Necessary Structure

Every PyGame program has three parts. It begins with this exact code:

```python
# beginning: must come first
import pygame
import gamebox
camera = gamebox.Camera(800, 600)
```

You can change the 800 and 600 to be the width and height of the window you want to make instead.

The middle will usually create various gameboxes and sounds to be used later. For example:

```python
# prep work: make the boxes, sounds, and variables to be used later
logo = gamebox.from_image(0, 0, "http://www.pygame.org/docs/_static/pygame_tiny.png")
music = gamebox.load_sound("http://www.gnu.org/fun/jokes/eternal-flame.ogg")
score = 0
```

Every PyGame program should end with an event loop. There are a lot of pieces to making these work, so gamebox adds a simpler version:

```python
# make a method that will be called every frame. Must have parameter “keys”
def tick(keys):
    if pygame.K_UP in keys: # you can check which keys are being pressed
        print("the up arrow key is currently being pressed")
    if camera.mouseclick: # true if some mouse button is being pressed
        logo.center = camera.mouse # the current mouse position
    camera.draw(logo)
camera.display() # you almost always want to end this method with this line

# tell gamebox to call the tick method 30 times per second
gamebox.timer_loop(30, tick)
# this line of code will not be reached until after the window is closed
```
A gamebox is an abstraction of an image or block of color inside a rectangle. They can move, bump into one another, and change appearance.

There are four ways to make a gamebox: from a color, an image, some text, or another gamebox. All begin with the x,y of the center of the gamebox:

```
#  x,  y,  color,  width,  height
b 1  =  gamebox.from_color(50, 100, "red", 20, 40)  #  x,  y,  color
b 2  =  gamebox.from_image(10, 30, "http://pygame.org/docs/_static/pygame_tiny.png")  #  x,  y,  url_or_filename
b 3  =  gamebox.from_text(50, 100, "Hi", "Arial", 12, "red", italic=True)  #  x,  y,  text,  font,  size,  color;  optionally  True/False  for  bold  and  italic  too
b 4  =  b 2 . copy_at(50, 100)  #  x,  y
```

You can move, rotate, mirror-image, and resize gameboxes and change their image or color:

```
b 1 . move( 5 ,  6 )  #  move  5  pixels  leftward  and  6  pixels  downward
b 1 . x  -=  5  #  move  5  pixels  rightward
b 1 . left  =  20  #  puts  edge  at  particular  place  (or  right/top/bottom)
b 1 . x  =  20  #  puts  midline  at  particular  place  (y  works  too)
b 1 . center  =  [ 50 ,  100 ]  #  centers  box  at  particular  location

b 1 . speedx  =  10  #  can  set  speed  in  x  and  y
b 1 . move_speed()  #  and  move  at  current  speed

b 2 . rotate(20)  #  rotates  20  degrees  (doesn’t  work  for  color  boxes)
b 2 . flip()  #  mirror  image
b 2 . scale_by(0.5)  #  half  current  size
b 2 . full_size()  #  as  large  as  original  image  was
b 2 . width  =  20  #  scales  uniformly  so  that  width  becomes  20
b 2 . size  =  20 ,  200  #  stretch  and  squash  to  make  box  exactly  20  by  200

b 1 . color  =  "green"  #  becomes  a  green  color  box
b 1 . image  =  "python.org/images/python-logo.gif"  #  becomes  an  image  box
```

gameboxes also can detect and remove collisions:

```
print(b 1 . touches(b 2 ))  #  True  if  they  touch,  False  if  they  don’t
print(b 1 . touches(b 2 , -5 , 10))  #  if  overlap  by  at  least  5  in  x  and  within  10  in  y
print(b 1 . bottom_touches(b 2 ))  #  True  if  b 1 ’s  bottom  edge  touches  b 2
print(b 1 . contains(17 , 21))  #  True  if  the  point  (17,21)  is  inside  b 1

b 1 . move_to_stop_overlapping(b 2 )  #  b 1  will  move,  not  b 2
b 1 . move_both_to_stop_overlapping(b 2 )  #  b 1  and  b 2  move  same  amount
```

The “padding” parameters (–5 and 10 above) may be added to any “touches” method.
Sounds
You can load a sound from an .ogg, .oga, or .wav file (not from an .mp3). One source of these is
https://commons.wikimedia.org/w/index.php?title=Category:Ogg_sound_files

```python
music = gamebox.load_sound("http://www.gnu.org/fun/jokes/eternal-flame.og")
```

When you play a sound it returns a “channel” object you can use to control it later.

```python
musicplayer0 = music.play()  # play once
musicplayer1 = music.play(2)  # play three times in a row
musicplayer3 = music.play(-1)  # play forever
musicplayer3.pause()
musicplayer3.unpause()
music.stop()  # stops all the musicplayers
```

Cameras
You can only make one camera per program. The camera controls what is on the screen.
What the camera draws is invisible until after you ask it to display what it drew.

```python
camera = gamebox.Camera( width, height )
print(camera.width)  # prints the width of the camera's view in pixels
camera.clear( "red" )  # fills the screen with red

camera.draw( someBox )  # draws someBox, which should be a gamebox object
camera.draw( "Hi", "Arial", 12, "blue", 15, 30 )  # draws text in 12-point arial font
    # top left corner of text is at point (15,30)
camera.display()  # makes what's currently drawn visible

camera.move( 3, 5 )  # moves camera 3 pixels to the right and 5 pixels down
camera.left = 100  # makes the left edge of the screen be x=100
    # Likewise for .right, .top, .bottom, .x, and .y
camera.center = [5,5]  # centers the camera's view on point (5,5)
    # Likewise for .topleft, .bottomleft, .topright, etc.
```

Animation
Animation is created by changing the image in a gamebox from frame to frame. Lots of example animations can be found via an image search for “sprite sheet filetype:png.”

Sprite sheets have a grid of frames and can be turned into a list like so:

```python
# Load a grid of 4 rows and 2 columns as a list of 8 images:
sheet = load_sprite_sheet(
    4, 2)

# make a gamebox from one of those images:
b3 = gamebox.from_image(sheet[3])

# to animate, change which image is being used each time you draw
b3.image = sheet[4]
```
You can also load many individual image files instead if you want.

**Other ideas:**

To only react to a key when it is first depressed, not as long as it is held down, add `keys.clear()` to the end of your `tick` method.

```python
def tick(keys):
    if pygame.K_UP in keys:  # you can check which keys are being pressed
        print("the up arrow key was just pressed")
        keys.clear()  # makes gamebox not report the key again until it is re-pressed
        camera.display()

gamebox.timer_loop(30, tick)
```

There is also a `keys_loop` method instead of `timer_loop` if you don't need to do anything at all between keys being typed.

```python
# make a method that will be called every frame. Must have parameter “keys”
def click(key):
    if pygame.K_UP == key:
        print("the up arrow key was pressed")
        camera.display()

# tell gamebox to call the click method each time a key is pressed
gamebox.keys_loop(click)
```

You can end the game by calling `gamebox.stop_loop()`

```python
def tick(keys):
    if pygame.K_q in keys:  # if they press the q key...
        gamebox.stop_loop()  # tick will never be called again, ending the program
        camera.display()

gamebox.timer_loop(30, tick)
```

You can pause the timer with the method `gamebox.pause()` and resume it with `gamebox.unpause()`. Note, though, that `tick` will not be called after `gamebox.pause()` so you'll need to unpause in some other way.