Name: DHWANI GUPTA

Enrollment Number: EBEON0223761985

Object Oriented Programming (OOPs) Theory Questions

1. **What is OOPs? Explain its key features.**
   - **OOPs stands for Object-Oriented Programming**. OOPs focuses on the concept of objects that contain both data (attributes) and behaviors (methods) and how these objects interact with each other.

   - The key features of OOPs are as follows:

     i. **Encapsulation**: Encapsulation is the mechanism of hiding the internal details and data of an object and exposing only the necessary interfaces to interact with the object. It helps in achieving data abstraction and improves security and maintainability of the code.

     ii. **Inheritance**: Inheritance allows the creation of new classes (derived classes) based on existing classes (base classes). The derived classes inherit the properties and behaviors of the base classes, which promotes code reuse and allows the creation of a hierarchical class structure. It enables the implementation of "is-a" relationships between classes.

     iii. **Polymorphism**: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It provides the ability to use a single interface to represent different types of objects. Polymorphism is achieved through method overriding (runtime polymorphism) and method overloading (compile-time polymorphism), enabling flexibility and extensibility in the code.

     iv. **Abstraction**: Abstraction is the process of identifying the essential characteristics and behavior of an object and creating a simplified model of the object. It focuses on what an object does rather than how it does it. Abstraction allows the creation of abstract classes and interfaces, providing a blueprint for defining common attributes and methods for a group of objects.

     v. **Modularity**: Modularity refers to the concept of breaking down a complex problem into smaller, manageable modules or classes. Each class represents a specific functionality or behavior, making the code easier to understand, maintain, and debug. Modularity enhances code reusability and promotes better organization and collaboration among developers.

2. **What is a class, and how is it different from an object?**

- In Python, a class is a blueprint for creating objects (instances) that share similar properties and behaviors. A class consists of two main components: attributes and methods. Attributes are variables that store data specific to each instance of the class, while methods are functions that define the behaviors or actions that the objects can perform.

- Here are the differences between a class and an object:

Class:

i. A class is a definition or a specification of a data structure and the methods or functions that operate on that data.
ii. It defines the attributes (data members) and behaviors (methods) that objects of that class will have.
iii. It serves as a template or a blueprint for creating objects with similar properties and behaviors.
iv. A class can be considered as a user-defined data type.
v. It can be used to create multiple instances or objects.

Object:

i. An object is an instance or a realization of a class.
ii. It represents a specific occurrence of a class, with its own unique state and behavior.
iii. It is created using the blueprint provided by the class.
iv. An object can store data in its attributes and perform actions using its methods.
v. Objects can interact with each other by invoking methods and accessing each other's attributes.

3. **What is the purpose of an init method in a class, and how is it used?**

- __init__ is the python constructor. The purpose of the __init__ is to create constructor and it is used, if we want to call Parent constructor or attributes/Methods in the child class.
- It is automatically called when an object is created from the class, and it is used to initialize the attributes of the object.
- It is used as:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

