Capítulo 3

Procedimentos (macros)

Primeiramente lemos a seguinte citação, copiada de Excel-Ajuda:

> O objetivo de uma macro é automatizar as tarefas usadas com mais freqüência. Embora algumas macros sejam simplesmente uma gravação de pressionamentos de teclas ou de cliques do mouse, macros VBA (Visual Basic for Applications (VBA):uma versão de linguagem macro do Microsoft Visual Basic usada para programar aplicativos do Microsoft Windows e incluída em vários programas da Microsoft.) mais potentes são criadas por desenvolvedores que utilizam um código capaz de executar vários comandos no computador. Por esse motivo, as macros VBA são consideradas um possível risco à segurança. Um usuário mal-intencionado poderá introduzir uma macro perigosa através de um documento que, se for aberto, permitirá que ela seja executada e possivelmente espalhe vírus (vírus: um programa de computador ou macro que "infecta" arquivos de computador inserindo cópias de si mesmo nesses arquivos. Quando o arquivo infectado é carregado na memória, o vírus pode infectar outros arquivos. Os vírus freqüentemente têm efeitos colaterais nocivos.) em seu computador.

Geralmente, denomina-se de **macro** um **procedimento** escrito em código VBA que executa certas tarefas como, por exemplo, selecionar, mover e copiar células, mudar o tipo de letra, ocultar ou apagar o conteúdo das células, etc. Como vimos no capítulo anterior, as macros automatizam tarefas e tornam mais fácil a vida do usuário. Os procedimentos são escritos ou gravados sobre um *módulo*.

O VBA permite dois tipos de procedimentos: **funções** e **sub-rotinas**.

Os **Sub**-procedimentos (macros) são muitas vezes gravados pelo gravador de macros e podem ser ativados usando um "shortcut-key". Outra maneira de executar uma macro é por meio da caixa de diálogo Macro que obtém-se por meio de *Ferramentas>Macros.*

2007: *Desenvolvedor>Gravar Macro.* Se a guia **Desenvolvedor** não for exibida, é preciso pressionar o Botão do Microsoft Office e clicar em *Opções do Excel*. Clique em *Personalizar* e, em seguida, marque a caixa de seleção *Mostrar guia Desenvovedor na Faixa de Opções*.

Os **Function**-procedimentos aumentam a biblioteca das funções intrínsecas do Excel. Uma função criada pelo usuário é usada na mesma forma como uma das 700 funções embutidas no Excel. Os procedimentos tipo "function" não podem ser gravados, devemos escrevê-los num "module sheet", ou simplesmente, num **módulo**. Uma função retorna um valor, ao passo que uma sub-rotina não.

Uma chamada de função tem o formato *Nome-da-Função (Lista-de-Parâmetros)*, onde Nome-da-Função é um nome qualquer iniciando com uma letra e Lista-de-Parâmetros é um número fixo de parâmetros que precisam ser fornecidos na ordem correta.

As funções começam com a palavra-chave **Function** e terminam com as palavras **End Function**.

Exemplo:

Para criar uma função, é necessário que exista um **módulo** onde se possa escrever o código. Seguimos os passos descritos a seguir:

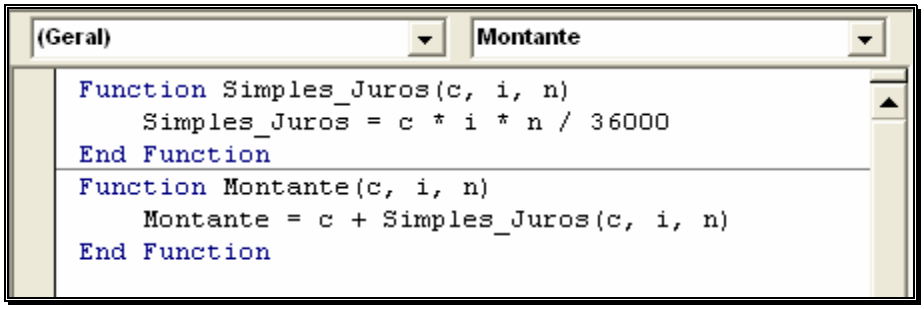
1. *Ferramentas>Macro>Editor do Visual Basic*

