



## Curso de Introdução à Programação de Jogos em Python

OFERTA PILOTO 2017

Material de estudo

123 8.6 ×|° “” [ , ] ( , ) { : } f x f py ↵ ↻ 📦 📁

## SUMÁRIO

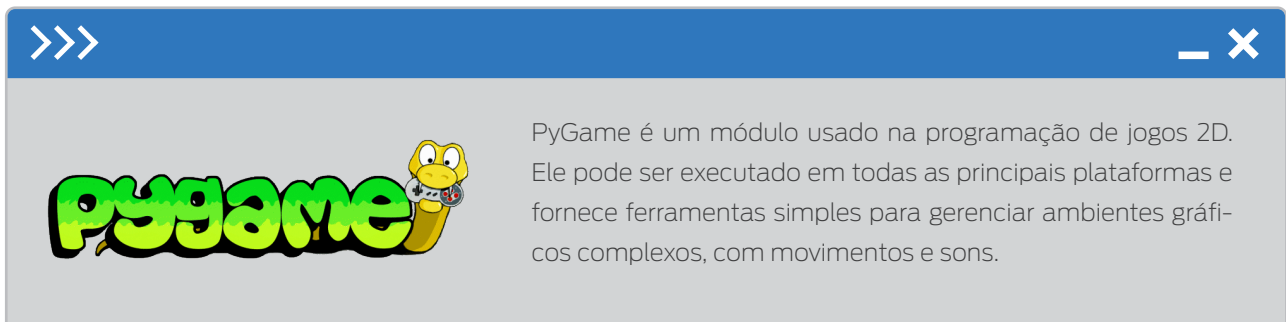
Introdução.....	3
Programa 1: Criando uma Janela.....	5
Programa 2: Figuras e Texto.....	12
Programa 3: Animação.....	24
Programa 4: Colisão.....	34
Programa 5: Teclado e Mouse.....	42
Programa 6: Imagem e Som.....	53
Programa 7: Jogo.....	64





Na primeira parte do curso adquirimos algumas noções de programação em Python. Lá vimos que programar consiste, basicamente, em trabalhar com dados. Vimos como os dados eram manipulados através de certas estruturas como loops, condicionais, funções, etc. Agora aplicaremos esse conhecimento para o verdadeiro objetivo do curso, que é fazer jogos. Mas quando pensamos em jogos nos vêm à cabeça aspectos como figuras, sons, animações, controle de personagens, etc. Para isto precisaremos de outras ferramentas. Neste caso, usaremos o Pygame.

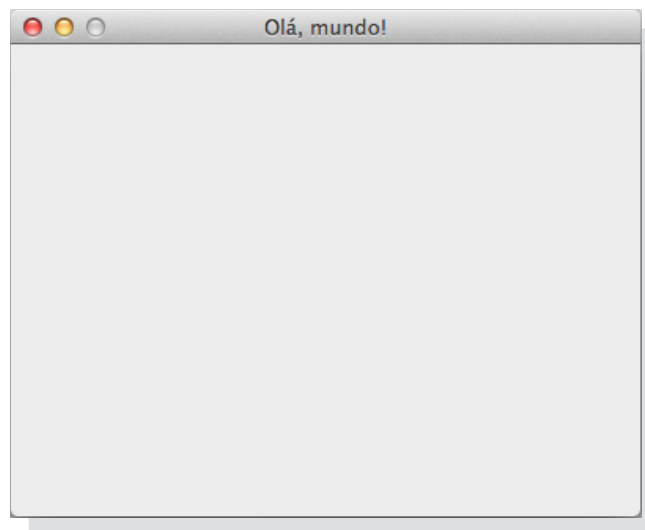




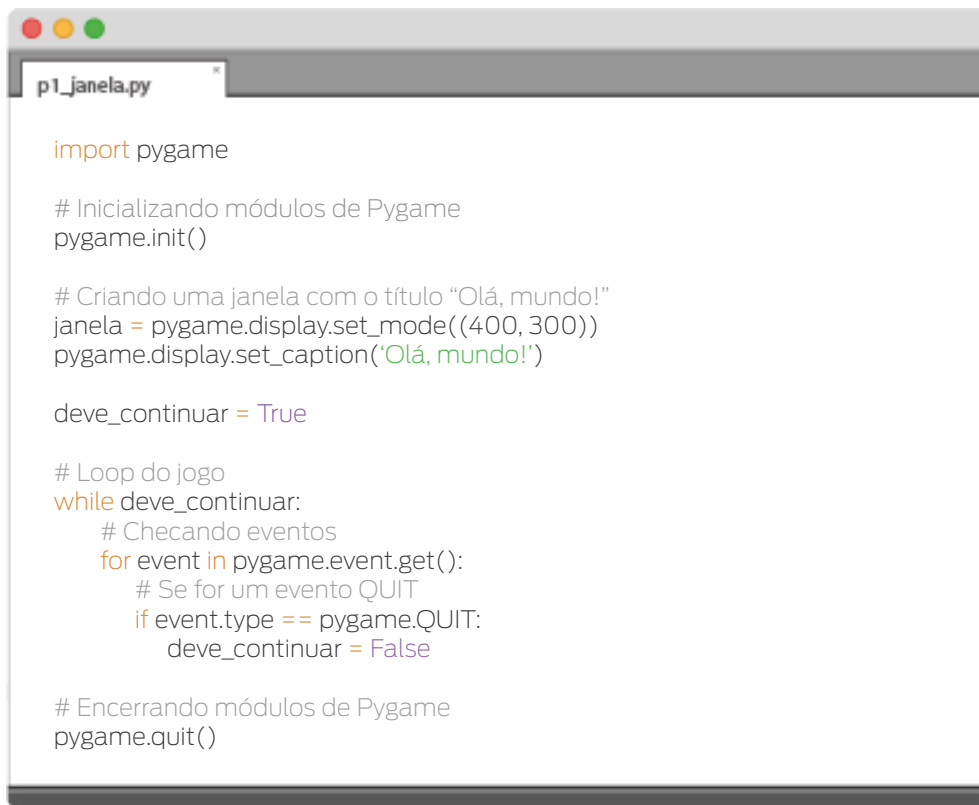
O módulo Pygame inclui vários outros módulos com funcionalidades para a parte gráfica, som, mouse e muitos outros. Neste tutorial abordaremos os módulos e funções básicas de Pygame. Para isto, faremos vários programas que nos mostrarão como utilizá-los.

### Relacionando alguns módulos com suas atribuições.

<i>cursors</i>	carrega imagens de cursores (N.T. mouse), inclui cursores padrão
<i>display</i>	controla a exibição da janela ou tela
<i>draw</i>	desenha formas simples sobre uma Surface
<i>event</i>	controla eventos e fila de eventos
<i>font</i>	disponibiliza fontes
<i>image</i>	salva e carrega imagens
<i>joystick</i>	controla dispositivos joystick
<i>key</i>	controla o teclado
<i>locals</i>	contém constantes de Pygame
<i>mixer</i>	carrega e executa sons
<i>mouse</i>	controla o mouse
<i>movie</i>	executa filmes no formato mpeg
<i>time</i>	controla a temporização
<i>transform</i>	permite redimensionar e mudar a orientação de imagens



Em nosso primeiro programa vamos criar uma janela. Esta janela ficará aberta até o usuário clicar no “x” no canto superior da janela. O código está no arquivo `pl_janela.py`.



```
import pygame

# Inicializando módulos de Pygame
pygame.init()

# Criando uma janela com o título "Olá, mundo!"
janela = pygame.display.set_mode((400, 300))
pygame.display.set_caption('Olá, mundo!')

deve_continuar = True

# Loop do jogo
while deve_continuar:
    # Checando eventos
    for event in pygame.event.get():
        # Se for um evento QUIT
        if event.type == pygame.QUIT:
            deve_continuar = False

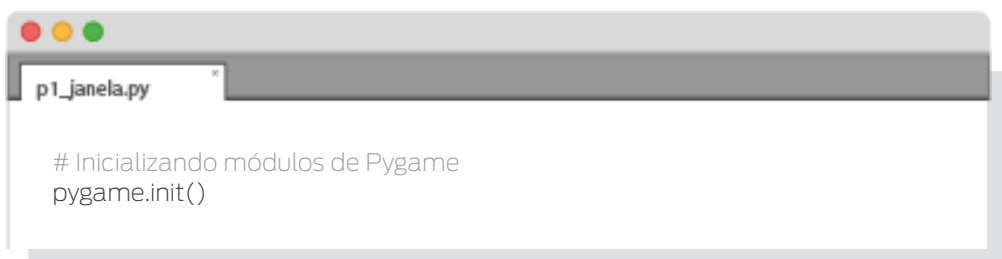
# Encerrando módulos de Pygame
pygame.quit()
```

Importamos o módulo *pygame* para que possamos usar os módulos e as funções contidas.



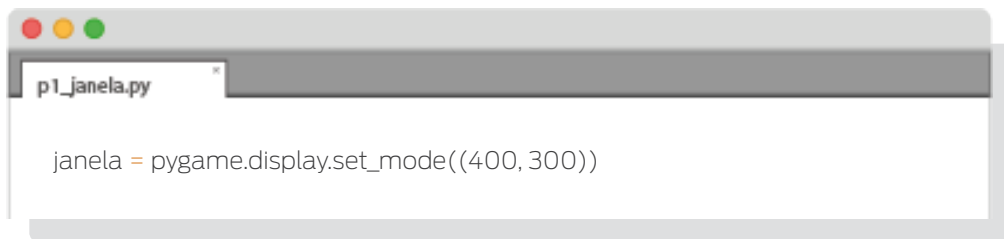
```
import pygame
```

*Pygame* é formado por outros módulos. Para poder usá-los é necessário inicializá-los. Isto é conseguido com *pygame.init()*.



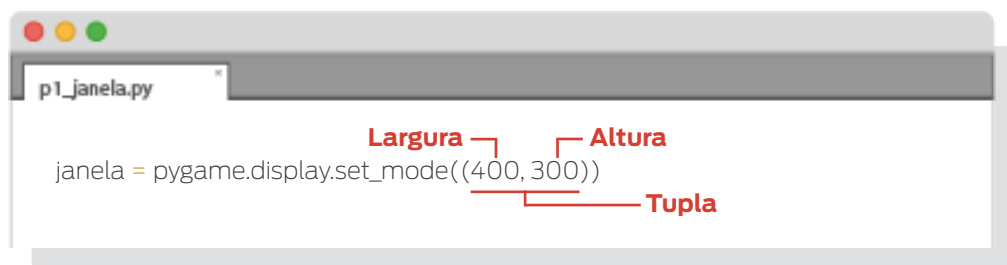
```
# Inicializando módulos de Pygame
pygame.init()
```

Continuando...

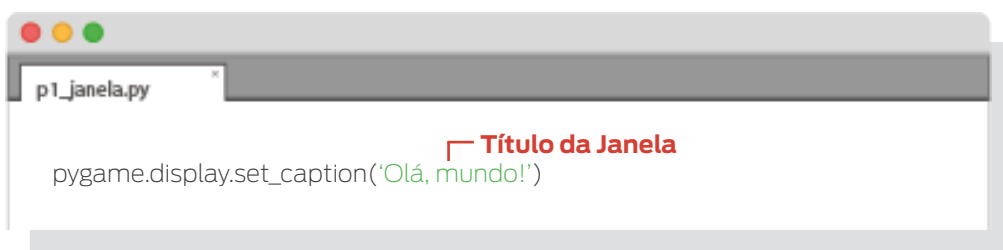


...criamos a janela com a função `set_mode()` do módulo `display`. Este módulo é um dos que está dentro do módulo `pygame`.

A função `set_mode()` cria um objeto `pygame.Surface` (ver quadro sobre objetos) e tem como parâmetro uma tupla com dois inteiros - (400,300) neste exemplo - referentes à largura e altura da janela, em pixels.




A seguir damos um título à janela com a função `pygame.display.set_caption()`.





### Objetos, métodos e atributos

Objetos, métodos e atributos serão palavras muito utilizadas a partir de agora neste tutorial. Já usamos métodos anteriormente, sem mencioná-los, como por exemplo no uso de listas. Vejamos o seguinte exemplo:



```
lista_var = [1,2,3]
lista_var.append(4)
```

Podemos ver que *lista\_var* contém uma lista como valor e nela adicionamos “.append” passando 4 como argumento.

Um **método** é como uma função que está acoplada a um valor, neste caso o método *append()* está acoplado ao valor *lista\_var*. O método *append()* é um dos muitos métodos de listas e, como explicado quando vimos listas, adiciona um elemento no final da lista.

Já um **objeto** nada mais é do que um valor que contém métodos (e atributos). No exemplo dado o objeto é *lista\_var*.

Assim como um método é uma função associada a um objeto, um **atributo** é uma variável associada a um objeto. Veremos vários casos de atributos ao longo deste texto.

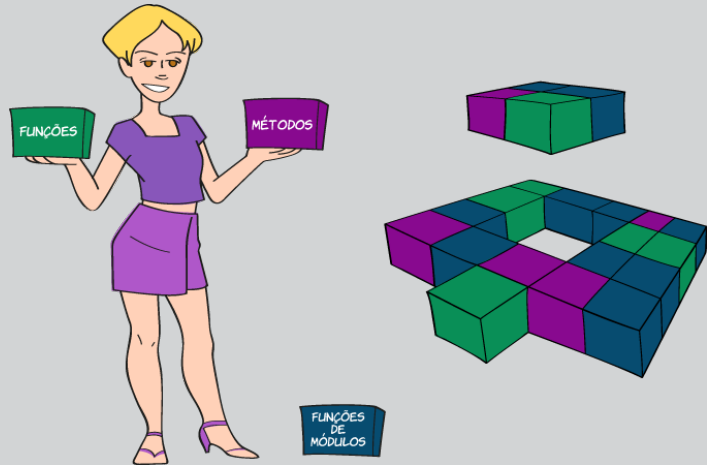




A notação de objetos é a mesma da usada com módulos. A diferença é que o módulo precisa ser importado e o objeto não.

Funções, métodos, funções de módulos... Tudo isso não é meio confuso? É sim... Vejamos alguns exemplos que poderão nos ajudar a distinguir melhor estes elementos (ou confundir-nos de vez...).

Abra o intérprete de Python (usando o IDLE ou executando o comando "python" desde um terminal) e escreva o código abaixo:



```
>>> import math

>>> lista_var = [1,2,3]

# função sum() de Python: recebe uma lista (neste caso) e devolve a soma de seus elementos
>>> sum(lista_var)      # retornará 6

# função sqrt() do módulo math: recebe um número e devolve a sua raiz quadrada
>>> math.sqrt(16)      # retornará 4.0

# método append() de listas: recebe um elemento e o adiciona no final da lista
>>> lista_var.append(4)

>>> print (lista_var)   # imprimirá [1,2,3,4]
```

Função de módulo

`math.sqrt(16)`

Módulo

Método

`lista_var.append(4)`

Objeto



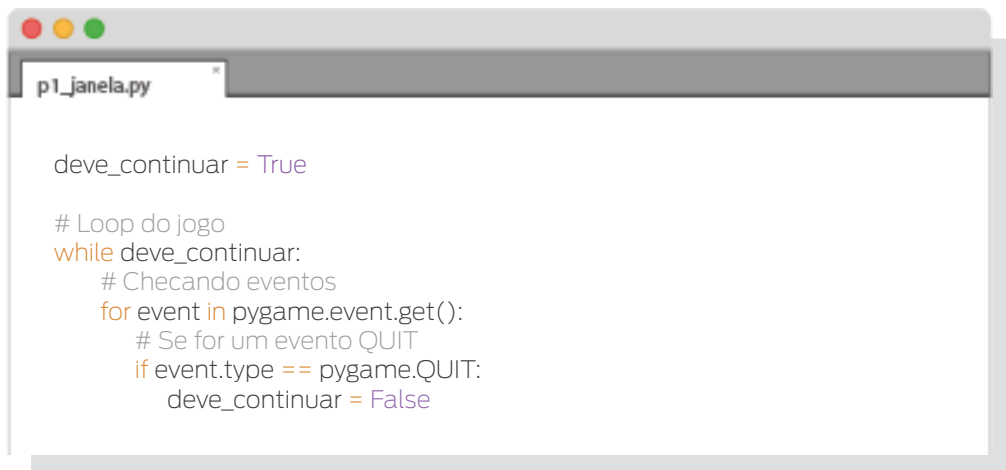
Talvez a melhor forma de lidar com esta situação seja não se preocupar muito se está sendo usado uma função de um módulo ou um método de um objeto, no próprio texto explicitaremos o que está sendo feito. O importante é entender que tanto funções como métodos podem receber dados e realizam alguma tarefa.

Antes de entrar no loop, foi criado um booleano com o nome *deve\_continuar* que nos indicará se o loop que está a seguir deve ser repetido ou não. Inicialmente damos o valor *True* para que o código dentro do loop seja executado.



```
deve_continuar = True
```

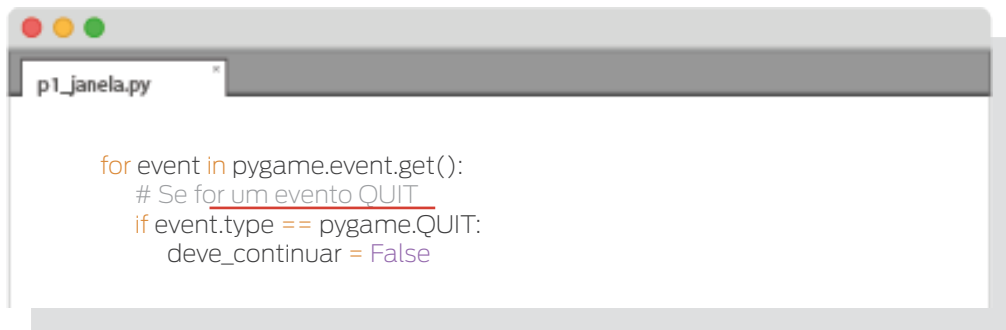
Após, temos um loop que será executado sempre que a variável *deve\_continuar* for *True*. No seu interior checamos por eventos. Eventos são gerados sempre que o usuário pressiona uma tecla ou dá um clique no mouse, por exemplo.



```
deve_continuar = True

# Loop do jogo
while deve_continuar:
    # Checando eventos
    for event in pygame.event.get():
        # Se for um evento QUIT
        if event.type == pygame.QUIT:
            deve_continuar = False
```

Chamando `pygame.event.get()` será retornado qualquer novo objeto do tipo `pygame.event.Event` que tenha sido gerado desde a chamada anterior a essa função (a cada execução deste loop `while`).

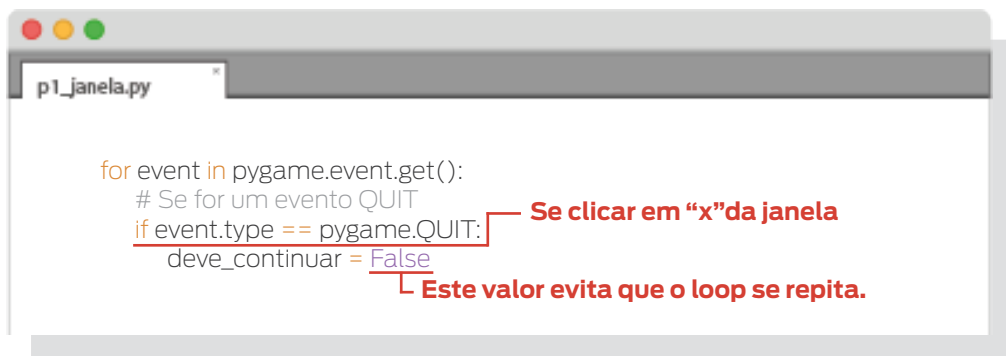


```

for event in pygame.event.get():
    # Se for um evento QUIT
    if event.type == pygame.QUIT:
        deve_continuar = False

```

Os objetos `pygame.event.Event` têm um atributo chamado `type` (tipo, em inglês). No código, checamos o tipo de cada evento. Pygame gera o evento `pygame.QUIT` (ou `QUIT`, para simplificar) quando o usuário dá um clique no “x” da janela. Este evento também é gerado se o computador é desligado e tenta terminar todos os programas que estão sendo executados. Se temos um evento deste tipo, segundo nosso código, mudamos o valor da variável `deve_continuar` para `False`, fazendo com que o loop não volte a ser executado.



```

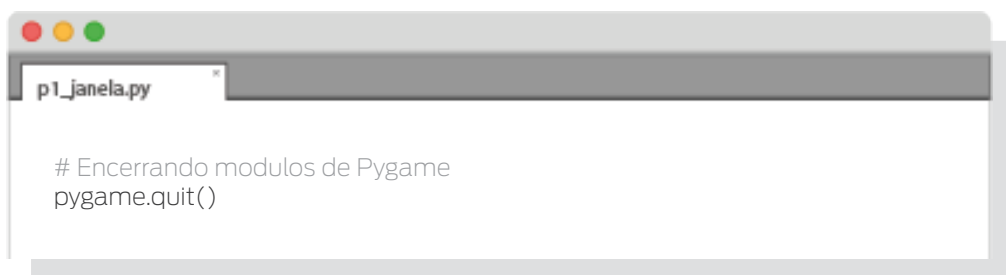
for event in pygame.event.get():
    # Se for um evento QUIT
    if event.type == pygame.QUIT:
        deve_continuar = False

```

Se clicar em “x” da janela

Este valor evita que o loop se repita.

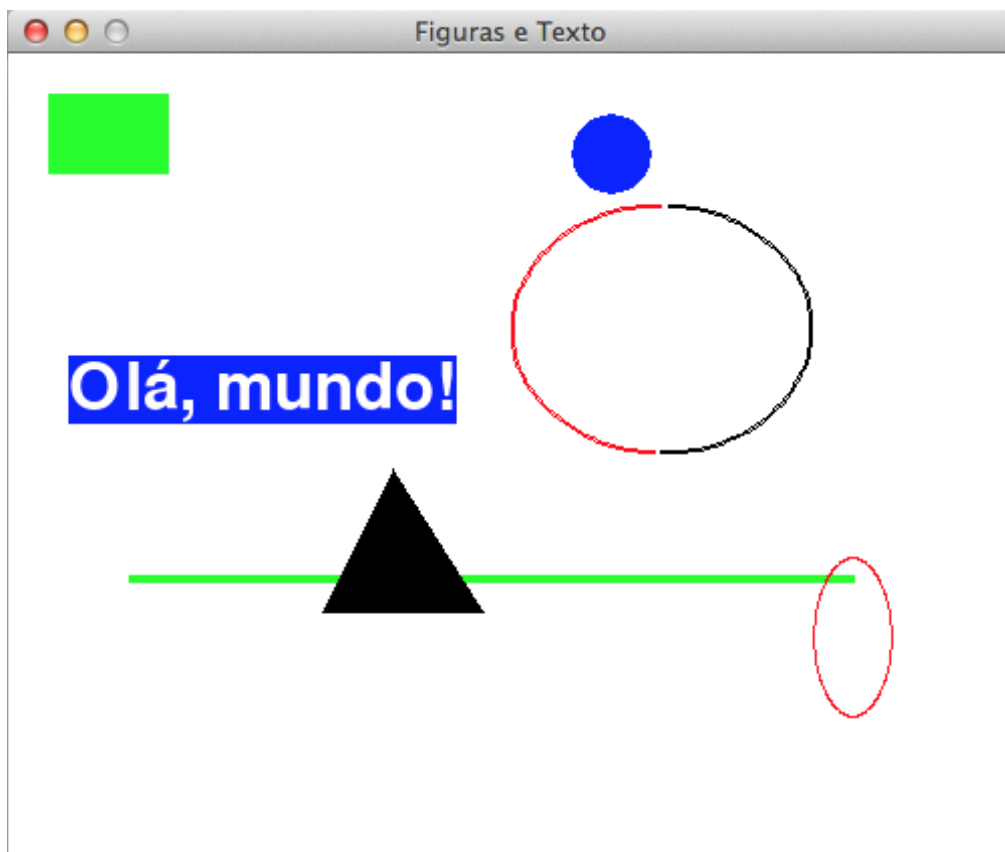
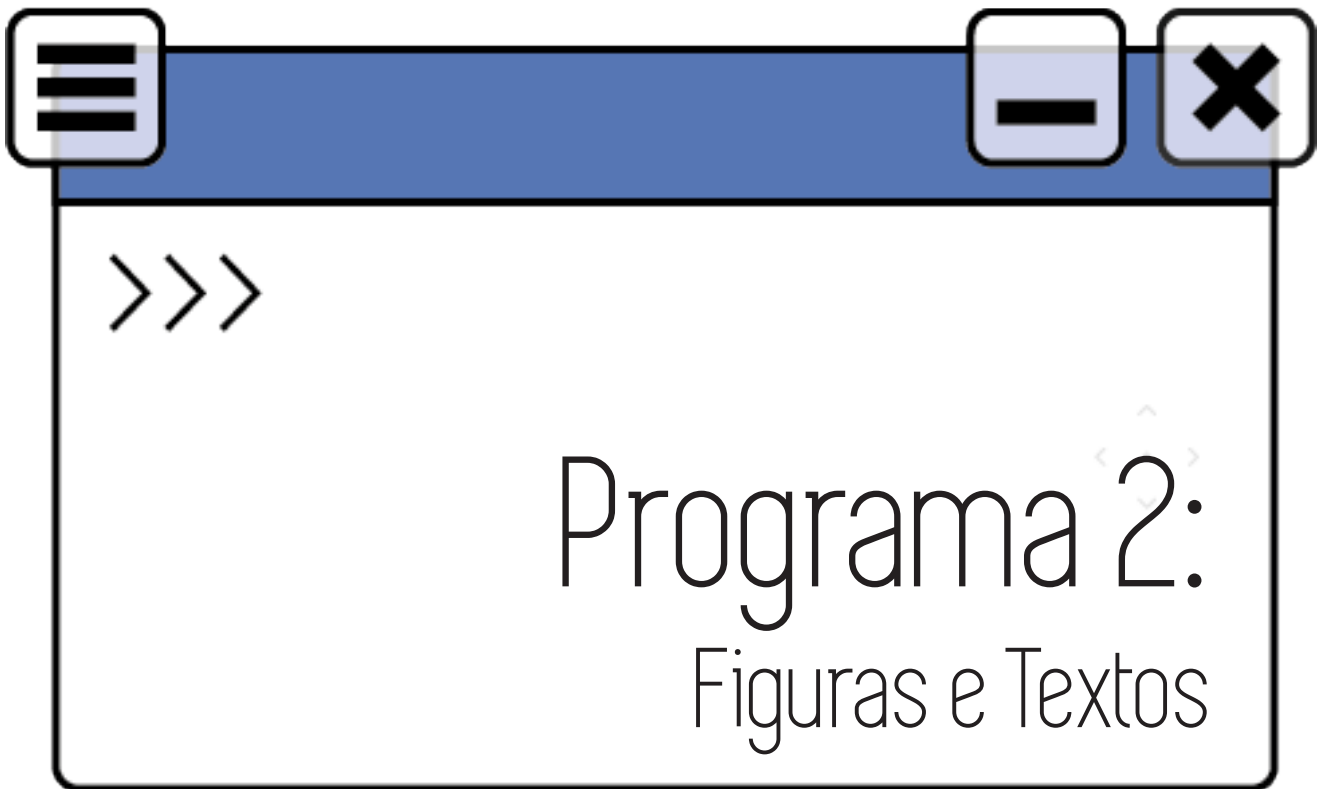
Para finalizar, chamamos a função `pygame.quit()` para fechar todos os módulos de `pygame`.



```

# Encerrando módulos de Pygame
pygame.quit()

```



No programa anterior vimos como criar uma janela. Agora veremos como adicionar algumas figuras e textos. O código se encontra no arquivo p2\_figuras.py.



```
import pygame

# Definindo as cores
PRETO = (0, 0, 0)
BRANCO = (255, 255, 255)
VERMELHO = (255, 0, 0)
VERDE = (0, 255, 0)
AZUL = (0, 0, 255)

# Definindo PI
PI = 3.1416

# Inicializamos os módulos de Pygame
pygame.init()

# Criamos a janela com título Figuras e texto
janela = pygame.display.set_mode((500, 400))
pygame.display.set_caption('Figuras e Texto')

# Preenchendo o fundo da janela com a cor branca
janela.fill(BRANCO)

# Trabalhando com texto
fonte = pygame.font.Font(None, 48)
texto = fonte.render('Olá, mundo!', True, BRANCO, AZUL)
janela.blit(texto, [30, 150])

# Desenhando figuras
pygame.draw.line(janela, VERDE, (60, 260), (420, 260), 4)
pygame.draw.polygon(janela, PRETO, ((191, 206), (236, 277), (156, 277)), 0)
pygame.draw.circle(janela, AZUL, (300, 50), 20, 0)
pygame.draw.ellipse(janela, VERMELHO, (400, 250, 40, 80), 1)
pygame.draw.rect(janela, VERDE, (20, 20, 60, 40), 0)
pygame.draw.arc(janela, VERMELHO, [250, 75, 150, 125], PI/2, 3*PI/2, 2)
pygame.draw.arc(janela, PRETO, [250, 75, 150, 125], -PI/2, PI/2, 2)

# mostrando na tela tudo o que foi desenhado
pygame.display.update()

deve_continuar = True

while deve_continuar:
    # Loop para checar eventos
    for event in pygame.event.get():
        # Condicional para sair do loop
        if event.type == pygame.QUIT:
            deve_continuar = False

pygame.quit()
```

