

# Microsoft Small Basic

---

*Uma introdução à programação*

## Small Basic e Programação

Programação de Computadores é definida como o processo de criar aplicativos de computador utilizando linguagens de programação. Assim como nós falamos e entendemos inglês, espanhol ou francês, computadores podem entender programas escritos em certas linguagens. Essas são chamadas de linguagens de programação. No começo existiam apenas algumas linguagens de programação e elas eram muito fáceis de aprender e compreender. Mas como computadores e aplicativos se tornaram mais e mais sofisticados, linguagens de programação evoluíram rapidamente, acumulando conceitos mais complexos com o passar do tempo. Como resultado, a maioria das modernas linguagens de programação e seus conceitos são bastante difíceis de entender para um iniciante. Este fato começou a desencorajar as pessoas de aprenderem ou tentarem programar.

Small Basic é uma linguagem de programação que é desenhada para tornar a programação extremamente fácil, acessível e divertida para iniciantes. A intenção do Small Basic é derrubar a barreira e servir como trampolim para o incrível mundo da programação.

## O Ambiente do Small Basic

Vamos começar com uma rápida introdução ao ambiente do Small Basic. Quando SmallBasic.exe é lançado pela primeira vez, você verá uma janela parecida com a da seguinte figura.

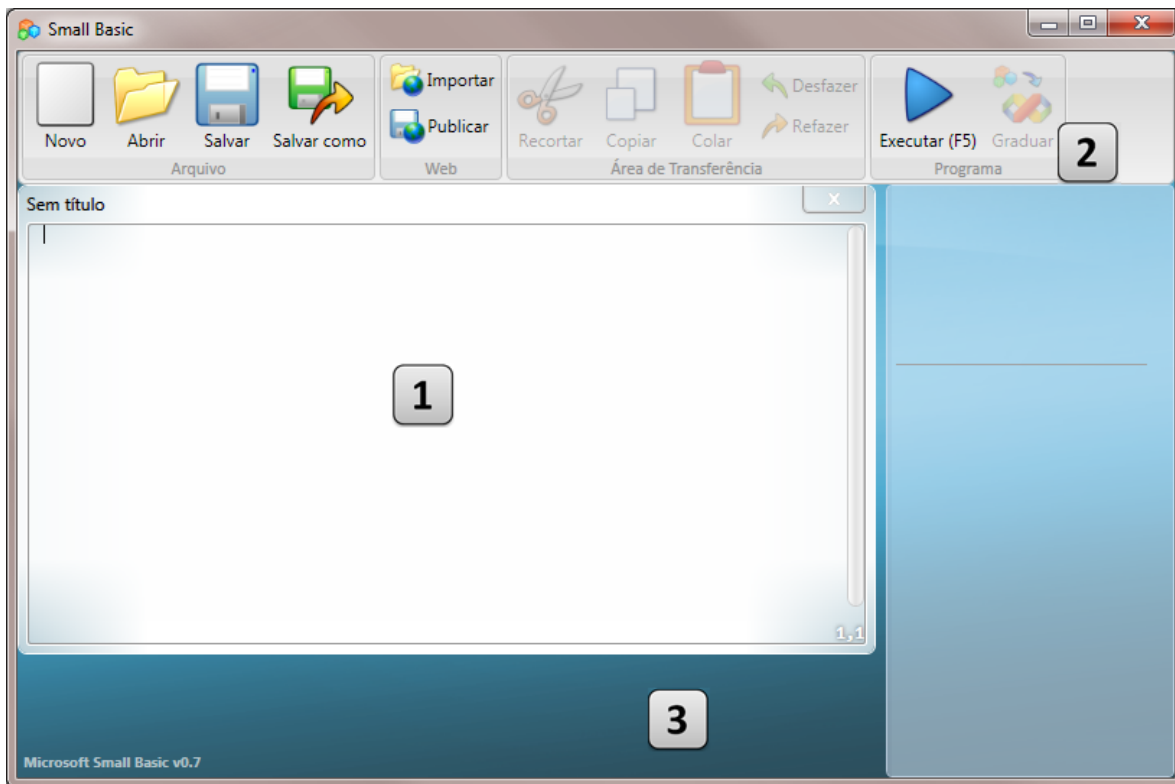


Figura 1 – O ambiente do Small Basic

Este é o ambiente do Small Basic, onde escreveremos e executaremos nossos programas Small Basic. Este ambiente tem vários elementos distintos que estão identificados por números.

O **Editor**, identificado pelo [1] é aonde escreveremos nossos programas Small Basic. Quando você abre um programa exemplo ou um programa salvo anteriormente, ele será exibido neste editor. Você pode então modificá-lo e salvá-lo para uso posterior.

Você também pode abrir e trabalhar com mais de um programa ao mesmo tempo. Cada programa com o qual você está trabalhando será exibido em um editor separado. O editor que contém o programa que você está atualmente trabalhando é chamado de *editor ativo*.

A **Barra de ferramentas**, identificada pelo [2] é usada para gerar comandos tanto para o *editor ativo* como para o ambiente. Nós aprenderemos sobre os vários comandos da barra de ferramentas conforme avançamos.

A **Superfície**, identificada pelo [3] é o lugar aonde todas as janelas do editor se encontram.

## Nosso Primeiro Programa

Agora que você está familiarizado com o ambiente do Small Basic, nós iniciaremos a programar nele. Como indicado anteriormente, o editor é o lugar aonde escrevemos nossos programas. Portanto digite a seguinte linha no editor.

```
TextWindow.WriteLine("Olá Mundo")
```

Este é o nosso primeiro programa em Small Basic. E se você digitou corretamente, você deveria ver algo similar à seguinte figura.

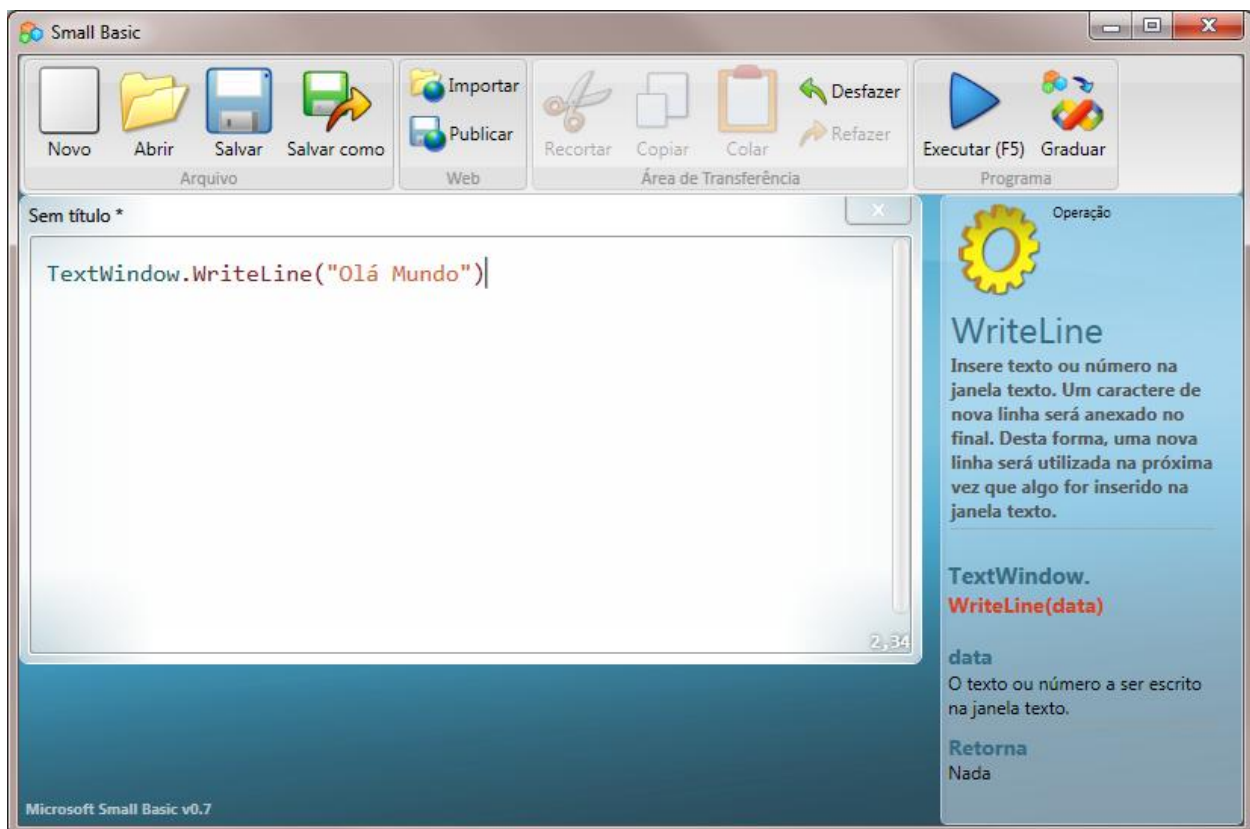


Figura 2 – Primeiro Programa

Agora que digitamos nosso novo programa, vamos executá-lo para ver o que acontece. Nós podemos executar nosso programa clicando no botão *Executar* na barra de ferramentas ou através do uso da tecla de atalho F5 no teclado. Se tudo correr bem, nosso programa deveria executar com o resultado como mostrado abaixo.

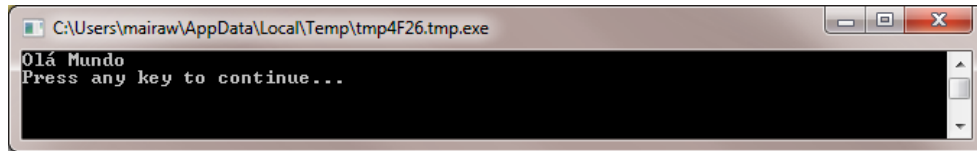


Figura 3 – Resultado do Primeiro Programa

Parabéns! Você acabou de escrever e executar seu primeiro programa em Small Basic. Um programa bem pequeno e simples, mas, no entanto, um grande passo rumo a tornar-se um verdadeiro programador! Agora, tem apenas mais um detalhe antes de criar programas maiores. Nós temos que entender o que acabou de acontecer – o que exatamente nós dissemos ao computador para fazer e como o computador sabe o que fazer? No próximo capítulo, analisaremos o programa que acabamos de escrever, para alcançar este entendimento.

*Enquanto você digitava seu primeiro programa, você pode ter observado que um menu contextual apareceu com uma lista de itens (Figura 4). Isto é chamado de “Intellisense” e lhe ajuda a digitar seu programa mais rapidamente. Você pode percorrer esta lista pressionando as teclas de direção para baixo e para cima, e quando você encontrar o que deseja, você pode pressionar a tecla Enter para inserir o item selecionado no seu programa.*

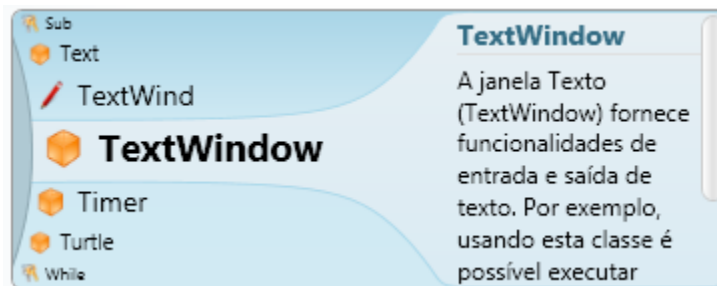


Figura 4 - Intellisense

## Salvando nosso programa

Se você deseja fechar o Small Basic e voltar mais tarde para trabalhar no programa que você acabou de digitar, você pode salvar o programa. De fato, é uma boa prática salvar os programas de tempos em tempos, para que você não perca informações caso o computador seja desligado acidentalmente ou uma falha de energia. Você pode salvar o programa atual clicando no ícone “Salvar” na barra de ferramentas ou através da tecla de atalho “Ctrl+S” (pressione a tecla S enquanto pressionando a tecla Ctrl).

## Entendendo Nosso Primeiro Programa

---

### O que realmente é um programa de computador?

Um programa é um conjunto de instruções para o computador. Essas instruções dizem ao computador exatamente o que fazer, e o computador sempre segue essas instruções. Assim como pessoas, computadores só podem seguir instruções se especificadas numa linguagem que eles possam entender. Elas são chamadas de linguagens de programação. Existem muitas linguagens que o computador pode entender e **Small Basic** é uma delas.

Imagine uma conversa entre você e seu amigo. Você e seus amigos usariam palavras, organizadas em frases para trocar informações. Da mesma forma, linguagens de programação contêm coleções de palavras que podem ser organizadas em frases para transmitir informações ao computador. E programas são basicamente conjuntos de frases (algumas vezes poucas e outras vezes milhares) que juntas fazem sentido tanto para o programador como para o computador.

*Existem várias linguagens que um computador pode entender. Java, C++, Python, VB, etc. são modernas e poderosas linguagens de programação que são usadas para criar tanto programas simples como complexos.*

### Programas Small Basic

Um programa típico em Small Basic consiste em um conjunto de *instruções*. Cada linha do programa representa uma instrução e cada instrução é um comando para o computador. Quando pedimos ao computador para executar um programa em Small Basic, ele pega o programa e lê a primeira instrução. Ele entende o que estamos tentando dizer e então executa

nossa instrução. Uma vez que ele termina a execução da primeira instrução, ele retorna ao programa e lê e executa a segunda linha. Ele continua fazendo isso até que ele alcança o fim do programa. Nosso programa então termina.

## Retomando o Nosso Primeiro Programa

Aqui está o primeiro programa que escrevemos:

```
TextWindow.WriteLine("Olá Mundo")
```

Este é um programa muito simples que consiste de uma *instrução*. Essa instrução diz ao computador para escrever uma linha de texto, que é **Olá Mundo**, na janela de texto.

Essa instrução traduz literalmente no computador como:

```
Escrever (Write) Olá Mundo
```

Você pode já ter observado que a instrução pode ser dividida em segmentos menores assim como frases podem ser divididas em palavras. A primeira instrução contém 3 segmentos distintos:

- a) TextWindow
- b) WriteLine
- c) "Olá Mundo"

O ponto, os parênteses e as aspas duplas são pontuações que devem ser colocadas nas posições corretas na instrução, para o computador entender nossa intenção.

Você pode ter notado a janela preta que apareceu quando executamos nosso primeiro programa. Esta janela preta é chamada de *TextWindow* (janela Texto) ou algumas vezes também é chamada de Console. É ali que o resultado do programa sai. **TextWindow**, no nosso programa, é chamado de *objeto*.

Existem diversos objetos disponíveis para usarmos em nossos programas. Podemos realizar diversas *operações* nesses objetos. Nós já utilizamos a operação *WriteLine* (Escrever Linha) no nosso programa. Você também pode ter notado que a operação *WriteLine* é seguida de **Olá Mundo** entre aspas. Esse texto é passado como entrada para a operação *WriteLine*, que então

*Pontuação tais como aspas duplas, espaços e parênteses são muito importantes em um programa de computador. Eles podem mudar o significado do que está sendo expresso baseado na sua posição e número.*

imprime este texto para o usuário. Isso é chamado de *entrada* para uma operação. Algumas operações aceitam um ou mais parâmetros enquanto algumas operações não aceitam nenhum.

## Nosso Segundo Programa

Agora que você entendeu nosso primeiro programa, vamos seguir adiante e adicionar algumas cores ao programa.

```
TextWindow.ForegroundColor = "Yellow"  
TextWindow.WriteLine("Olá Mundo")
```

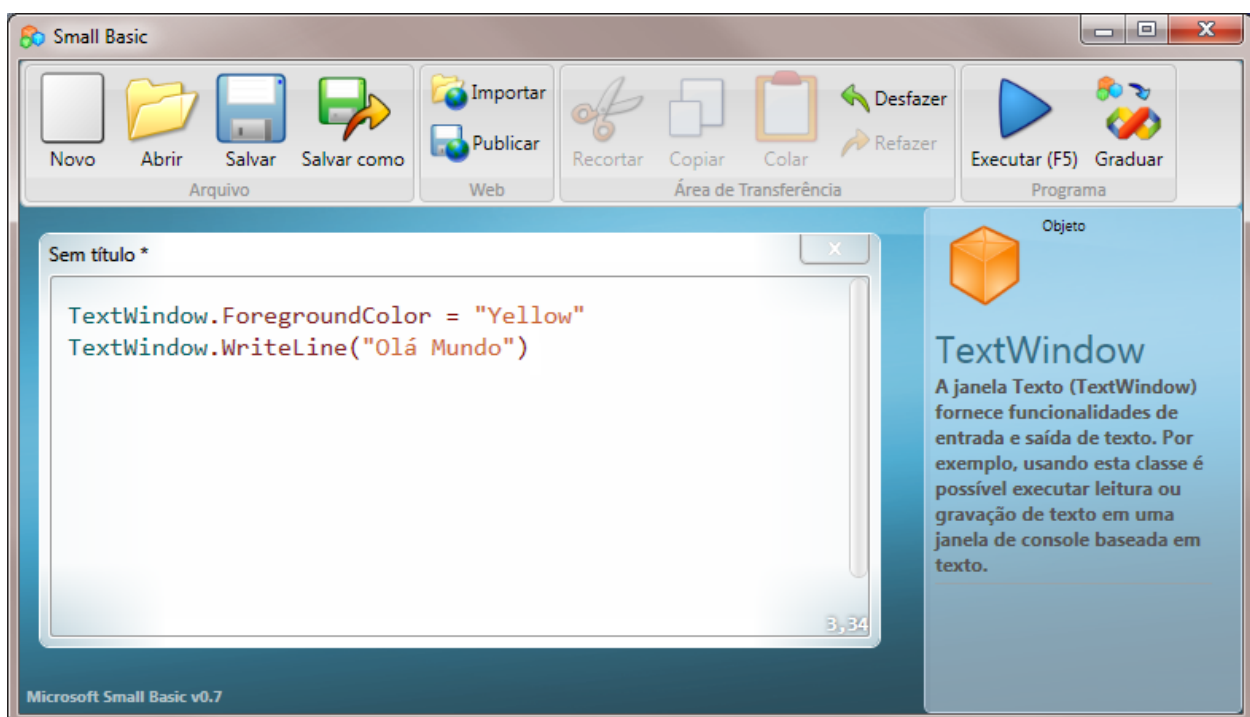


Figura 5 – Adicionando Cores

Quando você executa o programa acima, você irá notar que ele imprime a mesma frase "Olá Mundo" dentro da janela Texto (*TextWindow*), mas desta vez ele imprime o texto em amarelo (*Yellow*) ao invés de cinza como foi anteriormente.

