Object Oriented Programming

1. What is OOP? Why do we need to use OOPs?

Object-Oriented Programming (OOP) is a programming model based on objects, which combine data and methods. Key principles are:

1. Encapsulation: Bundling data and methods in one unit.

2. Abstraction: Hiding complexity by using simple interfaces.

3. Inheritance: Reusing features from existing classes.

4. Polymorphism: Using a unified interface for different data types.

OOP enhances code reusability and organization.

OOPs needs to be used for:

1. making programming clearer and problem-solving more concise
2. reusing code with the help of inheritance
3. reducing redundancy
4. encapsulation
5. data hiding
6. the division into subproblems
7. program flexibility using polymorphism

2. Difference between Procedural programming and OOPs?

2. What is multiple inheritance?

If one class shares the behavior and structure defined in another multiple class, it is called multiple inheritance.

3. Give an example of encapsulation.

The notion of data hiding is referred to as encapsulation. Protected and private members in C++ are examples. In a ‘Person’ class, encapsulation hides internal details like ‘age’ and ‘name’ from external access. Methods like ‘setAge()’ and ‘getName()’ provide controlled access to these attributes.

4. What is the difference between overloading and overriding?

Overloading is two or more methods having the same name but different parameters. It is solved during compile-time. Whereas, Overriding is an OOPs concept that allows sub-classes to have a specific implementation of a method already provided by its parent class. It is solved during runtime.

5. Define protected access modifier.

A protected access modifier is accessible by its own class and accessible by derived class but not accessible by the world.

6. What is the function of a super keyword?
The super keyword is used to forward a constructor's call to a constructor in the superclass. It invokes the overridden method that allows access to these methods and the superclass's hidden members.

7. What is compile time polymorphism?
When a polymorphic call is made, and the compiler knows which function is to be called; this is known as compile-time polymorphism. The features like function default arguments, overloading, and templates in C++ support compile-time polymorphism.

8. How can you call a base class method without creating an instance?
It is possible to call the base class without instantiation if it's a static method and some other subclass has inherited the base class.

9. Give a real-life example of data abstraction.
While driving a car, you know that on pressing the accelerator, the speed will increase. However, you do not know precisely how it happens. This is an example of data abstraction as the implementation details are concealed from the driver.

10. What is the purpose of the 'this' keyword?
To refer to the current object of a class, this keyword is used. It is used as a pointer that differentiates between the global object and the current object by referring to the current one.

11. Explain the concept of inheritance with a real-life example.
The parent class is a logical concept, such as a vehicle is a base class that defines the common properties shared by all vehicles. However, child classes are a more specific type of class such as truck, bus, car, etc. Inheritance allows subclasses to inherit common attributes of a vehicle and define specific attributes and methods to their own.

12. How is a structure different from a class?
A structure is a user-defined collection of variables having different data types. However, it is not possible to instantiate a structure or inherit from it. Thus, it's not an OOPs concept.

13. What is an abstract function?
An abstract function is a function declared only in the base class. It is redefined in the subclass as it does not contain any definition in the base class.

14. Name three operators that can't be overloaded.

1. "::" Scope resolution operator
2. ". *" Pointer to member operator
3. "." dot or Member access operator

15. How is encapsulation different from data abstraction?

Data abstraction refers to the ability to hide unwanted information. At the same time, encapsulation refers to hiding data as well as the method together.

16. Are there any limitations of inheritance? If yes, then what?

Yes. The limitations of inheritance are:

1. Increased execution effort and time
2. Tight coupling of parent and child class
3. Requires correct implementation
4. Requires jumping between different classes

17. Define virtual functions.

The functions that help achieve runtime polymorphism are a part of functions present in the parent class and overridden by a subclass.

18. List down the limitations of Object-Oriented programming.

- It requires intensive testing
- Not apt for minor problems
- It requires good planning
- It takes more time to solve problems
- Problems need to be thought in term of objects

19. What is the difference between a base class and a superclass?

The base class is the root class- the most generalized class. At the same time, the superclass is the immediate parent class from which the other class inherits.

20. What is the access modifier for methods inside an interface?

All the methods inside an interface are public by default, and no other modifier can be specified.

21. What is Object-Oriented Programming (OOP)?
OOP is a programming paradigm that organizes software design around objects and their interactions, emphasizing modularity and reusability.

22. What are the four principles of OOP?
Encapsulation, Abstraction, Inheritance, and Polymorphism.

23. What is encapsulation in Python?
Encapsulation refers to bundling the data (attributes) and methods (functions) that operate on the data into a single unit, i.e., a class. It helps in data hiding and prevents direct access to the data.

24. Explain inheritance in Python.
Inheritance is a mechanism by which a class can inherit properties and behaviors (attributes and methods) from another class. The class inheriting is called a subclass, and the class being inherited from is called a superclass.

25. What is the purpose of the `super()` function in Python?
The `super()` function is used to call methods of the superclass from the subclass, enabling access to inherited methods and properties.

26. How is polymorphism achieved in Python?
Polymorphism allows objects of different classes to be treated as objects of a common superclass. In Python, polymorphism is achieved through method overriding and method overloading.

27. What is method overriding in Python?
Method overriding occurs when a subclass defines a method that has the same name as a method in its superclass. The method in the subclass overrides the method in the superclass.

28. Explain the concept of method overloading in Python.
Method overloading in Python is achieved through default arguments or variable-length argument lists (`*args` and ` kwargs`). It allows a method to perform different tasks based on the number or type of parameters passed to it.

29. What is a class constructor in Python, and how is it defined?
A class constructor in Python is a special method called `__init__()` that is automatically invoked when a new instance of the class is created. It initializes the object's state.