Após importar pygame, definimos algumas cores e a variável PI. Usamos alguns nomes de variáveis em letras maiúsculas, pois queremos que estes valores sejam constantes, ou seja, em nenhum momento alteraremos o valor destas variáveis. O uso de variáveis em maiúsculas para constantes é apenas uma convenção na programação, ou seja, não é algo que deva ser feito desse jeito, mas é costume em muitas linguagens.

```
p2_figuras.py

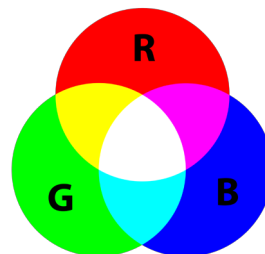
import pygame

# Definindo as cores
PRETO = (0, 0, 0)
BRANCO = (255, 255, 255)
VERMELHO = (255, 0, 0)
VERDE = (0, 255, 0)
AZUL = (0, 0, 255)

# Definindo PI
PI = 3.1416
```

Em dispositivos eletrônicos, como no caso de seu monitor, qualquer cor é formada a partir de três cores básicas:

- Vermelho (**R**ed, em ingles)
- Verde (**G**reen)
- Azul (**B**lue)

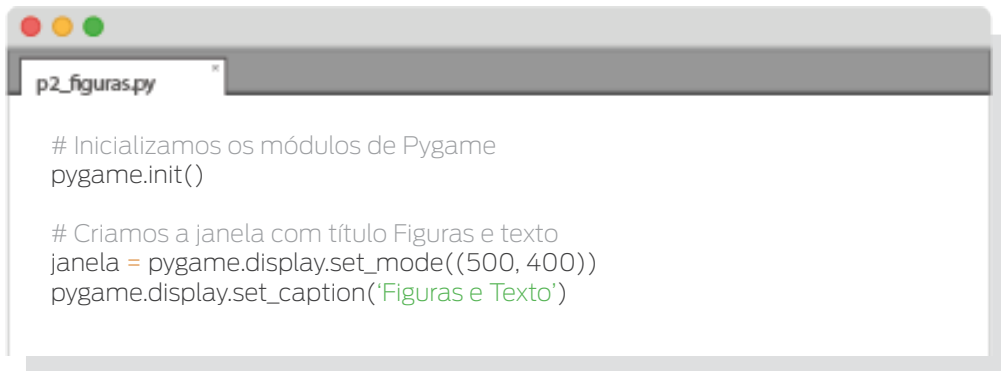


Estas cores formam o que é conhecido como sistema de cores RGB. Em pygame representamos estes valores como uma tupla de três inteiros com valores entre 0 e 255 cada um. Sendo que o primeiro valor da tupla corresponde à quantidade de vermelho, o segundo à quantidade de verde e o último à quantidade de azul.

```
p2_figuras.py

Valor de vermelho — Valor de azul
PRETO = (0, 0, 0)
          Valor de verde
```

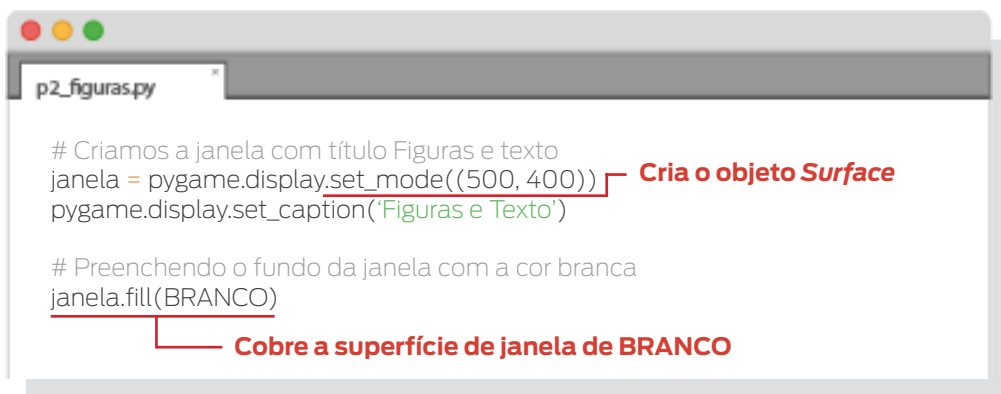
Inicializamos os módulos de pygame e criamos a janela.



```
# Inicializamos os módulos de Pygame
pygame.init()

# Criamos a janela com título Figuras e texto
janela = pygame.display.set_mode((500, 400))
pygame.display.set_caption('Figuras e Texto')
```

Como foi comentado no capítulo anterior, a função `set_mode()` do módulo `display` (que por sua vez está dentro do módulo `pygame`) cria um objeto do tipo `pygame.Surface`. Este tipo de objeto possui um método chamado `fill()`, que cobrirá completamente sua superfície com a cor passada como argumento. Neste caso, passamos a cor branca. Para facilitar a compreensão, “surface” significa “superfície” em inglês.



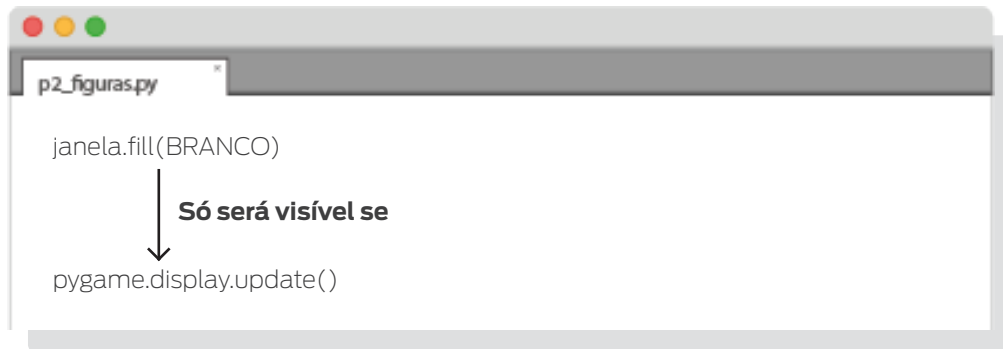
```
# Criamos a janela com título Figuras e texto
janela = pygame.display.set_mode((500, 400))
pygame.display.set_caption('Figuras e Texto')

# Preenchendo o fundo da janela com a cor branca
janela.fill(BRANCO)
```

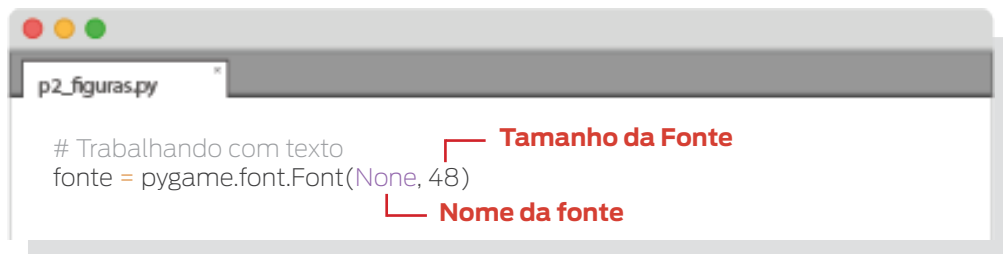
**Cria o objeto Surface**

**Cobre a superfície de janela de BRANCO**

Este método (assim como outros de desenho) mudarão o objeto *pygame.Surface* que o chamou, mas esta mudança não será, ainda, visível na tela do computador. Esta mudança será visível somente após a chamada da função *pygame.display.update()*, que será usada mais adiante.



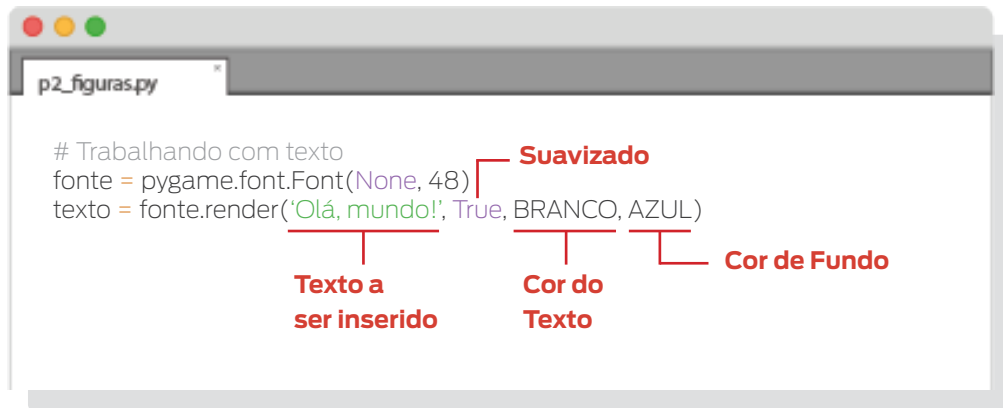
A seguir criamos um objeto do tipo *pygame.font.Font* que chamamos de *fonte*. O primeiro parâmetro é o nome da fonte a ser usada. Se passarmos *None* como argumento, será usada a fonte padrão do sistema. O segundo parâmetro é o tamanho da fonte.



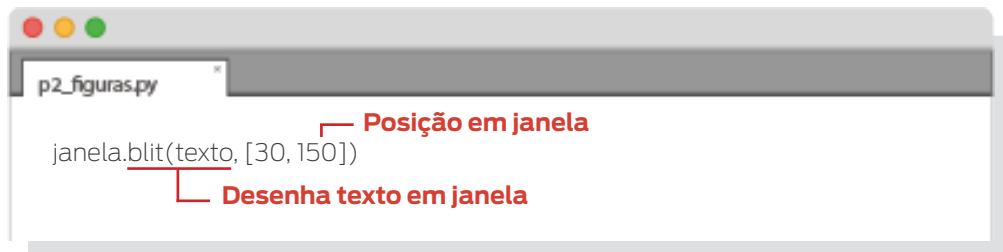
Após, é criado um outro objeto *pygame.Surface*, através do método *render()* chamado por um objeto *pygame.font.Font* (*fonte*, no caso), que conterà uma string. O primeiro parâmetro de *render()* é a string a ser inserida. O segundo parâmetro é um booleano determinando se será usado suavizado ou não. Suavizado (chamado anti-aliasing em inglês) é uma técnica que nos dá a ilusão de que uma imagem é “lisa”, “embaçando” as bordas de uma



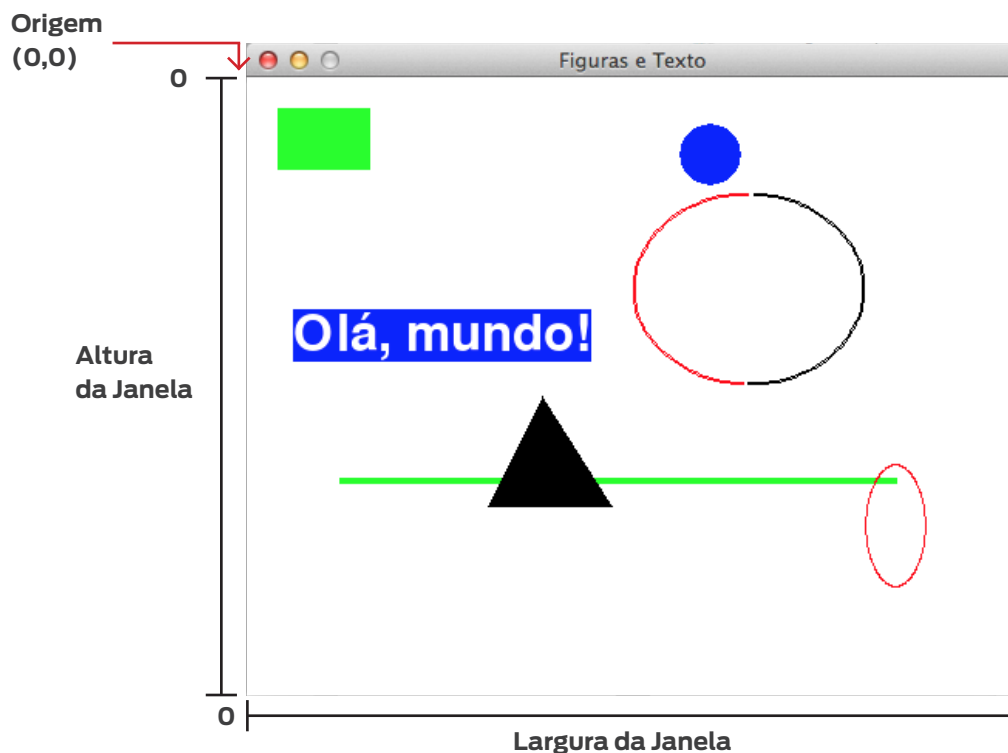
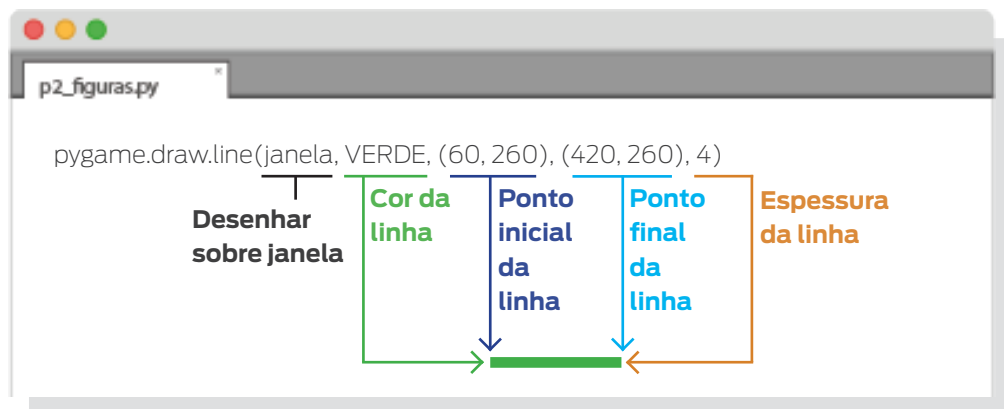
linha. No código foi passado o valor *True*, pois queremos suavizado. O terceiro parâmetro é a cor do texto e o quarto parâmetro a cor do fundo do texto. Com isto temos a imagem do texto, porém, ainda precisamos dizer onde o texto será colocado. Para isto precisamos do método *blit()*.



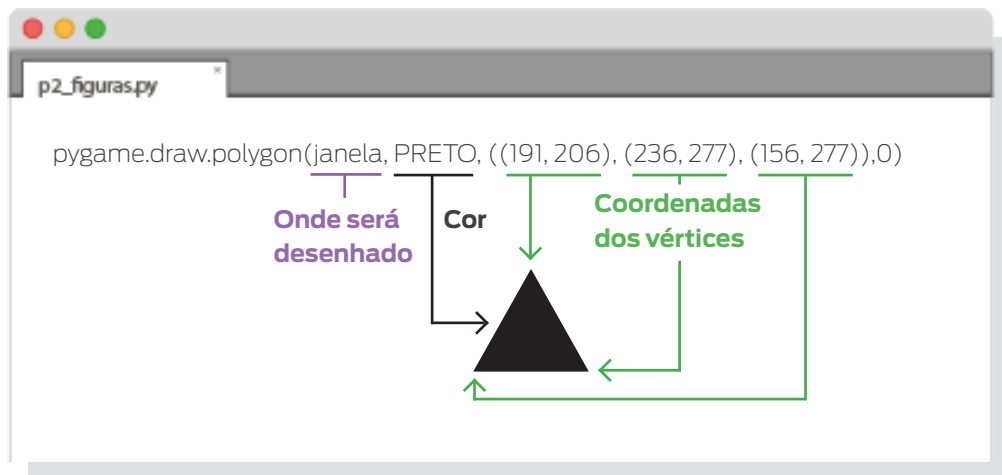
O método *blit()* desenhará o conteúdo de um objeto *pygame.Surface* (no caso, *texto*) no objeto *pygame.Surface* que chamou o método (no caso, *janela*). O segundo parâmetro é a posição, relativa ao objeto que chamou o método, onde o texto será colocado.



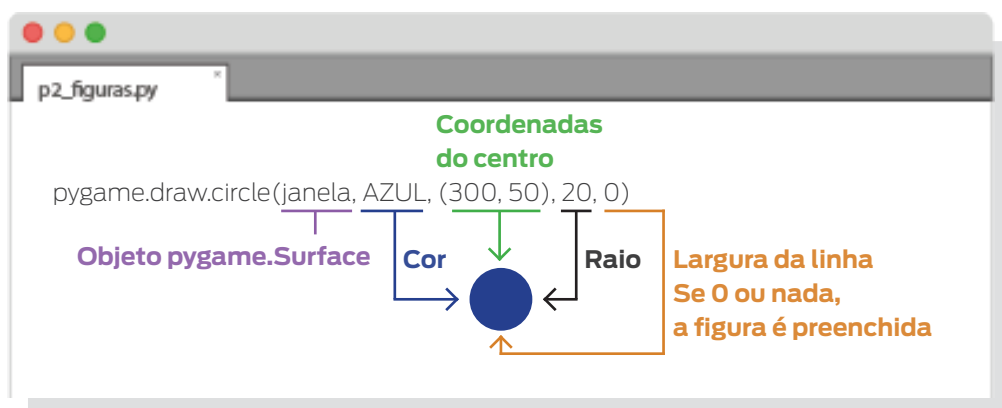
Continuando, desenhamos uma linha. Para isto, usamos a função `pygame.draw.line()`. O primeiro parâmetro desta função é o objeto `pygame.Surface` (no caso, `janela`) sobre o qual a linha será desenhada. O segundo parâmetro é a cor da linha. Depois temos as coordenadas dos pontos que serão as extremidades da linha (terceiro e quarto parâmetros). Os valores deverão ir de 0 até a largura da janela criada (da esquerda para a direita) e de 0 até a altura da janela (de cima para baixo) para que fiquem dentro da janela. A origem, o ponto (0,0), fica no canto superior esquerdo da janela. O quinto parâmetro é a largura da linha em pixels. Se a largura não for especificada, será considerada com o valor 1.



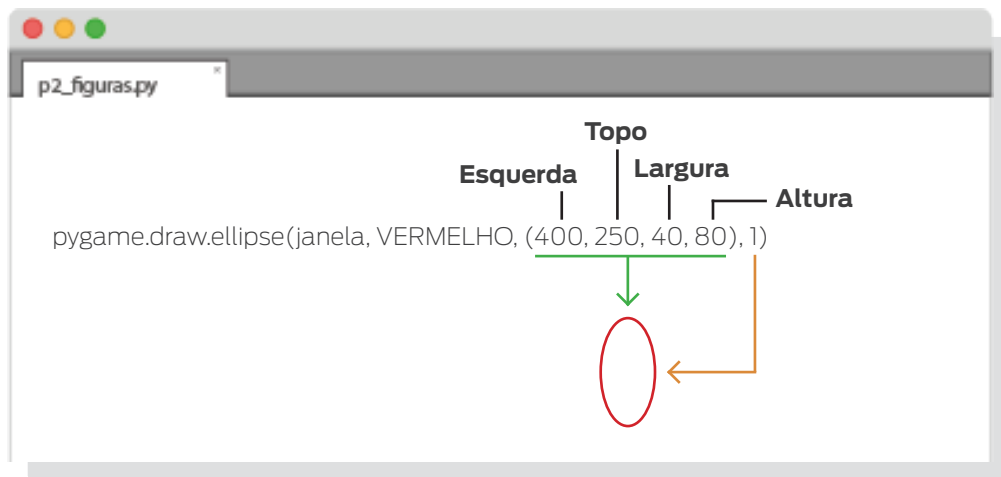
Depois desenhamos um polígono. Um polígono é qualquer figura plana limitada por segmentos de retas. A função `pygame.draw.polygon()`, que é usada para criar polígonos, tem como parâmetros o objeto `pygame.Surface` onde será desenhado, a cor, uma tupla (ou lista) contendo as coordenadas dos pontos que serão os vértices do polígono, no caso um triângulo; e a largura da linha.



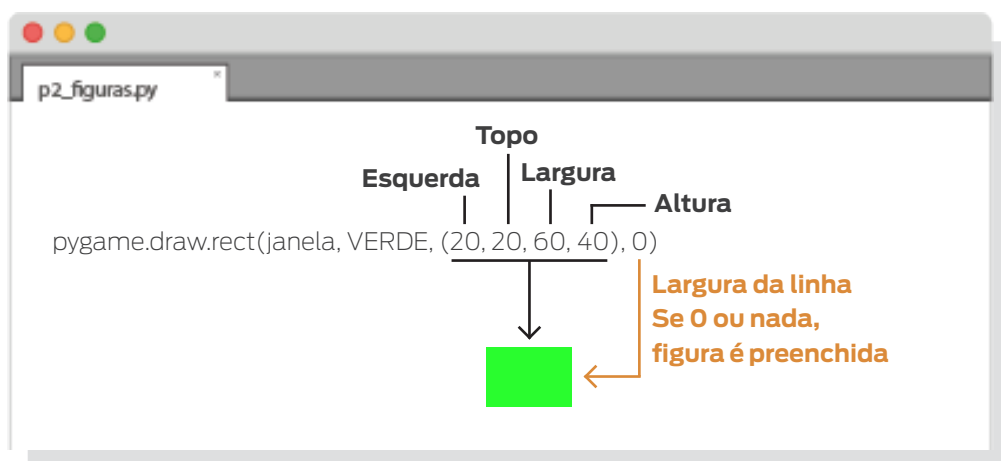
Desenhamos um círculo, a seguir, com a função `pygame.draw.circle()`. O primeiro parâmetro desta função é um objeto `pygame.Surface` onde o círculo será desenhado (`janela`). O segundo parâmetro é a cor. O terceiro são as coordenadas do centro do círculo em uma tupla ou lista. O quarto parâmetro é o raio do círculo em pixels. O último parâmetro indica a largura da linha. Se for passado 0 ou nada, será desenhado um círculo preenchido.



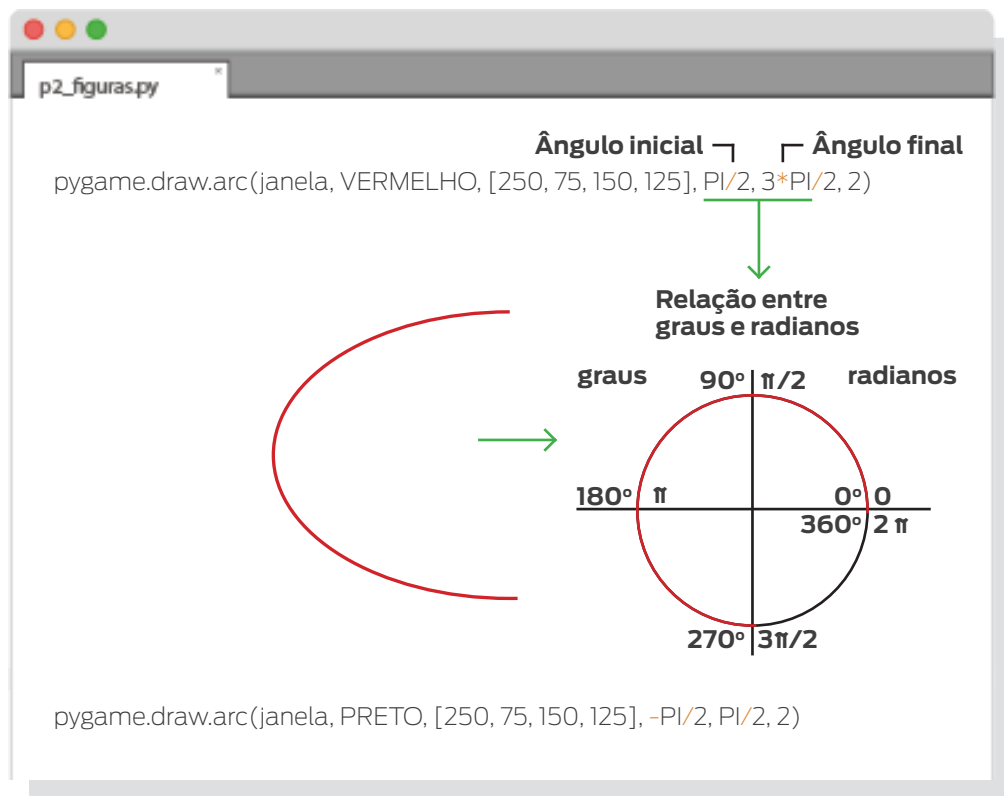
A seguinte figura é uma elipse, criada com a função `pygame.draw.ellipse()`. Esta função é similar à `pygame.draw.circle()`, com uma diferença: em vez de passar as coordenadas do centro do círculo e o raio, uma tupla de quatro inteiros indicando a posição da esquerda (em relação ao eixo x - horizontal), do topo (em relação ao eixo y - vertical), a largura e a altura da elipse é passada.



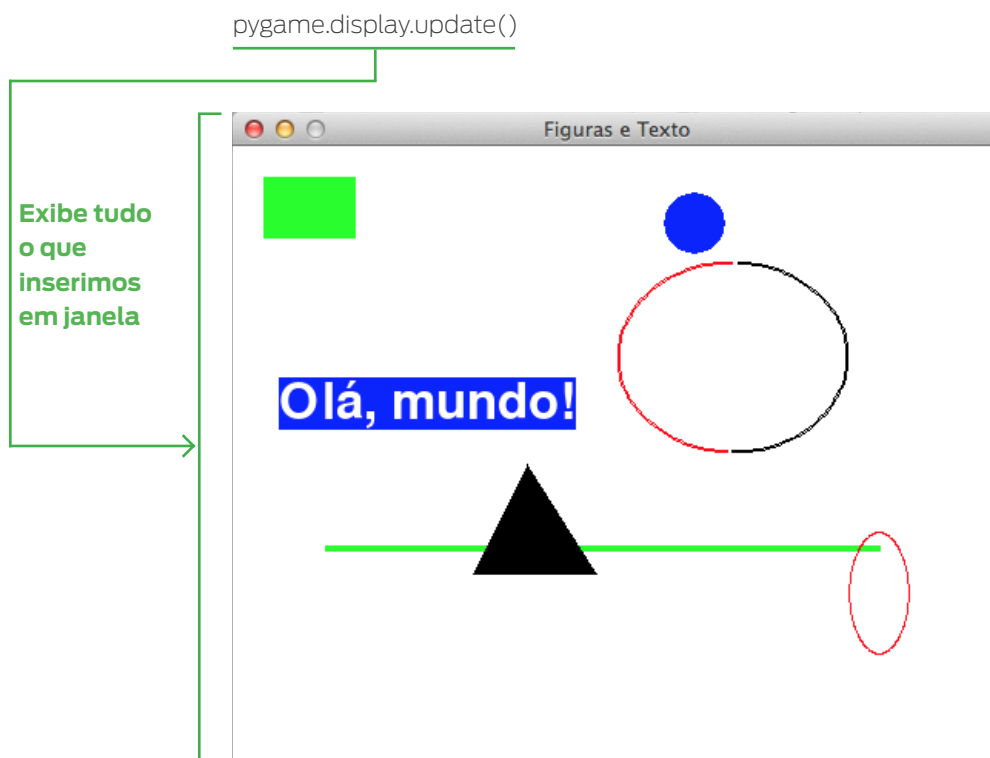
Após, desenhamos um retângulo, usando `pygame.draw.rect()`. O terceiro parâmetro desta função é uma tupla de quatro inteiros correspondentes à posição do lado da esquerda e do topo, largura e altura do retângulo. Os outros parâmetros são semelhantes aos das figuras recentemente analisadas.



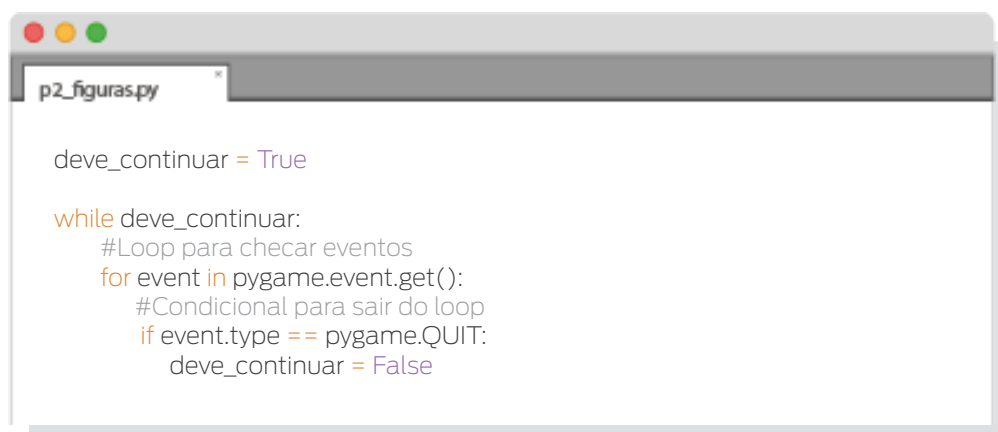
Para finalizar as figuras, desenhemos dois arcos para formarem uma elipse de duas cores. A diferença aqui são o quarto e quinto parâmetros. Estes se referem ao ângulo inicial e final, em radianos, com o 0 à direita.