2. *Inserir>Módulo*

(Ou mais fácil: Alt+F11>*Inserir> Módulo>Inserir Procedimento> Função* ou *Exibir>Project Explorer>Inserir Módulo*. 2007: Alt+F11>*Inserir>Módulo*)

Nesse momento, aparece uma área em branco onde pode-se digitar o texto da função (ou das funções). No canto superior direito verá uma caixa com seta para baixo *Declaração*, e no outro canto tem a caixa *Geral*. (Uma mesma macro pode ser usada por várias planilhas, desde que não seja criada diretamente em uma planilha específica, mas sim em um **módulo**.)

Digite agora o texto das duas funções, uma após a outra – o editor insira automaticamente a linha de separação.



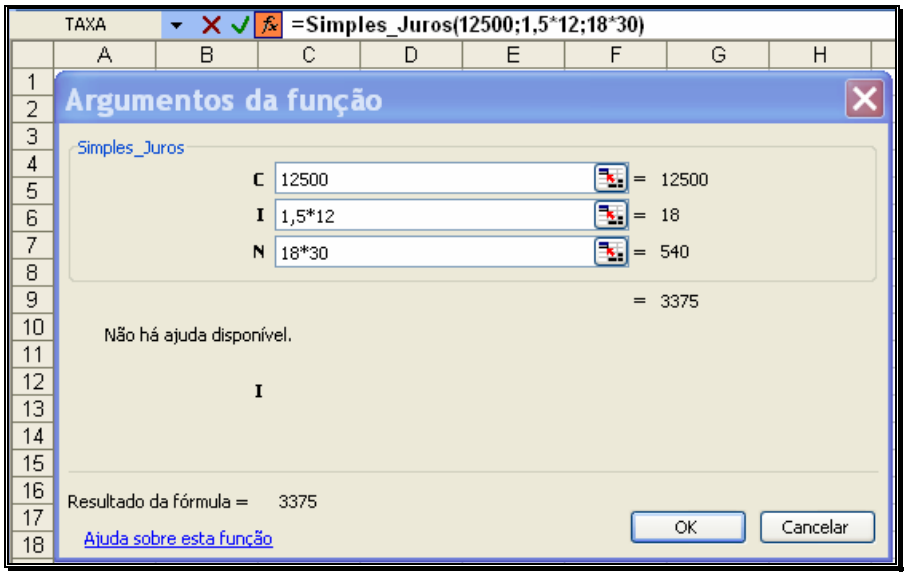
A caixa *Declaração* mostra sempre o nome da última função. A função *Montante* faz uma chamada à função *Simples_Juros* e produz com o valor, por esta determinado, a saida final da função *Montante*.

3. *Arquivo>Salvar Pasta1.xls* (Depois vem *Nome do arquivo* onde você pode escrever, p.ex., *Juros_Simples* ou outro nome significativo.) Mas, não é preciso salvar o módulo, é suficiente salvar a pasta de trabalho após terminar o seu trabalho. Em **2007**, clique no Botão do Microsoft Office: *Salvar como>Pasta de Trabalho Habilitada para Macro de Excel*.

4. Volte à janela do Excel (clicando no ícone do Excel - o primeiro botão da esquerda) e coloque o ponteiro dentro de qualquer célula.

