**Python OOP Assignment**

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

Ans:- The purpose of Python's Object-Oriented Programming (OOP) is to enable developers to organize their code into reusable, modular structures that model real-world objects and their behaviors. OOP allows developers to create classes, which define the characteristics and behaviors of an object, and objects, which are instances of a class with their own state and methods. This approach helps to make code more organized, easier to understand and modify, and more scalable. OOP also enables the use of inheritance, polymorphism, and encapsulation, which are powerful concepts that can help to reduce code duplication and improve code reuse.

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

Ans:- In Python's inheritance hierarchy, when an attribute is accessed on an object, the interpreter searches for that attribute in the following order:

* The instance's own attributes (i.e., attributes defined on the object itself)
* The class attributes (i.e., attributes defined on the class itself)
* The base class(es) in the order they are listed in the inheritance tuple (i.e., the "superclasses" of the class).

This order of attribute lookup is commonly referred to as the Method Resolution Order (MRO) and is determined by the C3 linearization algorithm, which creates a linear order of the class hierarchy that preserves the order of method calls among all classes. If the attribute is not found in any of these locations, a AttributeError will be raised.

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

In Python, a class object is an object that defines a class, whereas an instance object is an object that is created from a class. Here's how you can distinguish between the two:

* Class objects are created by the class statement, while instance objects are created by calling the class, like a function, with the parentheses. For example, class MyClass: defines a class object, and my\_object = MyClass() creates an instance object of that class.
* Class objects have attributes that are shared by all instances of the class. These attributes can be accessed using the class name, like MyClass.attribute. Instance objects, on the other hand, have their own set of attributes that are specific to that instance. These attributes can be accessed using the instance name, like my\_object.attribute.
* Class objects can have methods that operate on the class or on its instances. These methods can be called using the class name or the instance name. Instance objects can also have methods that operate on that specific instance. These methods can only be called using the instance name.
* Class objects are themselves instances of the type class, while instance objects are not. The type of a class object is type, while the type of an instance object is the class that it was created from.

In summary, a class object is a blueprint that defines the attributes and behaviors of a class, while an instance object is a specific realization of that class, with its own set of attributes and behaviors.

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

In Python, the first argument in a class's method function is conventionally named self and refers to the instance of the class that the method is being called on. This argument is automatically passed by the Python interpreter when an instance method is called, and it allows the method to access and manipulate the attributes of the specific instance on which it is being called.

The self argument is what makes instance methods different from static methods, which don't have access to any specific instance and don't need the self argument. The self argument is also what allows for the creation of instance-specific attributes and methods, which can vary from instance to instance even if they are created from the same class.

It's important to note that the use of self as the name of the first argument is a convention in Python, and other names could be used instead. However, it's generally recommended to stick to this convention for the sake of readability and consistency with other Python code.

Q5. What is the purpose of the init method?

Ans:- In object-oriented programming, the \_\_init\_\_ method (or constructor) is a special method that is called when an instance of a class is created. The purpose of the \_\_init\_\_ method is to initialize the attributes (properties) of the object and prepare it for use.

Specifically, the \_\_init\_\_ method is responsible for setting initial values of instance variables, which are attributes specific to each instance of the class. Instance variables can be used to store data that is unique to each object, and can be accessed and modified using the self keyword within the methods of the class.

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

Ans:-

* Define the class: First, you need to define the class that you want to create an instance of. The class defines the properties and methods that will be associated with the object.
* Create an object: To create an object (or instance) of a class, you need to call the class as if it were a function. This creates a new object in memory with the attributes and methods defined in the class. This is known as "instantiation".
* Initialize the object: After creating the object, you need to initialize its attributes by calling the \_\_init\_\_ method. This method takes the object as the first parameter (which is usually named self) and any other parameters that you have defined in the \_\_init\_\_ method.
* Access the attributes and methods: Once the object has been created and initialized, you can access its attributes and methods using dot notation. For example, if the object is named my\_object and it has an attribute named attribute1, you can access it using my\_object.attribute1. Similarly, if it has a method named method1, you can call it using my\_object.method1().

Q7. What is the process for creating a class?

