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

**OOPS** make the python more object oriented and makes the program easy to understand as well as more efficient. Since the **Class** is sharable the code can be reused. The **Data Abstraction** keeps the data safe and secure. It will be used to define the class with the **attributes** and **methods**.

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

Inheritance search looks for an attribute firstly in the instance object, then in the class the instance was created from and in all the super classes from left to right as a default.

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

|  |  |
| --- | --- |
| **CLASS OBJECT** | **INHERITANCE OBJECT** |
| * Class objects represent the class itself. | * Instance objects represent individual instances of the class. |
| * class objects can have class-level attributes and methods that are shared among all instances of the class | * instance objects have their own set of attributes and methods that are independent of other instances of the same class. |

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

The calling process is automatic while the receiving process is not. This is the reason the first parameter of a function in class must be the object **self**.

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

**\_\_ init \_\_** is one of the reserved methods in Python. In object-oriented programming, it is known as a constructor. The \_\_init\_\_ method can be called when an object is created from the class, and access is required to initialize the attributes of the class.

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

To create instances of a class, we call the class using the class name and pass in whatever arguments its \_\_init\_\_ method accepts.

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

**class** can be created by using the **keyword** class and followed by the **class name.**

**class ClassName :**

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

A superclass is the class from which many subclasses can be created. The subclasses inherit the characteristics of a superclass. The superclass is also known as the parent class or base class.

**{super class}** |-----🡪CAR

**Subclasses**

VEHICLE ---------🡪 | -----🡪TRUCK

|-----🡪BIKE

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

A **class** is more of a unit, and a **module** is essentially a loose collection of stuff like functions, variables, or even classes. In a public module, classes in the project have access to the functions and variables of the module. We don't have to specify the module name to address one.

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

Declaring the variable outside the function and inside the class is called **class** variable and the declaring the code inside the function or a constructor is called **instance** variable.

class Marks:

    maths = 100     #Class variable

    def \_\_init\_\_(self):         #Instance Variable

        self.tamil = 96

        self.english = 98

obj1 = Marks()

obj2 = Marks()

obj1.tamil =93

Marks.maths=94

print("Tamil :",obj1.tamil)

print("English :",obj1.english)

print("Maths :",obj2.maths)

#O/P

Tamil : 93

English : 98

Maths : 94

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

To define a class attribute, you place it outside of the \_\_init\_\_() method. Use class\_name. class\_attribute or object\_name. class\_attribute to access the value of the class\_attribute .

class Student:

    name = "Krishna"

    course = "BIG DATA"

Student1 = Student()

print(Student1.name)

print(Student1.course)

# Krishna

# BIG DATA

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

Instance attributes are defined in the constructor. Defined directly inside a class. Defined inside a constructor using the self-parameter.

class Marks:

    maths = 100     #Class variable

    def \_\_init\_\_(self):         #Instance Variable

        self.tamil = 96

        self.english = 98

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

The self is used to represent the instance of the class. With this keyword, we can access the attributes and methods of the class in python.

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

 Python that allows the same operator to have different meaning according to the context is called **operator overloading**.

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

When one or both operands are of a user-defined class or structure type, operator overloading makes it easier to specify user-defined implementation for such operations. This makes user-defined types more similar to the basic primitive data types in terms of behavior.

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

The most popular form of operator overloading is the **Addition operator( “+”).**

**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.**

* Indentation error
* Arithmetic error
* Key error

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

If we don't do something extra to treat an exception, then the program stops immediately and the line that caused the exception will not get executed.

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

try…, except…., else…., finally

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

**Try**: Catches exceptions raised by Python or a program.

**Raise**: A custom exception that triggers an exception manually.

**Q22. Identify two methods for specifying actions to be executed at termination time, regardless of**

**whether or not an exception exists.**

If **no exception** occurs during the execution, the execution will reach the break statement and the while loop will be left. If an **exception** occurs, then the rest of the **try** block will be skipped and the **except** block will be executed.

