PROGRAM CODE CLUB

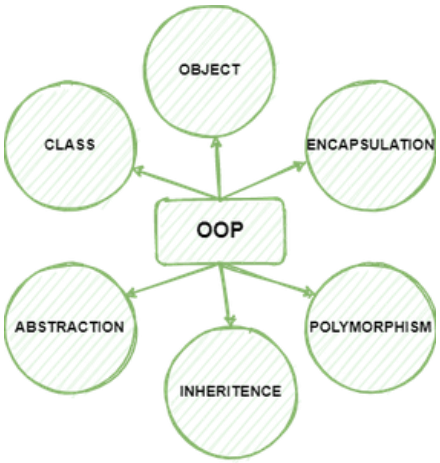
PYTHON OOPS INTERVIEW QUESTION

Introduction of Python OOPS:

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.

Main Concepts of Object-Oriented Programming (OOPs)

Class
Objects
Polymorphism
Encapsulation
Inheritance
Data Abstraction



What Is Class?

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator.

```
class ClassName:

  # Statement-1

  # Statement-N
```

What Is Objects?

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

The object is an entity that has a state and behaviour associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number 12 is an object, the string "Hello, world" is an object, a list is an object that can hold other objects, and so on. You've been using objects all along and may not even realize it.

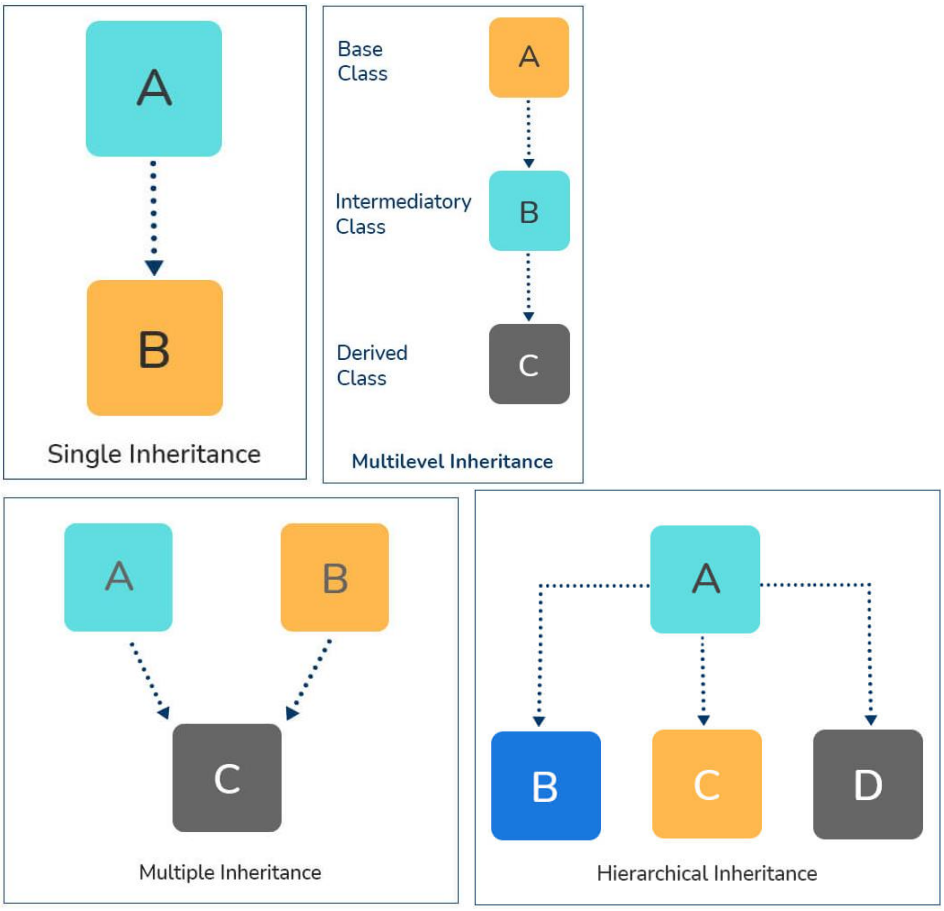
An object consists of:

- **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
- **Behaviour:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

What is Inheritance? How does inheritance work in python? Explain it with an example.