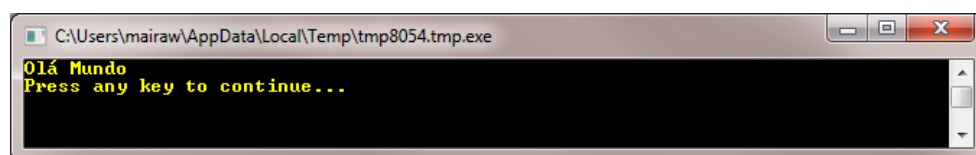


Figura 6 – Olá Mundo em Amarelo



Note a nova instrução que adicionamos ao nosso programa original. Ela utiliza uma nova palavra, *ForegroundColor* (cor de fundo) que nós igualamos ao valor de “Yellow” (amarelo). Isso significa que nós atribuímos “Yellow” a *ForegroundColor*. Agora, a diferença entre *ForegroundColor* e a operação *WriteLine* é que *ForegroundColor* não aceita nenhum parâmetro de entrada e também não necessitou de parênteses. Ao invés disso, foi seguido de um símbolo de *igual a* e uma palavra. Nós definimos *ForegroundColor* como uma *Propriedade* de *TextWindow*. Aqui está uma lista de valores que são válidos para a propriedade *ForegroundColor*. Tente substituir “Yellow” com um destes e veja os resultados – não se esqueça das aspas duplas, elas são pontuação obrigatória.

```
Black  
Blue  
Cyan  
Gray  
Green  
Magenta  
Red  
White  
Yellow  
DarkBlue  
DarkCyan  
DarkGray  
DarkGreen  
DarkMagenta  
DarkRed  
DarkYellow
```

## Introduzindo Variáveis

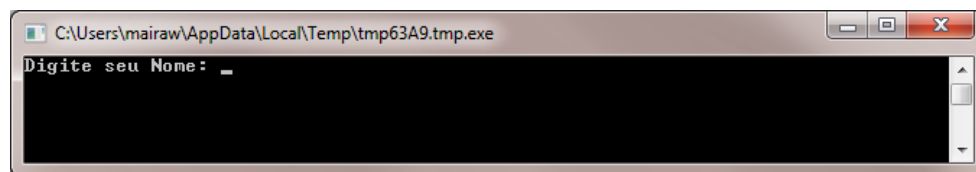
---

### Usando Variáveis no nosso programa

Não seria bacana se nosso programa pudesse dizer “Olá” com o nome dos usuários ao invés de dizer o genérico “Olá Mundo”? Para fazer isso, nós precisamos primeiro perguntar ao usuário o seu nome e então armazená-lo em algum lugar e então imprimir “Olá” com o nome do usuário. Vejamos como podemos fazer isso:

```
TextWindow.Write("Digite seu Nome: ")  
nome = TextWindow.Read()  
TextWindow.WriteLine("Olá " + nome)
```

Quando você digita e executa seu programa, você verá uma saída parecida com a saída a seguir:



**Figura 7 – Pergunte o nome do usuário**

E quando você digita seu nome e pressiona a tecla ENTER, você verá a seguinte saída:

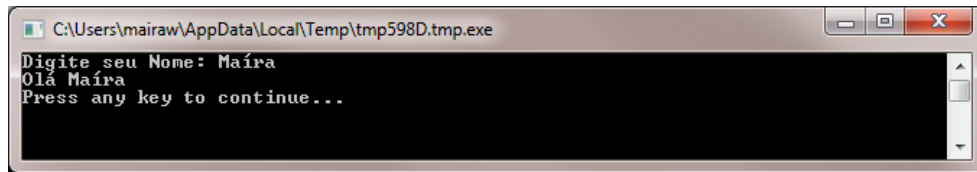


Figura 8 – Um Olá Caloroso

Agora, se você executar o programa novamente, a mesma pergunta será feita novamente. Você pode digitar um nome diferente e o computador irá dizer Olá com aquele nome.

## Análise do programa

No programa que você acabou de executar, a linha que pode ter chamado sua atenção é esta:

```
nome = TextWindow.Read()
```

*Read()* se parece com *WriteLine()*, porém sem entradas. *Read* (ler) é uma operação e basicamente diz ao computador para esperar que o usuário digite algo e pressione a tecla ENTER. Uma vez que o usuário pressiona a tecla ENTER, ele pega o que o usuário digitou e o retorna para o programa. O ponto interessante é que tudo aquilo que o usuário digitou agora está armazenado numa *variável* chamada **nome**. Uma *variável* é definida como um lugar onde você pode armazenar valores temporariamente e usá-los mais tarde. Na linha acima, **nome** foi usado para armazenar o nome do usuário.

*Write, assim como WriteLine é outra operação em TextWindow. Write permite que você escreva algo na TextWindow mas permite que texto sucessor seja escrito na mesma linha que o texto atual.*

A próxima linha também é interessante:

```
TextWindow.WriteLine("Olá " + nome)
```

Este é o lugar onde nós usamos o valor armazenado na nossa variável, **nome**. Nós pegamos o valor em **nome** e o anexamos ao fim de "Olá" e escrevemos isso na TextWindow.

Uma vez que uma variável é definida, você pode reusá-la inúmeras vezes. Por exemplo, você pode fazer o seguinte:

```
TextWindow.Write("Digite seu Nome: ")
nome = TextWindow.Read()
TextWindow.Write("Olá " + nome + ". ")
```

```
TextWindow.WriteLine("Como você está " + nome + "?")
```

E você verá a seguinte saída:

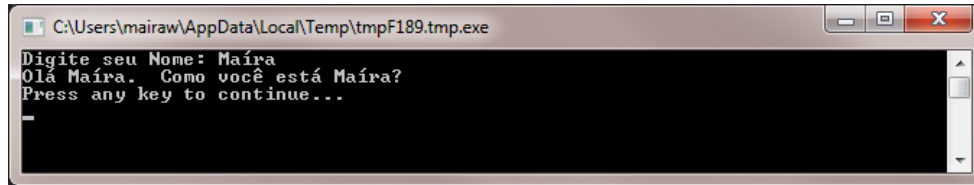


Figura 9 – Reusando uma Variável

## Regras para nomear Variáveis

Variáveis têm nomes associados a elas e é assim que você as identifica. Existem algumas regras simples e diretrizes muito boas para nomear essas variáveis. Elas são:

1. O nome deve começar com uma letra e não deve colidir com nenhuma outra palavra-chave como por exemplo **if**, **for**, **then**, etc.
2. Um nome pode conter qualquer combinação de letras, dígitos e sublinhados (\_).
3. É útil nomear variáveis de forma significativa – já que variáveis podem ser tão compridas quanto você desejar, use nomes de variáveis para descrever sua intenção.

## Brincando com Números

Nós acabamos de ver como você pode usar variáveis para armazenar o nome do usuário. Nos programas a seguir, nós veremos como podemos armazenar e manipular números em variáveis. Vamos começar com um programa bem simples:

```
numero1 = 10  
numero2 = 20  
numero3 = numero1 + numero2  
TextWindow.WriteLine(numero3)
```

Quando você executar este programa, você obterá a seguinte saída:

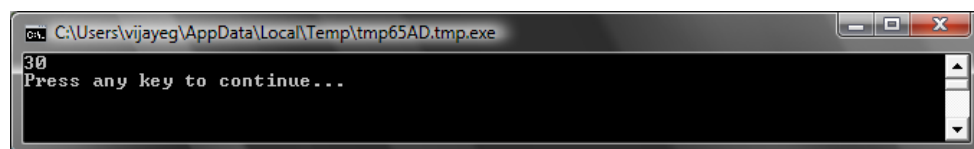


Figura 10 – Somando Dois Números

Na primeira linha do programa, você está atribuindo à variável **numero1** o valor 10. E na segunda linha, você está atribuindo à variável **numero2** o valor 20. Na terceira linha, você está somando **numero1** e **numero2** e então atribuindo o resultado da soma à variável **numero3**. Portanto, neste caso, **numero3** terá o valor 30. E isso é o que nós imprimimos em TextWindow.

*Note que os números não estão entre aspas duplas. Para números, aspas não são necessárias. Você somente necessita de aspas quando você está usando texto.*

Agora, vamos modificar ligeiramente esse programa e ver os resultados:

```
numero1 = 10  
numero2 = 20  
numero3 = numero1 * numero2  
TextWindow.WriteLine(numero3)
```

O programa acima multiplicará **numero1** por **numero2** e armazenará o resultado em **numero3**. E você pode ver o resultado deste programa abaixo:

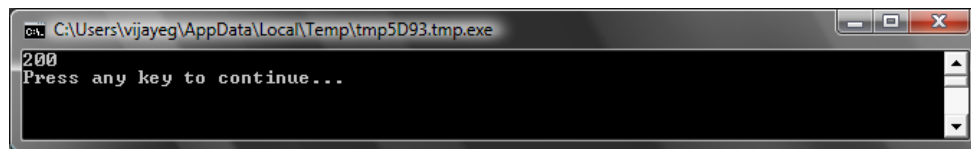


Figura 11 – Multiplicando Dois Números

Da mesma forma, você pode subtrair ou dividir números. Aqui está a subtração:

```
numero3 = numero1 - numero2
```

E o símbolo para a divisão é '/'. E o programa se parecerá com isso:

```
numero3 = numero1 / numero2
```

E o resultado da divisão seria:

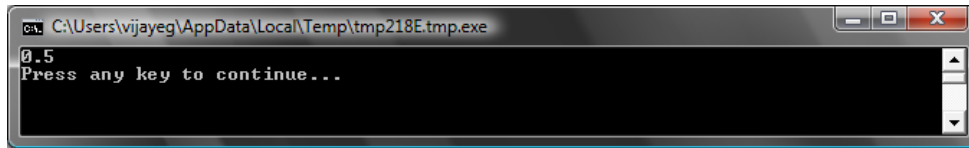


Figura 12 – Dividindo Dois Números

## Um Simples Conversor de Temperatura

Para o próximo programa, nós usaremos a fórmula  $^{\circ}\text{C} = \frac{5(^{\circ}\text{F}-32)}{9}$  para converter temperaturas em Fahrenheit para temperaturas em Célsius.

Primeiro, nós obteremos a temperatura em Fahrenheit do usuário e a armazenaremos em uma variável. Existe uma operação que nos deixa ler números digitados pelo usuário que é **TextWindow.ReadNumber**.

```
TextWindow.Write("Digite a temperatura em Fahrenheit: ")  
fahr = TextWindow.ReadNumber()
```

Uma vez que temos a temperatura em Fahrenheit armazenada em uma variável, nós podemos convertê-la para Célsius assim:

```
celsius = 5 * (fahr - 32) / 9
```

Os parênteses dizem ao computador para calcular a primeira parte **fahr – 32** e então processar o resto. Agora tudo que temos que fazer é imprimir o resultado para o usuário. Juntando tudo isto, nós temos este programa:

```
TextWindow.Write("Digite a temperatura em Fahrenheit: ")  
fahr = TextWindow.ReadNumber()  
celsius = 5 * (fahr - 32) / 9  
TextWindow.WriteLine("Temperatura em Célsius é " + celsius)
```

E o resultado do programa seria:

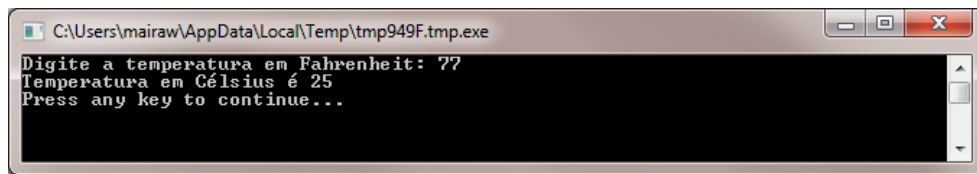


Figura 13 – Conversão de Temperatura

## Condições e Ramificação

---

Voltando ao nosso primeiro programa, não seria legal que ao invés de dizer o genérico *Olá Mundo*, nós pudéssemos dizer *Bom Dia Mundo*, ou *Boa Tarde Mundo* dependendo da hora do dia? Para o nosso próximo programa, nós faremos o computador dizer *Bom Dia Mundo* se o horário for antes do meio-dia; e *Boa Tarde* se o horário for após o meio-dia.

```
If (Clock.Hour < 12) Then
    TextWindow.WriteLine("Bom Dia Mundo")
EndIf
If (Clock.Hour >= 12) Then
    TextWindow.WriteLine("Boa Tarde Mundo")
EndIf
```

Dependendo de quando você executa o programa, você verá uma das seguintes saídas:

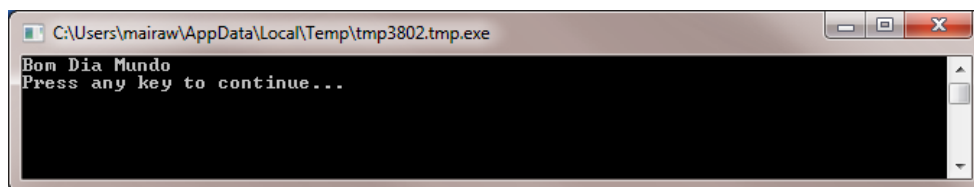


Figura 14 – Bom Dia Mundo



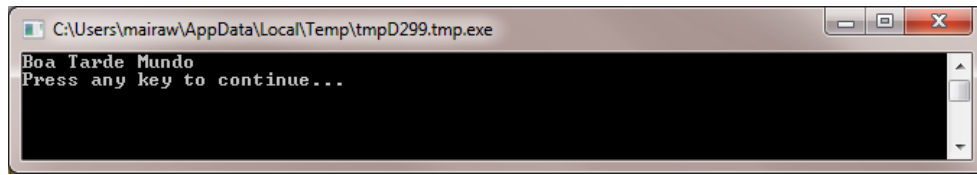


Figura 15 – Boa Tarde Mundo

Vamos analisar as três primeiras linhas do programa. Essas linhas dizem ao computador que se `Clock.Hour` é menor que 12, imprima “Bom Dia Mundo.” As palavras **If**, **Then** e **EndIf** são palavras especiais que são entendidas pelo computador quando o programa executa. A palavra **If** (se) é sempre seguida de uma condição, que neste caso é **(Clock.Hour < 12)**. Lembre que os parênteses são necessários para o computador entender suas intenções. A condição é seguida da palavra **Then** (então) e a real operação para executar. E depois da operação vem a palavra **EndIf** (final da condição if). Ela diz ao computador que a execução condicional terminou.

*Em Small Basic, você pode usar o objeto Clock para acessar a data e hora atuais. Ele também fornece a você uma série de propriedades que permitem que você obtenha o Dia, Mês, Ano, Hora, Minutos e Segundos atuais separadamente.*

Entre a palavra **Then** e a palavra **EndIf**, poderiam existir mais de uma operação e o computador irá executar todas se a condição for válida. Por exemplo, você poderia escrever algo similar a isto:

```
If (Clock.Hour < 12) Then
    TextWindow.Write("Bom Dia. ")
    TextWindow.WriteLine("Como estava o café da manhã?")
EndIf
```

## Else

No programa do início do capítulo, você pode ter notado que a segunda condição é um pouco redundante. O valor **Clock.Hour** poderia ser maior que 12 ou não. Nós não precisaríamos realmente fazer a segunda checagem. Nessas situações, nós podemos encurtar as duas instruções **if..then..endif** para ser apenas uma usando uma nova palavra, **Else** (*caso contrário*).

Se tivéssemos que escrever o programa usando **Else**, ele seria parecido com este:

```
If (Clock.Hour < 12) Then
    TextWindow.WriteLine("Bom Dia Mundo")
Else
```

```
TextWindow.WriteLine("Boa Tarde Mundo")
EndIf
```

E esse programa fará exatamente o mesmo que o outro, o que nos leva para uma importante lição em programação de computadores:

*“ Em programação, existem diversas maneiras de fazer a mesma coisa. Às vezes, um jeito faz mais sentido que o outro. A escolha fica a cargo do programador. Conforme você for escrevendo mais programas e ficando mais experiente, você começará a notar as diferentes técnicas e suas vantagens e desvantagens.*

## Recuo

Em todos os exemplos você pode ver que as instruções entre *If*, *Else* e *EndIf* estão recuadas. Este recuo não é necessário. O computador irá entender o programa normalmente sem o recuo. Entretanto, o recuo nos ajuda a ver e entender a estrutura do programa mais facilmente. Portanto, geralmente é considerado boa prática recuar as instruções entre este tipo de bloco de instruções.

## Par ou Ímpar

Agora que nós temos a instrução **If..Then..Else..EndIf** na nossa mala de truques, vamos escrever um programa que, dado um número, o programa dirá se o número é par ou ímpar.

```
TextWindow.Write("Digite um número: ")
num = TextWindow.ReadNumber()
resto = Math.Remainder(num, 2)
If (resto = 0) Then
    TextWindow.WriteLine("O número é Par ")
Else
    TextWindow.WriteLine("O número é Ímpar ")
EndIf
```

E quando você executa esse programa, você verá uma saída como esta:

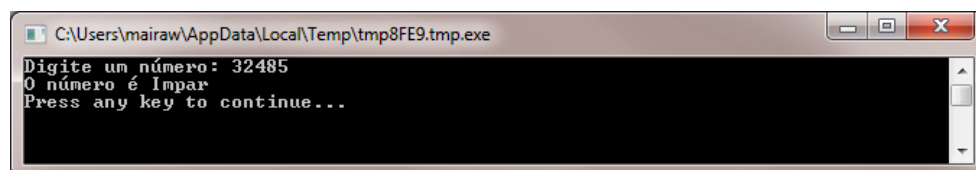


Figura 16 – Par ou Ímpar

Nesse programa, nós introduzimos outra poderosa operação, **Math.Remainder** (resto da divisão). **Math.Remainder**, como você pode imaginar, irá dividir o primeiro número pelo segundo número e então retornar o resto da divisão.

