CS: Objects in Python

Mutability

Lab - Mutability

Mutability Lab 1

Lab 1 - Intro to PyGame

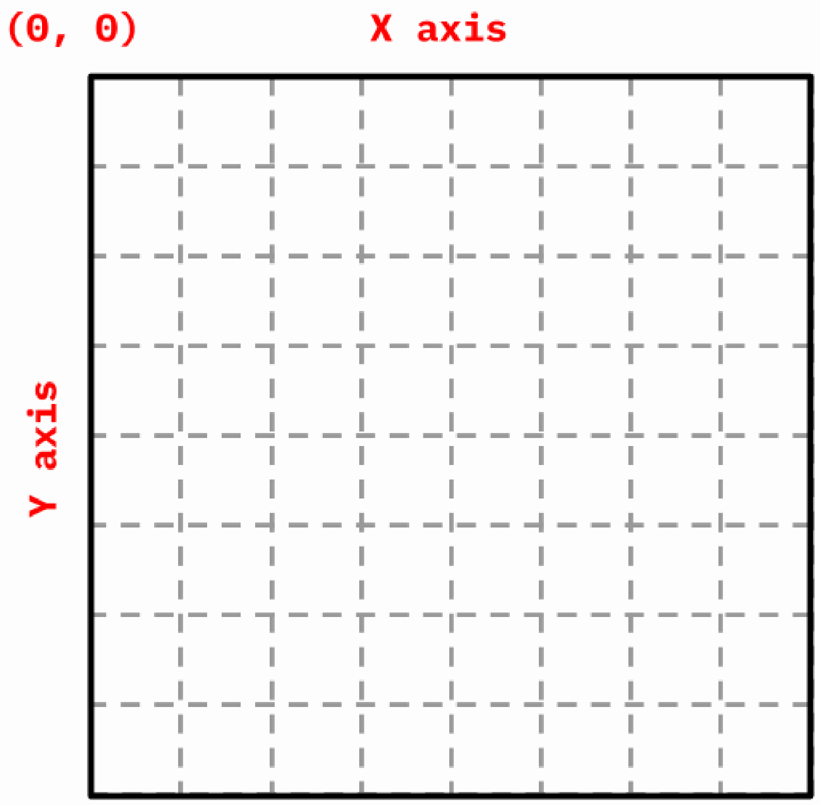
Pygame is module that allows users to quickly video games. These are typically simple, 2D games. Pygame is already installed, and we will be using it to create an animation using objects. The code below is the skeleton needed to get Pygame running on Codio.

1. Import the Pygame module.
2. Initialize the project. This, essentially, “turns on” the different features of Pygame like sound, the keyboard, the mouse, etc.
3. Create a window. This window will be 400 pixels wide by 400 pixels tall. There needs to be two sets of parentheses when creating the window.
4. Create a caption for the window
5. The clock controls how fast the animation runs
6. Create a main loop. The main loop is where the drawing and animating takes place. Without this loop, the window would appear and then disappear almost immediately. The loop is checking for the `pygame.QUIT` event, which happens when the user closes the Pygame window.
7. Quit Pygame once the main loop stops running.

```python
import pygame

pygame.init()
window = pygame.display.set_mode((400, 400))
pygame.display.set_caption("PyGame Practice")
clock = pygame.time.Clock()

run = True
while run:

  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      run = False

  clock.tick(30)
pygame.quit()
```

The Window

Before drawing anything in the window, it is important to understand how the window works. It is a Cartesian plane with an x-axis and a y-axis. However, the origin point (0, 0) is not in the center of the plane. Instead, the origin point is in the top-left corner. That means the x-values increase as you move to the right. Y-values increase as you move down the window. Keep this in mind as you draw and animate shapes on the window.



Color

Colors in Pygame are created using the RGB system. That means various amounts or red, green, and blue are mixed together to form a color. The amount of red, green, and blue are represented with a number from 0 to 255. There are many websites that can help you find the RGB values for any color.