Ans:

* Define the class name: Choose a name for the class that reflects its purpose. The name should start with a capital letter and use "camel case" (i.e., capitalize the first letter of each word in the name except the first word). For example, "Car" or "BankAccount".
* Define the class attributes: Think about the properties that objects of this class will have. These are called "attributes". Define the attributes inside the class using variable assignments. You can also define default values for the attributes if you like.
* Define the class methods: Think about the actions that objects of this class will be able to perform. These are called "methods". Define the methods inside the class using function definitions. The first parameter of each method should always be self, which refers to the instance of the class that the method is called on.
* Use the class: Once you've defined the class, you can create instances of it and use its methods and attributes.

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

Ans:- In object-oriented programming, a superclass (or parent class) is a class from which other classes can inherit attributes and methods. To define a superclass of a class in Python, you need to specify the name of the superclass inside the parentheses of the class definition.

Q9. What is the relationship between classes and modules?

Ans:-

In Python, a module is a file that contains Python definitions and statements. It can define functions, classes, and variables, and can be imported and used in other Python files.

Classes, on the other hand, are a way to organize code and data into reusable and modular units. A class is a template for creating objects that encapsulate data and behavior.

A module can contain one or more classes, along with other Python definitions and statements. Classes defined in a module can be used by other modules, and can also import other modules to use their definitions.

In general, modules are used to group related code together, while classes are used to define the structure and behavior of objects. A module can provide a set of related classes that can be used in other programs, and a class can be defined in one module and imported into another module.

Q10. How do you make instances and classes?

Ans:- To make an instance of a class in Python, you first need to define the class, and then create an object (also known as an instance) of that class using the class name followed by parentheses

Q11. Where and how should be class attributes created?

Ans:- Class attributes are attributes that are associated with a class rather than with an instance of that class. They are defined inside the class definition, but outside of any method definition. Class attributes are shared by all instances of the class, which means that they can be accessed and modified by any instance of the class, as well as by the class itself.

Q12. Where and how are instance attributes created?

Ans:- Instance attributes are attributes that are associated with a specific instance of a class, rather than with the class itself. They are created and initialized in the \_\_init\_\_ method of the class, which is called when a new instance of the class is created.

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

Ans:- In Python, self is a convention used in the definition of a method of a class to refer to the instance of that class. It is the first argument to any instance method in a class, and is used to access the attributes and methods of that instance.

Q14. How does a Python class handle operator overloading?

Ans:- Python provides a mechanism for operator overloading, which allows you to define the behavior of operators (such as +, -, \*, /, etc.) for objects of your own classes. This mechanism is implemented through special methods, also known as "magic methods" or "dunder methods" (short for "double underscore methods").

For example, to overload the + operator for a class MyClass, you can define the special method \_\_add\_\_(self, other):

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

Ans:- Operator overloading in Python can be useful when you want to define the behavior of operators (such as +, -, \*, /, etc.) for objects of your own classes in a way that makes sense for your application. This can make your code more intuitive, expressive, and easier to read, especially when working with complex data structures.

You might consider allowing operator overloading of your classes when:

* Your class represents a mathematical concept, such as a vector, matrix, or complex number, where the mathematical operators have a natural meaning.
* Your class represents a data structure, such as a list, set, or dictionary, where the operators can be used to combine or compare instances of the class.
* Your class represents a custom data type that has a specific behavior that can be naturally expressed using operators.

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

Ans:- In Python, the most popular form of operator overloading is probably the overloading of arithmetic operators such as +, -, \*, /, //, %, \*\*, and unary operators such as -, +, abs(), ~, and not. This is because many classes represent mathematical concepts, such as numbers, vectors, matrices, or complex numbers, where the arithmetic operators have a natural meaning.

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

Ans:-

In order to comprehend Python OOP code, it's important to grasp the following two key concepts:

* Classes and Objects: A class is a blueprint or template for creating objects, which are instances of the class. A class defines the properties (attributes) and behavior (methods) of the objects that can be created from it. To create an object of a class, you use the class name as a constructor and call it with parentheses.
* Inheritance and Polymorphism: Inheritance is a mechanism that allows a new class to be based on an existing class (the parent class), inheriting all its attributes and methods, and possibly adding new ones or overriding existing ones. Polymorphism is the ability of objects of different classes to be treated as if they were of the same class, by sharing a common interface.

By understanding these two concepts, you can create, modify, and reuse Python classes, and use them to model real-world problems in a more intuitive and efficient way. You can also take advantage of the built-in classes and modules provided by Python, and the many third-party libraries that extend the language and provide additional functionality.

Q18. Describe three applications for exception processing.

