

Tony Le

11/26/2019

IT FDN 100

Assignment 08

<https://github.com/TonyLe2/IntroToProg-Python-Mod08>

# Objects and classes

## Introduction

The assignment for this week is to create a list of products and their corresponding prices and be able to read and write it to a file. This assignment builds off of the program in Assignment 06 that displays a menu to the user that allows them to create a To-Do list. We modify the program to fit our new criteria and utilizes objects and classes in order to accomplish the task. We were given a starter file to start with.

## Declare Variables and Constants

The script in Figure 1 begins by declaring the variables and constants to be used at a later point.

```
strFileName = "products.txt" # The name of the data file
lstOfProductObjects = [] # A dictionary that acts as a 'table' of rows

# Declare variables and constants
objFile = None # An object that represents a file
dicRow = {} # A row of data separated into elements of a dictionary {Product, Price}
strChoice = "" # Capture the user option selection
strProduct = ""
strPrice = ""
```

Figure 1: Declaring variables

## Declaring a class

Figure 2 establishes the **product** class which contains the name and price of the products we will be adding to a list. This is the main component to our code and where we will be storing and getting the data from.

```

class Product:
    """Stores data about a product:

    properties:
        product_name: (string) with the products's name
        product_price: (float) with the products's standard price
    methods:
        changelog: (When,Who,What)
            RRoot,1.1.2030,Created Class
            Tony Le, 11.26.2019, Modified code to complete assignment 8
    """

    pass

    # TODO: Add Code to the Product class

    # -- Constructor --
    def __init__(self, product_name, product_price):
        #-- Attributes --
        self.product_name = product_name
        self.product_price = product_price

    # -- Properties --
    # product_name

    @property # DON'T USE NAME for this directive
    def product_name(self): # (getting or accessor)
        return str(self.__product_name).title() # Title case

    @product_name.setter # The NAME MUST MATCH the property's!
    def product_name(self, value): # (setter or mutator)
        if str(value).isnumeric() == False:
            self.__product_name = value
        else:
            raise Exception("Names cannot be numbers")

    # Last_name

    @property # DON'T USE NAME for this directive
    def product_price(self): # (getting or accessor)
        return str(self.__product_price).title() # Title case

    @product_price.setter # The NAME MUST MATCH the property's!
    def product_price(self, value): # (setter or mutator)
        self.__product_price = value

    def __str__(self):
        return self.product_name + ',' + self.product_price

```

Figure 2: Declaring a class

## Processing functions within FileProcessor class

Within the **FileProcessor** class, we now have functions that act as the processors for this code. When called upon later in the script, these functions are executed and then return back to where they were originally called from. The first function we establish in Figure 3 is the **read\_data\_from\_file( )** function. This function reads the text file **products.txt** and reads the data within the file and loads it into a python dictionary **dicRow**. We begin by opening the file and reading the data into the memory. We follow by iterating through each entry in the txt file and extract with [0] and [1] entries to be inserted into a dictionary with keys **Product** and **Price**. This dictionary row is then appended onto our list table. The next function **save\_data\_to\_file( )** writes the data logged into the list into a text file. It uses a **for loop** to record every dictionary row for each iteration in the list. The last function follows after a user inputs new data in the table. It takes the data, adds it to **dicRow** and **appends** it as a row to the list.

```
# Processing ----- #
class FileProcessor:
    """Processes data to and from a file and a list of product objects..."""
    pass
    # TODO: Add Code to process data from a file
    @staticmethod
    def read_data_from_file(file_name, list_of_rows):
        """Desc - Reads data from a file into a list of dictionary rows..."""
        file = open(file_name, "r")
        for line in file:
            data = line.split(",")
            row = {"Product": data[0].strip(), "Price": data[1].strip()}
            list_of_rows.append(row)
        file.close()
        return list_of_rows

    # TODO: Add Code to process data to a file
    @staticmethod
    def save_data_to_file(file_name, list_of_rows):
        """Desc - Writes data from a file into a list of dictionary rows..."""
        pass # TODO: Add code Here

        objFile = open(file_name, "w")
        for dicRow in list_of_rows: # Write each row of data to the file
            objFile.write(dicRow["Product"] + "," + dicRow["Price"] + "\n")
        objFile.close()

    @staticmethod
    def AddRowToList(product, price, list_of_rows):
        """Desc - Reads data from a file into a list of dictionary rows..."""
        dicRow = {"Product": product, "Price": price} # Create a new dictionary row
        list_of_rows.append(dicRow) # Add the new row to the list/table
```

