

Neetu Upadhyay  
June 24, 2023  
Foundations of Programming, Python  
Assignment 08- Final  
GitHubURL :-  
**GitHub Webpage :-**

## **Basic Writing and Executing a Python Script**

### **Introduction**

In this module, you will dive into the world of custom classes and learn how to create scripts that leverage their power. Classes serve as a fundamental building block in programming, enabling you to effectively organize functions and data within your code. Understanding the key concepts of classes is not only essential in Python but also applicable to other programming languages.

Throughout this module, you will explore the crucial elements of custom classes, gaining valuable insights into how they enhance code organization and reusability. By the end, you will have a solid foundation in creating scripts using custom classes, equipping you with valuable skills that can be applied across various programming contexts. Let's embark on this exciting journey of class-based programming!

### **Understanding the Distinction between Classes and Objects**

In object-oriented programming, a class is a blueprint or a template that defines the structure and behavior of objects. It serves as a blueprint for creating multiple instances of objects with similar properties and functionalities. On the other hand, objects are instances of a class that are created based on the class's blueprint.

Here's an example to illustrate the difference:

Class: Car  
Attributes: color, brand, model  
Methods: start\_engine(), accelerate(), brake()

In this example, the "Car" class acts as a blueprint for creating car objects. The class defines the common attributes that all cars will have, such as color, brand, and model. It also specifies the methods that all cars can perform, such as starting the engine, accelerating, and braking.

Object 1: MyCar  
Attributes: color = "blue", brand = "Toyota", model = "Corolla"

Object 2: FriendCar  
Attributes: color = "red", brand = "Honda", model = "Civic"

In this case, "MyCar" and "FriendCar" are two different objects (instances) of the "Car" class. Each object has its own set of attribute values, but they share the same structure and behavior defined by the class.

The main difference between the class and objects lies in their roles. The class serves as a blueprint that defines the common properties and behaviors of objects, while objects are specific instances created based on that blueprint, possessing their own unique attribute values.

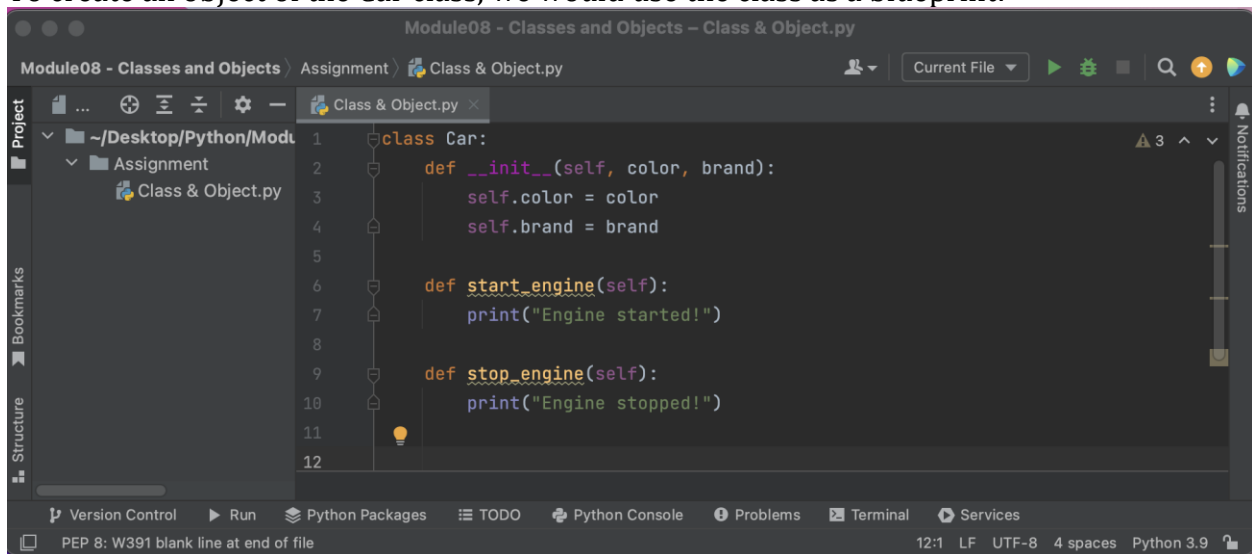
In summary, a class is a general representation of an object, providing a blueprint for creating multiple instances with shared characteristics. Objects, on the other hand, are specific instances created from a class, each having its own distinct attribute values but adhering to the structure and behavior defined by the class.

In programming, a class is like a blueprint or template that defines how an object should be created. It specifies the attributes (properties) and behaviors (methods) that the object will have.

On the other hand, an object is an instance or a specific occurrence of a class. It is created based on the blueprint provided by the class.

For example, let's consider a class called "Car" that represents cars. The class may have attributes like color, brand, and speed, as well as methods like start(), accelerate(), and stop().

To create an object of the Car class, we would use the class as a blueprint:

A screenshot of a code editor window titled "Module08 - Classes and Objects - Class & Object.py". The editor shows a Python class definition for "Car". The class has three methods: "\_\_init\_\_" which initializes "color" and "brand" attributes, "start\_engine" which prints "Engine started!", and "stop\_engine" which prints "Engine stopped!". The code is as follows:

```
1 class Car:
2     def __init__(self, color, brand):
3         self.color = color
4         self.brand = brand
5
6     def start_engine(self):
7         print("Engine started!")
8
9     def stop_engine(self):
10        print("Engine stopped!")
11
12
```

The editor interface includes a sidebar with "Project", "Bookmarks", and "Structure" views. The bottom status bar shows "PEP 8: W391 blank line at end of file", "12:1", "LF", "UTF-8", "4 spaces", and "Python 3.9".

Figure. 1

Now, we can create objects (car instances) based on the Car class:

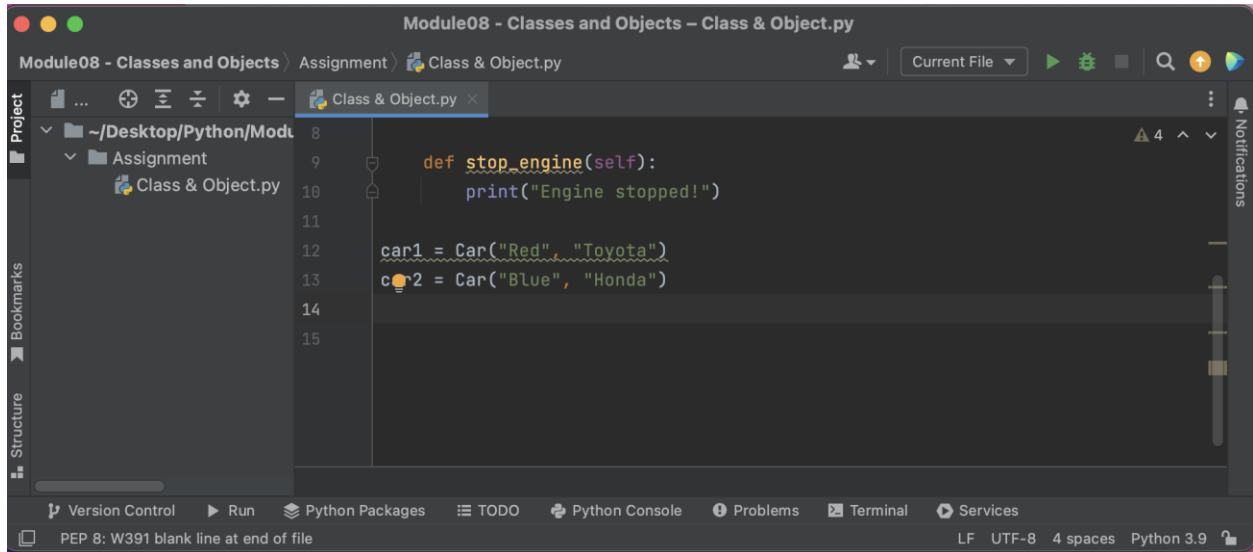


Figure.2

Each object (car1 and car2) will have its own set of attributes (color and brand) and can perform the defined methods (start(), accelerate(), and stop()).

