Object Oriented Programming

'list' object has no attribute 'upper'
'str' object has no attribute 'append'
In python there are two type of Classes:

1. Built-In Classes
2. User Defined Classes

Everything in python is Object.

procedural approach:

- In procedural programming, you write code in a linear way, focusing on procedures or steps to solve a problem. It's like following a recipe step by step.
- In object-oriented programming (OOP), you organize your code around objects that represent real-world things, like a car or a person. OOP helps make your code more organized, reusable, and easier to understand by grouping data and functions together. It's like organizing your tools into a toolbox instead of leaving them scattered around.

Go for specificity as per project

Kali Linux hackers use this operating system, all tools are inbuilt.

The main power of OOP is that the OOP gives the programmer the power to create his own data type.

Object-Oriented Programming (OOP) is a way of writing computer programs where you organize your code around "objects," which represent things in your program. These objects can have both data (like information about the object) and functions (things they can do). Think of objects as if they were real-world objects, like a car or a person, and you're writing code to describe and interact with them. OOP helps make your code more organized, easier to understand, and reusable.

OOPS consists of six principal:

1. Class
2. Object
3. Abstraction
4. Inheritance
5. Encapsulation
6. Polymorphism

CLASS and OBJECT

ALL String, Integer, Float, List, and Dictionary, are predefined classes
When created variable of that class, python call it as Object

Like : L = [1,2,3], the list is class while the L is the object

Class is blueprint, it is a set of rules, which decides/defines the behavior of its Object
Mobile Phone is Class and iPhone is the Object.
Class is the set of rules which an object follows
Class :

1. Data, Property, Attribute (Color, Body type of Car)
2. Function, Behaviour (Calculate average speed, efficiency of car)

Basic code of class:

```
Class Cars:
        Color = 'blue'        # data
        Model = 'sports'     #data

        Def calculate_average_speed(km,time):
                #somecodeing
```

Object is an instance of class

1. Car —> WagonR   WagonR = Car()
2. Sports —> Cricket   Cricket   = Sports()
3. Animal —> Cat   Cat   = Animal()
4. List() —---------->   L   L   = list()   ___we do the same in Python

To define a class, use Pascal Case which is : ToDefineClassUsePascalCaseWhichIs.
We will not get any output from class, until we do not create an object out of it.
Object_name = Class_name()

Constructor is a function in a class
Whenever we create a variable inside the class,

1. we have to create them inside the constructor.
2. The variable name should start with "self."

Constructor is a special function, as it has a super power.

We know, generally to execute a code from any function, we need to call that function, but here in constructor, we do not require to call the function to execute the code.
The code inside it automatically gets triggered as soon as we create an object out of that class.
What is the true benefit of constructor?
constructor is used to write configuration related code. Because the control of these code is not upto the user but the programmar.
The true benefit of a constructor is that it allows the programmer to set up initial values or configurations for an object automatically when it's created. This ensures that the object starts off in a valid state without requiring the user of the object to remember to initialize it correctly. It's like setting up a new game with all the pieces in the right places before you start playing, so everything works smoothly from the beginning.

Methods vs Function

The function within the class is called method, (it is indeed a function but we don't call it function, rather all it method)
And the functions outside the class are called Functions, these are independent in nature.
For an example:
L = [1,2,3]
len(L) —> Function >> it is outside the L list class
L.append(4) —> Method >> it is inside the L list class

- A function is a block of code that performs a specific task and can be called independently. It's like a standalone recipe that you can use whenever you need it.
- A method is a function that is associated with a particular object or class. It's like a special recipe that belongs to a specific cookbook (object or class). Methods can access and modify the data within the object they belong to.
-

Class Diagram

If **"-"** ahead of anything
it means it is
not visible outside the **class**
If **"+"** ahead of anything
it means it is
visible outside the **class**

ATM	**Class**
Pin Balance	**Data,Variables, Attributes**
Menu Create Pin Change Pin Check Balance Withdraw Exit	**Methods**

+	**Public**
-	**Private**

Magic Methods or Dunder Methods

Magic methods, also known as "dunder" (double underscore) methods, are special methods in Python that have double underscores before and after their names, like "**__init__**" or **"__str__**". These methods are called automatically by Python in certain situations.
For example, "__init__" is called when you create a new instance of a class to initialize it, while "__str__" is called when you try to convert an object to a string using "print()" or "str()".
Think of them as behind-the-scenes helpers that Python uses to perform common tasks like object initialization, string conversion, comparison, and more. They make it easier for you to work with objects in Python by providing default behaviors for certain operations.

