Introduction to Processing

Working with Sprites

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**.

Example

Suppose we like to write a game that contains many characters moving about on a map. Each character has a position(x and y), a speed(speedx, speedy). Consider the following code:

```
x1 = 0
y1 = 0
speedx1 = 5
speedy1 = 3

x2 = 10
y2 = 10
speedx2 = 2
speedy2 = 4
```

If we have many characters in the game, this code makes it difficult to keep track of all the characters.

Even with 5 characters, we need to keep track of 20 variables.

This is just counting variables. What if we want each character to have behaviors(functions) such as jumping, shooting?

Class

We like to write code once(called a class) and be able to reuse it for all of the objects of that class. A **class** bundles together **data** (instance variables or attributes) and **behavior or functionality**(functions).

x1 = 0

y1 = 0

speedx1 = 5

speedy1 = 3

player1

Two variables player1 and player2 instead of eight variables.

x2 = 10

y2 = 10

speedx2 = 2

speedy2 = 4

player2

If our class is called Sprite, player1 and player2 are two objects of that class.

Class

We like to be able to build our own classes to represent objects relevant to our game or application. Python provides the ability for programmers to design their own types or classes(**custom classes**).

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

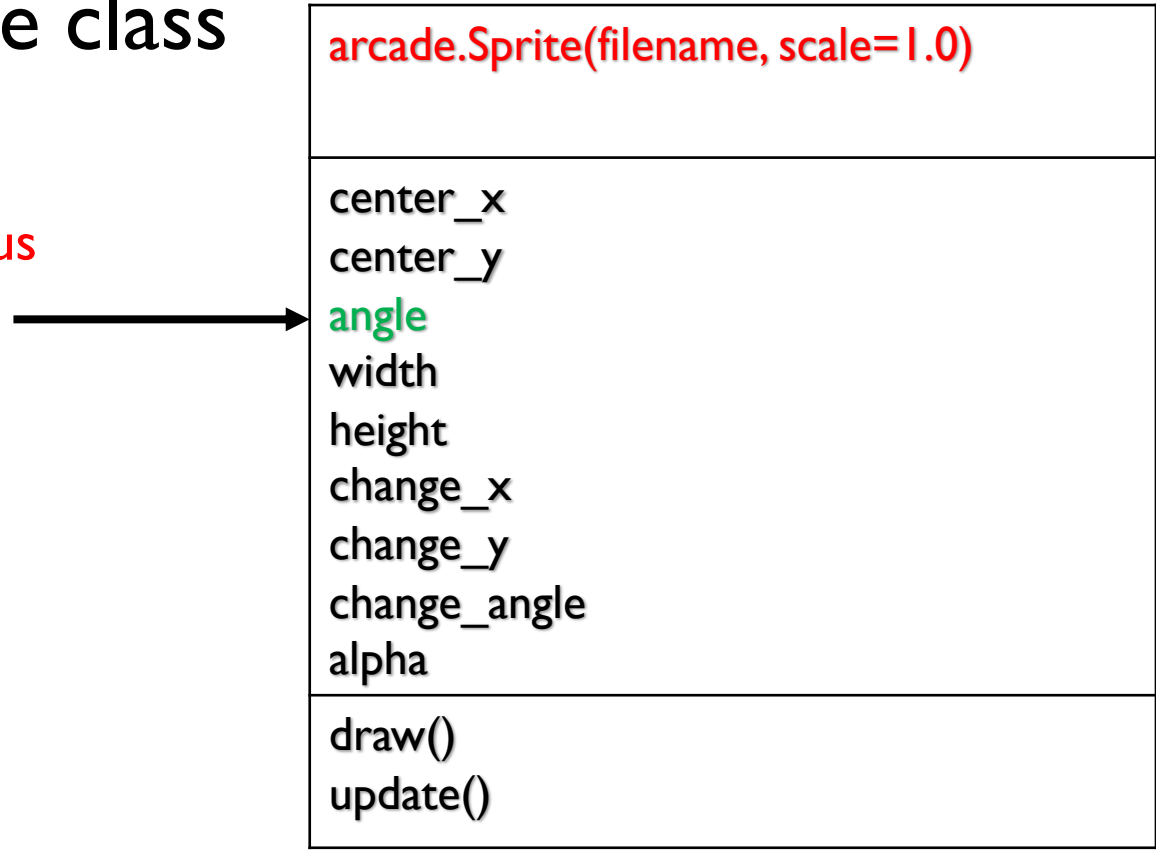
Default center is (0,0)

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

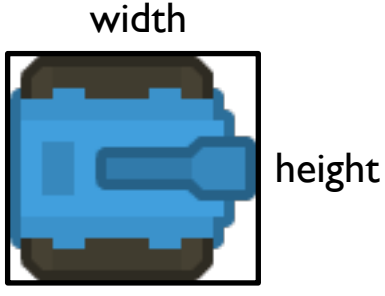
The angle attribute allows us to rotate the image of our Sprite.

