***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.

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

Ans. 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?

Ans. 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?

Ans. meth(args) becomes Class. 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?

Ans. The \_\_init\_\_ method **lets the class initialize the object's attributes and serves no other purpose**. It is only used within classes.

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.   
Create class.  
  
Enter the class name. (Optional) To enter a short description, grade level, or class time, tap Section and enter the details. (Optional) To enter the location for the class, tap Room and enter the details.

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

Ans.  A class that is derived from another class is called a subclass (also a derived class, extended class, or child class). **The class from which the subclass is derived** is called a superclass (also a base class or a parent class).

Q9. What is the relationship between classes and modules?

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

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**.

Q11. Where and how should be class attributes created?

Ans. Class attributes are the variables defined **directly in the class** that are shared by all objects of the class. 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.

Q12. Where and how are instance attributes created?

Ans. Instance attributes are **defined in the constructor**. Defined directly inside a class. Defined inside a constructor using the self parameter.   
Instance attributes are defined **in the \_\_init\_\_() function**.

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

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

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.  Two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary '+' operator it throws an error, because compiler don't know how to add two objects. So we define a method for an operator and that process is called operator overloading.

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

Ans. 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. 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.

Q18. Describe three applications for exception processing.

Ans.

|  |  |
| --- | --- |
| **AssertionError** | **Raised when an assert statement fails.** |
| **AttributeError** | **Raised when attribute assignment or reference fails.** |
| **EOFError** | **Raised when the input() function hits end-of-file condition.** |

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.  **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. To avoid such a scenario, 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.

Ans. 1. Use the 'finally' clause of a 'try-catch' block. 2. Use the 'atexit' module in Python to register a function to be executed at termination time.

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. The RAISE statement **stops normal execution of a PL/SQL block or subprogram and transfers control to an exception handler**. RAISE statements can raise predefined exceptions, such as ZERO\_DIVIDE or NO\_DATA\_FOUND , or user-defined exceptions whose names you decide.

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. The with statement is **a replacement for commonly used try/finally error-handling statements**. A common example of using the with statement is opening a file. To open and write to a file in Python, you can use the with statement as follows: with open("example.

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. **By using keyword arguments**.

Q30. What are Lambda Functions?

Ans. lambda 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. A lambda function can have multiple variables depending on what you want to achieve.

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 “Lion” is a Derived class. The inheritance will be Lion is an Animal.

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. It depends on the language you are using and the specific implementation of the inheritance rules. Generally, the version of func() that is invoked will be determined by the order in which the classes are listed in the inheritance hierarchy. If class A is listed first, then its version of func() will be invoked. If class B is listed first, then its version of func() will be invoked.

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

Ans. **Python has two built-in functions that work with 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.

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.