Ans:-

Exception processing is a powerful mechanism in Python that allows you to handle unexpected or exceptional situations in your code, and respond to them in a controlled and graceful way. Here are three applications for exception processing in Python:

* Error handling: One of the most common applications of exception processing is error handling. When a function or method encounters an error, such as a file not found, a division by zero, or an invalid input, it can raise an exception, which signals that something has gone wrong. By catching the exception with a try-except block, you can handle the error in a way that is appropriate for your application, such as displaying an error message, logging the error, or retrying the operation.
* Resource management: Another important application of exception processing is resource management. In Python, resources such as file handles, database connections, or network sockets, need to be released or closed when they are no longer needed, to avoid memory leaks or other problems. By using a try-finally block or a with statement, you can ensure that the resources are properly released, even if an exception occurs.
* Control flow: Exception processing can also be used for control flow. By raising or catching exceptions in a specific way, you can control the flow of execution in your code, and make it more robust and flexible. For example, you can use exceptions to break out of a loop, to skip over some parts of the code, or to handle multiple errors in a hierarchical way. This can make your code more readable and maintainable, and easier to debug and test.

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

Ans:-

If you don't do something extra to treat an exception in your Python code, the exception will propagate up the call stack until it is caught by a higher-level try-except block, or until it reaches the top level of the program, at which point it will terminate the program and print a traceback message.

This behavior is known as "unhandled exception", and it can be a serious problem in large or complex programs, where an unexpected error can have unforeseen consequences, such as data corruption, security breaches, or system crashes.

By handling exceptions with try-except blocks or other exception processing mechanisms, you can catch and handle errors in a controlled way, and take appropriate actions to recover from them, such as retrying the operation, logging the error, displaying a message to the user, or rolling back a transaction.

Handling exceptions can make your code more robust, reliable, and maintainable, and improve the user experience by providing meaningful feedback and preventing unexpected errors from disrupting the program flow.

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

Ans:- In Python, when an exception occurs in your script, there are several options for recovering from it, depending on the nature of the exception and the requirements of your application. Here are some common recovery options:

* Retry the operation: If the exception is caused by a temporary or recoverable error, such as a network connection failure or a file access error, you can retry the operation after a certain interval or after applying some corrective action.
* Log the error: If the exception is caused by a problem that needs to be investigated or fixed later, such as a programming error or a configuration problem, you can log the error message and other relevant information to a log file or a database, for later analysis.
* Graceful degradation: If the exception is caused by a feature or a service that is not essential for the application to function properly, such as a non-critical plugin or a user preference, you can degrade the functionality in a graceful way, by disabling the feature or providing a default value.
* Display an error message: If the exception is caused by a user input or a user action, such as an invalid input or a wrong password, you can display an error message that explains the problem and provides guidance on how to correct it.
* Rollback the transaction: If the exception is caused by a database operation or a transaction that needs to be rolled back, you can undo the changes and restore the previous state, to ensure consistency and data integrity.

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

Ans:- In Python, there are several ways to trigger exceptions in your script, depending on the situation and the requirements of your application. Here are two common methods for triggering exceptions:

* Raise an exception explicitly: You can raise an exception explicitly using the raise statement. This allows you to define custom exceptions and to specify the exception type and message.
* Use built-in functions that can raise exceptions: There are many built-in functions in Python that can raise exceptions, such as open() for file I/O, int() for type conversion, or list.pop() for list manipulation. If you use these functions in your code, it is important to handle the exceptions they may raise.

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

* The finally block: The finally block is used in conjunction with a try-except block to specify code that should be executed regardless of whether an exception is raised or not. The code in the finally block is executed after the try-except block has finished, and it is guaranteed to be executed, even if an exception is raised or the code inside the try block returns or breaks
* The atexit module: The atexit module is used to register functions that should be executed when the Python interpreter is about to exit. These functions are called "exit handlers" and they can be used to perform cleanup tasks or save state information before the program exits

Q23. What is the purpose of the try statement?

Ans:- In Python, the try statement is used to handle exceptions that may occur during the execution of a program. The purpose of the try statement is to specify a block of code that may raise an exception, and to provide a way to handle the exception if it is raised.

