o tamanho como ele será gravado no arquivo.

**Observação** Utilize a função **LenB** com dados de byte contidos em uma seqüência de caracteres. Em vez de retornar o número de caracteres em uma seqüência de caracteres, **LenB** retorna o número de bytes utilizados para representar aquela seqüência de caracteres. Com tipos definidos pelo usuário, **LenB** retorna o tamanho memorizado, incluindo qualquer item entre os elementos.

**Observação** **Len** pode não ser capaz de determinar o número real de bytes de armazenamento necessários quando utilizado com seqüências de caracteres de comprimento variável em tipos de dados definidos pelo usuário.

## Instrução LSet

Alinha uma seqüência de caracteres à esquerda dentro de uma variável de seqüência de caracteres, ou copia uma variável de um tipo definido pelo usuário para outra variável de um tipo diferente definido pelo usuário.

### Sintaxe

**LSet** *varseq* = *seqüenciadecaracteres*

**LSet** *nomedavar1* = *nomedavar2*

A sintaxe da instrução **LSet** apresenta as partes a seguir:

Parte	Descrição
<i>stringvar</i>	Obrigatório. Nome da <u>variável</u> de seqüência de caracteres.
<i>string</i>	Obrigatório. <u>Expressão de seqüência de caracteres</u> a ser alinhada à esquerda dentro de <i>varseq</i> .
<i>varname1</i>	Obrigatório. Nome de variável do tipo definido pelo usuário para a qual é feita a cópia.
<i>varname2</i>	Obrigatório. Nome de variável do tipo definido pelo usuário da qual é feita a cópia.

### Comentários

**LSet** substitui qualquer caractere deixado em *vaseq* por espaços.

Se *seqüênciadecaracteres* for maior do que *vaseq*, **LSet** coloca somente os caracteres mais à esquerda, até o comprimento de *vaseq*, em *vaseq*.

---

**Aviso** Não é recomendável utilizar **LSet** para copiar uma variável de um tipo definido pelo usuário para um tipo diferente. A cópia de dados de um tipo de dados para espaço reservado a um tipo de dados diferente pode acarretar resultados imprevisíveis.

Quando você copia uma variável de um tipo definido pelo usuário para outro, os dados binários de uma variável são copiados para o espaço de memória do outro, sem levar em consideração os tipos de dados especificados para os elementos.

---

## Funções LTrim, Rtrim e Trim

Retorna um **Variant (String)** que contém uma cópia de uma seqüência de caracteres especificada sem espaços à esquerda (**LTrim**), espaços à direita (**RTrim**) ou espaços à direita e à esquerda (**Trim**).

### Sintaxe

**LTrim**(*seqüênciadecaracteres*)

**RTrim**(*seqüênciadecaracteres*)

**Trim**(*seqüênciadecaracteres*)

O argumento *seqüênciadecaracteres* exigido é qualquer expressão de seqüência de caracteres. válida. Se *seqüênciadecaracteres* contiver **Null**, **Null** será retornado.

## Função Mid

Retorna um **Variant (String)** que contém um número especificado de caracteres de uma seqüência de caracteres.

### Sintaxe

**Mid**(*string*, *start*[, *length*])

A sintaxe da função **Mid** apresenta os argumentos nomeados a seguir:

<b>Parte</b>	<b>Descrição</b>
<b>string</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> da qual são retornados caracteres. Se <b>string</b> contiver <b>Null</b> , <b>Null</b> será retornado.
<b>start</b>	Obrigatório; <b>Long</b> . A posição do caractere em <b>string</b> na qual a parte a ser removida tem início. Se <b>start</b> for maior do que o número de caracteres em <b>string</b> , <b>Mid</b> retorna uma seqüência de caracteres de comprimento zero ("").
<b>length</b>	Opcional; <b>Variant (Long)</b> . Número de caracteres a ser retornado. Se omitido ou se houver menos caracteres no texto do que em <b>length</b> (incluindo o caractere em <b>start</b> ), todos os caracteres da posição <b>start</b> até o final da seqüência de caracteres serão retornados.

### Comentários

Para determinar o número de caracteres em **string**, utilize a função **Len**.

**Observação** Utilize a função **MidB** com dados de byte contidos em uma seqüência de caracteres. Em vez de especificar o número de caracteres, os argumentos especificam o número de bytes.

## Instrução Mid

Substitui um número especificado de caracteres em uma variável Variant (String) com caracteres de outra seqüência de caracteres.

