* **OOPS CONCEPTS**
* **OBJECT-ORIENTED PROGRAMING SYSTEM**
* **Object-oriented programing system (oops) is a concept of which is used to write the computer programs by using class and objects**
* **It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc.**
* **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**

* **Object**
* **Class**
* **Encapsulation**
* **Abstraction**
* **Inheritance**
* **Polymorphisms**

**Object**

* **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**.

**"An object consists of :**

* **State: It is represented by the attributes of an object. It also reflects the properties of an object.**
* **Behavior: 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.**
* **To understand the state, behavior, and identity let us take the example of the class dog (explained above).**
* **The identity can be considered as the name of the dog.  
  State or Attributes can be considered as the breed, age, or colour of the dog.**
* **The behavior can be considered as to whether the dog is eating or sleeping.**  
  **Creating an object:  
  obj = Dog()**
* **The \_\_init\_\_ method**
* **The \_\_init\_\_ method is similar to constructors in C++ and Java.**

* **It is run as soon as an object of a class is instantiated.**
* **The method is useful to do any initialization you want to do with your object**.

**REQUIREMENT :** **Sum of 2 given numbers**  
  
**STATE : - Data Initialization ==> Data types/data structures  
n1 = 10 int(input("Enter number1"))  
n2 = 20 int(input("Enter number2"))  
  
 BEHAVIOR - Implementation ==> Functions  
def get\_sum(num1, num2):  
 result = num1 + num2  
 return result  
  
print("Sum of 2 numbers is : ", get\_sum(n1, n2))**

## The \_\_init\_\_() Function

## class Person:

## def \_\_init\_\_(self, name, age):

## self.name = name

## self.age = age

## p1 = Person("John", 36)

## print(p1.name)

## print(p1.age)

**NOTE : The \_\_init\_\_() function is called automatically every time the class is being used to create a new object.**

## Object Methods

**class Person:**

**def \_\_init\_\_(self, name, age):**

**self.name = name**

**self.age = age**

**def myfunc(self):**

**print("Hello my name is " + self.name)**

**p1 = Person("John", 36)**

**p1.myfunc()**

**NOTE: Objects can also contain methods. Methods in objects are functions that belong to the object.**

## The self Parameter

* **The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.**
* **It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:**

## class Person:   def \_\_init\_\_(mysillyobject, name, age):     mysillyobject.name = name     mysillyobject.age = age   def myfunc(abc):     print("Hello my name is " + abc.name) p1 = Person("John", 36) p1.myfunc()

## Modify Object Properties

## class Person:

## def \_\_init\_\_(self, name, age):

## self.name = name

## self.age = age

## def myfunc(self):

## print("Hello my name is " + self.name)

## p1 = Person("John", 36)

## p1.age = 40

## print(p1.age)

## Delete Object Properties

**You can delete properties on objects by using the del keyword:**

**class Person:**

**def \_\_init\_\_(self, name, age):**

**self.name = name**

**self.age = age**

**def myfunc(self):**

**print("Hello my name is " + self.name)**

**p1 = Person("John", 36)**

**del p1.age**

**print(p1.age)**

## The pass Statement

**class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.**

**class Person:**

**def \_\_init\_\_(self, name, age):**

**self.name = name**

**self.age = age**

**def myfunc(self):**

**print("Hello my name is " + self.name)**

**p1 = Person("John", 36)**

**del p1.age**

**print(p1.age)**

**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.**

**Some points on Python class:**

* **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**

**class ClassName:**  
  **Statement1.  
 .  
 .  
 Statement-N"""**

**class structure  
  
class <class-name>:  
 *# 1. STATE* n1 = 10  
 n2 = 20**

***# 2. BEHAVIOR* def get\_sum(num1, num2):  
 result = num1 + num2  
 return result**

**Example :**

**class Student:  
  
 *# 1. STATE* def \_\_init\_\_(self, r\_no, name, marks):  
 self.r\_no = r\_no  
 self.name = name  
 self.marks = marks  
  
