**Q1. What is the purpose of Python's OOP?**

* Creating reusable code(inheritance)
* Modular programming
* Hiding internal details from user (Encapsulation)
* OOP models real-world entities as software objects that have some data associated with them and can perform certain functions.
* Restricting access to attributes to outside by using access modifiers

**Q2. Where does an inheritance search look for an attribute?**

* So, the search starts from the actual class
* Then to the parent class
* If the actual class has multiple parent classes, then the search starts from Parent class at the left to the right

The search stops right there if attribute is found or if the whole tree is traversed

**Q3. How do you distinguish between a class object and an instance object?**

* A class has multiple instance objects and stores data for each object in memory
* Class object is shared across all objects and is stored only once in memory

**Q4. What makes the first argument in a class’s method function special?**

* The first object in a method always refers to current instance

**Q5. What is the purpose of the init method?**

* To create an object and store the data of that object in memory

**Q6. What is the process for creating a class instance?**

* Declare class
* Create a \_\_init\_\_(self,args) method in class
* objName=className(arguments)

**Q7. What is the process for creating a class?**

* Declare class
* Create a \_\_init\_\_(self,args) method in class

**Q8. How would you define the superclasses of a class?**

* Class subclass(ParentClass)

**Q9. What is the relationship between classes and modules?**

* In python world, module is a python file (.py) inside a package. Package is a folder that has \_\_init\_\_.py in its root. It is a way to organize your codes physically (in files and folders). We can import the module and use the functions. A module may or may not contain classes
* Class is supposed to be a blueprint to create (many) instances of objects based on that blueprint. Moreover, classes can have sub-classes (inheritance).Therefore, if I need inheritance or (many) instantiations, I gather functions and variables under a class definition (methods and properties). It is a logical way to organize code.

**Q10. How do you make instances and classes?**

* To create classes

Class Name

* To create instances

Obj=className()

**Q11. Where and how should be class attributes created?**

* inside the class mostly on top , outside \_init\_()

**Q12. Where and how are instance attributes created?**

* Instance attributes are created in \_\_init\_\_ method

**Q13. What does the term "self" in a Python class mean?**

* Traditionally, it is used to refer the current instance of the class

**Q14. How does a Python class handle operator overloading?**

* Whenever you change the behavior of the **existing operator** through operator overloading, you have to redefine the special function that is invoked automatically when the operator is used with the objects.

|  |
| --- |
| * **class** A: * **def** \_\_init\_\_(self, a): * self.a **=** a * # adding two objects * **def** \_\_add\_\_(self, o): * **return** self.a **+** o.a * ob1 **=** A(1) * ob2 **=** A(2) * ob3 **=** A("Geeks") * ob4 **=** A("For") * print(ob1 **+** ob2) * print(ob3 **+** ob4) * # Actual working when Binary Operator is used. * **print**(A.\_\_add\_\_(ob1 , ob2)) * **print**(A.\_\_add\_\_(ob3,ob4)) * #And can also be Understand as : * print(ob1.\_\_add\_\_(ob2)) * **print**(ob3.\_\_add\_\_(ob4)) |

**Output**

3

GeeksFor

3

GeeksFor

3

GeeksFor

**Q15. When do you consider allowing operator overloading of your classes?**

* When we want user defined type to behave like built in types
* Overloading + operator in a user defined class to merge two objects in the form of list

**Q16. What is the most popular form of operator overloading?**

Addition + operatior

1. If used with numbers, it does addition
2. If used with strings, it does concatenation
3. If used with lists, it does merge

**Q17. What are the two most important concepts to grasp in order to comprehend Python OOP code?**

* Inheritance
* Polymorphism

**Q18. Describe three applications for exception processing.**

* When we try to access index not in array and to avoid abrupt rogram termination
* When we are trying to divide a number by 0

