**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: In Python, inheritance happens when an object is qualified, and involves searching an attribute definition tree (one or more namespaces). Every time you use an expression of the form object. Attribute where object is an instance or class object, Python searches the namespace tree at and above object, for the first attribute it can find.

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

Ans: The class is just another variable somewhere else in memory that holds the same exact information that the object does. Note the difference between the relationships: Object is a copy of the class. Instance is a variable that holds the memory address of the object.

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

Ans: The first parameter in the class method is the class on which you are calling the method, not (necessarily) the class that defines the method. (Having a variable that always holds the same class would probably not be that useful.)

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

Ans: The Default \_\_init\_\_ Constructor in Python. Constructors are used to initializing the object’s state. 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 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?

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

class shape:

# Calling Constructor

def \_\_init\_\_(self, edge, color):

self.edge = edge

self.color = color

# Instance Method

def finEdges(self):

return self.edge

# Instance Method

def modifyEdges(self, newedge):

self.edge = newedge

Q7. What is the process for creating a class?

Ans: In Python, we use classes to create objects. A class is a tool, like a blueprint or a template, for creating objects. It allows us to bundle data and functionality together.

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 below example, Vehicle is the Superclass and its subclasses are Car, Truck and Motorcycle.

Vehicle

Car Truck Motorcycle

Q9. What is the relationship between classes and modules?

Ans: Like everything else, class names always live within a module. Class statements are run during imports to define names, and these names become distinct module attributes. Since classes are attributes of modules, the dot operator must be used to reference the class unless it is an attributed of the \_\_main\_\_ 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. Example :

class shape:

# Calling Constructor

def \_\_init\_\_(self, edge, color):

self.edge = edge

self.color = color

# Instance Method

def finEdges(self):

return self.edge

# Instance Method

def modifyEdges(self, newedge):

self.edge = newedge

Q11. Where and how should be class attributes created?

Ans : When you access an attribute via an instance of the class, Python searches for the attribute in the instance attribute list. If the instance attribute list doesn’t have that attribute, Python continues looking up the attribute in the class attribute list. Python returns the value of the attribute as long as it finds the attribute in the instance attribute list or class attribute list.

Exam : class Class\_name:

    variable = value

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. Exam:

def function\_name():

    variable = value

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.

Q14. How does a Python class handle operator overloading?

Ans : In the Python programming language, classes (as opposed to modules) can override operators. This mechanism is called operators overloading. Operator overloading in a class – is a mapping to a Python language operator (for example, operator +, –, \*, etc.) of a corresponding method in order to call the code of that method when using the overloaded operator. In other words, operator overloading allows class objects to be used as operands in expressions or other built-in operations such as addition, multiplication, slicing, inference, etc. If an overloaded operator is called for a certain class object in the program, then the method is automatically called that implements the actions of this operator. Thus, when the operator + (addition) is overloaded, any actions can be programmed in the method that implements it, even subtraction, multiplication, or whatever. However, such code will not have logic, misleading the programmer himself. As usual, the method corresponding to the + operator must sum something.

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

Ans: 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.

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

Ans: 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. As mentioned on top, the plus symbol’s practice in dissimilar forms is the largest classic example of the operator level overloading process.

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

Ans: The two most important concepts to grasp in order to comprehend Python OOP code are inheritance and polymorphism.

Q18. Describe three applications for exception processing.

Ans:

1. Error handling: The exceptions get raised whenever Python detects an error in a program at runtime. As a programmer, if you don't want the default behavior, then code a 'try' statement to catch and recover the program from an exception. Python will jump to the 'try' handler when the program detects an error; the execution will be resumed.
2. Event Notification: Exceptions are also used to signal suitable conditions & then passing result flags around a program and text them explicitly.
3. Terminate Execution: There may arise some problems or errors in programs that it needs a termination. So try/finally is used that guarantees that closing-time operation will be performed. The 'with' statement offers an alternative for objects that support it.

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

Ans: In Python, Built-in exceptions can be raised when a program encounters an error. When these exceptions happen, the Python interpreter holds the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

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

Ans: try/except: catch the error and recover from exceptions hoist by programmers or Python itself.

try/finally: Whether exception occurs or not, it automatically performs the clean-up action.

assert: triggers an exception conditionally in the code.

raise: manually triggers an exception in the code.

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

Ans : raise allows you to throw an exception at any time.

assert enables you to verify if a certain condition is met and throw an exception if it isn’t.

In the try clause, all statements are executed until an exception is encountered.

except is used to catch and handle the exception (s) that are encountered in the try clause.

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

Ans: The quit() function is an inbuilt function that you can use to terminate a program in python. When the quit () function is executed, it raises the System Exit exception. This results in the termination of the program. You can observe this in the following example.

number = 10

if number >= 10:

print("The number is:", number)

quit()

print("This line will not be printed")

The exit() function is also defined in the site module. It works in a similar way to the quit() function and terminates the program when executed as shown in the following example

number = 10

if number >= 10:

print("The number is:", number)

exit()

print("This line will not be printed")

Q23. What is the purpose of the try statement?

Ans: The Python TRY command tries a given clause. If it’s not successful, it prints an error message or otherwise handles the error. Both the initial clause and the error clause will be user-defined.

Ans: What does try mean in python

The Python try statement is meant to deal with exceptions. Some examples of exceptions are: Attempting to do arithmetic with a string variable Opening a file for writing when the system has marked it read-only Pressing the keyboard combination that stops execution Exceptions pose two problems.

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

Ans : KeyboardInterrupt: when an unrequired key is pressed by the user

Value Error: when built-in function receives a wrong argument

Q25. What is the purpose of the raise statement?

Ans: The purpose of the raise statement is to raise an exception explicitly.

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

Ans: In Python, the assert statement is used to continue the execute if the given condition evaluates to True. If the assert condition evaluates to False, then it raises the Assertion Error exception with the specified error message.

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

Ans: with statement in Python is used in exception handling to make the code cleaner and much more readable. It simplifies the management of common resources like file streams.

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

Ans: 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-key worded, 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?

Ans: To pass optional or keyword parameters from one function to another, collect the arguments using the \* and \*\* specifies in the function’s parameter list.

def demo(\*car):

print("Car 1 = ",car[0])

print("Car 2 = ",car[1])

print("Car 3 = ", car[2])

print("Car 4 = ", car[3])

# call

demo("Tesla", "Audi", "BMW", "Toyota")

Q30. What are Lambda Functions?

Ans: A lambda function is an anonymous function (i.e., defined without a name) that can take any number of arguments but, unlike normal functions, evaluates and returns only one expression.

Q31. Explain Inheritance in Python with an example?

Ans: Inheritance is the capability of a class to inherit all methods of base class from which it is derived and can add new features to the class without modifying it.

Inheritance from Father to Son is the simple example.

Syntax: Class BaseClass:

{Body}

Class DerivedClass(BaseClass):

{Body}

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?

Ans: Both the versions A and B.

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

Ans: In Python isinstance () and issubclass () 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).

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.

def myfunc1():

x = "John"

def myfunc2():

x = "hello"

myfunc2()

return x

print(myfunc1())

Q35. What is the global keyword?

Ans : In Python, global keyword allows you 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.