Introduction to Processing

The Basics(Python Version)

Processing

- Processing started by Ben Fry and Casey Reas while both were graduate students at MIT Media Lab in 2001.
- The original language for Processing is Java. We will use the Python version. However, this Python implementation is built with Jython which still uses Python 2.x instead of the current Python 3.x. So for example, f-string(formatted-string) will not work. Otherwise, you will likely not notice the difference.
- Designed for visual artists with limited programming experience who want to create art without knowing complicated Java syntax.
- In its current version, hundred of libraries have been written for computer vision, data visualization, music composition, networking, 3D drawings and programming electronics.

Processing

Processing was created originally for the Java language. For this reason, the interface to Processing's Python version is not very "Pythonic".

I wrote some code to hide some of this interface and make it flow better with Python. In addition, I added some code to make writing games and working with images easier.(see arcade.py) Download the zip file that contains this code on our course website here.

Once you unzip the contents and open it with Processing. There should be three files:

processing_py.pyde(DO NOT MODIFY THIS FILE)

arcade.py(DO NOT MODIFY THIS FILE)

game.py(write all of your code here)

There is also a data folder where you should put all of your images for your game.

An Alternative to Processing

For those who are experienced at programming and can read and learn independently, you are highly encouraged to look at the Python Arcade library.

You will need to install Visual Studio Code or a similar IDE and follow the instructions for installing the Python Arcade Library from the website:

https://arcade.academy/

The Python Arcade library has some advanced features such as collision detection, physics engine for platformers, etc.. more than what we will cover with Processing.

Feel free to reach out with questions about Arcade. However, in class, we will only cover Processing.

game.py

All of your code should go here in game.py.

You will need to implement(provide code for) three methods/functions:

1) def __init__(self): Declare and initialize all your game/application variables.

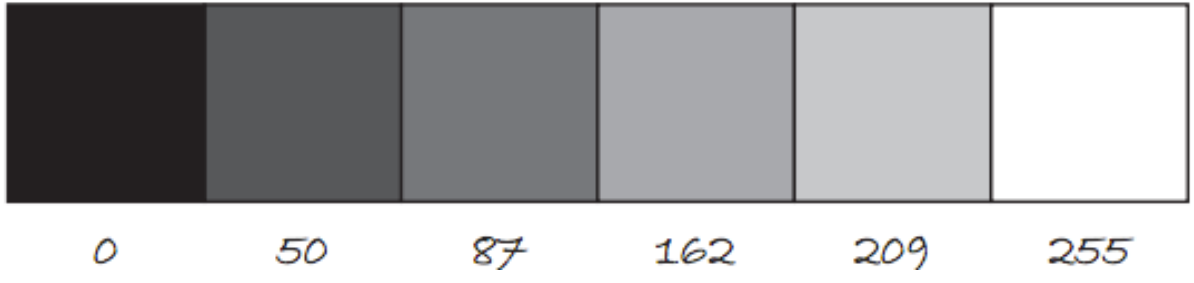
2) def on_draw(self): Called automatically 60 times a second to draw objects. Write code to draw all objects here.

3) def on_update(self): Called automatically 60 times a second to update our objects. Write code to update all objects here(for animation).

Color

Color is defined by a range of numbers.

In grayscale, 0 is black, 255 is white and any color in between is a shade of gray ranging from black to white.



Color

RGB Color is determined by three parameters. Each parameter is in the range 0-255. The first indicates the degree of red(R), the second the degree of green(G) and the last the degree of blue(B).

- Red + green = yellow
- Red + blue = purple
- Green + blue = cyan (blue-green)
- Red + green + blue = white
- No colors = black

Some Methods for Drawing Shapes

`fill(r, g, b)` : By calling fill BEFORE a shape will set the color of the shape. Call it again before drawing another shape to change color.

`line(x1, y1, x2, y2)` : draw line through (x1, y1) and (x2, y2).

`ellipse(x, y, width, height)` : center of ellipse is (x, y); width and height are the lengths of the axes.

`rect(x, y, width, height`: center of the rectangle is (x,y)

game.py

A method/function that is not implemented yet still need code in the body.
We use the pass statement to construct a body that does nothing
so that the interpreter does not throw an error.

```
class Window:
    def __init__(self):
        """ Declare/initialize all variables here."""
        pass
    def on_draw(self):
        """ Called automatically 60 times a second to draw all objects.
        # draw red circle at (20, 25) diameter = 300 pixels
        fill(255, 0, 0)
        ellipse(20, 25, 300, 300)
    def on_update(self):
        """ Called automatically 60 times a second to update all objects.
        pass
```

You must remove "pass" once you start implementing the code for the body of the method.