30. How do you create a class in Python?
You create a class in Python using the `class` keyword followed by the class name. For example:

```
class MyClass:
  # Class definition
```

31. What is a class attribute in Python?
A class attribute is a variable that is shared by all instances of the class. It is defined within the class but outside of any class methods.

32. How do you access class attributes in Python?
Class attributes can be accessed using the dot notation, specifying the class name followed by the attribute name.

33. What is an instance attribute in Python?
An instance attribute is a variable that is unique to each instance of the class. It is defined inside the class constructor (`__init__()` method) using the `self` keyword.

34. Explain the `self` parameter in Python classes.
The `self` parameter refers to the instance of the class itself. It is used to access variables and methods of the current instance within the class.

35. How do you create an instance of a class in Python?
You create an instance of a class by calling the class name followed by parentheses. For example:

```
obj = MyClass()
```

36. What is a static method in Python?
A static method is a method that belongs to the class rather than the instances. It can be called on the class itself and does not have access to instance variables.

37. How do you define a static method in Python?
You define a static method using the `@staticmethod` decorator before the method definition. It does not require the `self` parameter.

38. What is a class method in Python?

A class method is a method that is bound to the class rather than the instances. It can access and modify class-level variables.

39. How do you define a class method in Python?
You define a class method using the `@classmethod` decorator before the method definition. It takes the `cls` parameter as its first argument, referring to the class itself.

40. Explain the concept of multiple inheritance in Python.
Multiple inheritance in Python occurs when a class inherits attributes and methods from more than one parent class. It enables a subclass to inherit from multiple superclasses.

41. What is method resolution order (MRO) in Python?
Method resolution order (MRO) is the order in which Python searches for methods and attributes in a class hierarchy. It follows a depth-first, left-to-right traversal of the class hierarchy.

42. How do you access superclass methods in Python?
You can access superclass methods using either the `super()` function or by directly referencing the superclass.

43. What is the purpose of the `__str__()` method in Python?
The `__str__()` method is used to return a string representation of an object. It is called by the `str()` function and by the `print()` function when printing the object.

44. Explain the concept of operator overloading in Python.
Operator overloading allows operators such as `+`, `-`, `*`, `/`, etc., to be overloaded to perform different operations based on the operands.

45. How do you overload operators in Python?
Operator overloading is achieved by defining special methods with predefined names, such as `__add__()`, `__sub__()`, `__mul__()`, etc., corresponding to the operators to be overloaded.

46. What is the purpose of the `__del__()` method in Python?
The `__del__()` method is called when an object is about to be destroyed. It can be used to perform cleanup actions before the object is deallocated from memory.

47. Explain the concept of data hiding in Python.

Data hiding is the principle of restricting access to certain attributes or methods of a class, typically by prefixing them with double underscores (`__`). It enforces encapsulation and prevents direct modification from outside the class.

48. What is a destructor in Python?
In Python, the destructor is the `__del__()` method. It is automatically called when an object is destroyed and can be used to release resources or perform cleanup actions.

49. How do you create a child class in Python?
To create a child class in Python, you define a new class that inherits from one or more parent classes.
For example:
```
class ChildClass(ParentClass):
    # Class definition
```

50. What is the purpose of the `@property` decorator in Python?
The `@property` decorator is used to define properties in Python classes. It allows defining getter, setter, and deleter methods for accessing and modifying attributes as if they were regular class variables.

51. Explain the concept of multiple inheritance and its implications in Python.
Multiple inheritance occurs when a class inherits from more than one parent class. In Python, this can lead to the diamond problem, where ambiguity arises if the same method or attribute is defined in multiple parent classes. Python resolves this using Method Resolution Order (MRO), which follows a specific hierarchy to determine the order in which methods are searched for and executed.

52. Discuss the benefits and drawbacks of using class-level attributes versus instance-level attributes in Python.
Class-level attributes are shared among all instances of a class and are accessed using the class name. They are useful for defining properties that should be consistent across all instances. However, modifying class-level attributes can affect all instances simultaneously, which may not always be desirable. Instance-level attributes, on the other hand, are unique to each instance and are defined within the constructor (`__init__()` method). They allow for greater flexibility and encapsulation but can lead to increased memory usage if each instance has its own copy of the attribute.

53. How does Python implement encapsulation, and why is it important in data science projects?
Encapsulation in Python is achieved by bundling data (attributes) and methods (functions) that operate on the data into a single unit, i.e., a class. It helps in data hiding and prevents direct access to the data, promoting modularity, reusability, and

maintainability. In data science projects, encapsulation is crucial for managing complex data structures, ensuring data integrity, and abstracting implementation details, thereby facilitating collaborative development and code maintenance.

54. Discuss the concept of polymorphism and provide examples of how it can be applied in data analysis tasks.
Polymorphism allows objects of different classes to be treated as objects of a common superclass. In Python, polymorphism is achieved through method overriding, where a subclass provides a specific implementation of a method defined in its superclass. For example, polymorphism can be applied in data analysis tasks by defining a common interface for different data preprocessing or modeling techniques (e.g., feature scaling, dimensionality reduction, classification algorithms) and implementing specific versions of these techniques for different data types or scenarios.

55. Explain the significance of magic methods (dunder methods) in Python classes and provide examples of their usage in data science applications.
Magic methods, also known as dunder methods, are special methods in Python surrounded by double underscores (e.g., `__init__()`, `__str__()`, `__add__()`, `__getitem__()`). They allow classes to emulate built-in behavior and enable features such as operator overloading, iteration, and customization of object representation. In data science applications, magic methods can be used to define custom data structures, implement specialized operations on data objects (e.g., custom indexing, slicing), and facilitate interoperability with existing Python libraries and frameworks.