Claire Lynch

December 8, 2020

IT FDN 100 A

Assignment 08

https://github.com/claire-al/IntroToProg-Python-Mod08

Creating Scripts with Classes

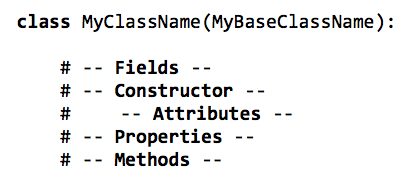
# Introduction

This week, I learned more about how to create scripts using classes. Classes are helpful as they can provide organization to functions and data, and they have some advanced features that can help streamline everything. Classes are helpful because they can group together data and functions, so they can be used repeatedly, instead of having to create new functions each time you want to do something.

# Class Patterns

Classes have several common features including:

* **Fields**: data members of a class, created using variables and constants.
* **Constructors**: special methods/functions that runs automatically when an object is created from a class. Often used to set initial values of Field data. Contains the “self” keyword, which is used to refer to data or functions found in an object instance, but not directly in the class. Other languages will use “this”.
* **Destructors**: although not as commonly used as they used to be, it automatically runs when an object instance is removed from memory.
* **Attributes**: “virtual” fields that hold internal data.
* **Properties**: functions that are used to manage field and attribute data. Has “getters” (gets the data) and “setters” (sets the data)- also sometimes called accessors/mutators.
  + **Setters**: lets you add code for validation and error handling; has @name\_of\_method.setter directive, where directive and function name need to match.
  + **Getters**: lets you add code for formatting a field or attribute’s data. @property directive.
* **Methods:** allow you to organize processing statements into named groups.



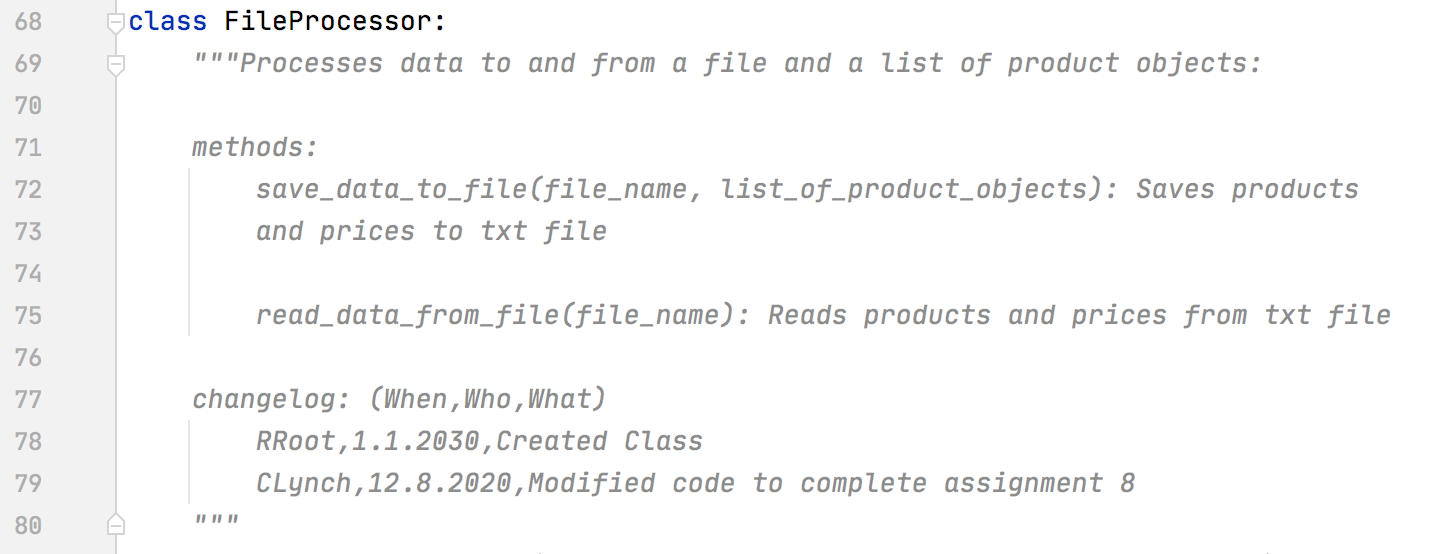
***Figure 1.1: A screenshot of a pseudo-code example of how all these different features fit together in a class.***

# DocStrings

DocStrings should be used in classes. DocStrings are comments at the beginning of a class, which describe the properties and methods, and supplies a change log to keep track of modifications to the class.

GitHub Desktop

While we’ve uploaded files to GitHub through web browsers, it’s also possible to upload to GitHub locally using the command prompt. This interaction is handled by the program, Git. Git makes copies of files and allows you to make changes to files without deleting previously saved versions.



***Figure 2.1: An example of a DocString in my FileProcessor class.***

Writing My Script

***Figure 2.2: My script running, showing first the four albums in the unpickled data, before pickling once again and ending up with three albums.***

# Summary

*# ------------------------------------------------------------------------ #  
# 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  
# Claire Lynch,12.8.2020,Modified code to complete assignment 8  
# ------------------------------------------------------------------------ #  
  
# Data -------------------------------------------------------------------- #*strFileName = **'products.txt'**lstOfProductObjects = []  
  
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  
 CLynch,12.8.2020,Modified code to complete assignment 8  
 """* def \_\_init\_\_(self, product\_name: str, product\_price: float):  
 try:  
 self.\_\_product\_name = str(product\_name)  
 self.\_\_product\_price = float(product\_price)  
 except Exception as e:  
 raise Exception(**"Error setting initial values:** \n**"** + str(e))  
  
 @property  
 def product\_name(self):  
 return str(self.\_\_product\_name)  
  
