Ryan L. Blake

7 September 2021

IT FDN 100

Assignment 08

Repository: https://github.com/rblake50/IntroToProg-Python-Mod08

Page: https://rblake50.github.io/IntroToProg-Python-Mod08/

**Pickling and Error Handling**

# Introduction

This document investigates objects within a class and how they can be initiated, constructed, and accessed. Objects within classes offer a number of advantages compared to simple variables such as organized error handling and function definition. This document will discuss the basics for objects in classes, the fundamentals of objects, and code management through GitHub desktop. The script example will illustrate all learnings and be published to GitHub through the desktop application.

# Topics

* Class objects
* Object fundamentals
* GitHub desktop

# Summary

## Objects in Classes

A general outline for objects in a class is written below:

class Product:

# Fields

# Constructors

# Attributes

# Properties

# Methods

Following this outline takes a linear approach to object initiation that can be simply understood by other programmers. Each topic will be explored in the next section.

## Object Fundamentals

### Fields

Fields comprise the data for an object within a class. These can be thought of as characteristics that would apply to any object. For example, a Product might be describe by a product name and a product price. Thus, product\_name and product\_price could be suitable fields for an object called Product.

class Product:

product\_name = ""

product\_price = 0.0

These fields act as placeholders for values that can be initiated and changed through constructors and properties as mentioned in the next sections.

### Constructors

Constructors are built-in methods to Python that help define an object when it is initiated. Two useful constructor methods are \_\_init\_\_(self) and \_\_str(self)\_\_. Respectively, these methods define what happens right when an object is initialized and what happens when an object is printed as text. Note that all constructors must be written within a class.

The initialization method can give default values for specified attributes. The example below creates a product with a blank product name (initially) and a price as a zero float.

def \_\_init\_\_(self):

print("A new product has been created!")

# Attributes

self.product\_name = ""

self.product\_price = 0.0

Attributes can also be made “private” such that the attribute cannot be directly accessed through the class. Private attributes are created with two underscores as a prefix to the attribute name. From the above example, the second attribute could be made private by writing self.\_\_product\_price = 0.0. More discussion on private attributes can be found in Chapter 8 of the textbook.

The string method constructs how Python prints an object. Without a string constructor, the text may not be intuitive to a user. The example below concatenates object attributes into a coherent message when the object is directly printed.

def \_\_str\_\_(self):

tmp = "name: " + self.product\_name + ", price: $.2f" % self.product\_price

return tmp

### Properties

Properties act as functions to manage attributes. For example, properties can ensure attribute definition is proper or format attribute data.

The nomenclature for properties can be confusing at first glance. Conversationally, properties can be thought of as “getters” or “setters.” Getters retrieve attributes when commanded. Setters assign values for attributes when commanded. The example below outlines the basic syntax for getters and setters.

@property # Getter

def product\_price(self):

return self.\_product\_price

@product\_price.setter # Setter

def product\_price(self,price):

try:

self.\_product\_price = float(price)

except ValueError:

print("D'oh! Price must be a number. Try again.")

As mentioned, the syntax is critical, and the variable name should be consistent with the attribute from earlier. The getter retrieves the object’s (self) attribute value. The setter assigns a value to the attribute and handles errors in the process. Though tedious, it is a best practice to write a getter and setter property for each attribute within a class.

### Methods

Methods are more or less functions within a class that do not manage attributes. Consider another class example to illustrate a simple method.

class FileProcessor:

# Save data to a pickled file

@staticmethod

def save\_data\_to\_file(file\_name, list\_of\_product\_objects):

with open(file\_name, 'wb'):

pickle.dump(list\_of\_product\_objects, file)

print("File successfully pickled to " + file\_name)

### GitHub Desktop

GitHub Desktop offers another avenue for managing files and committing changes on GitHub in addition to the web browser. GitHub Desktop can be installed from the [GitHub website](https://desktop.github.com/).

Some of the key benefits of GitHub Desktop include:

* File management using Windows explorer
* Clean user interface
* Collaboration

Though not explored in this module, the multi-user collaboration tools offer a way for multiple developers to work on the same files with clear change history.

## Assignment Description

This program allows a user to read list data from a pickled file and read or add data to the list. Then, the user can exit the program and save the data back to the pickled file. The data being stored is Product information containing the name and price of the Product. The Product is managed as a Python object through class properties.

The pseudocode provided is:

# Main Body of Script ---------------------------------------------------- #

# TODO: Add Data Code to the Main body

# 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

This code exploits classes to achieve the end result. Specifically, the code uses class Product to define the product object, class FileProcessor to handle all file read and write methods, and class IO to interact with the user. The code also builds on learnings from previous modules such as error handling, pickling, and while loop menus.

