Object Oriented Attributes With Functions - Lab

Introduction

We've been learning a lot about different parts of object oriented programming. We learned about classes and what purpose they serve. We've seen instance objects, instance variables, and instance methods and how these things all work with each other. In this lab, we will talk about what a **domain model** is and how it ties into object oriented programming.

Objectives

You will be able to:

- Understand the concept of a domain model
- Create a domain model
- Define instance methods that operate on nested data structures

What Is a Domain Model?

A domain model is the representation of a real-world concept or structure translated in to software. This is a key function of object orientation. So far, our Python classes have been used as blueprints or templates for instance objects of that class. As an example, a Driver class would create driver intsance objects, and the class would define a basic structure for what that driver instance object should look like and what capabilities it should have. But a class is only one part of a domain model just as, typically, a driver is only one part of a larger structure.

A domain model is meant to mirror that larger, real-world structure. It is more than just one class, it is an entire environment that often depends on other parts or classes to function properly. So, in keeping with a Driver class, we could use the example of a taxi and limousine service as our domain model. There are many more parts to a service like this than drivers alone. We could imagine dispatchers, mechanics, accountants, passengers, etc., all being part of the structure of this domain model. In a simplified example, we could have instance and class methods handle things like `dispatch_driver`, `calculate_revenue_from_rides`, `service_taxi`, or any other function of a taxi and limousine service.

As we become more fluent in object oriented programming and our programs become more complex, we will see that the other parts of a domain model like passengers, dispatchers, etc., will be classes of their own which interact with each other.

In this lab, we will be using a school as our domain model.

Part 1:

Create a class, School, in the school.py file in your local directory that can be initialized with a name. The School class would be referred to as a "model" in the domain model context.

> **Note:** you may want to load the autoreload extension from IPython
>
> ```
> %load_ext autoreload
> %autoreload 2
> ```

In [15]:
```
from school import School
```

In [28]:
```
school = School("Middletown High School")
```

Part 2:

A school should have a roster, which should be an empty dictionary upon initialization but will be built-out to contain keys of grade levels. The value of each key will be a list of student names (i.e. `{"9": ["John Smith", "Jane Donahue"]}`).

In [29]:
```
school.roster() #{}
```

Out[29]:
```
{}
```

Part 3:

You should be able to add a student to the school by calling the add_student method and giving it an argument of the student's name and their grade.

In [30]:
```
school.add_student("Peter Piper", 12)
school.roster() #{"12": ["Peter Piper"]}
```

Out[30]:
```
{12: ['Peter Piper']}
```

> **Hint:** if the dictionary starts out empty, how will we add keys which initially point to empty lists as their value? Let's look at an example below:

In [36]:
```
new_dict = {}
```

We start out with our empty dictionary and we want to add a student from the 10th grade, Timmy Turner, to our dictionary. The number `10` will be the key and it should point to an array containing the string `"Timmy Turner"`. Let's see if we can create a new key and add the name at the same time.

```
new_dict[10].append("Timmy Turner")
```

```
---------------------------------------------------------------------------
KeyError                         Traceback (most recent call last)
<ipython-input-37-e74b24b6fe3b> in <module>()
----> 1 new_dict[10].append("Timmy Turner")

KeyError: 10
```

Okay, so we see we get a KeyError because our dictionary doesn't yet have the key `10`, so, we can't just directly operate on it. So, to start we have to add the key and set an initial value for it.

In [21]:
```
new_dict[10] = []
new_dict[10].append("Timmy Turner")
new_dict
```

Out[21]:
```
{10: ['Timmy Turner']}
```

Awesome! So, we now know how to add a key and set its initial value when the key does not yet exist in our dictionary.

Remember, next time we add a student from grade `10` we do not want to reinitialize our list, we just want to add the name to the list that already exists.

In [22]:
```
# add Sofia Santana to grade 10
new_dict[10].append("Sofia Santana")
```

Now write the class function, `add_student` that takes in the grade and the name of the student and adds them to the school's roster and returns the new roster.

In [31]:
```
school.add_student("Kelly Slater", 9)
school.add_student("Tony Hawk", 10)
school.add_student("Ryan Sheckler", 10)
school.add_student("Bethany Hamilton", 11)
school.roster() # {9: ["Kelly Slater"], 10: ["Tony Hawk", "Ryan Sheckler"], 11: ["Bethany Hamilton"], 12: ["Peter Piper"]}
```

Out[31]:
```
{9: ['Kelly Slater'],
 10: ['Tony Hawk', 'Ryan Sheckler'],
 11: ['Bethany Hamilton'],
 12: ['Peter Piper']}
```

Part 4:

Next, define a method called `grade`, which should take in an argument of a grade and return a list of all the students in that grade:

In [33]:
```
print(school.grade(10)) # ["Tony Hawk", "Ryan Sheckler"]
print(school.grade(12)) # ["Peter Piper"]
```

```
['Tony Hawk', 'Ryan Sheckler', 'Tony Hawk']
['Peter Piper']
```

Part 5:

Define a method called `sort_roster` that returns the school's roster where the strings in the student arrays are sorted alphabetically. For instance: `{9: ["Kelly Slater"], 10: ["Ryan Sheckler", "Tony Hawk"], 11: ["Bethany Hamilton"], 12: ["Peter Piper"]}}`

> **Note:** since dictionaries are unordered, the order of the keys does not matter here, just the order of the student's names within each list.

In [34]:
```
school.sort_roster()
```

Out[34]:
```
{9: ['Kelly Slater'],
 10: ['Ryan Sheckler', 'Tony Hawk', 'Tony Hawk'],
 11: ['Bethany Hamilton'],
 12: ['Peter Piper']}
```

Summary

In this lab, we were able to mimic a complex domain model using a School class with a few instance methods and variables. Soon we will see that our domain models will use other classes, instance methods, and instance variables to create more functionality in our programs.