## Ramificação

Lembre-se, no segundo capítulo você aprendeu que o computador processa um programa uma instrução por vez, em ordem do início ao fim. Entretanto, existe uma instrução especial que pode fazer o computador pular para outra instrução fora de ordem. Vamos examinar o próximo programa.

```
i = 1
inicio:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
    Goto inicio
EndIf
```

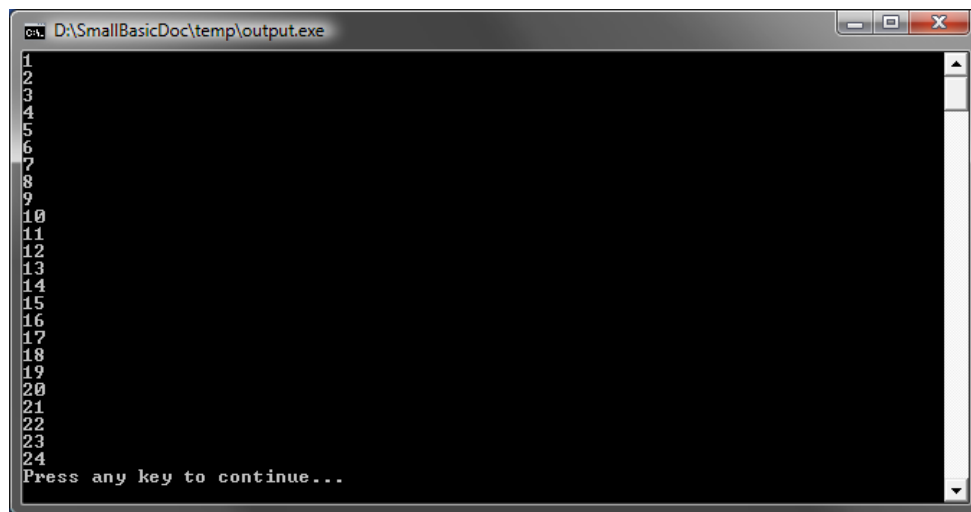


Figura 17 - Usando Goto

No programa acima, nós atribuímos o valor 1 à variável **i**. E então nós adicionamos uma nova instrução que termina em dois-pontos (:)

```
inicio:
```

Isto é chamado de *rótulo*. Rótulos são como indicadores que o computador entende. Você poder dar o nome que você quiser para o indicador e você pode adicionar quantos rótulos você quiser no programa, desde que todos eles tenham nomes únicos.

Outra instrução interessante é esta aqui:

```
i = i + 1
```

Isso apenas diz ao computador para adicionar 1 à variável **i** e atribuir o resultado de volta a **i**. Portanto, se o valor de **i** era 1 antes dessa instrução, ele será 2 depois que essa instrução é executada.

E finalmente,

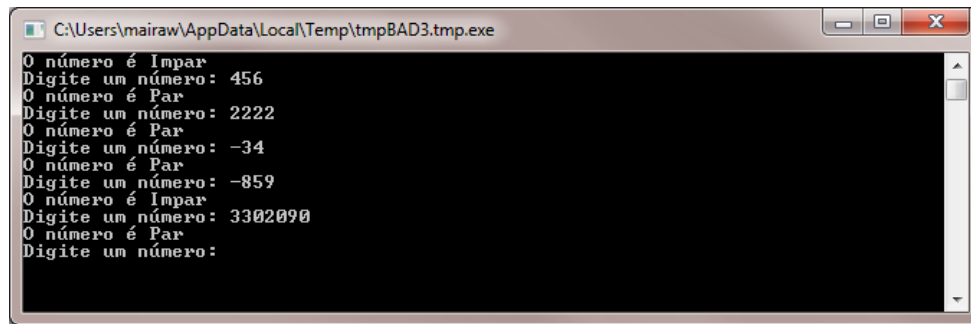
```
If (i < 25) Then  
    Goto inicio  
EndIf
```

Essa é a parte que diz ao computador que se o valor de **i** for menor que 25, o computador deve começar a processar as instruções a partir do indicador **inicio**.

## Execução sem fim

Usando a instrução **Goto**, você pode fazer o computador repetir algo qualquer número de vezes. Por exemplo, você pode pegar o programa Par ou Ímpar e modificá-lo como no exemplo abaixo, e o programa irá executar sem parar. Você pode parar o programa, clicando no botão Fechar (X) no canto superior direito da janela.

```
inicio:  
TextWindow.Write("Digite um número: ")  
num = TextWindow.ReadNumber()  
resto = Math.Remainder(num, 2)  
If (resto = 0) Then  
    TextWindow.WriteLine("O número é Par ")  
Else  
    TextWindow.WriteLine("O número é Ímpar ")  
EndIf  
Goto inicio
```



```
O número é Impar
Digite um número: 456
O número é Par
Digite um número: 2222
O número é Par
Digite um número: -34
O número é Par
Digite um número: -859
O número é Impar
Digite um número: 3302090
O número é Par
Digite um número:
```

Figura 18 – Par ou Ímpar executando sem parar

## Loop For

Vamos examinar o programa que escrevemos no capítulo anterior.

```
i = 1
inicio:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
    Goto inicio
EndIf
```

Este programa imprime números de 1 a 24 em ordem. Este processo de incrementar uma variável é muito comum na programação que linguagens de programação geralmente fornecem um método mais fácil de fazer isto. O programa acima é equivalente ao programa abaixo:

```
For i = 1 To 24
    TextWindow.WriteLine(i)
EndFor
```

E o resultado é:

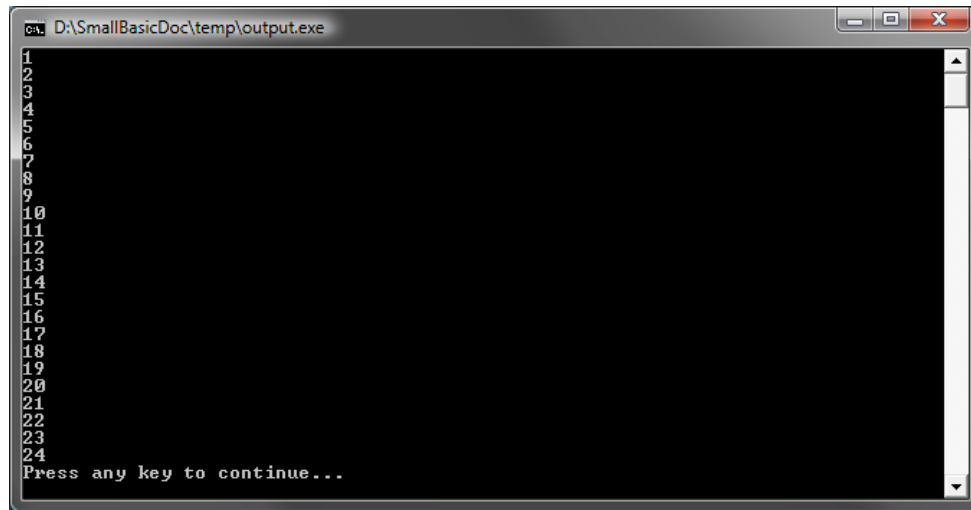


Figura 19 – Usando o Loop For

Note que reduzimos o programa de 8 linhas para um programa de 4 linhas, e ele ainda faz exatamente o mesmo que o programa de 8 linhas! Lembra que antes dissemos que geralmente existem várias maneiras de fazer a mesma coisa? Este é um ótimo exemplo.

**For..EndFor** é, em termos de programação, chamado de *loop*. Ele permite que você pegue uma variável, dê um valor inicial e final e deixe que o computador incremente a variável para você. Toda vez que o computador incrementa a variável, ele roda os comandos entre **For** e **EndFor**.

Mas se você quisesse que a variável fosse incrementada de 2 em 2 ao invés de 1 em 1 – por exemplo, se você quiser imprimir todos os números ímpares entre 1 e 24, você pode usar o loop para fazer isso também.

```
For i = 1 To 24 Step 2
    TextWindow.WriteLine(i)
EndFor
```



Figura 20 – Somente Números Ímpares

A parte **Step 2** do comando **For** diz ao computador para incrementar o valor de **i** em 2 ao invés do usual 1. Ao utilizar **Step** você pode especificar qualquer incremento que você quiser. Você pode até especificar um valor negativo para o passo e fazer o computador contar de trás para frente, como no exemplo abaixo:

```
For i = 10 To 1 Step -1
    TextWindow.WriteLine(i)
EndFor
```



Figura 21 – Contando de Trás pra Frente

## Loop While

O loop **While** (enquanto) é outro método de looping, que é especialmente útil quando o contador de loop não é conhecido de antemão. Enquanto um loop **For** é executado um número pré-determinado de vezes, o loop **While** é executado até que uma dada condição seja verdadeira. No exemplo abaixo, nós estamos dividindo um número ao meio enquanto o resultado é maior que 1.

```
numero = 100
While (numero > 1)
    TextWindow.WriteLine(numero)
    numero = numero / 2
EndWhile
```

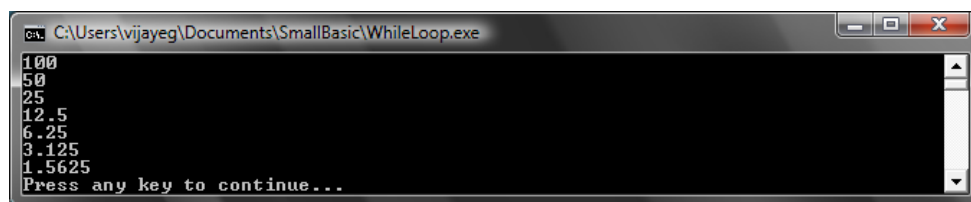


Figura 22 – Loop de Divisão ao Meio



No programa acima, nós atribuímos o valor 100 a *numero* e executamos o loop **While** enquanto o número for que 1. Dentro do loop, nós imprimimos o número e então o dividimos por dois, efetivamente dividindo-o ao meio. E como esperado, a saída do programa é números que são progressivamente divididos ao meio um após o outro.

Seria muito difícil escrever este programa usando um loop **For**, porque nós não sabemos quantas vezes o loop irá executar. Com um loop **While** é fácil checar a condição e pedir ao computador que ou continue o loop ou pare.

É interessante observar que todo loop **While** pode ser desembrilhado num comando **If..Then**. Por exemplo, o programa abaixo pode ser rescrito como segue, sem afetar o resultado final.

```
numero = 100
inicio:
TextWindow.WriteLine(numero)
numero = numero / 2

If (numero > 1) Then
    Goto inicio
EndIf
```

*Na verdade, o computador internamente reescreve todo loop While em comandos que usam If..Then com um ou mais comandos Goto.*

## Introdução a Gráficos

---

Até agora nos nossos exemplos, nós temos usado a `TextWindow` para explicar os conceitos básicos da linguagem Small Basic. Entretanto, Small Basic vem com um conjunto poderoso de recursos gráficos que nós começaremos a explorar neste capítulo.

### Apresentando a `GraphicsWindow`

Assim como nós temos a `TextWindow` (janela Texto) que nos permite trabalhar com texto e números, Small Basic também fornece uma **`GraphicsWindow`** (janela Gráficos) que podemos usar para desenhar coisas. Vamos começar mostrando a `GraphicsWindow`.

```
GraphicsWindow.Show()
```

Quando você executar este programa, você irá notar que ao invés da usual janela de texto preta, você obterá uma janela branca como a mostrada abaixo. Ainda não tem muita coisa para fazer nesta janela. Mas esta será a janela base na qual iremos trabalhar neste capítulo. Você pode fechar esta janela clicando no botão 'X' no canto superior direito.

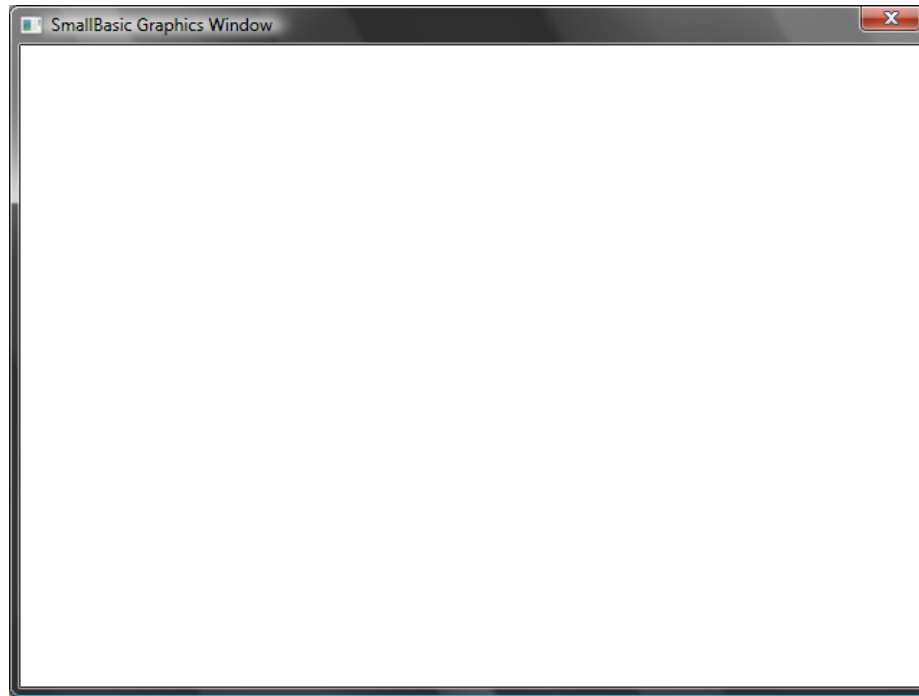


Figura 23 – Uma janela Gráficos vazia

## Configurando a janela Gráficos

A janela Gráficos permite que você personalize sua aparência como você desejar. Você pode alterar o título, a cor de fundo e seu tamanho. Vamos seguir adiante e modificá-la um pouco, apenas para se familiarizar com a janela.

```
GraphicsWindow.BackgroundColor = "SteelBlue"  
GraphicsWindow.Title = "Minha Janela Gráficos"  
GraphicsWindow.Width = 320  
GraphicsWindow.Height = 200  
GraphicsWindow.Show()
```

Aqui está como a janela Gráficos personalizada se parece. Você pode alterar a cor de fundo por um dos diferentes valores listados no Apêndice B. Brinque com essas propriedades para ver como você pode modificar a aparência da janela.

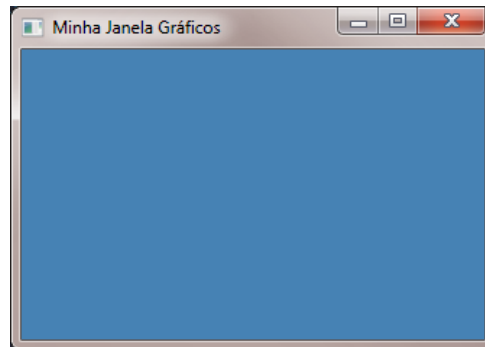


Figura 24 – Uma janela Gráficos Personalizada

## Desenhando Linhas

Uma vez que temos a *GraphicsWindow* ativa, nós podemos desenhar figuras, texto e até imagens nela. Vamos começar desenhando algumas figuras simples. Aqui está um programa que desenha algumas linhas na janela Gráficos.

```
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200  
GraphicsWindow.DrawLine(10, 10, 100, 100)  
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

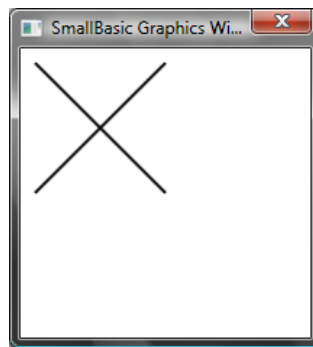


Figura 25 – Linhas Cruzadas

As primeiras duas linhas do programa configuram a janela e as duas próximas linhas desenharam as linhas cruzadas. Os primeiros dois números que seguem *DrawLine* (desenhar

*Em vez de usar nomes para cores, você pode usar a notação de cores da web (#RRGGBB). Por exemplo, #FF0000 designa o vermelho, #FFFF00 o amarelo, e assim por diante. Nós aprenderemos mais sobre cores no capítulo Cores.*

linha) especificam as coordenadas iniciais x e y e os outros dois números especificam as coordenadas finais x e y. É interessante observar que com computadores as coordenadas (0,0) iniciam no canto superior esquerdo da janela. Na verdade, no sistema de coordenadas a janela é considerada estando no 2º quadrante.

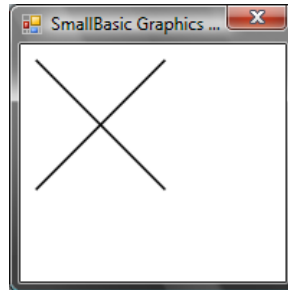


Figura 26 – O mapa de coordenadas

Se voltarmos ao programa da linha, é interessante notar que o Small Basic permite que você modifique as propriedades da linha, tais como a cor e sua espessura. Primeiro, vamos modificar as cores das linhas como mostrado no programa abaixo.