**Q19. What happens if you don't do something extra to treat an exception?**

* Program terminates abruptly
* Memory buffers are not cleared
* File pointers will be in open state
* Database connections are not closed

**Q20. What are your options for recovering from an exception in your script?**

* Use try-except-else-finally block to avoid abnormal termination
* Provide cleanup code in finally block

**Q21. Describe two methods for triggering exceptions in your script.**

* Raise exception
* Assert statement

**Q22. Identify two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists.**

* Finally block

**Q23. What is the purpose of the try statement?**

* Try statement used to handle exceptions gracefully and without interrupting our program abnormally

**Q24. What are the two most popular try statement variations?**

* Try-except
* Try-except-else :-  you can also use the else clause on the try-except block which must be present after all the except clauses. The code enters the else block only if the try clause does not raise an exception.
* Try—except-finally or try-except-else-finally

**Q25. What is the purpose of the raise statement?**

* Raise is used to tirgger an exception explicitly

Def reciprocal(n):

Try:

If n==0:

Raise Exception

Reci=1/n

Except exception as e:

Print€

**Q26. What does the assert statement do, and what other statement is it like?**

* Assert stmt raises an AssertionError ecepion if the condition is not satisfied
* Raise exceptionname is used if we want to trigger exception intentionally

**Q27. What is the purpose of the with/as argument, and what other statement is it like?**

* To pass arguments with variable length
* \*\* is for keyword based arguments
* def myFun(arg1, \*argv):
* print("First argument :", arg1)
* for arg in argv:
* print("Next argument through \*argv :", arg)

* myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')

|  |
| --- |
| **def** myFun(arg1, **\*\***kwargs):  **for** key, value **in** kwargs.items():          print("%s == %s" **%** (key, value))      # Driver code  myFun("Hi", first**=**'Geeks', mid**=**'for', last**=**'Geeks') |

**Q28. What are \*args, \*\*kwargs?**

**Q29. How can I pass optional or keyword parameters from one function to another?**

**Q30. What are Lambda Functions?**

* Anonymous,nameless,single line fuctions
* Will return one value

**Q31. Explain Inheritance in Python with an example?**

Python is a multi-paradigm programming language. It supports different programming approaches.

One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

* attributes
* behavior

Let's take an example:

A parrot is an object, as it has the following properties:

* name, age, color as attributes
* singing, dancing as behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

class Parrot:

# class attribute

species = "bird"

# instance attribute

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

# instantiate the Parrot class

blu = Parrot("Blu", 10)

woo = Parrot("Woo", 15)

# access the class attributes

print("Blu is a {}".format(blu.\_\_class\_\_.species))

print("Woo is also a {}".format(woo.\_\_class\_\_.species))

# access the instance attributes

print("{} is {} years old".format( blu.name, blu.age))

print("{} is {} years old".format( woo.name, woo.age))

**Q32. Suppose class C inherits from classes A and B as class C(A,B).Classes A and B both have their own versions of method func(). If we call func() from an object of class C, which version gets invoked?**

* Multiple inheritance in Pythonis dept-first and leftr to right
* Class A version of func() will be called

**Q33. Which methods/functions do we use to determine the type of instance and inheritance?**

isinstance() and issubclass()

The isinstance() method checks whether an object is an instance of a class whereas issubclass() method asks whether one class is a subclass of another class (or other classes).

class MyClass(object):

pass

class MySubClass(MyClass):

pass

print(isinstance(MySubClass, object))

print(issubclass(MySubClass, MyClass))

print(isinstance(MySubClass, MyClass))

**True**

**True**

**False**

**Q34.Explain the use of the 'nonlocal' keyword in Python.**

* Nonlocal keyword is used to access a variable of outer function in its nested function

def outer():

a=20

def inner():

nonlocal a

a=30

inner()

print(a)

**Output:- 30**

**Q35. What is the global keyword?**

* Global keyword is used to access global variables
* Global variables are those which are outside functions and which are not instance and class variables