The window is hard to see because its background is black, and the output area is also black. To change the color of the window, add the `window.fill` command inside the main loop. Then, at the end of the loop, add the command `pygame.display.update()` to update the window.

```
run = True
while run:
  pygame.time.delay(100)
  window.fill((66, 51, 255)) #change the color of the window

  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      run = False

  pygame.display.update() #update the window with any changes
  clock.tick(30)
pygame.quit()
```

Shapes and Lines

Drawing shapes and lines follow a similar pattern.

Shape	Command	Parameters
Rectangle	pygame.draw.rect	(surface, color, (x, y, width, height))
Circle	pygame.draw.circle	(surface, color, (x,y), radius)
Ellipse	pygame.draw.ellipse	(surface, color, (x, y, width, height))
Line	pygame.draw.line	(surface, color, (start_x, start_y), (end_x, end_y), width)

See the Pygame documentation for more information about other shapes and lines you can draw.

▼ **Why are there so many parentheses inside other parentheses?**
A sequence of data held inside parentheses is called a tuple. This is a data structure that will be covered at a later date. For now, it is important to know that these parentheses are required, and they contain a group of related data like x and y coordinates or red, green, and blue values.

challenge

Try these variations:

- Draw a rectangle at position 100, 100 with a width of 50 and a heght of 75. Choose any color you want.

▼ **Solution**

```
pygame.draw.rect(window, (143, 77, 112), (100, 100, 50, 75))
```

- Draw a circle at position 100, 100 with a radius of 20. Choose a different color from the rectangle.

▼ **Solution**

```
pygame.draw.circle(window, (13, 177, 212), (100, 100), 20)
```

- Draw an ellipse at position 200, 350 with a width of 200 and a heght of 50. Choose any color you want.

▼ **Solution**

```
pygame.draw.ellipse(window, (200, 20, 100), (200, 350, 200, 50))
```

- Draw a diagonal line from the top-left to the bottom-right. Use any color and width you want.

▼ **Solution**

```
pygame.draw.line(window, (231, 76, 60), (0, 0), (400, 400), 5)
```

▼ **The Position of the Code for Shapes and Lines is Important**

If shapes or lines are drawn before the `window.fill` command, they will not be seen. That is because the `window.fill` command covers the entire window (including the newly drawn shapes and lines) with the color. Similarly, the shapes and lines should be drawn before the `pygame.display.update()` command. Finally, do not draw shapes and lines inside of the for loop that checks for `pygame.QUIT`.

Mutability Lab 2

Lab 2 - Ball Class

The purpose of this lab is to build a bouncing ball animation with objects and Pygame. Before the animation can take place, the `Ball` class needs to be defined. Since the animation is built on top of Pygame, look at how to draw a circle in Pygame. That will inform you on how to structure the class.

```
pygame.draw.circle(surface, color, center, radius)
```

The parameters of the circle in Pygame will be the instance variables for the class. The color will be kept as a tuple (three numbers surrounded by parentheses), but the coordinates will be separate instance variables. These variables will change independently of one another, so it is easier to not use a tuple. Start the project by importing Pygame, and then begin building the `Ball` class.

```
import pygame

class Ball:
  def __init__(self, surface, color, x, y, r):
    self.surface = surface
    self.color = color
    self.x = x
    self.y = y
    self.r = r
```

Next comes all of the setup for getting Pygame to work — initialize Pygame, create a surface, set the caption, create a main loop, etc. Run the program when done. The window should be gray.

```python
import pygame

class Ball:
  def __init__(self, surface, color, x, y, r):
    self.surface = surface
    self.color = color
    self.x = x
    self.y = y
    self.r = r

pygame.init()
window = pygame.display.set_mode((400, 400))
pygame.display.set_caption("Bouncing Ball")
clock = pygame.time.Clock()

run = True
while run:
  window.fill((120, 120, 120))

  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      run = False

  pygame.display.update()
  clock.tick(30)
pygame.quit()
```

Now that Pygame is up and running, we can turn our attention to the `Ball` class. First, instantiate a `Ball` object called `ball`. The ball should start in the middle of the window, have the color red, and have a radius of 20. The `Ball` object should be created after `window` has been declared, but before the main loop.

```
pygame.init()
window = pygame.display.set_mode((400, 400))
pygame.display.set_caption("Bouncing Ball")
clock = pygame.time.Clock()
ball = Ball(window, (255, 0, 0), 200, 200, 20)

run = True
```