```
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200  
GraphicsWindow.PenColor = "Green"  
GraphicsWindow.DrawLine(10, 10, 100, 100)  
GraphicsWindow.PenColor = "Gold"  
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

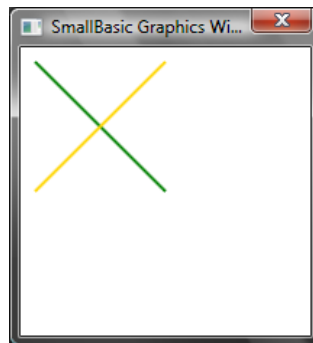


Figura 27 – Alterando a Cor da Linha

Agora, vamos modificar o tamanho também. No programa abaixo, nós alteramos a largura da linha para 10, ao invés do valor padrão que é 1.

```
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200
```

```
GraphicsWindow.PenWidth = 10  
GraphicsWindow.PenColor = "Green"  
GraphicsWindow.DrawLine(10, 10, 100, 100)  
GraphicsWindow.PenColor = "Gold"  
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

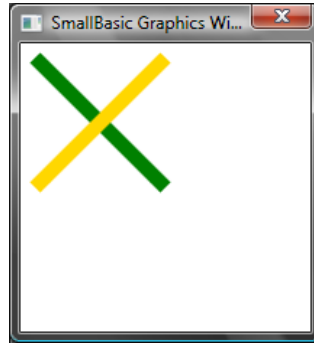


Figura 28 – Linhas Grossas e Coloridas

*PenWidth* (largura da caneta) e *PenColor* (cor da caneta) modificam a caneta com que essas linhas são desenhadas. Eles não apenas afetam linhas mas qualquer figura que seja desenhada depois que essas propriedades foram atualizadas.

Através do uso das instruções de loop que nós aprendemos nos capítulos anteriores, nós podemos facilmente escrever um programa que desenha múltiplas linhas aumentando a grossura da caneta.

```
GraphicsWindow.BackgroundColor = "Black"  
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 160  
GraphicsWindow.PenColor = "Blue"  
  
For i = 1 To 10  
    GraphicsWindow.PenWidth = i  
    GraphicsWindow.DrawLine(20, i * 15, 180, i * 15)  
endfor
```

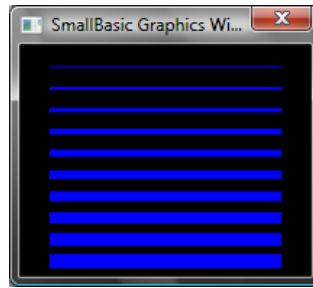


Figura 29 – Múltiplas Larguras de Caneta

A parte interessante deste programa é o loop, onde nós incrementamos *PenWidth* toda vez que o loop é executado e então desenhamos uma nova linha abaixo da anterior.

## Desenhando e Preenchendo Figuras

Quando se trata de desenhar figuras, geralmente existem dois tipos de operações para cada figura. Elas são as operações *Draw* para desenho e *Fill* para preenchimento. Operações *Draw* desenham o contorno da figura usando uma caneta, e operações *Fill* pintam a figura usando um pincel. Por exemplo, no programa abaixo, existem dois retângulos, um que é desenhado usando uma caneta vermelha (*Red*) e outro que é preenchido usando o pincel verde (*Green*).

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawRectangle(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillRectangle(60, 100, 300, 60)
```

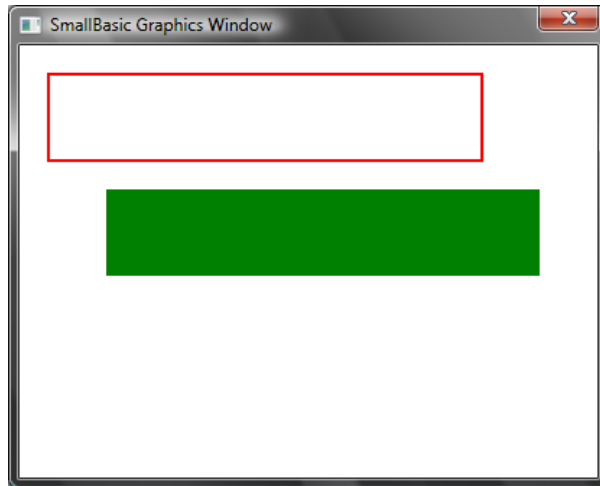


Figura 30 – Desenhando e Preenchendo

Para desenhar ou preencher um retângulo, você precisa de quatro números. Os dois primeiros números representam as coordenadas X e Y para o canto superior esquerdo do retângulo. O terceiro número especifica a largura do retângulo enquanto o quarto especifica sua altura. Na verdade, o mesmo se aplica para desenhar e preencher elipses, como mostrado no programa abaixo.

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(60, 100, 300, 60)
```



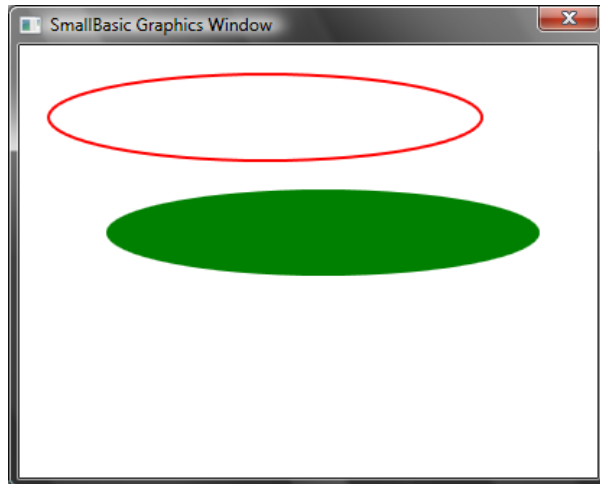


Figura 31 – Desenhando e Preenchendo Elipses

Círculos são apenas um tipo especial de elipses. Se você deseja desenhar círculos, você teria que especificar a mesma largura e altura para a elipse.

```
GraphicsWindow.Width = 400  
GraphicsWindow.Height = 300  
  
GraphicsWindow.PenColor = "Red"  
GraphicsWindow.DrawEllipse(20, 20, 100, 100)  
  
GraphicsWindow.BrushColor = "Green"  
GraphicsWindow.FillEllipse(100, 100, 100, 100)
```

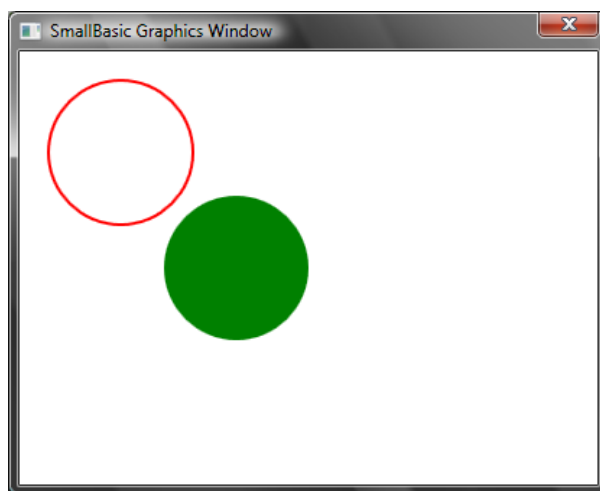


Figura 32 – Círculos

## Diversão com as Figuras

---

Nós iremos nos divertir neste capítulo com tudo que aprendemos até agora. Este capítulo contém exemplos que mostram algumas maneiras interessantes de combinar tudo que você aprendeu até agora para criar alguns programas visualmente bacanas.

### Retangular

Aqui nós desenhamos múltiplos retângulos em um loop, enquanto aumentamos seu tamanho.

```
GraphicsWindow.BackgroundColor = "Black"  
GraphicsWindow.PenColor = "LightBlue"  
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200  
  
For i = 1 To 100 Step 5  
    GraphicsWindow.DrawRectangle(100 - i, 100 - i, i * 2, i * 2)  
EndFor
```

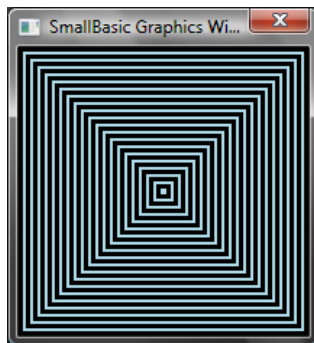


Figura 33 - Retangular

## Circular

Uma variante do programa anterior que desenha círculos ao invés de quadrados.

```
GraphicsWindow.BackgroundColor = "Black"  
GraphicsWindow.PenColor = "LightGreen"  
GraphicsWindow.Width = 200  
GraphicsWindow.Height = 200  
  
For i = 1 To 100 Step 5  
    GraphicsWindow.DrawEllipse(100 - i, 100 - i, i * 2, i * 2)  
EndFor
```

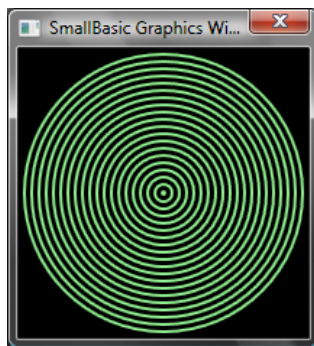


Figura 34 – Circular

## Aleatório

Este programa usa a operação `GraphicsWindow.GetRandomColor` (obter cor aleatória) para atribuir cores aleatórias para o pincel e então usa `Math.GetRandomNumber` (obter número aleatório) para atribuir as coordenadas x e y aos círculos. Essas duas operações podem ser combinadas de maneiras interessantes para criar programas atraentes que dão diferentes resultados cada vez que eles são executados.

```
GraphicsWindow.BackgroundColor = "Black"  
For i = 1 To 1000  
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()  
    x = Math.GetRandomNumber(640)  
    y = Math.GetRandomNumber(480)  
    GraphicsWindow.FillEllipse(x, y, 10, 10)  
EndFor
```

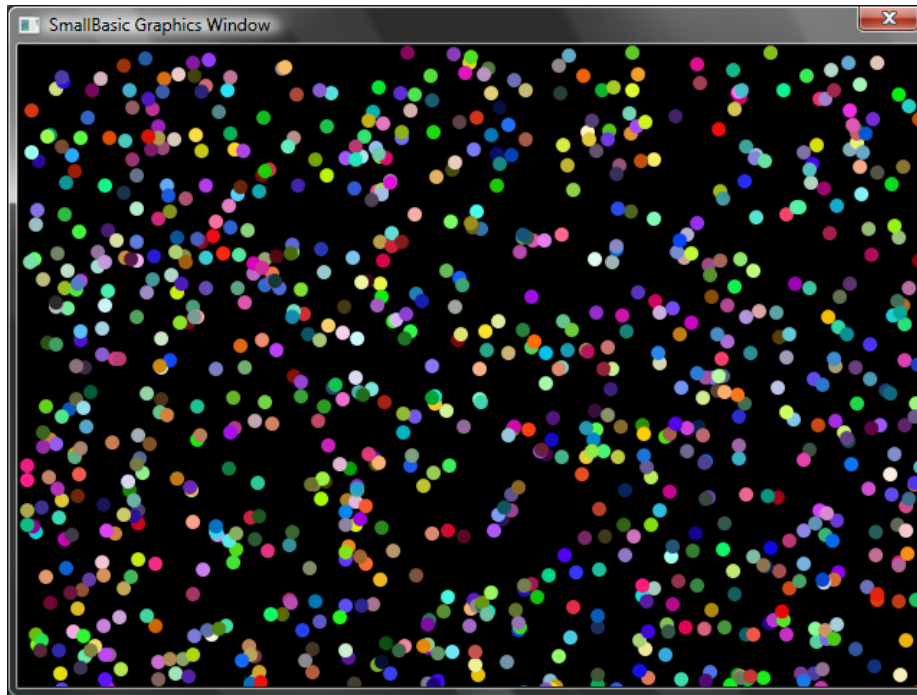


Figura 35 – Aleatório

## Fractais

O programa a seguir desenha um simples triângulo fractal usando números aleatórios. Um fractal é uma figura geométrica que pode ser subdividida em partes, cada uma delas se parece com a figura original exatamente. Neste caso, o programa desenha centenas de triângulos, onde cada um se parece com o triângulo original. E já que o programa executa por alguns segundos, você pode realmente ver os triângulos se formando lentamente a partir de meros pontos. A lógica em si é um pouco difícil de descrever e deixaremos como um exercício para você explorar.

```
GraphicsWindow.BackgroundColor = "Black"  
x = 100  
y = 100
```

```

For i = 1 To 100000
  r = Math.GetRandomNumber(3)
  ux = 150
  uy = 30
  If (r = 1) then
    ux = 30
    uy = 1000
  EndIf

  If (r = 2) Then
    ux = 1000
    uy = 1000
  EndIf

  x = (x + ux) / 2
  y = (y + uy) / 2

  GraphicsWindow.SetPixel(x, y, "LightGreen")
EndFor

```

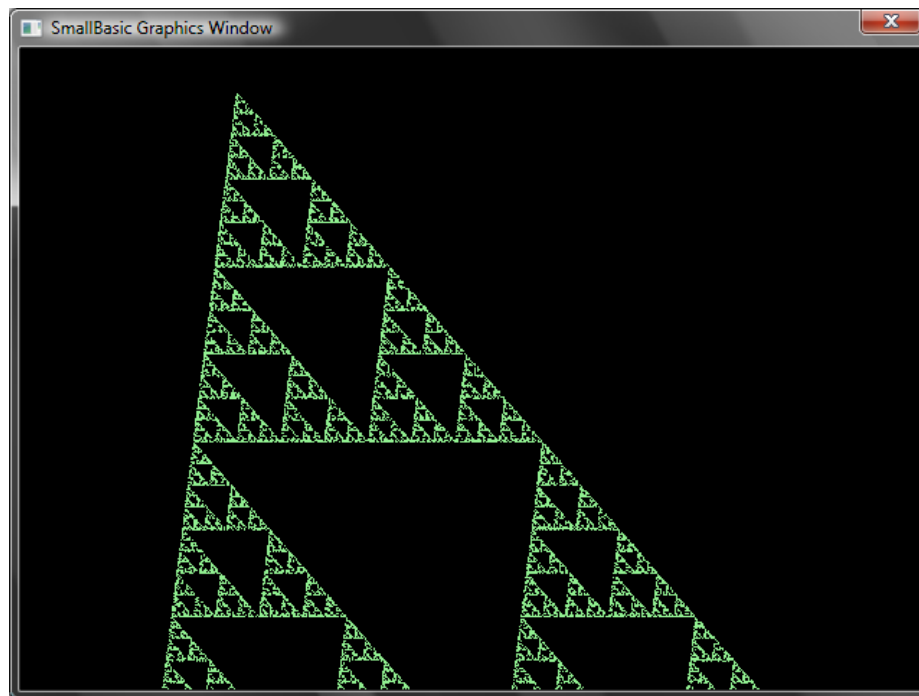


Figura 36 - Triângulo Fractal

Se você quiser realmente ver os pontos lentamente formando o fractal, você pode introduzir um atraso no loop através do uso da operação **Program.Delay**. Essa operação tem como

parâmetro um número que especifica em milissegundos quanto tempo retardar. Aqui está o programa modificado, com a linha alterada em negrito.

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
    r = Math.GetRandomNumber(3)
    ux = 150
    uy = 30
    If (r = 1) then
        ux = 30
        uy = 1000
    EndIf

    If (r = 2) Then
        ux = 1000
        uy = 1000
    EndIf

    x = (x + ux) / 2
    y = (y + uy) / 2

    GraphicsWindow.SetPixel(x, y, "LightGreen")
    Program.Delay(2)
EndFor
```

Aumentando o atraso fará seu programa executar mais lentamente. Experimente com os números para ver qual você gosta mais.

Outra modificação que você pode fazer é substituir a seguinte linha:

```
GraphicsWindow.SetPixel(x, y, "LightGreen")
```

por

```
color = GraphicsWindow.GetRandomColor()
GraphicsWindow.SetPixel(x, y, color)
```

Essa mudança fará com que o programa desenhe os pixels do triângulo usando cores aleatórias.

## Tartaruga Gráfica

---

### Logo

Nos anos 70, existia uma linguagem de programação bem simples mas poderosa chamada Logo que foi usada por alguns pesquisadores. Isso foi até que alguém adicionou o que foi chamado de “Tartaruga Gráfica” à linguagem e tornou disponível uma “tartaruga” que era visível na tela e respondia a comandos tais como “vá para frente”, “vire à direita”, “vire à esquerda”, etc. Usando a tartaruga, as pessoas eram capazes de desenhar formas interessantes na tela. Isso fez a linguagem imediatamente acessível e atraente para pessoas de todas as idades, e foi amplamente responsável por sua grande popularidade nos anos 80.

Small Basic vem com um objeto **Turtle** (tartaruga) com vários comandos que podem ser chamados a partir de programas Small Basic. Neste capítulo, nós usaremos a tartaruga para desenhar gráficos na tela.

### A Tartaruga

Para começar, nós precisamos fazer a tartaruga visível na tela. Isso pode ser alcançado através de uma simples linha de programa.

```
Turtle.Show()
```

Quando você executa este programa, você observará uma janela branca, assim como a que vimos no capítulo anterior, exceto que esta tem uma tartaruga no centro. É esta tartaruga que irá seguir nossas instruções e desenhá-la o que pedirmos a ela.