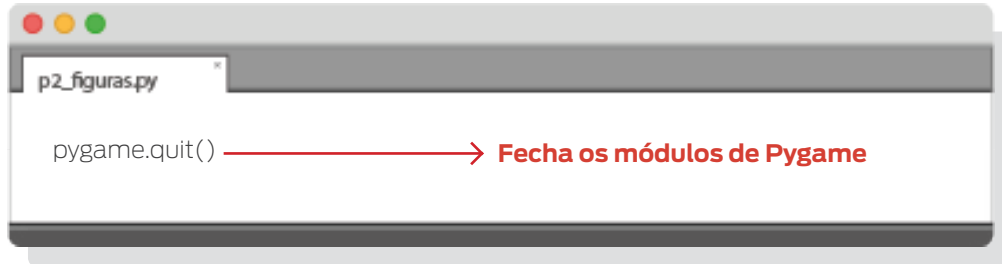
É a função `pygame.display.update()` que atualiza a janela, mostrando tudo o que foi desenhado. Sem o uso desta função, nada aparecerá.

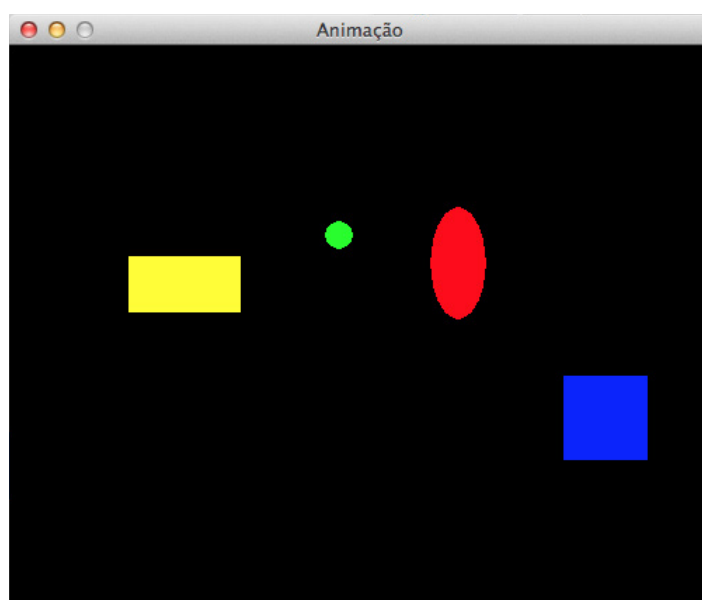
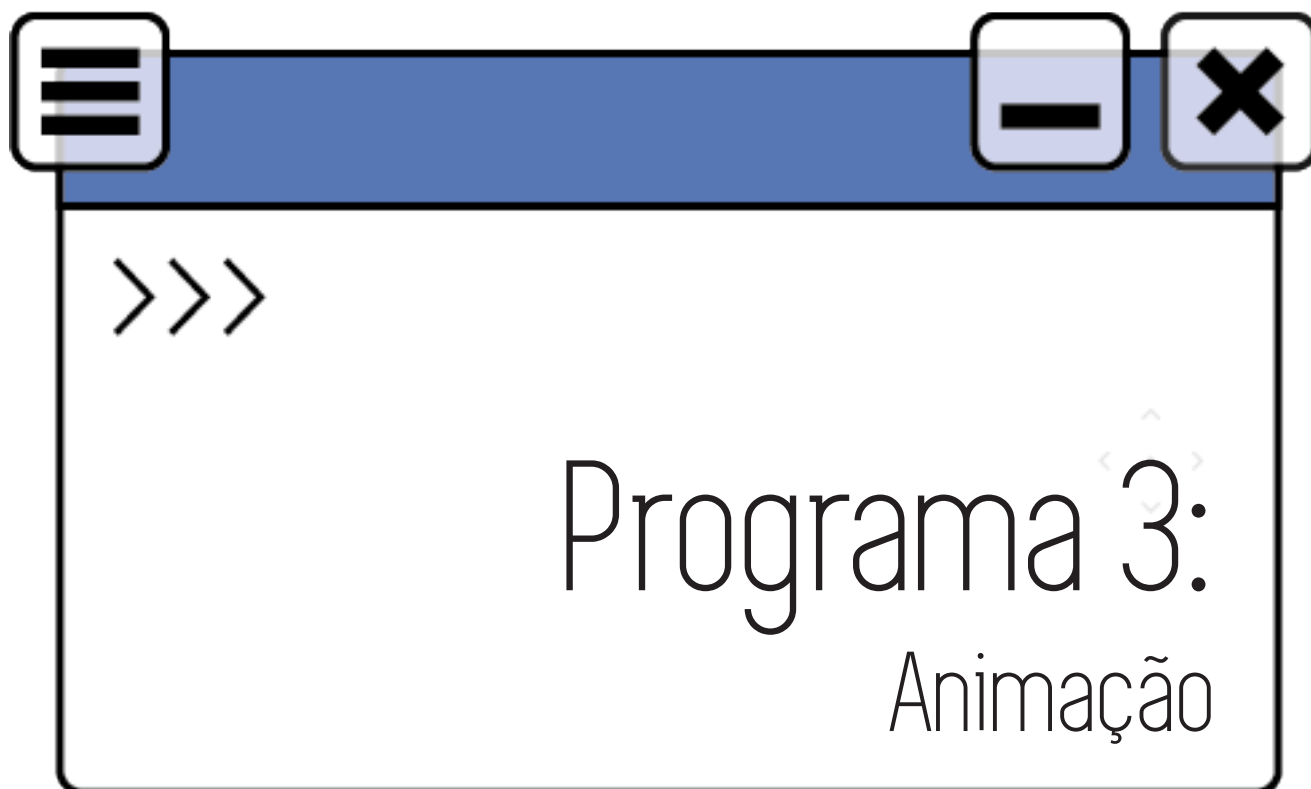


Assim como no programa anterior, este loop apenas mantém a janela aberta até que desejemos fechá-la. Futuramente, toda a ação do jogo estará aqui dentro.



Para finalizar, chamamos a função `pygame.quit()` para fechar todos os módulos de `pygame`. Esta função é chamada automaticamente ao finalizar um programa, não sendo, então, necessária a sua utilização neste contexto. A vantagem da sua utilização é que explicita que os módulos de `pygame` foram fechados.





No último programa desenhemos algumas figuras e colocamos texto na janela. Neste capítulo, veremos como dar movimentação às figuras, ou melhor, dar a impressão que elas estão se movendo. O código se encontra no arquivo `p3_animacao.py`.



```

p3_animacao.py

import pygame, time

# definindo as cores
PRETO = (0, 0, 0)
AMARELO = (255, 255, 0)
VERMELHO = (255, 0, 0)
VERDE = (0, 255, 0)
AZUL = (0, 0, 255)

# definindo outras constantes do jogo
LARGURAJANELA = 500
ALTURAJANELA = 400

# definindo a função mover()
def mover(figura, dim_janela):
    borda_esquerda = 0
    borda_superior = 0
    borda_direita = dim_janela[0]
    borda_inferior = dim_janela[1]
    if figura['objRect'].top < borda_superior or figura['objRect'].bottom > borda_inferior:
        # figura atingiu o topo ou a base da janela
        figura['vel'][1] = -figura['vel'][1]
    if figura['objRect'].left < borda_esquerda or figura['objRect'].right > borda_direita:
        # figura atingiu o lado esquerdo ou direito da janela
        figura['vel'][0] = -figura['vel'][0]
    figura['objRect'].x += figura['vel'][0]
    figura['objRect'].y += figura['vel'][1]

# inicializando módulos de pygame
pygame.init()

# criando a janela
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Animação')

# criando as figuras
f1 = {'objRect': pygame.Rect(300, 80, 40, 80), 'cor': VERMELHO, 'vel': [0, -5], 'forma': 'ELIPSE'}
f2 = {'objRect': pygame.Rect(200, 200, 20, 20), 'cor': VERDE, 'vel': [5, 5], 'forma': 'ELIPSE'}
f3 = {'objRect': pygame.Rect(100, 150, 60, 60), 'cor': AZUL, 'vel': [-5, 5], 'forma': 'RETANGULO'}
f4 = {'objRect': pygame.Rect(200, 150, 80, 40), 'cor': AMARELO, 'vel': [5, 0], 'forma': 'RETANGULO'}

figuras = [f1, f2, f3, f4]

deve_continuar = True

# loop do jogo
while deve_continuar:
    # checando se ocorreu um evento QUIT
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            deve_continuar = False

    # preenchendo o fundo com a cor preta
    janela.fill(PRETO)

    for figura in figuras:
        # reposicionando a figura
        mover(figura, (LARGURAJANELA, ALTURAJANELA))

        # desenhando a figura na janela
        if figura['forma'] == 'RETANGULO':
            pygame.draw.rect(janela, figura['cor'], figura['objRect'])
        elif figura['forma'] == 'ELIPSE':
            pygame.draw.ellipse(janela, figura['cor'], figura['objRect'])

    # atualizando na tela tudo o que foi desenhado
    pygame.display.update()

    # esperando 0.02 segundos
    time.sleep(0.02)

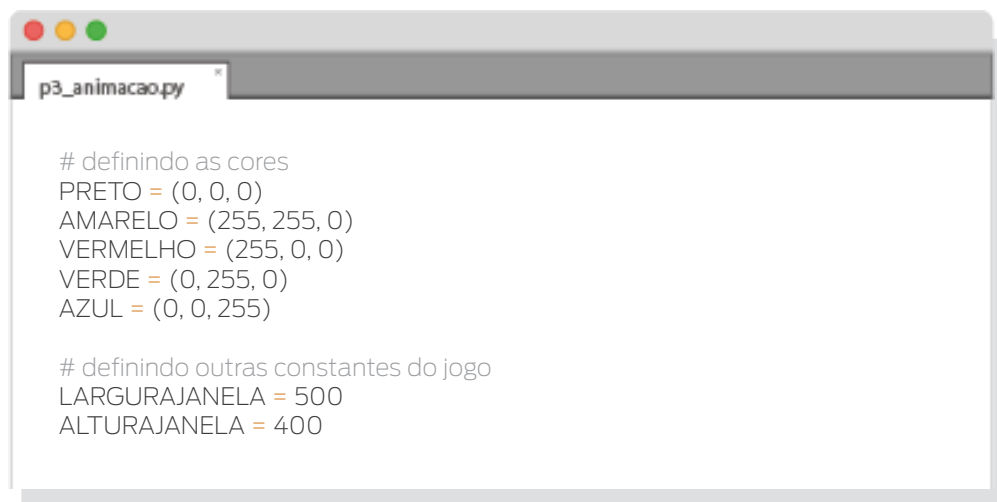
# encerrando módulos de Pygame
pygame.quit()

```

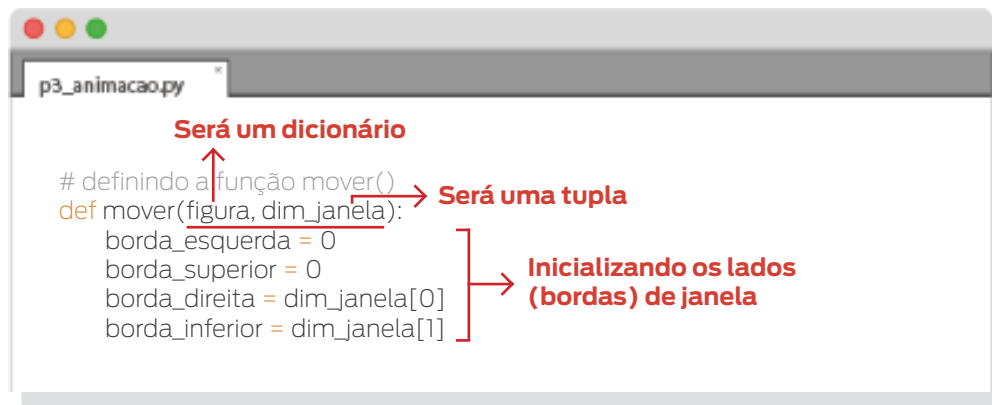
Foram importados os módulos *pygame* e *time*. Já falamos do módulo *time* na primeira parte do tutorial quando vimos o tópico “Módulos”.



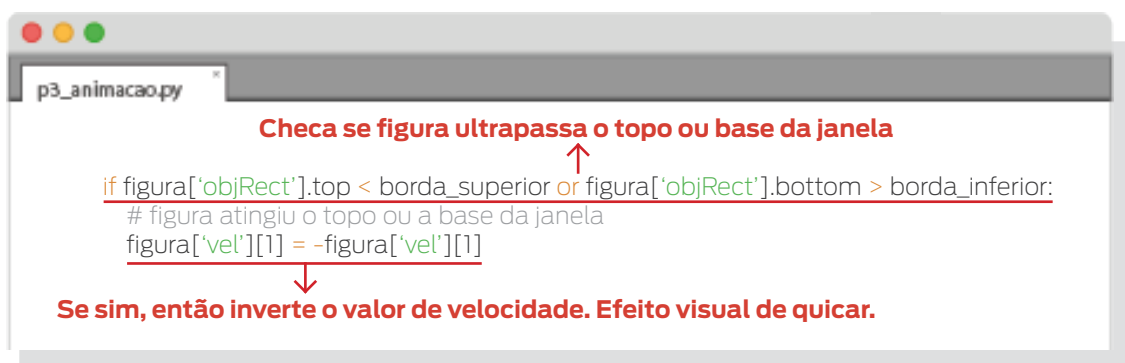
Também foram criadas algumas variáveis constantes para as cores e dimensões da janela.



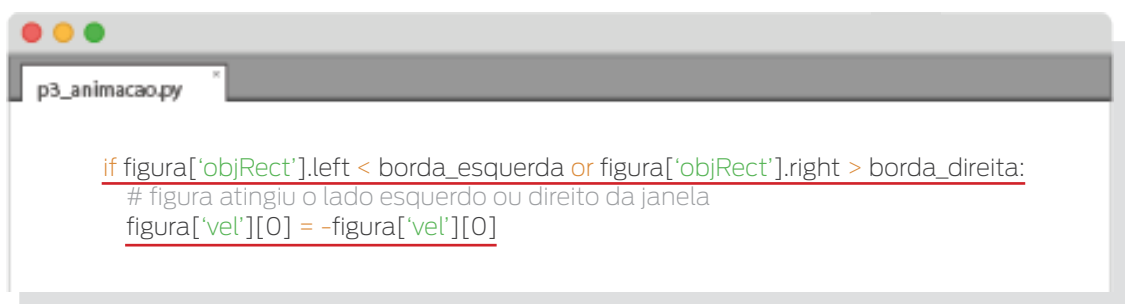
Continuando, foi definida a função *mover()*. Esta função terá como parâmetros uma figura e as dimensões da janela. Cada figura será um dicionário e as dimensões da janela estarão em uma tupla. A função se encarregará de dar as novas coordenadas às figuras que receber como argumento.




Dentro da função existem algumas características relacionadas às estruturas das figuras. Deixaremos os detalhes destas estruturas para quando as figuras forem criadas, mais adiante. Mas o que acontece no interior da função é basicamente o seguinte: checamos se alguma parte da figura atinge alguma das bordas da janela. Se isto acontecer, a figura inverte sua velocidade na direção correspondente.



Ou seja, se ela estava indo da direita para a esquerda, passará a se movimentar da esquerda para a direita e isto ocorrerá para todos os sentidos possíveis (direita-esquerda, esquerda-direita, cima-baixo, baixo-cima).

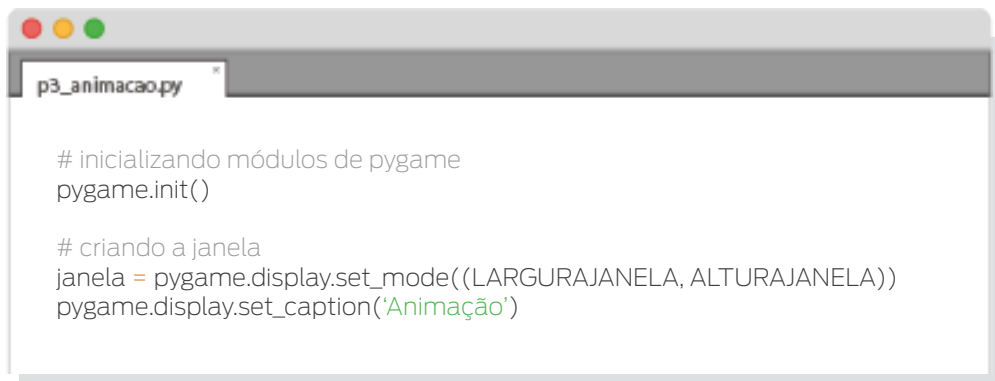


Finalmente, as novas coordenadas da figura correspondente serão, então, guardadas dentro da estrutura.



```
figura['objRect'].x += figura['vel'][0]
figura['objRect'].y += figura['vel'][1]
```

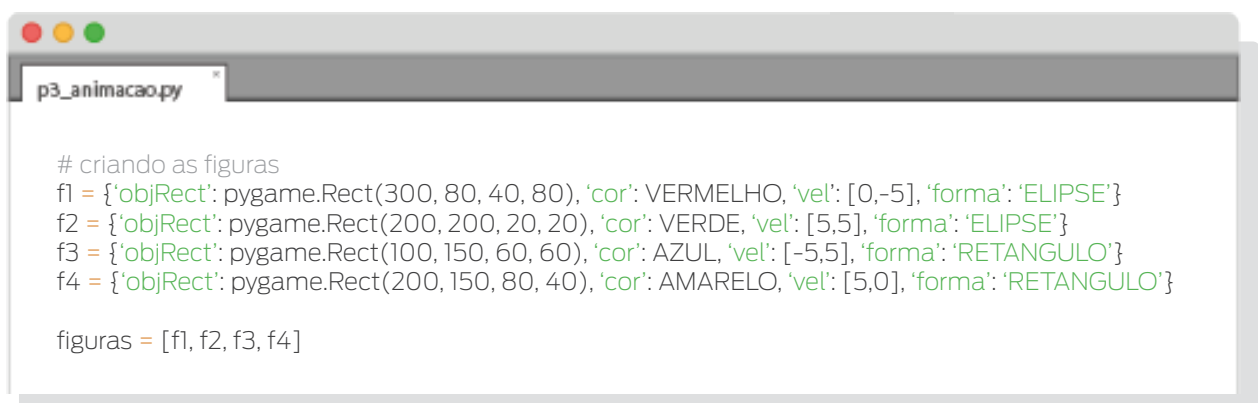
Em seguida, inicializamos os módulos de pygame e criamos a janela.



```
# inicializando módulos de pygame
pygame.init()

# criando a janela
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Animação')
```

Após, cada uma das figuras é criada e colocada na lista *figuras*.

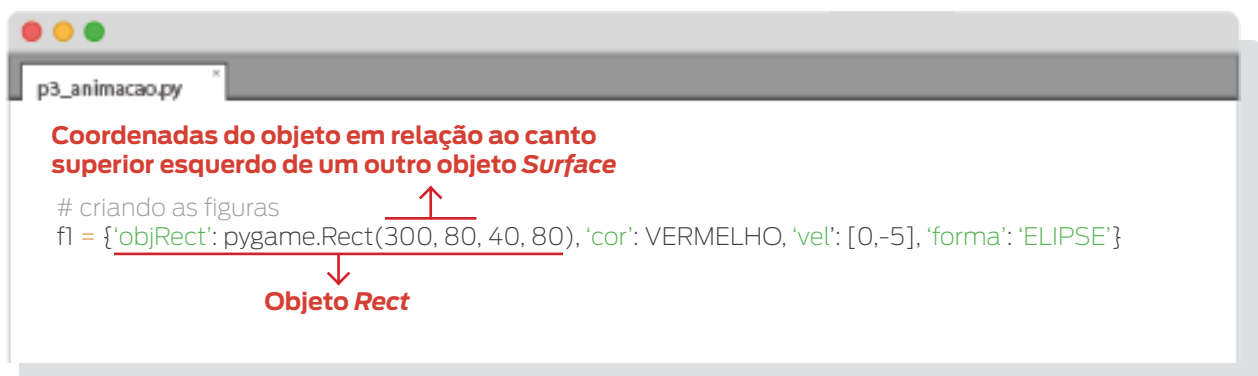


```
# criando as figuras
f1 = {'objRect': pygame.Rect(300, 80, 40, 80), 'cor': VERMELHO, 'vel': [0,-5], 'forma': 'ELIPSE'}
f2 = {'objRect': pygame.Rect(200, 200, 20, 20), 'cor': VERDE, 'vel': [5,5], 'forma': 'ELIPSE'}
f3 = {'objRect': pygame.Rect(100, 150, 60, 60), 'cor': AZUL, 'vel': [-5,5], 'forma': 'RETANGULO'}
f4 = {'objRect': pygame.Rect(200, 150, 80, 40), 'cor': AMARELO, 'vel': [5,0], 'forma': 'RETANGULO'}

figuras = [f1, f2, f3, f4]
```

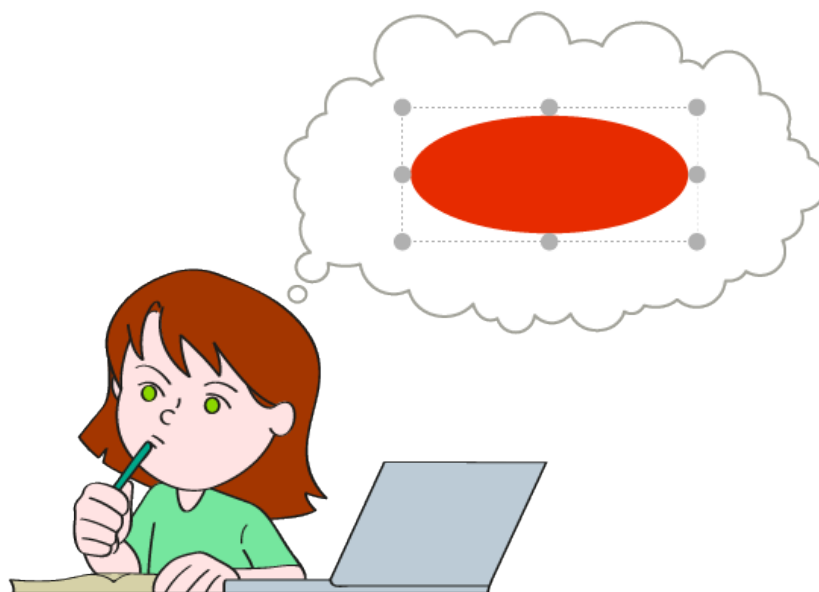
Chegou a hora de falar sobre a estrutura de cada figura. Cada figura é representada por um dicionário que contém 4 chaves (*objRect*, *cor*, *vel* e *forma*).

- A chave *objRect* contém um objeto *pygame.Rect* criado a partir de realizar *pygame.Rect(x,y,l,a)*, onde *x* e *y* correspondem às coordenadas do canto superior esquerdo do retângulo em relação a um outro objeto *Surface*, *l* à largura e *a* à altura da figura. Objetos do tipo *pygame.Rect* são úteis para armazenar e manipular áreas retangulares. Perceba que estamos usando este tipo de objeto até mesmo para trabalhar com elipses.

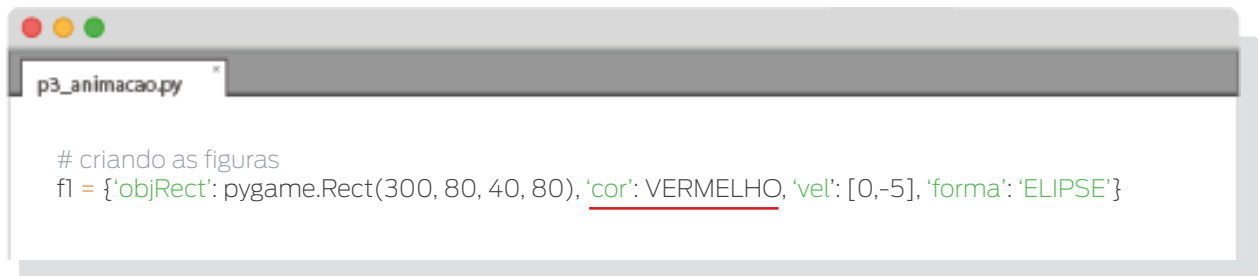


Pense nele como um retângulo invisível que envolve a figura que vamos desenhar.

Este tipo de objeto tem muitos atributos, entre eles *left* (ou *x*), *right*, *top* (ou *y*) e *bottom*, que guardam as posições de cada lado do retângulo.

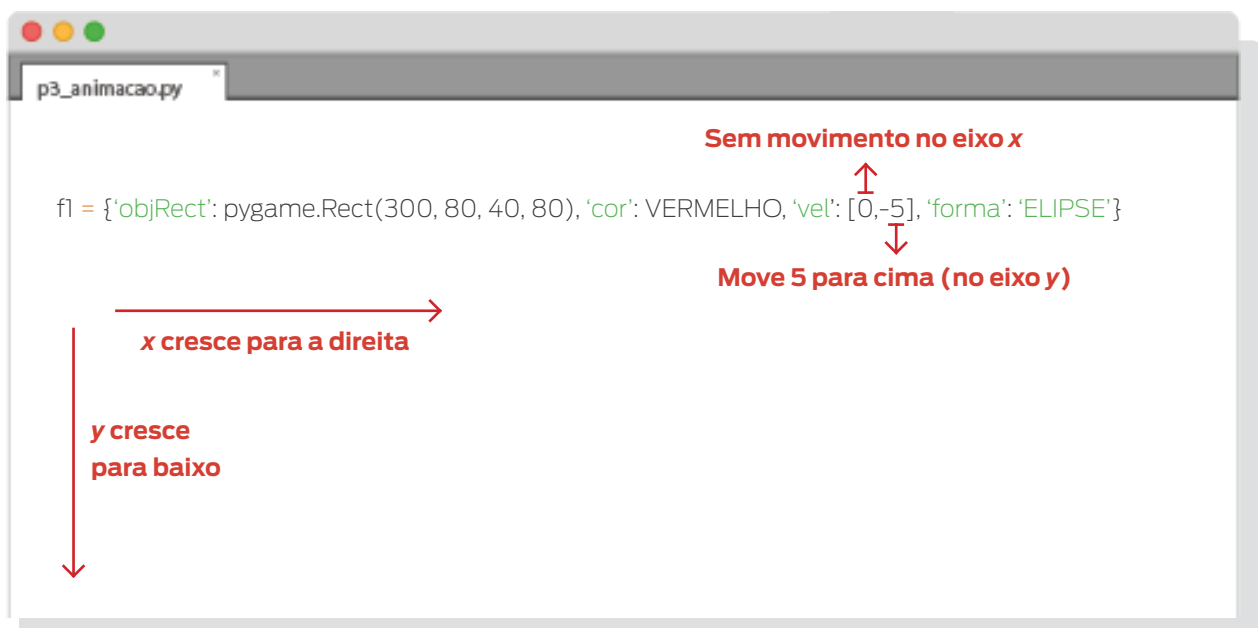


- A chave *cor* guarda uma tupla com a cor da figura.



```
# criando as figuras
f1 = {'objRect': pygame.Rect(300, 80, 40, 80), 'cor': VERMELHO, 'vel': [0, -5], 'forma': 'ELIPSE'}
```

- A chave *vel* guarda uma lista com dois elementos referentes ao valor em que as posições *x* e *y* irão variar. Lembrando que a coordenada *x* cresce para a direita e a coordenada *y* cresce para baixo, se tivermos  $[3, 3]$  significa que a figura se moverá 3 pixels para direita e 3 para baixo; se tivermos  $[3, -3]$  significa que a figura se moverá 3 pixels para a direita e 3 para cima; já se tivermos  $[-3, 0]$  a figura se moverá 3 pixels para a esquerda e não se moverá na vertical.



```
f1 = {'objRect': pygame.Rect(300, 80, 40, 80), 'cor': VERMELHO, 'vel': [0, -5], 'forma': 'ELIPSE'}
```

**Sem movimento no eixo x**

↑  
↓  
**Move 5 para cima (no eixo y)**

→  
**x cresce para a direita**

↓  
**y cresce para baixo**

- A chave *forma* guarda uma string com a forma da figura. No caso usamos apenas 'ELIPSE' e 'RETANGULO', já que círculos e quadrados são casos especiais destes tipos.

