Assignment 2

# Q1

The relationship between classes and modules in Python is that a module is a file containing Python code, and a class is a construct within that code. A module can contain multiple classes, as well as other objects such as functions, variables, and constants. Classes provide a way to define and create objects with specific properties and behaviours, while modules provide a way to organise and group related code.

# Q2

To make an instance of a class, we use the \_\_init\_\_ method of the class. The \_\_init\_\_ method takes a self argument, which is the object that is being created. The self argument can be used to access the object's attributes and methods.

To create a class in Python, we use the class keyword. The class keyword is followed by the name of the class, and then the body of the class. The body of the class can contain attributes, methods, and other class definitions.

# Q3

Class attributes are created within the class definition, outside of any methods. They are defined directly beneath the class name and are shared by all instances of the class. Class attributes are accessed using the class name itself or through an instance of the class.

# Q4

Instance attributes are created within the methods of a class, typically within the \_\_init\_\_ method. Each instance of a class can have its own set of instance attributes, which are specific to that instance. Instance attributes are defined using the self parameter, which refers to the instance itself.

# Q5

In Python, the term "self" is a convention used to refer to the instance of a class. It is the first parameter of instance methods within a class. By convention, it is named "self," but we can choose any valid variable name. The "self" parameter allows instance methods to access and operate on the instance's attributes and methods. It acts as a reference to the instance itself, enabling the instance to refer to its own data and perform actions on itself.

# Q6

Python classes handle operator overloading through special methods or dunder methods (double underscore methods). These methods define the behaviour of operators when applied to instances of the class. For example, the \_\_add\_\_ method defines the behaviour of the addition operator (+), the \_\_lt\_\_ method defines the behaviour of the less than operator (<), and so on.

# Q7

Operator overloading should be considered when we want to define custom behaviour for operators in our class. If we want instances of wer class to support common mathematical or comparison operations, we can define the appropriate special methods to enable operator overloading. This allows we to write code that is more expressive and intuitive, as instances of wer class can be used with operators just like built-in types.

# Q8

The most popular form of operator overloading in Python is arithmetic operator overloading. This includes defining the behaviour of operators such as +, -, \*, /, %, and others for instances of our class. By implementing the corresponding special methods (\_\_add\_\_, \_\_sub\_\_, \_\_mul\_\_, \_\_div\_\_, \_\_mod\_\_, etc.), we can customise how instances of our class behave when these operators are used.

# Q9

Classes and objects: Understanding the concept of classes and objects is fundamental to object-oriented programming. Classes define the structure and behaviour of objects, while objects are instances of classes that hold specific data and can perform actions through methods.

Inheritance and polymorphism: Inheritance allows classes to inherit attributes and behaviours from other classes, forming a hierarchical relationship. Polymorphism enables objects of different classes to be used interchangeably, as long as they share a common interface or base class. These concepts promote code reusability and provide flexibility in designing and organising code in object-oriented programs.