Alex Jehle

August 30th, 2020

IT FDN 110 A

Assignment 08

GitHub: https://github.com/anjehle/IntroToProg-Python-Mod08

Product Catalogue Script with Classes and Functions

# Introduction

The purpose of this program is to update a product catalogue based off of a menu selection and input from the user.

# Program

The program Assigment08-starter.py (Figure 1) consists of three classes as well as a main script body. It begins by opening and reading or creating the catalogue text file, ‘products.txt. The program then enters into a while loop for the menu, prompting the user for one of three choices: displaying the products, adding a new item, or saving and exiting. Each time data is processed, the program returns to the main script and printing the menu of options. The user then inputs their choice and the program enters into the relevant processing or IO functions based on if statements.

# ------------------------------------------------------------------------ #  
# Title: Assignment 08  
# Description: Working with classes  
  
# ChangeLog (Who,When,What):  
# RRoot,1.1.2030,Created started script  
# RRoot,1.1.2030,Added pseudo-code to start assignment 8  
# AJehle,8.24.2020, Modified code to complete assignment 8  
# ------------------------------------------------------------------------ #  
  
# Data -------------------------------------------------------------------- #  
strFileName = 'products.txt'  
  
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  
 AJehle,8.24.2020,Modified code to complete assignment 8  
 """* def \_\_init\_\_(self, product\_name, product\_price):  
 self.product\_name = product\_name  
 self.product\_price = product\_price  
  
# Data -------------------------------------------------------------------- #  
  
# Processing ------------------------------------------------------------- #  
class FileProcessor:  
 *"""Processes data to and from a file and a list of product objects:  
  
 methods:  
 save\_data\_to\_file(file\_name, list\_of\_product\_objects):  
  
 read\_data\_from\_file(file\_name): -> (a list of product objects)  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 AJehle,8.25.2020,Modified code to complete Assignment08  
 """* def \_\_init\_\_(self, list\_of\_product\_objects):  
 self.list\_of\_product\_objects = list\_of\_product\_objects  
  
 @staticmethod  
 def save\_data\_to\_file(file\_name, list\_of\_product\_objects):  
 objFile = open(file\_name, 'w')  
 for prods in list\_of\_product\_objects:  
 objFile.write(prods.product\_name + ',' + prods.product\_price + '\n')  
 objFile.close()  
  
 @staticmethod  
 def read\_data\_from\_file(file\_name):  
 lstTable = []  
 try:  
 objFile = open(file\_name, 'r')  
 for row in objFile:  
 strData = row.split(",")  
 objP = Product(product\_name=strData[0].strip(),  
 product\_price=strData[1].strip())  
 lstTable.append(objP)  
 objFile.close()  
 print(f'Opened list from: {file\_name}')  
 return lstTable

except FileNotFoundError:  
 print('No existing file. New catalogue created.')  
 return lstTable

except Exception as e:  
 print(f'List not imported. Received error: {e}')  
 return lstTable  
  
# Processing ------------------------------------------------------------- #  
  
# Presentation (Input/Output) -------------------------------------------- #  
class IO:  
 *"""Processes data in a list of product objects:  
  
 methods:  
 print\_menu():  
  
 input\_choice(): -> (menu choice)  
   
 print\_current\_products():  
   
 add\_product(): -> (product name and price)  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 AJehle,8.25.2020,Modified code to complete Assignment08  
 """* pass  
  
 @staticmethod  
 def print\_menu():  
 *""" Display a menu of choices to the user* ***:return****: nothing  
 """* print('''  
 Menu of Options  
 1) Display current products  
 2) Add new product  
 3) Save and Exit   
 ''')  
 print() # Add an extra line for looks  
  
 @staticmethod  
 def input\_choice():  
 *""" Gets the menu choice from a user* ***:return****: string  
 """* choice = str(input("Which menu option would you like to perform? [1 to 3] - ")).strip()  
 print() # Add an extra line for looks  
 return choice  
  
 @staticmethod  
 def print\_current\_products(list\_of\_rows):  
 *""" Shows the current products in the list* ***:param*** *list\_of\_product\_rows: (list) of products and prices you want to display* ***:return****: nothing  
 """* print("\*\*\*\*\*\*\* The current products are: \*\*\*\*\*\*\*")  
 for prods in list\_of\_rows:  
 print(prods.product\_name + ", $" + prods.product\_price)  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print() # Add an extra line for looks  
  