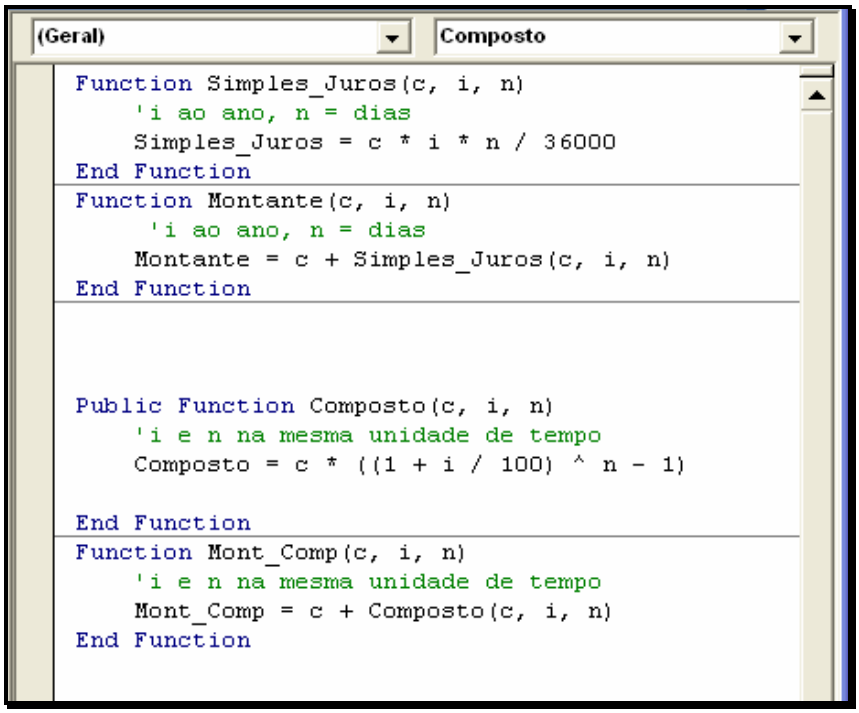
5. *Inserir>Função>Definida pelo usuário* (ou dê um clique duplo no símbolo f_x). As duas funções, por nos criadas, aparecem listadas em *Definida pelo usuário*. Pode selecionar *Simples_Juros* ou *Montante*. As duas funções calculam o Montante e os Juros simples de uma aplicação do capital **c** a **i**% ao ano durante **n** dias. (O Excel não contém embutida nenhuma função para juros simples.)

Exemplo numérico: Calcular o valor dos juros correspondentes a um empréstimo de R$ 12.500 pelo prazo de 18 meses, à taxa de 1,5% ao mês. (Nossas fórmulas querem a taxa *ao ano* e o prazo *em dias*. Por isso, temos que inserir: c = 12500, i = 1,5*12, n = 18*30)



Na figura a seguir temos adicionado também o caso dos juros compostos.

(No próximo capítulo, vamos dar mais informações sobre conceitos financeiras básicos.)



Adicionar um botão tipo formulário (forms)

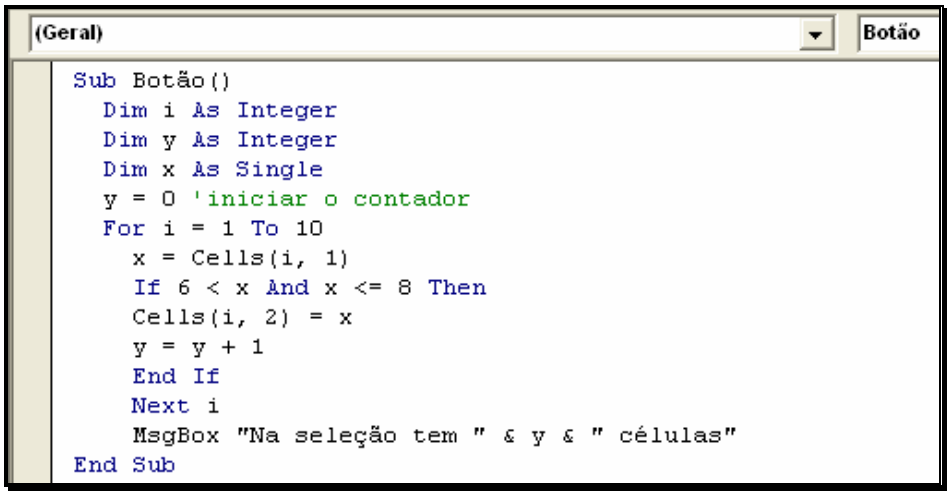
Agora vamos ver, outra vez, como se pode associar um **botão** a uma macro. No capítulo anterior, tiramos o botão da *Caixa de ferramentas de Controle* para obter um "CommandButton". Esta vez usamos o botão do menu *Formulários*.

