1Ans: In Python, 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 function that work on that together as a single unit as that no other part of the code can access this data.

2Ans: All of these objects are namespaces (packages of variables), and the inheritance search is simply a search of thetree from bottom to top looking for the lowest occurrence of an attribute name. Code implies the shape of such trees.

3Ans: Everything in python is an object such as integers, lists, dictionaries, functions and so on. Every object has a type and the object types are created using classes. Instance is an object that belongs to a class. For instance, list is a class in python.

4Ans: first argument in a class is self. Which is object, self is just like this keyword but its not a keyword in python. It describes about particular object.

5Ans: \_\_init\_\_ () is a special python method that runs when an object of a class is created.

\_\_init\_\_ () function is mostly used for assigning values to newly created objects.

\_\_init\_\_ () is a magic method which means it is called automatically by Python

\_\_init\_\_ () can also be invoked manually.

\_\_init\_\_ () also supports inheritance.

6Ans: myObject = ClassName()

where mObject is the name of the object, whereas ClassName is the name of the class. A class instance is often instantiated by passing parameters whose values are determined when the program is run. The values passed in as parameters are typically used by the functions in the class.

7Ans: class Employee:

   empCount = 0

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

      self.name = name

      self.salary = salary

      Employee.empCount += 1

   def displayCount(self):

   print("Total Employee : " , Employee.empCount)

   def displayEmployee(self):

      print("Name : ", self.name, ", Salary: ", self.salary)

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

emp1.displayEmployee()

emp2.displayEmployee()

print("Total Employee: " ,Employee.empCount)

Use keyword class, followed by a space, the name of the class, and a colon.

The variable empCount is a class variable whose value is shared among all instances of a this class. This can be accessed as Employee.empCount from inside the class or outside the class.

The first method \_\_init\_\_ is a special method, which is called class constructor or initialization method that Python calls when we create a new instance of this class.

You declare other class methods like normal functions with the exception that the first argument to each method is *self*. Python adds the self argument to the list for us; we do not need to include it when you call the methods.

8Ans: super function which allows us to access temporary object of the super class.

class Class():

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

print(x)

class SubClass(Class):

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

super().\_\_init\_\_(x)

x = [1, 2, 3, 4, 5]

a = SubClass(x)

output: [1,2,3,4,5]

9Ans: The difference between a class and a module in python is that a class is used to define a blueprint for a given object, whereas a module is used to reuse a given piece of code inside another program. A class can have its own instance, but a module cannot be instantiated. We use the ‘class’ keyword to define a class, whereas to use modules, we use the ‘import’ keyword. We can inherit a particular class and modify it using inheritance. But while using modules, it is simply a code containing variables, functions, and classes. Modules are files present inside a package, whereas a class is used to encapsulate data and functions together inside the same unit.

10Ans: To create instances of a class, you call the class using class name and pass in whatever arguments its \_\_init\_\_ method accepts.

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name

class Employee:

   empCount = 0

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

      self.name = name

      self.salary = salary

      Employee.empCount += 1

   def displayCount(self):

   print("Total Employee : " , Employee.empCount)

   def displayEmployee(self):

      print("Name : ", self.name, ", Salary: ", self.salary)

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

emp1.displayEmployee()

emp2.displayEmployee()

print("Total Employee: " ,Employee.empCount)

11Ans: Some attribute values are shared across all objects of a given class. Such attributes are associated with the class itself, rather than any individual instance of the class.

Class attributes are created by assignment statements in the suite of a class statement, outside of any method definition. class attributes are also called as class variables or static variables.

class Employee:

   empCount = 0

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

      self.name = name

      self.salary = salary

      Employee.empCount += 1

   def displayCount(self):

   print("Total Employee : " , Employee.empCount)

   def displayEmployee(self):

      print("Name : ", self.name, ", Salary: ", self.salary)

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

emp1.displayEmployee()

print("Total Employee: " ,Employee.empCount)

Here, in the above example empCount is the class attribute.

12Ans: Unlike class attributes, instance attributes are not shared by objects. Every object has its own copy of the instance attribute (In case of class attributes all object refer to single copy).

class Employee:

   empCount = 0

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

      self.name = name

      self.salary = salary

      Employee.empCount += 1

   def displayCount(self):

   print("Total Employee : " , Employee.empCount)

   def displayEmployee(self):

      print("Name : ", self.name, ", Salary: ", self.salary)

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

emp1.displayEmployee()

emp2.displayEmployee()

print("Total Employee: " ,Employee.empCount)

Here, in the above example name, salary are the instance attributes.

