**Part 1: Theoretical Questions**

**Question 1**

Object-Oriented Programming is a style of programming that includes identification of classes of objects that are linked to the methods (functions) with which they are associated. It also includes ideas of inheritance of attributes and methods. Object-Oriented Programming allows for better organization of code and promotes reusability, scalability, and maintainability.

The four main pillars of Object-Oriented Programming include;

1. **Encapsulation**

**This is a technique of restricting a user from directly modifying the data members or variables of a class in order to maintain the integrity of the data. It helps hide the internal implementation of a class and exposes only the necessary parts to the outside world, such as through methods that control access to an object’s attributes. This separation between the internal working of a class**

**and how it is accessed or used by other parts of the program improves code security and maintainability in several ways.**

1. **Inheritance**

This is a technique of acquiring the properties of another class having features in common. It allows us to increase the reusability and reduce the duplication of code. It also ensures that common features are centralized in a single location, making the code cleaner and easier to maintain. Inheritance allows creating a hierarchy of classes, where specific classes can build upon more general ones.

1. **Polymorphism**

This means “many forms”, it is a feature that allows you to perform an action in multiple or different ways. There are two types of polymorphism and they include;

1. **Static polymorphism**

This refers to the ability of a program to resolve method calls at compile time. It is achieved through method overloading or operator overloading.

1. **Dynamic polymorphism**

This refers to the ability of a program to resolve method calls at runtime. It is primarily achieved through method overriding in inheritance and interfaces.

1. **Abstraction**

This is a technique of providing only the essential details to the user by hiding the unnecessary details of an entity. This helps in reducing the operational complexity at the user-end. Abstraction enables one to provide a simple interface to a user without asking for complicated details to perform an action.

**Question 2**

The purpose of a constructor in Python is to create and initialize an object of a class.

The \_\_init\_\_ method is a constructor in python that allows one to pass parameters to an object upon instantiation and use them to set initial values for the object’s attributes.

**Example**

class Dog:

def \_\_init\_\_(self, breed, country\_of\_origin):

# Assign breed and country of origin to the instance

self.breed = breed

self.country\_of\_origin = country\_of\_origin

# Creating an object of the Dog class

Dog1 = Dog("Golden Retriever", "Scotland")

# Accessing the object's attributes

print(Dog1.breed

print(Dog1.country\_of\_origin

**Question 3**

**Class variables**

They are also known as static variables, is declared at the class level using the static keyword. Class variables hold a single shared value across all objects of the class. These variables are not associated with any specific object instance and are accessible and shared by all instances of the class, maintaining the same value for every object.

**Instance variables**

These are classes of variable without the static modifier. It is not shared across all instances of the class. Each object created from the class can have its own unique value for the instance variable. These variables are specific to the individual object and are not connected to other instances, meaning the values they hold are completely independent across different objects.

**Example**

For instance, in a library, all books belong to the same library system (class variable), but each book has its own barcode (instance variable).

* **Class Variable**: The library system is the same for all books. If the library changes its system, it affects all books in the collection.
* **Instance Variable**: Each book has a unique barcode, identifying that specific book in the system.

So, the library system is shared by all books, while the barcode is unique to each one.

**Question 4**

**1. Class Method:**

A class method is bound to the class, not its instances, and can modify class-level variables. It’s marked with @classmethod and takes cls as the first argument.

**Example**: A car factory tracks the total number of cars produced, not the details of a specific car.

**2. Static Method:**

A static method is independent of both class and instance data. It’s marked with @staticmethod and behaves like a regular function within the class.

**Example**: A function to calculate the cost of producing a car based on general factors, without using class or instance data.

**3. Instance Method:**

An instance method works on specific instances of a class and can modify the instance's data. It uses self to access instance attributes.

**Example**: Tracking the mileage or color of an individual car.

**Question 5**

**Real world Scenario**: A school tracks the number of students in each course.

### Explanation:

In this case, class variables are ideal because:

* **Shared Count**: All courses (like Math and Science) can share a single enrollment count, making it easy to track the total number of students.
* **Simplified Updates**: When a new student enrolls in a course, the overall count can be updated without having to adjust each course separately.
* **Consistency**: Changes to the total enrollment are instantly reflected across all courses, keeping data accurate and up-to-date.

**Github link to the work**