Delphi

1 - Aprendendo...

== Para ler este Guia... ==
Para que você possa ler o conteúdo deste guia prático, não é necessário que se tenha uma noção linguagens (no entanto seria interessante conhecer uma linguagem DOS ou Windows), pelo menos se você entender nossa parte de "Introdução à Programação". Caso você já tenha uma noção de programação, pode pular esta parte e ir para a PARTE I. O exemplo é uma boa forma de aprender sobre um comando, por isso a cada exemplo você encontrará informações sobre cada comando do mesmo.

== Aprendendo... ==

Instruções básicas

O guia consta de: Informações gerais sobre diferenças entre outras linguagens (como o não uso do sinal de igual para atribuir uma igualdade), Declarações de variáveis simples, Operadores (matemáticos...) etc. (PARTE I) e início a programação (PARTE II).

Iniciaremos com comandos muitos comuns de outras linguagens: FOR e WHILE... (ainda na introdução a programação). A PARTE II é sem dúvida uma das mais importantes, nela você se iniciará na programação Visual, fará bons programas. Procuraremos colocar os comandos Delphi em negrito. Para aprender mais sobre Delphi continue visitando este site.

== Introdução à Programação ==
Caso você já tenha uma base de programação pode pular esta parte, ou pode lê-la caso se sinta inseguro quanto ao seu conhecimento.
Bem, em primeiro lugar, o que este guia exige é uma noção de comandos básicos que todo (ou quase todo) tipo de linguagem exige.
Note que os exemplos serão baseados na programação Delphi, e não qualquer linguagem.

== Variáveis ==
Variável é um nome conhecido por qualquer programador (ou pelo menos deveria ser), este nome significa um *endereço na memória* onde um certo valor fica guardado. Vejamos um exemplo: se você declara a variável i:
`var i;` ("var" é nome usado para declarar uma variável em Delphi)
e estabelece um valor para i:
`i := 12;` este valor vai ficar "guardado na memória" para você poder usa-lo mais tarde, por exemplo, para fazer uma conta:

```
i := 4 * i;
```

Agora i vale 4 vezes 12 (48).

== Loops ==
Se você já fez algum programa em uma linguagem qualquer deve se lembrar dos famosos FOR e WHILE (loop), caso contrário, vamos a uma introdução a estes comandos:
Em uma linguagem de programação, loop é um comando que faz com que um certo código seja executado enquanto uma certa condição for atendida, um exemplo o ajudará a entender melhor:

```
var i //(declara a variável i)
for i := 1 to 30 do
ShowMessage ('O VALOR DE "I" É: ' + IntToStr(I));
```

Este comando fará com que "PARA" (FOR) I = 1, ou seja, o valor inicial de i é 1, "ATÉ" (TO) 30 (inclusive) exiba uma mensagem (SHOWMESSAGE, é o comando de exibir mensagens) escrevendo o valor de i. Portanto serão exibidas 30 mensagens. O valor de i e incrementado automaticamente ao fim do loop. Veja:
for i := 1 **to** 30 **do** (i=1) => ShowMessage(...) => (retorna a origem) **for** i := 1 **to** 30 **do** (i=2) => ShowMessage (...) =>
(...)
for i := 1 **to** 30 **do** (i=30) => ShowMessage (...) => Fim do loop.

O comando While é semelhante, veja um exemplo de seu uso:

```
var k; //(declara a variável k).
k := 0 //(define o valor de k como "0".
while k < 10 do
ShowMessage ('k é menor que 10);
```

"QUANDO" (WHILE) k é menor que 10 exibe a mensagem "k é menor que 10" (a mensagem em Delphi realmente deve estar entre apóstrofos (aspas simples) e não aspas duplas (como em muitas linguagens)). No entanto este loop não deve ser usado desta forma, pois, ao contrário de "for", while NÃO incrementa automaticamente o número, portanto o loop nunca acabará, pois k é 0 e sempre será 0 (foi definido esse valor em "k := 0"). Portanto seria melhor fazer:

```
var k; //(declara a variável k). k := 0 //(define o valor de k como "0". while k < 10 do
begin
ShowMessage ('k é menor que 10);
k := k + 1;
end;
```

Agora o valor de k será incrementado de 1 toda a vez que o loop "retornar", portanto depois de 10 mensagens o loop acabará (k vai de 0 até 9). Não se importe com o "begin" e "end" eles serão discutidos mais adiante.

== *PARTE I - Introdução a linguagem e migrando para Delphi* ==

== Variáveis ==

Talvez programadores oriundos de outras linguagens tenham certa dificuldade para se acostumar com os padrões delphi. A declaração de variáveis e seu uso, provavelmente parecerá chato para programadores Visual Basic. Vamos a algumas regras práticas:

1. Não é possível declarar variáveis em qualquer lugar, apenas no início do procedimento.
2. Os tipos de dados Delphi não são compatíveis (como no VB), por exemplo, você não pode multiplicar a **string** "X" pela **string** "Y", mas sim transformá-las em números inteiros e depois multiplica-los, veja: Z := StrToInt(X) * StrToInt(Y);.

Nem mesmo números inteiros são compatíveis com reais.

== Sinal de igual ==

Não use o sinal de igual para atribuir valores (como em grande parte das linguagens) mas sim **":="** como na linha abaixo:

```
i := 0;
```

O sinal de igual é usado apenas para comparação como em comandos "If", veja:

```
If i = 0 then
ShowMessage('O valor de i é 0');
else
ShowMessage('O valor de i não é 0';
```

Note que não se usa o *EndIf* como no caso do Visual Basic, por exemplo... Mas pode-se usar o **begin** e **end** para determinar o início e o fim como em:

```
if i = 0 then
begin
ShowMessage('O valor de i é 0');
ShowMessage('Foi usado Begin e End porque foram utilizadas duas
linhas');
end
else
ShowMessage('O valor de i não é 0';
```

Quando se usa uma única linha não há necessidade de usar **Begin** e **End**.