# Creating an instance of the Car class
my_car = Car("Toyota", "Corolla", 2021)
```

- In this example, the Car class has an __init__ method that takes three parameters: make, model, and year. Inside the __init__ method, these parameters are used to initialize the attributes make, model, and year of the object.

- When we create an instance of the Car class using my_car = Car("Toyota", "Corolla", 2021), the __init__ method is automatically called with the provided arguments. The attributes make, model, and year of the my_car object are set to "Toyota", "Corolla", and 2021, respectively.

4. What is inheritance, and how is it implemented?

- Inheritance is a **feature** of object-oriented programming. It specifies that one object acquires all the properties and behaviors of parent object.
- By using inheritance one can define a new class with little or no change to existing class.
- The new class is known as **derived class or child class** and from which it inherits the properties is call as **base class or parent class**.
- It provides re-usability of codes.
- There are 6 types of inheritance.

i. **Single Inheritance**: Single inheritance involves a class inheriting properties and behaviors from a single base class. The derived class inherits all the members of the base class, and it can add its own unique members. This is the simplest form of inheritance.

ii. **Multiple Inheritance**: Multiple inheritance occurs when a class inherits from more than one base class. In this case, the derived class inherits the properties and behaviors of all the base classes. Multiple inheritance allows for code reuse from multiple sources, but it can lead to complexities and conflicts if not managed carefully.

iii. **Multilevel Inheritance**: Multilevel inheritance involves a class being derived from another derived class. It forms a hierarchical structure where each derived class serves as the base class for the next level. This type of inheritance allows for creating a depth of inheritance hierarchy.

iv. **Hierarchical Inheritance**: Hierarchical inheritance refers to a situation where multiple derived classes inherit from a single base class. It forms a tree-like structure with one base class and multiple derived classes branching out from it. Each derived class inherits the properties and behaviors of the base class while adding its own specific features.

v. **Hybrid (or Mixed) Inheritance**: Hybrid inheritance is a combination of more than one inheritance. It involves multiple base classes and derived classes forming a complex inheritance structure. This type of inheritance is used when a class needs to inherit from multiple classes with different functionalities.

vi. **Interface Inheritance**: Interface inheritance, also known as implementation inheritance or interface implementation, is a concept where a class implements an interface. An interface defines a set of method signatures that the implementing class must provide. It allows for achieving code abstraction, polymorphism, and loose coupling between objects.

- Inheritance is implemented by using the following syntax in most object-oriented programming languages, including Python:

```
class BaseClass:
  # Base class definition
class DerivedClass(BaseClass):
  # Derived class definition
```

5. What is polymorphism, and how is it implemented?

- Polymorphism is a combination of two words poly and morph. Poly means many and morph means forms. It defines that one task can be done performed in different ways.
- For example, "+" it can be used as an operator to add two numbers and concatenate two values in string as well.
- It can be implemented by inheritance and overriding method.
  a. **Inheritance**: Polymorphism is achieved through inheritance when a class (subclass) inherits the properties and behaviors of another class (superclass). The subclass can be treated as an object of the superclass, allowing the use of superclass methods and properties. However, the subclass can also have its own specific methods and properties that are not present in the superclass.

  b. **Method Overriding**: Polymorphism is also implemented through method overriding, which involves defining a method in the subclass that has the same signature (name and parameters) as a method in the superclass. By doing so, the subclass provides its own implementation of the method, which is invoked when the method is called on an object of the subclass. This allows different classes to have their own implementation of a common method, providing flexibility and customization.

6. What is encapsulation, and how is it achieved?

- Encapsulation is a feature of (OOPs) Object Oriented Programming.
- It is used to restrict access to methods and variables. In **encapsulation,** code and data are wrapped together within a single unit from being modified by accident.
- We need **encapsulation,**
  i. Encapsulation to provide well – defined, readable code.
  ii. Prevent Accidental Modification or Deletion
  iii. **Encapsulation provides security.**

- To achieve encapsulation in Python, the following techniques are commonly used:

1. **Access Specifier**: In Python, access Specifier are not explicitly defined as in some other programming languages like Java. However, there are conventions that developers follow to indicate the level of visibility of attributes and methods.

   A. Public Access Specifier: In python by default, everything is public. In Public Access Specifier, we can access everything i.e. we can access within the class as well as outside the class.
   i. Access from Own class.
   ii. Accessible from Object.
   iii. Access from Derived class/ child class.

   B. Private Access Specifier: As we know in in python, everything is public. To make something private, we use double underscore "_ _ "/ "__".
   i. Access from Own class.
   ii. Not Accessible from Object.
   iii. Not Accessible from Derived class/ child class.

   C. Protected Access Specifier: As we know in in python, everything is public. To make something private, we use underscore "_ ".
   i. Access from Own class.
   ii. Accessible from Object.
   iii. Not Accessible from Derived class/ child class.

   D. Getters and Setters: By providing getter and setter methods, **you can control access to the attributes of a class**. These methods allow external code to retrieve or modify the attribute values indirectly, enabling you to enforce any necessary validation or logic.

7. What is the difference between a private and a protected attribute or method in a class?

- The difference between a private and a protected attribute or method in a class is as follows:

Parameters	Private Access Specifier	Protected Access Specifier
Restriction	Private is accessible to own class only.	Protected is accessible to own class and derived class.
Sign/ Symbol	To make something private in the python, double underscore "_ _ "/ "__" is used.	To make something protected in the python, underscore "_ " is used.
Access from Own class.	Yes	Yes
Accessible from Object.	No	No
Accessible from Derived class/ child class.	No	Yes

8. What is method overriding, and how is it implemented?

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Python. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.
- For example, consider a superclass Animal with a method makeSound(). The Dog class is a subclass of Animal and can override the makeSound() method to provide its own implementation. The Dog class would define its own makeSound() method with the same signature as the Animal class, and when called on a Dog object, the Dog class's implementation would be executed instead of the Animal class's implementation.

9. How does Python support multiple inheritance, and what are its benefits and drawbacks?

- Python supports multiple inheritance, which means a class can inherit attributes and methods from multiple base classes. To implement multiple inheritance, the class definition includes multiple base classes separated by commas in the parentheses.