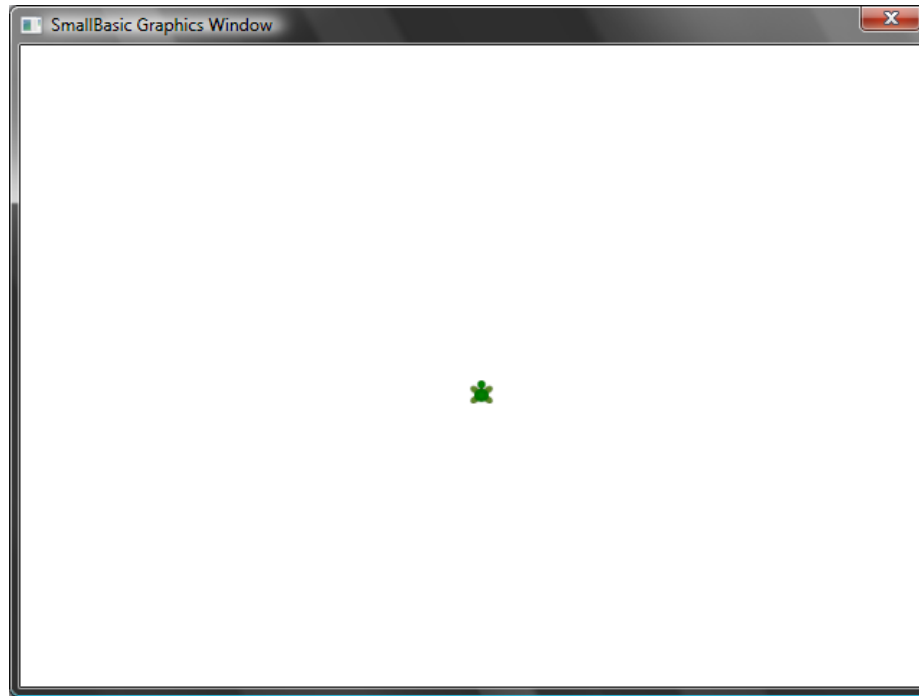


Figura 37 – A tartaruga está visível

## Movendo e Desenhando

Uma das instruções que a tartaruga entende é **Move** (mover). Esta operação recebe um número como entrada. Este número diz à tartaruga o quanto se mover. Digamos, no exemplo abaixo, nós pediremos à tartaruga para se mover 100 pixels.

```
Turtle.Move(100)
```

Quando você executa este programa, você pode realmente ver a tartaruga se mover lentamente uns 100 pixels para cima. Conforme ela se move, você também perceberá que ela está desenhando uma linha atrás dela. Quando a tartaruga tiver parado de se mover, o resultado se parecerá com o da figura abaixo.

*Quando usar operações de Turtle, não é necessário chamar o método Show(). A tartaruga se tornará visível automaticamente sempre que uma operação de Turtle for executada.*

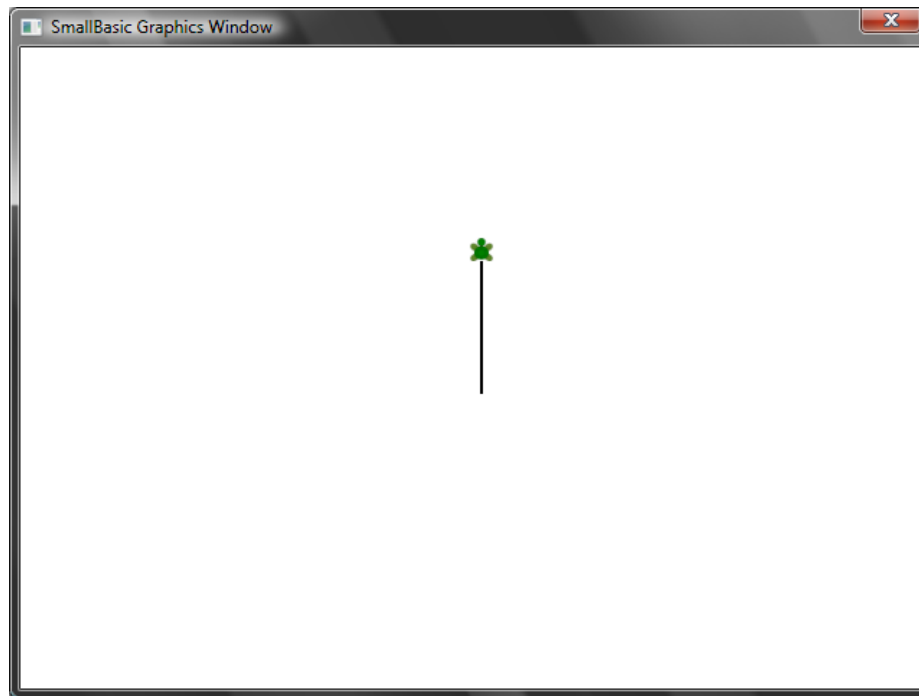


Figura 38 – Movendo cem pixels

## Desenhando um Quadrado

Um quadrado tem quatro lados, dois verticais e dois horizontais. Para desenhar um quadrado, nós precisamos ser capazes de fazer a tartaruga desenhar uma linha, virar à direita e desenhar outra linha e continuar isso até que os quatro lados estejam terminados. Se traduzíssemos isso em um programa, assim é como ele se pareceria.

```
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
```

Quando você executa este programa, você pode ver a tartaruga desenhando um quadrado, uma linha por vez, e o resultado se parece com a figura abaixo.

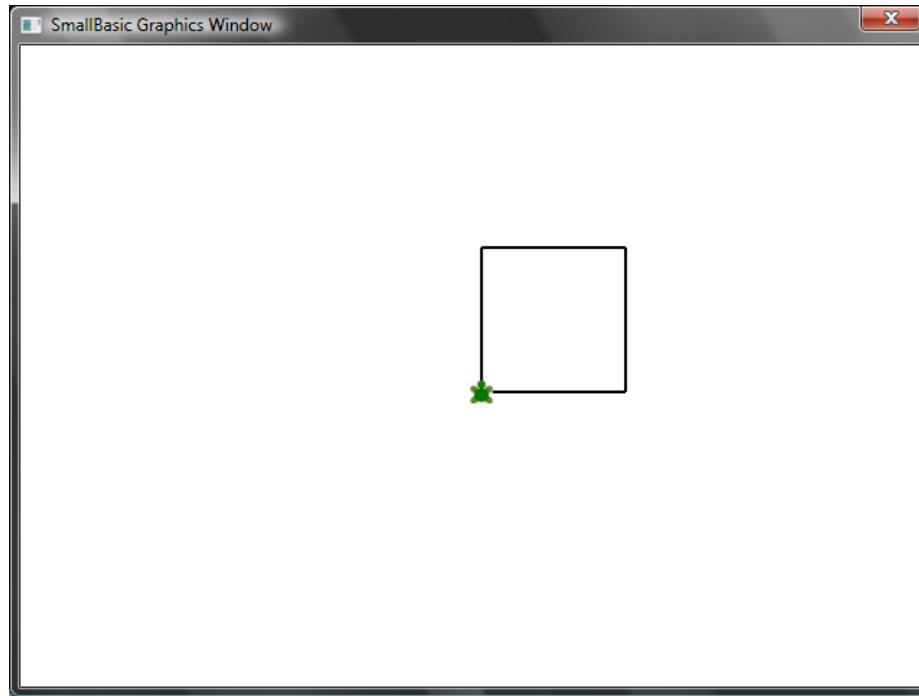


Figura 39 – A tartaruga desenhando um quadrado

Será interessante notar que nós estamos emitindo as mesmas duas instruções uma vez após a outra – quatro vezes para sermos mais precisos. E nós já aprendemos que tais comandos repetitivos podem ser executados usando loops. Portanto, se pegarmos o programa anterior e o modificarmos para usar o loop **For..EndFor**, nós terminaremos com um programa muito mais simples.

```
For i = 1 To 4
    Turtle.Move(100)
    Turtle.TurnRight()
EndFor
```

## Mudando as Cores

A tartaruga desenha na mesma *GraphicsWindow* (janela Gráficos) que nós vimos no capítulo anterior. Isso significa que todas as operações que nós aprendemos no capítulo anterior ainda são válidas aqui. Por exemplo, o programa a seguir desenhará o quadrado com cada lado com uma cor diferente.

```
For i = 1 To 4
    GraphicsWindow.PenColor = GraphicsWindow.GetRandomColor()
    Turtle.Move(100)
```

```
Turtle.TurnRight()  
EndFor
```

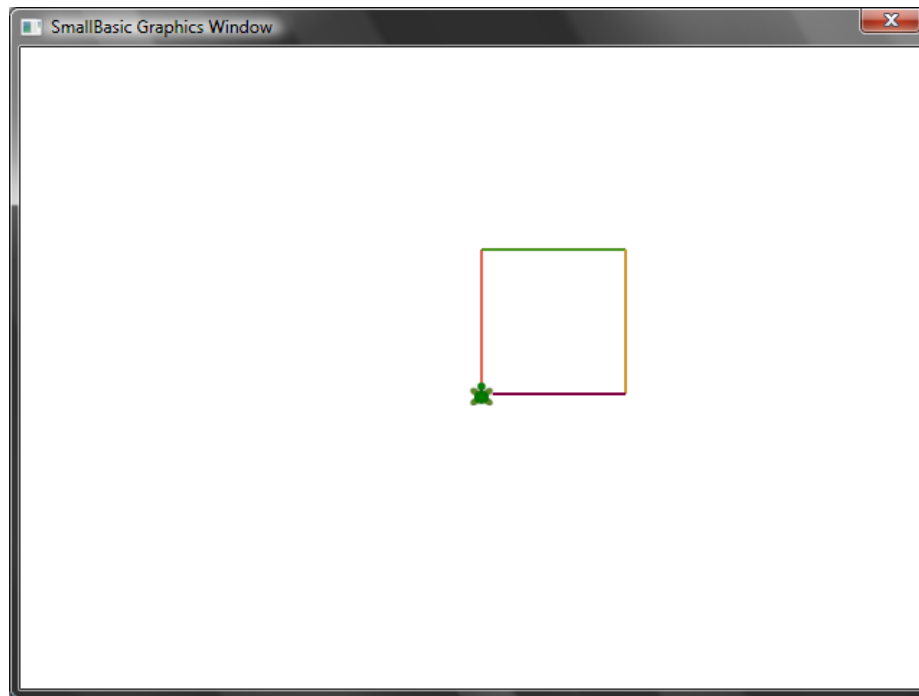


Figura 40 – Mudando as Cores

## Desenhando figuras mais complexas

A tartaruga, além das operações **TurnRight** (virar à direita) e **TurnLeft** (virar à esquerda), possui uma operação **Turn** (virar). Esta operação recebe uma entrada que especifica o ângulo de rotação. Usando esta operação, é possível desenhar um polígono com qualquer quantidade de lados. O programa a seguir desenha um hexágono (um polígono de seis lados).

```
For i = 1 To 6  
    Turtle.Move(100)  
    Turtle.Turn(60)  
EndFor
```

Experimente este programa para ver se ele realmente desenha um hexágono. Observe que como o ângulo entre os lados é de 60 graus, nós usamos **Turn(60)**. Para tal polígono, cujos lados são todos iguais, o ângulo entre os lados pode ser facilmente obtido dividindo 360 pelo número de lados. Munidos com esta informação e usando variáveis, nós podemos escrever um programa bastante genérico para desenhar um polígono com qualquer número de lados.

```
lados = 12

comprimento = 400 / lados
angulo = 360 / lados

For i = 1 To lados
    Turtle.Move(comprimento)
    Turtle.Turn(angulo)
EndFor
```

Usando este programa, nós podemos desenhar qualquer polígono apenas modificando a variável **lados**. Colocando 4 aqui nos daria o quadrado que nós começamos. Colocando um valor suficiente grande, digamos 50, faria o resultado indistinguível de um círculo.

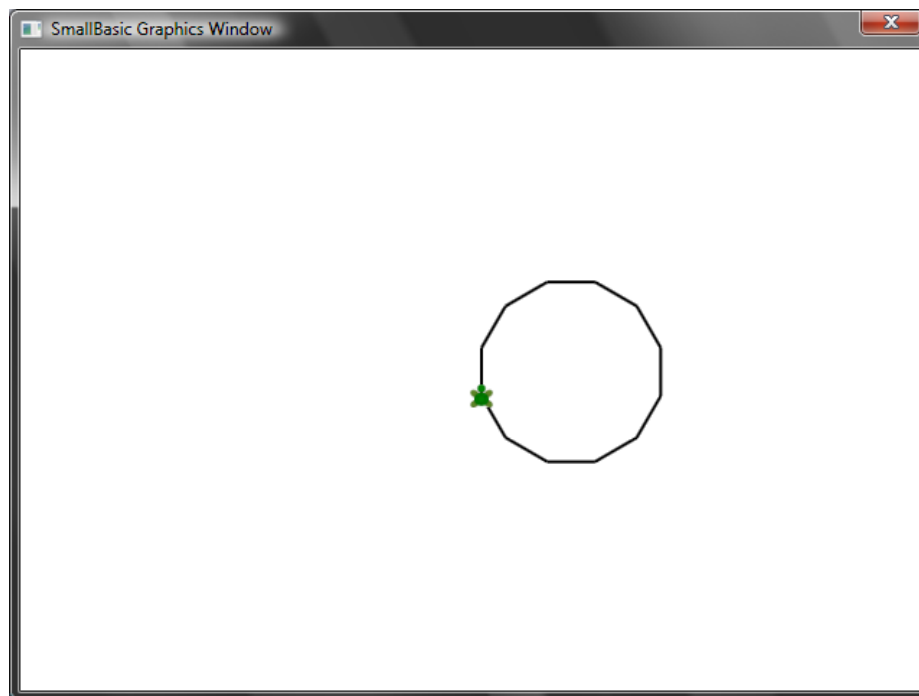


Figura 41 – Desenhando um polígono de 12 lados

Usando a técnica que acabamos de aprender, nós podemos fazer a tartaruga desenhar círculos cada vez com um pequeno movimento resultando em uma saída interessante.

```
lados = 50
comprimento = 400 / lados
angulo = 360 / lados

Turtle.Speed = 9
```

```

For j = 1 To 20
  For i = 1 To lados
    Turtle.Move(comprimento)
    Turtle.Turn(angulo)
  EndFor
  Turtle.Turn(18)
EndFor

```

*No programa acima, nós fizemos a tartaruga ir mais rápido, definindo a propriedade Speed (velocidade) para 9. Você pode configurar esta propriedade para qualquer valor entre 1 e 10 para fazer a tartaruga se mover o quão rápido você desejar.*

O programa acima tem dois loops **For..EndFor**, um dentro do outro. O loop interno (*i = 1 to lados*) é similar ao programa do polígono e é responsável por desenhar um círculo. O loop externo (*j = 1 to 20*) é responsável por virar a tartaruga um pouco cada vez que um círculo é desenhado. Isso diz à tartaruga para desenhar 20 círculos. Quando colocados juntos, este programa resulta num padrão muito interessante, como o mostrado abaixo.

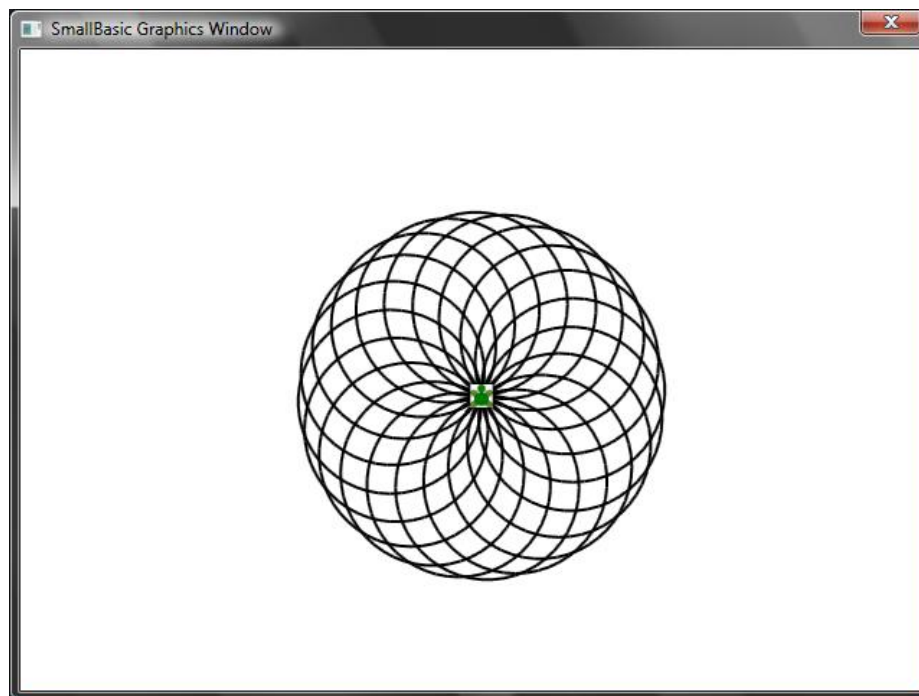


Figura 42 – Indo em círculos

## Movendo

Você pode fazer a tartaruga não desenhar usando a operação **PenUp** (levantar a caneta). Isso permite que você mova a tartaruga para qualquer lugar na tela sem desenhar uma linha.

Usando **PenDown** (abaixar a caneta) fará a tartaruga desenhar novamente. Isso pode ser usado para obter alguns efeitos interessantes, como por exemplo, linhas pontilhadas. Aqui está um programa que usa isso para desenhar um polígono com linhas pontilhadas.

```
lados = 6

comprimento = 400 / lados
angulo = 360 / lados

For i = 1 To lados
  For j = 1 To 6
    Turtle.Move(comprimento / 12)
    Turtle.PenUp()
    Turtle.Move(comprimento / 12)
    Turtle.PenDown()
  EndFor
  Turtle.Turn(angulo)
EndFor
```

Novamente, este programa tem dois loops. O loop interno desenha uma linha pontilhada, enquanto o loop externo especifica quantas linhas desenhar. No nosso exemplo, nós usamos 6 para a variável **lados** e portanto nós obtivemos um hexágono de linhas pontilhadas, como mostra a figura abaixo.