### Sintaxe

**Mid**(*vargseq*, *início*[, *comprimento*]) = *seqüênciadecaracteres*

A sintaxe da instrução **Mid** apresenta as partes a seguir:

<u>Parte</u>	<u>Descrição</u>
<i>vargseq</i>	Obrigatório. Nome da variável da seqüência de caracteres a ser modificada.
<i>início</i>	Obrigatório; <b>Variant (Long)</b> . Posição de caractere em <i>vargseq</i> onde a substituição de texto tem início.
<i>comprimento</i>	Opcional; <b>Variant (Long)</b> . Número de caracteres a serem substituídos. Se omitida, toda a <i>seqüênciadecaracteres</i> é utilizada.
<i>seqüênciadecaracteres</i>	Obrigatório. <u>Expressão de seqüência de caracteres</u> que substitui parte de <i>vargseq</i> .

### Comentários

O número de caracteres substituído é sempre menor ou igual ao número de caracteres em *vargseq*.

**Observação** Utilize a instrução **MidB** com dados de byte contidos em uma seqüência de caracteres. Na instrução **MidB**, *início* especifica a posição do byte dentro de *vargseq* onde a substituição tem início e *comprimento* especifica o número de bytes a ser substituído.

## Função Oct

Retorna um **Variant (String)** que representa o valor octal de um número.

### Sintaxe

**Oct**(*número*)

O argumento obrigatório *número* é qualquer expressão numérica válida ou expressão de seqüência de caracteres.

### Comentários

Se *número* não for um número inteiro, deve ser arredondado para o número inteiro mais próximo antes de ser avaliado.

<u>Se número for</u>	<u>Oct retornará</u>
<b>Null</b>	<b>Null</b>
<b>Empty</b>	Zero (0)
Qualquer outro número	Até 11 caracteres octais

Você pode representar números octais diretamente ao anteceder números no intervalo apropriado com &O. Por exemplo, &O10 é a notação octal para o decimal 8.

## Função Right

Retorna um **Variant (String)** que contém um número específico de caracteres do lado direito de uma seqüência de caracteres.

### Sintaxe

#### Right(*string*, *length*)

A sintaxe da função **Right** apresenta os argumentos nomeados a seguir:

Parte	Descrição
<i>string</i>	Obrigatório. <u>Expressão de seqüência de caracteres</u> da qual os caracteres mais à direita são retornados. Se <i>string</i> contiver <b>Null</b> , <b>Null</b> será retornado.
<i>length</i>	Obrigatório; <b>Variant (Long)</b> . <u>Expressão numérica</u> indicando quantos caracteres devem ser retornados. Se 0, uma seqüência de caracteres de comprimento zero ("") é retornada. Se maior ou igual ao número de caracteres em <i>string</i> , toda a seqüência de caracteres é retornada.

### Comentários

Para determinar o número de caracteres em *string*, utilize a função **Len**.

**Observação** Utilize a função **RightB** com dados de byte contidos em uma seqüência de caracteres. Em vez de especificar o número de caracteres a ser retornado, *length* especifica o número de bytes.

## Instrução RSet

Alinha uma seqüência de caracteres à direita dentro de uma variável de seqüência de caracteres.

### Sintaxe

#### RSet *valseq* = *seqüenciadecaracteres*

A sintaxe da instrução **RSet** apresenta as partes a seguir:

Parte	Descrição
<i>valseq</i>	Obrigatório. Nome de variável da seqüência de caracteres.
<i>seqüenciadecaracteres</i>	Obrigatório. <u>Expressão de seqüência de caracteres</u> a ser alinhada à direita dentro de <i>valseq</i> .

### Comentários

Se *valseq* for maior do que *seqüenciadecaracteres*, **RSet** substitui qualquer caractere deixado em *valseq* por espaços, de volta a seu início.

**Observação** **RSet** não pode ser utilizado com tipos definidos pelo usuário.

## Função Space

Retorna um **Variant (String)** que consiste em um número especificado de espaços.

### Sintaxe

#### Space(*número*)

O argumento obrigatório *número* é o número de espaços que você deseja na seqüência de caracteres.

### Comentários

A função **Space** é útil para formatar saídas e limpar dados em seqüências de caracteres de comprimento fixo.

## Função Str

Retorna uma representação de um número **Variant (String)**.

### Sintaxe

**Str**(*número*)

O argumento obrigatório *número* é um **Long** contendo qualquer expressão numérica.válida.

### Comentários

Quando números são convertidos em seqüências de caracteres, um espaço à esquerda é sempre reservado para o sinal de *número*. Se *número* for positivo, a seqüência de caracteres retornada terá um espaço à esquerda e o sinal de mais estará implícito.

Utilize a função **Format** para converter valores numéricos que você deseja formatar como datas, horas, moeda ou outros formatos definidos pelo usuário. Ao contrário de **Str**, a função **Format** não inclui um espaço à esquerda para o sinal de *número*.

**Observação** A função **Str** reconhece somente o ponto (.) como um separador decimal válido. Quando separadores decimais diferentes puderem ser utilizados (por exemplo, em aplicativos internacionais), utilize **CStr** para converter um número em uma seqüência de caracteres.

## Função StrComp

Retorna **Variant (Integer)** que indica o resultado de uma comparação de seqüência de caracteres.

