## **Python OOP Assignment**

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 is Data Type,You use this type to create object.Instance is Logical but object is Physical means occupies some memory.We can create an instance for abstract class as well as for interface, but we cannot create anobject for those.Object is instance of class and instance means representative of class i.e object.Instance refers to Reference of an object.Object is actually pointing to memory address of that instance.You can’t pass instance over the layers but you can pass the object over the layersYou can’t store an instance but you can store an objectA single object can have more than one instance.Instance will have the both class definition and the object definition where as in object it will have only the object definition.

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 specialmeaning 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. "\_\_init\_\_" is a reseved method in python classes. It is called as a constructor in object oriented terminology. This method is called when an object is created from a class and it allows the class to initialize the attributes of the class.

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?

Ans:-A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.To understand the need for creating a class in [Python](https://www.geeksforgeeks.org/python-programming-language/) let’s consider an example, let’s say you wanted to track the number of dogs that may have different attributes like breed, and age. If a list is used, the first element could be the dog’s breed while the second element could represent its age. Let’s suppose there are 100 different dogs, then howwould you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it’s the exact need for classes.

**Syntax:**Class Definition

class ClassName:

# Statement

**Syntax:**Object Definition  
obj = ClassName()

print(obj.atrr)

Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

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:-Classes are blueprints that allow you to create instances with attributes and bound functionality. Classes support inheritance, metaclasses, and descriptors.

Modules can't do any of this, modules are essentially singleton instances of an internal module class, and all their globals are attributes on the module instance. You can manipulate those attributes as needed (add, remove and update), but take into account that these still form the global namespace for all code defined in that module.

Q10. How do you make instances and classes?

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

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

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

emp1.displayEmployee()

emp2.displayEmployee()

print "Total Employee %d" % Employee.empCount

## **Example**

Now, putting all the concepts together −

#!/usr/bin/python

classEmployee:

   'Common base class for all employees'

   empCount =0

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

      self.name = name

      self.salary = salary

      Employee.empCount +=1

   def displayCount(self):

   print"Total Employee %d"%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 %d"%Employee.empCount

## **Output**

When the above code is executed, it produces the following result −

Name : Zara ,Salary: 2000

Name : Manni ,Salary: 5000

Total Employee 2

Q11. Where and how should be class attributes created?

Ans:-A class attribute is shared by all instances of the class. 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.
* Use class attributes for storing class contants, track data across all instances, and setting default values for all instances of the class.

Q12. Where and how are instance attributes created?

Ans:-Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor.Defined inside a constructor using the self parameter. Accessed using object dot notation e.g. object.instance\_attribute

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

Ans:-self represents the instance of the class. By using the “self” we can access the attributes and methods of the class in python. It binds the attributes with the given arguments. The reason you need to use self. is because Python does not use the @ syntax to refer to instance attributes. Python decided to do methods in a way that makes the instance to which the method belongs be passed automatically, but not received automatically: the first parameter of methods is the instance the method is called on.

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:- Ensures that objects of a class behave consistently with built-in types and other user-defined types. Makes it simpler to write code, especially for complex data types. Allows for code reuse by implementing one operator method and using it for other operators.

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

**And:-** 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:-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. These concepts are best explained via examples. Let’s start with creating a simple class.

class Employee(): def \_\_init\_\_(self, emp\_id, salary):  
 self.emp\_id = emp\_id  
 self.salary = salary def give\_raise(self):  
 self.salary = self.salary \* 1.05

We have created a class called Employee. It has two data attributes which are employee id (emp\_id) and salary. We have also defined a method called give\_raise. It applies a 5-percent increase on the salary of an employee.

We can create an instance of the Employee class (i.e. an object with Employee type) and apply the give\_raise method as follows:

emp1 = Employee(1001, 56000)print(emp1.salary)  
56000emp1.give\_raise()print(emp1.salary)  
58800.0

OOP allows us to create a class based on another class. For instance, we can create the Manager class based on the Employee class.

class Manager(Employee):  
 pass

In this scenario, Manager is said to be a child class of the Employee class. The child class copies the attributes (both data and procedural) from the parent class. This concept is called **inheritance.**

It is important to note that inheritance does not mean copying a class. We can partially inherit from a parent (or base class). Python also allows for adding new attributes as well as modifying the existing ones. Thus, inheritance comes with a great deal of flexibility.

We can now create a manager object just like we create an employee object.

mgr1 = Manager(101, 75000)  
print(mgr1.salary)  
75000

A child class can have new attributes in addition to the ones inherited from the parent class. Furthermore, we have the option to modify or override the inherited attributes.

Let’s update the give\_raise method so that it applies a 10-percent increase for the managers.

class Manager(Employee): def give\_raise(self):  
 self.salary = self.salary \* 1.10mgr1 = Manager(101, 75000)  
print(mgr1.salary)  
75000mgr1.give\_raise()  
print(mgr1.salary)  
82500

We will create another child class of the Employee class. The Director class inherits the attributes from the Employee class and modifies the give\_raise method with a 20-percent increase.

class Director(Employee): def give\_raise(self):  
 self.salary = self.salary \* 1.20

We now have three different class and they all have a give\_raise method. Although the name of the method is the same, it behaves differently for different type of objects. This is an example of **polymorphism**.

Polymorphism allows for leveraging the same interface for different underlying operations. Regarding our example of manager and director objects, we can use them as they were an instance of the employee class.