To access the attributes of an object, we use dot notation:

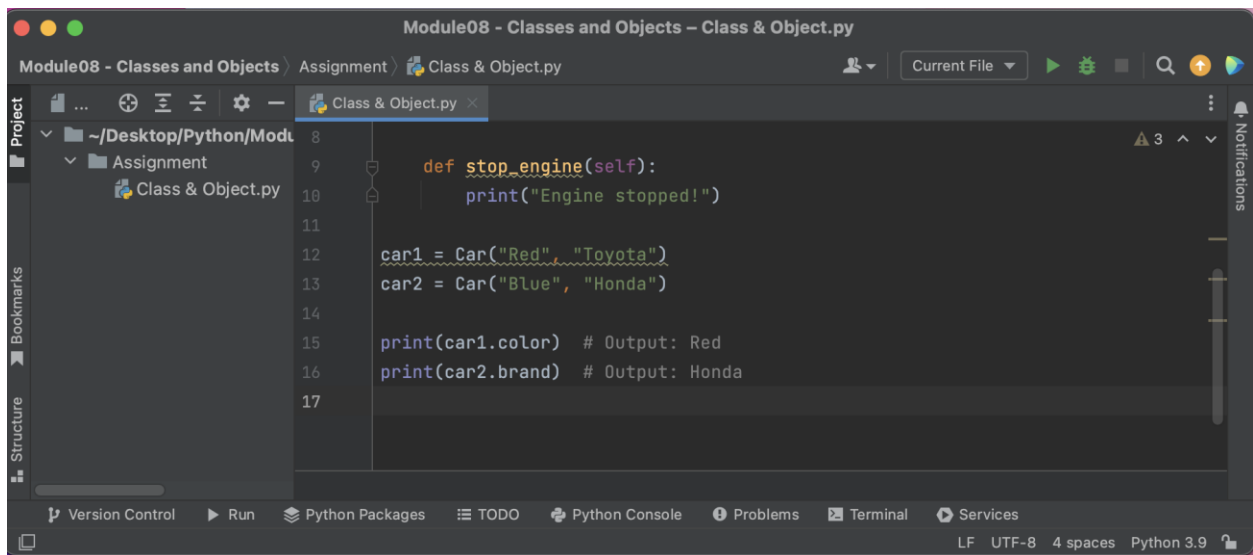


Figure. 3

To call the methods of an object, we also use dot notation:

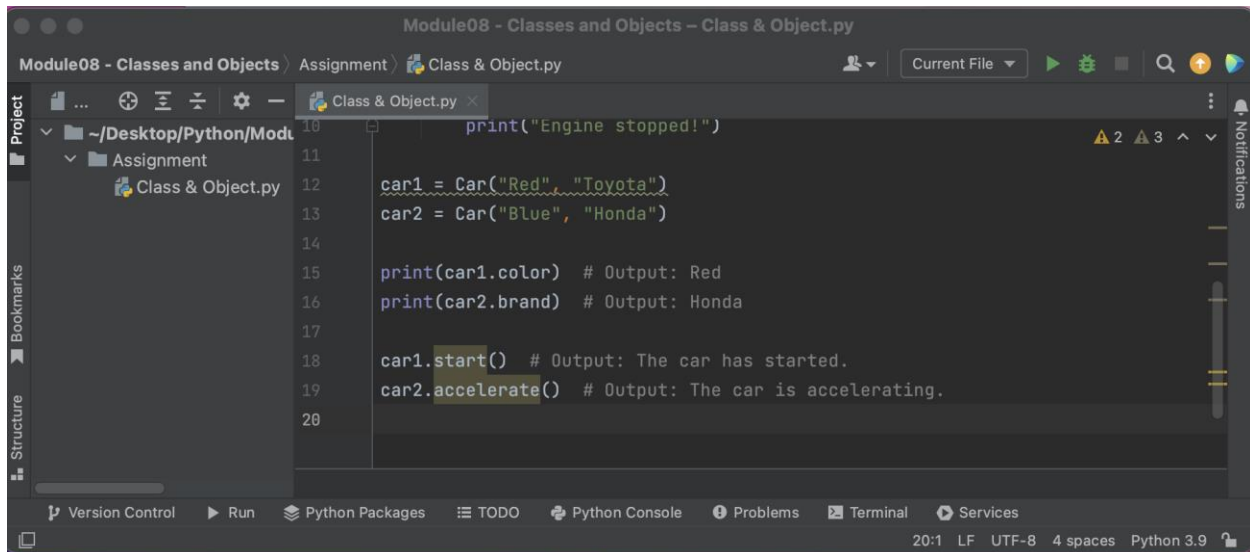


Figure.4

So, the main difference between a class and objects is that a class is a blueprint or template, while objects are instances created based on that blueprint.

### **Components of a Class: Standard Pattern**

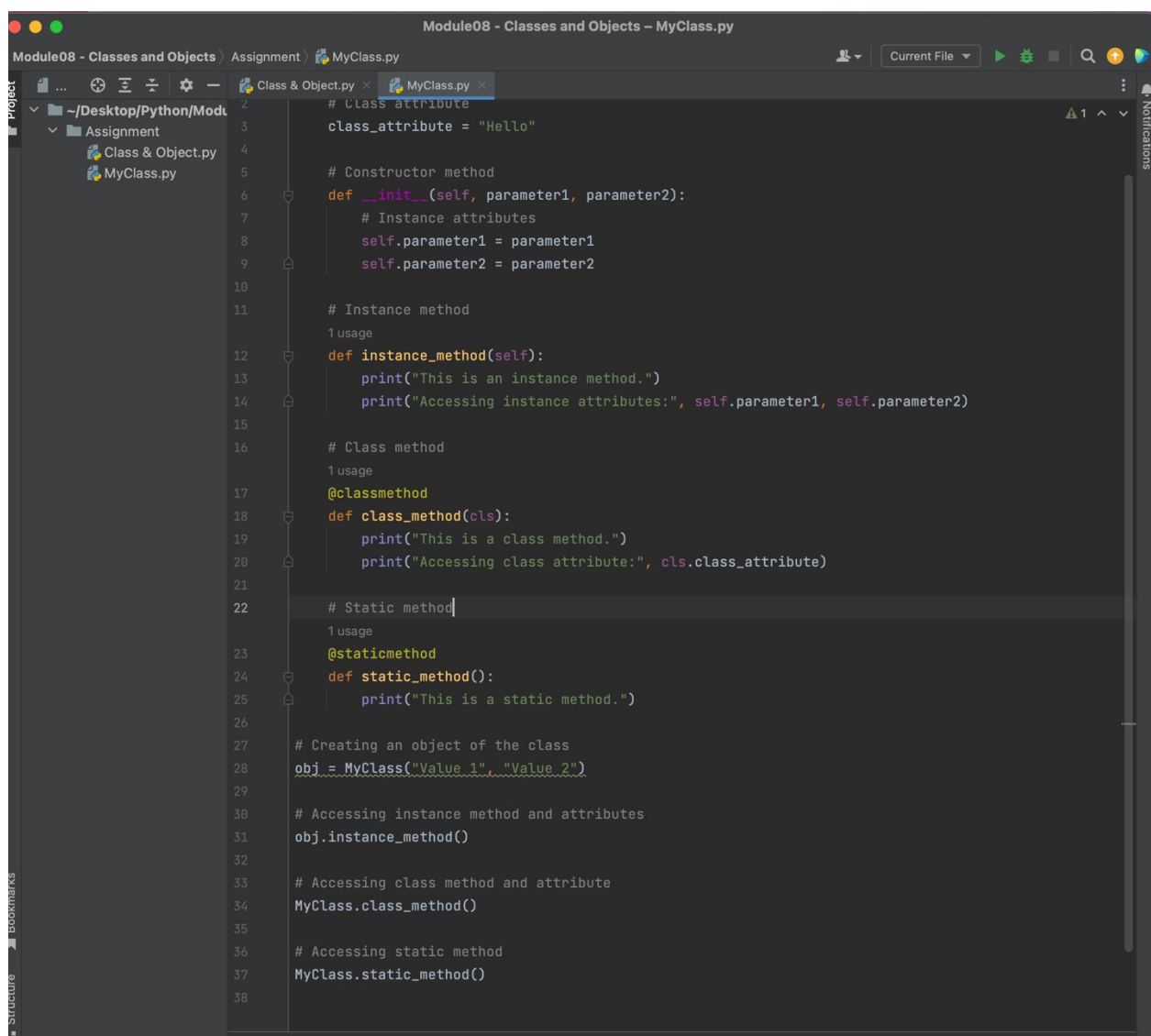
The standard pattern of a class in Python typically consists of the following components:

1. **Class declaration:** This is where the class is defined using the class keyword, followed by the name of the class. For example: `class MyClass:`.
2. **Constructor:** The constructor is a special method called `__init__()` that is used to initialize the object and its attributes. It is executed automatically when an object is created from the class. The constructor method typically takes parameters that initialize the attributes of the object. For example: `def __init__(self, param1, param2):`.
3. **Class attributes:** These are variables defined within the class that are shared by all instances (objects) of the class. They are typically defined outside of any method within the class. For example: `class_attribute = 10`.
4. **Instance attributes:** These are variables specific to each instance of the class. They are defined within the constructor method using the `self` keyword. For example: `self.instance_attribute = 20`.
5. **Methods:** These are functions defined within the class that perform specific actions or operations on the object. They can access and modify the object's attributes. Methods are defined using the `def` keyword, and the first parameter is usually `self`, which refers to the instance of the object. For example: `def method_name(self, param1, param2):`.

6. **Getters and setters:** These are special methods used to access (get) and modify (set) the values of the class attributes. They provide control over how the attributes are accessed and modified. Getters typically have names like `get_attribute()` and setters have names like `set_attribute()`, where attribute is the name of the class attribute.
7. **Other class methods:** In addition to the constructor and regular methods, classes can have other methods that perform specific operations or provide functionality related to the class.

These components together form the standard pattern of a class in Python and provide the structure and behavior for creating objects and working with them.