### Sintaxe

**StrComp**(*string1*, *string2* [, *compare*])

A sintaxe da função **StrComp** apresenta os argumentos nomeados a seguir:

Parte	Descrição
<b>string1</b>	Obrigatório. Qualquer <u>expressão de seqüência de caracteres</u> .válida.
<b>string2</b>	Obrigatório. Qualquer expressão de seqüência de caracteres válida.
<b>compare</b>	Opcional. Especifica o tipo de comparação de seqüência de caracteres. O <u>argumento compare</u> pode ser omitido ou pode ser 0, 1 ou 2. Especifique <b>0</b> (padrão) para realizar uma comparação binária. Especifique <b>1</b> para realizar uma comparação textual. Para o Microsoft Access somente, especifique <b>2</b> para realizar uma comparação baseada nas informações contidas em seu banco de dados. Se <b>compare</b> for <b>Null</b> , um erro será gerado. Se <b>compare</b> for omitido, a definição <b>Option Compare</b> determinará o tipo de comparação.

### Valores de retorno

Se	StrComp retornará
<b>string1</b> for menor do que <b>string2</b>	-1
<b>string1</b> for igual a <b>string2</b>	0
<b>string1</b> for maior do que <b>string2</b>	1
<b>string1</b> ou <b>string2</b>	Null

for **Null**

## Função StrConv

Retorna um **Variant (String)** convertido como especificado.

### Sintaxe

**StrConv**(*string, conversion*)

A sintaxe da função **StrConv** apresenta os argumentos nomeados a seguir:

Parte	Descrição
<b>string</b>	Obrigatório. <u>Expressão de seqüência de caracteres</u> a ser convertida.
<b>conversion</b>	Obrigatório; <b>Integer</b> . A soma dos valores especifica o tipo de conversão a ser realizada.

### Definições

As definições do argumento conversion são:

Constante	Valor	Descrição
<b>vbUpperCase</b>	1	Converte a seqüência de caracteres para caracteres em maiúsculas.
<b>VbLowerCase</b>	2	Converte a seqüência de caracteres para caracteres em minúsculas.
<b>VbProperCase</b>	3	Converte a primeira letra de cada palavra da seqüência de caracteres para maiúscula.
<b>vbWide*</b>	4*	Converte os caracteres estreitos (byte simples) da seqüência de caracteres em caracteres largos (byte duplo).
<b>VbNarrow*</b>	8*	Converte os caracteres largos (byte duplo) da seqüência de caracteres em caracteres estreitos (byte simples).
<b>vbKatakana**</b>	16**	Converte os caracteres Hiragana da seqüência de caracteres em caracteres Katakana.
<b>vbHiragana**</b>	32**	Converte os caracteres Katakana da seqüência de caracteres em caracteres Hiragana.
<b>vbUnicode</b>	64	Converte a seqüência de caracteres para <u>Unicode</u> utilizando a página de código padrão do sistema.
<b>vbFromUnicode</b>	128	Converte a seqüência de caracteres de Unicode para a página de código padrão do sistema.

\* Aplica-se a localidades do Extremo Oriente.

\*\* Aplica-se somente ao Japão.

**Observação** Essas constantes são especificadas pelo Visual Basic para Aplicativos. Dessa forma, elas podem ser utilizadas em qualquer lugar de seu código no lugar dos valores reais. A maioria delas pode ser combinada, por exemplo **vbUpperCase + vbWide**, exceto quando forem mutuamente exclusivas, como **vbUnicode + vbFromUnicode**. As constantes **vbWide**, **vbNarrow**, **vbKatakana** e **vbHiragana** causam erros em tempo de execução quando utilizadas em localidades onde não são aplicadas.

A seguir são apresentados separadores de palavras válidos para a utilização apropriada de letras maiúsculas e minúsculas: **Null** (**Chr\$(0)**), tab horizontal (**Chr\$(9)**), alimentação de linha (**Chr\$(10)**), tab vertical (**Chr\$(11)**), alimentação de formulário (**Chr\$(12)**), retorno de carro (**Chr\$(13)**), espaço (SBCS) (**Chr\$(32)**). O valor real de um espaço varia de acordo com o país para DBCS.

## Função String

Retorna um **Variant (String)** que contém uma seqüência de caracteres repetidos do comprimento especificado.

### Sintaxe

**String**(*number*, *character*)

A sintaxe da função **String** apresenta os argumentos nomeados a seguir:

<b>Parte</b>	<b>Descrição</b>
<b>number</b>	Obrigatório; <b>Long</b> . Comprimento da seqüência de caracteres retornada. Se <b>number</b> contiver <b>Null</b> , <b>Null</b> será retornado.
<b>character</b>	Obrigatório; <b>Variant</b> . <u>Código de caractere</u> que especifica o caractere ou a <u>expressão de seqüência de caracteres</u> cujo primeiro caractere é utilizado para construir a seqüência de caracteres de retorno. Se <b>character</b> contiver <b>Null</b> , <b>Null</b> será retornado.