- IMPORTANTE -
Você deve ter notado que em Delphi não se usam aspas duplas ("), mas aspas simples ('), ou apóstrofo, como preferir. Também é usado ponto e vírgula (;) no final de cada comando (mas não nos que terminam com comandos como **If**. Mas lembre-se que NÃO se deve usar ponto e vírgula antes de **else**.

Outro motivo pelo qual no final de cada declaração usa-se o ponto e vírgula é para mostrar que acabou. Por esta razão pode-se continuar o comando na linha de baixo (menos partir strings ao meio), veja:

Estaria CORRETO:
```
MessageDlg ('Esta é uma mensagem!',
mtinformation, [mbok], 0);
```

Mas estaria INCORRETO:
```
MessageDlg ('Esta é uma
mensagem!', mtinformation, [mbok], 0);
```

Se for necessário use:
```
MessageDlg ('Esta é uma' +
+ ' mensagem!', mtinformation, [mbok], 0);
```
Isso é correto.

== Comentários ==

Os comentários podem estar entre chaves:
*{Este é um comentário que pode estar
em mais de uma linha}*;

entre sinais de parênteses e asterisco, como o mostrado abaixo:
(este é um comentário
que pode estar em mais de uma linha
até que seja colocado o sinal de asterisco e fecha parênteses *)*

Ou pode estar depois de duas barras, para comentários rápidos de apenas uma linha, veja:
// Este comentário não pode passar de uma linha...

== Operadores ==

Eles são:

```
* - Multiplicação aritmética.
/ - Divisão de números que obtém um número real (exemplo: 5/2 = 2.5).
div - Divisão de números que obtém um número inteiro (exemplo: 5/2 =
2).
+ - Adição.
- - Subtração.
= - Igual.
<> - Diferente.
> - Maior.
>= - Maior ou igual.
< - Menor.
<= - Menor ou igual.
```

Exemplo:

```
Form1.Canvas.TextOut (1, 1, IntToStr((2 * 5) div 3));
```

Form1: Nome do formulário;
Canvas: Propriedade (não apenas do formulário) relacionada com desenho. Neste caso ela é usada para escrever no formulário (com o **TextOut**), algo como o **print** do Visual Basic. Sua cor de fundo padrão é branca, mas é possível mudá-la acessando a propriedade **brush**. O primeiro parâmetro é o *left*, o segundo é o *top* e o terceiro é o texto a ser escrito.
IntToStr: Transforma um inteiro em caracteres (como já foi dito anteriormente, o Delphi não possui compatibilidade entre Integers e Strings, a não ser que sejam usadas Variants).

Iniciantes:
Para quem ainda não está integrado com programação aqui vão alguns conceitos:
Parâmetros: São "valores extras", colocados entre parênteses para fornecer outras informações ao método (function ou procedure). No exemplo mostrado, a função precisa dos parâmetros inteiros (que foram determinado como 1 neste caso) para saber a que distância da margem esquerda e o topo, respectivamente;
Left: Valor inteiro que diz a distância até o canto esquerdo de um objeto;
Top: Valor inteiro que diz a distância até o topo de um objeto;

---

2 - Dicas...

2.1. Usando funções externas (de DLLs)...

Usar funções externas, de uma ou mais DLLs é muito simples, basta declara-las da seguinte forma:

```
function NomeDaFuncao(Parâmetros: Tipo): TipoRetorno; stdcall; external 'NomeDaDLL';
```

Você pode ver o nome de uma função da DLL na Visualização Rápida do Windows (se esta estiver sido instalada), para isso encontre o arquivo DLL, clique nele com o botão direito do mouse e escolha visualização rápida.

2.2. Manipulando "Application"...

A classe TApplication proporciona muitas manipulações úteis do aplicativo como um todo. Por exemplo, caso você queira que quando um aplicativo for minimizado apareça a hora atual no título da barra de tarefas, e quando ele for restaurado não apareça, use o seguinte código no evento OnLoad do formulário, por exemplo:

```
Application.OnMinimize := ShowDate;

Application.OnRestore := NShowDate;


procedure ShowDate;

begin

TForm1.Timer1Timer(Sender);

Timer1.Enabled := True;

end;


procedure NShowDate;

begin

Timer1.Enabled := False;

Application.Title := 'Título do Aplicativo';

end;
```

Este exemplo considera que o aplicativo possua um relógio que atualiza de tempos em tempos (a cada 250 milésimos de segundo por exemplo). Este é desabilitado quando o formulário é restaurado.

2.3. Undo na caixa de texto (Memo)...
Use o seguinte código para Desfazer a última ação em uma caixa de Texto (Memo):

```
Memo1.Perform(EM_UNDO, 0, 0);
```

o valor retornado de **Memo1.Perform(EM_CANUNDO, 0, 0)** será diferente de zero se o comando de Desfazer estiver disponível.

2.4. Pegando a posição do cursor...
Use o seguinte código para ver a posição do cursor na tela, ele é muito útil quando o evento usado não possui como parâmetros "X" e "Y":

```
procedure GetPosition (var X; var Y);

var CPos: TPoint;

begin

GetCursorPos(CPos);

X := CPos.X;

Y := CPos.Y;

end;
```