The code can execute through PyCharm. Note that the pickled file begins with 2 products currently on the list.

|  |
| --- |
| C:\\_PythonClass\Assignment08\venv\Scripts\python.exe C:/\_PythonClass/Assignment08/Lecture08/Assigment08-Starter.py  Here is your list of items:  name: computer monitor, price: $115.00  name: record player, price: $97.23  === MENU ===  1. Show current data  2. Add product  3. Save and exit  What is your choice? 2  You chose to add data to the list.  What is the product name? speaker system  What is the product price? 121.23  A new product has been added!  Press ENTER to return to menu.  === MENU ===  1. Show current data  2. Add product  3. Save and exit  What is your choice? 2  You chose to add data to the list.  What is the product name? fridge  What is the product price? i dunno  D'oh! Price must be a number. Try again.  Press ENTER to return to menu.  === MENU ===  1. Show current data  2. Add product  3. Save and exit  What is your choice? 1  Here is your list of items:  name: computer monitor, price: $115.00  name: record player, price: $97.23  name: speaker system, price: $121.23  Press ENTER to return to menu.  === MENU ===  1. Show current data  2. Add product  3. Save and exit  What is your choice? 3  You chose to save and exit.  Do you want to export a .txt file? [y]es or [n]o.  What is your choice? y  File successfully pickled to products.dat  Text file successfully written to products.txt |

The text file can be confirmed:

Text

Description automatically generated

Figure : Exported text file results.

The code can also execute through the Python command shell. This example will look at an initial file that does not exist.

Text

Description automatically generated

Figure : Program results from the Python shell window.

As shown, the program successfully recognizes that the file name (“NEWproducts.dat”) does not exist. The program eventually saves any added items to the desired file name.

# Appendix: Python Script

[Link to Python script](https://github.com/rblake50/IntroToProg-Python-Mod08/blob/main/Assigment08-Starter.py)

*# ------------------------------------------------------------------------ #  
# 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  
# rblake50, 09.04.2021,Modified code to complete assignment 8  
# ------------------------------------------------------------------------ #  
# DESCRIPTION  
# This program allows a user to read list data from a pickled file and  
# read or add data to the list. Then, the user can exit the program and  
# save the data back to the pickled file. The data being stored is Product  
# information containing the name and price of the Product. The Product is  
# managed as a Python object through class properties.  
# ------------------------------------------------------------------------ #  
# ASSUMPTIONS  
# The user is reading from and writing to a pickled file (such as .dat)  
# that is properly organized with Product objects***import** pickle  
**from** datetime **import** datetime  
  
*# Data -------------------------------------------------------------------- #*strFileName = **'products.dat'**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  
 rblake50,09.04.2021,Modified code to complete assignment 8  
 """  
  
 # Fields* product\_name = **""** product\_price = 0.0  
  
 *# Constructors* **def** \_\_init\_\_(self):  
 print(**"A new product has been added!"**)  
  
 *# Attributes* self.product\_name = **""** self.product\_price = 0.0  
  
 **def** \_\_str\_\_(self):  
 tmp = **"name: "** + self.\_product\_name + **", price: $%.2f"** % self.\_product\_price  
 **return** tmp  
  
 *# Properties* @property *# Getter* **def** product\_name(self):  
 **return** self.product\_name  
  
 @product\_name.setter  
 **def** product\_name(self,name):  
 self.\_product\_name = name  
  
 @property *# Getter* **def** product\_price(self):  
 **return** self.product\_price  
  
 @product\_price.setter  
 **def** product\_price(self,price):  
 **try**:  
 self.\_product\_price = float(price)  
 **except** ValueError:  
 print(**"D'oh! Price must be a number. Try again."**)  
  
*# Processing ------------------------------------------------------------- #***class** FileProcessor:  
 *"""Processes data to and from a file and a list of product objects:  
  
 properties:  
 N/A  
 methods:  
 save\_data\_to\_file(file\_name, list\_of\_product\_objects):* **:param** *file\_name (string) name of file to save* **:param** *list\_of\_product\_objects (list) of product objects  
  
 read\_data\_from\_file(file\_name): -> (a list of product objects)* **:param** *file\_name (string) name of file to read* **:return** *list of product objects  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 rblake50,09.04.2021,Modified code to complete assignment 8  
 """  
  
 # Save data to a pickled file* @staticmethod  
 **def** save\_data\_to\_file(file\_name, list\_of\_product\_objects):  
  
 **with** open(file\_name, **'wb'**) **as** file:  
 pickle.dump(list\_of\_product\_objects, file)  
  
 print(**"File successfully pickled to "** + file\_name)  
  
 *# Save data to .txt file* @staticmethod  
 **def** save\_data\_to\_txt(file\_name, list\_of\_product\_objects):  
  
 file\_prefix = file\_name.split(**"."**)[0]  
 file\_to\_save = file\_prefix + **".txt"  
  
 with** open(file\_to\_save, **'w'**) **as** file:  
  
 file.write(datetime.now().strftime(**"%m/%d/%y, %H:%M:%S\n======\n"**))  
 **for** item **in** list\_of\_product\_objects:  
 file.write(item.\_\_str\_\_() + **"\n"**)  
  
 print(**"Text file successfully written to "** + file\_to\_save)  
  