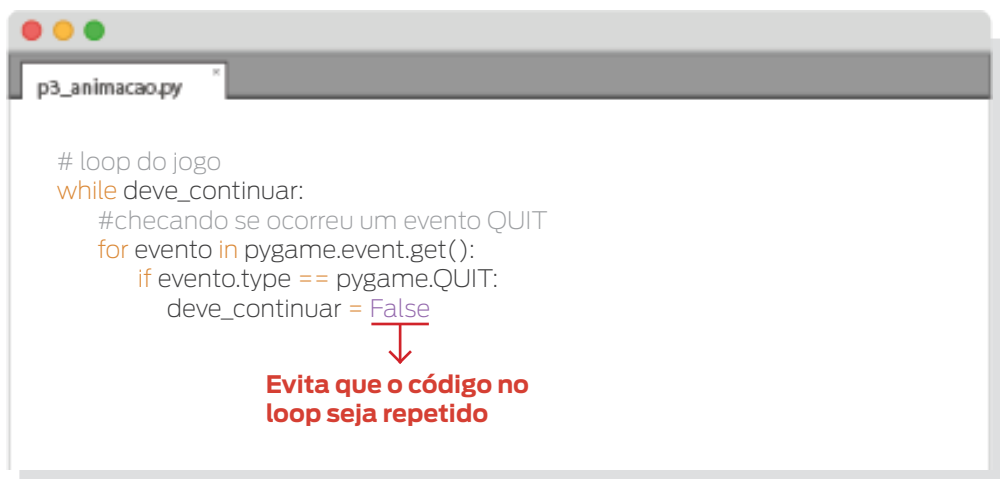
Podemos ver que a parte que corresponde ao loop está maior, se comparado com a dos programas anteriores. Antes de entrar no loop, inicializamos a variável *deve\_continuar* com o valor *True*, uma vez que queremos que o loop seja executado.



```
deve_continuar = True
```

Dentro do loop, ocorre:

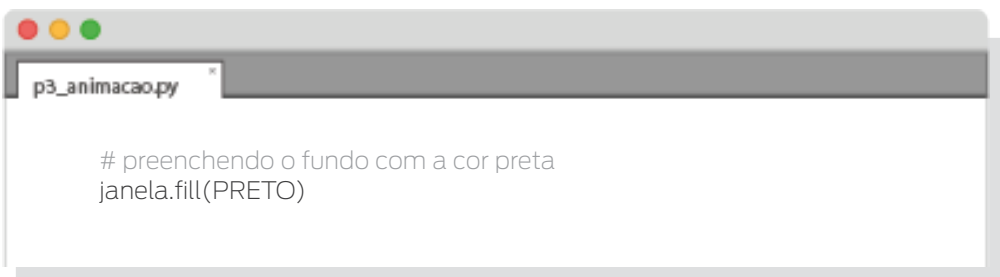
- Checamos os eventos ocorridos desde a última iteração (repetição). Se ocorreu um evento *QUIT*, mudamos o valor da variável *deve\_continuar* para *False*, fazendo com que a parte do código dentro do loop não seja mais repetida.



```
# loop do jogo
while deve_continuar:
    #checando se ocorreu um evento QUIT
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            deve_continuar = False
```

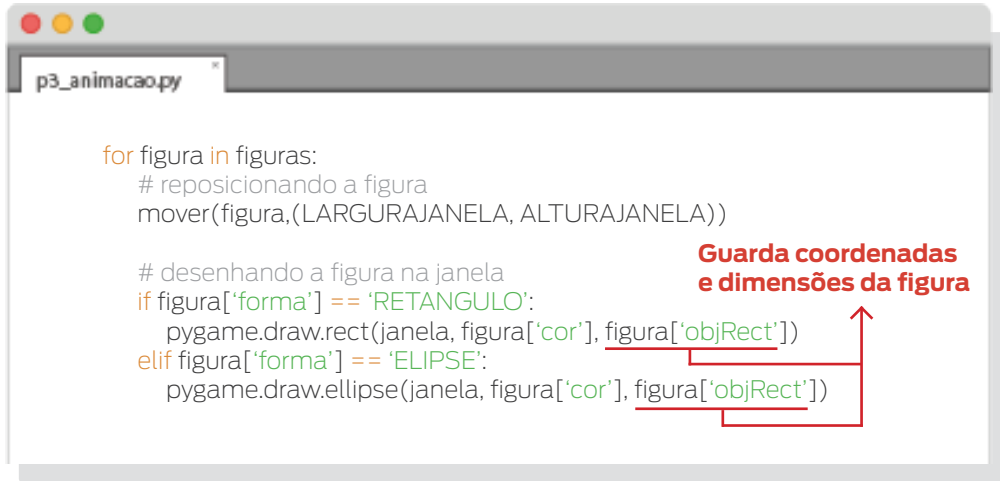
**Evita que o código no loop seja repetido**

- Preenchemos o fundo das janela com a cor preta.



```
# preenchendo o fundo com a cor preta
janela.fill(PRETO)
```

- Criamos um outro loop (loop *for*, desta vez) onde cada figura será reposicionada e desenhada. Perceba que, nas funções `pygame.draw.rect()` e `pygame.draw.ellipse()`, passamos o objeto `pygame.Rect` (ou simplesmente `Rect`, para simplificar) como terceiro argumento, enquanto que no programa anterior usávamos uma tupla contendo as coordenadas e dimensões da figura. Fazemos isto porque o objeto `Rect` de cada figura é que está guardando estes dados atualizados.

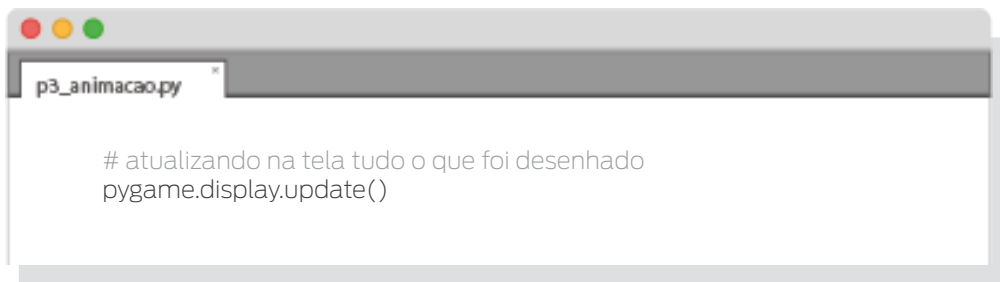


```
for figura in figuras:
    # reposicionando a figura
    mover(figura, (LARGURAJANELA, ALTURAJANELA))

    # desenhando a figura na janela
    if figura['forma'] == 'RETANGULO':
        pygame.draw.rect(janela, figura['cor'], figura['objRect'])
    elif figura['forma'] == 'ELIPSE':
        pygame.draw.ellipse(janela, figura['cor'], figura['objRect'])
```

**Guarda coordenadas e dimensões da figura**

- Atualizamos tudo o que foi desenhado.



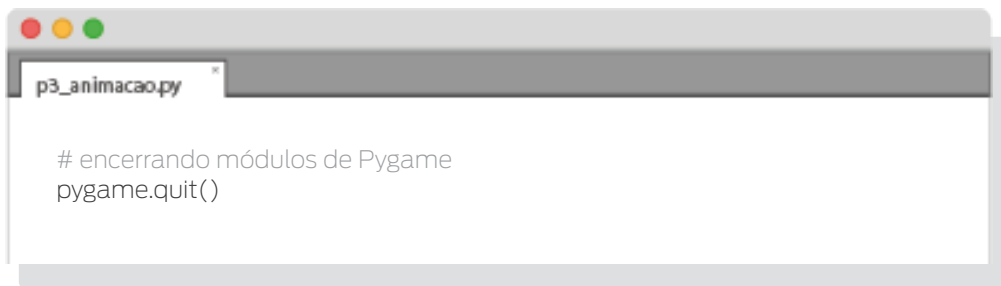
```
# atualizando na tela tudo o que foi desenhado
pygame.display.update()
```

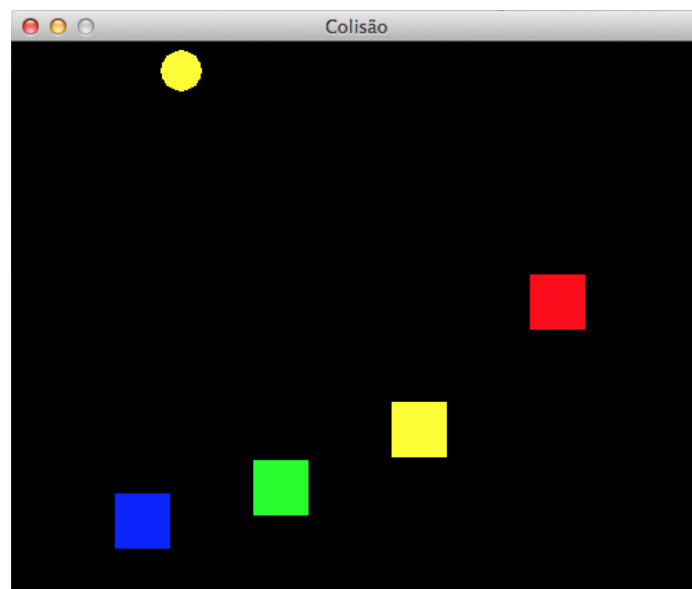


- Retardamos a execução por 0.02 segundos. Para isto usamos a função `sleep()` do módulo `time`. Ela recebe um número que será a quantidade de segundos na qual a execução ficará pausada.



Para finalizar, fechamos os módulos de pygame uma vez que saímos do loop.





No programa anterior vimos como fazer com que figuras se movimentem. Agora, queremos que exista algum tipo de interação entre as figuras. Ou seja, que aconteça alguma coisa quando entrem em contato. Por exemplo, um muro pode impedir que a personagem se movimente em certas direções; ou a personagem pode adquirir uma moeda quando toca nela. O código deste capítulo se encontra em `p4_colisao.py`.

```

p4_colisao.py

import pygame

# definindo as cores
PRETO = (0, 0, 0)
AMARELO = (255, 255, 0)
VERMELHO = (255, 0, 0)
VERDE = (0, 255, 0)
AZUL = (0, 0, 255)
BRANCO = (255, 255, 255)

# definindo outras constantes do jogo
LARGURAJANELA = 500
ALTURAJANELA = 400

# definindo a função mover(), que registra a posição de uma figura
def mover(figura, dim_janela):
    borda_esquerda = 0
    borda_superior = 0
    borda_direita = dim_janela[0]
    borda_inferior = dim_janela[1]
    if figura['objRect'].top < borda_superior or figura['objRect'].bottom > borda_inferior:
        # figura atingiu o topo ou a base da janela
        figura['vel'][1] = -figura['vel'][1]
    if figura['objRect'].left < borda_esquerda or figura['objRect'].right > borda_direita:
        # figura atingiu o lado esquerdo ou direito da janela
        figura['vel'][0] = -figura['vel'][0]
    figura['objRect'].x += figura['vel'][0]
    figura['objRect'].y += figura['vel'][1]

# inicializando pygame
pygame.init()

# criando um objeto pygame.time.Clock
relogio = pygame.time.Clock()

# criando a janela
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Colisão')

# criando os blocos e colocando-os em uma lista
b1 = {'objRect': pygame.Rect(375, 80, 40, 40), 'cor': VERMELHO, 'vel': [0, 2]}
b2 = {'objRect': pygame.Rect(175, 200, 40, 40), 'cor': VERDE, 'vel': [0, -3]}
b3 = {'objRect': pygame.Rect(275, 150, 40, 40), 'cor': AMARELO, 'vel': [0, -1]}
b4 = {'objRect': pygame.Rect(75, 150, 40, 40), 'cor': AZUL, 'vel': [0, 4]}
blocos = [b1, b2, b3, b4]

# criando a bola
bola = {'objRect': pygame.Rect(270, 330, 30, 30), 'cor': BRANCO, 'vel': [3, 3]}

deve_continuar = True

# loop do jogo
while deve_continuar:
    # checando se ocorreu um evento QUIT
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            deve_continuar = False

    # preenchendo o fundo com a cor preta
    janela.fill(PRETO)

    for bloco in blocos:
        # reposicionando o bloco
        mover(bloco, (LARGURAJANELA, ALTURAJANELA))

        # desenhando o bloco na janela
        pygame.draw.rect(janela, bloco['cor'], bloco['objRect'])

        # mudando a cor da bola caso colida com algum bloco
        mudarCor = bola['objRect'].colliderect(bloco['objRect'])
        if mudarCor:
            bola['cor'] = bloco['cor']

    # reposicionando e desenha a bola
    mover(bola, (LARGURAJANELA, ALTURAJANELA))
    pygame.draw.ellipse(janela, bola['cor'], bola['objRect'])

    # mostrando na tela tudo o que foi desenhado
    pygame.display.update()

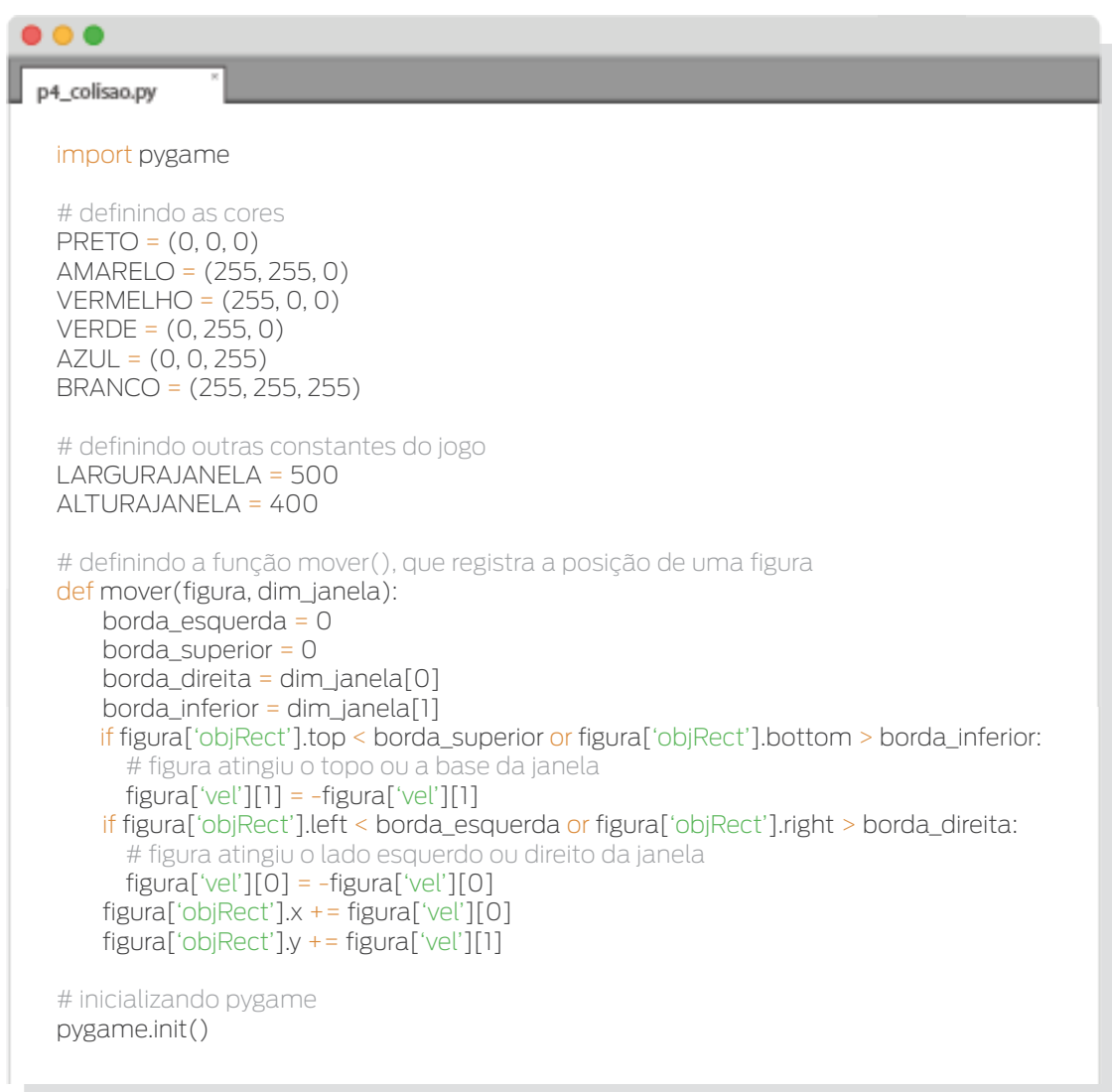
    # limitando a 40 quadros por segundo
    relogio.tick(40)

# encerrando módulos de Pygame
pygame.quit()

```

**Nota:** Já usamos a palavra “iteração” no programa anterior quando falamos do loop do jogo e usamos “interação” no parágrafo anterior. A falta do “n” não foi uma distração durante a escrita, elas são palavras distintas. “Iteração” significa “repetição” e “interação” significa “contato”.

Aqui podemos ver que foi importado o módulo *pygame*, definidas algumas cores, as dimensões da janela, a função *mover()* e inicializados os módulos de Pygame, como já visto antes.



```

import pygame

# definindo as cores
PRETO = (0, 0, 0)
AMARELO = (255, 255, 0)
VERMELHO = (255, 0, 0)
VERDE = (0, 255, 0)
AZUL = (0, 0, 255)
BRANCO = (255, 255, 255)

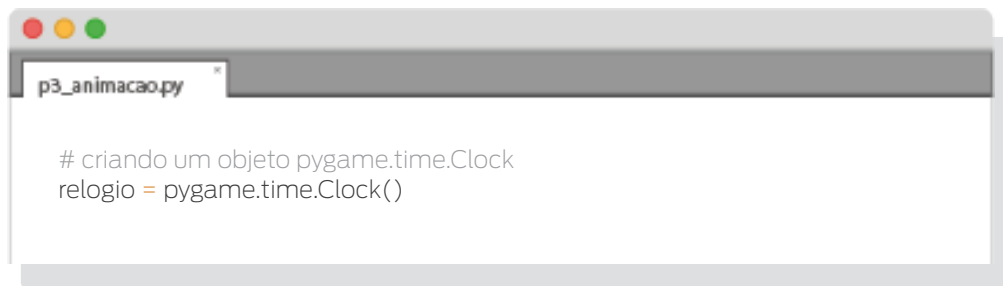
# definindo outras constantes do jogo
LARGURAJANELA = 500
ALTURAJANELA = 400

# definindo a função mover(), que registra a posição de uma figura
def mover(figura, dim_janela):
    borda_esquerda = 0
    borda_superior = 0
    borda_direita = dim_janela[0]
    borda_inferior = dim_janela[1]
    if figura['objRect'].top < borda_superior or figura['objRect'].bottom > borda_inferior:
        # figura atingiu o topo ou a base da janela
        figura['vel'][1] = -figura['vel'][1]
    if figura['objRect'].left < borda_esquerda or figura['objRect'].right > borda_direita:
        # figura atingiu o lado esquerdo ou direito da janela
        figura['vel'][0] = -figura['vel'][0]
    figura['objRect'].x += figura['vel'][0]
    figura['objRect'].y += figura['vel'][1]

# inicializando pygame
pygame.init()

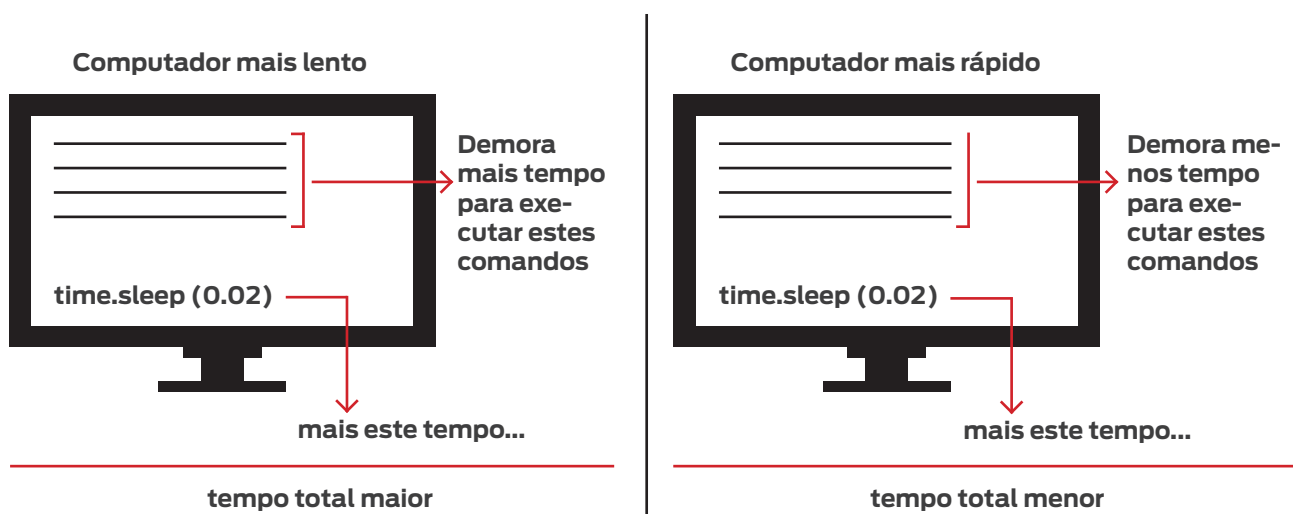
```

Na próxima linha temos algo não visto anteriormente: o objeto *Clock*. Este objeto pertence ao módulo *time* de *pygame*. Ele é criado com a chamada *pygame.time.Clock()* e guardado na variável *relogio*.



No programa anterior (p3\_animacao.py) usamos a função `time.sleep()` dentro do loop do jogo para retardar o programa, com a finalidade de ver a movimentação dos blocos com uma velocidade que consideramos aceitável. Se não usarmos essa função, o movimento será mais rápido. Por outro lado, se usarmos um valor maior no argumento da função, a movimentação ficará mais lenta.

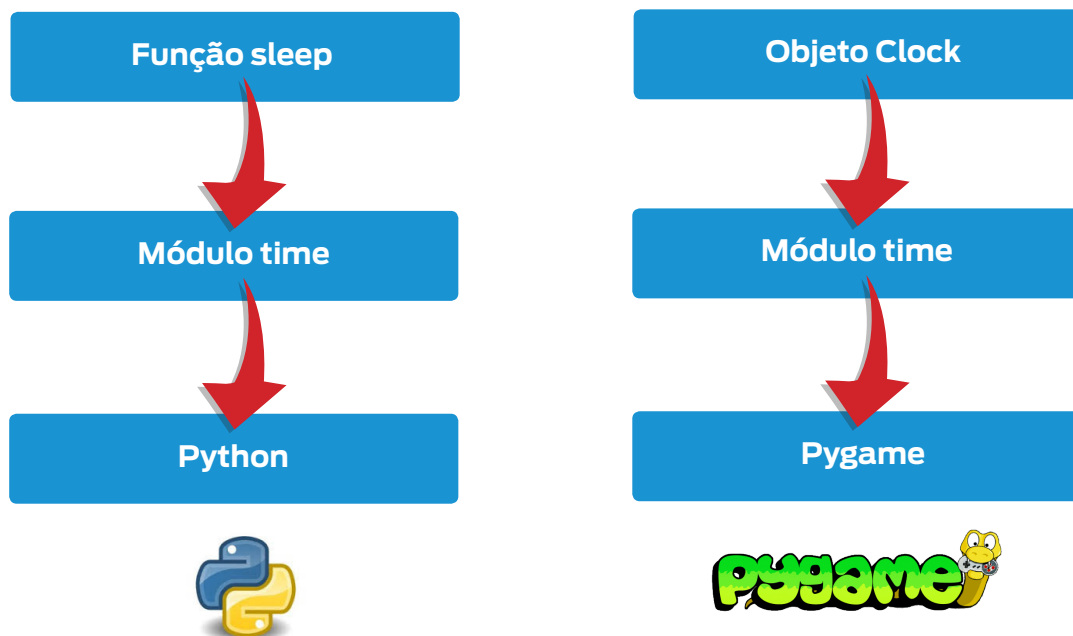
A função `time.sleep()` ajuda no controle da velocidade com que um programa é executado, mas não da forma mais apropriada. O problema está no fato que a velocidade de execução dependerá também da velocidade de processamento do computador. Um computador com um processador de maior frequência executará as instruções mais rapidamente do que um com processador de menor frequência.




O que queremos é que o comportamento independa da capacidade do computador e para isto queremos limitar o número de iterações máximas que existem por segundo no loop do jogo, ou seja, queremos que o que se encontra dentro desse loop se repita, como máximo, um certo número de vezes por segundo. Um objeto `pygame.time.Clock` nos possibilita isto. Mais adiante falaremos do método `tick()` destes objetos. Com isto podemos retardar um computador rápido e um lento para que executem o programa com a mesma velocidade (dentro de suas possibilidades).



Algo a ser notado, também, é que no programa anterior usamos uma função (*sleep()*) do módulo *time* de Python e aqui estamos usando o objeto *Clock* que está definido no módulo *time* de Pygame! Estes módulos, apesar de possuírem o mesmo nome, não são os mesmos!



Em seguida criamos a janela, alguns blocos e a bola (que se chocará com os blocos). Estes elementos foram criados de forma muito parecida com o realizado no programa anterior. Também inicializamos o valor da variável *deve\_continuar* antes de entrar no loop do jogo.



```
# criando a janela
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Colisão')

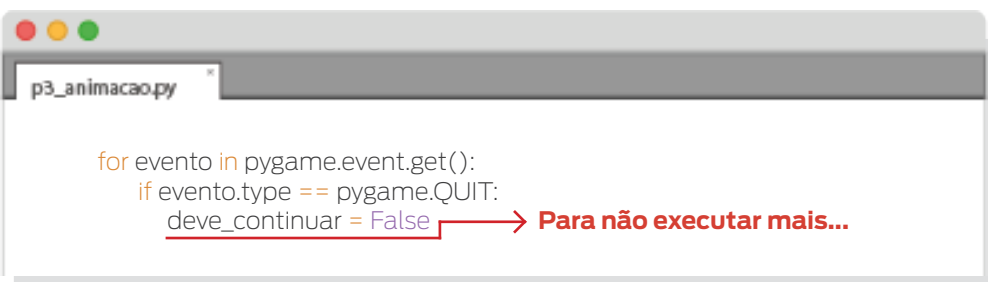
# criando os blocos e colocando-os em uma lista
b1 = {'objRect': pygame.Rect(375, 80, 40, 40), 'cor': VERMELHO, 'vel': [0,2]}
b2 = {'objRect': pygame.Rect(175, 200, 40, 40), 'cor': VERDE, 'vel': [0,-3]}
b3 = {'objRect': pygame.Rect(275, 150, 40, 40), 'cor': AMARELO, 'vel': [0,-1]}
b4 = {'objRect': pygame.Rect(75, 150, 40, 40), 'cor': AZUL, 'vel': [0,4]}
blocos = [b1, b2, b3, b4]

# criando a bola
bola = {'objRect': pygame.Rect(270, 330, 30, 30), 'cor': BRANCO, 'vel': [3,3]}

deve_continuar = True
```

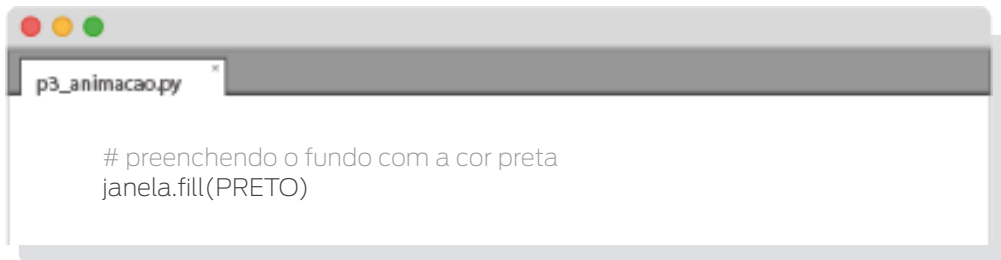
Chegamos ao loop do jogo! Lá, acontece:

- Checamos, dentro dos eventos ocorridos, se algum foi do tipo *QUIT*. Se foi, mudamos a variável *deve\_continuar* para *False* com a finalidade de que o código dentro do loop do jogo não volte a ser executado.



```
for evento in pygame.event.get():
    if evento.type == pygame.QUIT:
        deve_continuar = False → Para não executar mais...
```

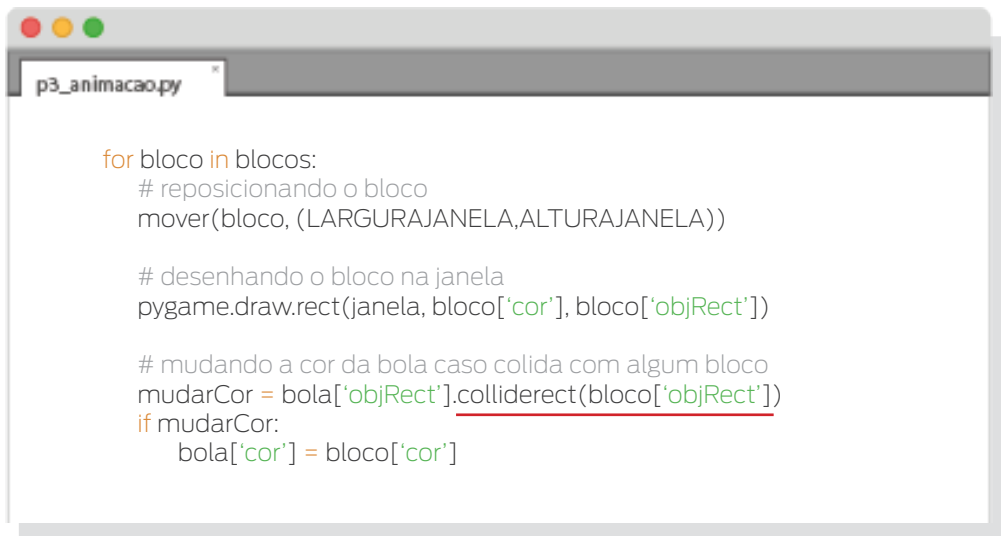
- Preenchemos o fundo da janela com a cor preta.



```
# preenchendo o fundo com a cor preta
janela.fill(PRETO)
```

- Para cada bloco na lista de blocos, o movemos (reposicionamos), o desenhamos, e checamos se ele colidiu com a bola para que ela mude de cor. Para verificar a colisão, usamos o método `colliderect()`, que é um método dos objetos `pygame.Rect` que devolve `True` se os objetos, o que o chamou e o que foi passado como argumento, se sobrepõem completa ou parcialmente. O valor obtido pelo método é guardado na variável `mudarCor`.

Se o bloco colidiu com a bola (`mudarCor = True`), então a bola muda de cor para a cor do bloco.