### Comentários

Se você especificar um número maior do que 255 para **character**, **String** converterá o número em um código de caracteres válido utilizando a seguinte fórmula:

**character Mod 256**

## Função UCase

Retorna um **Variant (String)** que contém a seqüência de caracteres especificada, convertida para letra maiúscula.

### Sintaxe

**UCase**(seqüência de caracteres)

O argumento seqüência de caracteres exigido é qualquer expressão de seqüência de caracteres válida. Se seqüência de caracteres contiver **Null**, **Null** será retornado.

### Comentários

Somente as letras minúsculas são convertidas para maiúsculas; todas as letras maiúsculas e caracteres diferentes de letras permanecem inalterados.

## Erros interceptáveis

Os erros interceptáveis podem ocorrer quando um aplicativo está sendo executado. Alguns erros interceptáveis podem ocorrer também durante o desenvolvimento ou a compilação. Você pode testar e responder a erros interceptáveis utilizando a instrução **On Error** e o objeto **Err**. Os números de erros não utilizados no intervalo de 1 a 1000 são reservados para uso futuro pelo Visual Basic.

<b>Código</b>	<b>Mensagem</b>
0	
3	<u>Return sem GoSub</u>
5	<u>Chamada de procedimento inválida</u>
6	<u>Sobrecarga</u>
7	<u>Memória insuficiente</u>
9	<u>Subscrito fora do intervalo</u>
10	<u>Esta matriz é fixa ou está temporariamente bloqueada</u>
11	<u>Divisão por zero</u>
13	<u>Tipo incompatível</u>
14	<u>Espaço insuficiente para seqüência de caracteres</u>
16	<u>Expressão muito complexa</u>
17	<u>Não é possível executar a operação solicitada</u>
18	<u>Ocorreu uma interrupção do usuário</u>
20	<u>Recomeçar sem erro</u>
28	<u>Espaço insuficiente para pilha</u>
35	<u>Sub, Function ou Property não definida</u>
47	<u>Número excessivo de clientes do aplicativo DLL</u>
48	<u>Erro ao carregar DLL</u>
49	<u>Convenção de chamada DLL inválida</u>
51	<u>Erro interno</u>
52	<u>Nome ou número de arquivo inválido</u>
53	<u>O arquivo não foi encontrado</u>
54	<u>Modo de arquivo inválido</u>
55	<u>O arquivo já está aberto</u>
57	<u>Erro de dispositivo de E/S</u>
58	<u>O arquivo já existe</u>
59	<u>Comprimento de registro inválido</u>
61	<u>Disco cheio</u>
62	<u>Entrada depois do fim do arquivo</u>
63	<u>Número de registro inválido</u>
67	<u>Número excessivo de arquivos</u>
68	<u>Dispositivo não disponível</u>
70	<u>Permissão negada</u>
71	<u>O disco não está pronto</u>
74	<u>Não é possível renomear com unidade de disco diferente</u>
75	<u>Erro de acesso a caminho/arquivo</u>
76	<u>Caminho não encontrado</u>
91	<u>A variável de objeto ou a variável de bloco With não foi definida</u>
92	<u>Loop 'For' não inicializado</u>
93	<u>Seqüência de caracteres padrão inválida</u>
94	<u>Uso de <b>Null</b> inválido</u>
97	<u>Não é possível chamar procedimento Friend para um objeto que não é uma instância da classe de definição</u>
298	<u>DLL do sistema não pode ser carregada</u>