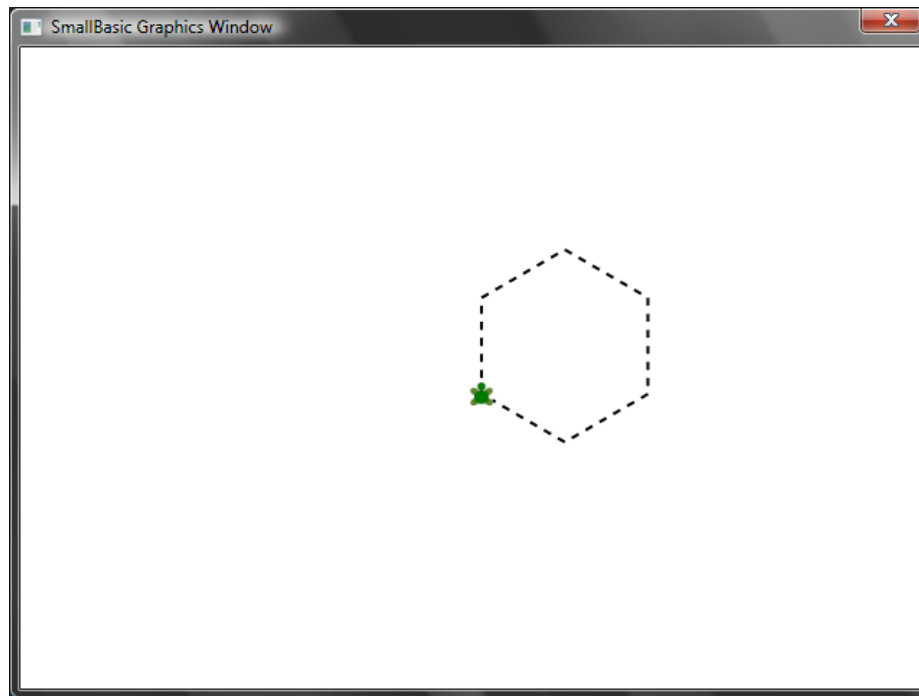


Figura 43 - Usando PenUp e PenDown



## Sub-rotinas

---

Muito frequentemente, enquanto escrevemos programas, nós encontramos casos onde teremos que executar os mesmos passos repetidamente. Nesses casos, não faria muito sentido reescrever as mesmas instruções múltiplas vezes. É então que *sub-rotinas* se tornam úteis.

Uma sub-rotina é uma porção de código dentro de um programa maior que geralmente faz algo bastante específico, e pode ser chamada de qualquer ponto no programa. Sub-rotinas são identificadas por um nome que segue a palavra-chave **Sub** e são terminadas pela palavra-chave **EndSub**. Por exemplo, o fragmento de código a seguir representa uma sub-rotina cujo nome é *ImprimirHora*, e ela realiza a tarefa de imprimir a hora atual na **TextWindow**.

```
Sub ImprimirHora
    TextWindow.WriteLine(Clock.Time)
EndSub
```

A seguir está um programa que inclui a sub-rotina e a chama a partir de vários lugares.

```
ImprimirHora()
TextWindow.Write("Digite seu nome: ")
nome = TextWindow.Read()
TextWindow.Write(nome + ", a hora agora é: ")
ImprimirHora()

Sub ImprimirHora
```

```
TextWindow.WriteLine(Clock.Time)
EndSub
```

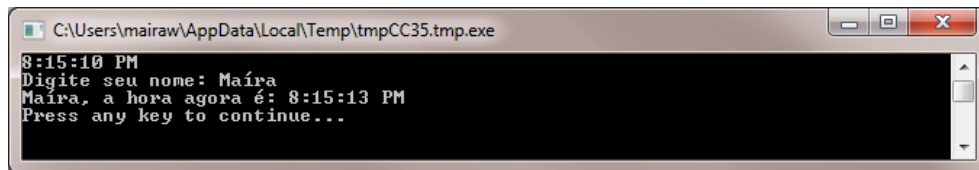


Figura 44 – Chamando uma simples sub-rotina

Você executa uma sub-rotina chamando *NomeSubrotina ()*. Como sempre, os parênteses “()” são necessários para dizer ao computador o que fazer quando executar a sub-rotina.

## Vantagens de usar sub-rotinas

Como acabamos de ver, sub-rotinas ajudam a reduzir a quantidade de código que você tem que digitar. Uma vez que você tem a sub-rotina *ImprimirHora* escrita, você pode chamá-la a partir de qualquer ponto no seu programa e ela imprimirá a hora atual.

*Lembre-se que você só pode chamar uma sub-rotina Small Basic a partir do mesmo programa. Você não pode chamar uma sub-rotina a partir de qualquer outro programa.*

Além disso, sub-rotinas ajudam a decompor problemas complexos em peças mais simples. Digamos que você tenha uma equação complexa para resolver, você pode escrever diversas sub-rotinas que resolvem peças menores da equação complexa. Então você pode juntar os resultados para obter a solução para a equação complexa original.

Sub-rotinas também podem auxiliar a melhorar a legibilidade de um programa. Em outras palavras, se você nomear bem as sub-rotinas para porções que são executadas frequentemente no seu programa, seu programa se torna fácil de ler e compreender. Isto é muito importante se você quiser entender o programa de outra pessoa ou se você quer que seu programa seja entendido por outros. Algumas vezes, é útil mesmo quando você quer ler seu próprio programa, digamos uma semana depois que você o escreveu.

## Usando variáveis

Você pode acessar e usar qualquer variável que você tem em um programa de dentro de uma sub-rotina. Por exemplo, o programa a seguir aceita dois números e imprime o maior dos dois. Observe que a variável *maior* é usada tanto dentro como fora da sub-rotina.

```

TextWindow.Write("Digite o primeiro número: ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Digite o segundo número: ")
num2 = TextWindow.ReadNumber()

DescobrirMaior()
TextWindow.WriteLine("O maior número é: " + maior)

Sub DescobrirMaior
    If (num1 > num2) Then
        maior = num1
    Else
        maior = num2
    EndIf
EndSub

```

E a saída do programa se parece com isso:

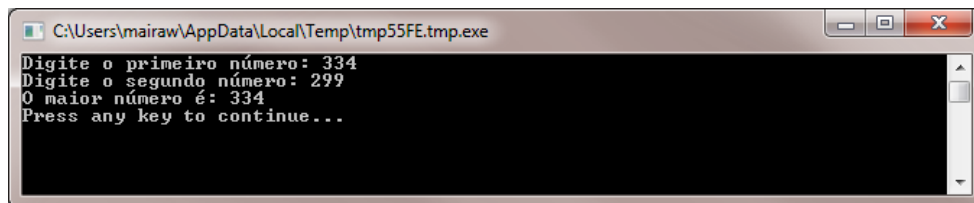


Figura 45 – O maior de dois números usando sub-rotina

Vamos examinar outro exemplo que ilustrará o uso de sub-rotinas. Desta vez usaremos um programa gráfico que calcula vários pontos que ele armazenará nas variáveis *x* e *y*. Então ele chama uma sub-rotina **DesenharCirculoUsandoCentro** que é responsável por desenhar um círculo usando *x* e *y* como o centro.

```

GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 480
For i = 0 To 6.4 Step 0.17
    x = Math.Sin(i) * 100 + 200
    y = Math.Cos(i) * 100 + 200

    DesenharCirculoUsandoCentro()
EndFor

Sub DesenharCirculoUsandoCentro
    inicioX = x - 40

```

```
inicioY = y - 40

GraphicsWindow.DrawEllipse(inicioX, inicioY, 120, 120)
EndSub
```

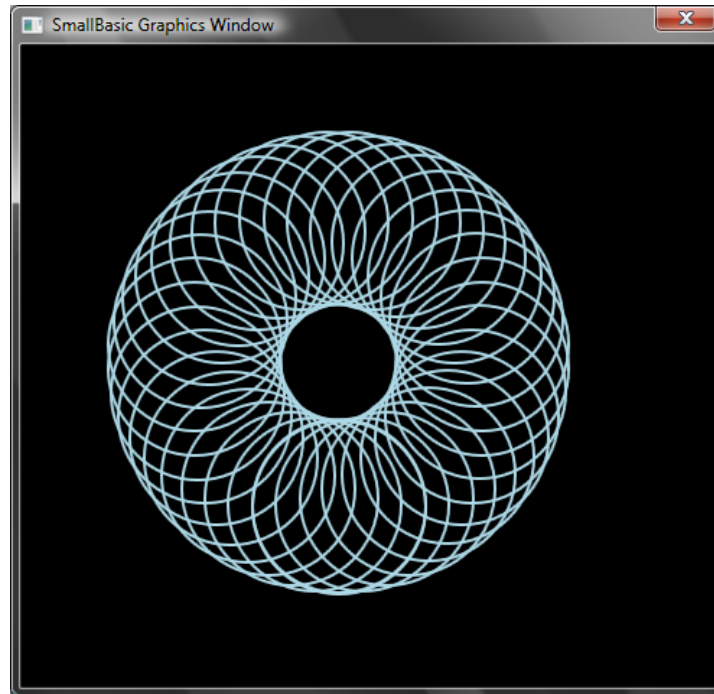


Figura 46 – Exemplo Gráfico para sub-rotinas

## Chamando sub-rotinas dentro de Loops

Algumas vezes sub-rotinas são chamados de dentro de um loop, durante o qual elas executam o mesmo conjunto de instruções mas com diferentes valores em uma ou mais variáveis. Por exemplo, digamos que você tenha uma sub-rotina chamada *VerificarPrimo* e esta sub-rotina determina se um número é primo ou não. Você pode escrever um programa que deixa o usuário digitar um valor e você pode então dizer se é primo ou não, usando esta sub-rotina. O programa abaixo ilustra isso.

```
TextWindow.Write("Digite um número: ")
i = TextWindow.ReadNumber()
primo = "True"
VerificarPrimo()
If (primo = "True") Then
    TextWindow.WriteLine(i + " é um número primo ")
Else
```

```

    TextWindow.WriteLine(i + " não é um número primo ")
EndIf

Sub VerificarPrimo
    For j = 2 To Math.SquareRoot(i)
        If (Math.Remainder(i, j) = 0) Then
            primo = "False"
            Goto FimLoop
        EndIf
    Endfor
FimLoop:
EndSub

```

A sub-rotina *VerificarPrimo* recebe o valor de *i* e tenta dividi-lo por números menores. Se um número divide *i* e tem resto zero, então *i* não é um número primo. Neste ponto, a sub-rotina configura o valor de *primo* para “False” (falso) e sai. Se o número era indivisível por números menores então o valor de *primo* permanece como “True” (verdadeiro).

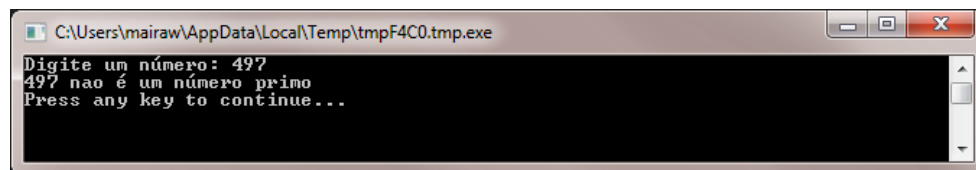


Figura 47 – Verificando se é um número primo

Agora que temos uma sub-rotina que pode fazer a verificação de números primos para nós, você pode querer usar isso para listar todos os números primos menores que, por exemplo, 100. É muito fácil modificar o programa anterior e chamar *VerificarPrimo* de dentro de um loop. Isso dá à sub-rotina um valor diferente para calcular toda vez que o loop executa. Vejamos como isso é feito no exemplo a seguir.

```

For i = 3 To 100
    primo = "True"
    VerificarPrimo()
    If (primo = "True") Then
        TextWindow.WriteLine(i)
    EndIf
EndFor

Sub VerificarPrimo
    For j = 2 To Math.SquareRoot(i)
        If (Math.Remainder(i, j) = 0) Then

```

```
    primo = "False"  
    Goto FimLoop  
EndIf  
Endfor  
FimLoop:  
EndSub
```

No programa anterior, o valor de  $i$  é atualizado toda vez que o loop é executado. Dentro do loop, uma chamada para a sub-rotina *VerificarPrimo* é feita. A sub-rotina *VerificarPrimo* então recebe o valor de  $i$  e calcula se  $i$  é ou não um número primo. Este resultado é armazenado na variável *primo* onde é então acessada pelo loop fora da sub-rotina. O valor de  $i$  é então impresso se ele for um número primo. E já que o loop inicia em 3 e vai até 100, nós obtemos a lista de todos os números primos que estão entre 3 e 100. Abaixo está o resultado do programa.

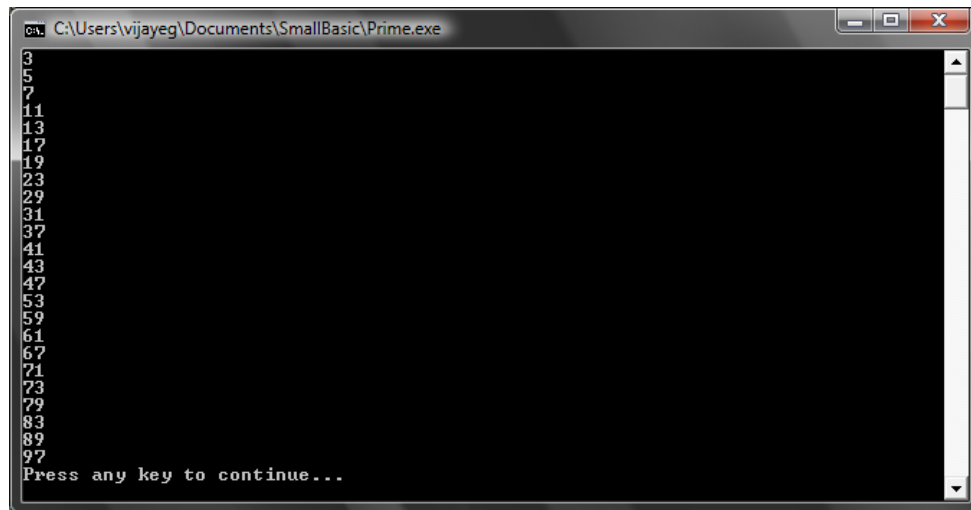


Figura 48 – Números Primos

A esta altura, você deve estar bem acostumado em como usar variáveis – afinal você chegou até este ponto e ainda está se divertindo, certo?

Vamos, por um momento, revisar o primeiro programa que você escreveu com variáveis:

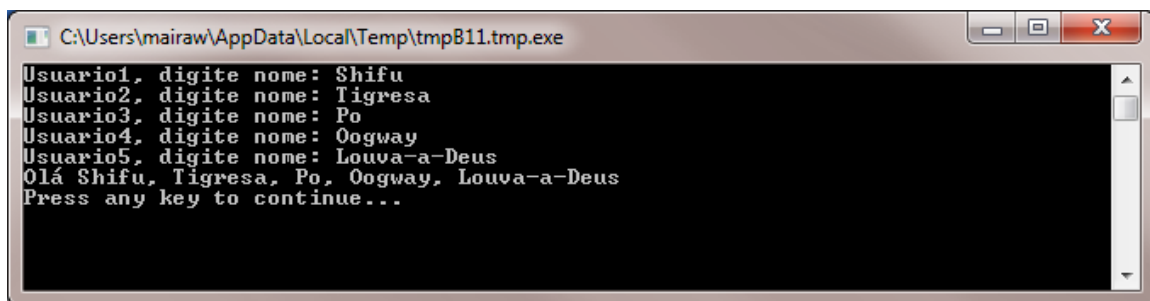
```
TextWindow.Write("Digite seu Nome: ")
nome = TextWindow.Read()
TextWindow.WriteLine("Olá " + nome)
```

Neste programa, nós recebemos e armazenamos o nome do usuário numa variável chamada **nome**. Depois nós dissemos “Olá” ao usuário. Agora, vamos dizer que existam mais de um usuário – digamos que existam 5 usuários. Como iríamos armazenar todos esses nomes? Uma maneira de fazer isso é:

```
TextWindow.Write("Usuario1, digite nome: ")
nome1 = TextWindow.Read()
TextWindow.Write("Usuario2, digite nome: ")
nome2 = TextWindow.Read()
TextWindow.Write("Usuario3, digite nome: ")
nome3 = TextWindow.Read()
TextWindow.Write("Usuario4, digite nome: ")
nome4 = TextWindow.Read()
TextWindow.Write("Usuario5, digite nome: ")
nome5 = TextWindow.Read()
```

```
TextWindow.Write("Olá ")
TextWindow.Write(nome1 + ", ")
TextWindow.Write(nome2 + ", ")
TextWindow.Write(nome3 + ", ")
TextWindow.Write(nome4 + ", ")
TextWindow.WriteLine(nome5)
```

Quando você executar este código, você obterá um resultado similar a este:



```
C:\Users\mairaw\AppData\Local\Temp\tmp811.tmp.exe
Usuario1, digite nome: Shifu
Usuario2, digite nome: Tigresa
Usuario3, digite nome: Po
Usuario4, digite nome: Oogway
Usuario5, digite nome: Louva-a-Deus
Olá Shifu, Tigresa, Po, Oogway, Louva-a-Deus
Press any key to continue...
```

Figura 49 – Sem usar matrizes

Claramente deve existir um forma mais fácil de escrever um programa tão simples, certo? Especialmente já que o computador é muito bom ao fazer tarefas repetitivas, porque deveríamos se dar ao trabalho de escrever o mesmo código vez após outra para cada novo usuário? O truque aqui é armazenar e recuperar mais de um nome de usuário usando a mesma variável. Se pudermos fazer isso, então podemos usar o loop **For** que aprendemos em capítulos anteriores. Aqui é aonde matrizes vêm nos ajudar.

## O que é uma matriz?

Uma matriz um tipo especial de variável que pode armazenar mais de um valor ao mesmo tempo. Basicamente, isso significa que ao invés de ter que criar **nome1**, **nome2**, **nome3**, **nome4** e **nome5** para armazenar cinco nomes de usuário, nós podemos apenas usar **nome** para armazenar os cinco nomes de usuário. A maneira como armazenamos valores múltiplos é através do uso de uma coisa chama “índice”. Por exemplo, **nome[1]**, **nome[2]**, **nome[3]**, **nome[4]** e **nome[5]** podem armazenar um valor cada um. Os números 1, 2, 3, 4 e 5 são chamados *índices* para a matriz.

Apesar de **nome[1]**, **nome[2]**, **nome[3]**, **nome[4]** e **nome[5]** parecerem que são variáveis diferentes, eles são na realidade apenas uma variável. E qual a vantagem disso, você pode perguntar. A melhor parte de armazenar valores em uma matriz é que você pode especificar o índice usando outra variável – que nos permite acessar matrizes dentro de loops.