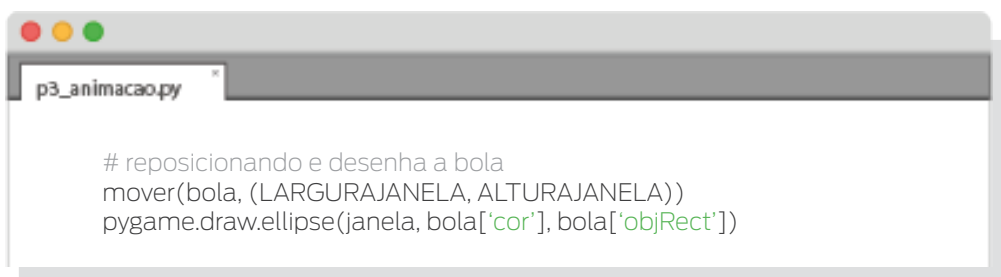


```
for bloco in blocos:
    # reposicionando o bloco
    mover(bloco, (LARGURAJANELA, ALTURAJANELA))

    # desenhando o bloco na janela
    pygame.draw.rect(janela, bloco['cor'], bloco['objRect'])

    # mudando a cor da bola caso colida com algum bloco
    mudarCor = bola['objRect'].colliderect(bloco['objRect'])
    if mudarCor:
        bola['cor'] = bloco['cor']
```

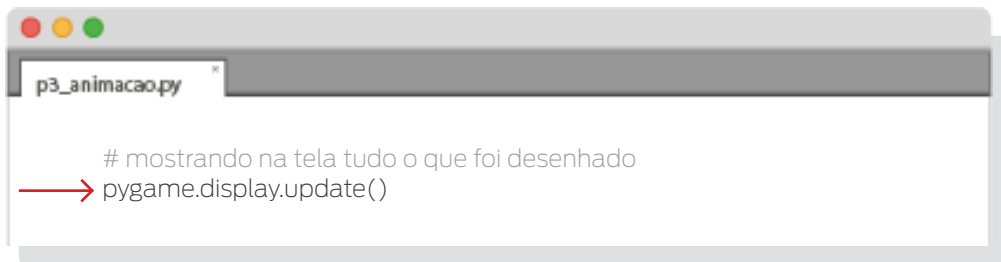
- Reposicionamos e desenhamos a bola.



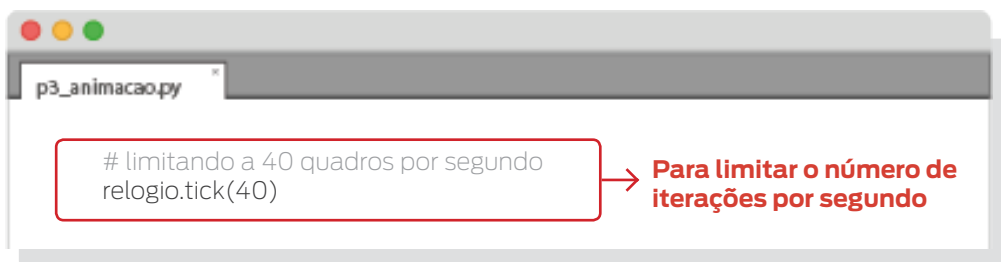
```
# reposicionando e desenha a bola
mover(bola, (LARGURAJANELA, ALTURAJANELA))
pygame.draw.ellipse(janela, bola['cor'], bola['objRect'])
```



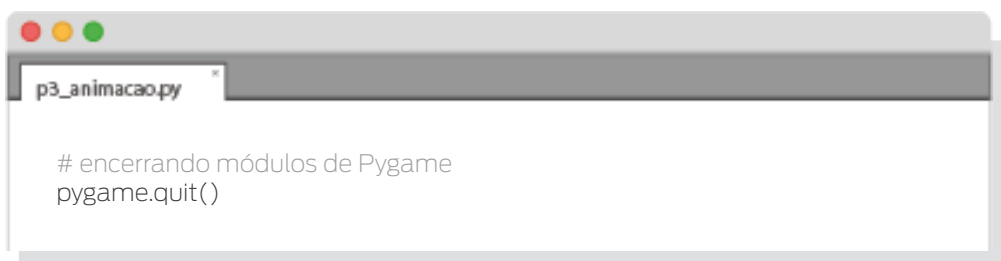
- Atualizamos tudo o que foi desenhado.

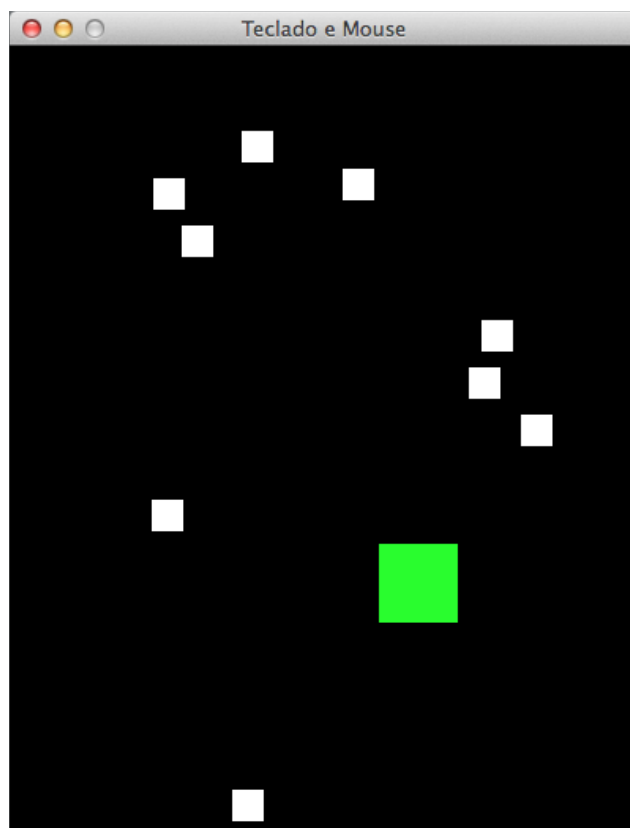


- Controlamos a velocidade de execução. O método `tick()` de um objeto *Clock* pausa a execução para que não haja mais iterações por segundo do que o número que foi passado por argumento, assegurando que o jogo não seja executado mais rápido do que esperamos. Este método deve ser chamado apenas uma vez dentro do loop.

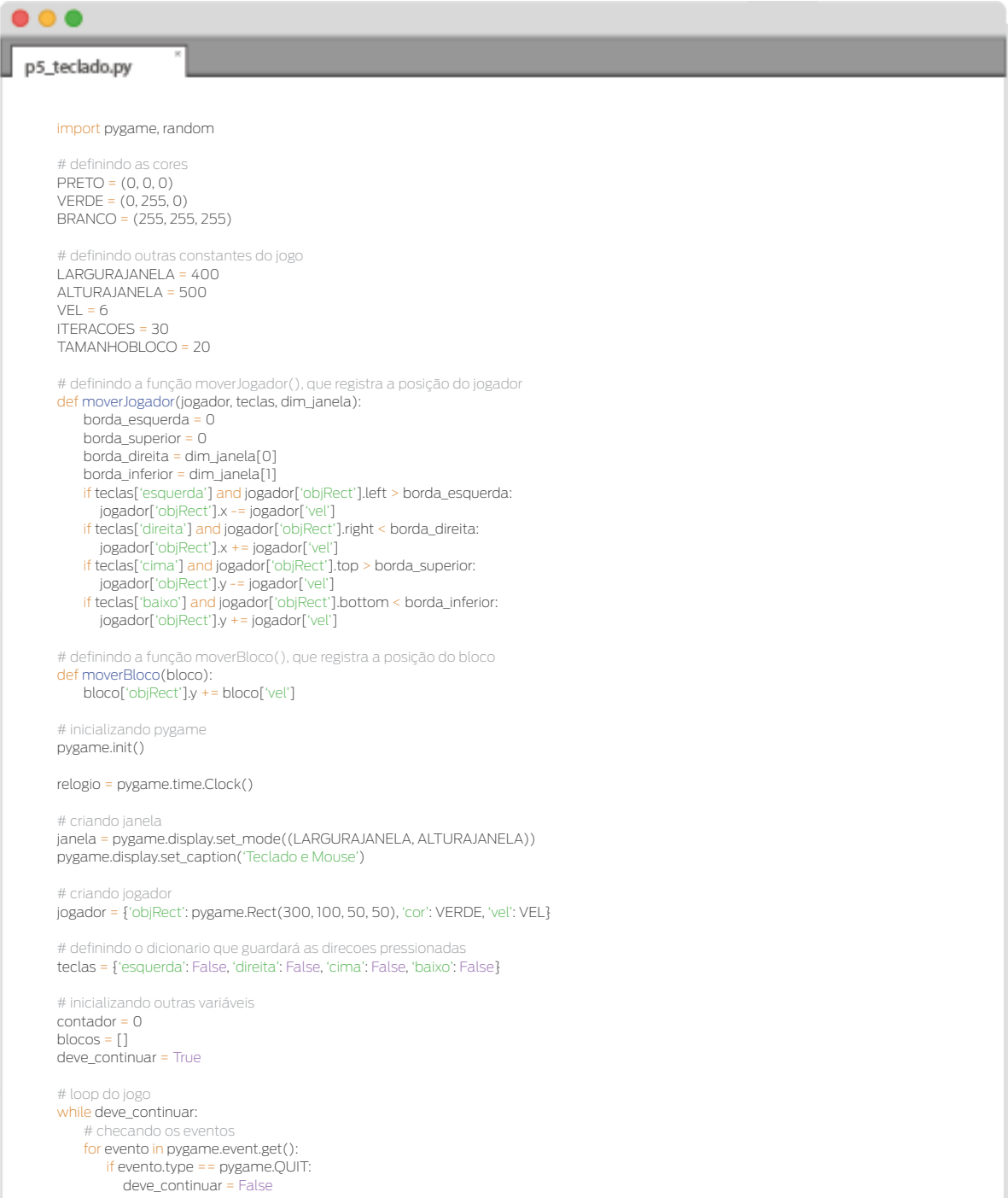


Finalmente, encerramos os módulos de Pygame.





No programa anterior vimos como figuras podem interagir, ou seja, que algo acontece quando elas entram em contato. No caso, o círculo mudava de cor quando colidia com algum quadrado. Agora, veremos que a colisão fará com que uma das figuras desapareça, como se ela fosse comida pela outra figura. Também veremos como controlar uma das figuras usando o teclado, e como criar outras usando o botão do mouse. Afinal, em todo jogo tem que haver algum tipo de interação do jogador.



```

import pygame, random

# definindo as cores
PRETO = (0, 0, 0)
VERDE = (0, 255, 0)
BRANCO = (255, 255, 255)

# definindo outras constantes do jogo
LARGURAJANELA = 400
ALTURAJANELA = 500
VEL = 6
ITERACOES = 30
TAMANHOBLOCO = 20

# definindo a função moverJogador(), que registra a posição do jogador
def moverJogador(jogador, teclas, dim_janela):
    borda_esquerda = 0
    borda_superior = 0
    borda_direita = dim_janela[0]
    borda_inferior = dim_janela[1]
    if teclas['esquerda'] and jogador['objRect'].left > borda_esquerda:
        jogador['objRect'].x -= jogador['vel']
    if teclas['direita'] and jogador['objRect'].right < borda_direita:
        jogador['objRect'].x += jogador['vel']
    if teclas['cima'] and jogador['objRect'].top > borda_superior:
        jogador['objRect'].y -= jogador['vel']
    if teclas['baixo'] and jogador['objRect'].bottom < borda_inferior:
        jogador['objRect'].y += jogador['vel']

# definindo a função moverBloco(), que registra a posição do bloco
def moverBloco(bloco):
    bloco['objRect'].y += bloco['vel']

# inicializando pygame
pygame.init()

relógio = pygame.time.Clock()

# criando janela
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Teclado e Mouse')

# criando jogador
jogador = {'objRect': pygame.Rect(300, 100, 50, 50), 'cor': VERDE, 'vel': VEL}

# definindo o dicionário que guardará as direções pressionadas
teclas = {'esquerda': False, 'direita': False, 'cima': False, 'baixo': False}

# inicializando outras variáveis
contador = 0
blocos = []
deve_continuar = True

# loop do jogo
while deve_continuar:
    # checando os eventos
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            deve_continuar = False

```

```

# quando uma tecla é pressionada
if evento.type == pygame.KEYDOWN:
    if evento.key == pygame.K_ESCAPE:
        deve_continuar = False
    if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
        teclas['esquerda'] = True
    if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
        teclas['direita'] = True
    if evento.key == pygame.K_UP or evento.key == pygame.K_w:
        teclas['cima'] = True
    if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
        teclas['baixo'] = True

# quando uma tecla é solta
if evento.type == pygame.KEYUP:
    if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
        teclas['esquerda'] = False
    if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
        teclas['direita'] = False
    if evento.key == pygame.K_UP or evento.key == pygame.K_w:
        teclas['cima'] = False
    if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
        teclas['baixo'] = False

# quando um botao do mouse é pressionado
if evento.type == pygame.MOUSEBUTTONDOWN:
    blocos.append({'objRect': pygame.Rect(evento.pos[0], evento.pos[1], TAMANHOBLOCO, TAMANHOBLOCO), 'cor': BRANCO, 'vel': 1})

contador += 1
if contador >= ITERACOES:
    # adicionando um novo bloco
    contador = 0
    posX = random.randint(0, LARGURAJANELA - TAMANHOBLOCO)
    posY = -TAMANHOBLOCO
    velRandom = random.randint(1, VEL + 3)
    blocos.append({'objRect': pygame.Rect(posX, posY, TAMANHOBLOCO, TAMANHOBLOCO), 'cor': BRANCO, 'vel': velRandom})

# preenchendo o fundo de janela com a cor preta
janela.fill(PRETO)

# movendo o jogador
moverJogador(jogador, teclas, (LARGURAJANELA, ALTURAJANELA))

# desenhando jogador
pygame.draw.rect(janela, jogador['cor'], jogador['objRect'])

# checando se jogador bateu em algum bloco ou se bloco saiu da janela para retirá-lo da lista
for bloco in blocos[:]:
    bateu = jogador['objRect'].colliderect(bloco['objRect'])
    if bateu or bloco['objRect'].y > ALTURAJANELA:
        blocos.remove(bloco)

# movendo e desenhando os blocos
for bloco in blocos:
    moverBloco(bloco)
    pygame.draw.rect(janela, bloco['cor'], bloco['objRect'])

# atualizando a janela
pygame.display.update()

relogio.tick(40)

# encerrando módulos de Pygame
pygame.quit()

```

Inicialmente importamos os módulos *pygame* e *random*. Lembre-se que já vimos o módulo *random* na primeira parte do tutorial, quando estudamos o tópico “*Módulos*”.

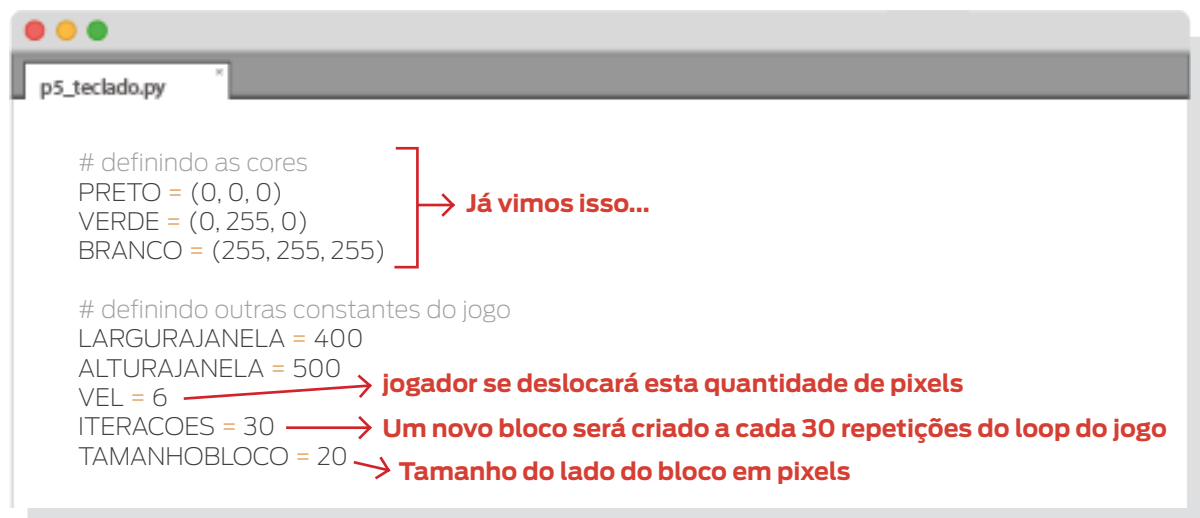


Depois definimos algumas constantes, muitas delas já usadas anteriormente, porém, existem algumas novas:

*VEL* é a “velocidade” da figura que chamaremos de “*jogador*”, que será o quadrado controlado pelo jogador. Jogador se moverá um número de pixels, na horizontal e/ou vertical, dado por esse valor.

*ITERACOES* nos guardará o número de iterações necessárias para a criação de um novo bloco (quadrados menores que se movem de cima para baixo). Ou seja, um novo bloco será criado a cada <valor de *ITERACOES*> repetições do loop do jogo.

*TAMANHOBLOCO* guardará o tamanho, em pixels, do lado de um bloco.



Também definimos a função *moverJogador()*. Ela é parecida com a função *mover()* de programas anteriores porém leva em consideração as teclas que foram pressionadas. As novas coordenadas de *jogador* serão modificadas de acordo com as teclas pressionadas e sua velocidade, desde que permaneça dentro da janela.

```
# definindo a função moverJogador(), que registra a posição do jogador
def moverJogador(jogador, teclas, dim_janela):
    borda_esquerda = 0
    borda_superior = 0
    borda_direita = dim_janela[0]
    borda_inferior = dim_janela[1]
    if teclas['esquerda'] and jogador['objRect'].left > borda_esquerda:
        jogador['objRect'].x -= jogador['vel']
    if teclas['direita'] and jogador['objRect'].right < borda_direita:
        jogador['objRect'].x += jogador['vel']
    if teclas['cima'] and jogador['objRect'].top > borda_superior:
        jogador['objRect'].y -= jogador['vel']
    if teclas['baixo'] and jogador['objRect'].bottom < borda_inferior:
        jogador['objRect'].y += jogador['vel']
```

**Jogador se move dentro das dimensões de janela**

**→ A posição de jogador são modificadas de acordo com as teclas e a velocidade**

Após, temos a definição da função *moverBloco()*. Com ela é que cada bloco guarda sua nova posição, de acordo com sua velocidade.

```
def moverBloco(bloco):
    bloco['objRect'].y += bloco['vel']
```

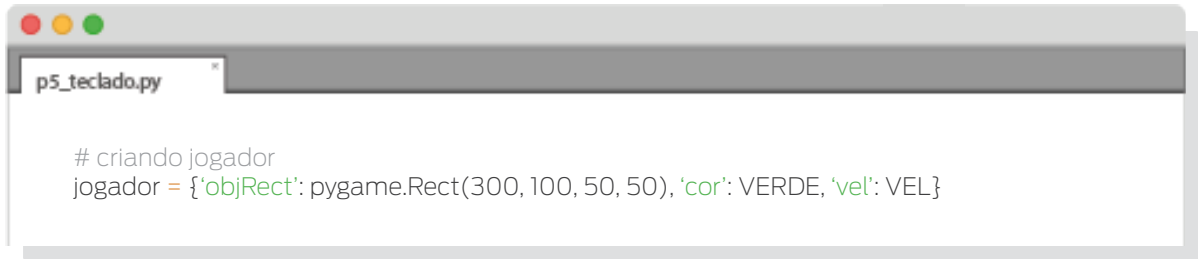
Nas seguintes linhas inicializamos os módulos de Pygame, criamos o objeto *Clock* (guardado em *relogio*) e a janela.

```
# inicializando pygame
pygame.init()

relogio = pygame.time.Clock()

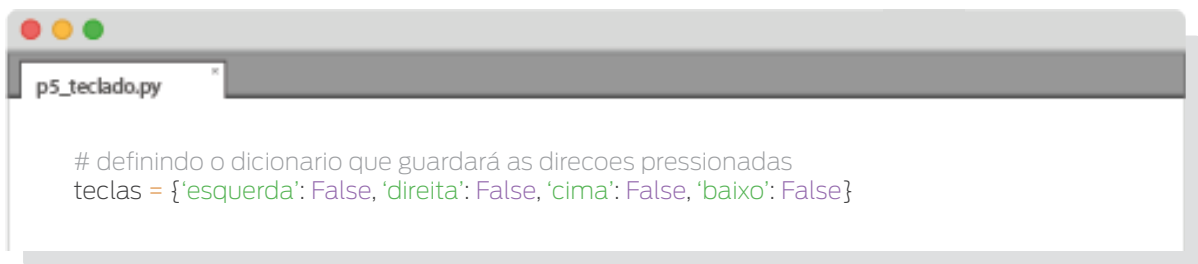
# criando janela
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Teclado e Mouse')
```

Depois criamos *jogador*, que será um dicionário com as chaves *objRect*, *cor* e *vel*. Estas chaves guardarão características referentes às coordenadas e dimensões (*objRect*), cor (*cor*) e velocidade (*vel*).



```
# criando jogador
jogador = {'objRect': pygame.Rect(300, 100, 50, 50), 'cor': VERDE, 'vel': VEL}
```

Na linha seguinte podemos ver que temos um dicionário guardado na variável *teclas*. Existem quatro chaves (*esquerda*, *direita*, *cima*, *baixo*), todas elas com o valor *False*. Este dicionário será o encarregado de registrar as teclas pressionadas. Quando alguma tecla direcional (teclas de setas e as teclas A, S, D e W) estiver sendo pressionada, o valor correspondente mudará para *True*.



```
# definindo o dicionario que guardará as direcoes pressionadas
teclas = {'esquerda': False, 'direita': False, 'cima': False, 'baixo': False}
```

Também foram criadas algumas variáveis que serão úteis, como *contador* e *blocos*, além de *deve\_continuar* que já foi usada e explicada em códigos anteriores.

A variável *contador* será responsável pelo número de repetições. Inicialmente terá o valor 0 e a cada execução do loop do jogo este valor será incrementado em uma unidade. Isto acontecerá até atingir o valor de *ITERACOES*, quando voltará ao valor 0.

Já a variável *blocos* será a encarregada de guardar todos os blocos presentes no jogo. Perceba que ela é uma lista vazia, pois, inicialmente, o jogo não contém blocos.



```
# inicializando outras variáveis
contador = 0
blocos = []
deve_continuar = True
```

Chegamos ao loop do jogo! Nele:

- Além de checar por eventos do tipo *QUIT*, checamos por eventos do tipo *KEY-DOWN*, *KEY-UP* e *MOUSEBUTTONDOWN*. Estes eventos ocorrem, respectivamente, quando pressionamos uma tecla, quando soltamos uma tecla e quando pressionamos o botão do mouse (dentro da janela).

```

while deve_continuar:
    # checando os eventos
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            deve_continuar = False

    # quando uma tecla é pressionada
    if evento.type == pygame.KEYDOWN:
        if evento.key == pygame.K_ESCAPE:
            deve_continuar = False
        if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
            teclas['esquerda'] = True
        if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
            teclas['direita'] = True
        if evento.key == pygame.K_UP or evento.key == pygame.K_w:
            teclas['cima'] = True
        if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
            teclas['baixo'] = True

    # quando uma tecla é solta
    if evento.type == pygame.KEYUP:
        if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
            teclas['esquerda'] = False
        if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
            teclas['direita'] = False
        if evento.key == pygame.K_UP or evento.key == pygame.K_w:
            teclas['cima'] = False
        if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
            teclas['baixo'] = False

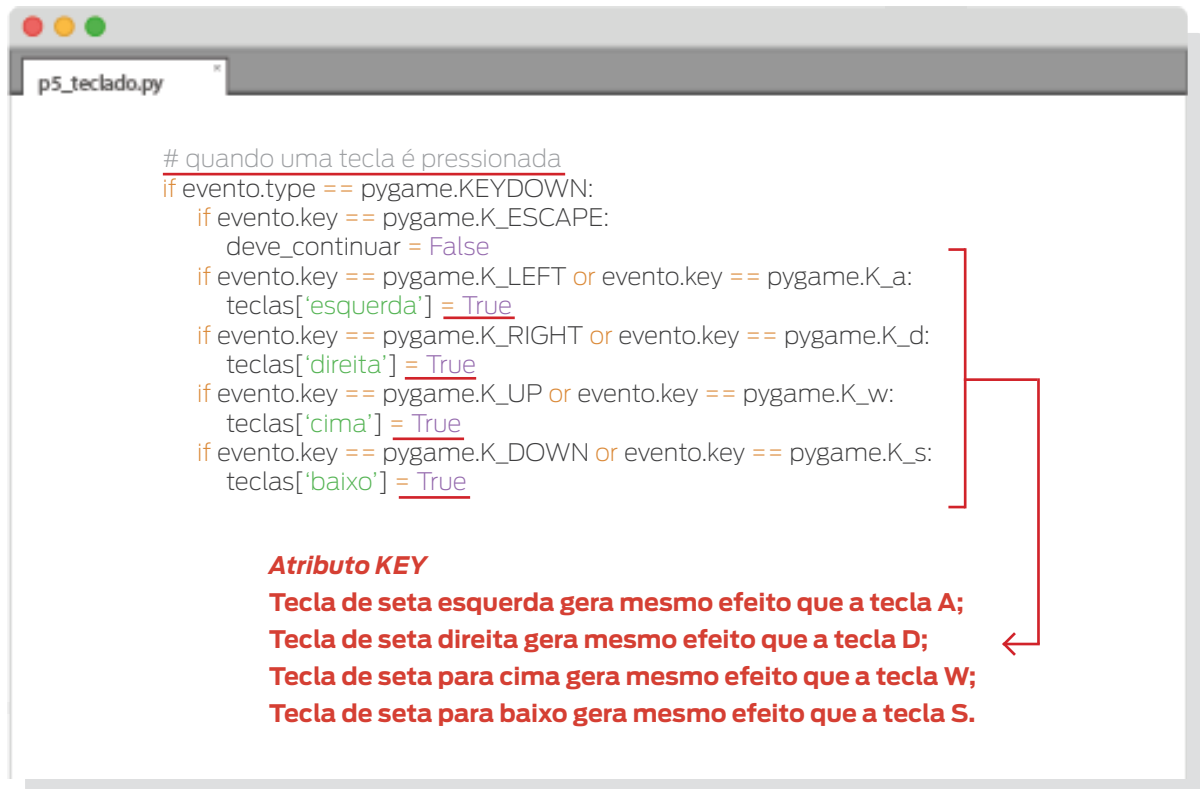
    # quando um botão do mouse é pressionado
    if evento.type == pygame.MOUSEBUTTONDOWN:
        blocos.append({'obj': Rect, 'pygame.Rect(evento.pos[0], evento.pos[1], TAMANHOBLOCO, TAMANHOBLOCO), 'cor': BRANCO, 'vel': 1})

```

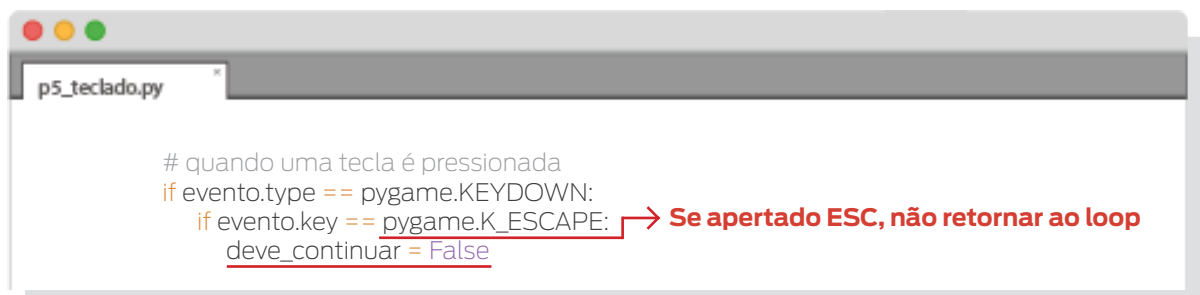
Quando o tipo de evento é *KEYDOWN*, o evento terá um atributo chamado *key*. Este atributo faz referência à tecla que provocou o evento. Podemos ver algumas “constantes” no código, como *K\_ESCAPE*, *K\_LEFT*, *K\_RIGHT*, *K\_UP* e *K\_DOWN*. Elas se referem à tecla ESC e às setas para esquerda, direita, cima e baixo, respectivamente.

As teclas alfanuméricas podem ser referidas através de *K\_<tecla>*. Para referirmos à tecla A, por exemplo, devemos fazer: *K\_a*. No código, pressionar a tecla A terá o mesmo comportamento do que pressionar a seta da esquerda. Isto mudará a chave *esquerda* do dicionário *teclas* para *True*, indicando que umas das teclas que movimentará *jogador* para a esquerda foi pressionada. Para as outras teclas de movimentação ocorre algo semelhante.





Também podemos ver, no código, que ao pressionarmos a tecla ESC a variável *deve\_continuar* mudará para *False*, com isso não se entrará no loop do jogo novamente, fazendo com que o jogo termine.

