Congratulations! You passed!

TO PASS 80% or higher

Keep Learning

GRADE
80%

Practice Quiz: Object-oriented Programming (Optional)

TOTAL POINTS 5

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! **1 / 1 point**

- bluejay.number = 0

  print(bluejay.number)

- print(bluejay.number.count())
- bluejay.count()

  print(bluejay.number)

- print(bluejay.number)

✓ **Correct**

Nice job! We must first call the count() method, which will populate the number attribute, allowing us to print number and receive a correct response.

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. **0 / 1 point**

```
1  # "If you have an apple and I have an apple and we exchange these apples ther
2  # you and I will still each have one apple. But if you have an idea and I hav
3  # an idea and we exchange these ideas, then each of us will have two ideas."
4  # George Bernard Shaw
5
6  class Person:
7      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 = martin.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.
    you.ideas = martin.ideas
    me.ideas = johanna.ideas
    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))
```

Run

Reset

Incorrect

Not quite. Did you properly add or equivalate the attributes of both instances of the Person() class?

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". **1 / 1 point**

```
# 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"
```

Run

Reset

Correct

Way to go! You're getting comfortable with the idea of class objects and what they can do!

4. What makes an object different from a class? **1 / 1 point**

An object represents and defines a concept

- An object is a specific instance of a class
- An object is a template for a class
- Objects don't have accessible variables

Correct

Awesome! Objects are an encapsulation of variables and functions into a single entity.

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}"

```
1  class Furniture:
2    color = ""
3    material = ""
4
5  table = Furniture()
6  table.color = 'brown'
7  table.material = 'wood'
8
9  couch = Furniture()
10 couch.color = 'red'
11 couch.material = 'leather'
12
13 def describe_furniture(piece):
14   return ("This piece of furniture is made of {} {}".format(piece.color, pie
       .material))
15
16 print(describe_furniture(table))
17 # Should be "This piece of furniture is made of brown wood"
18 print(describe_furniture(couch))
19 # Should be "This piece of furniture is made of red leather"
```

Run

Reset

Correct

Right on! You're working well with classes, objects, and instances!