***Python OOP Assignment***

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

OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for programs that are large, complex and actively updated or maintained. This includes programs for manufacturing and design, as well as mobile applications; for example, OOP can be used for manufacturing system simulation software.

OOP helps in reducing time and complexity of programs it is based on DRY ( do not repeat yourself ) , helping in saving time and money.

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

An inheritance search looks for an attribute first in the instance object, then it looks in the class the instance was created from and 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?**

I was not able not able to come to at a concrete answer for this question

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

The calling process is automatic while the receiving process is not . 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?**

When you make a class in Python the first method you will likely make is the \_\_init\_\_ method. The \_\_init\_\_ method allows you to accept arguments to your class.

More importantly, the \_\_init\_\_ method allows you to assign initial values to various attributes on your class instances.

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

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

**For example :**

Class Employee:

empCount = 0

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

self.name = name

self.salary = salary

Employee.empCount += 1

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

The class statement creates a new class definition. The name of the class immediately follows the keyword class followed by a colon.

**For example :**

Class car :

def \_\_init\_\_(self,mileage,cost)

self.milage = mileage

self.cost = cost

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

Let's say if there are many classes i.e. Vehicle , Truck , Car , Motorcycle etc

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?**

I was not able not able to come to at a concrete answer for this question

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

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

**For example :** class vehicle:

def \_\_init\_\_(self,colour,price)

self.colour = colour

self.price = price

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

A class attribute is common to 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 constants, track data across all instances, and setting default values for all instances of the class.

**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.

Instance attributes are defined in the \_\_init\_\_() function.

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

The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

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

I was not able not able to come to at a concrete answer for this question

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

I was not able not able to come to at a concrete answer for this question

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

Operator overloading is the process of using an operator in different ways depending on the operands. You can change the way an operator in Python works on different data-types.

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.

Operators in Python work for built-in classes, like int, str, list, etc. But you can extend their operability such that they work on objects of user-defined classes too.

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

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.

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

I was not able not able to come to at a concrete answer for this question

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

When an exception occurs at that time if you don’t handle it, the program terminates abruptly and the code that is there past the line that caused the exception will not get executed.

**Q20. What are your options for recovering from an exception in your script?**

I was not able not able to come to at a concrete answer for this question

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

I was not able not able to come to at a concrete answer for this question

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

I was not able not able to come to at a concrete answer for this question

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

The try block lets you test a block of code for errors.

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.These exceptions can be handled using the try statement

Try statement generally accompanied by except block if the argument or condition inside the try block raises an error it will automatically execute the Except block and the program won't return an error.

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

Try Except and Try Except Else are two popular try statement variations.

**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.

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

I was not able not able to come to at a concrete answer for this question

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

I was not able not able to come to at a concrete answer for this question

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

**\*args : -** \*args in function definitions in python are used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list.

**\*\*kwargs : -** it allows us to pass the variable length of keyword arguments to the function.In the function, we use the double asterisk \*\* before the parameter name to denote this type of argument. The arguments are passed as a dictionary and these arguments make a dictionary inside function with name same as the parameter excluding double asterisk \*\*.

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

I was not able not able to come to at a concrete answer for this question

**Q30. What are Lambda Functions?**

A lambda function is a small anonymous function.

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

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

Inheritance is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs

class Vehicle:

def \_\_init\_\_(self,mileage,cost):

self.mileage = mileage

self.cost = cost

def showVehicledetails(self):

print('mileage of vehicle :',self.mileage)

print('cost of vehicle is :',self.cost)

print('I am a vehicle')

class Car(Vehicle):

def \_\_init\_\_(self,mileage,cost,tyre,hp):

super().\_\_init\_\_(mileage,cost)

self.tyre = tyre

self.hp = hp

def showcardetails(self):

print('Numbers of tyre in a car :',self.tyre)

print('HP of a car is :',self.hp)

print('I am a car')

c1=Car(26,100000,4,2000)

c1.showcardetails()

**Output :- Numbers of tyre in a car : 89**

**HP of a car is : 9000**

**I am a car**

**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?**

I was not able not able to come to at a concrete answer for this question

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

Use isinstance() to check an instance's type: isinstance(obj, int) will be True only if obj.\_\_class\_\_ is int or some class derived from int

Use issubclass() to check class inheritance: issubclass(bool, int) is True since bool is a subclass of int

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

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?**

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 keyword is used inside a function only when we want to do assignments or when we want to change a variable. Global is not needed for printing and accessing.