ASSIGNMENT 19

# Q1

The relationship between a class and its instances is a one-to-many partnership. A class is a blueprint or template that defines the structure and behaviour of objects, while an instance is a specific object created from that class. Multiple instances can be created from the same class, each having its own unique state and behaviour.

# Q2

Instances hold specific data or state that is unique to each object. These can include instance variables, which are properties or attributes specific to an instance. Instance variables can hold different values for each instance of a class.

# Q3

A class stores knowledge in the form of class variables and methods. Class variables are shared among all instances of the class and hold data that is common to all objects. Methods, on the other hand, are functions defined within the class and can perform operations on instances of the class.

# Q4

A method is a function that is defined within a class and operates on instances of that class. It is associated with the class and can access the instance variables and class variables. The key difference between a method and a regular function is that a method is implicitly passed the instance as the first argument (usually named self) and can operate on the instance's data. Regular functions, on the other hand, do not have this implicit argument and are not associated with a specific class or instance.

# Q5

es, inheritance is supported in Python. Inheritance allows a class to inherit attributes and methods from a parent class, enabling code reuse and creating a hierarchical relationship between classes. The syntax for inheritance in Python is as follows:

class ChildClass(ParentClass):

# ChildClass inherits from ParentClass

# ChildClass can access attributes and methods of ParentClass

# Additional attributes and methods can be defined in ChildClass

# Q6

Python supports a certain level of encapsulation by using naming conventions and access modifiers. By convention, instance variables that should be treated as private are prefixed with an underscore (e.g., \_private\_variable). This indicates that they are intended to be used internally within the class and should not be accessed directly from outside the class.

# Q7

A class variable is defined within the class but outside any method, and it is shared among all instances of the class. It is typically used to store data that is common to all objects of the class.

An instance variable is defined within a method or the \_\_init\_\_ method of a class, and it is unique to each instance of the class. Each instance has its own separate copy of the instance variable, and its value can vary between different instances.

# Q8

The self parameter is typically included in a class's method definitions in Python. It represents the instance of the class on which the method is being called. By convention, it is the first parameter in the method definition and is named self, but we can technically use any name for it. Including self allows the method to access and manipulate the instance's attributes and call other instance methods.

# Q9

The \_\_add\_\_ method is called when the addition is performed with the object on the left side of the + operator. For example, obj1 + obj2 calls the \_\_add\_\_ method of obj1.

The \_\_radd\_\_ method is called when the addition is performed with the object on the right side of the + operator and the left operand does not support the addition operation. It allows the right-hand-side object to handle the addition. For example, if obj1 does not support addition but obj2 does, then obj1 + obj2 calls the \_\_radd\_\_ method of obj2.

# Q10

Reflection methods are used to introspect or modify an object's behaviour at runtime. They allow us to examine and manipulate attributes, methods, and other properties of an object dynamically. We typically need to use reflection methods when we want to perform operations or access information about an object that are not directly available through regular attribute or method access. However, not all situations require reflection methods, especially when we have direct access to the desired attributes or methods of an object.

# Q11

The \_\_iadd\_\_ method is called when the in-place addition (+=) operation is performed on an object. It is used to define the behaviour of the object when it is modified in-place by adding another object to it. The \_\_iadd\_\_ method updates the object itself rather than creating a new object, unlike the \_\_add\_\_ method.

# Q12

The \_\_init\_\_ method is not inherited by subclasses in Python. However, if a subclass defines its own \_\_init\_\_ method, it overrides the \_\_init\_\_ method of the parent class. If we need to customise the behaviour of the \_\_init\_\_ method within a subclass, we can define a new \_\_init\_\_ method in the subclass and call the \_\_init\_\_ method of the parent class using the super() function. This allows us to add additional initialization steps specific to the subclass while still leveraging the initialization logic of the parent class.