index

March 30, 2022

1 Instance Variables - Lab

1.1 Introduction

In this lab, you'll practice using instance variables, which you use to store information about a particular instance object. You will continue to use our `fuber` theme and create some methods that operate on our instance variables to return some valuable information about the passenger and driver instance objects.

1.2 Objectives

In this lab you will:

- Define and call an instance method
- Define and access instance attributes

1.3 Define classes

Below, define classes for both the `Driver` and `Passenger` classes – for now just define the classes and remember to include the keyword `pass` so that you'll have valid syntax for the classes.

[1]:
```
# Driver class
class Driver:
    pass
```

[2]:
```
# Passenger class
class Passenger:
    pass
```

Next, instantiate a new instance of a passenger and a new instance of a driver. Give the passenger a `rating` of `4.9` and give the driver a `miles_driven` attribute of `100,000`.

[3]:
```
# Assign a driver instance
driver = Driver()

# Give the driver instance object 'miles_driven' of 100000
driver.miles_driven = 100000
```

[4]:
```
# Assign a passenger instance
passenger = Passenger()
```

```
# Give the passenger instance object a 'rating' of 4.9
passenger.rating = 4.9
```

1.4 Search attributes using functions

Your next challenge is to build a function to find a driver with a given name. The function should take two inputs, `drivers` and `search_names`. `drivers` will be a list of driver objects (instances of the class you defined above) and `search_name` will be a string for the driver name you wish to search from.

The function should then return the first driver object from drivers whose name is an exact match to the search name. If there is no driver that matches the name searched for, then the function should return `None` and print a string stating "Sorry we couldn't find a driver with the name, _____! :(".

For example, if there were no results for the search name "Jack" your function should return `None` and print:

```
"Sorry, we couldn't find a driver with the name, Jack! :("
```

[5]:
```
def find_driver_by_name(drivers, name):
    # Write your code here
    for item in drivers:
        if item.name == name:
            return item
            break
    else:
        print(f"Sorry, we couldn't find a driver with the name, {name}! :(")
        return None
```

To test your function, here are some arbitrary definitions to create instances of your `Driver` class. Run the cell below to load them into memory:

[6]:
```
# Create drivers with relevant attributes
alex_driver = Driver()
alex_driver.name = "alex"
alex_driver.rating = 9.0

michelle_driver = Driver()
michelle_driver.name = "michelle"
michelle_driver.rating = 8.0

jake_driver = Driver()
jake_driver.name = "jake"
jake_driver.rating = 9.7

ashleigh_driver = Driver()
ashleigh_driver.name = "ashleigh"
```

```
ashleigh_driver.rating = 8.75

list_of_drivers = [alex_driver, michelle_driver, jake_driver, ashleigh_driver]
```

Use `find_driver_by_name()` along with `list_of_drivers` to check if the following drivers exist:

- `"jake"`
- `"michelle"`
- `"allison"`

[7]:
```
# Find "jake"
output_1 = find_driver_by_name(list_of_drivers, name = "jake")
output_1
```

[7]:
```
<__main__.Driver at 0x7fcb5833be20>
```

[8]:
```
# Find "michelle"
output_2 = find_driver_by_name(list_of_drivers, name = "michelle")
output_2
```

[8]:
```
<__main__.Driver at 0x7fcb57e98370>
```

[9]:
```
# Find "allison"
output_3 = find_driver_by_name(list_of_drivers, name = "allison")
output_3
print(type(output_3))
```

```
Sorry, we couldn't find a driver with the name, allison! :(
<class 'NoneType'>
```

If you correctly defined `find_driver_by_name()`, the first two calls should have returned `Driver` objects, while the third should have printed the apology statement and returned `None`. (You can further inspect the final output to verify this using the `type()` function which should reveal that the output is indeed a `Nonetype`).

While perhaps moderately useful, the function as written is rather brittle. Misspelling a driver's name will lead to no results. As such, write a more general function called `name_starts_with()` that will return a list of instance objects that start with a given substring.

[12]:
```
# Write your function here that returns the list of
# instance objects whose name starts with the given substring
def name_starts_with(drivers, substring):
    l = []
    for item in drivers:
        if item.name[0] == substring:
            l.append(item)
    return l
```

Use `name_starts_with()` and `list_of_drivers` to find all drivers whose name start with `'a'`:

[13]:
```
# Drivers whose name start with 'a'
name_starts_with(list_of_drivers, "a")
```

[13]:
```
[<__main__.Driver at 0x7fcb5833b520>, <__main__.Driver at 0x7fcb57e98280>]
```

Finally, define a function that returns the instance object of the driver with the highest rating:

[14]:
```
# Write your function here that returns the driver with the highest rating
def highest_rated_driver(drivers):

    highest_score = drivers[0].rating
    best_driver = drivers[0]
    for item in drivers:
        if item.rating > highest_score:
            best_driver = item
            highest_score = item.rating
    return  best_driver
```

Use this function to find the highest rated driver:

[15]:
```
# Find the driver with the highest rating
print(highest_rated_driver(list_of_drivers))
```

```
<__main__.Driver object at 0x7fcb5833be20>
```

1.5 Want more?

Define a `NewDriver` class with an instance method called `passenger_names()`. This method accesses the `passengers` attribute of the class and returns a list of all `names` associated with `passengers`.

[16]:
```
# Define the NewDriver class
class NewDriver:
    def passenger_names(self):

        names = []

        for item in self.passengers:
            names.append(item.name)

        return names
```

Before we proceed, run the following cell that creates four passenger objects and a `list_of_passengers`:

[17]:
```
# Passengers
alex_passenger = Passenger()
alex_passenger.name = "alex"
```

```
michelle_passenger = Passenger()
michelle_passenger.name = "michelle"

jake_passenger = Passenger()
jake_passenger.name = "jake"

ashleigh_passenger = Passenger()
ashleigh_passenger.name = "ashleigh"

list_of_passengers = [alex_passenger, michelle_passenger, jake_passenger,␣
↪ashleigh_passenger]
```

Now, instantiate a `NewDriver` class called `best_driver` that has the attributes `name`, `car_make`, `car_model`, `age`, and `passengers`:

[18]:
```
# Instantiate a NewDriver class object
best_driver = NewDriver()


# Add the name attribute and assign it 'Garol'
best_driver.name = "Garol"

# Add the car_make attribute and assign it 'toyota'
best_driver.car_make = "toyota"

# Add the car_model attribute and assign it 'camry'
best_driver.car_model = "camry"

# Add the age attribute and assign it 30
best_driver.age = 30

# Add the passengers attribute and assign it to list_of_passengers
best_driver.passengers = list_of_passengers
```

Alright, great! Now you have some attributes on the driver that you can work with. Create an instance method in the NewDriver class called `passenger_names` which returns a list of all the passengers' names/ Your output should look like `['alex', 'michelle', 'jake', 'ashleigh']`.

Well done! You have all the necessary information for your `best_driver`. In the previous cell, you assigned some passengers to this driver. What are their names?

[19]:
```
# Find the names of the passengers
names_of_passengers = best_driver.passenger_names()
print(names_of_passengers)
```

```
['alex', 'michelle', 'jake', 'ashleigh']
```

1.6 Summary

In this lab, you practiced creating instance variables that add information to our instance objects. You then used these instance methods to return information about the instances themselves.