**Practice Quiz: Object-oriented Programming (Optional)**

**1.Let’s test your knowledge of using dot notation to access methods and attributes in an object. Let’s say we have a class called Birds. Birds has two attributes: color and number. Birds also has a method called count() that counts the number of birds (adds a value to number). Which of the following lines of code will correctly print the number of birds? Keep in mind, the number of birds is 0 until they are counted!**

bluejay.count()

print(bluejay.number)

**2.Creating new instances of class objects can be a great way to keep track of values using attributes associated with the object. The values of these attributes can be easily changed at the object level. The following code illustrates a famous quote by George Bernard Shaw, using objects to represent people. Fill in the blanks to make the code satisfy the behavior described in the quote.**

# “If you have an apple and I have an apple and we exchange these apples then

# you and I will still each have one apple. But if you have an idea and I have

# an idea and we exchange these ideas, then each of us will have two ideas.”

# George Bernard Shaw

class Person:

apples = 0

ideas = 0

johanna = Person()

johanna.apples = 1

johanna.ideas = 1

martin = Person()

martin.apples = 2

martin.ideas = 1

def exchange\_apples(you, me):

#Here, despite G.B. Shaw's quote, our characters have started with #different amounts of apples so we can better observe the results.

#We're going to have Martin and Johanna exchange ALL their apples with #one another.

#Hint: how would you switch values of variables,

#so that "you" and "me" will exchange ALL their apples with one another?

#Do you need a temporary variable to store one of the values?

#You may need more than one line of code to do that, which is OK.

you.apples, me.apples = me.apples, you.apples

return you.apples, me.apples

def exchange\_ideas(you, me):

#"you" and "me" will share our ideas with one another.

#What operations need to be performed, so that each object receives

#the shared number of ideas?

#Hint: how would you assign the total number of ideas to

#each idea attribute? Do you need a temporary variable to store

#the sum of ideas, or can you find another way?

#Use as many lines of code as you need here.

totalIdeas = me.ideas + you.ideas

you.ideas = totalIdeas

me.ideas = totalIdeas

return you.ideas, me.ideas

exchange\_apples(johanna, martin)

print("Johanna has {} apples and Martin has {} apples".format(johanna.apples, martin.apples))

exchange\_ideas(johanna, martin)

print("Johanna has {} ideas and Martin has {} ideas".format(johanna.ideas, martin.ideas))

**3.The City class has the following attributes: name, country (where the city is located), elevation (measured in meters), and population (approximate, according to recent statistics). Fill in the blanks of the max\_elevation\_city function to return the name of the city and its country (separated by a comma), when comparing the 3 defined instances for a specified minimal population. For example, calling the function for a minimum population of 1 million: max\_elevation\_city(1000000) should return "Sofia, Bulgaria".**

# define a basic city class

class City:

name = ""

country = ""

elevation = 0

population = 0

# create a new instance of the City class and

# define each attribute

city1 = City()

city1.name = "Cusco"

city1.country = "Peru"

city1.elevation = 3399

city1.population = 358052

# create a new instance of the City class and

# define each attribute

city2 = City()

city2.name = "Sofia"

city2.country = "Bulgaria"

city2.elevation = 2290

city2.population = 1241675

# create a new instance of the City class and

# define each attribute

city3 = City()

city3.name = "Seoul"

city3.country = "South Korea"

city3.elevation = 38

city3.population = 9733509

def max\_elevation\_city(min\_population):

# Initialize the variable that will hold

# the information of the city with

# the highest elevation

return\_city = City()

# Evaluate the 1st instance to meet the requirements:

# does city #1 have at least min\_population and

# is its elevation the highest evaluated so far?

if city1.population >= min\_population and city1.elevation > return\_city.elevation:

return\_city = city1

# Evaluate the 2nd instance to meet the requirements:

# does city #2 have at least min\_population and

# is its elevation the highest evaluated so far?

if city2.population >= min\_population and city2.elevation > return\_city.elevation:

return\_city = city2

# Evaluate the 3rd instance to meet the requirements:

# does city #3 have at least min\_population and

# is its elevation the highest evaluated so far?

if city3.population >= min\_population and city3.elevation > return\_city.elevation:

return\_city = city3

#Format the return string

if return\_city.name:

return "{}, {}".format(return\_city.name, return\_city.country)

else:

return ""

print(max\_elevation\_city(100000)) # Should print "Cusco, Peru"

print(max\_elevation\_city(1000000)) # Should print "Sofia, Bulgaria"

print(max\_elevation\_city(10000000)) # Should print ""

**4.What makes an object different from a class?**

An object is a specific instance of a class

**5.We have two pieces of furniture: a brown wood table and a red leather couch. Fill in the blanks following the creation of each Furniture class instance, so that the describe\_furniture function can format a sentence that describes these pieces as follows: "This piece of furniture is made of {color} {material}"**

class Furniture:

color = ""

material = ""

table = Furniture()

table.color = "brown"

table.material = "wood"

couch = Furniture()

couch.color = "red"

couch.material = "leather"

def describe\_furniture(piece):

return ("This piece of furniture is made of {} {}".format(piece.color, piece.material))

print(describe\_furniture(table))

# Should be "This piece of furniture is made of brown wood"

print(describe\_furniture(couch))

# Should be "This piece of furniture is made of red leather"