- 320 Não é possível utilizar nomes de dispositivos em nomes de arquivos específicos
- 321 Formato de arquivo inválido
- 322 Não é possível criar arquivo temporário necessário
- 325 Formato inválido no arquivo de recursos
- 327 Nome do valor de dados não encontrado
- 328 Parâmetro ilegal; não é possível gravar matrizes
- 335 Não foi possível acessar registro do sistema
- 336 Componente ActiveX não foi registrado corretamente
- 337 Componente ActiveX não foi encontrado
- 338 Componente ActiveX não executou corretamente
- 360 Objeto já carregado
- 361 Não é possível carregar ou descarregar este objeto
- 363 Controle ActiveX especificado não foi encontrado
- 364 Objeto foi descarregado
- 365 Não é possível carregar dentro desse contexto
- 368 O arquivo especificado está desatualizado. Este programa exige uma versão posterior
- 371 O objeto especificado não pode ser utilizado como um formulário proprietário de Show
- 380 Valor de propriedade inválido
- 381 Índice de matriz de propriedades inválido
- 382 Propriedade Set não pode ser executada em tempo de execução
- 383 Propriedade Set não pode ser utilizada com uma propriedade somente leitura
- 385 É necessário o índice de matriz de propriedade
- 387 Propriedade Set não permitida
- 393 Propriedade Get não pode ser executada em tempo de execução
- 394 Propriedade Get não pode ser executada em propriedade somente gravação
- 400 Formulário já exibido; impossível exibir de forma modal
- 402 Código deve fechar o formulário modal superior
- 419 Permissão para utilizar objeto negada
- 422 Propriedade não encontrada
- 423 Propriedade ou método não foi encontrado
- 424 Objeto é obrigatório
- 425 Uso inválido de objeto
- 429 O componente ActiveX não pode criar um objeto ou retornar referência a esse objeto
- 430 A classe não aceita Automação
- 432 O nome do arquivo ou o nome da classe não foi encontrado durante a operação de Automação
- 438 O objeto não aceita esta propriedade ou método
- 440 Erro de automação
- 442 A conexão à biblioteca de objetos ou de tipos para processo remoto foi perdida
- 443 O objeto de Automação não possui um valor padrão
- 445 O objeto não suporta esta ação
- 446 O objeto não suporta argumentos nomeados
- 447 O objeto não aceita a definição atual de localidade
- 448 O argumento nomeado não foi encontrado

- 449 Argumento não opcional ou atribuição de propriedade inválida
- 450 Número de argumentos incorreto ou atribuição de propriedade inválida
- 451 Object não é uma coleção
- 452 Ordinal inválido
- 453 A função DLL especificada não foi encontrada
- 454 O recurso de código não foi encontrado
- 455 Erro de proteção de recurso de código
- 457 Esta tecla já está associada a um elemento desta coleção
- 458 A variável utiliza um tipo de automação não suportada no Visual Basic
- 459 Este componente não suporta eventos
- 460 Formato da área de transferência inválido
- 461 Formato especificado não corresponde ao formato dos dados
- 480 Não é possível criar imagem AutoRedraw
- 481 Figura inválida
- 482 Erro na impressora
- 483 Driver da impressora não aceita a propriedade especificada
- 484 Problemas ao obter informações da impressora a partir do sistema. Certifique-se de que a impressora esteja instalada corretamente
- 485 Tipo de figura inválido
- 486 Não é possível imprimir imagem de formulário neste tipo de impressora
- 520 Não é possível esvaziar a Área de transferência
- 521 Não é possível abrir a Área de transferência
- 735 Não é possível salvar arquivo no diretório TEMP
- 744 Texto procurado não encontrado
- 746 Substituições muito longas
- 31001 Memória insuficiente
- 31004 Nenhum objeto
- 31018 Classe não está definida
- 31027 Não é possível ativar objeto
- 31032 Não foi possível criar objeto incorporado
- 31036 Erro ao salvar o arquivo
- 31036 Erro ao carregar do arquivo
- 31037

## Conjunto de caracteres (128 a 255)

128	•	160	[espaço]	192	À	224	à
129	•	161	¡	193	Á	225	á
130	•	162	¢	194	Â	226	â
131	•	163	£	195	Ã	227	ã
132	•	164	¤	196	Ä	228	ä
133	•	165	¥	197	Å	229	å
134	•	166		198	Æ	230	æ
135	•	167	§	199	Ç	231	ç
136	•	168	¨	200	È	232	è
137	•	169	©	201	É	233	é
138	•	170	ª	202	Ê	234	ê
139	•	171	«	203	Ë	235	ë
140	•	172	¬	204	Ì	236	ì
141	•	173	-	205	Í	237	í
142	•	174	®	206	Î	238	î
143	•	175	¯	207	Ï	239	ï
144	•	176	°	208	Ð	240	ð
145	´	177	±	209	Ñ	241	ñ
146	´	178	²	210	Ò	242	ò
147	•	179	³	211	Ó	243	ó
148	•	180	´	212	Ô	244	ô
149	•	181	µ	213	Õ	245	õ
150	•	182	¶	214	Ö	246	ö
151	•	183	·	215	×	247	÷
152	•	184	¸	216	Ø	248	ø
153	•	185	¹	217	Ù	249	ù
154	•	186	º	218	Ú	250	ú
155	•	187	»	219	Û	251	û
156	•	188	¼	220	Ü	252	ü
157	•	189	½	221	Ý	253	ý
158	•	190	¾	222	Þ	254	þ
159	•	191	¿	223	ß	255	ÿ

- Estes caracteres não são aceitos pelo Microsoft Windows.

## Operador AddressOf

Um operador que faz com que o endereço do procedimento que ele precede seja passado a um procedimento API que espere um ponteiro de função naquela posição na lista de argumentos.

### Sintaxe

**AddressOf** *procedurename*

O *procedurename* obrigatório especifica o procedimento cujo endereço deve ser passado. Ele deve representar um procedimento em um módulo padrão no projeto em que é feita a chamada.