Figure 3: Fileprocessor( ) functions

## Input/output with Class IO

The first function to include in the input/output class is to create a menu that allows to user to control what the program does as shown in Figure 4. This function created is **PrintMenuItems()**. We create this to always call back to it after the user completes an action. The next function, **InputMenuChoice()** registers an input from the user to determine which action they'd like to perform from the menu. It returns the choice variable which is used later on in the script to determine which if statement to execute. The next function **ShowCurrentItemsInList()** shows the user what is currently in the list. This function is used when the user would like to reference what they've already inputted into the list. It does this with the use of a **for** loop that iterates and prints out every row in the list table. The last function of the IO class **userProductInput()** allows the user to enter in the tasks and priorities that they would like to add to the list and we assigned them to **objP1** to be used later. We make use of the **global** command here as the **strProduct** and **strPrice**, and **objP1** variables are needed for use outside of the function script.

```
class IO:
    # TODO: Add docstring
    """ A class for performing Input and Output """
    pass
    # TODO: Add code to show menu to user
    @staticmethod
    def PrintMenuItems():
        """
        ..:: ..
        """
        print('
Menu of Options
1) Show current data
2) Add a new product.
3) Save Data to File
4) Exit Program
')
        print() # Add an extra line for looks

    # TODO: Add code to get user's choice
    @staticmethod
    def InputMenuChoice():
        """
        ..:: ..
        """
        choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
        print() # Add an extra line for looks
        return choice

    # TODO: Add code to show the current data from the file to user
    @staticmethod
    def ShowCurrentItemsInList(list_of_rows):
        """Shows the current items in the list of dictionaries rows..."""
        print("***** The current products and their corresponding prices are: *****")
        for row in list_of_rows:
            print(row["Product"] + " (" + row["Price"] + ")")
        print("*****")
        print() # Add an extra line for looks

    # TODO: Add code to get product data from user
    @staticmethod
    def userProductInput():
        """
        ..:: ..
        """
        global strProduct
        global strPrice
        global objP1
        strProduct = str(input("What is the product? - ")).strip() # Get product from user
        strPrice = str(input("What is the price? - ")).strip() # Get price from user
        objP1 = Product(strProduct, strPrice)
        print() # Add an extra line for looks
```