 @staticmethod  
 def add\_product(list\_of\_rows: list):  
 product\_name = str(input("Which product would you like to add? ")).strip()  
 product\_price = str(input('What price does ' + product\_name + ' have?: $')).strip()  
 objP = Product(product\_name=product\_name,  
 product\_price=product\_price)  
 try:  
 list\_of\_rows.append(objP)  
 print(product\_name + " added!")  
 except:  
 print(product\_name + " was not successfully added")  
 return list\_of\_rows  
  
 pass  
# Main Body of Script ---------------------------------------------------- #  
# Load data from file into a list of product objects when script starts  
lstOfProductObjects = FileProcessor.read\_data\_from\_file(strFileName)  
  
while True:  
# Show user a menu of options  
 IO.print\_menu()  
  
# Get user's menu option choice  
 strChoice = IO.input\_choice()  
 if strChoice.strip() == '1':  
 # Show user current data in the list of product objects  
 IO.print\_current\_products(lstOfProductObjects)  
  
 elif strChoice.strip() == '2':  
 # Let user add data to the list of product objects  
 lstOfProductObjects = IO.add\_product(lstOfProductObjects)  
  
 elif strChoice.strip() == '3':  
 # let user save current data to file and exit program  
 try:  
 FileProcessor.save\_data\_to\_file(strFileName, lstOfProductObjects)  
 print("Save successful!")  
 break  
 except:  
 print("List not saved successfully")  
 else:  
 print("Please select valid option.")