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

Ans:- In Python, the two most popular try statement variations are:

* try-except block: This is the most common try statement variation, which allows you to handle specific exceptions that may be raised in the try block. You can specify one or more except blocks to catch specific exceptions, or you can use a generic except block to catch all exceptions.
* try-finally block: This variation is used when you need to ensure that certain code is executed regardless of whether an exception is raised or not. The finally block is executed after the try block finishes, and it is guaranteed to be executed, even if an exception is raised or the code inside the try block returns or breaks.

Q25. What is the purpose of the raise statement?

Ans:- In Python, the raise statement is used to manually raise an exception. The purpose of the raise statement is to signal that an error or an exceptional condition has occurred during the execution of a program, and to cause the program to terminate with an appropriate error message.

The general syntax of the raise statement is as follows:

raise ExceptionType("error message")

In this syntax, ExceptionType is the type of the exception to be raised, and "error message" is a string that provides more information about the error or the exceptional condition that occurred.

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

Ans:- In Python, the assert statement is used to check whether a given condition is true, and raise an exception with a specified error message if the condition is false. The general syntax of the assert statement is:

assert condition, error\_message

Here, condition is the expression to be checked for truth, and error\_message is the optional error message to be raised if the condition is false.

The assert statement is similar to the if statement in that it checks a condition and executes a code block if the condition is true. However, the key difference is that if the condition is false, the assert statement raises an AssertionError exception with the specified error message, and terminates the program.

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

Ans:- In Python, the with statement provides a way to simplify the management of resources that need to be acquired and released in a predictable and reliable way. The general syntax of the with statement is:

with expression as variable:

# code block

Here, expression is an object that represents a resource that needs to be managed, and variable is a name that is used to refer to the resource within the code block. The with statement automatically acquires the resource before the code block is executed, and releases it when the code block completes, even if an exception is raised.

The with statement is similar to a try/finally block, which also provides a way to ensure that resources are properly managed and released, even in the event of an exception. However, the with statement is generally considered to be a more elegant and concise way to manage resources, since it eliminates the need for explicit try/finally blocks and simplifies the code.

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

Ans:- In Python, \*args and \*\*kwargs are special syntax used in function definitions that allow the function to accept an arbitrary number of arguments.

The \*args syntax is used to pass a variable number of non-keyword arguments to a function. When \*args is used in a function definition, it allows the function to accept any number of arguments, which are passed as a tuple

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

Ans:- In Python, you can pass optional or keyword parameters from one function to another using the \*args and \*\*kwargs syntax.

The \*args syntax is used to pass a variable number of non-keyword arguments to a function, and the \*\*kwargs syntax is used to pass a variable number of keyword arguments to a function.

Q30. What are Lambda Functions?

Ans:- Lambda functions, also known as anonymous functions, are small, single-expression functions that can be defined in a single line of code. They are called "anonymous" functions because they do not have a name, and are defined on-the-fly as they are needed.

Q31. Explain Inheritance in Python with an example?

Ans:- Inheritance is a key feature of object-oriented programming that allows one class to inherit the properties and methods of another class. In Python, a subclass can inherit attributes and behaviors from a parent or superclass by using the keyword class and specifying the name of the superclass in parentheses.

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:-

If class C inherits from classes A and B as class C(A, B), and both A and B have their own versions of the method func(), then the version of func() that gets invoked when called from an object of class C depends on the order of the inheritance.

If A is listed first in the parentheses in the class definition (class C(A, B)), then the version of func() in A will be called. If B is listed first, then the version of func() in B will be called.

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

Ans:- In Python, we can use the built-in type() function to determine the type of an instance. We can also use the isinstance() function to check whether an object is an instance of a particular class or a subclass of that class.

To determine the inheritance hierarchy of an object, we can use the \_\_mro\_\_ attribute, which returns a tuple containing the method resolution order for the object. The method resolution order determines the order in which Python searches for methods to invoke when called on an object.

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

Ans:- The nonlocal keyword in Python is used to refer to a variable that is defined in the nearest enclosing scope that is not the global scope. This means that the nonlocal keyword allows you to assign a value to a variable in an outer (but not global) scope from within an inner scope.

Q35. What is the global keyword?

Ans:-

In Python, the global keyword is used to indicate that a variable is defined in the global scope, i.e., it should be accessible from anywhere in the code.

When a variable is defined inside a function, by default it is local to that function and cannot be accessed from outside the function. However, if we want to make the variable accessible outside the function or in other functions, we can use the global keyword to indicate that the variable should be treated as a global variable.