Agora, vamos ver como podemos colocar nosso novo conhecimento em prática rescrevendo nosso programa anterior com matrizes.

```
For i = 1 To 5
    TextWindow.Write("Usuario" + i + ", digite nome: ")
    nome[i] = TextWindow.Read()
EndFor

TextWindow.Write("Olá ")
For i = 1 To 5
    TextWindow.Write(nome[i] + ", ")
EndFor
TextWindow.WriteLine("")
```

Muito mais fácil de ler, não? Observe as duas linhas em negrito. A primeira armazena um valor na matriz e a segunda lê um valor da matriz. O valor que você armazena em **nome[1]** não será afetado pelo que você armazenar em **nome[2]**. Portanto, para a maioria dos casos, você pode tratar **nome[1]** e **nome[2]** como duas variáveis diferentes com a mesma identidade.

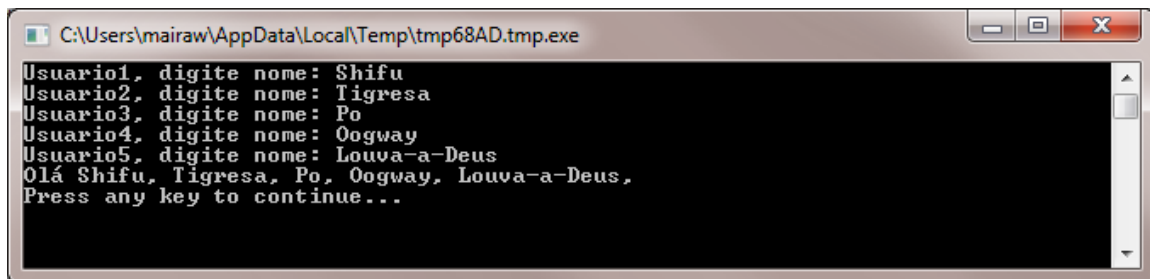


Figura 50 – Usando matrizes

O programa acima dá quase exatamente o mesmo resultado que o programa sem matrizes, exceto pela vírgula após *Louva-a-Deus*. Nós podemos consertar isso rescrevendo o loop de impressão assim:

```
TextWindow.Write("Olá ")
For i = 1 To 5
    TextWindow.Write(nome[i])
    If i < 5 Then
        TextWindow.Write(", ")
    EndIf
EndFor
TextWindow.WriteLine("")
```

*índices da matriz não distinguem maiúsculas de minúsculas, assim como variáveis regulares.*

## Indexando uma matriz

No nosso programa anterior, você viu que usamos números como índices para armazenar e recuperar valores da matriz. Acontece que índices não são restritos apenas a números e na verdade é muito útil usar índices textuais também. Por exemplo, no programa a seguir, nós solicitamos e armazenamos várias informações a respeito de um usuário e então imprimimos a informação que o usuário desejar.

```
TextWindow.Write("Digite nome: ")
usuario["nome"] = TextWindow.Read()
TextWindow.Write("Digite idade: ")
usuario["idade"] = TextWindow.Read()
TextWindow.Write("Digite cidade: ")
usuario["cidade"] = TextWindow.Read()
TextWindow.Write("Digite CEP: ")
usuario["CEP"] = TextWindow.Read()

TextWindow.Write("Que informação você deseja? ")
indice = TextWindow.Read()
TextWindow.WriteLine(indice + " = " + usuario[indice])
```

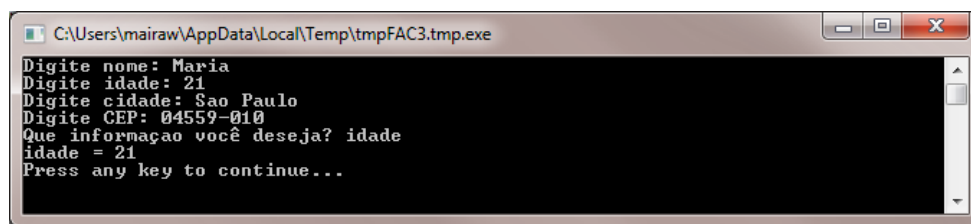


Figura 51 – Usando índices não numéricos

## Mais de uma dimensão

Digamos que você queira armazenar o nome e o telefone de todos seus amigos e então ser capaz de buscar seus telefones quando você precisar – uma espécie de agenda telefônica. Como você escreveria tal programa?

Neste caso, existem dois conjuntos de índices (também conhecidos como dimensão da matriz) envolvidos. Vamos assumir que nós identificamos cada amigo pelo seu apelido. Isso se torna nosso primeiro índice na matriz. Uma vez que usamos o primeiro índice para obtermos a

variável do nosso amigo, o segundo dos índices, **nome** e **telefone** nos ajudariam a obter o nome e telefone daquele amigo.

A maneira como armazenamos estes dados seria assim:

```
amigos["JC"]["Nome"] = "João Carlos"
amigos["JC"]["Telefone"] = "3221-1234"

amigos["Van"]["Nome"] = "Vanessa"
amigos["Van"]["Telefone"] = "3227-4567"

amigos["Carol"]["Nome"] = "Caroline"
amigos["Carol"]["Telefone"] = "3224-2345"
```

Já que temos dois índices na mesma matriz **amigos**, esta matriz é chamada de matriz de duas dimensões.

Uma vez que temos este programa configurado, nós podemos receber como entrada o apelido de um amigo e imprimir a informação que temos armazenada sobre eles. Aqui está o programa completo que faz isso:

```
amigos["JC"]["Nome"] = "João Carlos"
amigos["JC"]["Telefone"] = "3221-1234"

amigos["Van"]["Nome"] = "Vanessa"
amigos["Van"]["Telefone"] = "3227-4567"

amigos["Carol"]["Nome"] = "Caroline"
amigos["Carol"]["Telefone"] = "3224-2345"

TextWindow.Write("Digite o apelido: ")
apelido = TextWindow.Read()

TextWindow.WriteLine("Nome: " + amigos[apelido]["Nome"])
TextWindow.WriteLine("Telefone: " + amigos[apelido]["Telefone"])
```

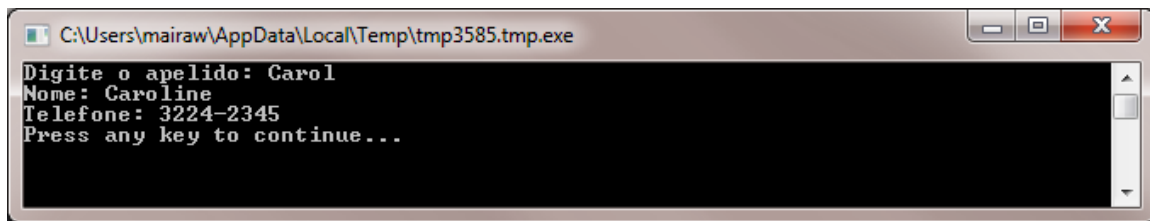


Figura 52 – Uma agenda telefônica simples

## Usando Matrizes para representar grades

Um uso muito comum de matrizes multidimensionais é representar grades/tabelas. Grades tem linhas e colunas, que podem se ajustar muito bem em uma matriz de duas dimensões. Um programa simples que dispõe caixas numa grade é dado a seguir:

```
linhas = 8
colunas = 8
tamanho = 40

For l = 1 To linhas
  For c = 1 To colunas
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    caixas[l][c] = Shapes.AddRectangle(tamanho, tamanho)
    Shapes.Move(caixas[l][c], c * tamanho, l * tamanho)
  EndFor
EndFor
```

Este programa adiciona retângulos e os posiciona para formarem uma grade 8x8. Além de posicionar essas caixas, o programa também armazena essas caixas numa matriz. Isso nos permite controlar essas caixas facilmente e usá-las novamente quando necessário.

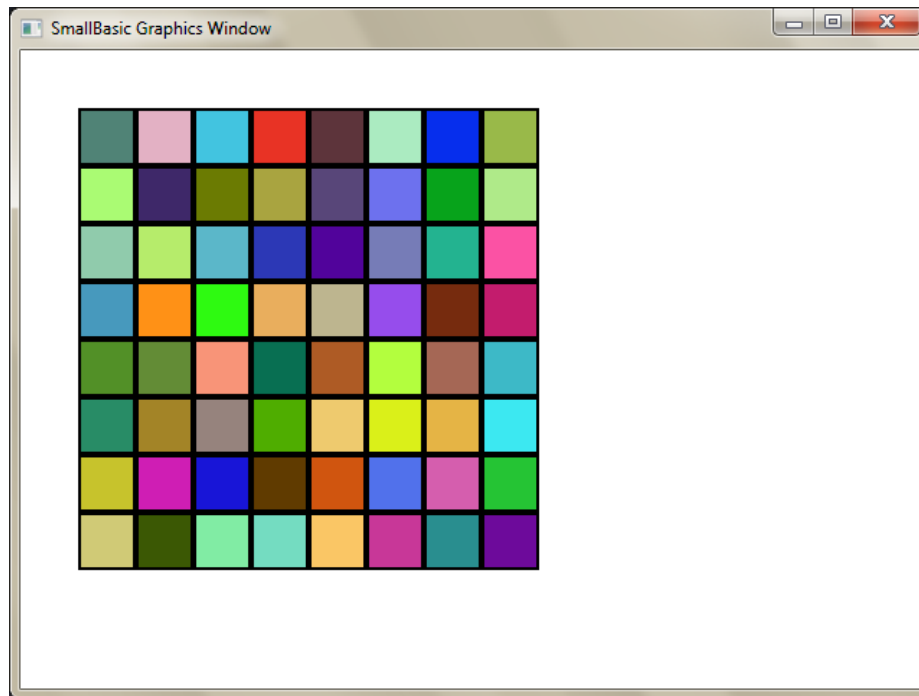


Figura 53 – Posicionando caixas numa grade

Por exemplo, adicionando o código a seguir no final do programa anterior faria essas caixas se moverem para o canto superior esquerdo.

```
For l = 1 To linhas
  For c = 1 To colunas
    Shapes.Animate(caixas[l][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```

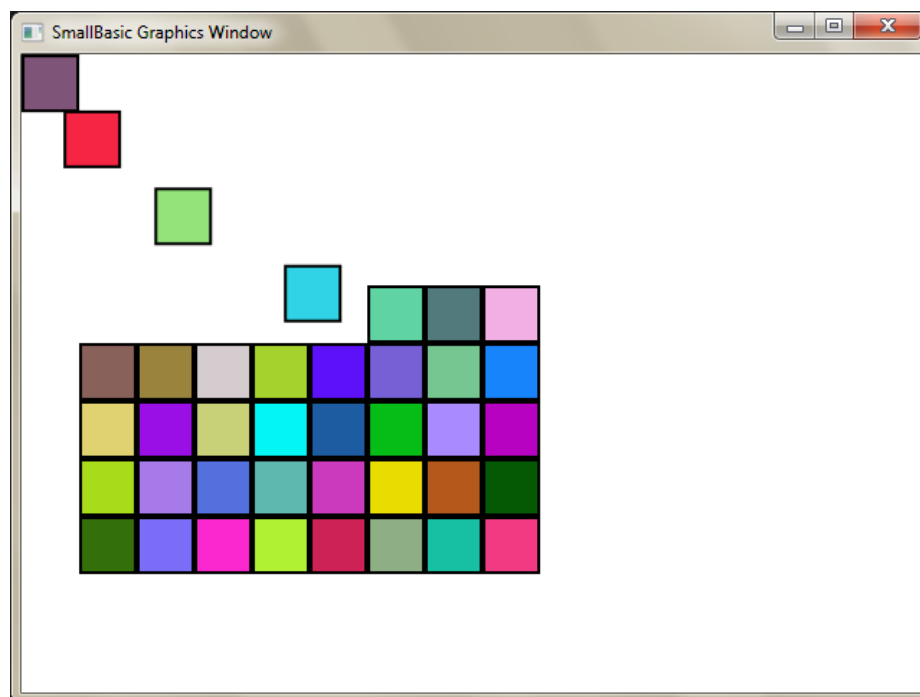


Figura 54 – Controlando as caixas na grade

## Eventos e Interatividade

---

Nos primeiros dois capítulos, nós introduzimos objetos que tem *Propriedades* e *Operações*. Além de propriedades e operações, alguns objetos têm o que chamamos de **Eventos**. Eventos são como sinais que são gerados, por exemplo, em resposta a ações do usuário, tais como mover o mouse ou clicar em um botão do mouse. De certa forma, eventos são o oposto de operações. No caso da operação, você como um programador chama a operação para fazer o computador fazer algo; onde no caso de eventos, o computador lhe informa quando algo interessante aconteceu.

### Como os eventos são úteis?

Eventos são centrais para introduzir interatividade em um programa. Se você quer permitir que um usuário interaja com seu programa, eventos são o que você vai usar. Digamos que você esteja escrevendo um jogo da velha. Você quer permitir que o usuário escolha a sua jogada, certo? Aí é onde eventos entram – você recebe a entrada do usuário de dentro do seu programa usando eventos. Se isso parece difícil de entender, não se preocupe, nós examinaremos um exemplo bem simples que ajudará você a entender o que são eventos e como eles podem ser usados.

A seguir está um programa bem simples que tem apenas uma instrução e uma sub-rotina. A sub-rotina usa a operação *ShowMessage* (mostrar mensagem) no objeto *GraphicsWindow* (janela Gráficos) para exibir um caixa de mensagem para o usuário.

```
GraphicsWindow.MouseDown = OnMouseDown
```

```
Sub OnMouseDown
    GraphicsWindow.ShowMessage("Você clicou.", "Olá")
EndSub
```

A parte interessante de observar no programa acima é que a linha onde nós atribuímos o nome da sub-rotina ao evento **MouseDown** do objeto *GraphicsWindow*. Você observará que o evento *MouseDown* se parece muito com uma propriedade – exceto que ao invés de atribuirmos um valor, nós estamos atribuindo a sub-rotina *OnMouseDown* a ele. Isto que é especial a respeito de eventos – quando o evento acontece, a sub-rotina é automaticamente chamada. Neste caso, a sub-rotina *OnMouseDown* é chamada toda vez que o usuário clica o mouse, na *GraphicsWindow*. Vamos lá, execute o programa e experimente. Cada vez que clicar na *GraphicsWindow* com seu mouse, você verá uma caixa de mensagem como a mostrada na figura abaixo.

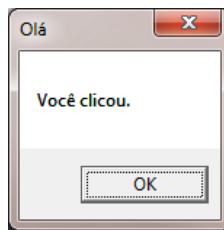


Figura 55 – Resposta a um evento

Este tipo de manipulação de eventos é muito poderoso e permite a criação de programas bem criativos e interessantes. Programas escritos desta forma são geralmente chamados de programas orientados a eventos.

Você pode modificar a sub-rotina *OnMouseDown* para fazer outras coisas além de exibir uma caixa de mensagem. Por exemplo, como no programa a seguir, você pode desenhar grandes círculos azuis onde o usuário clicar com o mouse.

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
    x = GraphicsWindow.MouseX - 10
    y = GraphicsWindow.MouseY - 10
    GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```



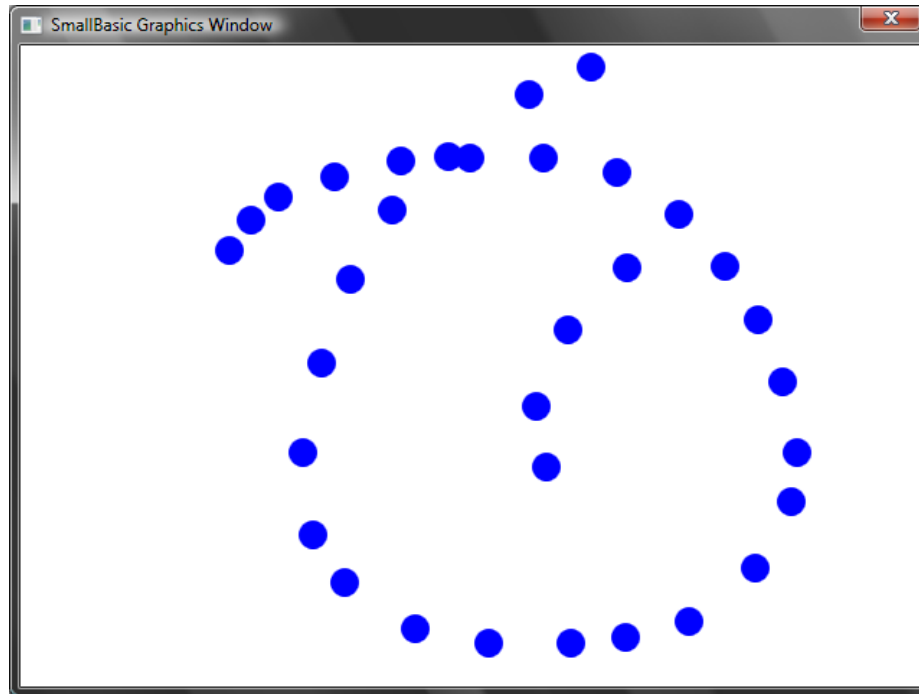


Figura 556 – Manipulando o evento Mouse Down

Observe que no programa acima nós usamos *MouseX* e *MouseY* para obter as coordenadas do mouse. Nós então usamos isto para desenhar um círculo usando as coordenadas do mouse como o centro do círculo.

## Manipulando múltiplos eventos

Não existem limites para quantos eventos você deseja manipular. Você pode até mesmo ter uma sub-rotina que manipula múltiplos eventos. Entretanto, você pode manipular um evento apenas uma vez. Se você tentar atribuir duas sub-rotinas ao mesmo evento, a segunda vence.

Para ilustrar isso, vamos pegar o exemplo anterior e adicionar uma sub-rotina que manipula pressionamento de teclas. Além disso, vamos fazer esta nova sub-rotina alterar a cor do pincel, para que quando você clicar com o mouse, você obterá um círculo de uma cor diferente.

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown
GraphicsWindow.KeyDown = OnKeyDown

Sub OnKeyDown
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
EndSub
```

```
Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
  y = GraphicsWindow.MouseY - 10
  GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```

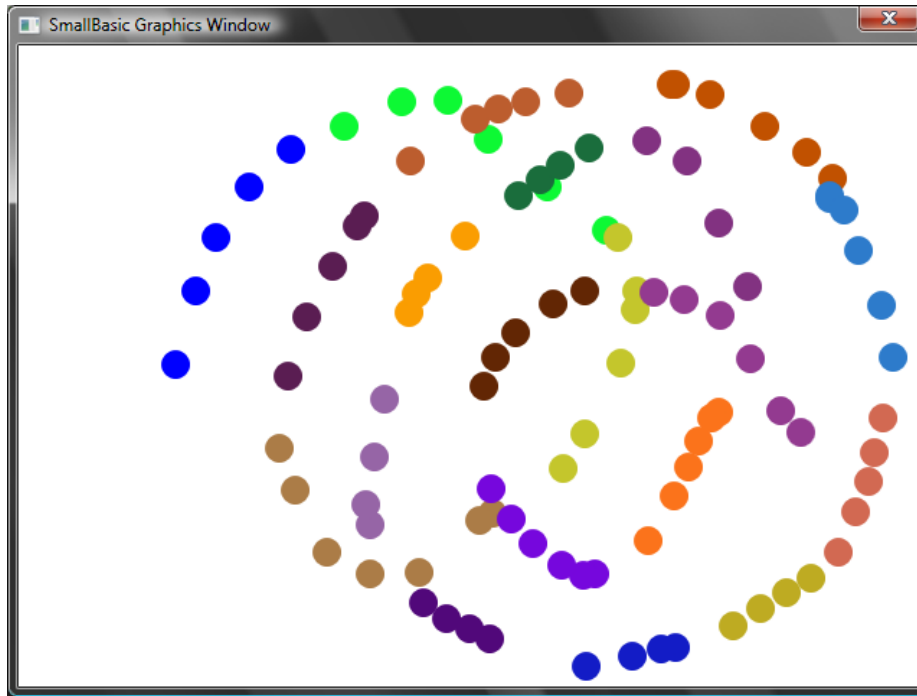


Figura 567 – Manipulando múltiplos eventos

Se você executar este programa e clicar na janela, você obterá um círculo azul. Agora, se você pressionar qualquer tecla uma vez e clicar novamente, você obterá um círculo com uma cor diferente. O que está acontecendo quando você pressiona uma tecla é que a sub-rotina *OnKeyDown* é executada, mudando a cor do pincel para uma cor randômica. Depois disso quando você clica com o mouse, um círculo é desenhado usando a nova cor que foi configurada – dando os círculos de cores randômicas.

## Um programa de desenho

Munido com eventos e sub-rotinas, nós agora podemos escrever um programa que deixa usuários desenhar na janela. É surpreendentemente fácil escrever tal programa, dado que nós podemos quebrar o problema em pequenas partes. Como primeiro passo, vamos escrever um programa que permite que usuários movam o mouse aonde eles quiserem na janela Gráficos, deixando um rastro por onde eles passam.