Exemplo:

Suponha que nas células A1 a A10 (Cells(1,i) com i variando de 1 a 10) fiquem os valores x, por exemplo as notas de 10 alunos. Queremos uma macro que selecione as notas x tais que $6 < x < 8$ e que as coloque na coluna B ao lado das células em que se encontravam.

- Clique em *Exibir>Barra de ferramentas>**Formulários***
- Clique no ícone do botão e note que o cursor virou um símbolo "+". Desenhe um retângulo para o botão. Logo aparecerá a janela "Atribuir macro".
- Atribua o botão à macro com o nome "Botão".
- Agora basta clicar no botão para que a macro seja executada.

2007: *Desenvolvedor>Inserir>Controles de Formulário* etc. (Os Controles Active X os usaremos no capítulo 18.)



Recursão (Recursividade)

Recursividade é uma técnica de programação em que uma função chama a si mesma. Ela faz isso para resolver um problema menor. O algoritmo termina, quando a função pode resolver um problema menor sem ter que se chamar novamente.

A recursão geralmente é utilizada porque simplifica um problema conceitualmente.

Os exemplos clássicos do emprego de funções recursivas são o cálculo do número **fatorial** e o cálculo dos termos da seqüência de **Fibonacci**.

```
Function Fatorial (n As Integer) As Integer
     If (n=1) Then
          Fatorial = 1
     Else
          Fatorial = Fatorial(n-1) * n  'fórmula de recursão
     End If
End Function
```

Vemos que o nome da função,"Fatorial", aparece também no lado direito da fórmula de recursão, onde a função está chamando-se a ela mesma.

Isso não pode acontecer num algoritmo não recursivo, como podemos ver nas duas seguintes versões do programa.

No primeiro programa, utilizamos a estrutura `While ... Wend`, no segundo programa trabalhamos com `For ... To`

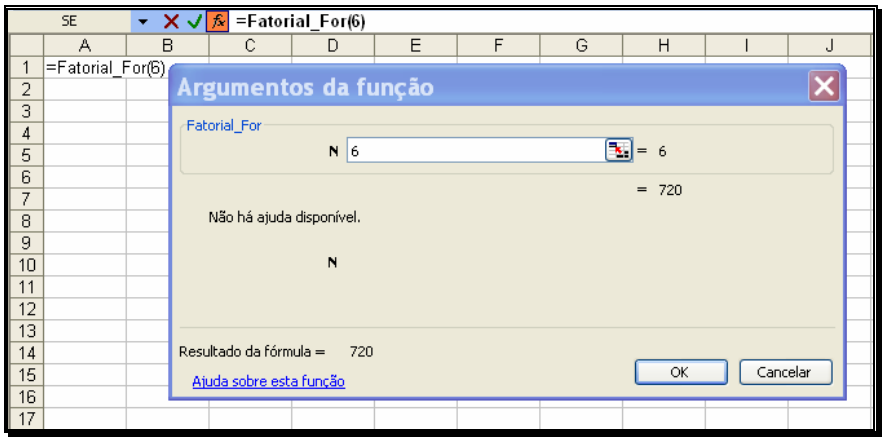
```
Function Fatorial_While (n As Integer) As Integer
Dim cont As Integer, fat As Integer
cont = 1
fat = 1
While cont < n
      fat = fat * (cont + 1)
      cont = cont +1
Wend
      Fatorial_While = fat ' valor de retorno
End Function
```