Let’s see polymorphism in action. We will define a function that applies raise to a list of employees.

def bulk\_raise(list\_of\_emps):  
 for emp in list\_of\_emps:  
 emp.give\_raise()

The bulk\_raise function takes a list of employees and apply the give\_raise function to each object in the list. The next step is to create a list of employees of different types.

emp1 = Employee(101, 45000)  
emp2 = Manager(103, 60000)  
emp3 = Director(105, 70000)list\_of\_emps = [emp1, emp2, emp3]

Our list contains one employee, one manager, and one director objects. We can now call the bulk\_raise function.

bulk\_raise(list\_of\_emps)print(emp1.salary)  
47250.0print(emp2.salary)  
66000.0print(emp3.salary)  
84000.0

Although each object in the list has a different type, we do not have to explicitly state it in our function. Python knows which give\_raise method to apply.

As we see in the examples, polymorphism is accomplished using inheritance. Subclasses (or child classes) make use of the methods from the parent class to establish polymorphism.

## Conclusion

Both inheritance and polymorphism are fundamental concepts of object oriented programming. These concepts help us to create code that can be extended and easily maintainable.

Inheritance is a great way to eliminate unnecessary repetitive code. A child class can inherit from the parent class partially or entirely. Python is quite flexible with regards to inheritance. We can add new attributes and methods as well as modify the existing ones.

Polymorphism contributes to Python’s flexibility as well. An object with a particular type can be used as if it belonged to a different type. We have seen an example of it with the give\_raise method.

Q18. Describe three applications for exception processing.

Ans:-Python uses try and except keywords to handle exceptions. Both keywords are followed by indented blocks.

Syntax:

try :

#statements in try block

except :

#executed when error in try block

The try: block contains one or more statements which are likely to encounter an exception. If the statements in this block are executed without an exception, the subsequent except: block is skipped.

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.

You can specify the type of exception after the except keyword. The subsequent block will be executed only if the specified exception occurs. There may be multiple except clauses with different exception types in a single try block. If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.

The rest of the statements after the except block will continue to be executed, regardless if the exception is encountered or not

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

Ans:-If you don't handle exceptions 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:- Firing** triggers tell tags when to fire. A tag will fire when the conditions for any one of its triggers are met. For example, if a tag has two triggers, one for all pages and another for only one specific page, the tag will always fire on all pages.

Triggers are set at the bottom of any tag configuration page. When a tag is created, you will be prompted to add a trigger to the tag configuration before the tag can be saved.

A **trigger exception** (also known as a "*blocking trigger*") is a kind of trigger that can be used to block another trigger's ability to fire under certain conditions. For example, if a tag has a trigger to fire on *all pages* and a trigger exception that is set to "*Page URL equals thankyou.html*", the tag will never fire on the page "*thankyou.html*".

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

Ans:- In Python, there are two methods for specifying actions to be executed at termination time, regardless of whether or not an exception exists:

• Using a finally block: Python has a finally block that allows you to specify actions to be executed at termination time, regardless of whether an exception occurs or not. The code inside a finally block is guaranteed to be executed, even if an exception is thrown, or if the program terminates early due to a return statement or an uncaught exception.

• Using a with statement: The with statement in Python provides a convenient way to specify actions to be executed at termination time. The with statement is used in conjunction with a context manager, which is an object that provides a \_\_enter\_\_ method and a \_\_exit\_\_ method. The \_\_exit\_\_ method is executed at termination time, regardless of whether an exception occurs or not.

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**. The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

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

Ans:- The Different Try/Except Variations. 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 .

Q25. What is the purpose of the raise statement?

Ans:- Python raise Keyword is used **to raise exceptions or errors**. The raise keyword raises an error and stops the control flow of the program. It is used to bring up the current exception in an exception handler so that it can be handled further up the call stack.

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

Ans:- assert statement takes an expression and optional message. assert statement is used **to check types, values of argument and the output of the function**. assert statement is used as debugging tool as it halts the program at the point where an error occurs.

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. Observe the following code example on how the use of with statement makes code cleaner.

# file handling

# 1) without using with statement

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

file.write('hello world !')

file.close()

# 2) without using with statement

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

try:

    file.write('hello world')

finally:

    file.close()

# using with statement

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

    file.write('hello world !')

An argument has two purposes: **change people's points of view or persuade them to accept new points of view**. persuade people to a particular action or new behavior.

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

Ans:- \*args and \*\*kwargs are special keyword which allows function to take variable length argument. \*args passes variable number of non-keyworded arguments and on which operation of the tuple can be performed

\*\*kwargs stands for keyword arguments. The only difference from args is that it uses keywords and returns the values in the form of a dictionary

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

Ans:- To pass, collect the arguments using the \* and \*\* in the function’s parameter list. Through this, you will get the positional arguments as a tuple and the keyword arguments as a dictionary. Pass these arguments when calling another function by using \* and \*\* −

def f(a, \*args, \*\*kwargs):

...

kwargs['width'] = '14.3c'

...

g(a, \*args, \*\*kwargs)

Q30. What are Lambda Functions?

Ans:-lambda function is a keyword in Python for defining the anonymous function. argument(s) is a placeholder, that is a variable that will be used to hold the value you want to pass into the function expression.

Q31. Explain Inheritance in Python with an example?

Ans:-Inheritance relationship defines the classes that inherit from other classes as derived, subclass, or sub-type classes. Base class remains to be the source from which a subclass inherits.

For example, you have a Base class of “Animal,” and a “Tiger” is a Derived class. The inheritance will be Tiger is an Animal.

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:- Global keyword is used when we want to read or write any global variable value inside the function. The global keyword used for a variable declared outside the function does not have any effect on it. In the same line, a variable cannot be declared global and assigned a value. E.g. global x = 5 is not allowed