# PROGRAMANDO COM GAMES

Programar computadores é um desafio que muitos podem curtir, e essa experiência pode ser ainda mais divertida, pois há games que nos auxiliam no aprendizado da programação.

Um exemplo é o jogo *Labirinto*, do site Blockly Games, que disponibiliza gratuitamente vários jogos para quem está aprendendo a programar. Nesse game, o jogador enfrenta dez desafios, nos quais deve guiar o personagem pelo mapa usando blocos de comandos.

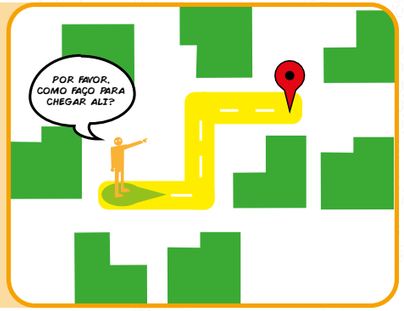
Existem, ainda, outros jogos voltados para diferentes faixas etárias, acessíveis por meio de um computador conectado à internet. Confira a seleção que fizemos.


**LABIRINTO CLÁSSICO:** para crianças menores, há o site Hora do Código. Nele, há um jogo chamado *Labirinto clássico*, que, na verdade, é um jogo de programação simples baseado no jogo de sucesso *Angry Birds*.

Link do jogo: https://studio.code.org/hoc/1

**LABIRINTO:** na programação em blocos do jogo *Labirinto*, do site Blockly Games, você movimenta o personagem para cumprir as missões enquanto aprende noções de programação.

Link do site: https://blockly.games/

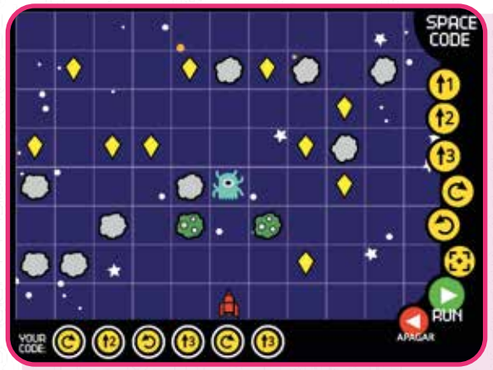




**MINECRAFT:** semelhante ao *Labirinto clássico*, o site Hora do Código também tem um jogo no qual você controla personagens do *Minecraft*, cumprindo missões como tosquiar ovelhas, pescar e construir barcos e casas.

Link do jogo: https://code.org/minecraft






**SPACE CODE:** com comandos simples, como **Frente 2**, **Girar para a direita** e **Atirar**, *Space Code* é um game para crianças a partir dos seis anos exercitarem a capacidade de planejamento, a antecipação de eventos próximos, a orientação espacial, além de aprenderem as noções básicas de programação de computadores e de robôs. Tudo isso controlando uma nave para coletar cristais no espaço enquanto um monstro espacial tenta atrapalhar. Jogue você também e não deixe que ele o alcance!

Link do jogo: scratch.mit.edu/projects/499998800

## JOGOS DE TABULEIRO

Nos dias em que faltar energia ou quando você não tiver acesso a um computador ou à internet, isso não será um problema. É possível aprender noções de programação com jogos de tabuleiro, o que pode ser bem divertido. Um dos mais famosos é o *Robot Turtles*.

**ROBOT TURTLES:** esse jogo foi desenvolvido para crianças entre três e oito anos. Os jogadores precisam mover as tartarugas por um labirinto, em busca de um cristal. É preciso programar as tartarugas com cartas do tipo "vire à esquerda", "vá para a frente" ou "dispare o laser".

Link do jogo: http://robotturtles.com/


Não encontramos jogos de tabuleiro desse tipo já lançados em língua portuguesa, mas não se preocupe. Na terceira parte deste livro, há uma série de desafios chamada Code Mission, que usa apenas este material, papel e lápis para ensinar noções de programação. Você pode se basear nele para construir seu próprio jogo de tabuleiro.

# LIDANDO COM ERROS

Antigamente, os computadores eram bem grandes. Por isso, era comum que insetos entrassem neles e causassem danos e mau funcionamento. Desde então, usa-se o termo "bug" ("inseto", em inglês) para se referir a um erro em programas de computador.

No início de qualquer aprendizado, os erros são muito comuns. Na programação, não é diferente. É possível cometer erros até em programas simples. Geralmente, há dois tipos de erro de programação: de sintaxe e de tipo.

## LIDANDO COM ERROS NO BASIC

Os erros de sintaxe ocorrem quando você digita alguma palavra errada. Veja a linha abaixo:

```
pront "Bom dia"
```

Talvez você cometa um erro e digite **pront** no lugar de **print**. Ao executar o programa, o BASIC! mostrará na tela a seguinte mensagem:

```
Syntax Error
pront "Bom dia"
```

O programa mostra o tipo de erro e em qual linha ele se encontra. Isso é útil para você poder identificar onde o erro está e, então, corrigi-lo. Agora, veja o seguinte código:


```
input "Qual é o seu nome?", a
print "Bom dia" + a
```

O primeiro problema desse código é a variável numérica **a** em relação à mensagem do comando **input**. A mensagem do **input** pergunta o nome da pessoa. Porém, ao tentar guardar esse nome em uma variável numérica, o BASIC! permitirá inserir apenas números! Ele mostrará o teclado numérico para você.

A não ser que seu nome seja um número, será difícil responder a essa pergunta!



Além disso, depois que você inserir o número, a tela exibirá a mensagem "Extraneous characters in line: a". Isso significa "Caracteres estranhos na linha: a".

