Kyle Odland
June 8, 2020
Foundations of Programming: Python
Assignment 08
https://github.com/oddlandd/IntroToProg-Python-Mod8

Using Classes

Introduction

The eighth assignment for Foundations of Programming: Python was to examine pseudo code in the Assignment08-Starter python script, determine what the program is meant to accomplish, and then fill in the classes and methods needed to add the desired functionality.

Starting the Code

The starter code given in the assignment provided a structure for how the program should operate. We were given three classes - Product, FileProcessor, and IO. From the pseudo-code, the Product class was meant to have two properties - product_name and product_price. As discussed in Module 8, this indicated to me that the Product class was meant to create object instances, with different products and their prices.

The FileProcessor class had descriptions of two methods, to save_data_to_file and to read_data_from_file. The docstring for the IO class was missing, but the pseudo-code indicated that there would be methods to show the user a menu, get the user's choice, show the user the current product data that was saved in a list, and get new product data from the user. The pseudo-code in the main body of the script gives the same indication about how the program should work - present the user a menu of options, then allow them to see the current product data, add new product data, or save the product data to a file.

The structure of this program, as laid out in the pseudo-code was very similar to Assignment 06, where the user was presented a menu of options, and was able to perform different actions on a to-do list. The main difference is that instead of using a dictionary to keep track of to-do items, this code needed to use the Product class. This was where I chose to start. If I could successfully create an object instance using my Product class, the rest of the code should fall into place.

Coding

I knew that I needed to have two properties in my Product class, the name of the product and the price. This was similar to the labs from Module 8. Each property needed a getter and a

setter. The getter tells the class what to return when the product_name or product_price is called. For product_name, I had the getter return the name that was entered in title case, and for product_price I had the getter return the price as a float. I didn't put limits into the setter, just use the value given when the property_price is set. I also added the __init__ constructor, to allow the initial value of product_name and product_price to be set when the object instance is created. Listing 1 shows the Product class from my code with docstring removed for brevity, with the constructor and properties as described.

```
class Product:
    # ----Constructor---
    def __init__(self, prod="", price=0.0):
        # ----Attributes----
        self.__product_name = prod
        self.__product_price = price
    # ----Properties-----
    # getter for product name, changes to title case
    @property
    def product_name(self):
        return str(self.__product_name).title()
    # setter for product name
    @product_name.setter
    def product_name(self, value: str):
        self.__product_name = value
    # getter for product price, changes to float
    @property
    def product_price(self):
        return float(self.__product_price)
    # setter for product price
    @product_price.setter
    def product_price(self, value: float):
        self.__product_price = value
```

Listing 1: Product class

After testing to ensure that my Product class was able to create object instances as desired, I moved on to the next part of the program, the FileProcessor class. This class is very close to what was done in Assignment 06, so I used that code as a starting point. There is no need to create object instances for this class, so both save_data_to_file and read_data_from_file were static methods. When I first tested getting Product objects, printing them was giving the name of the class and the location in memory, instead of the actual information in the object that I wanted. This meant that I couldn't just write a string of the product object to the file. I needed to pull the product name and product price data out of each object, and write that. I did this with a for loop, that pulls the product name and product price from each row in the list of product objects. Listing 2 shows the save_data_to_file method.

```
@staticmethod
def save_data_to_file(file_name, list_of_product_objects):
  f = open(file_name, "w")
  # write product & price of each object to file
  for row in list_of_product_objects:
    f.write(row.product_name + "," + str(row.product_price) + "\n")
  f.close()
```

Listing 2: save_data_to_file method in FileProcessor class

Reading data from a file was done in reverse of this, by splitting up the rows of strings in the file into a product and a price, then using those to create a Product object instance, and add each object instance to a list, which gets returned by the method.

```
@staticmethod
def read_data_from_file(file_name):
  list_of_product_objects = []
  f = open(file_name, "r")
  # get the data from each line of file, use it to initialize object
  for line in f:
    prod, price = line.split(",")
    obj = Product(prod, price)
    list_of_product_objects.append(obj)  # reads a row of data
  f.close()
  return list_of_product_objects
```

Listing 3: read_data_from_file method in FileProcessor class

For the IO class, most of the methods were very similar to what was used in Assignment 06, so I copied that code in and modified as needed. The print_menu_tasks method just displays the three choices that the user can make in the program - Show Existing List of Products, Add New Product to List, and Save to File and Exit Program. The input_menu_choice method just asks the user to input their choice from 1 to 3, then returns whatever they choose. I added a new method called print_products_in_list, which does basically the same thing as write_data_to_file, but with the print function instead. Listing 4 shows the code for the print_products_in_list method.

```
@staticmethod
def print_products_in_list(list_of_product_objects):
  print("******* The current Products are: *******")
  for row in list_of_product_objects:
    print(row.product_name + " ($%.2f)" % row.product_price)
  print("*****************************************")
```