### Comentários

Quando um nome de procedimento aparece em uma lista de argumentos, normalmente o procedimento é avaliado e o endereço do valor de retorno do procedimento é passado. **AddressOf** permite que o endereço do procedimento seja passado a uma função API Windows em uma biblioteca de vínculos dinâmicos (DLL), ao invés de passar o valor de retorno do procedimento. A função API pode então usar o endereço para chamar o procedimento em Basic, um processo conhecido como chamada de retorno. O operador **AddressOf** aparece somente na chamada ao procedimento API. Entretanto, na instrução **Declare** que descreve a função API à qual é passado o ponteiro, o argumento de endereço do procedimento deve ser declarado **As Any**.

Embora você possa usar **AddressOf** para passar ponteiros de procedimento entre procedimentos Basic, você não pode chamar uma função do tipo ponteiro a partir de Basic. Isto significa, por exemplo, que uma classe escrita em Basic não pode efetuar uma chamada de retorno a seu controlador usando este tipo de ponteiro. Ao usar **AddressOf** para passar um ponteiro de procedimento entre procedimentos dentro do Basic, o parâmetro do procedimento chamado deve ser digitado **As Long**.

---

**Atenção** Usar **AddressOf** pode provocar resultados imprevisíveis se você não compreender totalmente o conceito de chamadas de retorno de função. Você deve compreender como a parte Basic da chamada de retorno funciona e também o código da DLL à qual você está passando seu endereço de função. Depurar este tipo de interações é difícil, pois o programa é executado no mesmo processo que o ambiente de desenvolvimento. Em alguns casos, a depuração sistemática pode não ser possível.

---

## Método Assert

Suspende condicionalmente a execução na linha em que aparece o método.

### Sintaxe

*object.Assert booleanexpression*

A sintaxe do método **Assert** tem o seguinte qualificador de objeto e argumento:

Parte	Descrição
<i>object</i>	Obrigatório. Sempre o objeto <b>Debug</b> .
<i>Booleanexpression</i>	Obrigatório. Uma <u>expressão</u> que avalia ou <b>True</b> ou <b>False</b> .

### Comentários

Acionamentos de **Assert** somente funcionam dentro do ambiente de desenvolvimento. Quando o módulo é compilado em um executável, as chamadas do método sobre o objeto **Debug** são omitidas.

Todas as *booleanexpression* são sempre avaliadas. Por exemplo, mesmo se a primeira parte de uma expressão **And** for avaliada como **False**, a expressão inteira será avaliada.

## Friend

Modifica a definição de um procedimento em um módulo de classe para tornar o procedimento passível de chamada a partir de módulos que estejam fora da classe, mas que façam parte do projeto dentro do qual a classe é definida.

### Sintaxe

[**Private** | **Friend** | **Public**] [**Static**] [**Sub** | **Function** | **Property**] *procedurename*

O *procedurename* obrigatório é o nome do procedimento a ser tornado visível em todo o projeto, mas não visível a controladores da classe.

### Comentários

Procedimentos **Public** em uma classe podem ser chamados a partir de qualquer lugar, mesmo por controladores de ocorrências da classe. Declarar um procedimento como **Private** impede que controladores do objeto chamem o procedimento, mas também impede que o procedimento seja chamado a partir do projeto em que a própria classe está definida. **Friend** torna o procedimento visível através do projeto, mas não a um controlador de uma ocorrência do objeto. **Friend** somente pode aparecer em módulos de classe, e somente pode modificar nomes de procedimento, não variáveis ou tipos. Os procedimentos em uma classe podem acessar os procedimentos **Friend** de todas as outras classes em um projeto. Procedimentos **Friend** não aparecem na biblioteca de tipos de suas classes. Um procedimento **Friend** não pode ser posteriormente acoplado.

## Instrução RaiseEvent

Dispara um evento declarado a nível de módulo dentro de uma classe, formulário ou documento.

### Sintaxe

**RaiseEvent** *eventname* [(*argumentlist*)]

O *eventname* obrigatório é o nome de um evento declarado dentro do módulo e segue a convenção de nomeação de variável do Basic.

A sintaxe da instrução **RaiseEvent** tem estas partes:

Parte	Descrição
<i>eventname</i>	Obrigatório. Nome do evento a ser disparado.

*argumentlist* Opcional. Lista de variáveis, matrizes ou expressões delimitadas por vírgulas. A *argumentlist* deve ser colocada entre parênteses. Se não existir nenhum argumento, os parênteses devem ser omitidos.

