Assignment-3

Python OOP Assignment

Python_OOP_Assignment.md

iNeuron-Bigdata-BootCamp 2.0)

* Name-Surendra Kumar Yadav

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

The main purpose of OOP is to help organize and structure code in a way that is easier to understand and maintain. OOP allows you to create reusable blocks of code that can be used in multiple programs, and it makes it easier to make changes to your code because you only have to change the code in one place instead of searching through and modifying code in multiple places.

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

when you try to access an attribute of an object, the interpreter looks for the attribute in the object's own namespace. If the attribute is not found there, the interpreter will then look for the attribute in the object's class and its superclasses For example, if you have a class hierarchy with classes A, B, and C, where C is derived from B and B is derived from A, the MRO for C will be C, B, A. If you try to access an attribute or method that is not found in C, the interpreter will look for it in B, and then in A if it is not found in B.

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

A class is a template or blueprint for creating objects. It defines the attributes and behavior of a type of object. An instance is a specific object created from a class. Classes: A class is a template for creating objects in program. A class is a logical entity A class does not allocate memory space when it is created. You can declare class only once. Example: Car. Class generates objects Object: The object is an instance of a class. Object is a physical entity Object allocates memory space whenever they are created. You can create more than one object using a class. Example: Jaguar, BMW, Tesla, etc. Objects provide life to the class.class MyClass: pass obj = MyClass() Here, `MyClass` is a class object and `obj` is an instance object. You can check the type of each object using the `type` function:

In [3]:
```
print(type(MyClass))
print(type(obj))
#As you can see, MyClass is an instance of the type class, while obj is an instance of the MyClass class.
```

```
<class 'type'>
<class '__main__.MyClass'>
```

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

A class method is similar to an instance method, but it has a class object passed as its first argument. Recall that, when an instance method is called from an instance object, that instance object is automatically passed as the first argument to the method. The first argument in a method function of a class is usually called self. This argument is special because it refers to the instance of the class that is calling the method.

In [6]:
```
class MyClass:
    def greet(self):
        print("Hello!")

obj = MyClass()
obj.greet()
```

```
Hello!
```

Here, greet is a method of the MyClass class that prints a greeting message. When you call the greet method on an instance of MyClass, the self argument refers to that instance.

Q5. What is the purpose of the init method?

In Python, the __init__ method is a special method that is called when an object is created. It is used to initialize the attributes of the object and set them to the desired starting values. the __init__() function uses to assign the values to object properties, or other operations that are necessary to do when the object is being created:

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

To create an instance of a class in Python, you need to follow these steps: > Define the class using the class keyword. Inside the class definition, you can define the attributes and methods of the class. > Create an object of the class by calling the class name as if it were a function. This will create a new instance of the class and return a reference to the object.

In [7]:
```
#Here is an example of creating a class and an instance of that class:
class MyClass:
    def __init__(self, value):
        self.value = value

obj = MyClass(10)
```

Q7. What is the process for creating a class?

To create a class in Python, you need to follow these steps: > Use the class keyword to define a new class, followed by the name of the class. > Define the attributes of the class using the self keyword. The self keyword refers to the instance of the class, and it is used to access and modify the attributes of the class. > Define the methods of the class using the def keyword. The methods of the class are functions that are defined inside the class definition and that operate on the attributes of the class.

In [8]:
```python
#Here is an example of a class definition in Python:

class MyClass:
    def __init__(self, value):
        self.value = value

    def increment(self):
        self.value += 1
```

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

A subclass is a class that derives from another class. A subclass inherits state and behavior from all of its ancestors. The term superclass refers to a class's direct ancestor as well as all of its ascendant classes. > 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?

A Module is a file that contains code that can be imported and used in other Python code. A module can contain any number of classes, functions, variables, and other code. A class is a template or blueprint for creating objects. It defines the attributes and behavior of a type of object, and it can be used to create as many objects of that type as you like. You can define a class in a module, and then import the module and use the class to create objects.

Q10. How do you make instances and classes?

To create a class in Python, we can use the `class` keyword followed by the name of the class and a colon. Then, you can define the attributes and methods of the class using the `self` keyword to refer to the instance of the class.

In [9]:
```python
class MyClass:
    def __init__(self, value):
        self.value = value

    def increment(self):
        self.value += 1
```

Q11. Where and how should be class attributes created?

A class attribute is shared by all instances of the class. To define a class attribute, you place it outside of the __init__() method

---

Q12. Where and how are instance attributes created?

Instance attributes are defined in the constructor. Defined directly inside a class. Defined inside a constructor using the self parameter.

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 belongs to the class.

Q14. How does a Python class handle operator overloading?