Nota-se que as variáveis "cont" e "fat" são declaradas `As Integer`. Mais à frente vamos declarar fat como Double, veja o último exemplo deste capítulo.

```
Function Fatorial_For (n As Integer) As Integer
Dim cont As Integer, fat As Integer
cont = 1
 fat = 1
For cont = 1 To n Step 1
      fat = fat * cont
Next
      Fatorial_For = fat ' valor de retorno

End Function
```

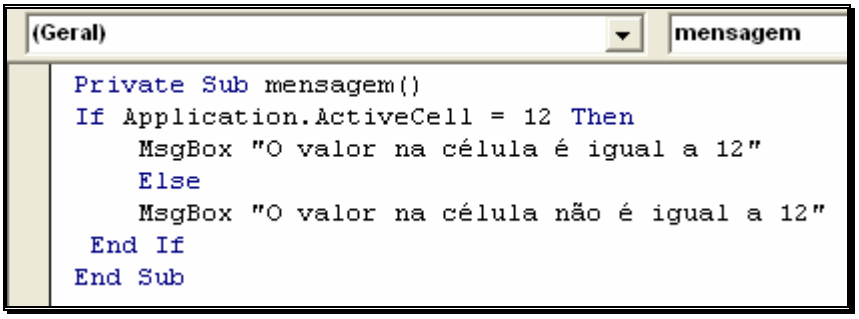
Para executar a última macro, clicamos sobre f_x. Na tela "Inserir função" selecionamos *Definida pelo usuário* e escolhemos "Fatorial_for". Em seguida vemos uma tela que pede um argumento. Com o valor N = 6 obteremos o calor 720.



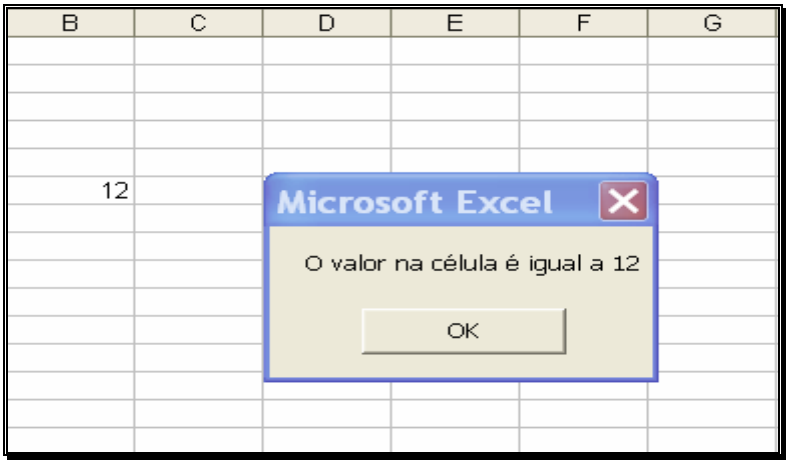
Trabalhar com as funções MsgBox e InputBox

Vou demonstrar agora o uso de uma caixa de mensagens (message box).

Utilizamos esta função para exibir mensagens em uma pequena janela. Podemos exibir texto e também valores de variáveis. Na maioria dos casos temos que exibir uma mistura de texto e números. Para lograr isso, utilizamos o operador de concatenação "&". Considere alguns exemplos:



Neste exemplo usamos a instrução condicional `IF...Then...Else`. A linha `Application.ActiveCell` refere-se a qualquer célula da planilha. Clique na planilha e insira 12 numa célula (certifique-se de que o cursor permaneça nessa célula). Faz Alt+F11 para voltar a sua janela de código e clique em Executar ou pressione F5 para executar a macro. Você verá uma caixa de mensagem informando que o valor na célula é igual a 12.