The `ball` object does not do anything right now. Add the instance method `draw` which will draw the ball on the window. This method will use the instance variables to draw a circle using the Pygame syntax.

```
class Ball:
  def __init__(self, surface, color, x, y, r):
    self.surface = surface
    self.color = color
    self.x = x
    self.y = y
    self.r = r

  def draw(self):
    pygame.draw.circle(self.surface, self.color, (self.x, self.y), sel
```

Call this method to draw the ball to the screen. Since the ball will eventually be animated, the `draw` method should be called from within the main loop and after the window has been filled.

```python
run = True
while run:
  window.fill((120, 120, 120))
  ball.draw()

  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      run = False

  pygame.display.update()
  clock.tick(30)
pygame.quit()
```

Mutability Lab 3

Lab 3 - Animating the Ball

Now that the ball appears on the screen, it is time to make the ball move. The movement should be two dimensional. That means `self.x` and `self.y` should both change over time. If the same change is applied to `self.x` and `self.y` equally, the ball will only move at a 45-degree angle. A more realistic animation will allow for a greater variation in movement. So two more instance variables need to be added to the `Ball` class. One will control the velocity in the x-direction, and the other will control the velocity in the y-direction.

```
class Ball:
  def __init__(self, surface, color, x, y, r):
    self.surface = surface
    self.color = color
    self.x = x
    self.y = y
    self.r = r
    self.vel_x = 1
    self.vel_y = 2
```

Next, there needs to be a new instance method that updates the position of the ball based on the newly created instance variables. Create the method `update`. It will add the x-velocity to the x-position, as well as add the y-velocity to the y-position. Note, Pygame requires that positions on the screen be expressed as integer values.

```
  def update(self):
    self.x += self.vel_x
    self.y += self.vel_y
```

Call this method after `ball` is drawn to the screen.

```
run = True
while run:
  window.fill((120, 120, 120))
  ball.draw()
  ball.update()

  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      run = False

  pygame.display.update()
  clock.tick(30)
pygame.quit()
```

▼ **Why is the animation not always smooth?**
This has to do with how Pygame works and how Codio was built. Pygame uses something called X server to display graphical output. X server runs on a machine, not in a browser. Codio was designed to have a coding environment run in your browser. To get Pygame output into your browser, X server is running on a server farm somewhere far away. Your Pygame code gets sent to the server farm, Pygame output is generated, and then sent to your browser. This means Pygame output depends on network speeds. If your internet connection is not very good, or there is lots of network traffic, this will decrease the quality of your animation.

The animation should work, but the ball disappears off the screen. It is time to make the ball bounce. The general steps to getting ball to bounce are:

1. Use `self.x` and `self.y` and ask if the ball is at the edge of the window
2. If yes, then change direction of the velocity

The ball will bounce, but the animation will not look quite right. That is because the location of the circle is its center. So asking if `self.x` is less than 0 (the left side of the window) means that this boolean expression will be true when half the circle is off of the window. A better way to build the animation is to ask if `self.x` is less than `self.r` (which is the radius of the circle). The animation on the left tests if the ball is at the edge of the window. The animation on the right tests if the ball is touching the edge of the window plus or minus the radius.

▼ **Why is this animation much smoother than mine?**

The animation above is written in JavaScript. This language runs entirely in the browser. So internet connectivity or network traffic will not affect the animation once the JavaScript code has been downloaded to your browser.