Operator overloading allows you to define the behavior of built-in operators (such as +, -, *, etc.) when they are applied to objects of a class. To overload a built-in operator in a Python class, you need to define special methods in the class with names that start and end with `__` and that contain the operator symbol as the middle part of the name. These methods are called "magic methods" or "dunder methods".

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

The ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +.

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

A very popular and convenient example is the Addition (+) operator. 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?

Both inheritance and polymorphism are fundamental concepts of OOP's python. We now have three different class and they all have a give_raise method. Although the name of the method is the same, it behaves differently for different type of objects. This is an example of polymorphism. Inheritance allows us to define a class that inherits all the methods and properties from another class.

Q18. Describe three applications for exception processing.

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.

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

If you don't do anything to handle an exception that is raised in your Python code, the exception will propagate up the call stack until it is caught by an exception handler or until it reaches the top of the call stack, at which point the program will terminate.

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

You can also 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.

Try – This method catches the exceptions raised by the program. Raise – Triggers an exception manually using custom exceptions. try: x = int(input("Enter a positive integer: ")) if x <= 0: raise ValueError("It is not a positive number!") except ValueError as val_e: print(val_e)

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

There are two ways to specify actions to be executed at termination time, regardless of whether or not an exception exists: 1. Using the `finally` clause in a `try`-`except` block: The finally clause in a try-except block allows you to specify a block of code that will always be executed, regardless of whether an exception is raised or not. For example: 2. Using the `atexit` module: The atexit module allows you to register a function to be executed at termination time, regardless of whether an exception exists. To use it, you can import the atexit module and use the register function to register a function

---

Q23. What is the purpose of the try statement?

The try statement allows you to define a block of code to be tested for errors while it is being executed.

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

In Python, the try statement is used to handle exceptions that may be raised in your code. There are two main variations of the try statement: 1. `try`-`except`: The try-except statement allows you to catch and handle exceptions that are 2. `try`-`except`-`else`: The try-except-else statement is similar to the try-except statement, but it also includes an else clause that is executed if no exception is raised in the

---

Q25. What is the purpose of the raise statement?

The raise statement is used to raise an exception. It allows you to signal that an error or exceptional condition has occurred in your code and to specify the details of the exception.

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

In Python, the raise statement is used to raise an exception. It allows you to signal that an error or exceptional condition has occurred in your code and to specify the details of the exception. Here is an example of using the raise statement in Python:

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

With statement is used in exception handling to make the code cleaner and much more readable.

Q28. What are *args, **kwargs?

*args specifies the number of non-keyworded arguments that can be passed and the operations that can be performed on the function in Python. **kwargs is a variable number of keyworded arguments that can be passed to a function that can perform dictionary operations.

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

collect the arguments using the * and ** in the function's parameter list. Through this, you will get the positional arguments as a tuple and the keyword arguments as a dictionary. Pass these arguments when calling another function by using * and **

Q30. What are Lambda Functions?

A lambda function can take any number of arguments, but can only have one expression. x = lambda a, b, c : a + b + c print(x(5, 6, 2))

Q31. Explain Inheritance in Python with an example?

Inheritance relationship defines the classes that inherit from other classes as derived, subclass, or sub-type classes

In [10]:
```
class Base1(object):
    def __init__(self):
        self.str1 = "str1"
        print("Base1")
class Base2(object):
```

```
            def __init__(self):
                self.str2 = "str2"
                print("Base2")
        class Derived(Base1, Base2):
            def __init__(self):
                Base1.__init__(self)
                Base2.__init__(self)
                print("Derived")
            def printStrs(self):
                print(self.str1, self.str2)
        ob = Derived()
        ob.printStrs()
```

```
Base1
Base2
Derived
str1 str2
```

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?

First it will search in amin class C nad then it start searching from left to right from Class A to Class B. So it will the func from Class A

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

The isinstance() method checks whether an object is an instance of a class. issubclass() method asks whether one class is a subclass of another class (or other classes).

In [11]:
```
    class MyClass(object):
      pass
    class MySubClass(MyClass):
      pass
    print(isinstance(MySubClass, object))
    print(issubclass(MySubClass, MyClass))
    print(isinstance(MySubClass, MyClass))
```

```
True
True
False
```

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.

Q35. What is the global keyword?

List can be created in python using []. lst = [] --> empty list lst = [1,2,3] --> numeric list lst = ['a', 'bc', 'd'] --> string list lst = ['abc', 'xyz', 1] --> alphanumeric listIn Python, the global keyword is used to indicate that a variable is a global variable, i.e., a variable that is defined outside of any function or class and is available in the global scope. Here is an example of using the global keyword in Python:

In [12]:
```
x = 10  # global variable

def func():
    global x
    x = 20  # modifying the global variable

func()
print(x)  # Output: 20
```

```
20
```

The End