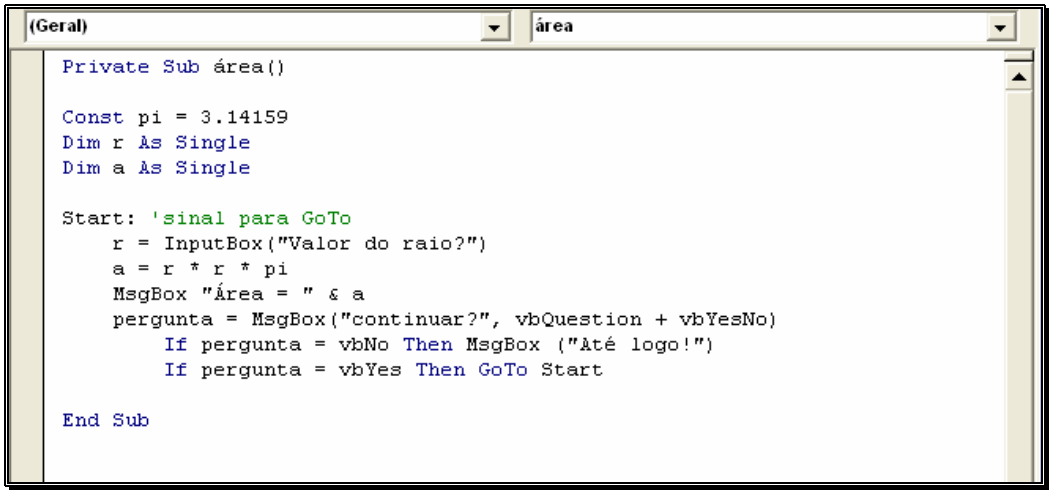


Se o valor não for 12, então verá a outra mensagem.

(O objeto "Application" representa o próprio Excel e tem 218 propriedades e métodos. Visto que a propriedade "ActiveCell" é global, podemos escrever

`ActiveCell=12` em vez de `Application.ActiveCell=12`)

O seguinte exemplo mostra uma **MsgBox-Yes/No** de diálogo:



`vbYesNo`, `vbNo`, `vbYes` são constantes. `vbYes` toma o valor 6 se o botão Yes for pressionado, `vbNo` toma o valor 7 se o botão No for pressionado. Veja também a seguinte versão

```
Private Sub área()

Const pi = 3.14159
Dim r As Single
Dim a As Single

Start: 'sinal para GoTo
    r = InputBox("Valor do raio?")
    a = r * r * pi
    MsgBox "Área = " & a
    pergunta = MsgBox("continuar?", vbQuestion + vbYesNo)

        If pergunta <> vbYes Then Exit Sub

        GoTo Start

End Sub
```

Aparece aqui a instrução `Exit Sub`, com a qual você poderá sair de uma sub-rotina ou de um "loop" (laço) usando a instrução `Exit`.

Agora utilizamos o loop `While...Wend`. Ele continuará a ser executado enquanto uma condição especificada for verdadeira. Será encerrado, assim que a condição for falsa. Em nosso caso, a execução do loop terminará quando o raio for 0.

```
Private Sub área()

Const pi = 3.14159
Dim r As Single
Dim a As Single

Start: 'rótulo para GoTo
    r = InputBox("Valor do raio? <>0")
    While r <> 0
        a = r * r * pi
        MsgBox "Área = " & a
        pergunta = MsgBox("continuar?", vbQuestion + vbYesNo)

        If pergunta <> vbYes Then Exit Sub
        GoTo Start
     Wend

End Sub
```