If GOD is a Programmer, Earth is Class, what is in the Constructor Class, assuming we Human Beings are the Object?

Death

In Python you can't rename constructor (example **__init__** will always be __init__)

Self

Where is self used:

1. The default parameter for every method created
2. In the beginning of every attribute/variables in the constructor
3. Whenever calling for any method, it will start with self.

Self is like "Shree" "Shree Shree"

Golden rule of Object Oriented Programming
Class Object
Rule: 1. attribute addition
2. Addition of methods

Golden Rule: The only thing that can access all the things from the Class is the Object.
Self is nothing but the current object itself, we can prove it by showcasing the id of both, the location of both will precisely be the same.
Why to use self:
As the golden rule of OOP says, there can't be any communication between the

different functions of the class , so to establish this communication, we need **an object of the class inside the class,** for this sole purpose, we need the “self”.
Self is just a name, we can use any word!
Gangadhar he Shaktiman hai!

Parameterized Constructor
A parameterized constructor is one that accepts parameters during object creation to initialize its attributes with specific values.

1. **"__init__":** Initializes newly created objects.
2. **"__str__":** Returns a string representation of the object.
3. **"__len__":** Returns the length of the object.
4. **"__getitem__":** Gets an item from the object using square brackets.
5. **"__setitem__":** Sets an item in the object using square brackets.
6. **"__delitem__":** Deletes an item from the object using square brackets.
7. **"__iter__":** Returns an iterator for the object.
8. **"__next__":** Returns the next item from the iterator.
9. **"__contains__":** Checks if an item is contained in the object.
10. **"__eq__":** Checks if two objects are equal.
11. **"__lt__":** Checks if one object is less than another.
12. **"__gt__":** Checks if one object is greater than another.
13. **"__add__":** Add two objects together.
14. **"__sub__":** Subtracts one object from another.
15. **"__mul__":** Multiplies an object by another.
16. **"__div__":** Divides an object by another.
17. **"__call__":** Makes an object callable like a function.
18. **"__getattr__":** Gets an attribute of the object.
19. **"__setattr__":** Sets an attribute of the object.
20. **"__delattr__":** Deletes an attribute of the object.

Reference Variables

```
PP = Person('Darshan','India')
# here Person('Darshan','India') is an Object
# and PP is the reference of that object so we can say, PP is the
Reference Variable
```

Reference Variables holds the object
We can create Objects without Reference Variable
An Object can have multiple Reference Variables
Assigning a new Reference Variable to existing Object does not create a new Object

Just like list, set and dictionaries, user defined methods are mutables

Instance Variable

A variable whose values are different for different objects.

```
class Person:

  def __init__(self,name,gender):
    self.name = name
    self.gender = gender
```

Instance variables are variables that are tied to a specific instance of a class. Each instance of the class can have its own values for these variables. They are defined within methods of the class and are prefixed with `self.` to indicate that they belong to the instance. Instance variables store data that is unique to each object created from the class, allowing each object to maintain its state independently.

Encapsulation

__balance becomes _Atm__balance in memory

Encapsulation in Python involves bundling the data (attributes) and methods (functions) that operate on the data within a single unit, typically a class. This shields the internal implementation details of a class from external access, ensuring data integrity and security. Access to the class's attributes and methods is controlled through defined interfaces, allowing for data abstraction and hiding complexity. Encapsulation promotes modularity, enabling easier maintenance and updates to the codebase. By encapsulating related data and behavior, Python facilitates building robust and scalable applications while enhancing code readability, reusability, and maintainability.

Static Variable

Instance Variable are the variable of Object, **Static Variable** are the variable of Class
Instance Variable value for every object is different.
Static Variable value for every object remains the same.
If self is ahead of the variable then it is Instance Variable
If classname is ahead of the variable then it is Static Variable.
Instance variables are unique to each object created from a class. Each object can have its own values for these variables, and they belong to that specific object. For example, if you have a class representing a car, each car object may have its own color or speed.

Static variables, on the other hand, are shared among all instances of a class. They are not tied to any particular object but belong to the class itself. For example, if you have a class representing a car with a static variable for the number of wheels, all car objects will share the same value for the number of wheels.