Ou seja, o programa está dizendo que a variável **a** é um caractere estranho, pois não é possível unir um texto ("Bom dia") a uma variável numérica. Isso provoca um erro de tipo, pois um texto só pode ser unido a outro texto ou a uma variável de tipo textual.

Sabemos que, em Basic, uma variável textual (String) é rotulada corretamente com um $ no final, como em **A$** ou **nome$**. Portanto, o programa corrigido fica assim:

```
input "Qual é o seu nome?", A$
print "Bom dia" + A$
```

Ao usar uma variável textual, o BASIC! entende que receberá um texto e exibirá o teclado completo (o alfanumérico, com letras e números).

Lembre-se de que números inseridos em uma variável textual são considerados parte de um texto. Não é possível realizar cálculos com eles, como no código abaixo:

```
A$ = "5"
B$ = "10"
print A$+B$
```

O resultado desse código será "510", não 15, pois quando somamos textos o programa apenas junta os conteúdos. Para calcular de fato, sempre use variáveis numéricas, como no código abaixo revisado:

```
A = 5
B = 10
print A+B
```

Não se preocupe se houver um ou mais erros no programa. Apenas preste atenção no que o BASIC! alerta e tente fazer as correções tranquilamente.

## LIDANDO COM ERROS EM PYTHON

Em Python, caso você se esqueça de fechar as aspas em **print (“Bom dia)** e rodar o código, o sistema vai mostrar uma mensagem como esta:

```
File “main.py”, line 1
    print(“Bom dia)
                  ^
SyntaxError: EOL while scanning string literal
```

Ele dirá a linha onde o erro se encontra (line 1) e onde faltam as aspas.

É possível que, ao escrever errado o nome de uma função, como **pront** em vez de **print**, a mensagem seja semelhante a esta:

```
File “main.py”, line 1, in <module>
    pront(“Bom dia”)
NameError: name ‘pront’ is not defined
```

O sistema alerta sobre um erro de nome afirmando que **pront** não está definido. Isso ocorre porque, em Python, você pode criar instruções com o nome que quiser, usando a instrução **def**, como veremos mais adiante, no código do Caçador de Robôs.

### Variáveis de tipos diferentes em Python

Em Python, uma variável será textual ou numérica, de acordo com o que guardamos nela.

Se declararmos que **a = 10**, ela será numérica; porém, se declaramos que **a = “10”** ou que **a = “maracujá”**, ela será textual (String). A diferença está no uso das aspas. Agora, veja o código abaixo:

```
a = input (“Digite um número: ”)
b = input (“Digite outro número: ”)
print (“A soma é: ”)
print (a + b)
```



Em Python, o comando **input( )** recebe sempre uma variável do tipo String (ou seja, textual). No programa da página anterior, se você inserir os valores 4 e 5, o resultado será 45, pois o sinal de + faz o Python juntar o conteúdo de variáveis de texto. Para somar matematicamente, você precisa transformar a variável de texto em variável numérica usando a função **int( )**, conforme o código abaixo.

```
a = input (“Digite um número”)
a = int(a)
b = input (“Digite outro número”)
b = int(b)
print (“A soma é:”)
print (a + b)
```

### Erro de indentação em Python

Outro erro comum em Python é o erro de indentação, que é o espaçamento que o programador dá antes de iniciar uma linha de código.

Na maioria das linguagens, esse espaçamento é feito apenas para deixar o código mais legível e organizado visualmente. No caso do Python, é uma necessidade. Por exemplo, quando usamos o comando de repetição **for**, o trecho do programa que será repetido deve estar com recuo em relação ao **for**. Veja o código a seguir como exemplo:

```
for x in range (50):
   print (x)
```

O programa acima funcionará normalmente, mas o de baixo retornará com um erro.

```
for x in range (50):
print (x)

File “main.py”, line 2
    print (x)
    ^
IndentationError: expected an indented block
```

Isso ocorrerá, pois, no segundo código, o **print** não está com o espaço necessário.

Também é preciso indentar trechos do código quando lidamos com tomadas de decisão. Veja as estruturas condicionais **if** abaixo:

```
a = input (“Qual sua última nota em história?”)
a = int(a)
if a>6:
  print (“Parabéns!”)
  b = input (“Você estudou muito?”)
  print (b + “!? Entendi.”)
if a<7:
  print (“Precisa estudar mais.”)
  b = input (“Qual a matéria mais fácil para você?”)
  print (“Entendi. ” + b + “é interessante!”)
print (“Foi bom conversar. Até logo!”)
```

Por exemplo, caso você retire o espaçamento da quinta linha, o Python pensará que essa linha está fora do `if a>6`.

Alguns aplicativos de programação Python já colocam esse espaçamento automaticamente após você inserir alguns comandos, em especial após laços de repetição (**for**, **while**, etc.) ou estruturas condicionais (**if**, **else**).



# O CELULAR DORMINHOCO

Este código em Basic demonstra como você pode acessar os sensores de seu celular Android. Programe o código abaixo e o execute.

Ao digitar esse código, note que as linhas quebradas devem ser digitadas em uma linha única. Caso contrário, vai gerar um erro na hora da execução.

```
tts.init
sensors.open 1
pause 2000
wakelock 1

do
 sensors.read 1,x,y,z
print y

if y>9 then tts.speak “Estou de pé. Por favor, coloque-me deitado. Estou com sono”
if y>1 & y<9 then tts.speak “Estou inclinado. Não consigo dormir desse jeito”
if y<-1 then tts.speak “Estou ficando de cabeça para baixo. Ninguém merece”
if y>-1 & y<1 then print “Boa noite. ZZZZZ…”

pause 3000
until 0

end
```