1 Tasks

1.1 Creating an Object - create.py

1. Why are classes useful if we already have functions? **There are many reasons classes are important, but in particular, classes allow for *object oriented programming*, where we can manipulate individual 'objects' and better organize our code. If we want to work with students, for example, having a class that holds all the student IDs and demographics can be helpful! It's almost like making a new data type.**

2. What is the purpose of the `__init__()` function? When is the function called? **The init function is called every time we call an instance of the class. It allows us to create objects of that type**

```python
#filename = create.py
class NewClass():
    def __init__(self, the_list):
        self.the_list = the_list

    def print_params(self):
        for val in self.the_list:
            print(val)

    def find_average(self):
        return sum(self.the_list)/len(self.the_list)

    def find_min(self):
        return min(self.the_list)

    def find_max(self):
        return max(self.the_list)

my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
new_object = NewClass(my_list)
print(new_object.the_list)
print('params test:')
new_object.print_params()
print('average test:', new_object.find_average())
print('min test:', new_object.find_min())
print('max test:', new_object.find_max())

print('##########################')
print('Test #2, Unsorted List')
print('##########################')
my_list_2 = [100, 122, 3, 4, -7, 3, 1]
new_object_2 = NewClass(my_list_2)
print('params test:')
new_object_2.print_params()
print('average test:', new_object_2.find_average())
print('min test:', new_object_2.find_min())
print('max test:', new_object_2.find_max())
```

1.2 Using an Object as an Attribute - pet.py

```
class Dog():
    def __init__(self, name, weight, breed):
        self.name=name
        self.weight=weight
        self.breed=breed

    def eat(self, amount):
        self.weight+=amount

    def exercise(self, amount):
        self.weight-=amount

class Person():
    def __init__(self, name, dog, generosity):
        self.name=name
        self.dog=dog
        self.generosity=generosity

    def feedDog(self):
        amt = self.generosity
        self.dog.eat(amt)
        return None

## Create a Dog object.
my_doggy = Dog('Albert', 12, 'Chihuahua')
## Create a Person object.
me = Person('Zoe', my_doggy, 0.7)
## Have your Person object feed their pet dog.
weight = my_doggy.weight
me.feedDog()
new_weight = my_doggy.weight

print('old weight was:', weight)
print('new weight is:', new_weight)
print('the difference is:', new_weight-weight, 'when we expected', me.generosity)
if abs(new_weight-weight-me.generosity) <= 0.001:
    print('good enough!')
else:
    print('check again!')
```

Checkoff #2— Which object – Person or Dog – is "within" the other object? If it were the other way around, would you be able to feed the dog by calling the `feedDog()` method on `my_person`? **We have a 'dog' object 'inside' the person object. If we instead had their human as a DOG attribute, we could feed the dog by saying something like dog.person.feedDog(), where we get the person of that dog, and then have that person feed them.**

1.3 Animating a wheel - animate.py

```
#here is the testing code I wrote!
win = GraphWin('Wheel', 320, 240)
w = Wheel(Point(100, 100), 50, 70)#This creates a new object w that is a Wheel.
w.draw(win)
```

```
w.set_color('blue', 'black')
w.animate(win, 1, 0, 100)

#to move faster: we can increase the x/y values,
#so each 'unit' of time it moves further
#we can ALSO edit the 'wait time' from 100ms

#for moving longer, we can increase the value of n
# or make each 'unit' of movement longer!
```

2 Optional Challenge[1]:

2.1 Drawing a car - car.py

```
class Car:
    def __init__(self, wheel1, wheel2, rect):
        self.wheel1 = wheel1
        self.wheel2 = wheel2
        self.rect = rect

    def draw(self, window):
        self.wheel1.draw(window)
        self.wheel2.draw(window)
        self.rect.draw(window)

    def set_color(self, tire_color, wheel_color, body_color):
        self.rect.setFill(body_color)
        self.wheel1.set_color(wheel_color, tire_color)
        self.wheel2.set_color(wheel_color, tire_color)

    def animate(self, window, dx, dy, n):
        if n > 0:
            self.wheel1.move(dx, dy)
            self.wheel2.move(dx, dy)
            self.rect.move(dx, dy)
            win.after(100, self.animate, win, dx, dy, n - 1)


wheel1 = Wheel(Point(50, 50), 10, 15)
wheel2 = Wheel(Point(100, 50), 10, 15)
rect = Rectangle(Point(50, 50), Point(100, 10)) #height = 50-1 = 40
win = GraphWin("Car", 300, 300)
car = Car(wheel1, wheel2, rect)
car.set_color('black', 'grey', 'pink')
car.animate(win, 1, 0, 400)
car.draw(win)
```

[1] optional problems are provided for those who have further interests and want to explore more. You are not responsible for those questions, however.