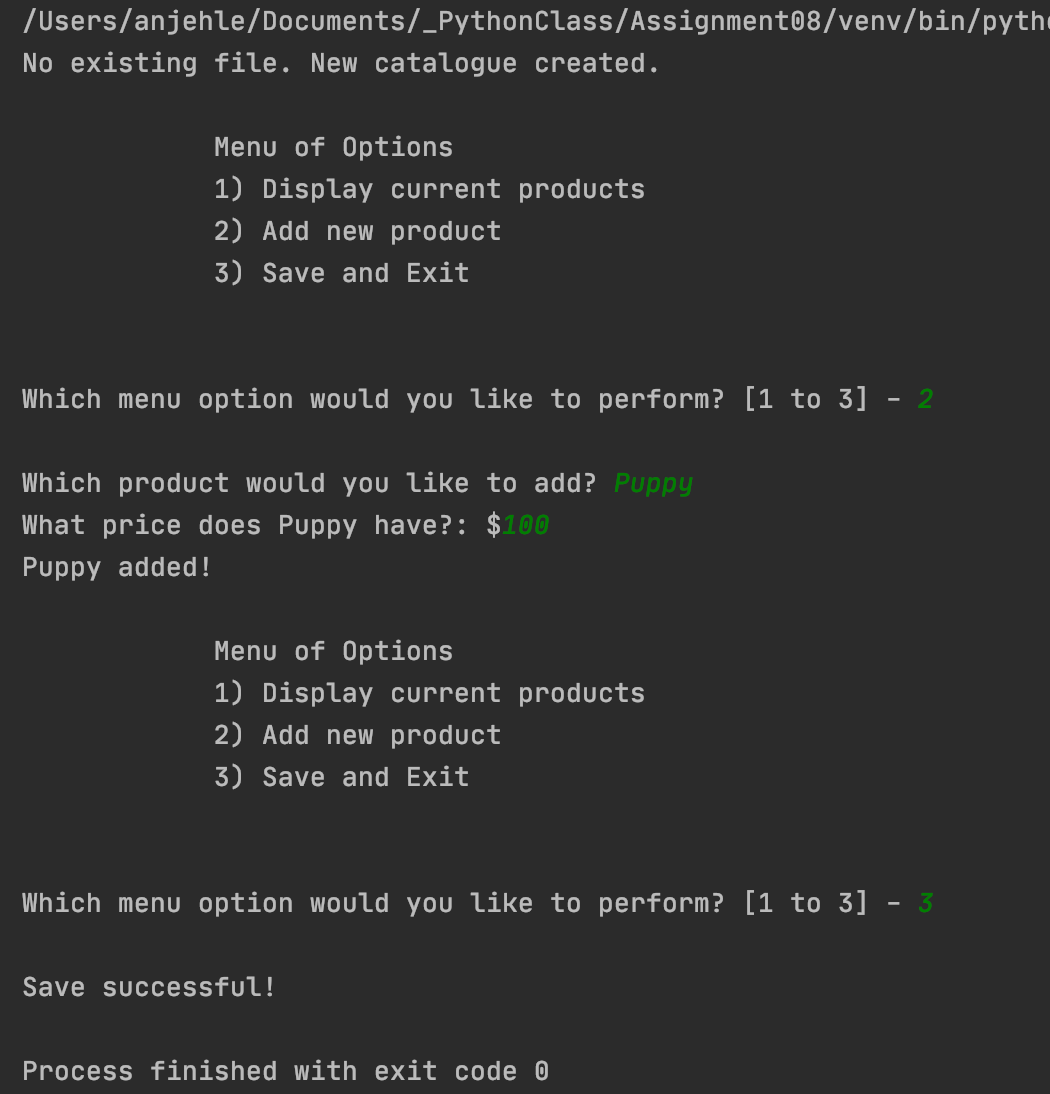
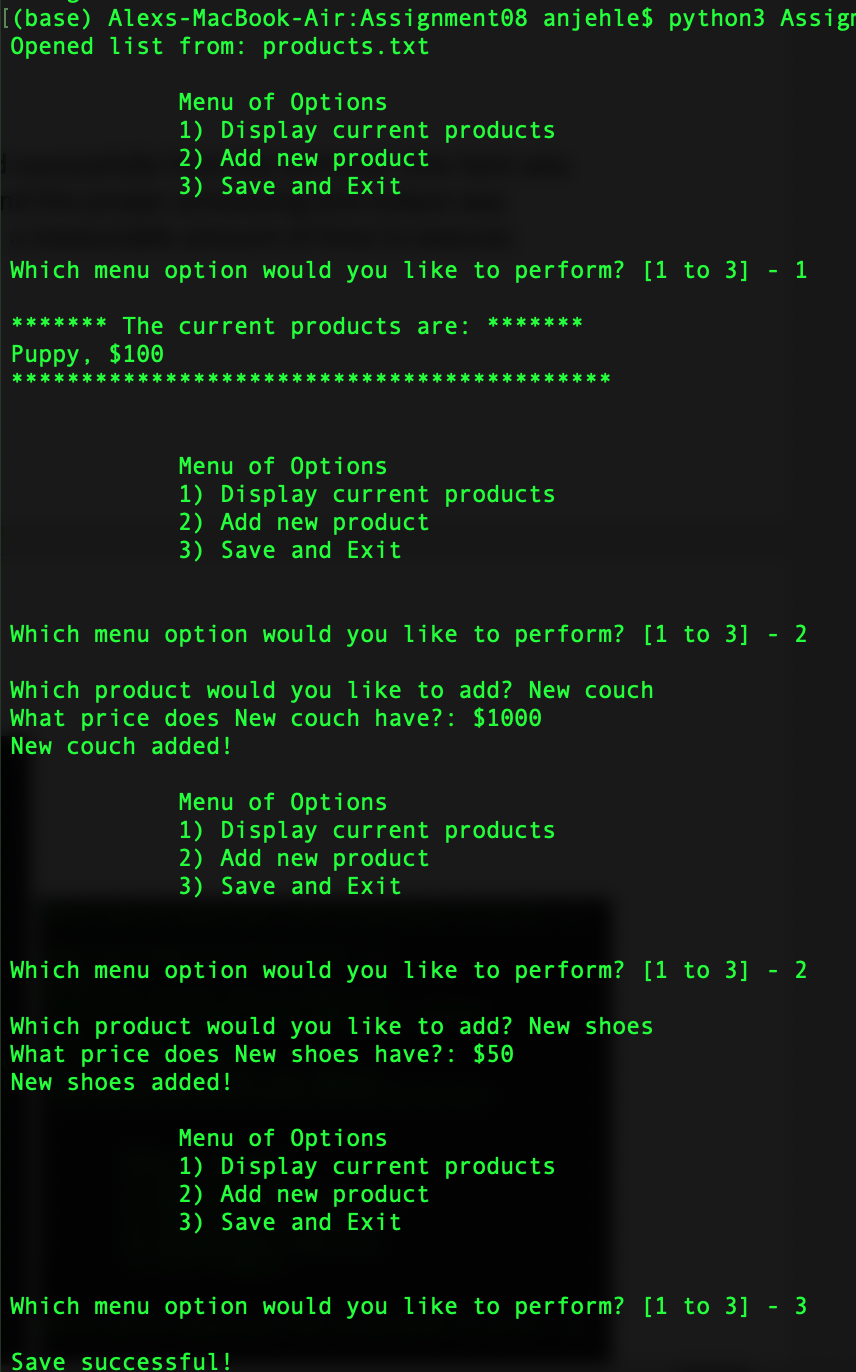
***Figure 1: ‘Assigment08-starter.py’ Script***