```
class BaseClass1:
    def method1(self):
        print("BaseClass1 method")

class BaseClass2:
    def method2(self):
        print("BaseClass2 method")

class DerivedClass(BaseClass1, BaseClass2):
    def method3(self):
        print("DerivedClass method")
```

- **Benefits of Multiple Inheritance:**

i. **Code Reusability**: Multiple inheritance allows a class to inherit attributes and methods from multiple base classes, promoting code reuse. It helps avoid duplicating code and encourages modular design.

ii. **Flexibility and Expressiveness**: Multiple inheritance provides the flexibility to combine different functionalities from multiple classes. It allows for creating complex relationships and implementing diverse behaviors in a single derived class.

iii. **Modeling Real-World Relationships**: Multiple inheritance is useful for modeling real-world relationships where a class may have characteristics or behaviors from multiple categories or sources.

- **Drawbacks of Multiple Inheritance:**

  i. **Complexity**: Multiple inheritance can introduce complexity, especially when dealing with conflicting method names or attribute naming clashes between the base classes. It requires careful design and management to handle potential conflicts.

  ii. **Diamond (or Deadly) Inheritance Problem**: The diamond inheritance problem occurs when a derived class inherits from two base classes, and both base classes inherit from a common base class. It can lead to ambiguity and confusion in method resolution. In Python, this issue is resolved using a method resolution order (MRO) algorithm called C3 linearization.

  iii. **Increased Coupling:** Multiple inheritance can increase the coupling between classes, making the code more tightly interconnected. Changes in one base class can have unintended effects on derived classes, leading to maintenance challenges.

10. What are abstract classes and methods in Python OOPs, and how are they used?

- Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes cannot be instantiated, and require subclasses to provide implementations for the abstract methods.
- Abstract classes and methods are useful for enforcing a consistent interface across multiple subclasses. They allow you to define a set of methods that must be implemented by any concrete subclass, ensuring that your code is robust and maintainable.

11. What is the difference between a static method and a class method, and when would you use each?

- In Python, both static methods and class methods are used to define methods that can be called without creating an instance of a class. However, they differ in their behavior and intended use.

Parameters	Static Method	Class Method
Definition	**A static method is a method that belongs to a class, but does not operate on any instance of the class. It can be called using the class name or an instance of the class, but it does not have access to the instance or class state. Static methods are defined using the @staticmethod decorator and do not take either the self or cls parameter.**	**A class method, on the other hand, is a method that operates on the class itself and has access to the class state. It is defined using the @classmethod decorator and takes the cls parameter as the first argument. A class method can be used to create alternative constructors for a class or to modify class-level state.**

Code	class MyClass:
	  @staticmethod
	  def my_static_method(arg):
	  print(f"This is a static method with arg={arg}")
	
	MyClass.my_static_method("hello")
		class MyClass:
		  count = 0
		
		  def __init__(self):
		  MyClass.count += 1
		
		  @classmethod
		  def get_instance_count(cls):
		  return cls.count
		
		my_instance1 = MyClass()
		my_instance2 = MyClass()
		print(MyClass.get_instance_count())
		# Output: 2

Use	Use a static method when you need to define a method that does not access the instance or class state.	Use a class method when you need to define a method that operates on the class itself and has access to the class state.

12. What are global, protected and private attributes?

i. **Global attributes:** *Global attributes are variables or data that can be accessed from any part of the program, including different modules or classes. They have a global scope and can be declared outside of any function or class.* Global attributes can be accessed and modified by any part of the program, which can make them susceptible to unintended changes and make code harder to maintain. It is generally recommended to use global attributes sparingly and only when necessary.