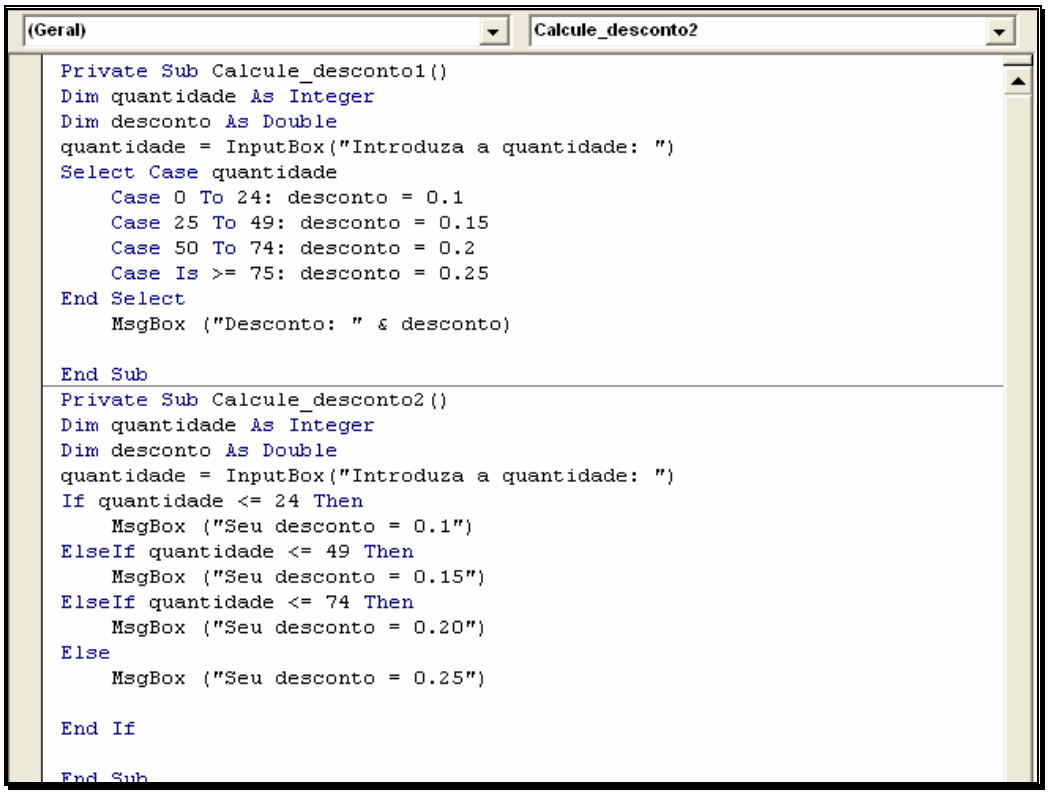
Com um `Do...Until-Loop` podemos lograr a forma mais breve e elegante. Ele continua a execução da rotina até uma condição específica ser atendida, em nosso exemplo "pergunta = vbYes". Geralmente isso significa esperar que uma variável chegue a um valor específico. Quando a condição é atendida, o loop é encerrado e o programa continua a ser executado na instrução seguinte ao loop, aqui com um MsgBox:

```
Private Sub área()

Const pi = 3.14159
Dim r As Single
Dim a As Single
 pergunta = vbYes

    Do Until pergunta = vbNo
        r = InputBox("Valor do raio?")
        a = r * r * pi
        MsgBox "Área = " & a
        pergunta = MsgBox("continuar?", vbQuestion + vbYesNo)
    Loop
        MsgBox ("Até logo!")
End Sub
```

Existe também um **InputBox** para dizer ao usuário o que deve fazer, por exemplo: InputBox("Introduza um número inteiro"). Veja o próximo exemplo.