Inheritance gives the power to a class to access all attributes and methods of another class. It aids in code reusability and helps the developer to maintain applications without redundant code. The class inheriting from another class is a child class or also called a derived class. The class from which a child class derives the members are called parent class or superclass.

Python supports different kinds of inheritance, they are:



- **Single Inheritance**: Child class derives members of one parent class.

```
# Parent class
class ParentClass:
    def par_func(self):
         print("I am parent class function")

# Child class
class ChildClass(ParentClass):
    def child_func(self):
         print("I am child class function")

# Driver code
obj1 = ChildClass()
obj1.par_func()
obj1.child_func()
```

- **Multi-level Inheritance:** The members of the parent class, A, are inherited by child class which is then inherited by another child class, B. The features of the base class and the derived class are further inherited into the new derived class, C. Here, A is the grandfather class of class C.

```
# Parent class
class A:
    def __init__(self, a_name):
        self.a_name = a_name

# Intermediate class
class B(A):
    def __init__(self, b_name, a_name):
        self.b_name = b_name
        # invoke constructor of class A
        A.__init__(self, a_name)

# Child class
class C(B):
    def __init__(self,c_name, b_name, a_name):
        self.c_name = c_name
        # invoke constructor of class B
        B.__init__(self, b_name, a_name)

    def display_names(self):
        print("A name : ", self.a_name)
        print("B name : ", self.b_name)
        print("C name : ", self.c_name)

#  Driver code
obj1 = C('child', 'intermediate', 'parent')
print(obj1.a_name)
obj1.display_names()
```

- **Multiple Inheritance:** This is achieved when one child class derives members from more than one parent class. All features of parent classes are inherited in the child class.

```
# Parent class1
class Parent1:
    def parent1_func(self):
        print("Hi I am first Parent")

# Parent class2
class Parent2:
    def parent2_func(self):
        print("Hi I am second Parent")

# Child class
class Child(Parent1, Parent2):
    def child_func(self):
        self.parent1_func()
        self.parent2_func()

# Driver's code
obj1 = Child()
obj1.child_func()
```

- **Hierarchical Inheritance:** When a parent class is derived by more than one child class, it is called hierarchical inheritance.

```
# Base class
class A:
     def a_func(self):
         print("I am from the parent class.")

# 1st Derived class
class B(A):
     def b_func(self):
         print("I am from the first child.")

# 2nd Derived class
class C(A):
     def c_func(self):
         print("I am from the second child.")

# Driver's code
obj1 = B()
obj2 = C()
obj1.a_func()
obj1.b_func()     #child 1 method
obj2.a_func()
obj2.c_func()     #child 2 method
```

What is Encapsulation ?

Encapsulation is one of the key features of object-oriented programming. Encapsulation refers to the bundling of attributes and methods inside a single class. It prevents outer classes from accessing and changing attributes and methods of a class. This also helps to achieve data hiding.

What is Polymorphism?

Polymorphism is another important concept of object-oriented programming. It simply means more than one form. That is, the same entity (method or operator or object) can perform different operations in different scenarios.

What is Data Abstraction?

It hides the unnecessary code details from the user. Also, when we do not want to give out sensitive parts of our code implementation and this is where data abstraction came.

Data Abstraction in Python can be achieved through creating abstract classes.

Difference between Method Overloading and Method Overriding in Python

Method Overloading:

Method Overloading is an example of Compile time polymorphism. In this, more than one method of the same class shares the same method name having different signatures. Method overloading is used to add more to the behaviour of methods and there is no need of more than one class for method overloading.

Note: Python does not support method overloading. We may overload the methods but can only use the latest defined method.