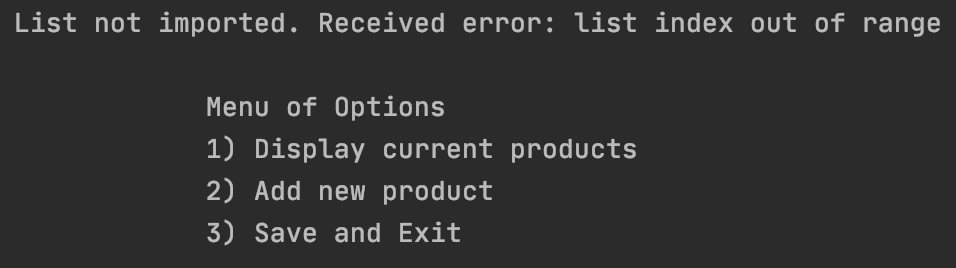
## Classes

There are three main classes in the script. The first is the Product class, which initializes the product type. Each product has a product name and price stored. The next two classes process the data. The FileProcessor class reads in and saves data based on the strFileName variable. The IO class is based on the menu options. It takes in the users input, prints the menu and catalogue, and adds a product when that option is selected. All of these classes are used exclusively from the main script which passes the file name and list of products between the various functions.

## Reading in Data

The program reads in data from the text file ‘products.txt’. Each line is loaded in as a Product type, with the product\_name and product\_price stored in the class as strings. The program checks whether the file exists first with a try/except testing for a FileNotFoundError. If the file does exist, if prints “Opened list from \_\_\_” with the name of the file. If it does not exist it prints ‘No existing file. New catalogue created.’ If another error is encountered there’s a final except which prints out a notification that the import failed and the error type. These are seen below in Figure 2.

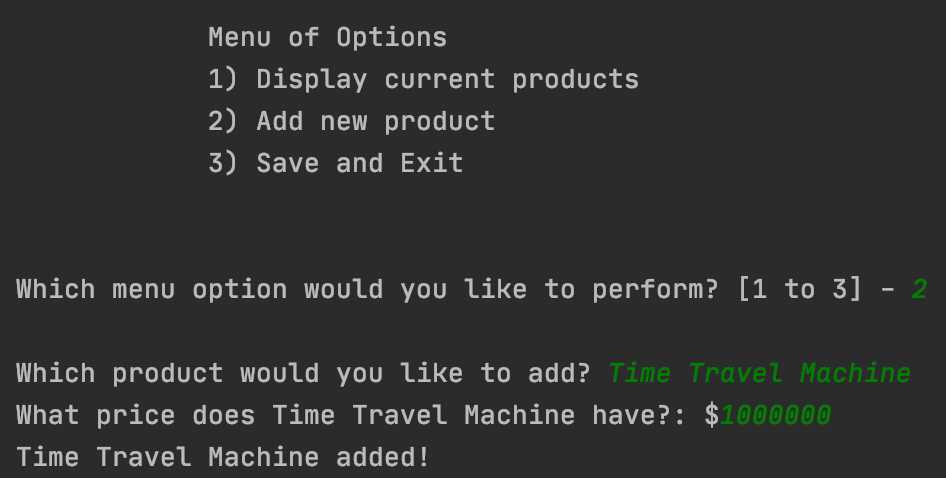
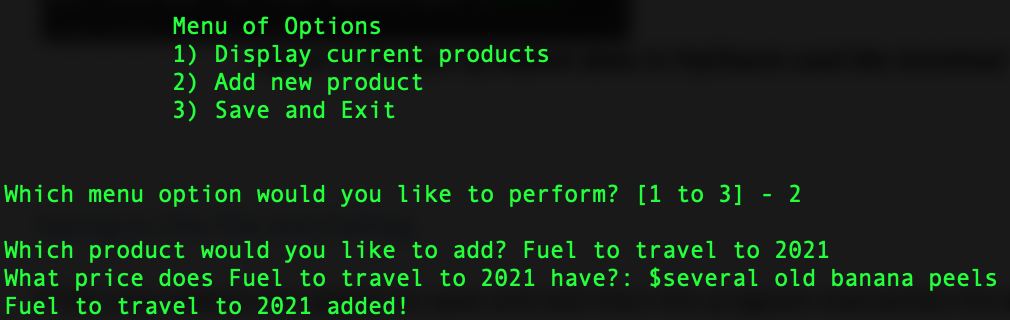
 