### Comentários

Se o evento não foi declarado em um módulo em que ele foi provocado, ocorre um erro. O fragmento abaixo ilustra uma declaração de evento e um procedimento em que o evento é provocado.

```
' Declarar um evento a nível de módulo de um módulo de classe
Event LogonCompleted (UserName as String)

Sub
    ' Produzir o evento.
    RaiseEvent LogonCompleted ("AntoineJan")
End Sub
```

Se o evento não tem nenhum argumento, incluindo parênteses vazios, no **RaiseEvent**, seu acionamento produz um erro. Você não pode usar **RaiseEvent** para disparar eventos que não sejam explicitamente declarados no módulo. Por exemplo, se um formulário tiver um evento Click, você não poderá dispará-lo usando **RaiseEvent**. Se você declarar um evento Click no módulo de formulário, ele encobrirá o próprio evento Click do formulário. Você ainda pode acionar o evento Click do formulário usando a sintaxe normal para chamar o evento, mas não usando a instrução **RaiseEvent**.

O disparo do evento é feito na ordem em que as conexões são estabelecidas. Como os eventos podem ter parâmetros **ByRef**, o processo que conecta posteriormente pode receber parâmetros que foram alterados por um manipulador de eventos anterior.

### Exemplo do operador AddressOf

O exemplo abaixo cria um formulário com uma caixa de listagem contendo uma lista em ordem alfabética das fontes de seu sistema.

Para executar este exemplo, crie um formulário contendo uma caixa de listagem. O código para o formulário é o seguinte:

```
Option Explicit

Private Sub Form_Load()
    Module1.FillListWithFonts List1
End Sub
```

Coloque o código abaixo em um módulo. O terceiro argumento na definição da função EnumFontFamilies é um **Long** que representa um procedimento. O argumento deve conter o endereço do procedimento, ao invés do valor retornado pelo procedimento. Na chamada a EnumFontFamilies, o terceiro argumento exige o operador **AddressOf** para retornar o endereço do procedimento EnumFontFamProc que é o nome do procedimento de retorno de chamado que você fornece ao chamar a função API Windows **EnumFontFamilies**. O Windows chama EnumFontFamProc uma vez para cada uma das famílias de fonte no sistema quando você passa **AddressOf** EnumFontFamProc a **EnumFontFamilies**. O último argumento passado a **EnumFontFamilies** especifica a caixa de listagem onde as informações são exibidas.

```
'Tipos de enumeração de fonte
Public Const LF_FACESIZE = 32
Public Const LF_FULLFACESIZE = 64

Type LOGFONT
    lfHeight As Long
    lfWidth As Long
    lfEscapement As Long
    lfOrientation As Long
    lfWeight As Long
```

```

lfItalic As Byte
lfUnderline As Byte
lfStrikeOut As Byte
lfCharSet As Byte
lfOutPrecision As Byte
lfClipPrecision As Byte
lfQuality As Byte
lfPitchAndFamily As Byte
lfFaceName(LF_FACESIZE) As Byte

```

End Type

Type NEWTEXTMETRIC

```

tmHeight As Long
tmAscent As Long
tmDescent As Long
tmInternalLeading As Long
tmExternalLeading As Long
tmAveCharWidth As Long
tmMaxCharWidth As Long
tmWeight As Long
tmOverhang As Long
tmDigitizedAspectX As Long
tmDigitizedAspectY As Long
tmFirstChar As Byte
tmLastChar As Byte
tmDefaultChar As Byte
tmBreakChar As Byte
tmItalic As Byte
tmUnderlined As Byte
tmStruckOut As Byte
tmPitchAndFamily As Byte
tmCharSet As Byte
ntmFlags As Long
ntmSizeEM As Long
ntmCellHeight As Long
ntmAveWidth As Long

```

End Type

```

' Sinalizadores de campo ntmFlags
Public Const NTM_REGULAR = &H40&
Public Const NTM_BOLD = &H20&
Public Const NTM_ITALIC = &H1&

```

```

' Sinalizadores tmPitchAndFamily
Public Const TMPF_FIXED_PITCH = &H1
Public Const TMPF_VECTOR = &H2
Public Const TMPF_DEVICE = &H8
Public Const TMPF_TRUETYPE = &H4

```

```

Public Const ELF_VERSION = 0
Public Const ELF_CULTURE_LATIN = 0

```

```

' Máscaras EnumFonts
Public Const RASTER_FONTTYPE = &H1
Public Const DEVICE_FONTTYPE = &H2
Public Const TRUETYPE_FONTTYPE = &H4

```

```

Declare Function EnumFontFamilies Lib "gdi32" Alias _
    "EnumFontFamiliesA"
    (ByVal hDC As Long, ByVal lpszFamily As String,
    ByVal lpEnumFontFamProc As Long, LParam As Any) As Long
Declare Function GetDC Lib "user32" (ByVal hWnd As Long) As Long