ii. **Protected attributes:** *Protected attributes are denoted by a single underscore (_) prefix in their name. They are intended to indicate that the attribute should be treated as "protected" or "internal" within the class or module, but they are still accessible from outside the class or module.*

iii. **Private attributes:** *Private attributes are denoted by a double underscore (__) prefix in their name. They are intended to indicate that the attribute should be treated as "private" or "internal" to the class and should not be accessed or modified from outside the class.* Private attributes are a way to encapsulate data and prevent direct access to it from other parts of the program.

13. What is the use of self in Python?

- Self-represent the current instance/ object of the class.
- Self can be used to access the variable and methods inside the same class.
- Self is the first parameter in the constructor.

14. Are access specifiers used in python?

- Yes, Access specifiers are used in the python. It is used to restrict the access of class variables and methods out side of class. This can be achieved by public, private and protected Keywords.

15. Is it possible to call parent class without its instance creation?

- Yes, it is possible. We can do by using static method.

16. How is an empty class created in python?

- In Python, you can create an empty class by using the pass statement inside the class definition. The pass statement acts as a placeholder that does nothing, allowing you to define an empty class without any attributes or methods**.**
- **Syntax : class EmptyClass:**
  pass

17. How will you check if a class is a child of another class?

- We can check if a class is a child of another class by using " issubclass()" .

18. What is docstring in Python?

- In Python, docstring is used to provide documentation or a description of a module, function, class, or method. Docstrings serve as a way to document code, providing information about the purpose, usage, parameters, return values, and any other relevant details of the object. They are an essential part of good programming practice.

19. Is Python Object-oriented or Functional Programming?

- Python supports both object oriented and functional programming as well.
  i. **Object-oriented programming (OOP)** is a programming paradigm that organizes code around objects, which encapsulate data (attributes) and behavior (methods) together. In Python, you can define classes, create objects (instances), and leverage concepts like inheritance, encapsulation, and polymorphism to build modular and reusable code using the principles of OOP.
  ii. **Functional programming (FP)**, on the other hand, is a programming paradigm that focuses on writing code using pure functions, which have no side effects and produce the same output for the same input. FP promotes immutability, higher-order functions, and functions as first-class citizens. Python supports functional programming features such as higher-order functions, lambda functions, map, filter, and reduce functions, as well as support for immutable data types.

20. What does an object() do?

- The object() function in Python creates a new object of the base class object. The object class is the ultimate base class for all other classes in Python. It serves as a default class from which all other classes inherit.
- The object() function returns an empty object. You cannot add new properties or methods to this object. This object is the base for all classes, it holds the built-in properties and methods which are default for all classes.

21. What is the purpose of the super function in inheritance, and how is it used?

- The purpose of super function in inheritance is used to give access to methods and properties of a parent or sibling class. The super() function returns an object that represents the parent class.
- In short, Whenever you want to call parent constructor or method, we use super function.

22. What is data abstraction ?

- Data abstraction is a fundamental concept in object-oriented programming that's allows to hide unnecessary details and only exposes the necessary details. And it can be achieved by abstract class where we only declare the method, and don't define it.