Animation

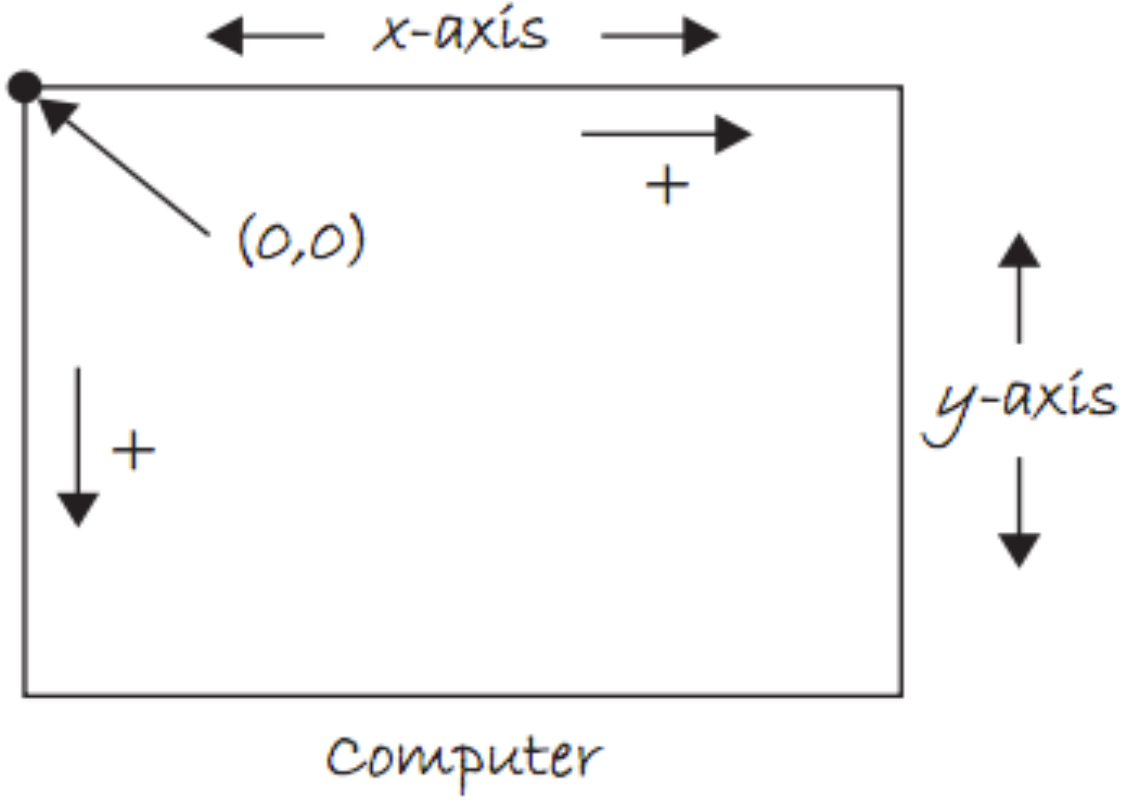
Animation only takes five lines of code!

When declaring/initializing a global variable that is used throughout the game, use self and the dot notation.

WIDTH and HEIGHT are width/height of the window.

```
class Window:
    def __init__(self):
        """ Declare/initialize all variables here."""
        self.x = WIDTH/2
        self.y = HEIGHT/2
    def on_draw(self):
        """ Called automatically 60 times a second to draw all objects.
        # draw red circle at (20, 25) diameter = 300 pixels
        fill(255, 0, 0)
        ellipse(self.x, self.y, 300, 300)
    def on_update(self):
        """ Called automatically 60 times a second to update all objects.
        self.x += 5
```

The Coordinate System



Class vs Objects

A **class** bundles together *data* (instance variables or attributes) and *functionality* (methods). Another name for class is **type**.

Everything in Python is a class. A list is a class. So is an integer, a string, a tuple, even functions!

The following creates two list **objects**.

```
a = [1, 2, 3]
b = [8, -5.3 "hi"]
print(type(a)) # list
```

Thus, in this example, list is a **class**(or **type**) and a and b are two of its **objects**.

Custom Classes

A **class** bundles together **data** (instance variables or attributes) and **functionality** (methods).

Example: A list has data(the elements of the list). It also has methods that manipulate those data(append, insert, pop, remove, etc…).

