OOP in Python

(Object-Oriented Programming)

- Object-Oriented Programming (OOP) is a programming paradigm that uses objects to organize and structure code.
- Python is a multi-paradigm programming language, and it supports object-oriented programming.
- OOP is a powerful way to design and structure your code, making it more modular, reusable, and maintainable.
- In Python, **everything is an object**, and you can use OOP principles to create and manipulate these objects.

OOPs Concepts in Python

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

1. Python 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.

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.

- 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. Eg.: Myclass.Myattribute

In [ ]:
```
class ClassName:
    # Statement-1
    .
    .
```

```
    .
    # Statement-N
```

Creating an Empty Class in Python

In [ ]:
```
# Python3 program to
# demonstrate defining
# a class

class Dog:
    pass
```

2. Python Objects

- 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.

In [ ]:
```
obj = Dog()
```

The Python self

When working with classes in Python, the term "self" refers to the instance of the class that is currently being used. It is customary to use "self" as the first parameter in instance methods of a class. Whenever you call a method of an object created from a class, the object is automatically passed as the first argument using the "self" parameter. This enables you to modify the object's properties and execute tasks unique to that particular instance.

In [ ]:
```
class mynumber:
    def __init__(self, value):
        self.value = value

    def print_value(self):
        print(self.value)

obj1 = mynumber(17)
obj1.print_value()
```

```
17
```

Python Class self Constructor

When working with classes, it's important to understand that in Python, a class constructor is a special method named **init** that gets called when you create an instance (object) of a class. This method is used to initialize the attributes of the object. Keep in mind that the self parameter in the constructor refers to the instance being created and allows you to access

and set its attributes. By following these guidelines, you can create powerful and efficient classes in Python.

In [ ]:
```
# _init_
class Subject:

    def __init__(self, attr1, attr2):
        self.attr1 = attr1
        self.attr2 = attr2


obj = Subject('Maths', 'Science')
print(obj.attr1)
print(obj.attr2)
```

```
Maths
Science
```