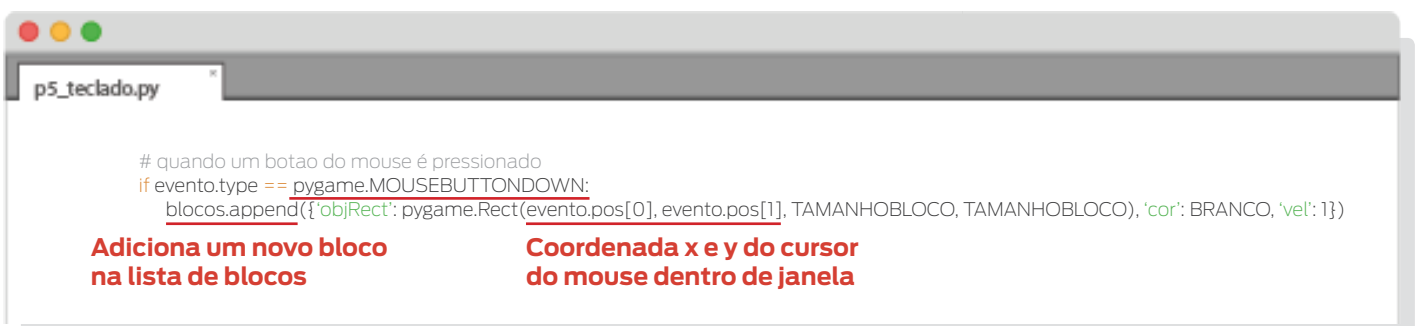


Se deixarmos de pressionar algumas das teclas, o valor da chave que indica sua direção será mudado para *False*.



```
# quando uma tecla é solta
if evento.type == pygame.KEYUP:
    if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
        teclas['esquerda'] = False
    if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
        teclas['direita'] = False
    if evento.key == pygame.K_UP or evento.key == pygame.K_w:
        teclas['cima'] = False
    if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
        teclas['baixo'] = False
```

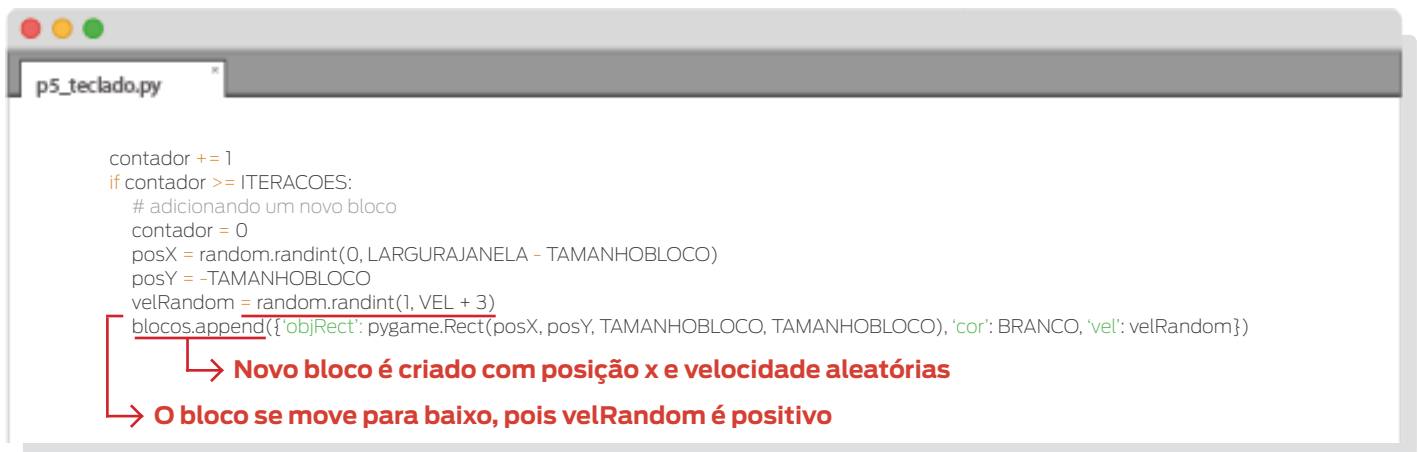
Também podemos ver que algo acontece quando pressionamos um dos botões do mouse. Isto provoca o surgimento de um novo *bloco* no local onde se encontra o cursor, no momento em que o botão é pressionado (dentro da janela). Eventos relacionados a alguma ação do mouse possuem um atributo chamado *pos*, que consiste de uma tupla contendo as coordenadas x (*pos[0]*) e y (*pos[1]*) do cursor do mouse na janela.



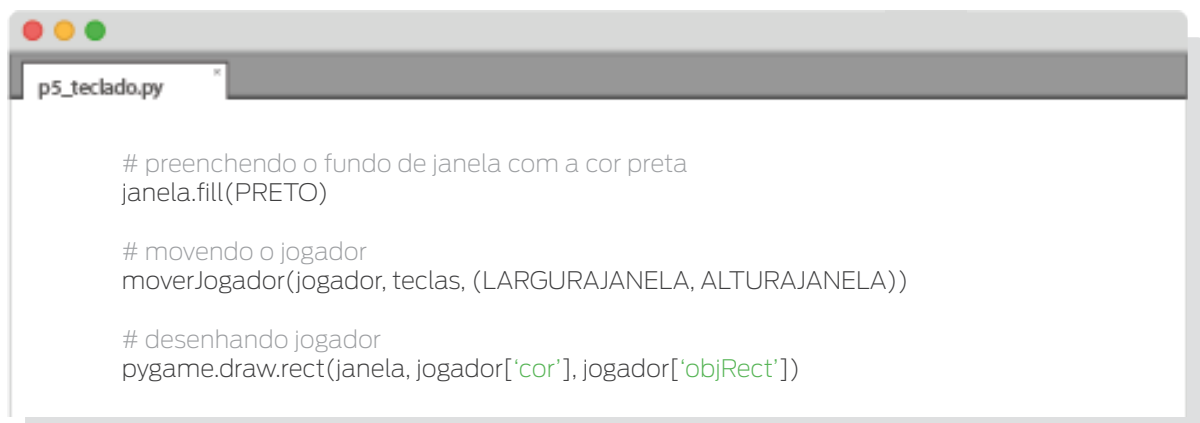
```
# quando um botao do mouse é pressionado
if evento.type == pygame.MOUSEBUTTONDOWN:
    blocos.append({'objRect': pygame.Rect(evento.pos[0], evento.pos[1], TAMANHOBLOCO, TAMANHOBLOCO), 'cor': BRANCO, 'vel': 1})
```

**Adiciona um novo bloco na lista de blocos**      **Coordenada x e y do cursor do mouse dentro de janela**

- A variável *contador* é incrementada. Se o valor dela for maior ou igual ao valor de *ITERACOES* (bastaria com que fosse igual), um novo bloco é criado em uma posição aleatória dentro da janela (no eixo x) e logo acima (no eixo y). O bloco também é criado com uma velocidade positiva no eixo y, obtida aleatoriamente entre certos valores. Como a velocidade é positiva, os objetos se “moverão” para baixo (o eixo y cresce para baixo). Para obter valores aleatórios usamos a função *randint()* do módulo *random*, ela devolve um inteiro entre os valores passados como argumento, como foi explicado quando vimos “Módulos” na primeira parte do tutorial.

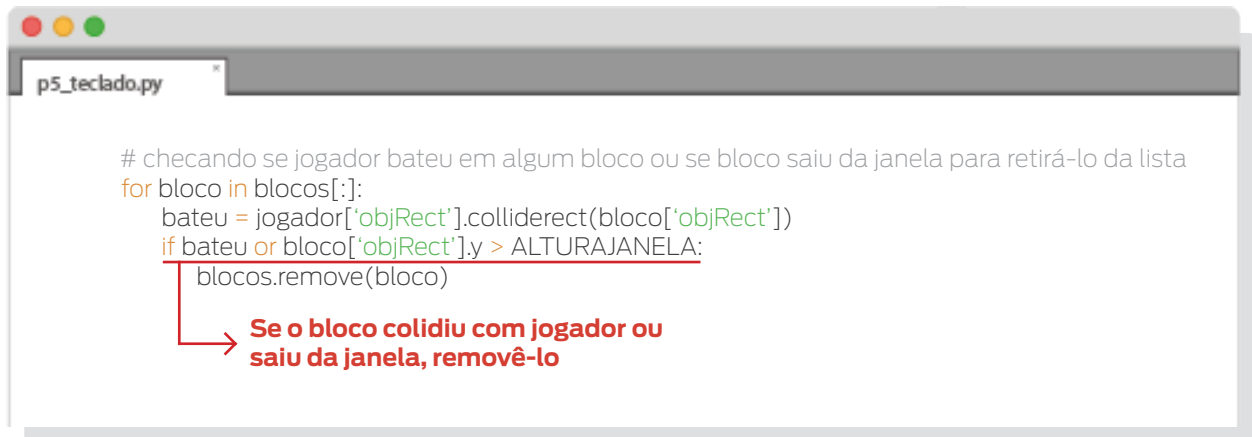


- Preenchemos o fundo da janela de preto, movemos *jogador*, e o desenhamos na janela.

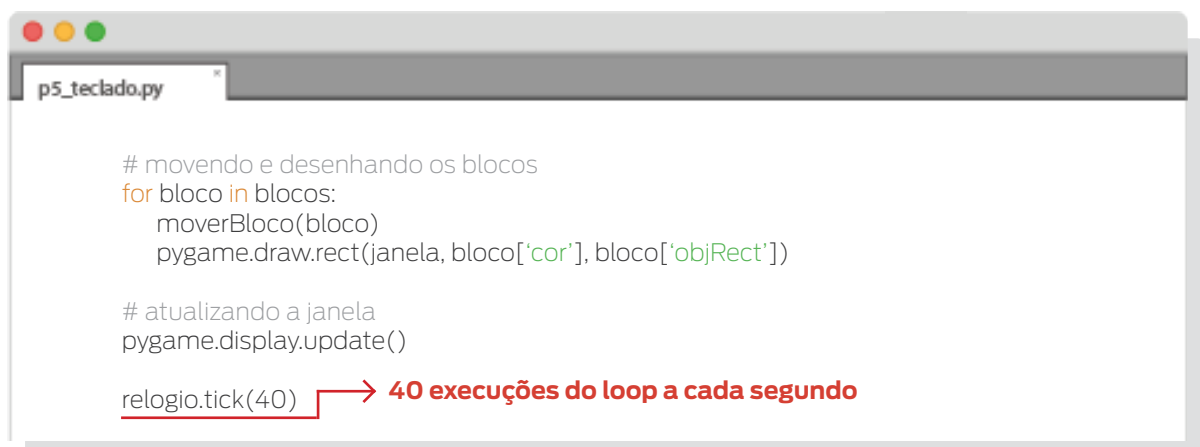


- Examinamos cada bloco para verificar se colidiu com *jogador*. Se isto ocorreu, ou se o bloco saiu da janela, o eliminamos da lista de blocos usando o método *remove()* de listas. Já falamos sobre este método quando falamos de listas na primeira parte do tutorial. Revisando, o método *remove()* elimina da lista o elemento passado como argumento.

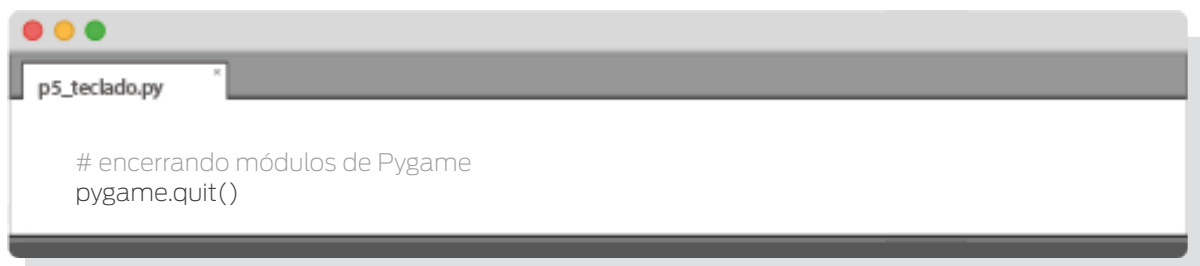
Aqui vem outra questão importante. Poderia se pensar na não necessidade de remover o bloco quando ele sai da janela, uma vez que ele não interagirá mais com *jogador*. Por mais que sua representação gráfica não esteja presente na tela, o restante de seus dados continua na memória do computador. É para evitar que informação desnecessária ocupe lugar na memória que removemos o bloco.

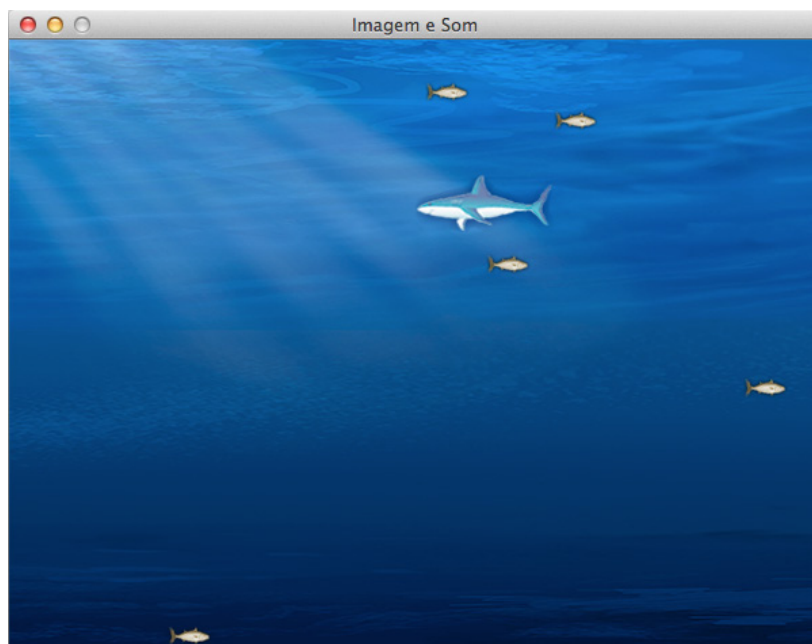


- Movemos e desenhamos cada bloco, atualizamos a janela com seus elementos e limitamos a velocidade de execução do programa a, no máximo, 40 iterações por segundo. Ou seja, em um segundo o loop do jogo se executará 40 vezes no máximo.



Uma vez fora do loop, encerramos os módulos de Pygame.





No programa anterior tivemos controle sobre algumas ações durante a execução. Pressionando algumas teclas e usando o mouse podíamos mover ou criar figuras. Neste programa trabalharemos com imagens em vez de figuras. Também aprenderemos como inserir sons em nossos programas. O código se encontra no arquivo `p6_imagensom.py`.

```

p6_imagensom.py

import pygame, random

# carregando imagens
imagemTubarao = pygame.image.load('tubarao.png')
imagemPeixe = pygame.image.load('peixe.png')
imagemFundo = pygame.image.load('cenario.png')

# definindo algumas constantes
LARGURAJANELA = imagemFundo.get_width()
ALTURAJANELA = imagemFundo.get_height()
LARGURAPEIXE = imagemPeixe.get_width()
ALTURAPEIXE = imagemPeixe.get_height()
LARGURATUBARAO = imagemTubarao.get_width()
ALTURATUBARAO = imagemTubarao.get_height()
VEL = 6
ITERACOES = 30

# definindo a função moverJogador(), que registra a posição do jogador
def moverJogador(jogador, teclas, dim_janela):
    borda_esquerda = 0
    borda_superior = 0
    borda_direita = dim_janela[0]
    borda_inferior = dim_janela[1]
    if teclas['esquerda'] and jogador['objRect'].left > borda_esquerda:
        jogador['objRect'].x -= jogador['vel']
    if teclas['direita'] and jogador['objRect'].right < borda_direita:
        jogador['objRect'].x += jogador['vel']
    if teclas['cima'] and jogador['objRect'].top > borda_superior:
        jogador['objRect'].y -= jogador['vel']
    if teclas['baixo'] and jogador['objRect'].bottom < borda_inferior:
        jogador['objRect'].y += jogador['vel']

# definindo a função moverPeixe(), que registra a posição do peixe
def moverPeixe(peixe):
    peixe['objRect'].x += peixe['vel']

# inicializando pygame
pygame.init()

relógio = pygame.time.Clock()

# criando janela
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Imagem e Som')

# criando jogador
jogador = {'objRect': pygame.Rect(300,100,LARGURATUBARAO,ALTURATUBARAO), 'imagem': imagemTubarao, 'vel': VEL}

# configurando o som
somComer = pygame.mixer.Sound('comer.wav')
pygame.mixer.music.load('musica.mid')
pygame.mixer.music.play(-1, 0.0)
somAtivado = True

# definindo o dicionário que guardará as direções pressionadas
teclas = {'esquerda': False, 'direita': False, 'cima': False, 'baixo': False}

# inicializando outras variáveis
contador = 0
peixes = []
deve_continuar = True

# loop do jogo
while deve_continuar:
    # checando os eventos
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            deve_continuar = False

```

```

# quando uma tecla é pressionada
if evento.type == pygame.KEYDOWN:
    if evento.key == pygame.K_ESCAPE:
        deve_continuar = False
    if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
        teclas['esquerda'] = True
    if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
        teclas['direita'] = True
    if evento.key == pygame.K_UP or evento.key == pygame.K_w:
        teclas['cima'] = True
    if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
        teclas['baixo'] = True
    if evento.key == pygame.K_m:
        if somAtivado:
            pygame.mixer.music.stop()
            somAtivado = False
        else:
            pygame.mixer.music.play(-1, 0.0)
            somAtivado = True

# quando uma tecla é solta
if evento.type == pygame.KEYUP:
    if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
        teclas['esquerda'] = False
    if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
        teclas['direita'] = False
    if evento.key == pygame.K_UP or evento.key == pygame.K_w:
        teclas['cima'] = False
    if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
        teclas['baixo'] = False

# quando um botão do mouse é pressionado
if evento.type == pygame.MOUSEBUTTONDOWN:
    peixes.append({'objRect': pygame.Rect(evento.pos[0], evento.pos[1], LARGURAPEIXE, ALTURAPEIXE), 'imagem': imagemPeixe, 'vel': VEL - 3})

contador += 1
if contador >= ITERACOES:
    # adicionando um novo peixe
    contador = 0
    posY = random.randint(0, ALTURAJANELA - ALTURAPEIXE)
    posX = -LARGURAPEIXE
    velRandom = random.randint(VEL - 3, VEL + 3)
    peixes.append({'objRect': pygame.Rect(posX, posY, LARGURAPEIXE, ALTURAPEIXE), 'imagem': imagemPeixe, 'vel': velRandom})

# preenchendo o fundo de janela com a sua imagem
janela.blit(imagemFundo, (0,0))

# movendo jogador
moverJogador(jogador, teclas, (LARGURAJANELA, ALTURAJANELA))

# desenhando jogador
janela.blit(jogador['imagem'], jogador['objRect'])

# checando se jogador comeu algum peixe ou se o peixe saiu da janela para retirá-lo da lista
for peixe in peixes[:]:
    comeu = jogador['objRect'].colliderect(peixe['objRect'])
    if comeu and somAtivado:
        somComer.play()
    if comeu or peixe['objRect'].x > LARGURAJANELA:
        peixes.remove(peixe)

# movendo e desenhando os peixes
for peixe in peixes:
    moverPeixe(peixe)
    janela.blit(peixe['imagem'], peixe['objRect'])

# atualizando a janela
pygame.display.update()

religio.tick(40)

# encerrando módulos de Pygame
pygame.quit()

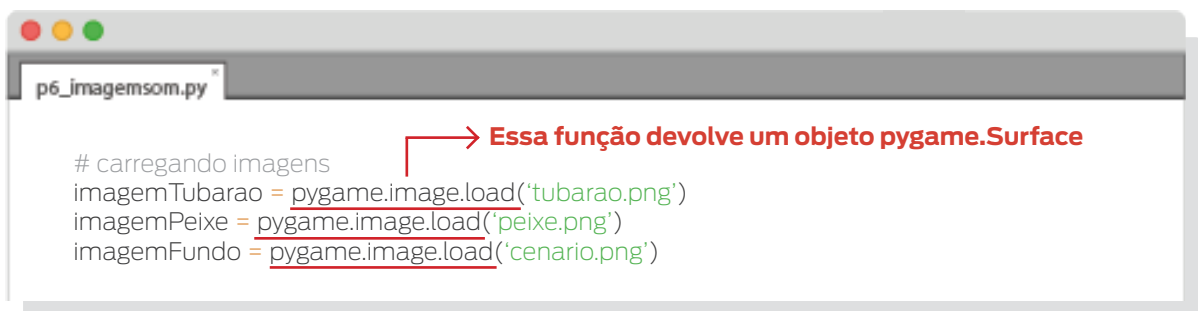
```

Inicialmente, como de costume, fazemos a importação de módulos. Para este programa importamos os módulos *pygame* e *random*.



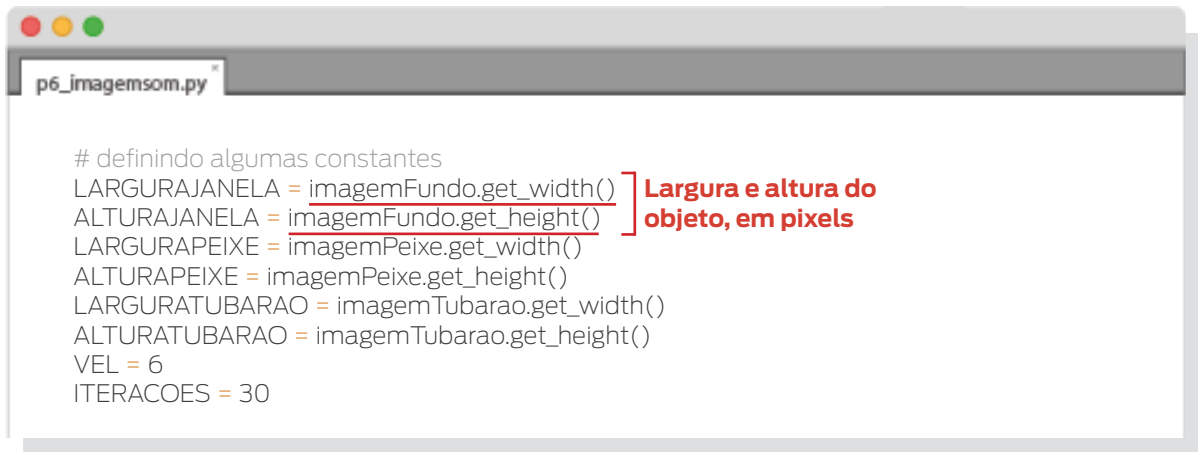
A seguir, carregamos as imagens que usaremos. No caso serão três imagens. Uma para representar o elemento que controlamos (tubarao), outra para representar o elemento com que interagimos (peixe) e a última será a imagem de fundo.

Para carregar as imagens usamos a função *load()* do módulo *image*. Devemos passar como argumento uma string com o nome do arquivo da imagem a ser carregada. Essa função devolve um objeto do tipo *pygame.Surface* que contém a imagem do arquivo desenhada na sua superfície. No código, guardamos os respectivos objetos nas variáveis *imagemTubabao* e *imagemPeixe*. Note que os arquivos de imagem devem estar na mesma pasta do arquivo com o código que as usará (no caso, *p6\_imagensom*) ou será necessário, então, colocar o endereço completo do arquivo.



Definimos algumas constantes. Algumas delas foram definidas a partir das dimensões das imagens. Objetos *Surface* possuem os métodos *get\_width()* e *get\_height()* que nos devolvem a largura e a altura do objeto em pixels. Desta forma encontramos as dimensões que usaremos mais adiante na hora de criar os dicionários que representarão o tubarão e os peixes. Note que as imagens do tubarão e peixe são bem menores da que usaremos para o fundo da janela. Caso contrário teríamos que redimensioná-las para que fiquem em um tamanho adequado.





```
# definindo algumas constantes
LARGURAJANELA = imagemFundo.get_width()
ALTURAJANELA = imagemFundo.get_height()
LARGURAPEIXE = imagemPeixe.get_width()
ALTURAPEIXE = imagemPeixe.get_height()
LARGURATUBARAO = imagemTubarao.get_width()
ALTURATUBARAO = imagemTubarao.get_height()
VEL = 6
ITERACOES = 30
```

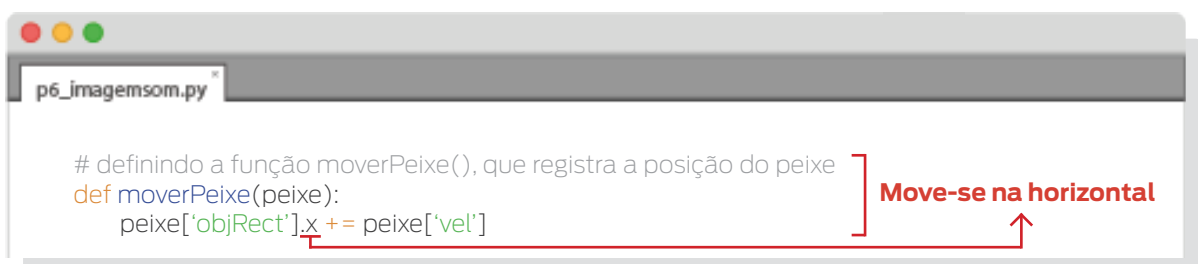
**Largura e altura do objeto, em pixels**

Definimos a função *moverJogador()* da mesma forma que fizemos no programa anterior.



```
# definindo a função moverJogador(), que registra a posição do jogador
def moverJogador(jogador, teclas, dim_janela):
    borda_esquerda = 0
    borda_superior = 0
    borda_direita = dim_janela[0]
    borda_inferior = dim_janela[1]
    if teclas['esquerda'] and jogador['objRect'].left > borda_esquerda:
        jogador['objRect'].x -= jogador['vel']
    if teclas['direita'] and jogador['objRect'].right < borda_direita:
        jogador['objRect'].x += jogador['vel']
    if teclas['cima'] and jogador['objRect'].top > borda_superior:
        jogador['objRect'].y -= jogador['vel']
    if teclas['baixo'] and jogador['objRect'].bottom < borda_inferior:
        jogador['objRect'].y += jogador['vel']
```

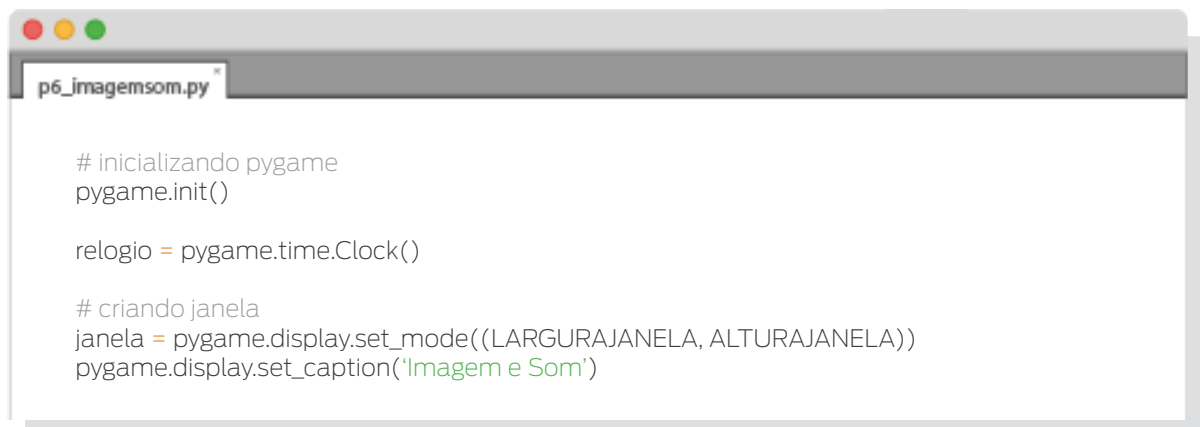
Já *moverBloco()* ficou ligeiramente alterada, também alteramos seu nome para *moverPeixe()*. No programa anterior os blocos se moviam na vertical; agora seus equivalentes, os peixes, se moverão na horizontal.



```
# definindo a função moverPeixe(), que registra a posição do peixe
def moverPeixe(peixe):
    peixe['objRect'].x += peixe['vel']
```

**Move-se na horizontal**

Inicializamos Pygame. Criamos o objeto *Clock* que cuidará da velocidade de execução. Criamos a janela e colocamos um título.