Figure 4: Menu script

## Main body of the script

Now we get into the main body of the script in Figure 5 where we use all of the classes and functions that were established prior to this section of the script. The first function that is called on is **FileProcessor.read\_data\_from\_file( )**. This loads data into the table that is already existing in the text file. The next function is the display the menu with **IO.PrintMenuItems( )**. The next step is to take in user input for the action to be executed and assign it to the variable **strChoice**. We use the **strChoice** variable to execute the following **if** statement. This **if** statement executes depending on the input of the user. If the user inputs “1” then we show the current items in the list table with function **IO.ShowCurrentItemsInList( )**. With an input of “2” the user inputs a task and priority and those are inserted into the list as a dictionary row. The inputs are registered under **IO.userTaskInput( )** and processed under **FileProcessor.AddRowToList( )**. After each of these functions, the current list table is displayed. If the input is “3,” then it will prompt to user to determine whether they would like to save the code or not. If they do, the **FileProcessor.WriteDataToListFile( )** function is called upon and the list is saved to the products.txt file otherwise the data is not saved through the use of an **else** command. Finally, an input of “6” breaks the code and exits from there.

```
# Step 1 - When the program starts, Load data from products.txt.
FileProcessor.read_data_from_file(strFileName, lstOfProductObjects) # read file data

# Step 2 - Display a menu of choices to the user
while True:
    IO.PrintMenuItems() # Shows menu
    strChoice = IO.InputMenuChoice() # Get menu option

    # Step 3 - Process user's menu choice
    # Step 3.1 Show current data
    if (strChoice.strip() == '1'):
        IO.ShowCurrentItemsInList(lstOfProductObjects) # Show current data in the List/table
        continue # to show the menu

    # Step 3.2 - Add a new item to the List/Table
    elif strChoice.strip() == '2':

        # Step 3.2.a - Ask user for new product and price
        IO.userProductInput()
        IO.ShowCurrentItemsInList(lstOfProductObjects) # Show current data in the List/table

        # Step 3.2.b Add item to the List/Table
        FileProcessor.AddRowToList(objP1.product_name, objP1.product_price, lstOfProductObjects)
        IO.ShowCurrentItemsInList(lstOfProductObjects) # Show current data in the list/table
        continue # to show the menu

    # Step 3.4 - Save products to the products.txt file
    elif strChoice == '3':

        #Step 3.4.a - Show the current items in the table
        IO.ShowCurrentItemsInList(lstOfProductObjects) # Show current data in the List/table

        #Step 3.4.b - Ask if user if they want save that data
        if "y" == str(input("Save this data to file? (y/n) - ")).strip().lower(): # Double-check with user

            FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)

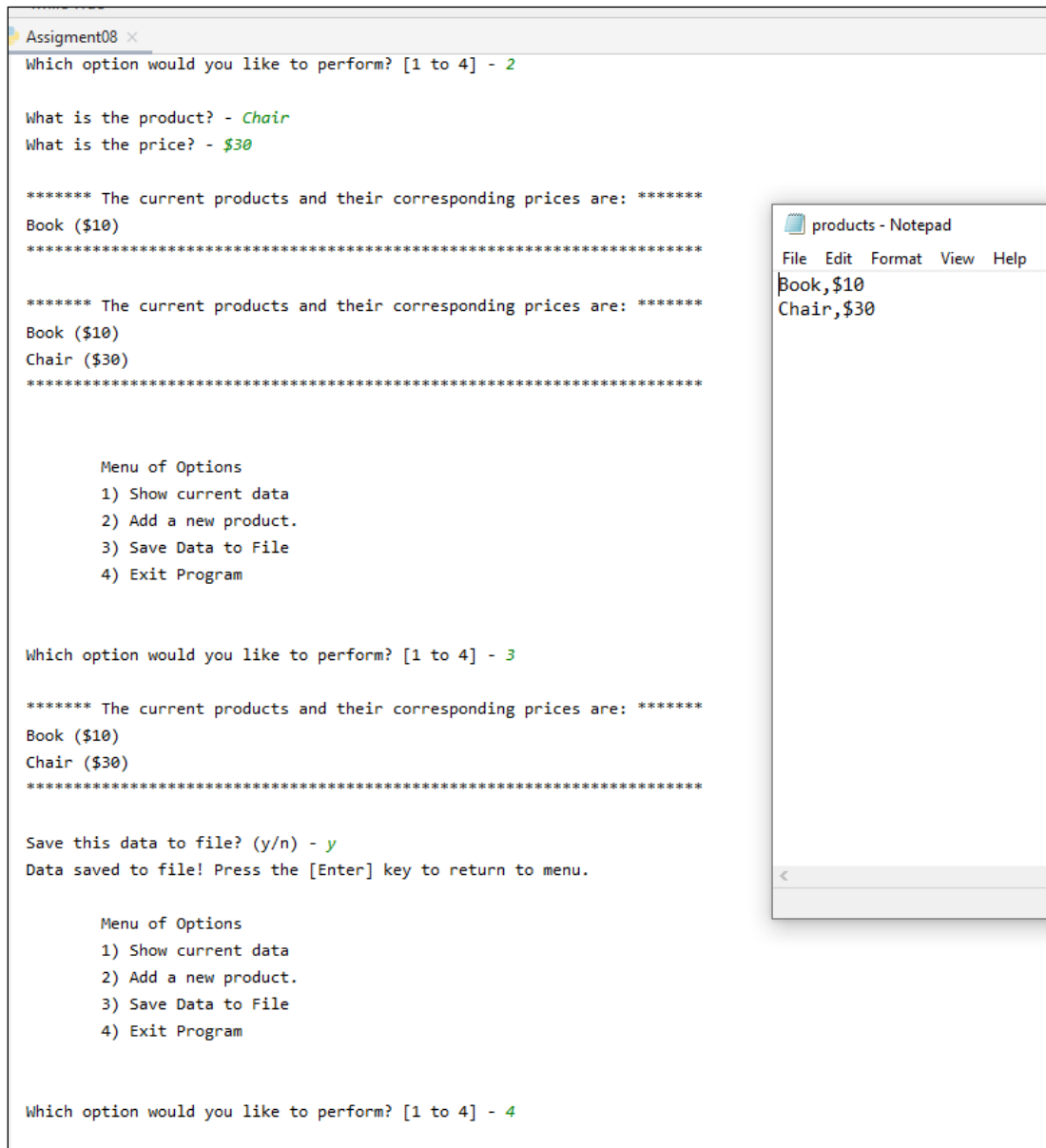
            input("Data saved to file! Press the [Enter] key to return to menu.")
        else: # Let the user know the data was not saved
            input("New data was NOT Saved, but previous data still exists! Press the [Enter] key to return to menu.")
        continue # to show the menu

    # Step 3.6 - Exit the program
    elif strChoice == '4':
        break # and Exit
```

Figure 5: Main body

## Summary

Figure 6 shows the output of the completed script and clearly shows that the data has been saved into the products.txt file. The intent of this script was to create an interactive menu and allow the user to create a product list and their respective prices. This assignment utilizes new skills we've learned in Module 8 and tests our understanding of objects and classes.



The image shows a terminal window titled "Assignment08" and a Notepad window titled "products - Notepad".

**Terminal Window Output:**

```
Which option would you like to perform? [1 to 4] - 2

What is the product? - Chair
What is the price? - $30

***** The current products and their corresponding prices are: *****
Book ($10)
*****

***** The current products and their corresponding prices are: *****
Book ($10)
Chair ($30)
*****

Menu of Options
1) Show current data
2) Add a new product.
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4] - 3

***** The current products and their corresponding prices are: *****
Book ($10)
Chair ($30)
*****

Save this data to file? (y/n) - y
Data saved to file! Press the [Enter] key to return to menu.

Menu of Options
1) Show current data
2) Add a new product.
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4] - 4
```

**Notepad Window Content:**

```
File Edit Format View Help
Book,$10
Chair,$30
```

Figure 6: Print out of full name entered by the user