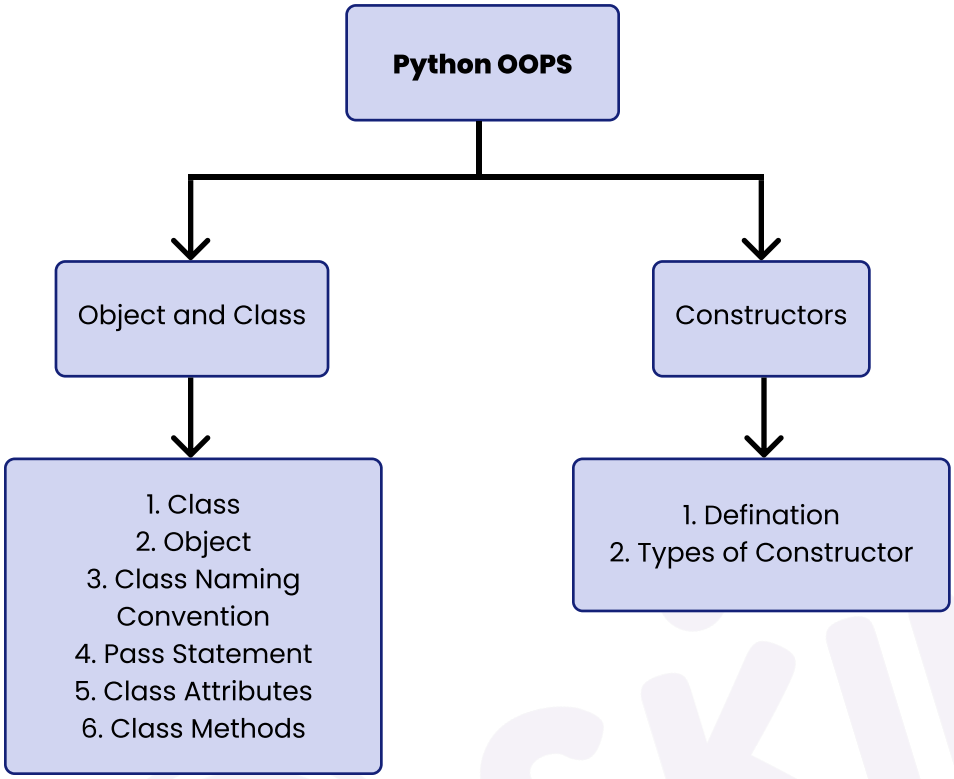
PW SKILLS

# Lesson Plan

# Objects and Classes

# Topic to covered:

- Python object and class
- Python Constructors




**Class:** The class is a user-defined data structure that combines the methods and data members into a single entity. For the production of objects, classes serve as blueprints or code templates. You can make as many objects as you desire using a class.

**Object:** An instance of a class is an object. It consists of a set of methods and attributes. Actions are carried out using a class's object. Objects have two qualities: They exhibit states and actions. (The object is equipped with properties and methods) Methods represent its behaviour, while attributes represent its state. We can alter its status by using its techniques.

Every object has following properties:

**Identity:** Any object needs to have unique identification.

**State:** An attribute of an object reflects both the properties and the state of the object.

**Behaviour:** Methods represent the behaviour of an object.

# Syntax of class:

```
# Syntax of class
class class_name:
  """Doc String"""
  statement 1
  statement 2
  ..........
```

**Ex:**

```
class Student:
  """Creating a class for Student details"""
  def __init__(self,name,roll_no):
    self.name=name
    self.roll_no =roll_no

  def show(self):
    print('Name: ',self.name,'Roll No. ',self.roll_no)

stud = Student('Alok',25) #object name is  stud
stud.show()
```