 @product\_name.setter  
 def product\_name(self, value:str):  
 if str(value).isnumeric():  
 self.\_\_product\_name = value  
 else:  
 raise Exception(**"Names cannot be numbers"**)  
  
 @property  
 def product\_price(self):  
 return float(self.\_\_product\_price)  
  
 @product\_price.setter  
 def product\_price(self, value: float):  
 if str(value).isnumeric():  
 self.\_\_product\_price = float(value) *#cast to float* else:  
 raise Exception(**"Prices must be numbers"**)  
  
 *# -- Methods -- #* def to\_string(self):  
 *""" alias of \_\_str\_\_(), converts product data to string """* return self.\_\_str\_\_()  
  
 def \_\_str\_\_(self):  
 *""" Converts product data and price to string """* return self.product\_name + **","** + str(self.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): Saves products  
 and prices to txt file  
  
 read\_data\_from\_file(file\_name): Reads products and prices from txt file  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 CLynch,12.8.2020,Modified code to complete assignment 8  
 """* def save\_data\_to\_file(file\_name: str, list\_of\_product\_objects: list):  
 *""" Saves products and prices to the txt file  
  
 :param list\_of\_product\_objects: list of products and prices  
 :return: True/False if data was successfully written to file  
 """* success\_status = False  
 try:  
 file = open(file\_name, **"w"**)  
 for product in list\_of\_product\_objects:  
 file.write(product.\_\_str\_\_() + **"**\n**"**)  
 file.close()  
 success\_status = True  
 except Exception as e:  
 print(**"There was a general error!"**)  
 print(e, e.\_\_doc\_\_, type(e), sep=**"**\n**"**)  
 return success\_status  
  
 def read\_data\_from\_file(file\_name):  
 *""" Reads products and prices from txt file  
  
 :return: list of rows of products/prices from txt file  
 """* list\_of\_product\_rows = []  
 try:  
 file = open(file\_name, **"r"**)  
 for line in file:  
 data = line.split(**","**)  
 row = Product(data[0], data[1])  
 list\_of\_product\_rows.append(row)  
 file.close()  
 except Exception as e:  
 print(**"There was a general error!"**)  
 print(e, e.\_\_doc\_\_, type(e), sep=**"**\n**"**)  
 return list\_of\_product\_rows  
  
*# Processing ------------------------------------------------------------- #  
  
# Presentation (Input/Output) -------------------------------------------- #*class IO:  
 *""" Handles user input and output:  
  
 methods:  
 print\_menu\_Tasks(): Prints menu options  
  
 input\_menu\_choice(): Handles user choice from menu options  
  
 print\_current\_products\_in\_list(list\_of\_rows: list): Shows current  
 products/prices in list  
  
 input\_new\_product\_and\_price(): Asks user for product/price info  
 to input  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 CLynch,12.8.2020,Modified code to complete assignment 8  
 """* @staticmethod  
 def print\_menu\_Tasks():  
 *""" Display a menu of choices to the user  
  
 :return: nothing  
 """* print(**'''  
 Menu of Options  
 1) Show current data  
 2) Add a new product  
 3) Save current data  
 4) Exit program  
 '''**)  
 print()  
  
 @staticmethod  
 def input\_menu\_choice():  
 *""" Gets the menu choice from a user  
  
 :return: string  
 """* choice = str(input(**"Which option would you like to perform? [1 to 4] - "**)).strip()  
 print() *# Add an extra line for looks* return choice  
  
 @staticmethod  
 def print\_current\_products\_in\_list(list\_of\_rows: list):  
 *""" Shows the current products in the list of rows  
  
 :param list\_of\_rows: (list) of rows you want to display  
 :return: nothing  
 """* print(**"\*\*\*\*\*\*\* The current Products and Prices are: \*\*\*\*\*\*\*"**)  
 for row in list\_of\_rows:  
 print(row.product\_name + **" ("** + str(row.product\_price) + **")"**)  
 print(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 print() *# Add an extra line for looks* @staticmethod  
 def input\_new\_product\_and\_price():  
 *""" Asks the user for product and price information  
  
 :return: The product/price info the user inputs  
 """* try:  
 product = str(input(**"What is the name of the product? "**).strip())  
 price = float(input(**"How much does the product cost? "**).strip())  
 print()  
 input\_data = Product(product\_name=product, product\_price=price)  
 except Exception as e:  
 print(e)  
 return input\_data;  
  
*# Presentation (Input/Output) -------------------------------------------- #  
  
# Main Body of Script ---------------------------------------------------- #  
# Load data from file into a list of product objects when script starts  
# Show user a menu of options  
# Get user's menu option choice  
 # Show user current data in the list of product objects  
 # Let user add data to the list of product objects  
 # let user save current data to file and exit program*try:  
 lstOfProductObjects = FileProcessor.read\_data\_from\_file(strFileName)  
  
 while (True):  
 IO.print\_menu\_Tasks()  
 strChoice = IO.input\_menu\_choice()  
  
 if strChoice.strip() == **'1'**: *#show current data* IO.print\_current\_products\_in\_list(lstOfProductObjects)  
 continue  
 elif strChoice.strip() == **'2'**: *#add a new product* lstOfProductObjects.append(IO.input\_new\_product\_and\_price())  
 continue  
 elif strChoice.strip() == **'3'**: *#save data* FileProcessor.save\_data\_to\_file(strFileName, lstOfProductObjects)  
 continue  
 elif strChoice.strip() == **'4'**: *#exit program* break  
except Exception as e:  
 print(**"There was an error! Check file permissions!"**)  
 print(e, e.\_\_doc\_\_, type(e), sep=**"**\n**"**)  
  
*# Main Body of Script ---------------------------------------------------- #*

***Figure 3.1: My whole script, with both pickling and unpickling, and exception handling.***