Here's an explanation of the components of a class, illustrated with code:



```
Module08 - Classes and Objects - MyClass.py
Class & Object.py x MyClass.py
~/Desktop/Python/Mod...
Assignment
  Class & Object.py
  MyClass.py

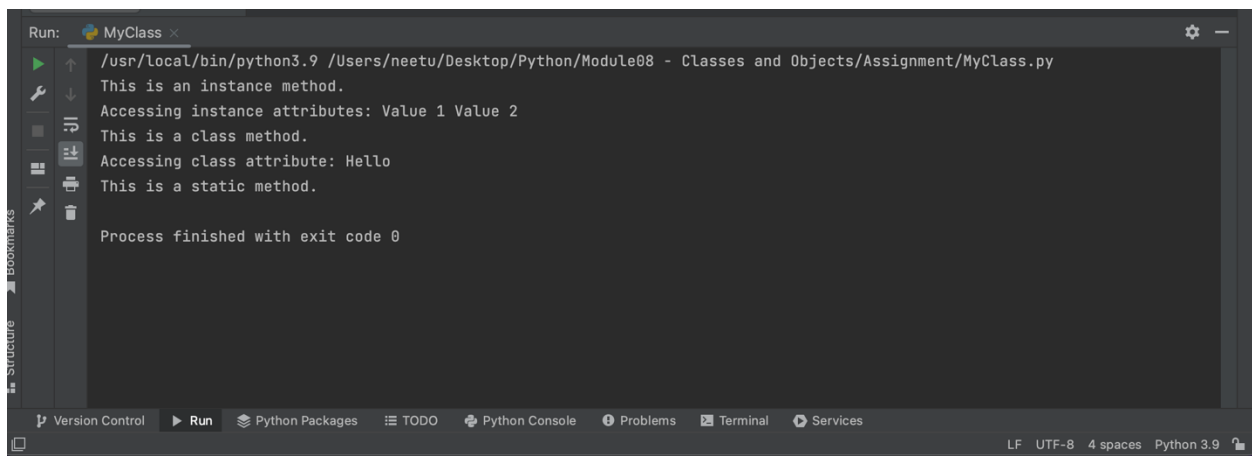
2 # Class attribute
3 class_attribute = "Hello"
4
5 # Constructor method
6 def __init__(self, parameter1, parameter2):
7     # Instance attributes
8     self.parameter1 = parameter1
9     self.parameter2 = parameter2
10
11 # Instance method
12 1 usage
13 def instance_method(self):
14     print("This is an instance method.")
15     print("Accessing instance attributes:", self.parameter1, self.parameter2)
16
17 # Class method
18 1 usage
19 @classmethod
20 def class_method(cls):
21     print("This is a class method.")
22     print("Accessing class attribute:", cls.class_attribute)
23
24 # Static method
25 1 usage
26 @staticmethod
27 def static_method():
28     print("This is a static method.")
29
30 # Creating an object of the class
31 obj = MyClass("Value 1", "Value 2")
32
33 # Accessing instance method and attributes
34 obj.instance_method()
35
36 # Accessing class method and attribute
37 MyClass.class_method()
38
39 # Accessing static method
40 MyClass.static_method()
```

**Figure.5**

In the code above, we have a class called MyClass. It demonstrates the standard components of a class:

1. Class attribute (class\_attribute): It is a variable defined at the class level and is shared by all instances of the class.
2. Constructor method (\_\_init\_\_): It is called when creating an object of the class and initializes the instance attributes (parameter1 and parameter2).
3. Instance method (instance\_method): It is a method that operates on the instance of the class and can access the instance attributes.
4. Class method (class\_method): It is a method that operates on the class itself rather than instances. It can access the class attributes.
5. Static method (static\_method): It is a method that doesn't depend on the instance or class state. It is defined using the @staticmethod decorator.

By running the code, you can see how each component is used and accessed within the class.



```
Run: MyClass x
/usr/local/bin/python3.9 /Users/neetu/Desktop/Python/Module08 - Classes and Objects/Assignment/MyClass.py
This is an instance method.
Accessing instance attributes: Value 1 Value 2
This is a class method.
Accessing class attribute: Hello
This is a static method.
Process finished with exit code 0
```

Figure.6

### **Explanation:**

- When we call the instance\_method() on the obj object, it prints the message "This is an instance method" and then accesses the instance attributes parameter1 and parameter2, printing their respective values.
- When we call the class\_method() on the MyClass class itself, it prints the message "This is a class method" and then accesses the class attribute class\_attribute, printing its value.
- When we call the static\_method() on the MyClass class, it simply prints the message "This is a static method".

## Purpose of a Class Constructor

The purpose of a class constructor is to initialize the object of a class. It is a special method that is automatically called when an object is created from the class. The constructor method is typically named `__init__()` and it allows us to define and set the initial state (attributes) of the object.

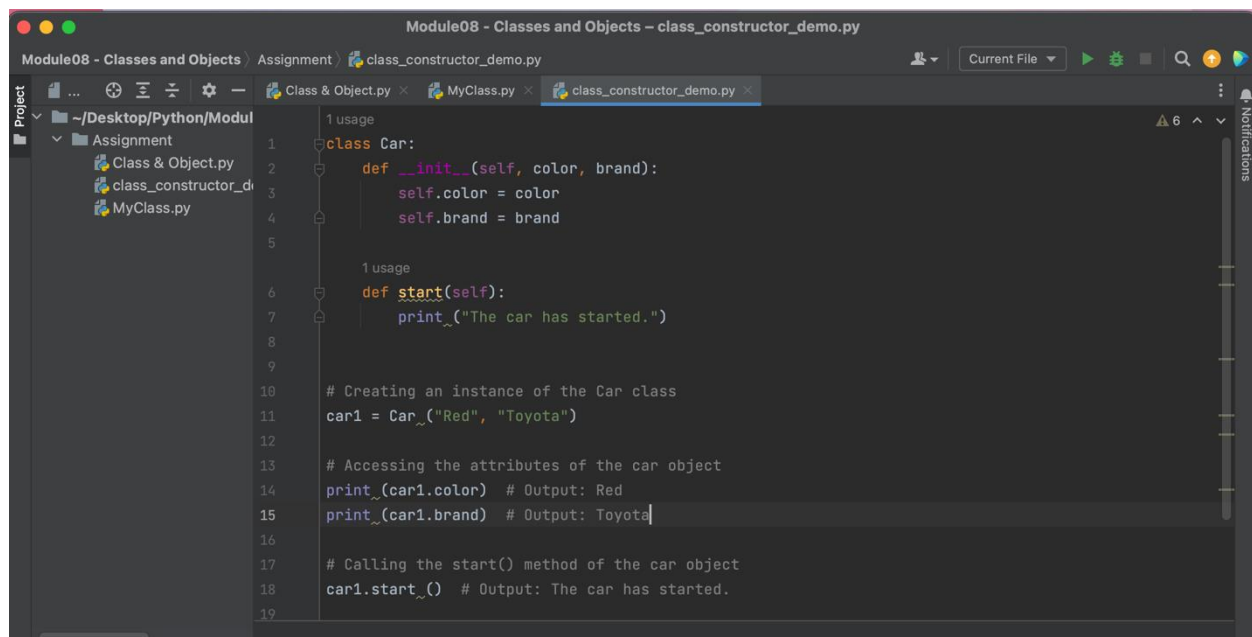
The constructor is useful for performing any necessary setup or initialization tasks before we start working with the object. It allows us to specify the initial values of the object's attributes, which can be passed as arguments when creating an instance of the class.

By defining a constructor, we ensure that the necessary attributes are properly initialized and ready for use when the object is created. It helps in maintaining the integrity and consistency of the object's state throughout its lifecycle.

In summary, the purpose of a class constructor is to set up the initial state of an object by initializing its attributes, ensuring that the object is in a valid and usable state from the moment of its creation.

In Python, a class constructor is a special method that is automatically called when an object of a class is created. It is used to initialize the object's attributes or perform any necessary setup operations.

Here's an example code that demonstrates the use of a class constructor:

A screenshot of a code editor window titled "Module08 - Classes and Objects - class\_constructor\_demo.py". The editor shows a Python class named "Car" with an "\_\_init\_\_" method that takes "color" and "brand" as arguments and assigns them to "self.color" and "self.brand". There is also a "start" method that prints "The car has started.". Below the class definition, there is an instance creation: "car1 = Car("Red", "Toyota")", followed by attribute access: "print(car1.color)" (output: Red) and "print(car1.brand)" (output: Toyota), and finally a method call: "car1.start()" (output: The car has started.). The editor interface includes a project sidebar on the left, a top toolbar with icons for file operations, and a right sidebar for notifications.

```
1 class Car:
2     def __init__(self, color, brand):
3         self.color = color
4         self.brand = brand
5
6     def start(self):
7         print("The car has started.")
8
9
10 # Creating an instance of the Car class
11 car1 = Car("Red", "Toyota")
12
13 # Accessing the attributes of the car object
14 print(car1.color) # Output: Red
15 print(car1.brand) # Output: Toyota
16
17 # Calling the start() method of the car object
18 car1.start() # Output: The car has started.
19
```

Figure.7

In the above code, the `__init__` method is the constructor of the Car class. It takes two parameters (color and brand) along with the self parameter, which refers to the instance of the class. Inside the constructor, the attributes color and brand are assigned values based on the arguments passed during object creation.

When we create an object `car1` of the `Car` class, the constructor `__init__` is automatically called, and the attributes `color` and `brand` are initialized. We can then access these attributes using dot notation (`car1.color`, `car1.brand`).

The class constructor allows us to set the initial state of an object and provide values to its attributes. It helps ensure that the object is properly initialized before any other operations are performed on it.

## The Self Keyword

The keyword `"self"` is used in Python within the methods of a class to refer to the instance of the class itself. It is a convention to name the first parameter of a method as `"self"` (though you can technically use any valid variable name).