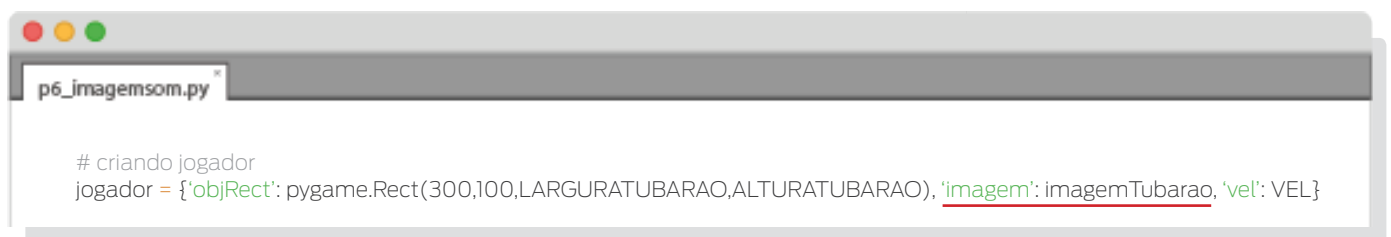


```
# inicializando pygame
pygame.init()

relogio = pygame.time.Clock()

# criando janela
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Imagem e Som')
```

Logo a seguir criamos o dicionário que representará o jogador. Ele é muito parecido ao do programa anterior, a diferença básica é que em vez de uma cor, agora usamos uma imagem. Para isso, substituímos a chave cor pela chave imagem, que terá como valor o objeto *Surface* que conterá a imagem.



```
# criando jogador
jogador = {'objRect': pygame.Rect(300,100,LARGURATUBARAO,ALTURATUBARAO), 'imagem': imagemTubarao, 'vel': VEL}
```

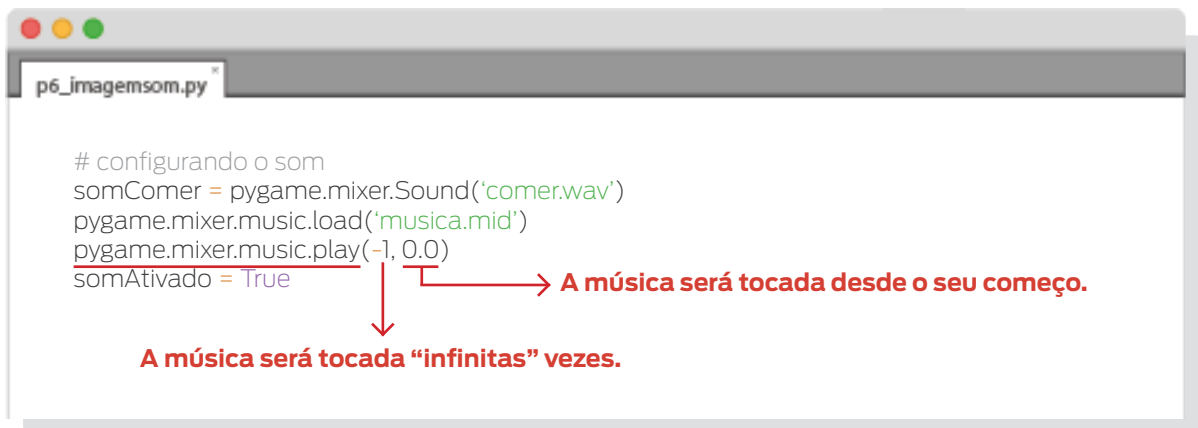
Agora temos algo novo. Vamos configurar o som a ser usado.

O módulo *pygame.mixer* é responsável pelos efeitos sonoros. Ligado a ele, encontra-se o módulo *pygame.mixer.music*, que é usado para tocar a música. Vejamos como eles funcionam.

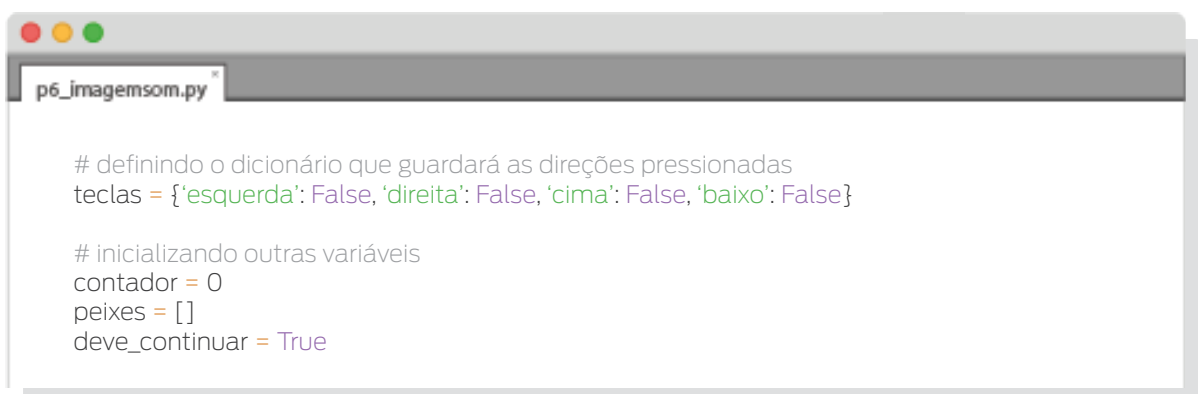
Com *pygame.mixer.Sound()* criamos um objeto do tipo *pygame.mixer.Sound* (como já feito com outros objetos, o chamaremos apenas de *Sound*). Este tipo de objeto tem o método *play()* que tocará o efeito sonoro quando for chamado. Para carregar a música, devemos chamar a função *pygame.mixer.music.load()*, passando o arquivo de som a ser tocado. Uma vez carregada, podemos tocá-la chamando a função *pygame.mixer.music.play()*. O primeiro parâmetro desta função indica quantas vezes a música será tocada após a primeira vez. Ou seja, se passarmos 5, será tocada 6 vezes (5 vezes além da primeira vez). No caso de passarmos -1, será tocada de forma repetida infinitas vezes, ou melhor, até que o programa termine.

O segundo parâmetro indica em que ponto da música ela iniciará. Passando 0.0, a música começará a ser tocada desde o começo. Se for passado 2.5, por exemplo, a música de fundo começará na posição correspondente a 2.5 segundos desde seu início.

Por último, temos uma variável booleana chamada *somAtivado* que indicará se a música e os efeitos sonoros estão ligados ou não, caso o jogador prefira jogar sem som.



Criamos o dicionário que registrará as direções pressionadas, um contador para saber quando deve ser criado um novo peixe, uma lista que conterá os peixes, e a variável que nos indicará se devemos sair do loop do jogo. Coisas com que já lidamos antes.



Enquanto isso, no loop do jogo...

- Checamos os eventos e mudamos o valor das variáveis de acordo ao realizado no último programa, com uma pequena modificação. Desta vez também checamos se a tecla M foi pressionada. Se isto ocorreu e a variável *somAtivado* for *True* (o som está ativado), desligamos a música com a função *pygame.mixer.music.stop()* e mudamos o valor da variável para *False*. Se, pelo contrário, a variável tiver o valor *False* (o som está desativado), a função *pygame.mixer.music.play()* ativará o som novamente; também mudamos o valor de *somAtivado* para *True*.

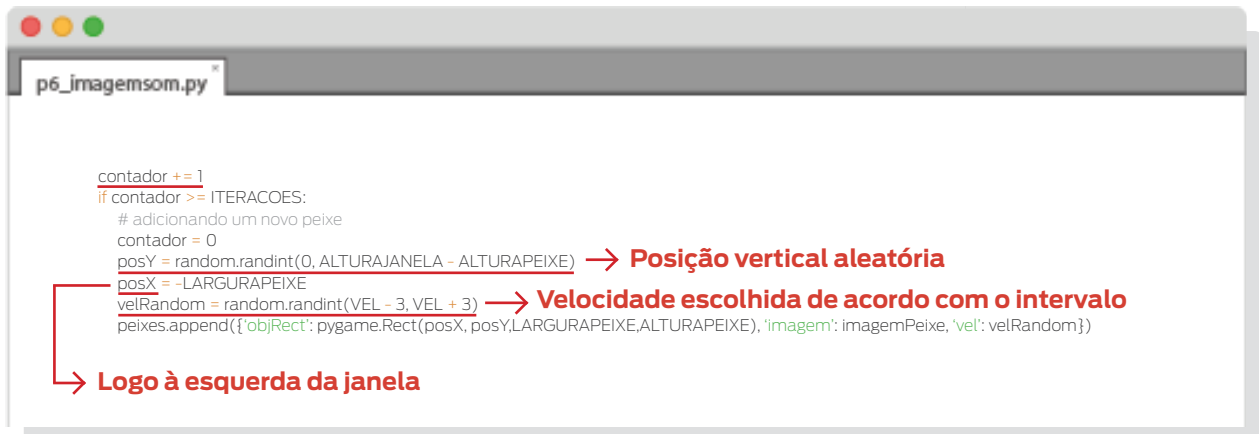
```
# loop do jogo
while deve_continuar:
    # checando os eventos
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            deve_continuar = False

    # quando uma tecla é pressionada
    if evento.type == pygame.KEYDOWN:
        if evento.key == pygame.K_ESCAPE:
            deve_continuar = False
        if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
            teclas['esquerda'] = True
        if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
            teclas['direita'] = True
        if evento.key == pygame.K_UP or evento.key == pygame.K_w:
            teclas['cima'] = True
        if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
            teclas['baixo'] = True
        if evento.key == pygame.K_m:
            if somAtivado:
                pygame.mixer.music.stop()
                somAtivado = False
            else:
                pygame.mixer.music.play(-1, 0.0)
                somAtivado = True

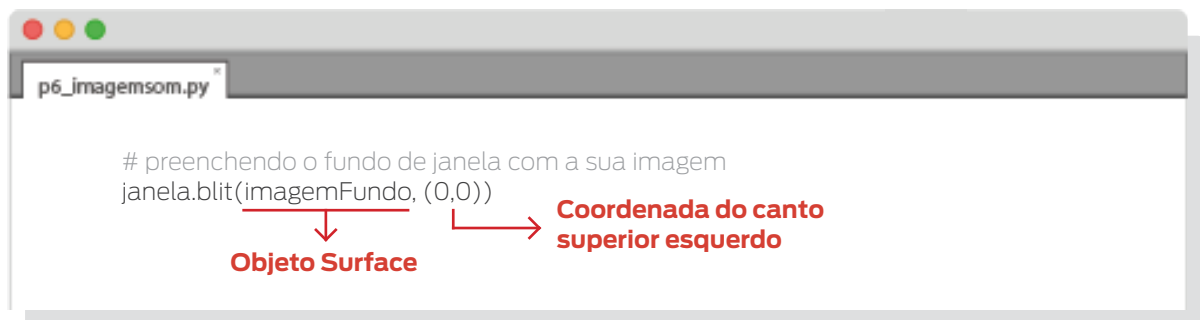
    # quando uma tecla é solta
    if evento.type == pygame.KEYUP:
        if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
            teclas['esquerda'] = False
        if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
            teclas['direita'] = False
        if evento.key == pygame.K_UP or evento.key == pygame.K_w:
            teclas['cima'] = False
        if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
            teclas['baixo'] = False

    # quando um botão do mouse é pressionado
    if evento.type == pygame.MOUSEBUTTONDOWN:
        peixes.append({'objRect': pygame.Rect(evento.pos[0], evento.pos[1], LARGURAPEIXE, ALTURAPEIXE), 'imagem': imagemPeixe, 'vel': VEL - 3})
```

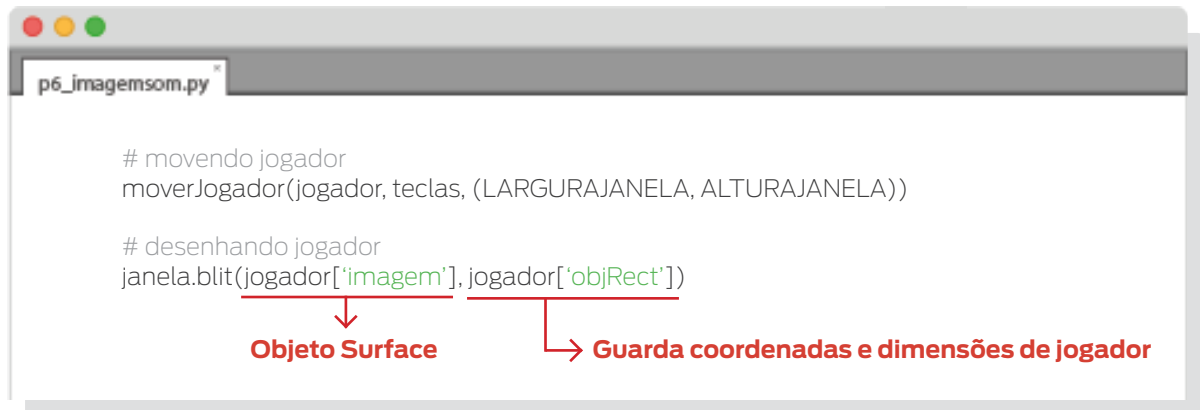
- Criamos um novo peixe se o valor de *contador* assim o indicar. Este peixe será criado logo à esquerda da janela e em uma posição vertical aleatória dentro dela. Também terá uma velocidade ligeiramente diferente daquela de *jogador*, escolhida entre certos valores.



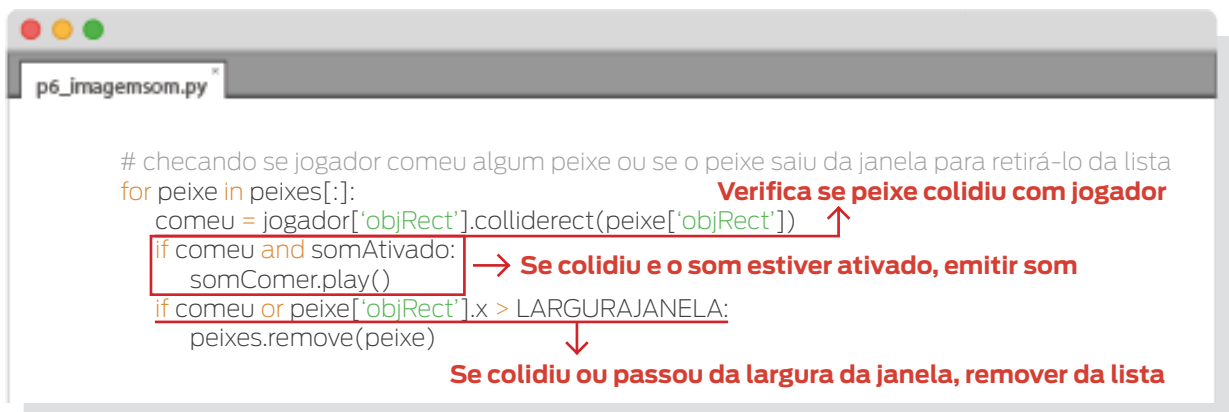
- Preenchemos o fundo da janela com a imagem desejada (guardada em *imagemFundo*). Usamos o mesmo método *blit()* visto no capítulo “Figuras e Texto”, mas desta vez passamos como argumento o objeto *Surface* que contém a imagem e a coordenada onde será colocada dentro do objeto *Surface* que o chamou (*janela*). Lembre-se que (0,0) é a coordenada do canto superior esquerdo.



- Movemos e desenhamos *jogador* na janela. Como agora temos uma imagem para *jogador*, usamos o método *blit()* de *janela*, passando o objeto *Surface* com a imagem (*jogador['imagem']*) e o objeto *Rect* que guarda as coordenadas e dimensões (*jogador['objRect']*).



- Checamos cada peixe para ver se ele colidiu com *jogador*. Se isto ocorreu e o som estiver ativado, será emitido um som. Ele também será eliminado da lista se tiver colidido ou passado da largura da janela.



- Movemos e desenhamos os peixes.



- Atualizamos a janela com seus elementos e controlamos a velocidade de execução.



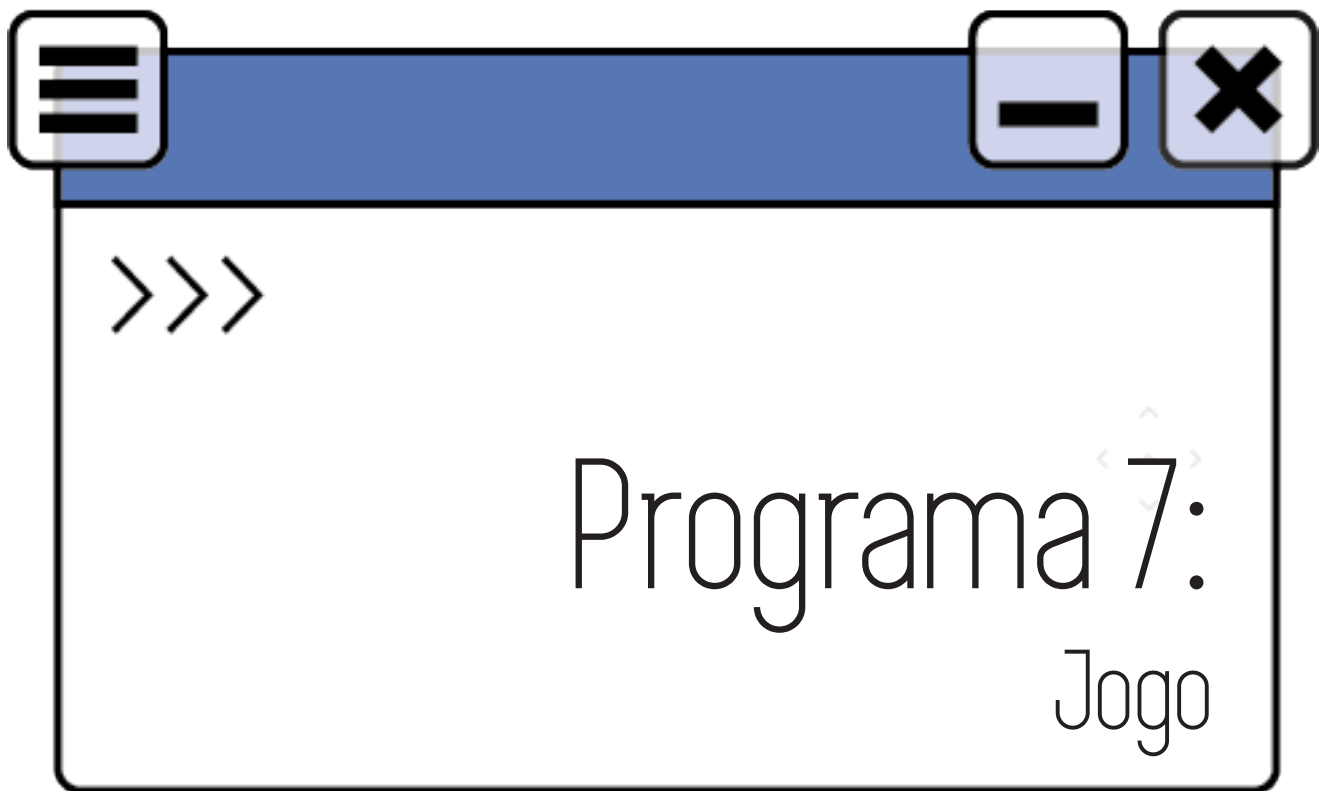
```
# atualizando a janela
pygame.display.update()

relogio.tick(40)
```

Encerramos os módulos de Pygame.



```
# encerrando módulos de Pygame
pygame.quit()
```





Começamos esta parte do curso com um programinha que nos mostrou como criar uma janela. Fomos evoluindo esse programa com a inserção de figuras, movimentação, interação entre elas, controle, uso de imagens e som; para finalmente chegarmos ao objetivo: um jogo! O código do jogo se encontra no arquivo `p7_jogo.py`. Ele consiste em uma nave, controlada pelo jogador, que tem que evitar se chocar com asteroides. Para isto ela pode desviá-los ou atirar raios laser para destruí-los. A pontuação está relacionada com o tempo de vida (mais precisamente, com o número de repetições do loop do jogo), quanto maior o tempo em que conseguir evitar os asteroides, maior a pontuação do jogador.

```

import pygame, random

# Carregando as imagens.
imagemNave = pygame.image.load('nave.png')
imagemAsteroide = pygame.image.load('asteroide.png')
imagemRaio = pygame.image.load('raio.png')
imagemFundo = pygame.image.load('magellanic-clouds.png')

LARGURAJANELA = 600          # largura da janela
ALTURAJANELA = 600          # altura da janela
CORTEXTO = (255, 255, 255)   # cor do texto (branca)
QPS = 40                    # quadros por segundo
TAMMINIMO = 10              # tamanho mínimo do asteroide
TAMMAXIMO = 40              # tamanho máximo do asteroide
VELMINIMA = 1               # velocidade mínima do asteroide
VELMAXIMA = 8               # velocidade máxima do asteroide
ITERACOES = 6               # número de iterações antes de criar um novo asteroide
VELJOGADOR = 5              # velocidade da nave
VELRAIO = (0, -15)          # velocidade do raio

LARGURANAVE = imagemNave.get_width()
ALTURANAVE = imagemNave.get_height()
LARGURARAIO = imagemRaio.get_width()
ALTURARAIO = imagemRaio.get_height()

def moverJogador(jogador, teclas, dim_janela):
    borda_esquerda = 0
    borda_superior = 0
    borda_direita = dim_janela[0]
    borda_inferior = dim_janela[1]
    if teclas['esquerda'] and jogador['objRect'].left > borda_esquerda:
        jogador['objRect'].x -= jogador['vel']
    if teclas['direita'] and jogador['objRect'].right < borda_direita:
        jogador['objRect'].x += jogador['vel']
    if teclas['cima'] and jogador['objRect'].top > borda_superior:
        jogador['objRect'].y -= jogador['vel']
    if teclas['baixo'] and jogador['objRect'].bottom < borda_inferior:
        jogador['objRect'].y += jogador['vel']

def moverElemento(elemento):
    elemento['objRect'].x += elemento['vel'][0]
    elemento['objRect'].y += elemento['vel'][1]

def terminar():
    # Termina o programa.
    pygame.quit()
    exit()

```

```

def aguardarEntrada():
    # Aguarda entrada por teclado ou clique do mouse no "x" da janela.
    while True:
        for evento in pygame.event.get():
            if evento.type == pygame.QUIT:
                terminar()
            if evento.type == pygame.KEYDOWN:
                if evento.key == pygame.K_ESCAPE:
                    terminar()
        return

def colocarTexto(texto, fonte, janela, x, y):
    # Coloca na posição (x,y) da janela o texto com a fonte passados por argumento.
    objTexto = fonte.render(texto, True, CORTEXTO)
    rectTexto = objTexto.get_rect()
    rectTexto.topleft = (x, y)
    janela.blit(objTexto, rectTexto)

# Configurando pygame, relógio, janela.
pygame.init()
relógio = pygame.time.Clock()
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Asteroides Troianos')

# Ocultando o cursor e redimensionando a imagem de fundo.
pygame.mouse.set_visible(False)
imagemFundoRedim = pygame.transform.scale(imagemFundo, (LARGURAJANELA, ALTURAJANELA))

# Configurando a fonte.
fonte = pygame.font.Font(None, 48)

# Configurando o som.
somFinal = pygame.mixer.Sound('final_fx.wav')
somRecorde = pygame.mixer.Sound('record.wav')
somTiro = pygame.mixer.Sound('laser.wav')
pygame.mixer.music.load('trilha_nave.wav')

# Tela de início.
colocarTexto('Asteroides Troianos', fonte, janela, LARGURAJANELA / 5, ALTURAJANELA / 3)
colocarTexto('Pressione uma tecla para começar', fonte, janela, LARGURAJANELA / 20, ALTURAJANELA / 2)
pygame.display.update()
aguardarEntrada()

recorde = 0
while True:
    # Configurando o começo do jogo.
    asteroides = [] # lista com os asteroides
    raios = [] # lista com os raios
    pontuacao = 0 # pontuação
    deve_continuar = True # indica se o loop do jogo deve continuar
    # direções de movimentação
    teclas = {}
    teclas['esquerda'] = teclas['direita'] = teclas['cima'] = teclas['baixo'] = False
    contador = 0 # contador de iterações
    pygame.mixer.music.play(-1, 0.0) # colocando a música de fundo

    # Criando jogador.
    posX = LARGURAJANELA / 2
    posY = ALTURAJANELA - 50
    jogador = {'objRect': pygame.Rect(posX, posY, LARGURANAVE, ALTURANAVE), 'imagem': imagemNave, 'vel': VELJOGADOR}

    while deve_continuar:
        pontuacao += 1

        if pontuacao == recorde:
            somRecorde.play()

        # Checando os eventos ocorridos.
        for evento in pygame.event.get():
            if evento.type == pygame.QUIT:
                terminar()

```

```

if evento.type == pygame.KEYDOWN:
    if evento.key == pygame.K_ESCAPE:
        terminar()
    if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
        teclas['esquerda'] = True
    if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
        teclas['direita'] = True
    if evento.key == pygame.K_UP or evento.key == pygame.K_w:
        teclas['cima'] = True
    if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
        teclas['baixo'] = True
    if evento.key == pygame.K_SPACE:
        raio = {'objRect': pygame.Rect(jogador['objRect'].centerx, jogador['objRect'].top, LARGURARAIO, ALTURARAIO),
                'imagem': imagemRaio,
                'vel': VELRAIO}
        raios.append(raio)
        somTiro.play()

if evento.type == pygame.KEYUP:
    if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
        teclas['esquerda'] = False
    if evento.key == pygame.K_RIGHT or evento.key == pygame.K_d:
        teclas['direita'] = False
    if evento.key == pygame.K_UP or evento.key == pygame.K_w:
        teclas['cima'] = False
    if evento.key == pygame.K_DOWN or evento.key == pygame.K_s:
        teclas['baixo'] = False

if evento.type == pygame.MOUSEMOTION:
    # Se o mouse se move, movimenta jogador para onde o cursor está.
    centroX_jogador = jogador['objRect'].centerx
    centroY_jogador = jogador['objRect'].centery
    jogador['objRect'].move_ip(evento.pos[0] - centroX_jogador, evento.pos[1] - centroY_jogador)

if evento.type == pygame.MOUSEBUTTONDOWN:
    raio = {'objRect': pygame.Rect(jogador['objRect'].centerx, jogador['objRect'].top, LARGURARAIO, ALTURARAIO),
            'imagem': imagemRaio,
            'vel': VELRAIO}
    raios.append(raio)
    somTiro.play()

# Preenchendo o fundo da janela com a imagem correspondente.
janela.blit(imagemFundoRedim, (0,0))

# Colocando as pontuações.
colocarTexto('Pontuação: ' + str(pontuacao), fonte, janela, 10, 0)
colocarTexto('Recorde: ' + str(recorde), fonte, janela, 10, 40)

# Adicionando asteroides quando indicado.
contador += 1
if contador >= ITERACOES:
    contador = 0
    tamAsteroide = random.randint(TAMMINIMO, TAMMAXIMO)
    posX = random.randint(0, LARGURAJANELA - tamAsteroide)
    posY = - tamAsteroide
    vel_x = random.randint(-1,1)
    vel_y = random.randint(VELMINIMA, VELMAXIMA)
    asteroide = {'objRect': pygame.Rect(posX, posY, tamAsteroide, tamAsteroide),
                 'imagem': pygame.transform.scale(imagemAsteroide, (tamAsteroide, tamAsteroide)),
                 'vel': (vel_x, vel_y)}
    asteroides.append(asteroide)

# Movimentando e desenhando os asteroides.
for asteroide in asteroides:
    moverElemento(asteroide)
    janela.blit(asteroide['imagem'], asteroide['objRect'])

# Eliminando os asteroides que passam pela base da janela.
for asteroide in asteroides[:]:
    topo_asteroide = asteroide['objRect'].top
    if topo_asteroide > ALTURAJANELA:
        asteroides.remove(asteroide)

# Movimentando e desenhando os raios.
for raio in raios:
    moverElemento(raio)
    janela.blit(raio['imagem'], raio['objRect'])

```

```

#Eliminando os raios que passam pelo topo da janela.
for raio in raios[:]:
    base_raio = raio['objRect'].bottom
    if base_raio < 0:
        raios.remove(raio)

# Movimentando e desenhando jogador(nave).
moverJogador(jogador, teclas, (LARGURAJANELA, ALTURAJANELA))
janela.blit(jogador['imagem'], jogador['objRect'])

# Checando se jogador ou algum raio colidiu com algum asteroide.
for asteroide in asteroides[:]:
    jogadorColidiu = jogador['objRect'].colliderect(asteroide['objRect'])
    if jogadorColidiu:
        if pontuacao > recorde:
            recorde = pontuacao
            deve_continuar = False
        for raio in raios[:]:
            raioColidiu = raio['objRect'].colliderect(asteroide['objRect'])
            if raioColidiu:
                raios.remove(raio)
                asteroides.remove(asteroide)

pygame.display.update()
relogio.tick(QPS)

# Parando o jogo e mostrando a tela final.
pygame.mixer.music.stop()
somFinal.play()
colocarTexto('GAME OVER', fonte, janela, (LARGURAJANELA / 3), (ALTURAJANELA / 3))
colocarTexto('Pressione uma tecla para jogar.', fonte, janela, (LARGURAJANELA / 10), (ALTURAJANELA / 2))
pygame.display.update()

# Aguardando entrada por teclado para reiniciar o jogo ou sair.
aguardarEntrada()
somFinal.stop()

```

Importamos os módulos e carregamos as imagens que usaremos. Definimos algumas constantes e a função que cuidará da movimentação da nave.

```

p7_jogo.py

import pygame, random

# Carregando as imagens.
imagemNave = pygame.image.load('nave.png')
imagemAsteroide = pygame.image.load('asteroide.png')
imagemRaio = pygame.image.load('raio.png')
imagemFundo = pygame.image.load('magellanic-clouds.png')

LARGURAJANELA = 600      # largura da janela
ALTURAJANELA = 600      # altura da janela
CORTEXTO = (255, 255, 255) # cor do texto (branca)
QPS = 40                 # quadros por segundo
TAMMINIMO = 10           # tamanho mínimo do asteroide
TAMMAXIMO = 40           # tamanho máximo do asteroide
VELMINIMA = 1            # velocidade mínima do asteroide
VELMAXIMA = 8            # velocidade máxima do asteroide
ITERACOES = 6            # número de iterações antes de criar um novo asteroide
VELJOGADOR = 5           # velocidade da nave
VELRAIO = (0,-15)        # velocidade do raio

LARGURANAVE = imagemNave.get_width()
ALTURANAVE = imagemNave.get_height()
LARGURARAIO = imagemRaio.get_width()
ALTURARAIO = imagemRaio.get_height()

def moverJogador(jogador, teclas, dim_janela):
    borda_esquerda = 0
    borda_superior = 0
    borda_direita = dim_janela[0]
    borda_inferior = dim_janela[1]
    if teclas['esquerda'] and jogador['objRect'].left > borda_esquerda:
        jogador['objRect'].x -= jogador['vel']
    if teclas['direita'] and jogador['objRect'].right < borda_direita:
        jogador['objRect'].x += jogador['vel']
    if teclas['cima'] and jogador['objRect'].top > borda_superior:
        jogador['objRect'].y -= jogador['vel']
    if teclas['baixo'] and jogador['objRect'].bottom < borda_inferior:
        jogador['objRect'].y += jogador['vel']

```

Usaremos a função *moverElemento()* para mover tanto os asteroides como os raios laser, uma vez que a movimentação destes elementos se comporta de forma parecida.

```

p7_jogo.py

def moverElemento(elemento):
    elemento['objRect'].x += elemento['vel'][0]
    elemento['objRect'].y += elemento['vel'][1]

```

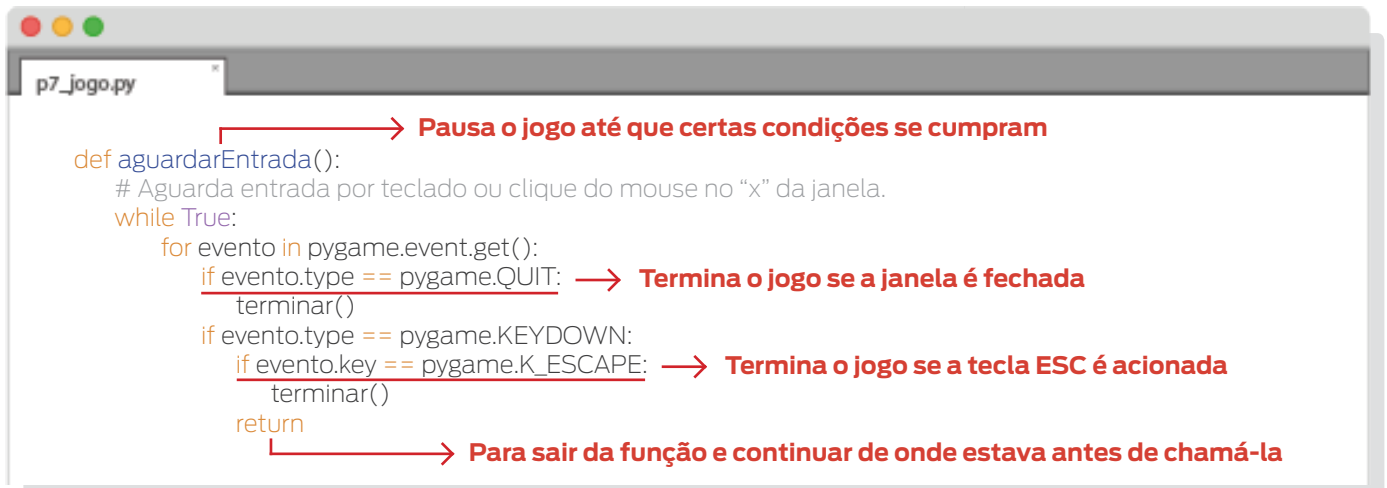
Definimos algumas outras funções, também.