Bouncing should take place in the `update` method. This function currently updates the position of the circle. To make things more clear, we are going to refactor (rewrite) this function. The `update` method should move the ball and then bounce the ball if necessary. These two tasks will have their own methods. All the `update` method should do is call the methods `move` and `bounce`.

```
def update(self):
  self.move()
  self.bounce()
```

The `move` method updates the x and y coordinates of the circle with their respective velocities.

```python
def move(self):
  self.x += self.vel_x
  self.y += self.vel_y
```

The `bounce` method will change the direction of the ball if the distance from the center of the ball to the edge of the window is less than the radius. The boolean expressions that ask if the position is `< self.r` are testing the left and top edges of the window. The boolean expressions that ask if the position is `> 400 - self.r` are testing the right and bottom edges of the window. The act of bouncing happens when the velocity changes direction. Multiplying a number by -1 changes the sign from positive to negative or from negative to positive. So if the ball is touching the left or right sides, multiply `self.vel_x` by -1, and if the ball is touching the top or bottom multiply `self.vel_y` by -1.

```python
def bounce(self):
    if self.x < self.r or self.x > 400 - self.r:
      self.vel_x *= -1
    if self.y < self.r or self.y > 400 - self.r:
      self.vel_y *= -1
```

You should now have a complete bouncing ball animation.

▼ **Solution**

```python
  import pygame
  class Ball:
    def __init__(self, surface, color, x, y, r):
      self.surface = surface
      self.color = color
      self.x = x
      self.y = y
      self.r = r
      self.vel_x = 1
      self.vel_y = 2

    def draw(self):
      pygame.draw.circle(self.surface, self.color, (self.x, self.y), s
    def update(self):
      self.move()
      self.bounce()

    def move(self):
      self.x += self.vel_x
      self.y += self.vel_y

    def bounce(self):
      if self.x < self.r or self.x > 400 - self.r:
        self.vel_x *= -1
      if self.y < self.r or self.y > 400 - self.r:
        self.vel_y *= -1

  pygame.init()
  window = pygame.display.set_mode((400, 400))
  pygame.display.set_caption("Bouncing Ball")
  clock = pygame.time.Clock()
  ball = Ball(window, (255, 0, 0), 20, 20, 20)
  run = True
  while run:
    window.fill((120, 120, 120))
    ball.draw()
    ball.update()

    for event in pygame.event.get():
      if event.type == pygame.QUIT:
        run = False
    pygame.display.update()
    clock.tick(30)
  pygame.quit()
```

Mutability Lab 4

Lab 4 - Tips and Tricks

Change Colors

One way to make the animation more interesting would be to have the ball change color each time it bounces. Since color is represented by three numbers, you could add to or subtract from these numbers. Doing so would only make minor changes to the color. Colors are represented by number between 0 and 255, so you have to think about what happens when the color values are smaller than 0 or greater than 255. A better way to implement this is to choose three random numbers each time the ball bounces. To do this, import the `random` module at the top of the program.

```
import pygame
import random

class Ball:
  def __init__(self, surface, color, x, y, r):
```

Next, modify the `bounce` method so that `self.color` has a new value. Be sure to add the method call for `self.change_color()` to both conditionals. If not, the ball will only change color when it hits two of the four walls.

```
def bounce(self):
    if self.x < self.r or self.x > 400 - self.r:
      self.vel_x *= -1
      self.change_color()
    if self.y < self.r or self.y > 400 - self.r:
      self.vel_y *= -1
      self.change_color()
```

Finally, declare the `change_color` method. To make the code easy to read, the variables `red`, `green`, and `blue` each get a random integer between 0 and 255. These new values will be the new color. Do not forget the parentheses as colors in Pygame are stored as a tuple.

```
def change_color(self):
  red = random.randint(0, 255)
  green = random.randint(0, 255)
  blue = random.randint(0, 255)
  self.color = (red, green, blue)
```

Random Direction

The animation always starts in the same way. It would be more interesting if the ball moved in a randomly selected direction. Using the `randint` method from above, a random value for `self.vel_x` and `self.vel_y` seems pretty easy; just use `random.randint(-3, 3)`. However, there is a one-in-seven chance that 0 will be selected. That means the ball will not move diagonally, and perhaps not move at all if both velocities are 0. What you really want to do is pick a random number between -3 and -1 or between 1 and 3. In the constructor, set the values of `self.vel_x` and `self.vel_y` to `Ball.random_velocity`.

```
class Ball:
  def __init__(self, surface, color, x, y, r):
    self.surface = surface
    self.color = color
    self.x = x
    self.y = y
    self.r = r
    self.vel_x = Ball.random_velocity()
    self.vel_y = Ball.random_velocity()
```