You use the keyword `"self"` when defining methods within a class to access the instance variables and other methods of the class. It allows you to refer to the specific instance of the class on which the method is called.

For example, consider the following code snippet:

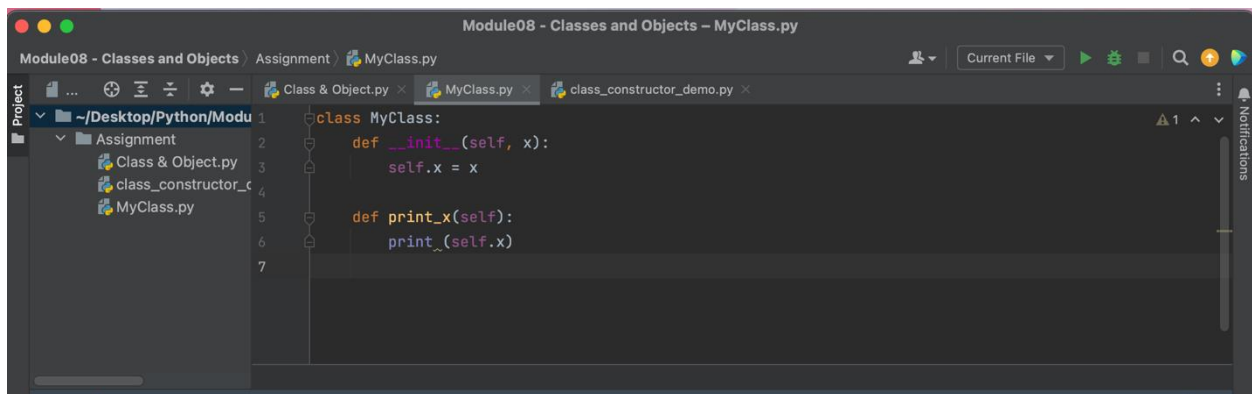


Figure.8

In this example, the `'self'` parameter is used in both the `'__init__'` method and the `'print_x'` method. When creating an instance of the `'MyClass'` class, the `'self'` parameter in the `'__init__'` method refers to the newly created instance, and `'self.x'` is used to assign a value to the instance variable `'x'`.

Similarly, when calling the `'print_x'` method on an instance of the class, the `'self'` parameter inside the method refers to that specific instance, allowing us to access its `x` variable using `'self.x'`.

Using the `'self'` keyword ensures that each instance of a class has its own separate set of instance variables and can access its own methods and attributes.



## Using the @staticmethod Decorator in Python

The `@staticmethod` decorator is used in Python to define a static method within a class. A static method is a method that belongs to the class itself rather than an instance of the class. It can be called directly on the class without creating an object of the class.

You would use the `@staticmethod` decorator when you have a method in a class that doesn't access or modify any instance-specific data. It is independent of the state of the object and doesn't require any access to instance variables or methods. Static methods are commonly used for utility functions or operations that are not specific to any particular instance.

Here's an example to illustrate the usage of `@staticmethod`:

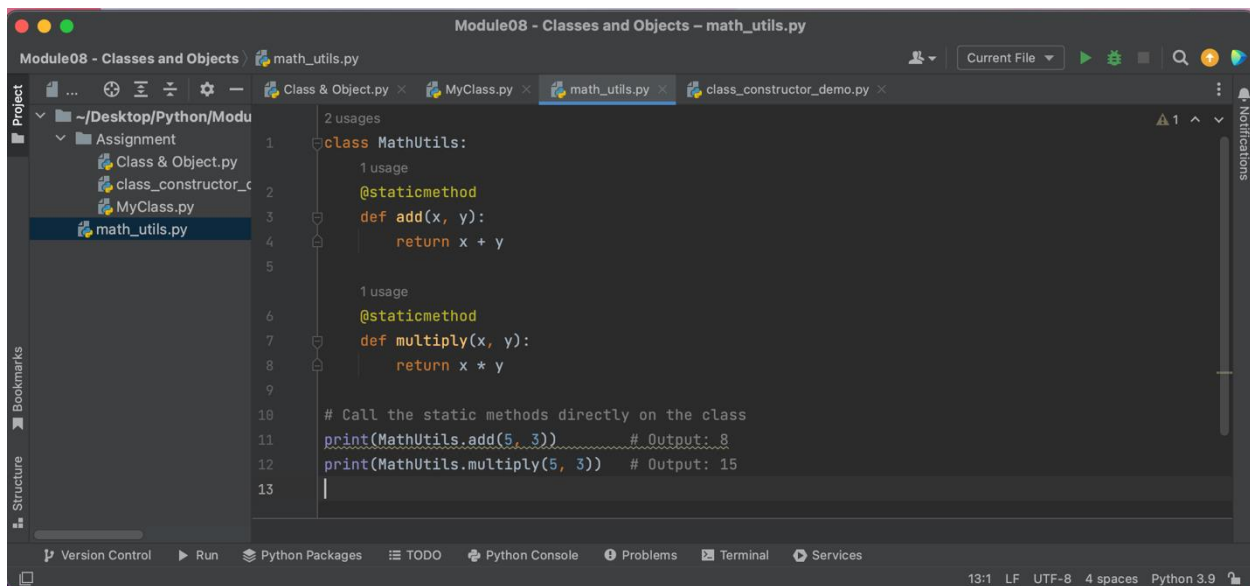
A screenshot of a code editor window titled "Module08 - Classes and Objects - math\_utils.py". The editor shows a Python class named `MathUtils` with two static methods: `add` and `multiply`. Both methods are decorated with `@staticmethod`. The `add` method takes two arguments `x` and `y` and returns their sum. The `multiply` method takes two arguments `x` and `y` and returns their product. Below the class definition, there are two lines of code that call these static methods directly on the class: `print(MathUtils.add(5, 3))` and `print(MathUtils.multiply(5, 3))`. The output of these calls is shown as comments: `# Output: 8` and `# Output: 15`. The editor interface includes a project sidebar on the left, a top toolbar with various icons, and a bottom status bar showing the file encoding (UTF-8), indentation (4 spaces), and Python version (3.9).

Figure .9

In the above code, we have a **MathUtils** class with two static methods: **add** and **multiply**. These methods don't require any instance of the class and can be called directly on the class itself.

By using the `@staticmethod` decorator, we indicate that these methods are static methods. It's important to note that static methods do not have access to instance variables or methods. They are self-contained and operate solely on the arguments passed to them.

## Understanding the Relationship Between Fields, Attributes, and Property Functions

Fields, attributes, and property functions are related concepts in object-oriented programming. Here's an explanation of their relationship:

**Fields:** Fields, also known as instance variables, are data members that belong to an object of a class. They hold the state or data associated with the object. Fields are declared within a class and can have different data types.

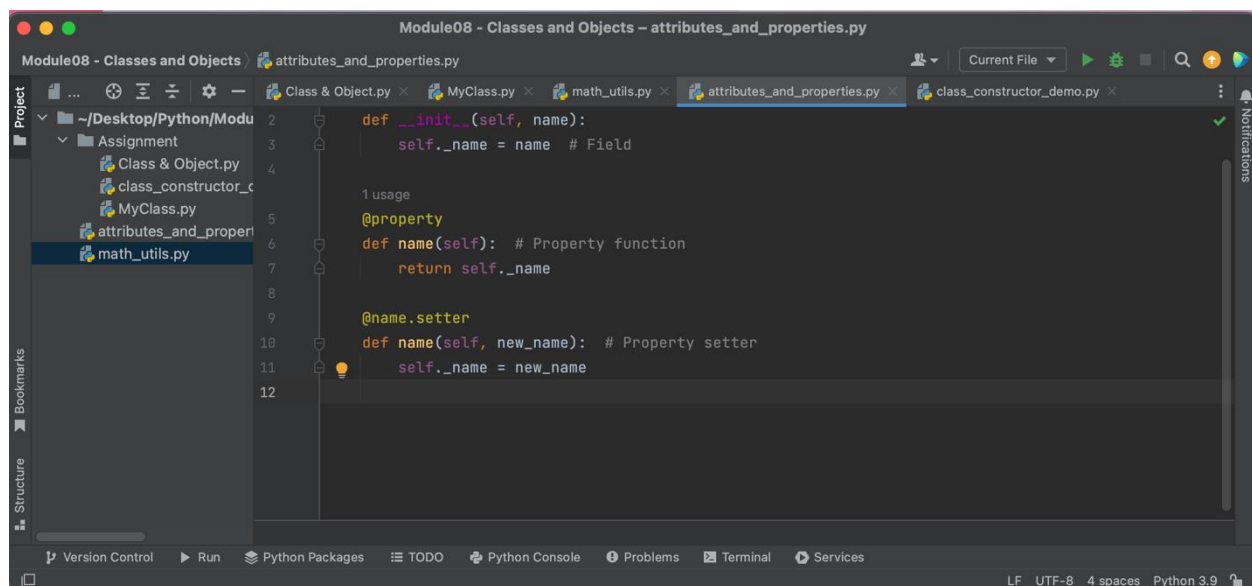
**Attributes:** In Python, attributes are used to access and manipulate the fields of an object. Attributes provide a way to interact with the fields of an object from outside the class. They can be accessed using the dot notation, like **object.attribute**.