 *# Read data from a pickled file* @staticmethod  
 **def** read\_data\_from\_file(file\_name):  
  
 lstData = [] *# Initiate as list* **try**:  
 **with** open(file\_name, **'rb'**) **as** fileIncoming:  
 lstData = pickle.load(fileIncoming)  
  
 **except** FileNotFoundError: *# File does not exist in immediate directory* print(**"The file "** + file\_name + **" does not exist! Add data and save to create file."**)  
  
 **except** EOFError: *# File exists in immediate directory but has no information* input(**"The file has no content! Please add data."**)  
  
 **return** lstData  
  
*# Presentation (Input/Output) -------------------------------------------- #***class** IO:  
 *"""Manages user input and file output  
  
 properties:  
 methods:  
 show\_menu()  
 1. Show user current data in the list of product objects  
 2. Let user add data to the list of product objects  
 3. let user save current data to file and exit program  
  
 return\_to\_menu()  
  
 get\_choice()* **:return** *string of user choice  
  
 show\_data(file\_name)* **:param** *list\_of\_product\_objects (list) of product objects to show  
  
 get\_data()* **:return** *Product object with name and price of product  
  
 changelog: (When,Who,What)  
 rblake50,09.04.2021,Modified code to complete assignment 8  
 """  
 # Show menu to user* @staticmethod  
 **def** show\_menu():  
 strMenu = **"=== MENU ===\n1. Show current data\n2. Add product\n3. Save and exit"** print(strMenu)  
  
 *# Show prompt to return to menu* @staticmethod  
 **def** return\_to\_menu():  
 input(**"Press ENTER to return to menu."**)  
  
 *# Get user's choice* @staticmethod  
 **def** get\_choice():  
 strChoice = input(**"What is your choice? "**)  
 **return** strChoice  
  
 *# Show the current data from the file to user* @staticmethod  
 **def** show\_data(list\_of\_product\_objects):  
  
 **if** len(list\_of\_product\_objects) != 0:  
 **try**:  
 print(**"Here is your list of items:"**)  
 **for** item **in** list\_of\_product\_objects:  
 print(item)  
  
 **except** TypeError:  
 input(**"List is the wrong type! Please add data."**)  
  
 **else**:  
 print(**"The list is empty. Please add items."**)  
  
 *# Get product data from user* @staticmethod  
 **def** get\_data():  
  
 **try**:  
 name = input(**"What is the product name? "**)  
 price = float(input(**"What is the product price? "**))  
  
 prod = Product()  
 prod.product\_name = name  
 prod.product\_price = price  
  
 **return** prod  
  
 **except** ValueError:  
 print(**"D'oh! Price must be a number. Try again."**)  
  
*# Main Body of Script ---------------------------------------------------- #  
  
# Load data from file into a list of product objects when script starts*lstOfProductObjects = FileProcessor.read\_data\_from\_file(strFileName)  
  
*# Show data from loaded file*IO.show\_data(lstOfProductObjects)  
  
**while True**:  
  
 *# Show user a menu of options* IO.show\_menu()  
  
 *# Get user's menu option choice* choice = IO.get\_choice()  
  
 *# 1. Show user current data in the list of product objects* **if** choice == **"1"**:  
  
 IO.show\_data(lstOfProductObjects)  
  
 IO.return\_to\_menu()  
  
 *# 2. Let user add data to the list of product objects* **elif** choice == **"2"**:  
 print(**"You chose to add data to the list."**)  
  
 *# Get data for new Product from user* product = IO.get\_data()  
 **if** product **is not None**: *# get\_data() will return None for invalid entry* lstOfProductObjects.append(product)  
  
 *# Prompt return to menu* IO.return\_to\_menu()  
  
 *# 3. let user save current data to file and exit program* **elif** choice == **"3"**:  
  
 print(**"You chose to save and exit."**)  
  
 *# Prompt user for a text file copy of the data* print(**"Do you want to export a .txt file? [y]es or [n]o."**)  
 choice = IO.get\_choice().lower()  
  
 *# User wants to save text file* **if** choice == **"y" or** choice == **"yes"**:  
  
 *# Save data to pickled file and text file with same file name (less extension)* FileProcessor.save\_data\_to\_file(strFileName, lstOfProductObjects)  
 FileProcessor.save\_data\_to\_txt(strFileName, lstOfProductObjects)  
 **break** *# User does not want to save text file* **elif** choice == **"n" or** choice == **"no"**:  
  
 *# Save data to pickled file but \*NOT\* to text file* FileProcessor.save\_data\_to\_file(strFileName, lstOfProductObjects)  
 print(**"File \*NOT\* written to text file. Good-bye!"**)  
 **break** *# Invalid entry* **else**:  
  
 print(**"Invalid selection! Data will not be saved."**)  
 IO.return\_to\_menu()  
  
 *# Invalid choice* **else**:  
 print(**"D'oh! Your choice is invalid."**)  
 IO.return\_to\_menu()