***Figure 2: Output of the read\_data\_from\_file function in PyCharm and the terminal***

## Adding Data

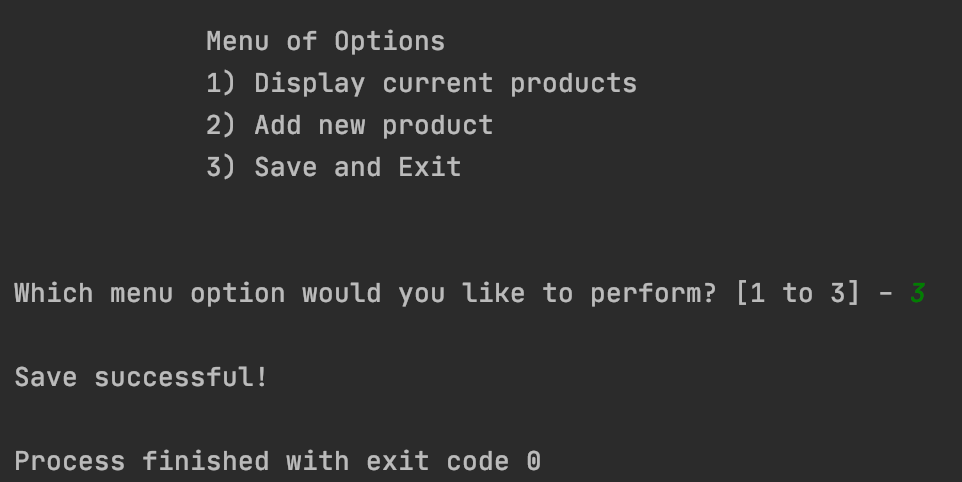
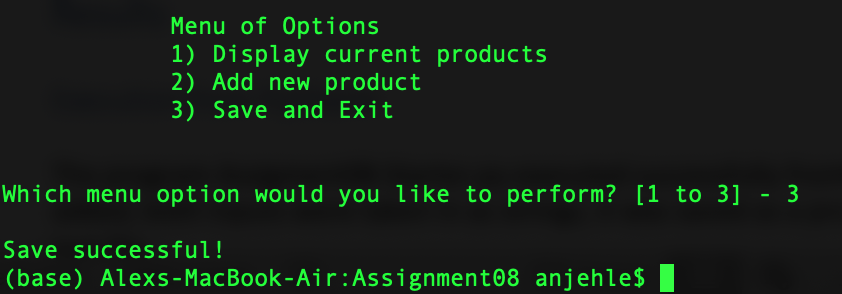
If the user selects the option to add a new item, the program prompts the user to enter the product they wish to add, and then prompts for the price. The program appends these items to the catalogue as products. If this is successful the program informs the user that the product was added and then returns to the initial menu option. This is shown in Figure 3.

***Figure 3: Output of added data in PyCharm and the terminal***

## Saving to the File and Exiting

If the user selects the option to save and exit then the program overwrites the current ‘products.txt’ with a copy of the product list read in at the beginning of the program and any changes applied. The program displays a confirmation message and exits the program. This process is shown in Figure 4.