**Property Functions:** Property functions, also known as getter and setter methods, are special methods used to control the access and modification of attributes. They allow you to define custom behavior when getting or setting the value of an attribute. Property functions are used to enforce data encapsulation and provide data validation or transformation. The relationship between fields, attributes, and property functions can be summarized as follows:

- Fields are the actual data members that hold the state of an object.
- Attributes are used to access and manipulate the fields of an object from outside the class.
- Property functions provide a way to define custom behavior for accessing and setting the attributes, allowing for data encapsulation and validation.

By using property functions, you can define how attribute access and assignment should be handled, enabling more controlled and consistent interaction with the object's data.

Certainly! Here's an example code snippet to illustrate the relationship between fields, attributes, and property functions:

A screenshot of a code editor window titled "Module08 - Classes and Objects - attributes\_and\_properties.py". The editor shows a Python class with a private field and property functions. The code is as follows:

```
1 class MyClass:
2     def __init__(self, name):
3         self.__name = name # Field
4
5     1 usage
6     @property
7     def name(self): # Property function
8         return self.__name
9
10    @name.setter
11    def name(self, new_name): # Property setter
12        self.__name = new_name
```

The editor interface includes a project explorer on the left, a toolbar at the top, and a status bar at the bottom showing "LF UTF-8 4 spaces Python 3.9".

Figure.10

In this code, we have a **Person** class with a field called **\_name**. The field is initialized in the constructor (**\_\_init\_\_**) and marked with an underscore to indicate that it's intended to be a private attribute.

To access this field in a controlled manner, we define a property function named **name** using the **@property** decorator. The **name** property allows us to get the value of the **\_name** field without directly accessing it.

Additionally, we provide a property setter using the **@name.setter** decorator. This setter allows us to modify the value of the **\_name** field through the **name** property.

Here's an example of how we can use this class:

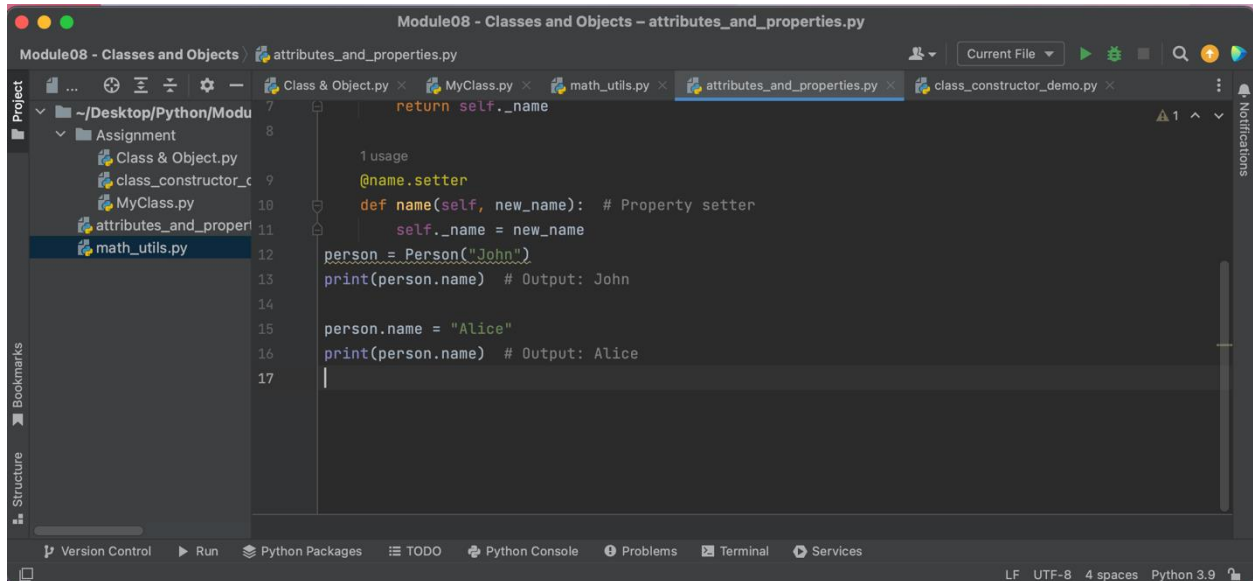


Figure.11

In the above code, we create a **Person** object and assign the name "John" to it. We then use the **name** property to retrieve and print the person's name. After that, we update the name using the **name** property setter, and again retrieve and print the updated name.

This example demonstrates how fields, attributes, and property functions are related. The field **\_name** acts as the underlying data storage, while the property function **name** provides controlled access to this data, allowing us to get and set its value.

### Distinguishing Properties and Methods in Python Classes

A property and a method are both components of a class, but they serve different purposes.

A method is a function that is defined within a class and performs some specific actions or calculations. It typically takes parameters, performs operations, and may return a value. Methods are used to implement the behavior of an object and can be called using dot notation on an object instance.

On the other hand, a property is a special kind of attribute that provides access to its underlying data or performs specific actions when getting, setting, or deleting its value. Properties are used to encapsulate the access to attributes and allow for controlled access and manipulation of data within an object.

The main difference between a property and a method is in how they are accessed and used:

- A method is called explicitly using parentheses, and it can take arguments and return a value. For example, **object.method()**.
- A property, on the other hand, is accessed like an attribute, without using parentheses. It appears as a normal attribute, but behind the scenes, it has associated getter, setter, and deleter methods that are called automatically when accessing or modifying the property's value. For example, **'object.property'**.

In summary, methods are used for performing actions or calculations, while properties are used to provide controlled access to attributes and perform specific actions when accessing or modifying them.

### Understanding the use of super() in Python for inheritance and method overriding.

Including a docstring in a class is important for providing documentation and information about the class to developers who may be using or extending it. A docstring is a string literal that serves as documentation for a specific object, such as a class, method, or module.

Here are some reasons why including a docstring in a class is beneficial:

1. **Documentation for users:** A well-written docstring helps users understand the purpose, behavior, and usage of the class. It provides an overview of what the class does, its attributes, methods, and any important considerations or limitations.
2. **Improved code readability:** A clear and concise docstring enhances the readability of the code. It allows other developers (including yourself) to quickly understand the class without diving into the implementation details.
3. **Autogenerated documentation:** Docstrings can be used by documentation generators (such as Sphinx) to automatically generate documentation in various formats, including HTML, PDF, or plain text. This makes it easier to create comprehensive and up-to-date documentation for your codebase.
4. **Aid in debugging and troubleshooting:** When encountering issues or bugs related to the class, a well-written docstring can provide insights into how the class is intended to be used, common pitfalls, and any specific guidelines for troubleshooting.

To include a docstring in a class, you can simply add a multiline string as the first statement within the class body. Here's an example:

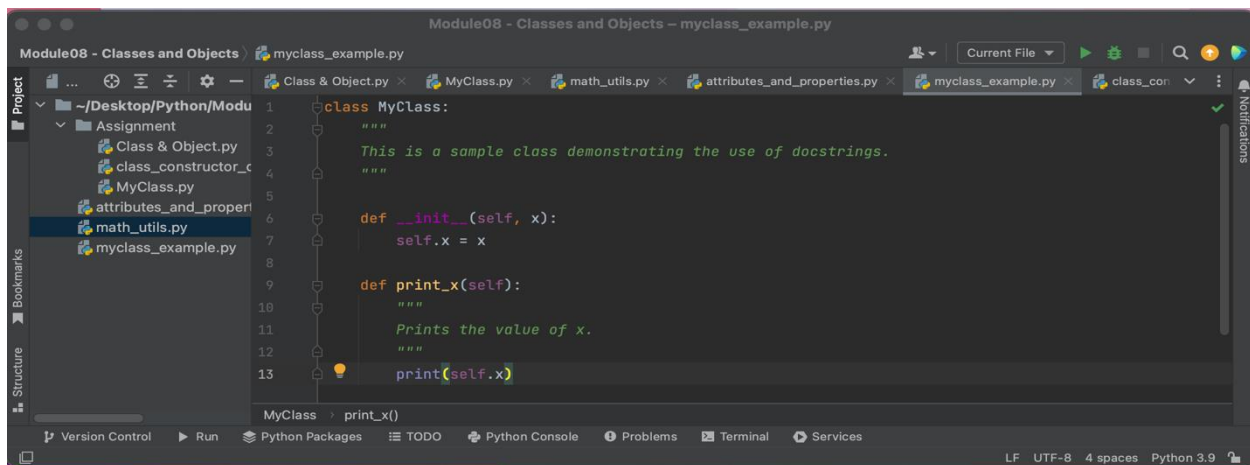


Figure.12

This code includes a class called **MyClass** with a docstring both at the class level and the method level. You can use this code as a reference to understand how to include docstrings in your own class.

## **Difference Between Git and GitHub**

Git and GitHub are related but serve different purposes:

**Git:** Git is a distributed version control system that allows you to track changes to your code files over time. It provides features like creating branches, committing changes, merging code, and more. Git is primarily a command-line tool and can be used locally on your computer or within a team.

**GitHub:** GitHub is a web-based platform that hosts Git repositories. It provides additional collaboration features on top of Git, making it easier for multiple developers to work together on a project. GitHub offers features like pull requests, issue tracking, project management, code reviews, and more. It also provides a graphical user interface (GUI) to interact with Git repositories.

In summary, Git is the version control system itself, while GitHub is a platform built around Git that adds collaboration and team management features. You can use Git without GitHub, but GitHub leverages the power of Git and adds extra functionality for collaborative software development.

