Exercise 1.5: Object-Oriented Programming in Python

Reflection Questions

1. In your own words, what is object-oriented programming? What are the benefits of OOP?

OOP is a paradigm that structures code into objects. Some of the benefits are being able to reduce the amount of code that a developer needs to create. They can define a class, and reuse the class for each instance of an object that they need. With inheritance, you can start with an existing class, and change the relevant properties you need, to create another class.

2. What are objects and classes in Python? Come up with a real-world example to illustrate how objects and classes work.

Classes are the blueprint that defines the behavior of each object that is created from the class.

In a real world example, a class could be “Car”. Each car will have a different color, year, model, mileage, etc. All of those different details will be the attributes of the object that was created from the class “car”.

3. 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 the ability to pass down behaviors from a parent class to a child class(also called superclass and subclass). In our car example from earlier, the superclass could be Vehicle, if we wanted some subclasses for Car, Truck, Semi, Boat, Plane, etc. The subclasses could share some of the attributes of the superclass vehicle. For example, every type of vehicle might have material, top speed, year built, while just the Plane subclass could have attributes for landing gear, wings, things that other subclasses wouldn't need.
Polymorphism	Polymorphism allows for the same method to be executed according to the specific class that method is called upon. In our vehicle example, if we have an accelerate method across all our different vehicle classes, that acceleration can be laid out differently for a Plane than it might be for a Car. For both the Plane and the Car classes, we would define how acceleration works for that vehicle, but we don't need "accelerate_car" & "accelerate_plane" commands, we can just use "accelerate" and it will work for each class how it should.

Operator Overloading	Operator Overloading allows the developer to define how common methods like add and subtract, equals and not equals, etc should be handled with these unique objects. For example, we could define that if we want to ask if one vehicle == another, we could have them compare just the make and model. If we want to compare them using < or >, we could define that the manufacture year should be compared, or the miles, or whatever we pick.