Method Overriding:

Method overriding is an example of run time polymorphism. In this, the specific implementation of the method that is already provided by the parent class is provided by the child class. It is used to change the behaviour of existing methods and there is a need for at least two classes for method overriding. In method overriding, inheritance always required as it is done between parent class(superclass) and child class (child class) methods.

#	Method Overloading	Method Overriding
1.	In the method overloading, methods or functions must have the same name and different signatures.	Whereas in the method overriding, methods or functions must have the same name and same signatures.
2.	Method overloading is a example of compile time polymorphism.	Whereas method overriding is a example of run time polymorphism.
3.	In the method overloading, inheritance may or may not be required.	Whereas in method overriding, inheritance always required.
4.	Method overloading is performed between methods within the class.	Whereas method overriding is done between parent class and child class methods.

#	Method Overloading	Method Overriding
5.	It is used to add more to the behaviour of methods.	Whereas it is used to change the behaviour of exist methods.
6.	In method overloading, there is no need of more than one class.	Whereas in method overriding, there is need of at least of two classes.

Matplotlib

Matplotlib is the oldest and most widely used Python library for data visualization. It was created by neurobiologist **John D. Hunter** to plot data of electrical activity in the brains of epilepsy patients, but today is used in several fields.

When analysts and data scientists use matplotlib, they're usually using it in tandem with other Python libraries. Matplotlib is designed to work with NumPy, a numerical mathematics library and is a core part of the SciPy stack—a group of scientific computing tools for Python.

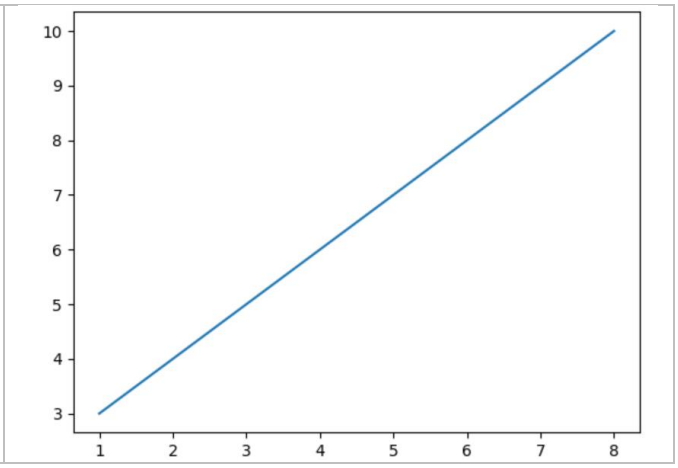
Some libraries like pandas and Seaborn are "wrappers" over matplotlib—they allow you to access several matplotlib's methods with less code. For instance, pandas. plot()' combines multiple matplotlib methods into a single method, so you can plot a chart in a few lines.

The large amount of code required in matplotlib to generate a nice-looking plot is often its biggest criticism. As a result, other visualization libraries like Seaborn, Bokeh, and plotly have emerged.

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```



Numpy

NumPy is a scientific computing library for Python. It offers high-level mathematical functions and a multi-dimensional structure (known as ndarray) for manipulating large data sets.

While NumPy on its own offers limited functions for data analysis, many other libraries that are key to analysis—such as SciPy, matplotlib, and pandas are heavily dependent on NumPy. SciPy, for instance, offers advanced mathematical functions built on top of NumPy's array data structure, **ndarray**.

NumPy, along with the libraries mentioned above, is a part of the core SciPy stack—a group of tools for scientific computing in Python.

