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

Ans. In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

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

Ans. An inheritance search looks for an attribute first in the instance object, then in the class the instance was created from, then in all higher superclasses, progressing from left to right (by default). The search stops at the first place the attribute is found.

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

Ans. class object is like a blueprint for instance object but instance object is a concrete item in our code.

class object inherits the attributes of its parent classes.

* instance objects are new namespaces, thay start out empty but inherit object attributes that live in class objects.
* The first argument of class functions(self) references the instance object and assignments to attributes of self change data in the instance.
* instance object inherits the attributes of the class object from which it was created.

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

Ans.Classes are a kind of factory for creating multiple instances. Classes also support operator overloading methods, which instances inherit, and treat any functions nested in the class as methods for processing instances.

Q5. What is the purpose of the init method?

Ans. The \_\_init\_\_ method is the Python equivalent of the [C++ constructor](https://www.udacity.com/blog/2021/03/what-is-a-constructor-in-c.html?utm_source=rss&utm_medium=rss&utm_campaign=what-is-a-constructor-in-c) in an object-oriented approach. The \_\_init\_\_ function is called every time an object is created from a class. The \_\_init\_\_ method lets the class initialize the object’s attributes and serves no other purpose.

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

Ans.class MyClass:

x = 5

p1 = MyClass()

print(p1.x)

Q7. What is the process for creating a class?

Ans. class My class:

X= 5

print(My class)

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

Ans.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. In the above example, Vehicle is the Superclass and its subclasses are Car, Truck and Motorcycle.

Q9. What is the relationship between classes and modules?

Ans.Modules are collections of methods and constants. They cannot generate instances. Classes may generate instances (objects), and have per-instance state (instance variables).

Q10. How do you make instances and classes?

Ans.object

class MyClass:

x = 5

p1 = MyClass()

print(p1.x)

Class

class My class:

X= 5

print(My class)

Q11. Where and how should be class attributes created?

Ans. To define a class attribute, you place it outside of the \_\_init\_\_() method. For example, the following defines pi as a class attribute

Class Test:

x=10

def\_init\_(self):

self.x=20

test=Test()

print(test.x)

print(Test.x)

Q12. Where and how are instance attributes created?

Ans. class emp:

def \_\_init\_\_(self):

self.name = 'xyz'

self.salary = 4000

def show(self):

print(self.name)

print(self.salary)

e1 = emp()

print("Dictionary form :", vars(e1))

print(dir(e1))

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

Ans. The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

Q14. How does a Python class handle operator overloading?

Ans.[Python operators](https://www.programiz.com/python-programming/operators) work for built-in classes. But the same operator behaves differently with different types. For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.

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

Ans. Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because ‘+’ operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called *Operator Overloading*.

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

Ans. A very popular and convenient example is the Addition (+) operator. Just think how the '+' operator operates on two numbers and the same operator operates on two strings. It performs “Addition” on numbers whereas it performs “Concatenation” on strings.

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

Ans. In this article, we will elaborate on two key concepts of OOP which are inheritance and polymorphism.

Both inheritance and polymorphism are key ingredients for designing robust, flexible, and easy-to-maintain software.

Q18. Describe three applications for exception processing.

Ans. Try Statement

A try statement includes keyword try, followed by a colon (:) and a suite of code in which exceptions may occur. It has one or more clauses.

### catch Statement

Catch blocks take one argument at a time, which is the type of exception that it is likely to catch. These arguments may range from a specific type of exception which can be varied to a catch-all category of exceptions.

### Finally Statement

### Finally block always executes irrespective of an exception being thrown or not. The final keyword allows you to create a block of code that follows a try-catch block.

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

Ans. When an exception occurred, if you don't handle it, the program terminates abruptly and the code past the line that caused the exception will not get executed.

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

Ans. You can also provide a generic except clause, which handles any exception. After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception. The else-block is a good place for code that does not need the try: block's protection.

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

Ans. To avoid such a scenario, there are two methods to handle Python exceptions: Try – This method catches the exceptions raised by the program. Raise – Triggers an exception manually using custom exceptions.

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

Ans. Python uses try and except keywords to handle exceptions.

If the exception does occur, the program flow is transferred to the except: block. The statements in the except: block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message.

If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.

Q23. What is the purpose of the try statement?

Ans. 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.So far we’ve used a **try/except** and even a **try/except/except**, but this is only two-thirds of the story.There are two other optional segments to a try block: **else** and **finally**. Both of these optional blocks will come after the try and the **except**. Also, there’s nothing stopping you from using both **else** and **finally** in a single statement — but keep them in that order if you do.Let’s go through each individually and see how they extend the behavior of a simple try/except.

Q25. What is the purpose of the raise statement?

Ans. The RAISE statement stops normal execution of a PL/SQL block or subprogram and transfers control to an exception handler. RAISE statements can raise predefined exceptions, such as ZERO\_DIVIDE or NO\_DATA\_FOUND , or user-defined exceptions whose names you decide.

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

Ans. The assert keyword is used when debugging code. The assert keyword lets you test if a condition in your codereturnsTrue, if not, the program will raise an AssertionError.

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

Ans. In Python, with statement is used in exception handling to make the code cleaner and much more readable. It simplifies the management of common resources like file streams.

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

Ans. \*args specifies the number of non-keyworded arguments that can be passed and the operations that can be performed on the function in Python whereas \*\*kwargs is **a** variable number of keyword arguments that can be passed to a function that can perform dictionary operations.

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

Ans. To pass optional or keyword parameters from one function to another, collect the arguments using the \* and \*\* specifiers in the function’s parameter list But, at first, do know what are \*args and \*\*args in Python.

Q30. What are Lambda Functions?

Ans. A lambda function is an anonymous function (i.e., defined without a name) that can take any number of arguments but, unlike normal functions, evaluates and returns only one expression.

A lambda function in Python has the following syntax:

lambda parameters: expression

The anatomy of a lambda function includes three elements:

* The keyword lambda — an analog of def in normal functions
* The parameters — support passing positional and keyword  
  arguments, just like normal functions
* The body — the expression for given parameters being evaluated  
  with the lambda function

Q31. Explain Inheritance in Python with an example?

Ans. Inheritance allows us to define a class that inherits all the methods and properties from another class.Parent class is the class being inherited from, also called base class.Child class is the class that inherits from another class, also called derived class.

Example

class Person:

def \_\_init\_\_(self, f name, l name):

self.firstname = f name

self.last name = l name

def printname(self):

print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("prasuna", "kondapalli")

x.printname()

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?

Ans. When we inherit from multiple classes, and if their method name conflicts, then the first-class name takes precedence.

We take a,b,c classes. inherit the c(a,b) and if their method name conflicts, then the first-class a takes precedence.

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

Ans. Use isinstance() to check an instance's type: isinstance(obj, int) will be True only if obj.\_\_class\_\_ is int or some class derived from int .

* Use issubclass() to check class inheritance: issubclass(bool, int) is True since bool is a subclass of int .

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

Ans. 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?

Ans. The global keyword is used to create global variables from a no-global scope, e.g. inside a function.