Python OOPs Concepts :-

- Object Oriented Programming is a fundamental concept in Python, empowering developers to build modular, maintainable, and scalable applications. By understanding the core OOP principles—classes, objects, inheritance, encapsulation, polymorphism, and abstraction—programmers can leverage the full potential of Python's OOP capabilities to design elegant and efficient solutions to complex problems.

OOPs Concepts in Python :-

- Class in Python
- Objects in Python
- Polymorphism in Python
- Encapsulation in Python
- Inheritance in Python
- Data Abstraction in Python

1. Class in Python :-

- 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.
- To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, and age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's the exact need for classes.

2.Objects in Python :-

- In object oriented programming Python, The object is an entity that has a state and behavior 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.

3.Polymorphism in Python :-

- **This code demonstrates the concept of Python oops inheritance and method overriding in Python classes. It shows how subclasses can override methods defined in their parent class to provide specific behavior while still inheriting other methods from the parent class.**

Example of polymorphism in python :-

```
class Bird:


   def intro(self):

      print("There are many types of birds.")


   def flight(self):

      print("Most of the birds can fly but some cannot.")


class sparrow(Bird):


   def flight(self):

      print("Sparrows can fly.")


class ostrich(Bird):


   def flight(self):

      print("Ostriches cannot fly.")


obj_bird = Bird()
```

```
obj_spr = sparrow()

obj_ost = ostrich()


obj_bird.intro()

obj_bird.flight()


obj_spr.intro()

obj_spr.flight()


obj_ost.intro()

obj_ost.flight()
```

Output :-

```
There are many types of birds.

Most of the birds can fly but some cannot.

There are many types of birds.

Sparrows can fly.

There are many types of birds.

Ostriches cannot fly.
```

4.Encapsulation in Python :-

- **In Python object oriented programming, Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object's** variable can only be changed by an object’s method. Those types of variables are known as private variables.

- A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.

Example of encapsulation in Python :-

```
class Base:

   def __init__(self):

      self.a = "GeeksforGeeks"

      self.__c = "GeeksforGeeks"

class Derived(Base):

   def __init__(self):

      Base.__init__(self)

      print("Calling private member of base class: ")

      print(self.__c)

obj1 = Base()

print(obj1.a)
```

Output :-

```
Geeks for Geeks
```

5.Inheritance in Python :-

- **In Python object oriented Programming, Inheritance is the capability of one class to derive or inherit the properties from another class. The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class.**

Types of Inheritance :-

- **Single Inheritance: Single-level inheritance enables a derived class to inherit characteristics from a single-parent class.**
- **Multilevel Inheritance: Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.**
- **Hierarchical Inheritance: Hierarchical-level inheritance enables more than one derived class to inherit properties from a parent class.**
- **Multiple Inheritance: Multiple-level inheritance enables one derived class to inherit properties from more than one base class.**

Example of Inheritance in Python :-

```
class Person(object):

  def __init__(self, name, idnumber):

    self.name = name

    self.idnumber = idnumber

  def display(self):

    print(self.name)

    print(self.idnumber)

  def details(self):

    print("My name is {}".format(self.name))

    print("IdNumber: {}".format(self.idnumber))

class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post
        Person.__init__(self, name, idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
        print("Post: {}".format(self.post))
a = Employee('Rahul', 886012, 200000, "Intern")
a.display()
a.details()
```

Output:-

```
Rahul
886012
My name is Rahul
IdNumber: 886012
Post: Intern
```

6.Data Abstraction in Python :-

- **It hides 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 by creating abstract classes.**

Example in Data Abstraction in Python :-

```
class Rectangle:

   def __init__(self, length, width):

      self.__length = length

      self.__width = width

   def area(self):

      return self.__length * self.__width

def perimeter(self):

      return 2 * (self.__length + self.__width)

rect = Rectangle(5, 3)

print(f"Area: {rect.area()}")

print(f"Perimeter: {rect.perimeter()}")

private attributes
```

Output :-

```
Area: 15

Perimeter: 16
```