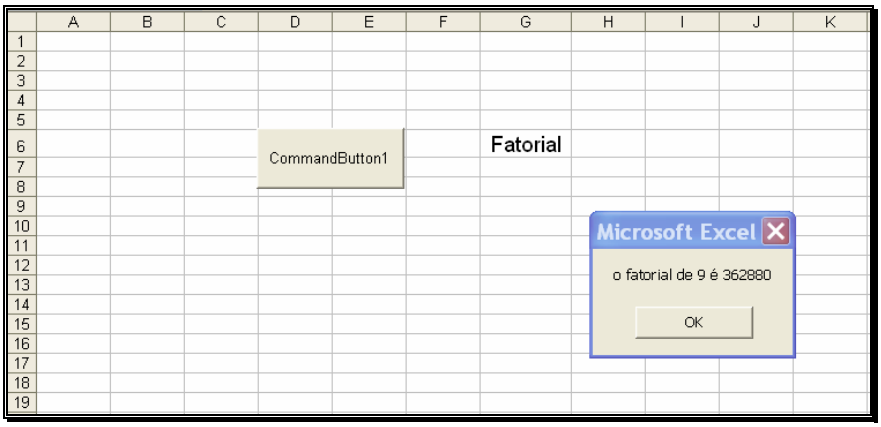


Vemos aplicada a função InputBox em duas diferentes formas num cálculo de desconto. Os códigos foram escritos sobre o mesmo formulário. Antes de aplicar uma ou outra versão, é preciso de eleger na caixa das "Declarações" a macro desejada, no caso temos "Calcule_desconto2". Ativa-se a macro com F5.

- A macro "Calcule_desconto1" foi escrita usando a instrução `Select Case`. O código é muito eficiente e faz uso só uma vez da MsgBox.
- Em "Calcule_desconto2" fazemos uso da instrução `IF...ElseIf ...Then`. Esse código funciona, mas é muito ineficiente. MsgBox foi quatro vezes utilizado! Observe que a última aplicação do MsgBox é com `Else` e não com `ElseIf`.

No exemplo a seguir, introduziremos num programa para calcular o fatorial uma caixa de mensagem junto com uma caixa para introduzir o número cujo fatorial queremos calcular.

Note que a variável **fac** é definida como número de tipo Double (15 dígitos significativos e E_{max}=308), pois o número fatorial pode facilmente superar o tamanho de um número do tipo Integer (-32,768 até 32,767). (Na linguagem Delphi, um número do tipo Integer vai de -2147483648 até 2147483647.) Quando um número é muito longo, ele automaticamente é visualizado na sua forma exponential.




```
Sub CommandButton1_Click()
Dim num As Integer
    Dim fac As Double
    Dim cont As Integer
    num = InputBox("Entre o número")
    fac = 1
    For cont = 1 To num
    fac = fac * cont
    Next
    MsgBox ("o fatorial de " & num & _
    " é " & fac)
End Sub
```

Observe que o texto no MsgBox foi escrito em duas linhas. Depois do símbolo & deve-se deixar um espaço e escrever _ .

Alguns comentários sobre a palavra-chave DIM

A sintaxe geral de uma declaração de variável é:

Dim nome da variável **As** tipo de dado

VBA permite usar DIM sem mencionar um tipo de dado: DIM altura. Neste caso, VBA tratará a variável como tendo o tipo Variant. Exemplo: Se "altura" for declarada como integer, DIM altura As Integer, ela ocuparia 2 bytes. Agora, uma variável Variant requer 16 bytes, ou seja, o não declarar a variável como integer significa um desperdício de memória de 14 bytes. Num programa grande e complexo, o desperdício de memória pode ser significativo, pois ele reduzirá também a velocidade com a qual o programa corre, resultando num desempenho pobre. Por esse motivo, é uma boa idéia declarar todas as variáveis.

Mas, se estamos desenvolvendo um programa pequeno, a declaração de todas as variáveis pode chegar a competir com o tamanho do programa próprio.

Neste caso, não se justifica tal declaração, a menos que o desenvolvedor anote os DIMs por principio.

O que faz a declaração pouco amável é o fato de que se deve declarar cada variável por separado. Na declaração DIM a,b,c,d As Integer somente d é declarado como integer, a,b,c são do tipo Variant. Por outro lado, é possível colocar mais de uma declaração em uma linha, escrevendo só uma vez DIM, por exemplo

DIM altura As Integer, Name As String, fac As Double

Claro que, desta forma, não economizamos muito espaço, mas também pouco é algo. Normalmente anotamos sempre as declarações das variáveis (por princípio), mas, às vezes, e especialmente nos últimos capítulos, deixamos as declarações para o leitor, como exercício.

Uma boa noticia é o fato de que VBA permite o uso de certos sufixos como indicadores de tipo de dados. Por exemplo, o **#** anexado na variável **a** significa que **a#** é do tipo Double (% indica integer, & longo, $ string etc.)