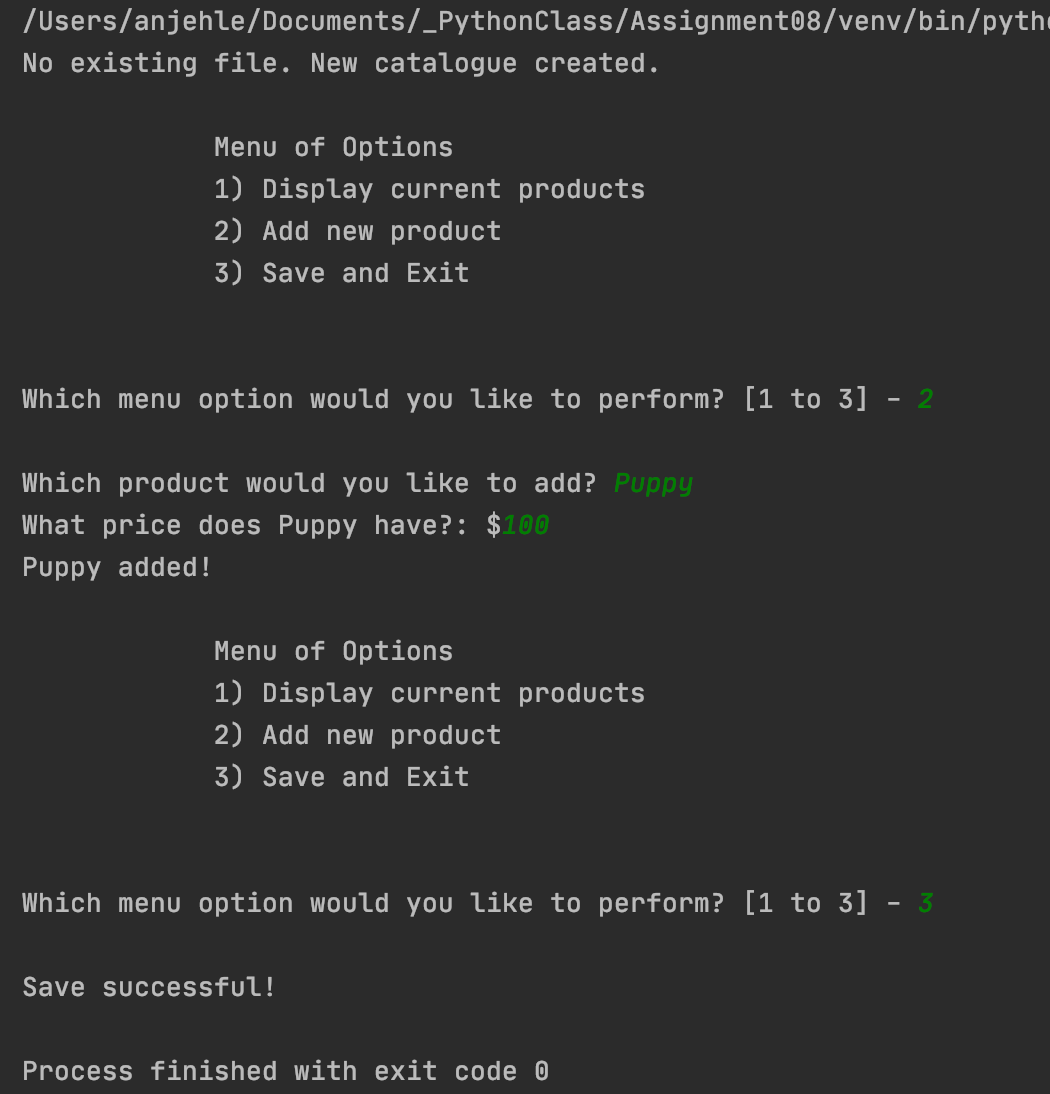
 

***Figure 4: Output after the data is saved in PyCharm and the terminal***

# Results

## Execution from PyCharm

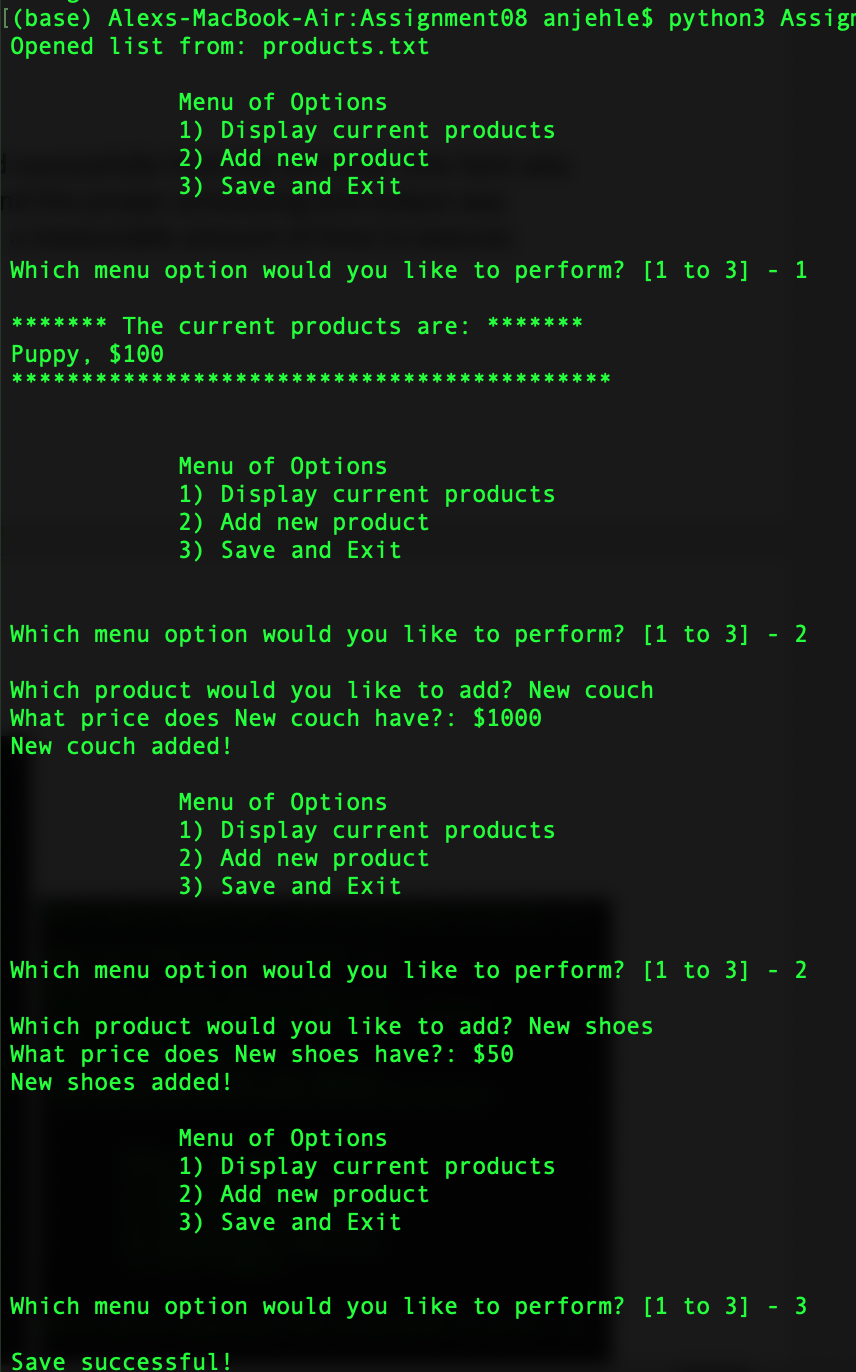
The program Assigment08-Starter.py executed successfully from PyCharm. One item was added, both inputs were taken in as strings, it was saved as a product, and then written to the text file. This is seen in Figure 5.



***Figure 5: Execution of ‘Assigment08-Starter.py’ in PyCharm***

## Execution from the terminal

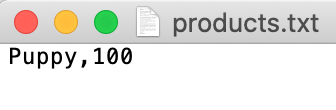
The program Assigment08-Starter.py executed successfully from the terminal. The item added in PyCharm was read in and displayed, two items were added, each item was saved as a product and the proper processing and output was observed (Figure 6). This program did not take a measurable amount of time to execute.



***Figure 6: Execution of ‘Assigment08-Starter.py’ in the terminal***

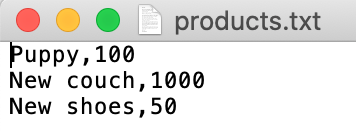
## Results

The program ran successfully through PyCharm first. The program created the file ‘products.txt’ and added one product and its price, as seen in Figure 7.



***Figure 7: Results stored in ‘products.txt’ after PyCharm execution***

The program ran successfully through the terminal second. The program opened the file ‘products.txt’ and appended two more items and their price to the end, as seen in Figure 8.



***Figure 8: Results stored in ‘products.txt’ after terminal execution***

# Summary

The program consists of 174 lines including comments and headers. The intended result to update the catalogue in one living file was achieved with no unintended consequences.