13Ans: We'll use **self** in classes to represent the instance of an object. We can create multiple of a class and each instance will have different values. And **self** helps us to get those property values within the class instance. We are defining the properties of a class as **self.[something].** So, whenever we create an instance of the class, the **self** will refer to a different instance from which we are accessing class properties or methods. python sends a **reference** to the instance by default while accessing it **methods** or And the **reference** is captured in **self**. So, for each instance the reference is different. And we will get the respective instance properties.

class Employee:

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

      self.name = name

      self.salary = salary

   def displayEmployee(self):

      print("Name : ", self.name, ", Salary: ", self.salary)

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

emp1.displayEmployee()

emp2.displayEmployee()

Here, in the above example self first refer to emp1 and then if refers to emp2. Self act as pointer.

14Ans: **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. 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. To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator, the magic method \_\_add\_\_ is automatically invoked in which the operation for + operator is defined.

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("Praveen")

ob4 **=** A("Nakkanaboina")

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

PraveenNakkanaboina

3

PraveenNakkanaboina

3

PraveenNakkanaboina

15Ans: 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. To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator, the magic method \_\_add\_\_ is automatically invoked in which the operation for + operator is defined. To avoid error, we’ll use operator overloading and to perform various operations across objects we’ll use operator overloading.

16Ans: The most popular operator overloading form is ‘+’(\_\_add\_\_(self,other)). where it can be used for the usual addition and also for combining two different strings.

17Ans: Inheritance and polymorphism both are principal concepts of object-oriented programming. These ideas help us to write code that is easily maintainable. Inheritance is a great way to sort out unnecessary, duplicative code. We can inherit a child class from the parent class partially or entirely. Python is very flexible with regard to inheritance. In this method, we can add new attributes and methods as well as modify the existing ones. On the other hand, polymorphism is also contributing to Python's flexibility as well. We can use an object with a particular type as if it belonged to a different kind.

18Ans: 1.Closing a file, we use finally block. In finally block we write the code to close the open files as the finally block execute 100%.

2. Exception Handling is the process of responding to unwanted events when a computer program runs. Exception Handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

3. An exception is a python object which represents an error. As with code comments, exceptions helps us to remind what the program expects. It clarifies the code and enhances readability.

19Ans: Exceptions are the unusual event that occurs during the execution of the program that interrupts the normal flow of the program. Generally, exceptions occur when the code written encounters a situation it cannot cope with. Whenever an exception is raised, the program stops the execution, and thus the further code is not executed. Therefore, an exception is a python object that represents a run-time error. An exception is a Python object that represents an error. If we don’t do something extra to treat an exception the code after the line where the exception occurs will not be executed.

20Ans: We can recover from the exception using try block, except statement.

A=5

try:

Result=A/0

except ZeroDivisionError:

print(“Error occurred because of division by 0”)

else:

print(result)

output: Error occurred because of division by 0

here, in the above case we recovered from ZeroDivisionError using try block and except statement.

21Ans: If we have some *suspicious* code that may raise an exception, we can defend our program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

A=5

try:

Result=A/0

except ZeroDivisionError:

print(“Error occurred because of division by 0”)

else:

print(result)

22Ans:

A=5

try:

Result=A/0

except ZeroDivisionError:

print(“Error occurred because of division by 0”)

else:

print(result)

finally:

print(“Doesn’t matter try-except but I will print myself”)

if the code encounters any exception then print statement inside except block will get print. Otherwise print statement in else will get print. Finally block always gets executed either exception is generated or not

23Ans: Try statement is used to handle these errors within our code in Python. The try block is used to check some code for errors i.e the code inside the try block will execute when there is no error in the program. Whereas the code inside the except block will execute whenever the program encounters some error in the preceding try block.

try:

# Some Code

except:

# Executed if error in the

# try block

A=5

try:

Result=A/0

except ZeroDivisionError:

print(“Error occurred because of division by 0”)

else:

print(result)

output: Error occurred because of division by 0

A=5

try:

Result=A/2

except ZeroDivisionError:

print(“Error occurred because of division by 0”)

else:

print(result)

output: 2.5

24Ans: try statement has one or more except clauses, as well as an optional else clause.

try

statement(s)

except [ expression [ , target]]:

statement(s):

[else

Statement(s)]

25Ans: The raise keyword is used to raise an exception. we 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")

26Ans: assert statement has a condition and if the condition is not satisfied the program will stop and give AssertionError.

assert statement can also have a condition and a optional error message. If the condition is not satisfied assert stops the program and gives AssertionError along with the error message.