## **GitHub Desktop: Simplifying Git Repository Management and Collaboration**

GitHub Desktop is a graphical user interface (GUI) application that provides an intuitive way to interact with Git repositories hosted on GitHub. It is designed to simplify the process of managing and collaborating on Git projects, especially for users who are less familiar with the command line interface.

With GitHub Desktop, you can perform various Git-related tasks such as creating and cloning repositories, creating branches, committing changes, merging branches, resolving merge conflicts, and pushing changes to remote repositories. The application provides a visual representation of your repository's history and allows you to view and compare changes between different versions of your code.

GitHub Desktop also integrates seamlessly with GitHub.com, allowing you to easily sync your local repositories with your remote repositories on GitHub. You can publish your local changes, pull updates from remote repositories, and collaborate with others through pull requests, all from within the GitHub Desktop interface.

Overall, GitHub Desktop provides a user-friendly and accessible way to work with Git and GitHub, making it easier for individuals and teams to manage and collaborate on software projects.

## Python Script

Writing my 8th Python script. To do this, I opened pycharm, which is a Python environment for writing and running scripts. I watched some videos recommended by Professor Root to learn how to complete my tasks.

Here's a simplified step-by-step description of the code screenshot is attached please see figure 13, 14 & 15:

To describe and run the Python script step by step, follow these instructions:

1. Start by opening PyCharm.
2. Create a new Python file and save it with a `_PythonClass\Assignment08.py` extension.
3. Copy the entire code from the given script and paste it into your Python file.
4. Save the file & run the command.
5. Open a command prompt or terminal and navigate to the directory where you saved the Python file.
6. Run the script by executing the command.
7. The script will start running, and you will see the menu of options displayed in the terminal.
8. Follow the menu instructions and enter your choice (1-4) based on the action you want to perform.
  - If you choose **1**, it will display the current data in the list of product objects.
  - If you choose **2**, it will prompt you to enter the product name and price to add a new item.
  - If you choose **3**, it will save the current data to the file and exit the program.
  - If you choose **4**, it will exit the program without saving the data.
9. The script will continue running in a loop until you choose to exit.
10. The script handles errors in a few places:
  - When reading data from the file, if an error occurs (e.g., file not found, permission denied), it will catch the **IOError** and display an error message.
  - When entering the product price, if the input is not a numeric value, it will catch the **ValueError** and prompt you to enter a valid numeric value.
11. You can test the error handling by deliberately causing errors, such as entering a non-numeric value for the product price or providing an invalid file name.
12. After running the script, observe the output in the terminal based on your actions and inputs.

By following these steps, you can describe and run the Python script step by step, observe the error handling, and see the output generated by the script.

