**## Python OOP Assignment**

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

In Python’s OOP, we use 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. It provides a clear program structure and a clean code, facilitates easy maintenance and modification of existing code.

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

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?**

A class is a type of blueprint that you can use to make objects. A concrete 'thing' that you constructed using a certain class is an object, which is an instance of a class. So, while the terms 'object' and 'instance' are interchangeable, the term 'instance' refers to an object's relationship to its class.

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

Whenever you call a method of an object created from a class, the object is automatically passed as the first argument using the “self” parameter. This enables you to modify the object's properties and execute tasks unique to that particular instance.

**Q5. What is the purpose of the init method?**

“\_\_init\_\_” is a Constructor in Python. The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object

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

To create an instance of a class, you call the class as if it were a function. For example, consider the following class:

class MyClass:

    pass

The following statement creates an instance of the class MyClass:

x = MyClass()

**Q7. What is the process for creating a class?**

To create a class, use the keyword “class:”. Create a class named “MyClass”, with a property named x

class MyClass:

  x = 5

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

In Python, the super() function is used to refer to the parent class or superclass. It allows you to call methods defined in the superclass from the subclass, enabling you to extend and customize the functionality inherited from the parent class.

class Emp():

    def \_\_init\_\_(self, name):

        self.name = name

class Freelance(Emp):

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

        # Using super() function.

        super().\_\_init\_\_(name)

        self.Email = Email

Emp\_1 = Freelance("Avneesh Kumar", "xyz@gmails")

print('The Name is:', Emp\_1.name)

print('The Emails is:', Emp\_1.Email)

**Q9. What is the relationship between classes and modules?**

A python module is nothing but a package to encapsulate reusable code. Modules can contain functions but also classes. Modules are imported using the import keyword.

Classes, in the other hand, can be defined in your main application code or inside modules imported by your application. Classes are the code of Object-Oriented Programming and can contain properties and methods.

**Q10. How do you make instances and classes?**

To create instances of a class, you call the class using class name and pass in whatever arguments its \_\_init\_\_ method accepts. And the class keyword indicates that you are creating a class followed by the name of the class.

class Emp():

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

        self.name = name

        self.Email = Email

Emp\_1 = Emp("Avneesh Kumar", "xyz@gmails")

print('The Name is:', Emp\_1.name)

print('The Emails is:', Emp\_1.Email)

**Q11. Where and how should be class attributes 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.

class emp:

    #Class attribute

    name = "Avneesh"

    def \_\_init\_\_(self):

        self.salary = 4000

    def show(self):

        print(self.salary)

e1 = emp()

print("Name is: ", e1.name, "and Salary is: ", e1.salary)

**Q12. Where and how are instance attributes created?**

Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor. Defined directly inside a class. Defined inside a constructor using the self parameter.

class emp:

    name = "Avneesh"

    def \_\_init\_\_(self):

        # instance attribute created.

        self.salary = 4000

    def show(self):

        print(self.salary)

e1 = emp()

print("Name is: ", e1.name, "and Salary is: ", e1.salary)

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

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.it is used to access variables that belongs to the class. it has to be the first parameter of any function in the class.

**Q14. How does a Python class handle operator overloading?**

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.

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

We consider allowing operator overloading, because it:

* 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 popular form of operator overloading is addition operator '+' is for concatenating and combining two different strings and intergers.

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

Both ‘Inheritance’ and ‘Polymorphism’ are the most important and fundamental concepts of object-oriented programming. These concepts help us to create code that can be extended and easily maintainable.

**Q18. Describe three applications for exception processing.**

* Try: This block will test the excepted error to occur
* Except: Here you can handle the error
* Else: If there is no exception then this block will be executed
* Finally: Finally block always gets executed either exception is generated or not

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

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?**

Try and except statements are used to catch and handle exceptions in Python. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.

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

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.**

The `finally` keyword in the try-except block is always executed, irrespective of whether there is an exception or not.

**Q23. What is the purpose of the try statement?**

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.

def divide(x, y):

    try:

        result = x // y

        print("Yeah ! Your answer is :", result)

    except ZeroDivisionError:

        print("Sorry ! You are dividing by zero ")

divide(3, 2)

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

There are two most popular 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.

def divide(x, y):

    try:

        result = x // y

    except ZeroDivisionError:

        print("Sorry ! You are dividing by zero ")

    else:

        print("Yeah ! Your answer is :", result)

    finally:

        # this block is always executed

        # regardless of exception generation.

        print('This is always executed')

# parameters passed for execution.

divide(3, 2)

divide(3, 0)

**Q25. What is the purpose of the raise statement?**

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.

a = 5

if a % 2 != 0:

    raise Exception("The number shouldn't be an odd integer")

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

Assert 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.

a = 4

b = 0

# using assert to check for 0

print("The value of a / b is : ")

assert b != 0, "Zero Division Error"

print(a / b)

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

The ‘with’ statement in Python replaces a ‘try-catch’ block with a simple shorthand. More significantly, it ensures that resources are closed immediately after processing. Reading or writing to a file is a common use of the with statement.

# using ‘with/as’ statement

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

    file.write('hello world !')

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

“We use the “wildcard” or “\*” notation like this – \*args OR \*\*kwargs – as our function’s argument when we have doubts about the number of arguments we should pass in a function.”

* \*args (Non-Keyword Arguments)
* \*\*kwargs (Keyword Arguments)

The special syntax \*args in function definitions in Python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

The special syntax \*\*kwargs in function definitions in Python is used to pass a keyworded, variable-length argument list. We use the name kwargs with the double star. The reason is that the double star allows us to pass through keyword arguments (and any number of them).

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

In Python, when we define functions with default values for certain parameters, it is said to have its arguments set as an option for the user. Users can either pass their values or can pretend the function to use theirs default values which are specified.

# Here b is predefined and hence is optional.

def func(a, b=1098):

    return a+b

#Passing optional argument of 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?**

Python Lambda Functions are anonymous function means that the function is without a name. As we already know that the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.

a = int(input("Enter first number: "))

b = int(input("Enter Second number: "))

# Using Lambda function

x = lambda a, b : a + b

print("Addition of numbers using lambda function: a + b = ",  x(a,b))

**Q31. Explain Inheritance in Python with an example?**

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, fname, lname):

    self.firstname = fname

    self.lastname = lname

  def print\_name(self):

    print(self.firstname, self.lastname)

x = Person("Avneesh", "Kumar")

x.print\_name()

# Inheritance - Below Student class, inherit Person class.

class Student(Person):

  pass

x = Student("Vishal", "Sharma")

x.print\_name()

**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?**

According to question, C does not contain its own version of func(). So the interpreter searches in a left-to-right fashion, if it finds the method in A, then it does not go to look for it in B.

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

The isinstance() method is used to determine if an object belongs to a specific class, and

The issubclass() method checks if one class is derived from another class or classes, establishing a subclass relationship.

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

The nonlocal is a keyword in python that is used to declare any variable as not local but instead comes from the nearest enclosing scope that is not global.

def display():

    name = "Avneesh" # Our local variable

    def display\_nonlocal():

        nonlocal name          # Reference name in the upper scope

        name = 'Avneesh\_Kumar' # Overwrite this variable

        print(name)

    display\_nonlocal()

    print(name)

display()

**Q35. What is the global keyword?**

A global keyword is a keyword that allows a user to modify a variable outside the current scope. It is used to create global variables in Python from a non-global scope, i.e. inside a function.

# global variable

a = 15

b = 10

def add():

    c = a + b

    print(c)

add()