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

Ans-  **provides a means of structuring programs so that properties and behaviors are bundled into individual objects**.

Reuse of code through inheritance.

Flexibility through polymorphism. ..

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

Ans- **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)**.

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

Ans- **A class is a blueprint which you use to create objects.** **An object is an instance of a class** - it's a concrete 'thing' that you made using a specific class. So, 'object' and 'instance' are the same thing, but the word 'instance' indicates the relationship of an object to its class.

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

Ans- The calling process is automatic while the receiving process is not (its explicit). This is the reason the first parameter of a function in class must be **the object itself**. Writing this parameter as self is merely a convention. It is not a keyword and has no special meaning in Python.

Q5. What is the purpose of the init method?

Ans- The \_\_init\_\_ method **lets the class initialize the object's attributes and serves no other purpose**. It is only used within classes.

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

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

Q7. What is the process for creating a class?

class Employee:

    def \_\_init\_\_(self,name,age):

        self.emp\_name=name

        self.emp\_age=age

    def displayEmp\_details(self):

        print("employee name: ",self.emp\_name,"age: ",self.emp\_age)

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

Ans- class Employee:

    def \_\_init\_\_(self,name,age):

        self.emp\_name=name

        self.emp\_age=age

    def displayEmp\_details(self):

        print("employee name: ",self.emp\_name,"age: ",self.emp\_age)

class Person(Employee):

    p1=Employee('prakhar', 24)

    p1.displayEmp\_details()

Q9. What is the relationship between classes and modules?

Ans- **a module in python is simply a way to organize the code, and it contains either python classes or just functions**.

Q10. How do you make instances and classes?

class Employee:

    def \_\_init\_\_(self,name,age):

        self.emp\_name=name

        self.emp\_age=age

Q11. Where and how should be class attributes created?

class Employee:

    def \_\_init\_\_(self,name,age):

        self.emp\_name=name

        self.emp\_age=age

name and age are class variable.

Q12. Where and how are instance attributes created?

Ans- class Employee:

    def \_\_init\_\_(self,name,age):

        self.emp\_name=name

        self.emp\_age=age

emp\_name,Emp\_age is instance attribute.

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

Ans-it acts as pointer point towards current object.

Q14. How does a Python class handle operator overloading?

Ans- The operator overloading in Python means provide extended meaning beyond their predefined operational meaning. Such as, we use the "+" operator for adding two integers as well as joining two strings or merging two lists. We can achieve this as the "+" operator is overloaded by the "int" class and "str" class.

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

Ans- Consider that we have two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary '+' operator it throws an error, because compiler don't know how to add two objects. So we define a method for an operator and that process is called operator overloading.

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

Ans-plus operator overloading.

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

Ans-inheritance and polymorphism.

Q18. Describe three applications for exception processing.

Ans-1-raised when sepcifed key is not found in dictionary.

2-raised when any integer is divided by 0

3-raised when identifier is not found in local or global namespace.

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

Ans-if you don’t handle it. the program terminates abruptly and 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.

1. **Try**– This method catches the exceptions raised by the program
2. **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-try and finally – Allows to specify termination or cleanup actions, irrespective of exceptions occur or not.

Q23. What is the purpose of the try statement?

Ans-try block lets you test the block of code for errors.

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

Ans-try/except

Try/except/except

Try/except/finally

Q25. What is the purpose of the raise statement?

Ans- 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.

x = -1  
  
if x < 0:  
  raise Exception("Sorry, no numbers below zero")

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 code returns True, if not, the program will raise an AssertionError.

You can write a message to be written if the code returns False,

x = "hello"  
  
#if condition returns False, AssertionError is raised:  
assert x == "goodbye", "x should be 'hello'"

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

Ans- The **with statement** in Python is used for resource management and exception handling. You’d most likely find it when working with file streams. For example, the statement ensures that the file stream process doesn’t block other processes if an exception is raised, but terminates properly.

The code block below shows the try-finally approach to file stream resource management.

file = open('file-path', 'w')   
try:   
    file.write('Lorem ipsum')   
finally:   
    file.close()

Normally, you’d want to use this method for writing to a file, but the with statement offers a cleaner approach:

with open('file-path', 'w') as file:   
    file.write('Lorem ipsum')

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

Ans- \*args--🡪Non-Key valued arguments??

def example\_nonkeyed\_args(\*argv):

    for param in argv:

        print(param)

example\_nonkeyed\_args('hello',1234,'how are you !!!')

\*\*kwargs-🡪Key valued type of arguments in python??

def example\_of\_kwargs(\*\*kwargs):

    # for k,v in kwargs.items():

    #     print(("key is: ",k,"values is : ",v))

    print("value of host: ",kwargs['host'])

    print("value of port: ",kwargs['port'])

    print("value of password: ",kwargs['pswd'])

example\_of\_kwargs(host='120.33.44.32',port=32,pswd='abcddef')

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

Ans- def func(a, b=1098):

    return a+b

print(func(2, 2))

# this 1 is represented as 'a' in the function and

# function uses the default value of b

print(func(1))

Q30. What are Lambda Functions?

Ans- also known as inline function or anonymous function.

#syntax= lambda arguments :expression

a=10

b=20

lambda\_max=lambda x,y:X if x>y else y

print(lambda\_max(a,b))

Q31. Explain Inheritance in Python with an example?

Ans-means inheriting all the behaviours and functionality from its parent class.

e.g.

class Person:

    def \_\_init\_\_(self,name):

        self.person\_name=name

    def displayName(self):

        print("name: ",self.person\_name)

    def isEmployed(self):

        print(self.person\_name,"is un-employed !!")

class Employee(Person):

    def isEmployed(self):

        print(self.person\_name,"is employed !!")

emp=Person("shashank")

emp.displayName()

emp.isEmployed()

emp1=Employee("Rahul")

emp1.displayName()

emp1.isEmployed()

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-it will go from left to right so version func() of class will be invoked.

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

Ans- isinstance() function.

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.

e.g

def myfunc1():  
  x = "John"  
  def myfunc2():  
    x = "hello"  
  myfunc2()  
  return x  
  
print(myfunc1())

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

def myfunction():  
  global x  
  x = "hello"  
  
#execute the function:  
myfunction()  
  
#x should now be global, and accessible in the global scope.  
print(x)