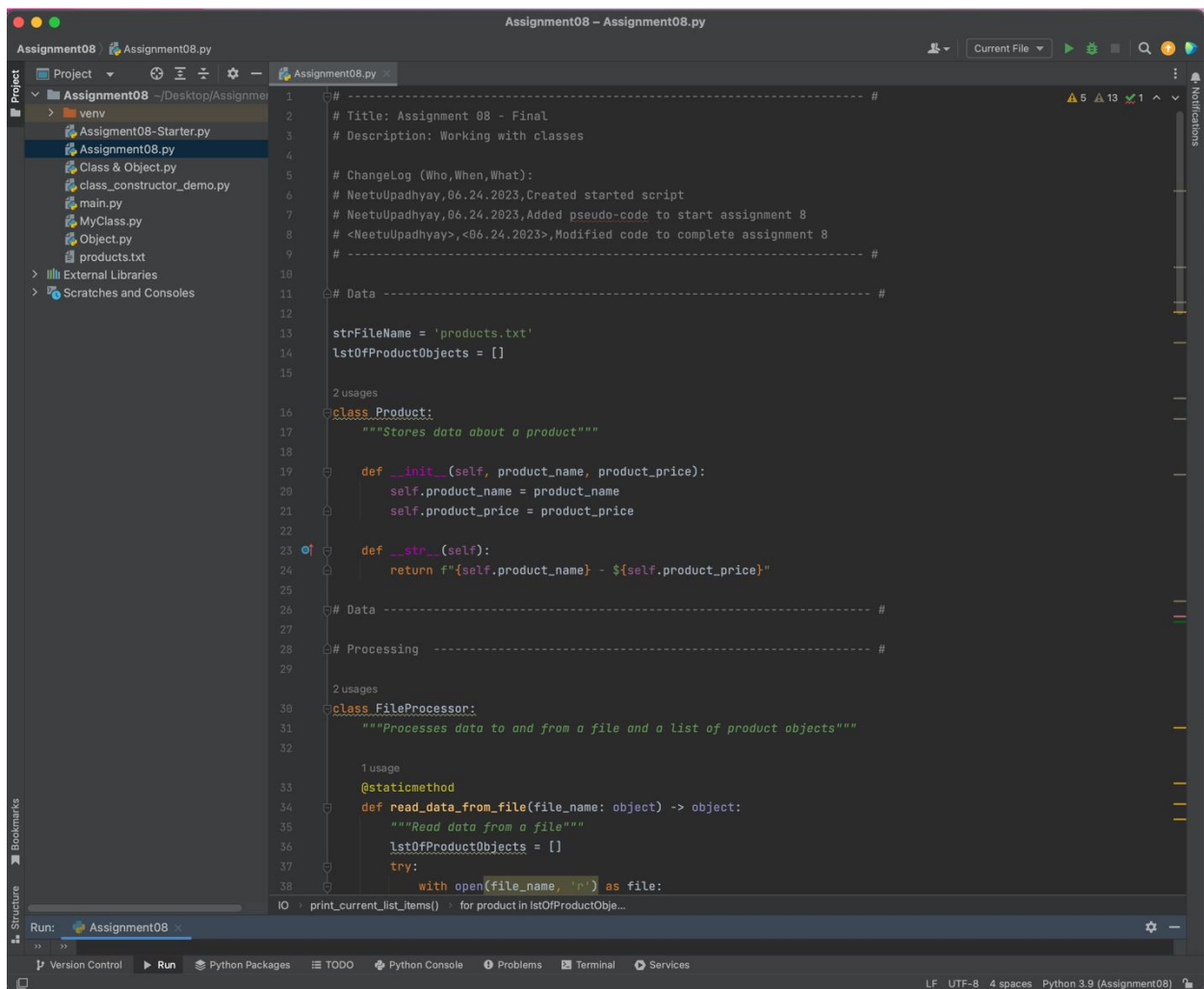


Figure 13

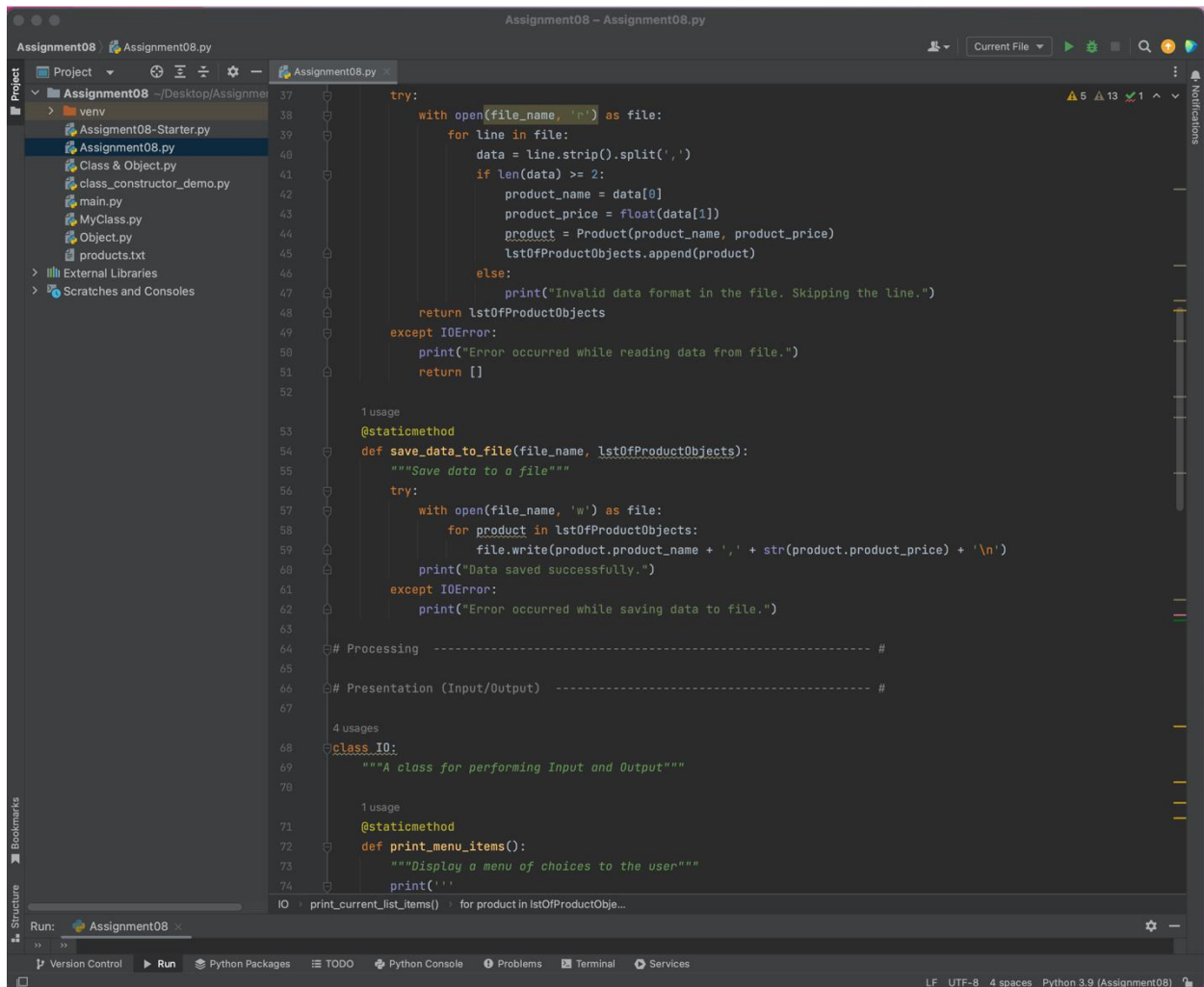


Figure.14



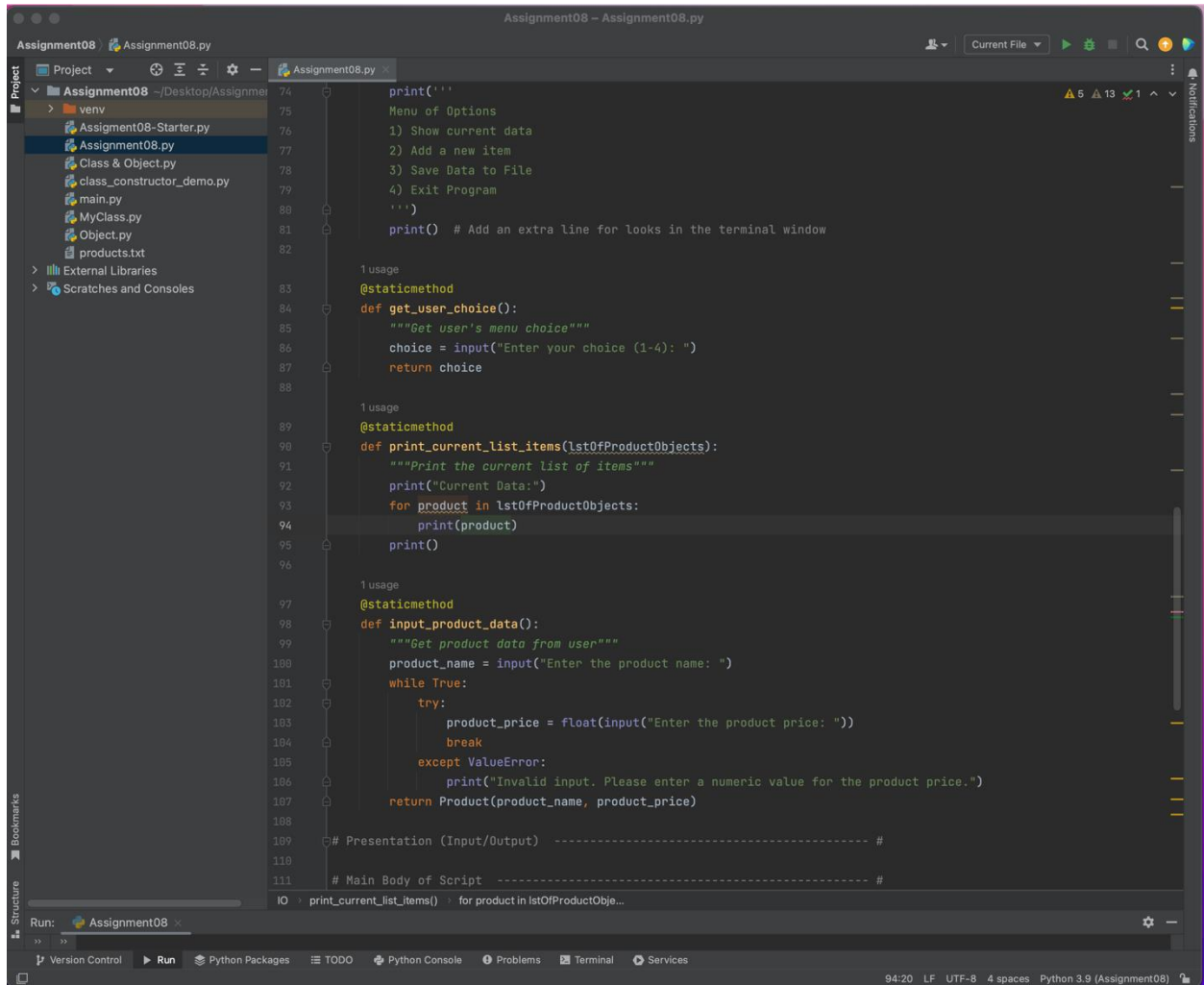


Figure 15

Figure.16

```
196         print("Invalid input. Please enter a numeric value for the product price.")
197         return Product(product_name, product_price)
198
199 # Presentation (Input/Output) ----- #
200
201 # Main Body of Script ----- #
202
203 # Load data from file into a list of product objects when script starts
204 lstOfProductObjects = FileProcessor.read_data_from_file(strFileName)
205
206 while True:
207     # Show user a menu of options
208     IO.print_menu_items()
209
210     # Get user's menu option choice
211     user_choice = IO.get_user_choice()
212
213     if user_choice == '1':
214         # Show user current data in the list of product objects
215         IO.print_current_list_items(lstOfProductObjects)
216     elif user_choice == '2':
217         # Let user add data to the list of product objects
218         product = IO.input_product_data()
219         lstOfProductObjects.append(product)
220     elif user_choice == '3':
221         # Save current data to file and exit program
222         FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)
223         break
224     elif user_choice == '4':
225         # Exit program
226         break
227     else:
228         print("Invalid choice. Please try again.")
229         continue
230
231 # Main Body of Script ----- #
```

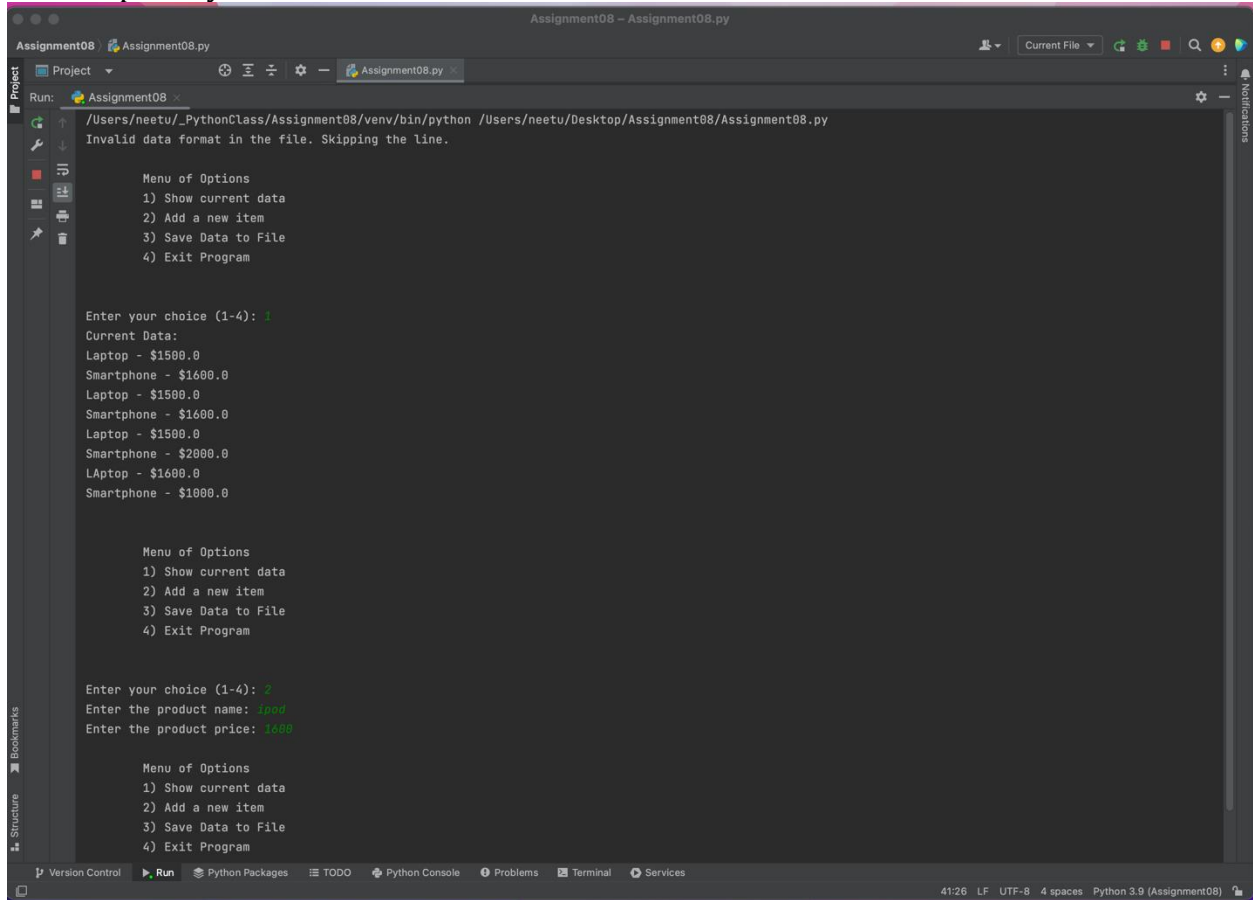
IO > print\_current\_list\_items() > for product in lstOfProductObj...