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

The **try** **statement** allows you to define a block of code to be tested for errors while it is being executed

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

**Ans: *Try and except***

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

The **raise** keyword is used to **raise an exception**. You can define what kind of error to raise, and the text to print to the user.

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

It is used when we are **debugging** the code. It allows us to test if the condition is True, if not it causes the assertion error.

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

In Python, the **with** statement replaces a **try-catch** **block** with a concise shorthand. More importantly, it ensures closing the resources right after processing them. A common example of using the **with** **statement** is reading or writing to a file.

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

* **\*args** (Non-Keyword Arguments) **-** The special syntax *\*args* in function definitions in Python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

def myFun(arg1, \*argv):

    print("First argument :", arg1)

    for arg in argv:

        print("Next argument through \*argv :", arg)

myFun('Hello', "I'm", 'learning', 'BIG DATA')

# output:

# First argument : Hello

# Next argument through \*argv : I'm

# Next argument through \*argv : learning

# Next argument through \*argv : BIG DATA

* **\*\*kwargs** (Keyword Arguments) - The special syntax ***\*\*kwargs*** in function definitions in Python is used to pass a keyworded, variable-length argument list. We use the name ***kwargs*** with the **double** **star**. The reason is that the double star allows us to pass through keyword arguments.

def myFun(\*\*kwargs):

    for key, value in kwargs.items():

        print("%s == %s" % (key, value))

myFun(first='BIG', mid='DATA', last='ENGINEER')

#OUTPUT

first == BIG

mid == DATA

last == ENGINEER

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

To pass optional or keywords parameters from one function to another by collecting the arguments using the **\*** and **\*\*** specifiers in functions parameter list.

**Q30. What are Lambda Functions?**

**Lambada** functions are small anonymous functions. It can take any number of arguments but it can have only one expression.

x = lambda a, b: a \* b

print(x(5, 6))

# output : 30

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

Inheritance is the capability of one class to derive or inherit the properties from another class.

class Person(object):

  # Constructor

  def \_\_init\_\_(self, name, id):

    self.name = name

    self.id = id

  def Display(self):

    print(self.name, self.id)

emp = Person("Ash", 1204)

emp.Display()

# output 🡺 Ash 1204

**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?**

Both the characters in A and B gets invoked to class c.

class A:

    def feature1(self):

        print("Feature 1 is working")

    def feature2(self):

        print("Feature 2 is working")

class B:

    def feature3(self):

        print("Feature 3 is working")

    def feature4(self):

        print("Feature 4 is working")

class C(A,B):

    def feature5(self):

        print("Feature 5 is working")

b1= B()

c1 = C()

# print(b1.feature1())

print(c1.feature2())

print(c1.feature3())

print(c1.feature5())

# Output

Feature 2 is working

None

Feature 3 is working

None

Feature 5 is working

None

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

The isinstance() function checks if the object (first argument) is an instance or subclass of the [class](https://www.geeksforgeeks.org/object-oriented-programming-in-python-set-1-class-and-its-members/) info class (second argument).

**Syntax:** [isinstance](https://www.geeksforgeeks.org/python-isinstance-method/)(object, classinfo)

weight = isinstance(23.7, float)

print("Is a float ?:", weight)

num = isinstance(55, int)

print("Is an integer ?:", num)

string = isinstance("learn untill you die", str)

print("Is a string ?:", string)

# OUTPUT

Is a float ?: True

Is an integer ?: True

Is a string ?: True

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

The **nonlocal** keyword is used to work with variables **inside nested functions**, where the variable should not belong to the inner function. Use the keyword nonlocal to declare that the variable is not local.

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

A **global** keyword is a keyword that allows a user to **modify a variable outside the current scope**.  Global keyword is used inside a function only when we want to do assignments or when we want to change a variable. Global is not needed for printing and accessing.

def change():

    # using a global keyword

    global x

    # increment value of a by 5

    x = x + 5

    print("Value of x inside a function :", x)

change()

print("Value of x outside a function :", x)

# Output

Value of x inside a function : 20

Value of x outside a function : 20