```

```
Declare Function ReleaseDC Lib "user32" (ByVal hWnd As Long, _
    ByVal hDC As Long) As Long

Function EnumFontFamProc(lpNLF As LOGFONT, lpNTM As NEWTEXTMETRIC, _
    ByVal FontType As Long, LParam As ListBox) As Long
Dim FaceName As String
Dim FullName As String
    FaceName = StrConv(lpNLF.lfFaceName, vbUnicode)
    LParam.AddItem Left$(FaceName, InStr(FaceName, vbNullChar) - 1)
    EnumFontFamProc = 1
End Function

Sub FillListWithFonts(LB As ListBox)
Dim hDC As Long
    LB.Clear
    hDC = GetDC(LB.hWnd)
    EnumFontFamilies hDC, vbNullString, AddressOf EnumFontFamProc, LB
    ReleaseDC LB.hWnd, hDC
End Sub
```

### Exemplo do método Assert

O exemplo abaixo mostra como usar o método **Assert**. O exemplo exige um formulário com dois controles de botão nele. Os nomes de botão padrão são Command1 e Command2.

Quando o exemplo é executado, clicar no botão Command1 liga e desliga o texto no botão entre 0 e 1. Clicar Command2 nada provoca ou provoca uma declaração, dependendo do valor exibido em Command1. A declaração encerra a execução com a última instrução executada, a linha `Debug.Assert`, realçada.

```
Option Explicit
Private blnAssert As Boolean
Private intNumber As Integer

Private Sub Command1_Click()
    blnAssert = Not blnAssert
    intNumber = IIf(intNumber <> 0, 0, 1)
    Command1.Caption = intNumber
End Sub

Private Sub Command2_Click()
    Debug.Assert blnAssert
End Sub

Private Sub Form_Load()
    Command1.Caption = intNumber
    Command2.Caption = "Assert Tester"
End Sub
```

### Exemplo Friend

Quando colocado em um módulo de classe, o código abaixo torna a variável membro `dblBalance` acessível a todos os usuários da classe dentro do projeto. Qualquer usuário da classe pode obter o valor; somente o código dentro do projeto pode designar um valor àquela variável.

```
Private dblBalance As Double

Public Property Get Balance() As Double
    Balance = dblBalance
End Property

Friend Property Let Balance(dblNewBalance As Double)
    dblBalance = dblNewBalance
End Property
```

### Exemplo da instrução RaiseEvent

O exemplo abaixo usa eventos para uma contagem regressiva dos segundos durante uma demonstração da mais rápida corrida de 100 metros rasos. O código ilustra todos os métodos, propriedades e instruções relacionados com evento, incluindo a instrução **RaiseEvent**.

A classe que provoca um evento na origem do evento e as classes que implementam o evento são os coletores. Uma origem de evento pode ter múltiplos coletores para os eventos que ela gera. Quando a classe provoca o evento, o evento é disparado em todas as classes que escolheram coletar eventos para aquela ocorrência do objeto.

O exemplo também usa um formulário (`Form1`) com um botão (`Command1`), um rótulo (`Label1`), e duas caixas de texto (`Text1` e `Text2`). Quando você clica no botão, a primeira caixa de textos exibe "A partir de agora" e a segunda inicia a contagem dos segundos. Quando o tempo total (9,84 segundos) tiver decorrido, a primeira caixa de texto exibe "Até agora" e a segunda exibe "9.84"

O código para `Form1` especifica os estados inicial e final do formulário. Ele também contém o código executado quando eventos são provocados.

```
Option Explicit

Private WithEvents mText As TimerState

Private Sub Command1_Click()
    Text1.Text = "A partir de agora"
    Text1.Refresh
    Text2.Text = "0"
    Text2.Refresh
    Call mText.TimerTask(9.84)
End Sub

Private Sub Form_Load()
    Command1.Caption = "Clique para iniciar o cronômetro "
    Text1.Text = ""
    Text2.Text = ""
    Label1.Caption = "Os 100 metros mais rápidos jamais corridos levaram este tempo:"
    Set mText = New TimerState
End Sub

Private Sub mText_ChangeText()
    Text1.Text = "Até agora"
    Text2.Text = "9,84"
End Sub

Private Sub mText_UpdateTime(ByVal dblJump As Double)
    Text2.Text = Str(Format(dblJump, "0"))
```

```
DoEvents  
End Sub
```

O código restante está no módulo de classe chamado TimerState. Incluído entre os comandos neste módulo estão as instruções **RaiseEvent**.

```
Option Explicit  
Public Event UpdateTime(ByVal dblJump As Double)  
Public Event ChangeText()  
  
Public Sub TimerTask(ByVal Duration As Double)  
    Dim dblStart As Double  
    Dim dblSecond As Double  
    Dim dblSoFar As Double  
    dblStart = Timer  
    dblSoFar = dblStart  
  
    Do While Timer < dblStart + Duration  
        If Timer - dblSoFar >= 1 Then  
            dblSoFar = dblSoFar + 1  
            RaiseEvent UpdateTime(Timer - dblStart)  
        End If  
    Loop  
  
    RaiseEvent ChangeText  
  
End Sub
```