NumPy features

Linear algebra

- Computing the eigenvalues of a matrix
- Manipulating linear matrices
- Vectorization

Statistics

- Finding the min, max, and percentiles of a dataset
- Calculating averages and variances of a dataset, such as the mean, median, and standard deviation
- Computing the histogram of a dataset

E.g.:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr) #output  [1 2 3 4 5]
```

Pandas

Pandas is a Python library for data analysis. Started by Wes McKinney in 2008 out of a need for a powerful and flexible quantitative analysis tool, pandas have grown into one of the most popular Python libraries. It has an extremely active community of contributors.

Pandas is built on top of two core Python libraries—matplotlib for data visualization and NumPy for mathematical operations. Pandas acts as a wrapper over these libraries, allowing you to access many of matplotlib's and NumPy's methods with less code. For instance, pandas' .plot() combines multiple matplotlib methods into a single method, enabling you to plot a chart in a few lines.

Pandas data structures

Series

You can think of a series as a single column of data. Each value in the series has a label, and these labels are collectively referred to as an index. This is demonstrated in the output below. 0-4 is the index and the column of numbers to the right contain the values.

0 22

1 27

2 31

3 33

4 34

DataFrames

While series are useful, most analysts work with the majority of their data in DataFrames. DataFrames store data in the familiar table format of rows and columns, much like a spreadsheet or database. DataFrames makes a lot of analytical tasks easier, such as finding the averages per column in a dataset.

You can also think of DataFrames as a collection of series—just as multiple columns combined make up a table, multiple series make up a DataFrame.

	home_page_visits	like_messages	messages	searches
0	784	492	292	102
1	793	500	287	106
2	253	172	110	40
3	134	95	55	33
4	501	331	182	119

Note: In Mode, the results of your SQL queries are automatically converted into DataFrames and made available in the list variable "datasets." To describe or transform the results of Query 1, use datasets [0], for the results of Query 2, use datasets[1] and so on.

For more on manipulating pandas data structures, check out Greg Reda's three-part tutorial, which approaches the topic from a SQL perspective.

Pandas features

Time series analysis

- Time Series / Date functionality (Official Pandas Documentation)
- Times series analysis with pandas (EarthPy)
- Timeseries with pandas (Jupyter)
- Complete guide to create a Time Series Forecast (with Codes in Python) (Analytics Vidhya)

split-apply-combine

Split-apply-combine is a common strategy used during analysis to summarize data—you split data into logical subgroups, apply some function to each subgroup, and stick the results back together again. In pandas, this is accomplished using the groupby() function and whatever functions you want to apply to the subgroups.

- Group By: split-apply-combine (Official Pandas Documentation)
- Summarizing Data in Python with Pandas (Brian Connelly)
- Using Pandas: Split-Apply-Combine (Duke University)

Data visualization

- Visualization (Official Pandas Documentation)
- Simple Graphing with IPython and Pandas (Chris Moffitt)
- Beautiful Plots With Pandas and Matplotlib (The Data Science Lab)

Pivot tables

- Reshaping and Pivot Tables (Official Pandas Documentation)
- Pandas Pivot Table Explained (Chris Moffitt)

Python standard library

- **TensorFlow:** This library was developed by Google in collaboration with the Brain Team. It is an open-source library used for high-level computations. It is also used in machine learning and deep learning algorithms. It contains a large number of tensor operations. Researchers also use this Python library to solve complex computations in Mathematics and Physics.
- **SciPy:** The name "SciPy" stands for "Scientific Python". It is an open-source library used for high-level scientific computations. This library is built over an extension of Numpy. It works with Numpy to handle complex computations. While Numpy allows sorting and indexing of array data, the numerical data code is stored in SciPy. It is also widely used by application developers and engineers.
- **Scrapy:** It is an open-source library that is used for extracting data from websites. It provides very fast web crawling and high-level screen scraping. It can also be used for data mining and automated testing of data.
- **Scikit-learn:** It is a famous Python library to work with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy and SciPy.
- **PyGame:** This library provides an easy interface to the Standard Directmedia Library (SDL) platform-independent graphics, audio, and input libraries. It is used for developing video games using computer graphics and audio libraries along with Python programming language.
- **PyTorch:** PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks.
- **PyBrain:** The name "PyBrain" stands for Python Based Reinforcement Learning, Artificial Intelligence, and Neural Networks library. It is an open-source library built for beginners in the field of Machine Learning. It provides fast and easy-to-use algorithms for machine learning tasks. It is so flexible and easily understandable and that's why is really helpful for developers that are new in research fields.