The classes int, bool, str, list, tuple, etc… are built-in classes.

Python provides the ability for programmers to design their own types or classes(**custom classes**).

Class

We like to be able to build our own classes to represent objects relevant to our game or application.

A sprite is an image(.png or .jpg) that represent a character or object in a game.

In arcade.py, I have written a simple custom class: the Sprite class. It allows us to easily draw, scale and animate sprites. We may create several Sprite **instances** or **objects**.

This **reusability** feature is important especially when we need to create many objects(for example enemies) with similar data and behaviors.

The Sprite Class

The Sprite class' constructor allows us to create a Sprite object. It has many parameters to help us initialize a Sprite object for our game.

Usually, we specify only the image filename and scaling and set the other attributes as needed.

player = arcade.Sprite("player.png", 0.5)

arcade.Sprite(filename, scale=1.0)
center_x center_y angle width height change_x change_y change_angle alpha
draw() update()

The Sprite Class

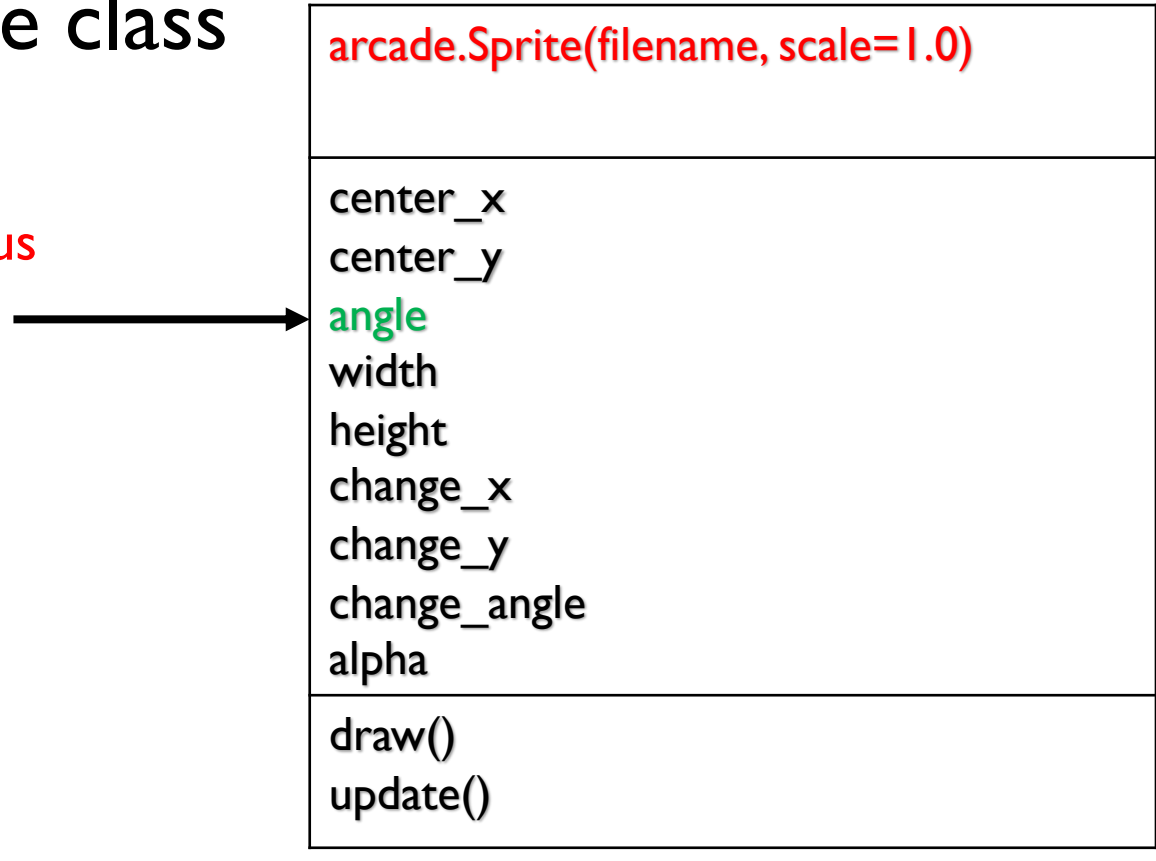
Default center is (0,0)

arcade.Sprite(filename, scale=1.0)
center_x center_y angle width height change_x change_y change_angle alpha
draw() update()

More on the Sprite class

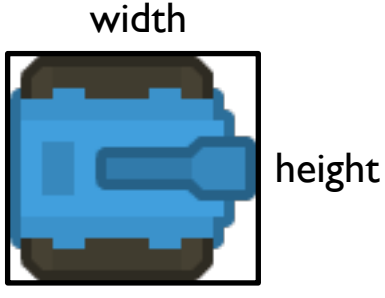
The angle attribute allows us to rotate the image of our Sprite.