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()

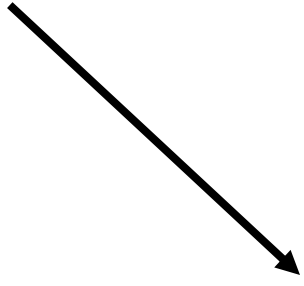
The Sprite Class

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.



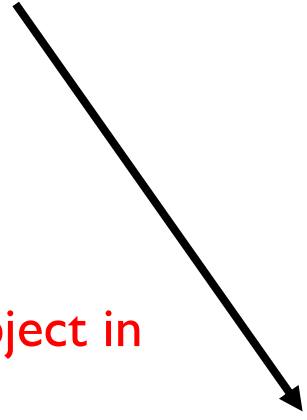
arcade.Sprite(filename, scale=1.0)
center_x center_y width height left right top bottom change_x change_y alpha
draw() update()

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()



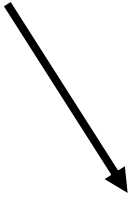
arcade.Sprite(filename, scale=1.0)
center_x center_y width height left right top bottom change_x change_y alpha
draw() update()

Sprite Example 1

```
class Window:
    def __init__(self):
        """ Initialize all variables here. """
        self.player = arcade.Sprite("tank.png")
        self.player.scale = 2.0
        self.player.center_x = 100
        self.player.center_y = 200

    def on_draw(self):
        """ Called automatically 60 times a second to draw all objects."""
        self.player.draw()

    def on_update(self):
        """ Called automatically 60 times a second to update all objects."""
```



These four lines are equivalent to:
`self.player = arcade.Sprite("tank.png", 2.0, 100, 200)`

Sprite Example 2: Moving the tank

```
class Window:
    def __init__(self):
        """ Initialize all variables here. """
        self.player = arcade.Sprite("tank.png", 2.0, 100, 200)
        self.player.change_x = 5
    def on_draw(self):
        """ Called automatically 60 times a second to draw all objects."""
        self.player.draw()

    def on_update(self):
        """ Called automatically 60 times a second to update all objects."""
        self.player.update()
```

Controlling a Character

To control a character on the screen using the keyboard, the trick is to always update a character's position by adding velocity to position in the on_update() method. Then, if a user presses a key, change the velocity component according to which key was pressed. If a key is released, reset the velocity in that direction to 0.

```
def on_key_press(self, key):
        if key == RIGHT:
            self.player.change_x = 5

def on_key_release(self, key):
        if key == RIGHT:
            self.player.change_x = 0
```

on_key_press

An important to note is that when a user presses two keys simultaneously, `on_key_press()` only detects the latest key. Thus, if we want to move a character right and up at the same time, `on_key_press()` alone is not sufficient.

Using `on_key_release()`, we can better control a character on the screen.

List of Sprite Objects Lab

Download the zip file that contains a starter's template code for processing on our course website here.

REMEMBER TO SAVE BEFORE RE-RUNNING YOUR CODE!!!

Do the following:

1) Declare, initialize a sprite object using the image "tank.png". Draw it on the screen in the on_draw method.
2) Create a list containing 10 "coin.png" sprite objects. Randomize their positions. Display them on the screen on the on_draw method.
3) Display the the number of coins on the screen by using the text() function. For example, "Coins: 10".

Control Sprite with Keyboard Lab

Modify the previous "List of Sprite Objects" lab to allow for controlling the tank with keyboard inputs.

Implement both `on_key_press` and `on_key_release` to respond to arrow keys: UP, DOWN, LEFT, RIGHT.

Each of the keys should move the tank in that direction. If two keys are pressed, for example, UP and RIGHT, the tank should move in the diagonal direction.