

✔ Congratulations! You passed!

Go to next item

Grade received **100%** To pass 80% or higher

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

```
1  # "If you have an apple and I have an apple and we exchange these apples then
2  # you and I will still each have one apple. But if you have an idea and I have
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
8      ideas = 0
9
10     johanna = Person()
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         temp = you.apples
20         you.apples = me.apples
21         me.apples = temp
22         return you.apples, me.apples
23
24     #Here, despite G.B. Shaw's quote, our characters have started with          #different amounts of apples so we can better observe the results.
25     #We're going to have Martin and Johanna exchange ALL their apples with #none another.
26     #Hint: how would you switch values of variables,
27     #so that "you" and "me" will exchange ALL their apples with one another?
28     #Do you need a temporary variable to store one of the values?
29     #You may need more than one line of code to do that, which is OK.
30
31     def exchange_ideas(you, me):
32         # "you" and "me" will share our ideas with one another.
33         #What operations need to be performed, so that each object receives
34         #the shared number of ideas?
35         #Hint: how would you assign the total number of ideas to
36         #each idea attribute? Do you need a temporary variable to store
37         #the sum of ideas, or can you find another way?
38         #Use as many lines of code as you need here.
39         temp = you.ideas
40         you.ideas += me.ideas
41         me.ideas += temp
```

Run

Reset

✔ 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 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

```
1  # define a basic city class
2  class City:
3      name = ""
4      country = ""
5      elevation = 0
6      population = 0
7
8  # create a new instance of the City class and
9  # define each attribute
10     city1 = City()
11     city1.name = "Cusco"
12     city1.country = "Peru"
```

```

13 city1.elevation = 3399
14 city1.population = 358052
15
16 # create a new instance of the City class and
17 # define each attribute
18 city2 = City()
19 city2.name = "Sofia"
20 city2.country = "Bulgaria"
21 city2.elevation = 2290
22 city2.population = 1241675
23
24 # create a new instance of the City class and
25 # define each attribute
26 city3 = City()
27 city3.name = "Seoul"
28 city3.country = "South Korea"
29 city3.elevation = 38
30 city3.population = 9733509
31
32 def max_elevation_city(min_population):
33     # Initialize the variable that will hold
34     # the information of the city with
35     # the highest elevation
36     return_city = City()
37
38     # Evaluate the 1st instance to meet the requirements:
39     # does city #1 have at least min_population and
40     # is its elevation the highest evaluated so far?

```

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 / 1 point

```

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, piece.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!