Listing 4: print_products_in_list method in IO class

The final method that I needed to create was to allow the user to create a new object instance, with a new product name and price. Input_new_product just asks the user for a product name

and a price, creates a new object with that information and returns it. Listing 5 shows the input_new_product method.

```
@staticmethod
def input_new_product():
    """ Asks user for a new task and priority to add

    :return: (object) of Product
    """
    prod = input("Enter a new product: ")
    price = float(input("Enter the price of this product: "))
    obj = Product(prod, price)
    return obj
```

Listing 5: input_new_product method in IO class

Now that I had completed my classes, I needed to construct the main body of the script, to use the classes for the desired program functionality. The program begins by loading data from a file into a list of product objects, with my FileProcessor.read_data_from_file method. I added error handling to create the file if there is a FileNotFoundError.

I structured the main body of the script as a while loop. Each time through the loop, the program shows the menu of options to the user, and asks them to choose an option. When the user makes a choice, I have the method raise an exception if they choose a number outside of the range. This prints an error message to choose a number from 1 to 3, then continues onto the next loop of the program and prompts the user again.

If the user chooses 1, to print the current data in the list, the IO.print_products_in_list method gets called. If the user chooses 2, to add a new product, the IO.input_new_product method gets called, and the product object it returns is appended to the list of product objects. There is error handling for a ValueError, if the user enters in a non-numeric value for the price, which prints an error message and continues to the next loop of the program. If the user chooses 3, to save and exit, the FileProcessor.write_data_to_file gets called and the code breaks out of the while loop and ends the program. Listing 6 shows the while loop in the main body of the code.

```
while True:
  # Show user a menu of options
  IO.print_menu_tasks()

  # Get user's menu option choice
  # if user enters a number not in the menu, ask for new choice
  try:
    strChoice = IO.input_menu_choice()  # get menu option
  except:
    print("Please make a choice from 1 to 3")
    continue
  # Show user current data in the list of product objects
  if strChoice.strip() == '1':  # print the list of products
    IO.print_products_in_list(lstOfProductObjects)  # print the list of products\
  elif strChoice == '2':  # add a new product
    # if user enters non-numeric value for price, ask for new input
    try:
      lstOfProductObjects.append(IO.input_new_product())
    except ValueError:
      print("Price should be a number")
      continue
  # let user save current data to file and exit program
  elif strChoice == '3':  # Save and exit program
    FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)  # write to file
    print("Data Saved")
    break  # exit
```

Listing 6: Main body of Assignment 08 script

Running the Code

Figure 1 shows an example of the Assignment 08 code running in PyCharm, where the user adds a new product object to the list.

```
        Menu of Options
        1) Show Existing List of Products
        2) Add New Product to List
        3) Save to File and Exit Program

Which option would you like to perform? [1 to 3] - 2

Enter a new product: Pen
Enter the price of this product: .25
```

Figure 1: Adding product object to the list in PyCharm

```
        Menu of Options
        1) Show Existing List of Products
        2) Add New Product to List
        3) Save to File and Exit Program

Which option would you like to perform? [1 to 3] - 1

******* The current Products are: *******
Pencil ($0.45)
Stapler ($2.50)
Notebook ($1.00)
Pen ($0.25)
*****************************************
```

Figure 2: Displaying list of products in PyCharm

Running the code as above gives the output txt file, shown in Figure 3.

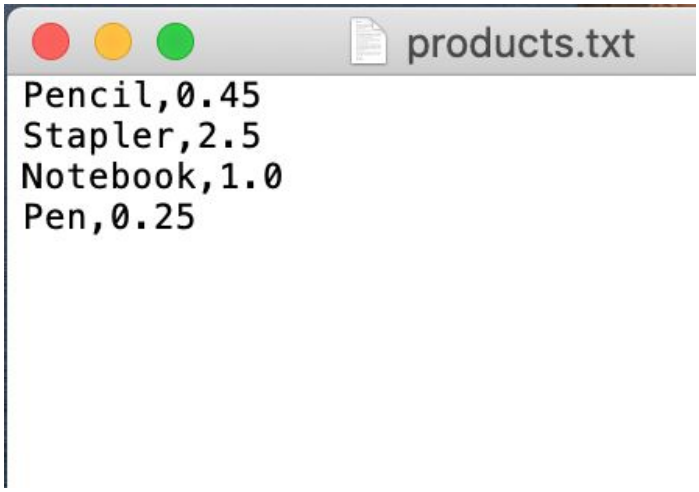

Figure 3: Output text file with a task list

Figure 4 shows an example of the error handling, while running the code in the macOS terminal.

Figure 4: Output text file with a task list

Summary

This assignment was very similar to the program from Assignment06, with modifications to use object instances from the Product class. It was a good example of how information can be stored in objects, rather than in dictionaries or lists of strings.