This angle is measured in degrees with counterclockwise as the positive direction.



The Sprite Class

width and height are automatically initialized based on the image file and scale(default is 1.0)



arcade.Sprite(filename, scale=1.0)
center_x center_y angle width height change_x change_y change_angle alpha
draw() update()

The Sprite Class

For moving the sprite.
(velocity)

arcade.Sprite(filename, scale=1.0)
center_x center_y angle width height change_x change_y change_angle alpha
draw() update()

The Sprite Class

For rotating the sprite.

arcade.Sprite(filename, scale=1.0)
center_x center_y angle width height change_x change_y change_angle alpha
draw() update()

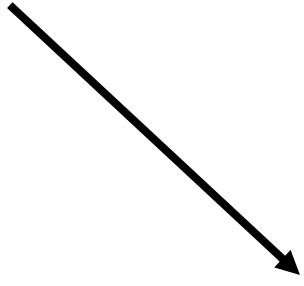
The Sprite Class

For transparency, 0 is fully transparent and 255 is fully opaque.
One use for transparency is "respawning" a sprite.

arcade.Sprite(filename, scale=1.0)
center_x center_y angle width height change_x change_y change_angle alpha
draw() update()

The Sprite Class

The method draw() will draw the image.



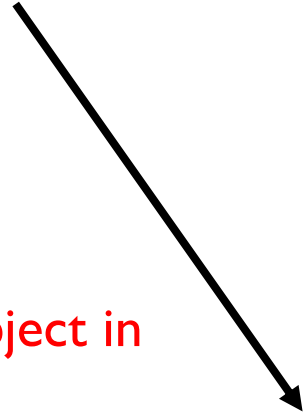
arcade.Sprite(filename, scale=1.0)
center_x center_y width height left right top bottom change_x change_y alpha
draw() update()

The Sprite Class

update() will
automatically animate the
sprite:

```
center_x += change_x
center_y += change_y
angle += change_angle
```

call update() on each object in
on_update()



arcade.Sprite(filename, scale=1.0)
center_x center_y width height left right top bottom change_x change_y alpha
draw() update()

Adding Text

The `text(str, x, y)` function draws text on the screen. You can set the text size and color by using `textSize(s)` and `fill(r, g, b)` before drawing the text.

```
textSize(32);
fill(255, 0, 0);
text("Hello, World!", 100, 200);
```

The Console

Messages can be printed on the console(for error-checking purposes, etc..) by using the command **`println()`**.

```
println(4);
println(4 + 3/2);
println("Hello, world");
```

on_key_press

The `on_update() and on_draw()` loops run continuously until they are interrupted by an event, for example, a keyboard or mouse event.

If a key is pressed, `on_update()` and `on_draw()` temporarily halt, Processing then jump execution to the on_key_press `()` function, runs the function's code then return control to the `on_update and on_draw` loops.

The key that is pressed is store in the `key` variable.

Similarly, if a key is released, `on_key_release()` is called.

Processing: Mouse Events

Processing keeps track of the position of the mouse at any given time through the variables `mouseX, mouseY.`

Similar to on_key_press, which responds to keyboard inputs, `on_mouse_press` is a function that can be implemented to respond to the mouse being pressed. Similarly for on_key_release.

`def on_mouse_press(self, x, y, button)`: x, y location of the mouse; button: LEFT, RIGHT, CENTER

`def on_mouse_release(self, x, y, button)`: x, y location of the mouse; button: LEFT, RIGHT, CENTER

mouseX, mouseY

mouseX and mouseY are variables that keep track of the position of the mouse.

What does the following simple program do?

```
class Window:
    def __init__(self):
            """ Declare/initialize all variables here."""
          pass
    def on_draw(self):
        """ Called automatically 60 times a second to draw all objects.
        # draw red circle at (20, 25) diameter = 300 pixels
          fill(255, 0, 0)
          ellipse(mouseX, mouseY, 100, 100)
    def on_update(self):
         pass
```

Download Processing

Download Processing!

Your computer is probably a 64-bit computer if it's fairly recent.

http://www.processing.org