OOP -what, why
Class Object Constructor
Encapsulation
Static Variables, Reference Variables, Instance Variable
Inheritance
Polymorphism
Abstraction
Project
Interview Questions
Exceptional Handling, Modules, Packages

Class Relationship

- Aggregation
- Inheritance

Aggregation - has the relationship One class owns the other class

Customer has Address , Restaurant has Menu

Customer Class owns Address Class

Restaurant Class owns Menu Class

Aggregation in Class Relationships

Aggregation is a fundamental concept in object-oriented programming that describes a "has-a" relationship between classes. Unlike composition, where one class is a part of another and cannot exist independently, aggregation signifies a looser coupling, indicating that one class contains a reference to another class, but the two can exist independently.

In an aggregation relationship, the containing class (often termed the "whole") maintains a reference to the contained class (the "part"). This relationship implies that the part class can belong to multiple whole classes simultaneously and can exist without being a part of any specific whole class instance.

Key characteristics of aggregation:

1. Multiplicity: Unlike composition, where the contained class instance is unique to the containing class instance, aggregation allows multiple instances of the containing class to reference the same instance of the contained class.
2. Independence: In aggregation, the contained class can exist independently of the containing class. It doesn't depend on the lifecycle of the containing class instance.
3. Flexibility: Aggregation provides flexibility in designing class relationships, allowing for a modular and reusable design. It enables building complex systems by composing smaller, independent components.
4. Weak Ownership: The containing class does not own the contained class instance in an aggregation relationship. It merely maintains a reference to it. As a result, the lifecycle of the contained class is not tied to the lifecycle of the containing class.

	Customer
-	name
-	gender
-	address
+	Print_Address
+	Edit_Address

	Address
-	city
-	pin
-	state
+	get_city
+	edit_address

Inheritance

Inheritance, a fundamental concept in object-oriented programming (OOP), plays a pivotal role in establishing relationships between classes. It embodies the principle of reusability and facilitates the creation of hierarchical structures within codebases. Here's a concise exploration of this essential relationship:

What is Inheritance?

Inheritance is a mechanism in OOP that allows a new class (derived class or subclass) to inherit properties and behaviors (methods and fields) from an existing class (base class, superclass, or parent class). This process enables the subclass to reuse, extend, or modify the functionality of the superclass, promoting code reusability and maintainability.

Key Characteristics:

1. Code Reusability: Inheritance encourages the reuse of existing code by enabling subclasses to inherit attributes and behaviors from their parent classes. This reduces redundancy and promotes modularization.
2. Hierarchical Structure: Inheritance facilitates the creation of class hierarchies, where subclasses can further specialize or extend the functionality of their parent classes. This hierarchical structure enhances code organization and understanding.
3. Single and Multiple Inheritance: Depending on the programming language, inheritance can be single or multiple. Single inheritance allows a class to inherit from only one superclass, while multiple inheritance permits a class to inherit from multiple superclasses. However, multiple inheritance can lead to complexities and is not supported in all programming languages.

Code Reusability

1. Child Class can not access Private members from Parents Class
2. Child Class can not access Private Class if it has its own Constructor
3. Child Class can access Private Class if it has its own Constructor

Method Overriding

Child Class and Parent Class have methods with the same names, the execution will always be of Child Class.

Constructor Overriding

Always asked keywords on OOP: 1. Self 2. Static 3. Super

To trigger a Parent Constructor, we have to use the **super()** keyword if we have a Child Constructor too.

We can call any method from the Parent from Child Constructor using the super() keyword.

Super keyword is always used in Class. Generally used in Child Class.

We can not access Attributes using super, We can only access Methods.

1) super cannot access variables
2) super cannot be used outside the class
3) super() is used in child class

- A class can inherit from another class.
- Inheritance improves code reuse
- Constructor, attributes, methods get inherited to the child class
- The parent has no access to the child class
- Private properties of parent are not accessible directly in child class
- Child class can override the attributes or methods. This is called method overriding
- super() is an inbuilt function which is used to invoke the parent class methods and constructor

Types of Inheritance

1. **Single Inheritance**
2. **Multilevel Inheritance**
3. **Hierarchical Inheritance**
4. **Multiple Inheritance(Diamond Problem)**
5. **Hybrid Inheritance**

Diamond Problem

MRO - method resolution order

Polymorphism

- **Method Overriding** > if we have same name object in Parent and Child Class, Child class object will get executed
- **Method Overloading** > If we can have more than one method by the same name in the same Class but behavior of both are different. It will execute the one which has the same number of input as required in the respective Method. In Python,

this does not work. One of the ways to implement it is by using if-else statements.

- **Operator Overloading >** same operator reflects/behave differently on different type of input