A função *terminar()*, como seu nome indica, termina o jogo. Podemos ver que fechamos os módulos de Pygame e chamamos a função *exit()*. Esta é uma função de Python que termina imediatamente um programa.



```
def terminar():
    # Termina o programa.
    pygame.quit()
    exit()
```

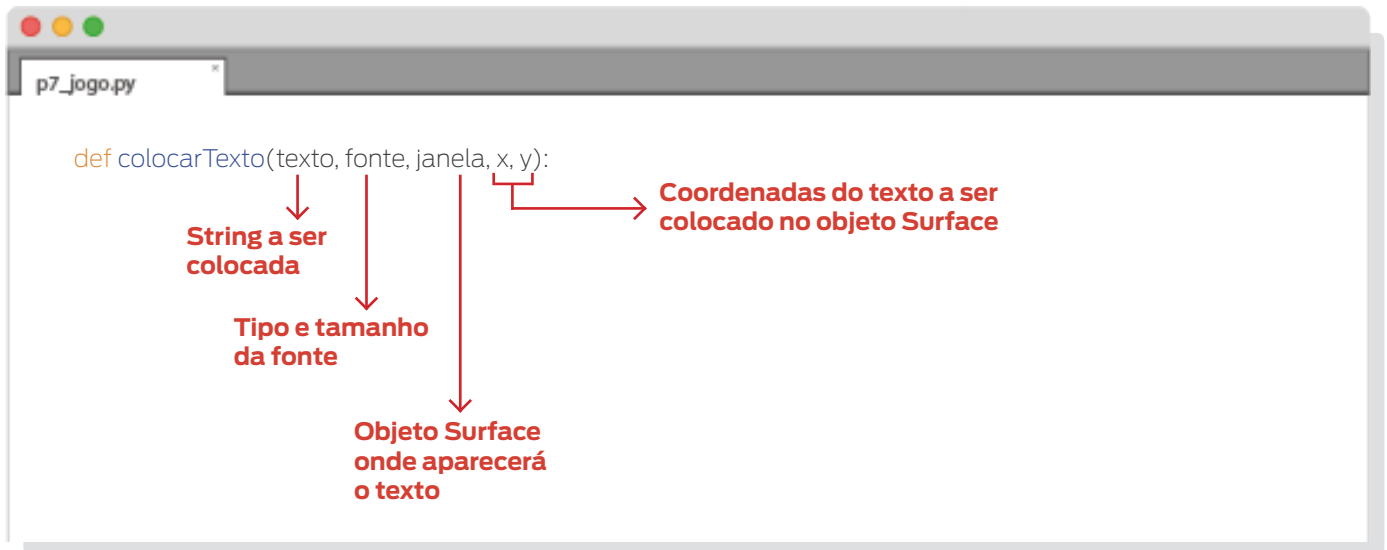
A função *aguardarEntrada()*, basicamente, pausa o jogo até que certas condições se cumpram. Se esta função for chamada, checará por eventos, sendo que terminará o jogo se houver um evento QUIT (a janela for fechada) ou se for pressionada a tecla ESC. Também pode ser visto que, se for pressionada outra tecla que não seja a ESC, é executado o comando *return*. Isto faz com que o código na função deixe de ser executado e se continue a partir do ponto de onde a função foi chamada.



```
def aguardarEntrada():
    # Aguarda entrada por teclado ou clique do mouse no "x" da janela.
    while True:
        for evento in pygame.event.get():
            if evento.type == pygame.QUIT: → Pausa o jogo até que certas condições se cumpram
                terminar() → Termina o jogo se a janela é fechada
            if evento.type == pygame.KEYDOWN:
                if evento.key == pygame.K_ESCAPE: → Termina o jogo se a tecla ESC é acionada
                    terminar()
        return → Para sair da função e continuar de onde estava antes de chamá-la
```

No capítulo “Figuras e Texto” mostramos como incluir um texto na janela. Neste programa também usaremos texto.

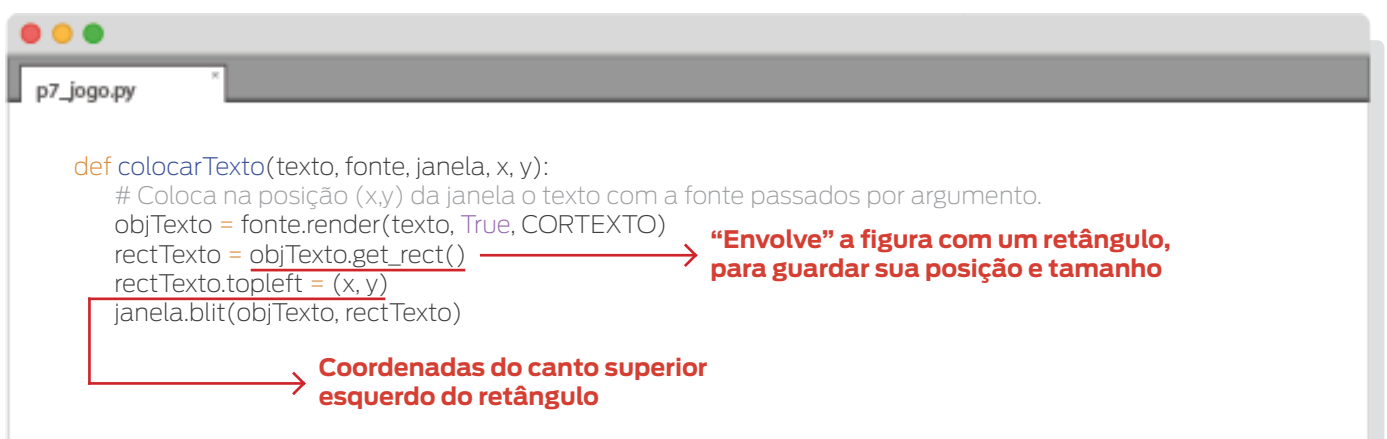
A função *colocarTexto()* se encarrega de pôr um determinado texto em um determinado lugar. Ela tem os parâmetros *texto*, *fonte*, *janela*, *x* e *y*. O primeiro é a string que será colocada; o segundo é a fonte (que inclui o tamanho) a ser usada; o terceiro é o objeto *Surface* onde aparecerá tal texto; e os dois últimos são as coordenadas, no objeto *Surface* passado como argumento, onde será colocado o texto.



No seu interior foi usado o método `get_rect()` de objetos do tipo *Surface* (*objTexto*, neste caso). Este método devolve um retângulo (objeto do tipo *Rect*) “cobrindo” a superfície do objeto que o chamou. Lembrando que um objeto *Rect* guarda as informações de tamanho e posição do objeto *Surface*.

Já falamos de alguns atributos de um objeto *Rect* como *bottom*, *top*, *left* e *right*. Outro atributo destes objetos é o *topleft*, que é como uma combinação dos atributos *top* e *left*. Assim, com uma tupla (ou lista) passamos as coordenadas do canto superior esquerdo do retângulo.

Esta função termina com o uso do método `blit()` que “desenha” o texto na janela.



Após inicializarmos os módulos de Pygame, criamos o objeto *relogio* que controlará a velocidade de execução, criamos *janela* e colocamos um título.

```
p7_jogo.py

# Configurando pygame, relogio, janela.
pygame.init()
relogio = pygame.time.Clock()
janela = pygame.display.set_mode((LARGURAJANELA, ALTURAJANELA))
pygame.display.set_caption('Asteroídes Troianos')
```

Ocultamos o cursor do mouse com a função `pygame.mouse.set_visible()` e passando *False* como argumento (*True* o torna visível). Fazemos isto porque queremos mudar a posição da nave com o uso do mouse, além do teclado, e não queremos que o cursor se sobreponha à nave.

Também mudamos o tamanho da figura de fundo, para que ela se ajuste ao tamanho da janela. No programa anterior fizemos o contrário, criamos a janela a partir do tamanho da imagem de fundo. Haverá problema neste último procedimento quando a imagem for maior do que a resolução do monitor. Para redimensionar usamos a função `pygame.transform.scale()`, passando como argumento o objeto *Surface* da imagem e uma tupla (ou lista) contendo a largura e altura que queremos para a imagem.

```
p7_jogo.py

# Ocultando o cursor e redimensionando a imagem de fundo.
pygame.mouse.set_visible(False) → Para tornar invisível o cursor do mouse
imagemFundoRedim = pygame.transform.scale(imagemFundo, (LARGURAJANELA, ALTURAJANELA))
```

↓

**Ajuste da imagem ao tamanho da janela**



Em seguida configuramos a fonte e carregamos os sons que queremos para o som final, o som quando um recorde é batido e o som quando é disparado um raio laser. Também carregamos a música do jogo.

```
p7_jogo.py

# Configurando a fonte.
fonte = pygame.font.Font(None, 48)

# Configurando o som.
somFinal = pygame.mixer.Sound('final_fx.wav')
somRecorde = pygame.mixer.Sound('record.wav')
somTiro = pygame.mixer.Sound('laser.wav')
pygame.mixer.music.load('trilha_nave.wav')
```

Usamos a função que definimos *colocarTexto()* para colocar os textos iniciais, atualizamos a janela e pausamos o jogo até que seja pressionada uma tecla que não seja ESC para iniciar o jogo.

```
p7_jogo.py

# Tela de inicio.
colocarTexto('Asteroides Troianos', fonte, janela, LARGURAJANELA / 5, ALTURAJANELA / 3)
colocarTexto('Pressione uma tecla para começar.', fonte, janela, LARGURAJANELA / 20, ALTURAJANELA / 2)
pygame.display.update() → Atualizar janela
aguardarEntrada()
    → Pausamos o jogo até que uma tecla (que não seja ESC) seja acionada
```

Chegamos a um loop *while* que contém em seu interior o loop do jogo. Esse loop se repete enquanto o jogador quiser continuar jogando após ter batido em um asteroide, evitando ter que iniciar o programa novamente para jogar. Vejamos o que acontece no loop. Mas antes de entrar nele, inicializamos a variável *recorde*. Ela guardará os recordes atualizados.

Nesse loop, acontece:

- Inicializamos todas as variáveis necessárias, ativamos a música e criamos o dicionário *jogador*, guardando nele o objeto *Rect* que o representará, a imagem e sua velocidade.

```

p7_jogo.py

recorde = 0
while True:
    # Configurando o começo do jogo.
    asteroides = []                # lista com os asteroides
    raios = []                    # lista com os raios
    pontuacao = 0                 # pontuação
    deve_continuar = True         # indica se o loop do jogo deve continuar
    # direções de movimentação
    teclas = {}
    teclas['esquerda'] = teclas['direita'] = teclas['cima'] = teclas['baixo'] = False
    contador = 0                  # contador de iterações
    pygame.mixer.music.play(-1, 0.0) # colocando a música de fundo

    # Criando jogador.
    posX = LARGURAJANELA / 2
    posY = ALTURAJANELA - 50
    jogador = {'objRect': pygame.Rect(posX, posY, LARGURANAVE, ALTURANAVE), 'imagem': imagemNave, 'vel': VELJOGADOR}

```

- Aqui vem toda a ação, é o que temos chamado de loop do jogo ao longo desta parte do tutorial. Tudo desde a aparição do cenário com a nave e asteroides, até que um asteroide se choca com ela, acontece aqui. Falaremos deste loop depois.
- Uma vez que saímos do loop do jogo (a nave se chocou com um asteroide), a música é interrompida, executando o som de fim de jogo, são colocados dois textos e a função `pygame.display.update()` é usada para mostrar tudo o que deve aparecer na janela.

```

p7_jogo.py

# Parando o jogo e mostrando a tela final.
pygame.mixer.music.stop()
somFinal.play()
→ colocarTexto('GAME OVER', fonte, janela, (LARGURAJANELA / 3), (ALTURAJANELA / 3))
→ colocarTexto('Pressione uma tecla para jogar.', fonte, janela, (LARGURAJANELA / 10), (ALTURAJANELA / 2))
pygame.display.update()

```

- Finalmente, ocorre uma pausa até que o jogador pressione uma tecla para continuar jogando ou feche o jogo (pressionando ESC ou fechando a janela). O som de fim de jogo é interrompido (caso ele já não tenha terminado).

```
p7_jogo.py

# Aguardando entrada por teclado para reiniciar o jogo ou sair.
aguardarEntrada()
somFinal.stop()
```

Agora, sim, vamos comentar mais detalhadamente o que acontece no loop do jogo.

- Incrementamos a pontuação. Se ela for igual ao recorde, o som de recorde é tocado.

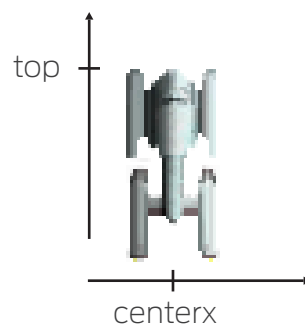
```
p7_jogo.py

while deve_continuar:
    → pontuacao += 1

    if pontuacao == recorde:
        → somRecorde.play()
```

- Checamos por eventos. Esta parte é muito parecida com a do programa anterior. As principais mudanças são a inserção da tecla ESPAÇO e a movimentação com o uso do mouse.

Quando pressionamos a tecla ESPAÇO (*pygame.K\_SPACE*, no código) criamos um raio. Perceba que o raio é criado a partir da posição do objeto *Rect* de jogador (nave). Onde foi usado a posição do centro da nave na horizontal (atributo *centerx*) e a posição do topo da nave na vertical (atributo *top*).



Após, colocamos a raio na lista de raios e tocamos o som de disparo. Tudo isto também é realizado quando pressionamos um dos botões do mouse na janela.

```
p7_jogo.py

if evento.key == pygame.K_SPACE:
    raio = {'objRect': pygame.Rect(jogador['objRect'].centerx, jogador['objRect'].top, LARGURARAIO, ALTURARAIO),
            'imagem': imagemRaio,
            'vel': VELRAIO}
    raios.append(raio)
    somTiro.play()
```

```
p7_jogo.py

if evento.type == pygame.MOUSEBUTTONDOWN:
    raio = {'objRect': pygame.Rect(jogador['objRect'].centerx, jogador['objRect'].top, LARGURARAIO, ALTURARAIO),
            'imagem': imagemRaio,
            'vel': VELRAIO}
    raios.append(raio)
    somTiro.play()
```

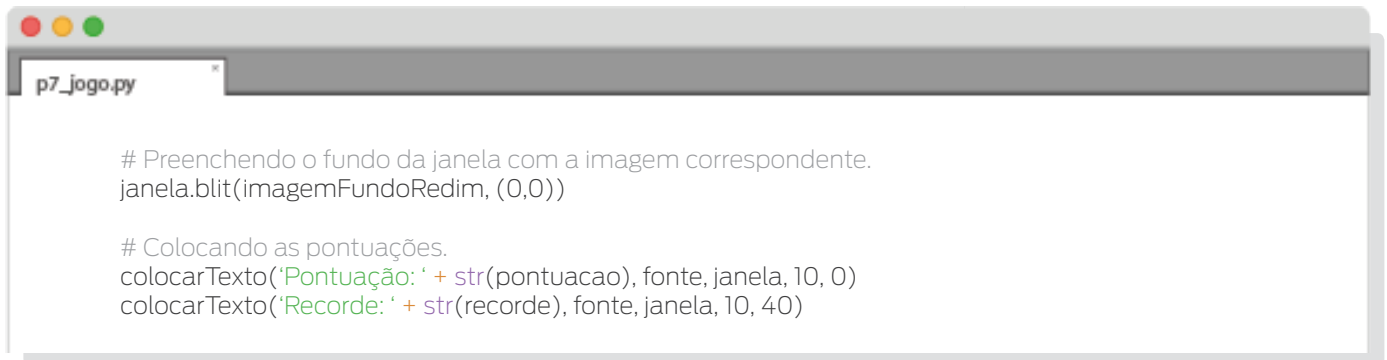
Outra diferença nesta parte é o evento do tipo *pygame.MOUSEMOTION*. Este tipo de evento é gerado quando o mouse é movimentado. Objetos deste tipo possuem um atributo chamado *pos* que guarda uma tupla com as coordenadas *x* e *y* da posição do cursor dentro da janela.

Com esta posição e com as coordenadas do centro da nave (obtidas através dos atributos *centerx* e *centery* do objeto *Rect* de *jogador*) chamamos *move\_ip()* para fazer a movimentação da nave, sendo que o centro da nave estará no lugar do cursor. O método *move\_ip()* de um objeto *Rect* movimenta-o tantos pixels para a direita e para baixo quanto os valores dos respectivos argumentos. Para mover para a esquerda ou para cima serão necessários valores negativos.

```
p7_jogo.py

if evento.type == pygame.MOUSEMOTION:
    # Se o mouse se move, movimenta jogador para onde o cursor está.
    centroX_jogador = jogador['objRect'].centerx
    centroY_jogador = jogador['objRect'].centery
    jogador['objRect'].move_ip(evento.pos[0] - centroX_jogador, evento.pos[1] - centroY_jogador)
```

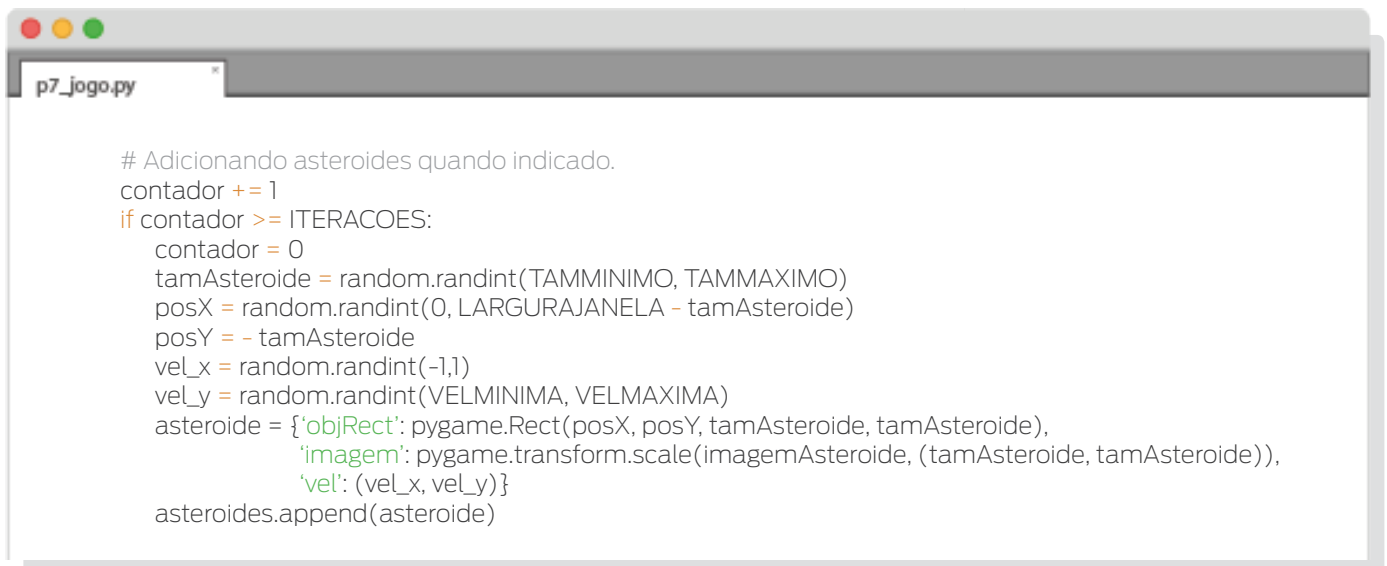
- Preenchemos o fundo da janela com a imagem desejada e colocamos as palavras “Pontuação” e “Recorde” na janela. Para colocar os textos, usamos a função `colocarTexto()`. Podemos ver que foi usada a função de *Python* `str()` para converter os números em strings.



```
# Preenchendo o fundo da janela com a imagem correspondente.
janela.blit(imagemFundoRedim, (0,0))

# Colocando as pontuações.
colocarTexto('Pontuação: ' + str(pontuacao), fonte, janela, 10, 0)
colocarTexto('Recorde: ' + str(recorde), fonte, janela, 10, 40)
```

- Quando indicado, adicionamos asteroides. Estes serão criados com tamanhos, posições e velocidades aleatórias (dentro de certos intervalos) e adicionados à lista de asteroides.



```
# Adicionando asteroides quando indicado.
contador += 1
if contador >= ITERACOES:
    contador = 0
    tamAsteroide = random.randint(TAMMINIMO, TAMMAXIMO)
    posX = random.randint(0, LARGURAJANELA - tamAsteroide)
    posY = - tamAsteroide
    vel_x = random.randint(-1,1)
    vel_y = random.randint(VELMINIMA, VELMAXIMA)
    asteroide = {'objRect': pygame.Rect(posX, posY, tamAsteroide, tamAsteroide),
                 'imagem': pygame.transform.scale(imagemAsteroide, (tamAsteroide, tamAsteroide)),
                 'vel': (vel_x, vel_y)}
    asteroides.append(asteroide)
```

- Movimentamos e desenhamos os asteroides. Também excluimos da lista de asteroides aqueles que passaram pela base da janela.

```

p7_jogo.py

# Movimentando e desenhando os asteroides.
for asteroide in asteroides:
    moverElemento(asteroide)
    janela.blit(asteroide['imagem'], asteroide['objRect'])

# Eliminando os asteroides que passam pela base da janela.
for asteroide in asteroides[:]:
    topo_asteroide = asteroide['objRect'].top
    if topo_asteroide > ALTURAJANELA:
        asteroides.remove(asteroide)

```

- Fazemos o mesmo com os raios laser. Porém, eliminamos da lista de raios aqueles que passaram pelo topo da janela.

```

p7_jogo.py

# Movimentando e desenhando os raios.
for raio in raios:
    moverElemento(raio)
    janela.blit(raio['imagem'], raio['objRect'])

# Eliminando os raios que passam pelo topo da janela.
for raio in raios[:]:
    base_raio = raio['objRect'].bottom
    if base_raio < 0:
        raios.remove(raio)

```

- Movimentamos e desenhamos a nave.

```

p7_jogo.py

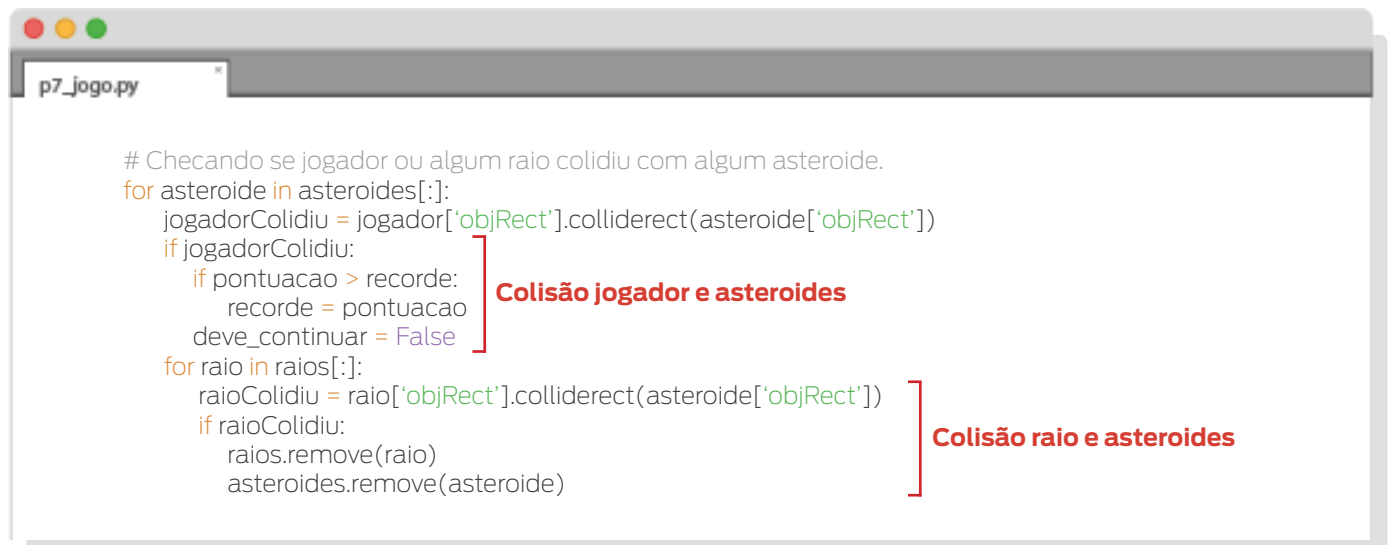
# Movimentando e desenhando jogador(nave).
moverJogador(jogador, teclas, (LARGURAJANELA, ALTURAJANELA))
janela.blit(jogador['imagem'], jogador['objRect'])

```

- Checamos se algum asteroide colidiu com a nave ou algum raio.

Se algum asteroide colidiu com a nave, verificamos se foi batido o recorde, se foi, guardamos esse valor. Também mudamos o valor da variável *deveTerminar* para *True*, com a finalidade de sair do loop do jogo após essa iteração.

Se algum raio laser colidiu com algum asteroide, removemos tanto o raio como o asteroide das respectivas listas.



```
# Checando se jogador ou algum raio colidiu com algum asteroide.
for asteroide in asteroides[:]:
    jogadorColidiu = jogador['objRect'].colliderect(asteroide['objRect'])
    if jogadorColidiu:
        if pontuacao > recorde:
            recorde = pontuacao
            deve_continuar = False
        for raio in raios[:]:
            raioColidiu = raio['objRect'].colliderect(asteroide['objRect'])
            if raioColidiu:
                raios.remove(raio)
                asteroides.remove(asteroide)
```

**Colisão jogador e asteroides**

**Colisão raio e asteroides**

- Mostramos todos os elementos na janela e controlamos a velocidade de execução.



```
pygame.display.update()
relógio.tick(QPS)
```

## Créditos:

Coordenação: Gilson Oliveira Barreto

Conteúdo: Rodrigo Castro e Giovanna Parizotto

Projeto gráfico: Igor Avelar

Editoração eletrônica: João Marcos G. Oliveira

Ilustração: Biagy

Revisão ortográfica: Maria Lourença