```

GraphicsWindow.MouseMove = OnMouseMove

Sub OnMouseMove
    x = GraphicsWindow.MouseX
    y = GraphicsWindow.MouseY
    GraphicsWindow.DrawLine(prevX, prevY, x, y)
    prevX = x
    prevY = y
EndSub

```

Entretanto, quando você executa este programa, a primeira linha sempre começa a partir do canto superior esquerdo da janela (0, 0). Nós podemos consertar este problema manipulando o evento *MouseDown* e capturando os valores *prevX* e *prevY* quando o evento ocorre.

Além disso, nós apenas precisamos do rastro quando o usuário está pressionando o botão do mouse. Nas outras vezes, nós não deveríamos desenhar a linha. Para obter este comportamento, nós usaremos a propriedade *IsLeftButtonDown* do objeto **Mouse**. Esta propriedade diz se o botão esquerdo do mouse está sendo pressionado ou não. Se o valor for verdadeiro, desenharemos a linha, senão pularemos.

```

GraphicsWindow.MouseMove = OnMouseMove
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
    prevX = GraphicsWindow.MouseX
    prevY = GraphicsWindow.MouseY
EndSub

Sub OnMouseMove
    x = GraphicsWindow.MouseX
    y = GraphicsWindow.MouseY
    If (Mouse.IsLeftButtonDown) Then
        GraphicsWindow.DrawLine(prevX, prevY, x, y)
    EndIf
    prevX = x
    prevY = y
EndSub

```

## Exemplos Divertidos

### Fractal com a Tartaruga

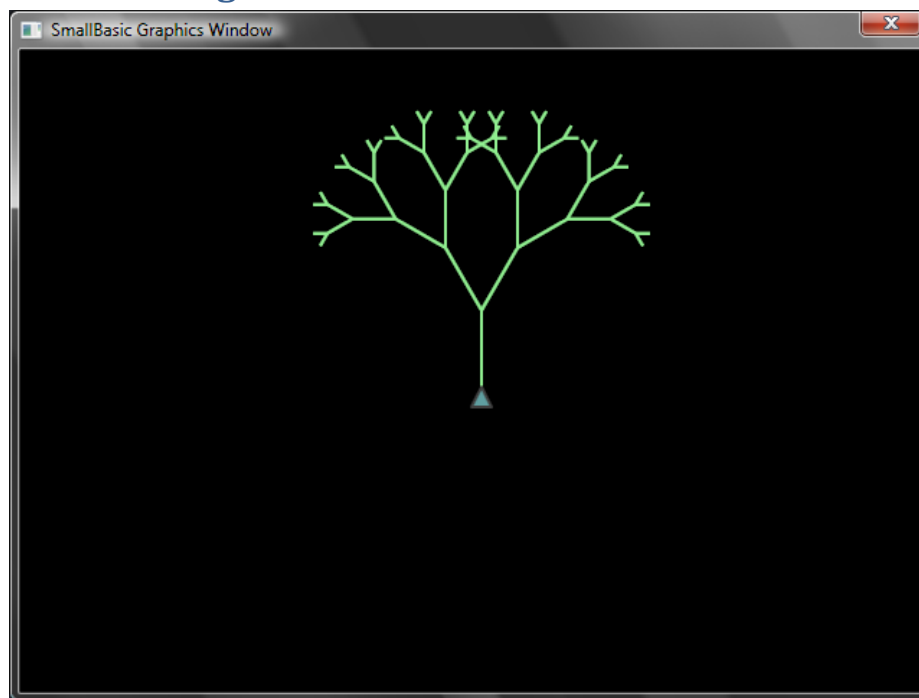


Figura 578 – Tartaruga desenhando um fractal em forma de árvore

```
angulo = 30  
delta = 10  
distancia = 60  
Turtle.Speed = 9
```

```

GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
DrawTree()

Sub DrawTree
    If (distancia > 0) Then
        Turtle.Move(distancia)
        Turtle.Turn(angulo)

        Stack.PushValue("distancia", distancia)
        distancia = distancia - delta
        DrawTree()
        Turtle.Turn(-angulo * 2)
        DrawTree()
        Turtle.Turn(angulo)
        distancia = Stack.PopValue("distancia")

        Turtle.Move(-distancia)
    EndIf
EndSub

```

## Fotos do Flickr

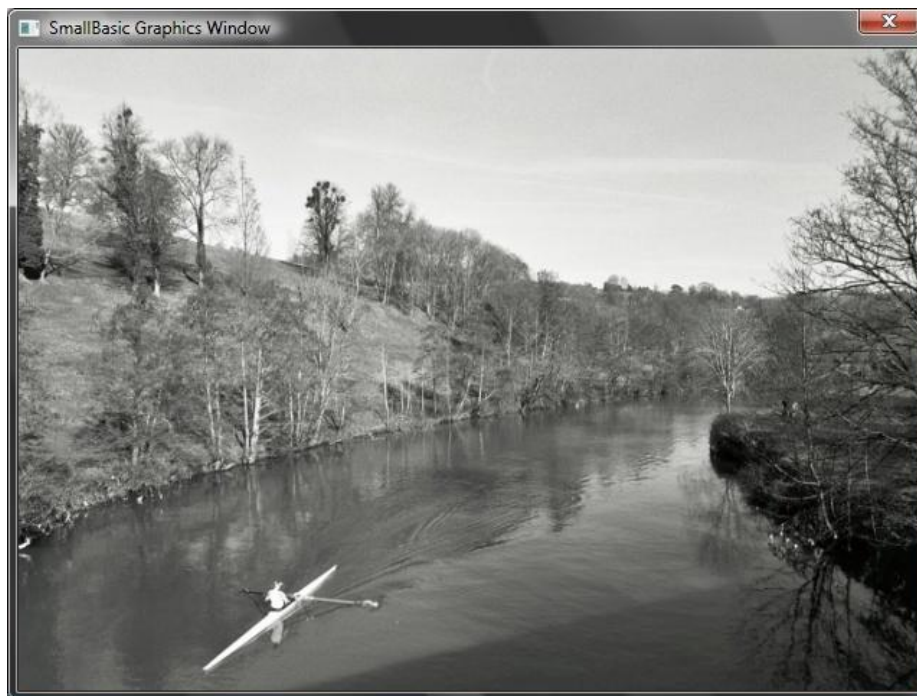


Figura 589 - Recuperando fotografias do Flickr

```
GraphicsWindow.BackgroundColor = "Black"  
GraphicsWindow.MouseDown = OnMouseDown  
  
Sub OnMouseDown  
    foto = Flickr.GetRandomPicture("montanhas, rio")  
    GraphicsWindow.DrawResizedImage(foto, 0, 0, 640, 480)  
EndSub
```

## Papel de Parede da Área de Trabalho Dinâmico

```
For i = 1 To 10  
    foto = Flickr.GetRandomPicture("montanhas")  
    Desktop.SetWallPaper(foto)  
    Program.Delay(10000)  
EndFor
```

## O Jogo de Paddle

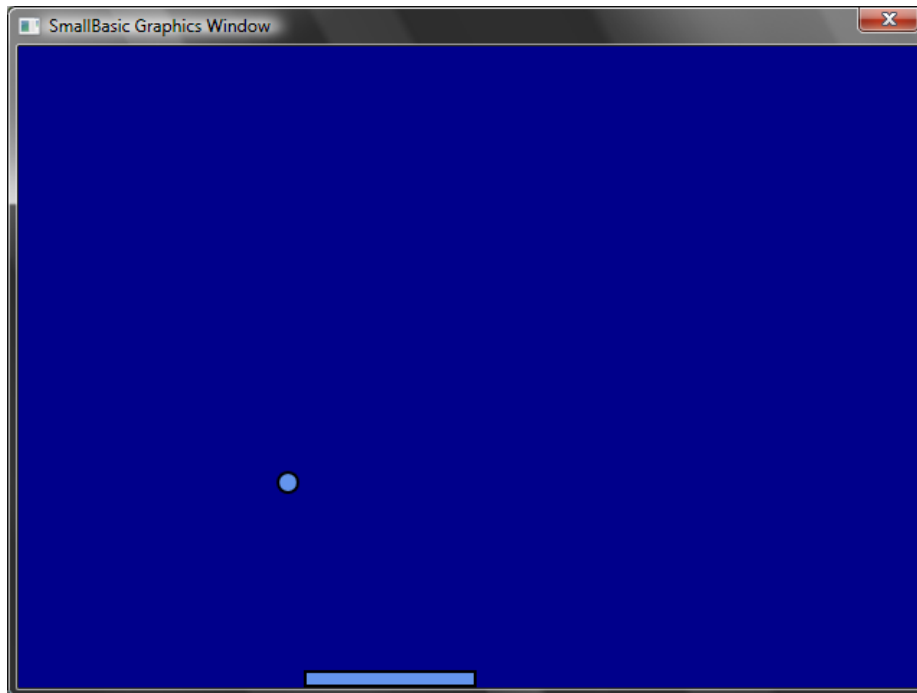


Figura 59 – O Jogo de Paddle

```
GraphicsWindow.BackgroundColor = "DarkBlue"  
paddle = Shapes.AddRectangle(120, 12)  
bola = Shapes.AddEllipse(16, 16)
```

```

GraphicsWindow.MouseMove = OnMouseMove

x = 0
y = 0
deltaX = 1
deltaY = 1

LoopExecucao:
    x = x + deltaX
    y = y + deltaY

    gw = GraphicsWindow.Width
    gh = GraphicsWindow.Height
    If (x >= gw - 16 or x <= 0) Then
        deltaX = -deltaX
    EndIf
    If (y <= 0) Then
        deltaY = -deltaY
    EndIf

    padX = Shapes.GetLeft(paddle)
    If (y = gh - 28 and x >= padX and x <= padX + 120) Then
        deltaY = -deltaY
    EndIf

    Shapes.Move(bola, x, y)
    Program.Delay(5)

    If (y < gh) Then
        Goto LoopExecucao
    EndIf

GraphicsWindow.ShowMessage("Você Perdeu ", "Paddle")

Sub OnMouseMove
    paddleX = GraphicsWindow.MouseX
    Shapes.Move(paddle, paddleX - 60, GraphicsWindow.Height - 12)
EndSub

```

Aqui está uma lista de nomes de cores suportadas pelo Small Basic, classificadas pela sua matiz.

### Vermelhos

IndianRed	#CD5C5C
LightCoral	#F08080
Salmon	#FA8072
DarkSalmon	#E9967A
LightSalmon	#FFA07A
Crimson	#DC143C
Red	#FF0000
FireBrick	#B22222
DarkRed	#8B0000

### Rosas

Pink	#FFC0CB
LightPink	#FFB6C1
HotPink	#FF69B4
DeepPink	#FF1493
MediumVioletRed	#C71585
PaleVioletRed	#DB7093

### Laranjas

LightSalmon	#FFA07A
Coral	#FF7F50
Tomato	#FF6347
OrangeRed	#FF4500
DarkOrange	#FF8C00
Orange	#FFA500

### Amarelos

Gold	#FFD700
Yellow	#FFFF00
LightYellow	#FFFFE0
LemonChiffon	#FFFACD
LightGoldenrodYellow	#FAFAD2
PapayaWhip	#FFEFD5
Moccasin	#FFE4B5
PeachPuff	#FFDAB9



PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

## Roxos

Lavender	#E6E6FA
Thistle	#D8BFD8
Plum	#DDA0DD
Violet	#EE82EE
Orchid	#DA70D6
Fuchsia	#FF00FF
Magenta	#FF00FF
MediumOrchid	#BA55D3
MediumPurple	#9370DB
BlueViolet	#8A2BE2
DarkViolet	#9400D3
DarkOrchid	#9932CC
DarkMagenta	#8B008B
Purple	#800080
Indigo	#4B0082
SlateBlue	#6A5ACD
DarkSlateBlue	#483D8B
MediumSlateBlue	#7B68EE

## Verdes

GreenYellow	#ADFF2F
Chartreuse	#7FFF00
LawnGreen	#7CFC00
Lime	#00FF00
LimeGreen	#32CD32
PaleGreen	#98FB98
LightGreen	#90EE90

MediumSpringGreen	#00FA9A
SpringGreen	#00FF7F
MediumSeaGreen	#3CB371
SeaGreen	#2E8B57
ForestGreen	#228B22
Green	#008000
DarkGreen	#006400
YellowGreen	#9ACD32
OliveDrab	#6B8E23
Olive	#808000
DarkOliveGreen	#556B2F
MediumAquamarine	#66CDAA
DarkSeaGreen	#8FBC8F
LightSeaGreen	#20B2AA
DarkCyan	#008B8B
Teal	#008080

## Azuís

Aqua	#00FFFF
Cyan	#00FFFF
LightCyan	#E0FFFF
PaleTurquoise	#AFEEEE
Aquamarine	#7FFFD4
Turquoise	#40E0D0
MediumTurquoise	#48D1CC
DarkTurquoise	#00CED1
CadetBlue	#5F9EA0
SteelBlue	#4682B4
LightSteelBlue	#B0C4DE
PowderBlue	#B0E0E6
LightBlue	#ADD8E6
SkyBlue	#87CEEB

LightSkyBlue	#87CEFA
DeepSkyBlue	#00BFFF
DodgerBlue	#1E90FF
CornflowerBlue	#6495ED
MediumSlateBlue	#7B68EE
RoyalBlue	#4169E1
Blue	#0000FF
MediumBlue	#0000CD
DarkBlue	#00008B
Navy	#000080
MidnightBlue	#191970

## Marrons

Cornsilk	#FFF8DC
BlanchedAlmond	#FFEBCD
Bisque	#FFE4C4
NavajoWhite	#FFDEAD
Wheat	#F5DEB3
BurlyWood	#DEB887
Tan	#D2B48C
RosyBrown	#BC8F8F
SandyBrown	#F4A460
Goldenrod	#DAA520
DarkGoldenrod	#B8860B
Peru	#CD853F
Chocolate	#D2691E
SaddleBrown	#8B4513
Sienna	#A0522D
Brown	#A52A2A
Maroon	#800000

## Branços

White	#FFFFFF
Snow	#FFFAFA
Honeydew	#F0FFF0
MintCream	#F5FFFA
Azure	#F0FFFF
AliceBlue	#F0F8FF
GhostWhite	#F8F8FF
WhiteSmoke	#F5F5F5
Seashell	#FFF5EE
Beige	#F5F5DC
OldLace	#FDF5E6
FloralWhite	#FFFAF0
Ivory	#FFFFF0
AntiqueWhite	#FAEBD7
Linen	#FAF0E6
LavenderBlush	#FFF0F5
MistyRose	#FFE4E1

## Cinzas

Gainsboro	#DCDCDC
LightGray	#D3D3D3
Silver	#C0C0C0
DarkGray	#A9A9A9
Gray	#808080
DimGray	#696969
LightSlateGray	#778899
SlateGray	#708090
DarkSlateGray	#2F4F4F
Black	#000000

