Exercise 1.5: Object-Oriented Programming in Python

Reflection Questions :

- *In your own words, what is object-oriented programming? What are the benefits of OOP?*
  - **A:** Everything in Python is an object, and all objects can contain their own respective attributes. Assigning these attributes as classes gives them a syntax reference for them to be able to be interacted with throughout your code, as well as set a standard for this class designation in additional attributes that are intended to serve the same purpose and/or behave the same way and/or corresponding attributes and objects elsewhere.

- *What are objects and classes in Python? Come up with a real-world example to illustrate how objects and classes work.*
  - **A:** An object is virtually anything that exists in Python: the items that the code will interact. Classes can act as templates for assigning rules/behavior to a specific attribute which designates what it will do and/or what will define it. Once a class is defined, objects can be easily made that will inherit the attributes defined by the class. An example of this could be made using Cars. We could create a class called "Car" that has attributes like "Brand" and "model":
    `
    ```
    class Car: .
      Def __init__(self, brand, model):
        self.brand = brand
        self.model = model
    ```
    `
    Then, we can create any number of objects that would inherit this model for defining it:
    `
    ```
    Sean_Car = Car('Honda', 'Accord')
    Ashley_Car = Car{'Jeep', 'Compass')
    ```
    `

- *In your own words, write brief explanations of the following OOP concepts; 100 to 200 words per method is fine.*

Method	**Description**
Inheritance	Inheritance is an object-oriented programming concept that allows a class (the subclass or inherited class) to inherit attributes and behaviors from another class (the parent or base class). This allows it to be reused and promotes a hierarchical organization of code. The subclass can inherit methods and attributes from the parent class, and it can also extend or override them to define its behavior. This establishes a relationship between the classes, where an instance of the subclass is also considered an instance of the parent class. Inheritance promotes a modular approach to designing and organizing code, making it easier to maintain, extend, and repeat in large projects.
Polymorphism	Polymorphism is a concept where objects of different names/types can be treated as objects of a common names/type. It allows a single interface to represent different sets of data, enabling flexibility and adaptability in coding methods. It can enhance code reusability and promote more flexible coding structures, allowing for code that can work with objects of different types in a consistent manner. Different classes can be made to employ the same one data type, despite them being applied to different things, and/or returning different results, and/or serving different purposes. The creativity and varying coding preferences of various developers/teams can be expressed due to polymorphing facilitating the

	countless ways code could be written
Operator Overloading	“Operator overloading” refers to the ability to define custom behavior for built-in operators (that would otherwise default to a given scenario, or return a ‘TypeError’) when applied to the objects of classes. By implementing special methods in a class, you can specify how instances of that class should respond to common operators such as `+`, `-`, `*`, and others. For example, defining the `__add__` method in a class enables the use of the `+` operator with instances of that class, allowing customization of addition behavior. This feature allows you to make the classes more complex, customized, expressive, and more adaptive to specific use cases. Operator overloading enhances code readability and minimizes code by enabling objects to behave in a way that is closer to their real-world counterparts (more intuitive to the user).