

C1M5L3_Code_Reuse

March 29, 2020

1 Code Reuse

Let's put what we learned about code reuse all together. First, let's look back at **inheritance**. Run the following cell that defines a generic `Animal` class.

```
In [ ]: class Animal:
        name = ""
        category = ""

        def __init__(self, name):
            self.name = name

        def set_category(self, category):
            self.category = category
```

What we have is not enough to do much – yet. That's where you come in. In the next cell, define a `Turtle` class that inherits from the `Animal` class. Then go ahead and set its category. For instance, a turtle is generally considered a reptile. Although modern cladistics call this categorization into question, for purposes of this exercise we will say turtles are reptiles!

```
In [ ]:
```

Run the following cell to check whether you correctly defined your `Turtle` class and set its category to reptile.

```
In [ ]: print(Turtle.category)
```

Was the output of the above cell reptile? If not, go back and edit your `Turtle` class making sure that it inherits from the `Animal` class and its category is properly set to reptile. Be sure to re-run that cell once you've finished your edits. Did you get it? If so, great!

Next, let's practice **composition** a little bit. This one will require a second type of `Animal` that is in the same category as the first. For example, since you already created a `Turtle` class, go ahead and create a `Snake` class. Don't forget that it also inherits from the `Animal` class and that its category should be set to reptile.

```
In [ ]:
```

Now, let's say we have a large variety of Animals (such as turtles and snakes) in a Zoo. Below we have the Zoo class. We're going to use it to organize our various Animals. Remember, inheritance says a Turtle is an Animal, but a Zoo is not an Animal and an Animal is not a Zoo – though they are related to one another.

Fill in the blanks of the Zoo class below so that you can use `zoo.add_animal()` to add instances of the Animal subclasses you created above. Once you've added them all, you should be able to use `zoo.total_of_category()` to tell you exactly how many individual Animal types the Zoo has for each category! Be sure to run the cell once you've finished your edits.

```
In [ ]: class Zoo:
        def __init__(self):
            self.current_animals = {}

        def add_animal(self, animal):
            self.current_animals[animal.name] = animal.category

        def total_of_category(self, category):
            result = 0
            for animal in self.___.values():
                if ___ == category:
                    result += 1
            return result

zoo = Zoo()
```

Run the following cell to check whether you properly filled in the blanks of your Zoo class.

```
In [ ]: turtle = Turtle("Turtle") #create an instance of the Turtle class
        snake = Snake("Snake") #create an instance of the Snake class

        zoo.add_animal(turtle)
        zoo.add_animal(snake)

        print(zoo.total_of_category("reptile")) #how many zoo animal types in the reptile cate.
```

Was the output of the above cell 2? If not, go back and edit the Zoo class making sure to fill in the blanks with the appropriate attributes. Be sure to re-run that cell once you've finished your edits. Did you get it? If so, perfect! You have successfully defined your Turtle and Snake subclasses as well as your Zoo class. Do you see how this could be useful in a wide variety of circumstances?

Okay, one last bit of practice before we move on. Let's take another quick peek at **modules**. There are lots of useful modules for Python. Many of them are in the standard library. However, some modules are not loaded by default. So, before we can use any of the built-in methods of a module that is not in the standard library, the module must first be imported. Let's review this by importing the calendar module.

Run the following cell that imports the calendar module.

```
In [ ]: import calendar
```

One built-in method of the `calendar` module is the **`weekday()`** method. The `weekday()` method returns the day of the week (0 is Monday) for a given year, month, and day passed into the function. The syntax is `weekday(year, month, day)` where the input parameters are all of number type. By importing the `calendar` module, the `weekday()` method becomes available for us to use. In the following cell, fill in the blanks of the `weekday()` method with date arguments to return the day of the week for that date.

```
In [ ]: calendar.weekday(____,____,____)
```

Did you get an integer representing the day of the week (0 is Monday) for the date arguments passed into the function? If not, go back and edit your input parameters. Once you output an integer representing the day of the week for the date arguments passed into `weekday()`, you are all done with this notebook. Great work!