Run: Assignment08

Version Control Run Python Packages TODO Python Console Problems Terminal Services

94:20 LF UTF-8 4 spaces Python 3.9 (Assignment08)

## Run script in Pycharm



```
Assignment08 - Assignment08.py
Run: Assignment08
/Users/neetu/.PythonClass/Assignment08/venv/bin/python /Users/neetu/Desktop/Assignment08/Assignment08.py
Invalid data format in the file. Skipping the line.

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

Enter your choice (1-4): 1
Current Data:
Laptop - $1500.0
Smartphone - $1600.0
Laptop - $1500.0
Smartphone - $1600.0
Laptop - $1500.0
Smartphone - $2000.0
Laptop - $1600.0
Smartphone - $1000.0

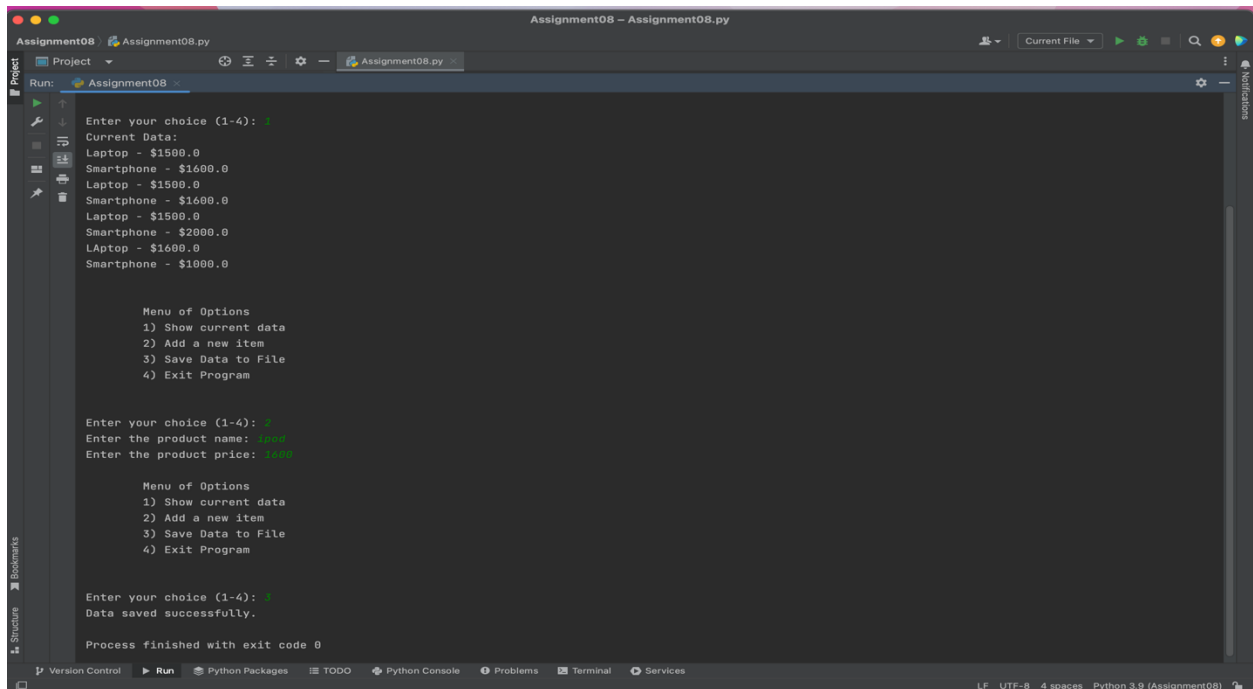
Menu of Options
1) Show current data
2) Add a new item
3) Save Data to File
4) Exit Program

Enter your choice (1-4): 2
Enter the product name: ipod
Enter the product price: 1600

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

41:26 LF UTF-8 4 spaces Python 3.9 (Assignment08)

Figure.17



```
Assignment08 - Assignment08.py
Run: Assignment08
Enter your choice (1-4): 1
Current Data:
Laptop - $1500.0
Smartphone - $1600.0
Laptop - $1500.0
Smartphone - $1600.0
Laptop - $1500.0
Smartphone - $2000.0
Laptop - $1600.0
Smartphone - $1000.0

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

Enter your choice (1-4): 2
Enter the product name: ipod
Enter the product price: 1600

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

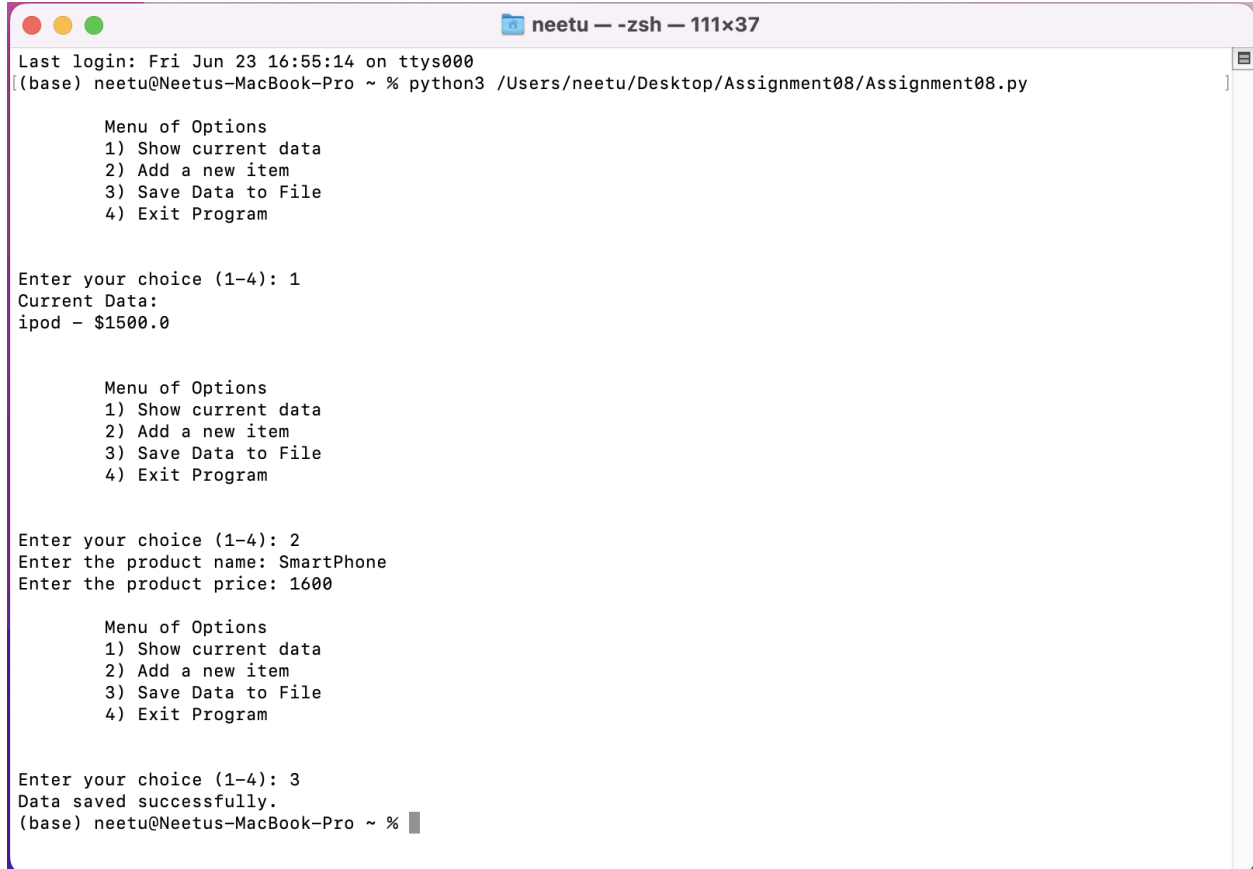
Enter your choice (1-4): 3
Data saved successfully.

Process finished with exit code 0
```

LF UTF-8 4 spaces Python 3.9 (Assignment08)

Figure.18

## Run script in Terminal



```
neetu — zsh — 111x37
Last login: Fri Jun 23 16:55:14 on ttys000
[(base) neetu@Neetus-MacBook-Pro ~ % python3 /Users/neetu/Desktop/Assignment08/Assignment08.py

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

Enter your choice (1-4): 1
Current Data:
ipod - $1500.0

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

Enter your choice (1-4): 2
Enter the product name: SmartPhone
Enter the product price: 1600

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

Enter your choice (1-4): 3
Data saved successfully.
(base) neetu@Neetus-MacBook-Pro ~ %
```

Figure.19

### **Summary:**

In the INTRO TO PROGRAMMING (PYTHON) ASSIGNMENT 08, we covered various topics related to Python programming. Here's a brief overview of what we did:

We explored custom classes in Python, which serve as blueprints for creating objects. The summary provides a simple explanation of custom classes, covering key components such as the constructor, class attributes, and instance attributes. It also discusses methods, the "self" keyword, and the @staticmethod decorator. The summary clarifies the distinctions between fields, attributes, property functions, properties, and methods. Overall, it offers a concise overview of custom classes in Python and their role in creating objects with shared characteristics.