Python OOP Assignment

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

In python, object oriented programming is a programming paradigm that uses classes and objects in programming. It aims to implement real-world entities like inheritance, encapsulation, polymorphism and so on. In other words, it binds data and functionalities together to work as a unit so that no other part of code can access this data.

The main purpose is the code reusability instead of starting from scratch.

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

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

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

Multiple instances can be created from class and also supports overloading methods, whereas an instance object inherits any functions nested in the class as methods for processing instances.

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

The first parameter of a function in a class must be the object itself. Writing this parameter as *self* is merely convention.

Q5. What is the purpose of the init method?

The __init__ method is called every time an object is created from a class. It is used to assign values to the object properties.

Example:

```
class Dog:
  attr = "mammal"


  def __init__(self,name):
     self.name = name
```

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

It is created by calling the class name and parenthesis associated with it just like a function. When arguments are passed to the class instance then it executes the instance method (__init__ method) of the class.
Example:

x = Myclass()
Y = Myclass(arg1, arg2)

Q7. What is the process for creating a class?

Class is created using a keyword class as shown below:

```
#Creating an empty class
class Myclass:
  pass
```

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

The class which inherits a class is called a parent class or superclass.

Q9. What is the relationship between classes and modules?

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

Modules may be mixed into classes and other modules. The mixed in module's constants and methods blend into that class's own, augmenting the class's functionality. Classes, however, cannot be mixed into anything.

A class may inherit from another class, but not from a module.A module may not inherit from anything.

Q10. How do you make instances and classes?

In an object-oriented programming language, a class plays the leading role. It means classes are building blocks of an object-oriented programming language.

Class is the most basic entity of Python because it acts as the core of object-oriented programming. Everything that a user sees in Python is an object, such as integers, functions, dictionaries, lists, and many other entities. It is a subpart of a class. Every Python object has a type, and users can create the object types using classes.

Instances are objects of a class in Python. In other words, users can define an instance of a particular class as an individual object.

- Firstly, the "def" keyword to define an instance method.
- Secondly, while defining the instance methods, users use the "self" as the first parameter within the instance method. The self parameter directs to the existing object.
- Lastly, users can use the self parameter to access or change the existing attributes of the Python object.

Q11. Where and how should class attributes be created?

Class attributes belong to the class itself; they will be shared by all the instances. Such attributes are defined in the class body parts usually at the top, for legibility.

Q12. Where and how are instance attributes created?

The instance attributes are not shared by objects. Every object has its own copy of the instance attribute.

To list the attributes of an instance/object, we have two functions:-

1. vars()– This function displays the attribute of an instance in the form of a dictionary.
2. dir()– This function displays more attributes than vars function,as it is not limited to instance. It displays the class attributes as well. It also displays the attributes of its ancestor classes.

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

The term self is nothing but the pointer to the data inside the class.

Q14. How does a Python class handle operator overloading?

When we use an operator on user-defined data types then automatically a special function or magic function associated with that operator is invoked. Changing the behavior of an operator is as simple as changing the behavior of a method or function. When we use + operator, the magic method **__add__** is automatically invoked in which the operation for + operator is defined. Thereby changing this magic method's code, we can give extra meaning to the + operator.

Whenever you change the behavior of the existing operator through operator overloading, you have to redefine the special function that is invoked automatically when the operator is used with the objects.

```
class Test:
  def __init__(self, a):
     self.a = a
  # adding two objects
  def __add__(self, o):
```

```
        return self.a + o.a
obj1 = Test(1)
obj2 = Test(2)
obj3 = Test("Asha")
obj4 = Test("Latha")
print(obj1 + obj2) #output: 3
print(obj3 + obj4)   #output: AshaLatha
# Actual working when Binary Operator is used.
print(Test.__add__(obj1 , obj2))  #output: 3
print(Test.__add__(obj3,obj4)) #output: AshaLatha
#And can also be Understand as :
print(obj1.__add__(obj2)) #output: 3
print(obj3.__add__(obj4)) #output: AshaLatha
```

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

- Improves code readability by allowing the use of familiar operators.
- 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?

The most frequent instance is the adding up operator '+', where it can be used for the usual addition and also for combining two different strings.

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

Inheritance and Polymorphism are the two most important concepts to grasp in python.

Q18. Describe three applications for exception processing.

ZeroDivisionError: Raised when division or modulo by zero takes place for all numeric types.

AttributeError: Raised in case of failure of attribute reference or assignment.

EOFError: Raised when there is no input from either the raw_input() or input() function and the end of file is reached.

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

When an exception occurs, 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?

- By providing 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.

In Python, exceptions can be handled by two new methods:

- 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.

Finally, the 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-except block.

Finally, the clause is optional. It is intended to define clean-up actions which should be executed in all conditions.

Q23. What is the purpose of the try statement?

The Try and Except 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.

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

The two most popular try statement variations are try-except and try-finally.

Q25. What is the purpose of the raise statement?

The purpose of the raise statement is to create custom exception handling.

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

In python, assert keyword helps in achieving this task. This statement takes as input a boolean condition, which when returns true doesn't do anything and continues the normal flow of execution, but if it is computed to be false, then it raises an AssertionError along with the optional message provided.

```
#using assert keyword
a = 4
b = 0
# using assert to check for 0
print("The value of a / b is : ")
assert b != 0
print(a / b)
```

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

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.

```
# without using with statement
file = open('file_name', 'w')
file.write('hello world !')
file.close()

#using with statement
with open('file_name','w') as file:
  file.write("Hello")
```

There is no need to call file.close() when using with statement. The with statement itself ensures proper acquisition and release of resources. Thus, with statement helps avoiding bugs and leaks by ensuring that a resource is properly released when the code using the resource is completely executed.

Q28. What are *args, **kwargs?

*args: We can pass any number of inputs that are non-keyword arguments of variable length argument list.

**kwargs: It is a key-worded argument which means variable length of key and values pairs will be provided as input in the function.

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

Using **kwargs we can pass optional or keyword parameters from one function to another.

Q30. What are Lambda Functions?

The lambda functions are anonymous which don't have a name and lambda is the reserved keyword.

Q31. Explain Inheritance in Python with an example?

Inheritance has the capability of inheriting or deriving properties from another class. It provides the reusability of the code.

```
#Inheritence:
class xyz:
  def __init__(self, a, b, c):
     self.a = a
     self.b = b
     self.c = c

  def test(self):
     print("This is the method of xyz")
  def test1(self):
     print("This is the method of test 1")
  def test2(self):
     print("This is the mentod of test 2")

class xyz1(xyz):
  def test(self):
     print("This method is of class xyz1")

# p = xyz(1,2,3)
```

```
# p.test()
# p.test1()
# p.test2()

q = xyz1(1, 2, 3)
q.test()
print(q.a)
```

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?

The sync() from class A gets invoked.

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

Using isinstance() function, we can test whether an object/variable is an instance of the specified type or class such as int or list. In case inheritance, we can check if the specified class is the parent class of an object.

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

The non-local keyword won't work on local or global variables and therefore must be used to reference variables in another scope. It is used in nested functions to reference a variable in the parent function(The variable should not belong to the inner function)

Q35. What is the global keyword?

A global keyword is a keyword that allows the user to modify a variable outside the current scope. It is used to create global variables inside a function and a global keyword is used inside a function only when we want to do an assignment or when we want to change a variable.