Congratulations! You passed!
TO PASS 80% or higher

Keep Learning

GRADE
100%

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.

1 / 1 point

```
11  johanna.apples = 1
12  johanna.ideas = 1
13
14  martin = Person()
15  martin.apples = 2
16  martin.ideas = 1
17
18  def exchange_apples(you, me):
19  #Here, despite G.B. Shaw's quote, our characters have started with
20  #different amounts of apples so we can better observe the results.
21  #We're going to have Martin and Johanna exchange ALL their apples with
22  #one another.
23  #Hint: how would you switch values of variables,
24  #so that "you" and "me" will exchange ALL their apples with one another?
25  #Do you need a temporary variable to store one of the values?
26  #You may need more than one line of code to do that, which is OK.
27          you.apples, me.apples = me.apples, you.apples
28          return you.apples, me.apples
29
30  def exchange_ideas(you, me):
31      #"you" and "me" will share our ideas with one another.
32      #What operations need to be performed, so that each object receives
33      #the shared number of ideas?
34      #Hint: how would you assign the total number of ideas to
35      #each idea attribute? Do you need a temporary variable to store
36      #the sum of ideas, or can you find another way?
37      #Use as many lines of code as you need here.
38      you.ideas += me.ideas
39      me.ideas = you.ideas
40      return you.ideas, me.ideas
41
42  exchange_apples(johanna, martin)
43  print("Johanna has {} apples and Martin has {} apples".format(johanna.apples
44  exchange_ideas(johanna, martin)
45  print("Johanna has {} ideas and Martin has {} ideas".format(johanna.ideas, m
46
47
48
49
```

Run
Reset

```
Johanna has 2 apples and Martin has 1 apples
Johanna has 2 ideas and Martin has 2 ideas
```

Correct
Awesome! You're getting used to using instances of class objects and assigning them attributes!

3. The City class has the following attributes: name, country (where the city is located), elevation (measured in

1 / 1 point

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".

```
# 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.e
        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.e
        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.e
        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 ""
```

Run

Reset

```
Cusco, Peru
Sofia, Bulgaria
```

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 / 1 point

```
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, p

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"
```

Run

Reset

```
This piece of furniture is made of brown wood
This piece of furniture is made of red leather
```

Correct

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