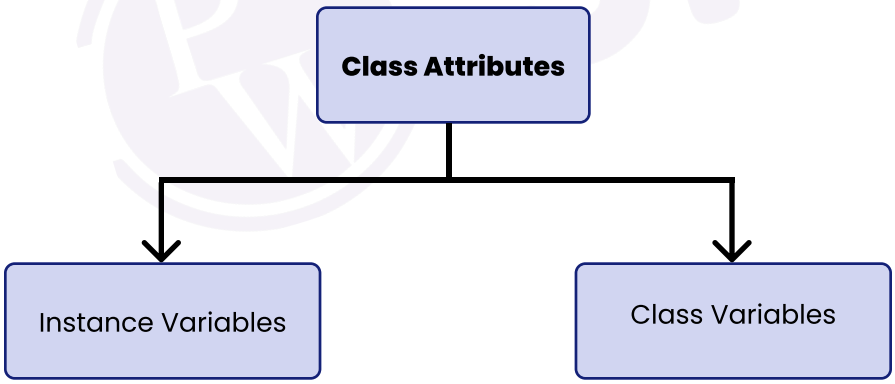
```
Name:  Alok Roll No.  25
```

# Class naming convention:

- UpperCamelCase should be used for class names.
- Classes that handle exceptions should end with error
- If a class can be called (called from somewhere), then we can give it a name similar to a function.
- The built-in classes in Python are often words with lowercase letters.

**Pass statement in class:** The pass statement in Python is a null statement. As a result, when the pass statement is run, nothing happens.

```
class Person:
  pass
```




1. **Instance variables:** Instance variables are properties that are joined to a class instance. In the constructor (the class's init() method), we define instance variables.

2. **Class Variables:** A variable that is declared inside a class but outside of any instance methods or init() methods is referred to as a class variable.

**Ex:**

```
class Employee_info:
    # class variables
    company_name = 'iNeuron'

    # constructor
    def __init__(self, name, age,emp_id):
        # instance variables
        self.name = name
        self.age = age
        self.emp_id = emp_id

emp1 = Employee_info("Aditya",29,1265)

# access instance variables
print('Details: \n',"Name:",emp1.name, "Age:",emp1.age,'Id: ',emp1.emp_id)

# access class variable
print('Company_name: ', Employee_info.company_name)

# Modify instance variables
emp1.name = 'Imran'
emp1.age = 28
emp1.emp_id= 2564

print('Details: \n',"Name:",emp1.name, "Age:",emp1.age,'Id: ',emp1.emp_id)

# Modify class variables
Employee_info.company_name = 'PwSkills'
print('Company name:', Employee_info.company_name)
```

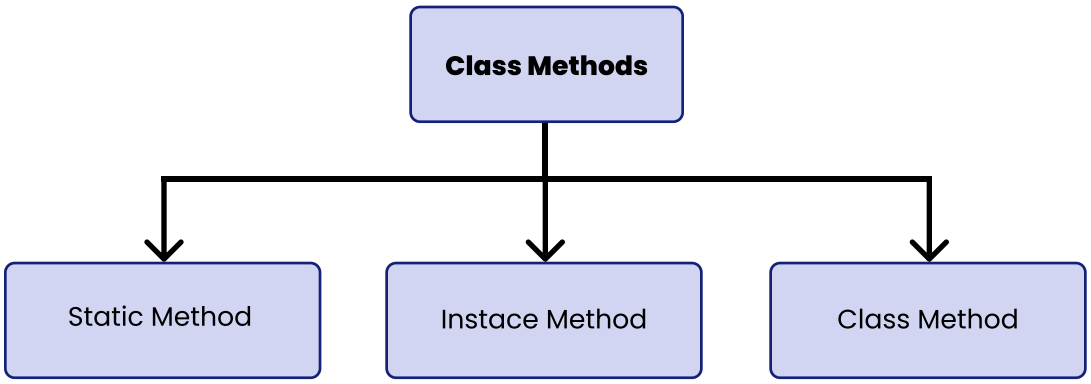
```
Details:
 Name: Aditya Age: 29 Id:  1265
Company_name:  iNeuron
Details:
 Name: Imran Age: 28 Id:  2564
Company name: PwSkills
```

**Accessing Multiple Objects:** Working with classes and objects is a common task in Python object-oriented programming (OOP). OOP access to numerous objects depends on the definition of your classes and the management of their instances.