Since the `random_velocity` method is not changing an instance variable (it is just returning a value), this would be a good time to use a static method. That also explains why the method call is `Ball.random_velocity` and not `self.random_velocity`. This method is going to use `random.choice` which takes a list as a parameter. Python will randomly select one element from the list. The list `direction` contains the strings `"positive"` and `"negative"`. If `"positive"` is selected, the velocity will be between 1 and 3. If `"negative"` is selected, the velocity will be between -3 and -1.

```
@staticmethod
def random_velocity():
  direction = random.choice(["positive", "negative"])
  if direction == "positive":
    return random.randint(1, 3)
  else:
    return random.randint(-3, -1)
```

Avoiding Hard Coding

The ball always starts in the middle of the window. That is because the window is 400 by 400 and the ball's starting position is (200, 200). Change the window dimensions to 500 by 500, and the ball is no longer in the middle. That is because the starting position is hard coded into the program. That is, the starting position is a fixed number that is independent of the window dimensions. A better way to create the animation is to make the starting position dependent upon the window. Before `window` is declared, create the variables `WIDTH` and `HEIGHT` and set their values to 400. Use these variables when setting the size of the window. Finally, set the ball's starting position to middle of the window (divide these variables by 2). **Note**, Pygame requires that x and y positions be represented as an integer. Make sure the dimensions are even numbers or use floor division to avoid floating point positions.

```
pygame.init()
WIDTH = 400
HEIGHT = 400
window = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Bouncing Ball")
clock = pygame.time.Clock()
ball = Ball(window, (255, 0, 0), WIDTH/2, HEIGHT/2, 20)
```

▼ **Why are some variables in all caps?**

A constant is a variable whose value never changes. Using all capital letters is a Python convention for declaring a constant. Since the animation window should not change once it starts, this is a good example of when to use a constant.

challenge

Try these variations:

- Change the size of the window and run the animation.
- Add a method that increases the speed of the ball each time it bounces.

▼ **Possible Solution**

Note, the x velocity increases when the ball hits the left or right sides of the window, and the y velocity when the ball hits the top or bottom of the window. So there are two methods to increase the velocity.

```
def bounce(self):
  if self.x < self.r or self.x > 400 - self.r:
    self.vel_x *= -1
    self.change_color()
    self.increase_vel_x()
  if self.y < self.r or self.y > 400 - self.r:
    self.vel_y *= -1
    self.change_color()
    self.increase_vel_y()

def increase_vel_x(self):
  if self.vel_x > 0:
    self.vel_x += 1
  else:
    self.vel_x -= 1

def increase_vel_y(self):
  if self.vel_y > 0:
    self.vel_y += 1
  else:
    self.vel_y -= 1
```

- Add a method that makes the ball grow each time it bounces.

▼ **Possible Solution**

```
def bounce(self):
  if self.x < self.r or self.x > 400 - self.r:
    self.vel_x *= -1
    self.change_color()
    self.increase_vel_x()
    self.grow_ball()
  if self.y < self.r or self.y > 400 - self.r:
    self.vel_y *= -1
    self.change_color()
    self.increase_vel_y()
    self.grow_ball()

def grow_ball(self):
  self.r += 1
```

Mutability Lab Challenge

Copy and paste the `Zoo` class below into the code editor.

```
class Zoo:
  def __init__(self, big_cats, primates, reptiles, birds):
    self.big_cats = big_cats
    self.primates = primates
    self.reptiles = reptiles
    self.birds = birds
```

Add the following methods to the class:
* `total_animals` - returns the total number of animals
* `total_mammals` - returns the number of mammals
* `most_animals` - returns the number of primates

Expected Output

If the following code is added to your program:

```
my_zoo = Zoo(10, 30, 90, 120)
print(my_zoo.total_animals())
print(my_zoo.total_mammals())
print(my_zoo.most_animals())
```

Then the output would be:

```
250
40
birds
```