Assert Statement without Error Message:

def avg(marks):

assert len(marks) != 0

return sum(marks)/len(marks)

mark1 = []

print("Average of mark1:",avg(mark1))

output: AssertionError

Assert Statement with Error Message:

def avg(marks):

assert len(marks) != 0,"List is empty."

return sum(marks)/len(marks)

mark1 = []

print("Average of mark1:",avg(mark1))

Output: AssertionError: List is empty.

27Ans: **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.

# 1) without using with statement

file **=** open('file\_path', 'w')

file.write('hello world !')

file.close()

# using with statement

with open('file\_path', 'w') as file:

    file.write('hello world !')

unlike the first two implementations, there is no need to call file.close() when using with statement. The with statement itself ensures proper acquisition and release of resources. An exception during the file.write() call in the first implementation can prevent the file from closing properly which may introduce several bugs in the code, i.e. many changes in files do not go into effect until the file is properly closed.

28Ans: \*args parameter lets the function accept any number of arguments. In this example, we will write an addition function that can accept any number of arguments and returns the addition of all these arguments.

def addition(\*args):

result = 0

for arg in args:

result += arg

return result

if \_\_name\_\_ == "\_\_main\_\_":

sum = addition(2, 5, 1, 9)

print(sum)

sum = addition(5)

print(sum)

output: 17

5

[\*\*kwargs](https://pythonexamples.org/python-kwargs/) can accept any number of named arguments.

def myFunction(\*args, \*\*kwargs):

print(args)

print(kwargs)

if \_\_name\_\_ == "\_\_main\_\_":

myFunction("hello", "mars", a = 24, b = 87, c = 3, d = 46)

output:

('hello', 'mars')

{'a': 24, 'b': 87, 'c': 3, 'd': 46}

29Ans: There are two main ways to pass optional parameters in python

* Without using keyword arguments.
* By using keyword arguments.

Without using keyword arguments:

* The order of parameters should be maintained i.e. the order in which parameters are defined in function should be maintained while calling the function.
* The values for the non-optional parameters should be passed otherwise it will throw an error.
* The value of the default arguments can be either passed or ignored.

# Here b is predefined and hence is optional.

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))

output: 4

1099

By using keyword arguments:

* In this case, we are not required to maintain the order of passing the values.
* There should be no difference between the passed and declared keyword names.

# Here string2 is the default string used

def fun2(string1, string2**=**"Praveen"):

    print(string1 **+** string2)

# This can be a way where no order is needed.

fun2(string2**=**'Praveen', string1**=**"Nakkanaboina")

# since we are not mentioning the non-default argument

# so it will give error.

fun2(string2**=**'Praveen')

30Ans: A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax:

lambdaarguments :expression

Example:

x = lambda a, b, c : a + b + c  
print(x(5, 6, 2))

31Ans: 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.

Class Person():

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

self.name=name

def displayName(self):

print(self.name)

def isEmployed(self):

print(self.name,” is unemployed!!”)

Class Employee(Person):

def isEmployed(self):

print(self.name,” is employed!!”)

emp=Employee(‘Praveen’)

emp.displayName()

emp.isEmploye()

output: Praveen

Praveen is Employed

32Ans: Python will always call the first thing that is found in the method resolution order. Since you specified the inheritance as A, B then A will be found first, so its method will be called.

class A():

def func(self):

print('in A')

class B():

def func(self):

print('in B')

class C(A, B):

pass

C().func() # prints 'in A'

33Ans: In Python, by using isinstance() and issubclass() we can determine the type of instance and inheritance. 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).

34Ans: **Python nonlocal keyword**is used to reference a variable in the nearest scope.

def foo():

    name **=** "python" # Our local variable

    def bar():

        nonlocal name          # Reference name in the upper scope

        name **=** 'OOPs' # Overwrite this variable

        print(name)

    # Calling inner function

    bar()

    # Printing local variable

    print(name)

foo()

output: OOPs

OOPs

35Ans: In Python, the global keyword allows us to modify the variable outside of the current scope.

It is used to create a global variable and make changes to the variable in a local context.

**Rules of global Keyword**

The basic rules for global keyword in Python are:

* When we create a variable inside a function, it is local by default.
* When we define a variable outside of a function, it is global by default. You don't have to use the global keyword.
* We use the global keyword to read and write a global variable inside a function.
* Use of the global keyword outside a function has no effect.

**Access and Modify Python Global Variable**

# global variable

c = 1

def add():

# increment c by 2

c = c + 2

print(c)

add()

output: Throws an error.

# global variable

c = 1

def add():

# use of global keyword

global c

# increment c by 2

c = c + 2

print(c)

add()

# Output: 3