**Ex:**

```
# Example

class Course:
    def __init__(self,name,price,course_duration):
        self.name = name
        self.price = price
        self.course_duration = course_duration

    def show(self):
        print("Course name:",self.name,",price is:",self.price,"and course duration is:", self.course_duration)

# creating object of the class
obj1 = Course("Full Stack Data Science BootCamp",25000,"1.5 years")

obj2 = Course("Full Stack Web Development",15000,'1 year')

#accesing multiple objects

total_course_obj = [obj1,obj2]
for obj in total_course_obj:
    obj.show()
```

```
Course name: Full Stack Data Science BootCamp ,price is: 25000 and course duration is: 1.5 years
Course name: Full Stack Web Development ,price is: 15000 and course duration is: 1 year
```




1. **Static Method:** It is a general utility method that handles a single task. Since this method is static and doesn't have access to class attributes, we don't use instance or class variables within it.
2. **Instance Method:** used to view or change an object's state. A method is referred to as an instance method if it uses instance variables.
3. **Class Method:** used to access or alter the class state. If we exclusively use class variables when implementing a method, we should declare that method as a class method.

```
class Teacher:
  def __init__(self,name):
    self.name = name

  def instance_method(self):
    print(f"This is instace method. Welcome, {self.name}")

  @classmethod
  def class_method(cls):
    print('This is class method')

  @staticmethod
  def static_method():
    print('This is static method.')

obj = Teacher('Rohit')
obj.instance_method()
obj.class_method()
obj.static_method()
```

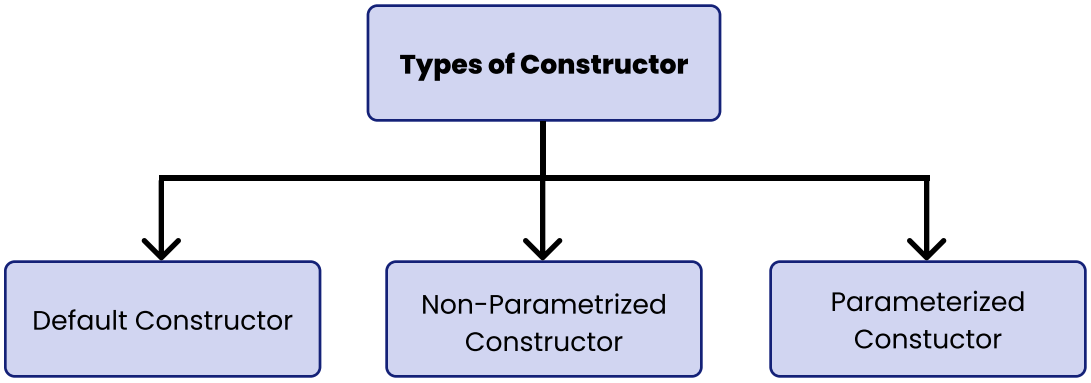
```
This is instace method. Welcome, Rohit
This is class method
This is static method.
```

**What is Constructor:** A constructor is a particular method used in object-oriented programming to generate and initialise an object of a class. In the class, this method is defined.

When an object is created, the constructor is automatically run.Declaring and initialising a class's instance and data member variables is the main function of the constructor. An object's attributes are initialised by the constructor, which is a collection of statements (i.e., instructions) that run when an object is created.

# Syntax of constructor :

```
def __init__(self):
  #body of constructor
```




**1. Default Constructor:** If no constructor is defined, Python will provide a default constructor. When we forget to define or include a constructor in a class, Python automatically adds a default constructor. While initialising the items, it does nothing else. It has no body and is an empty constructor.

```
[7]  class Person:

         def display(self):
             print('Inside Display')

     man = Person()
     man.display()

     Inside Display
```

**2. Non-Parameterized Constructor:** Non-parameterized constructors are those that take no parameters. Each object is given default settings when it is initialised using this kind of constructor. When creating an object, this constructor doesn't accept any arguments. It initialises each object with the same set of data instead.

```
[9]  class Car:
       def __init__(self):
         self.color='red'
         self.fuel_type = 'petrol'

       def show(self):
         print('Car color is :',self.color,' Car fuel type is :',self.fuel_type)

     maruti = Car()
     maruti.show()

     Car color is : red  Car fuel type is : petrol
```

**3. Parameterized Constructor :** A parameterized constructor is one with specified arguments or parameters. By utilising a parameterized constructor, we may give each object a different value when it is created. Self, a reference to the thing being produced, is the constructor's initial parameter. The programmer supplies the other arguments. Any number of arguments can be passed to a parameterized constructor.

```
[10] class Company:
       #parameterized constructor
       def __init__(self,name,location):
         self.name=name
         self.location = location

       def details(self):
         print('Comany name is',self.name,'and location is',self.location)

     obj = Company('iNeuron','Bengaluru')
     obj.details()

     Comany name is iNeuron and location is Bengaluru
```

PW SKILLS

# THANK YOU !