 *# 2. BEHAVIOR* def get\_sinfo(self):  
 print("Student details are ", self.r\_no, self.name, self.marks)  
  
madhu = Student(23, 'Madhu Nettem', 65)**

**madhu.get\_sinfo()**

**classmethod() in Python:**

**The classmethod() is an inbuilt function in Python, which returns a class method for a given function.  
  
Syntax: classmethod(function)**

**Parameter**:**This function accepts the function name as a parameter.**

**Return Type**:**This function returns the converted class method**.

**Syntax:**   
  
**@classmethod  
 def fun(cls, arg1, arg2, ...):  
   
   
 Where,   
  
fun: the function that needs to be converted into a class method  
returns: a class method for function.  
classmethod() methods are bound to a class rather than an object.  
 Class methods can be called by both class and object.  
 These methods can be called with a class or with an object.**

**Class method vs Static Method:**

* **A class method takes cls as the first parameter while a static method needs no specific parameters.**
* **A class method can access or modify the class state while a static method can’t access or modify it.**
* **In general, static methods know nothing about the class state. They are utility-type methods that take some parameters and work upon those parameters.**
* **On the other hand class methods must have class as a parameter.**
* **We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.**

**Instance variable :** **Individual to each object  
Class variable : Sharable and modifiable by all objects  
  
CV ----> CM  
IV ----> IM  
  
CV ----> IM Correct  
IV ----> CM XXX  
  
'''  
'''  
STATE   
 - Fields  
 - Class variables   
 - Instance variables   
   
BEHAVIOR  
 - Methods  
 - Class method  
 - Instance method  
 - Static method  
   
STATE - BEHAVIOR  
Fields - Methods**  
  
**OOPs concepts:**  
--------------------  
**Class Object  
1. Encapsulation  
2. Inheritance  
3. Polymorphism  
4. Abstraction**  
  
  
**Encapsulation : Binding the data members and   
 member methods into single entity  
 - data members ==> fields  
 - member methods ==> methods  
 Ex: Class, object  
   
 Logical Physical   
 Class Fields Methods  
 object Methods Fields  
   
Abstraction : Hiding implementation details and exposing only method signature   
  
Abstraction : Class - 0% abstraction  
 Abstract class - 0% to 100% abstraction  
 Interface - 100% abstraction**  
**class,object  
  
1. Encapsulation** :

**Definition: Binding the data members & member methods into single entity  
  
entity : class/object  
data members : Fields/Variables/Attributes (class,instance variables)  
member methods : Methods (Instance Methods,Class Methods,Static Methods)  
  
madhu = Employee(100,'Madhu N, 15000)  
Class ===> Logical -- DATA Physical -- METHODS  
Object ===> Physical -- DATA Logical -- METHODS (Through method access)  
  
Ex : class is an example for encapsulation  
 object is also an example  
  
  
madhu = Employee1(100,"MadhuN",15000)**  
**2. Abstraction :**  
  
**Hiding the implementation details in the methods of a class  
  
In a "Normal class" Abstraction is 0% *# all concrete methods*In "Abstract Class" Abstraction is 0-100 % *# 1. all concrete methods,* 2. all abstract methods  
 3. Comb of concrete+abstract methods  
In an "Interface" Abstraction is 100% *# all abstract methods*****During inheritance : Normalclass,**  
**3. Inheritance :**  
  
**super class, sub class mechanism**  
 **4. Polymorphism :**  
 **- Static Polymorphism -- Method overloading  
 - Dynamic Polymorphism -- Method overriding**  
 **1. Class Defined and provided special method i.e,   
 \_\_init\_\_(constructor) method to initialize instance variables,   
 define respective methods to get the BEHAVIOR**

**2. Create object for the respective class.**

**3. Python creates empty object for us,and gives reference to self** **parameter** **Reamining parameters, we are passing the arguments  
In empty object, instance variables will be initialized with the given data**

**4. Finally whole object reference will be given to LHS  
5. We can perform method calls using created object**

**class var instance var  
-------------------------- ----------------------------  
while loading class at the time of object creation  
  
class var instance var  
class methods instance methods  
  
 instance variables cant be used inside class method\*\*  
  
++ Within instance methods we can use class variables\*\*\*\*\*  
viceversa is not True ==> within class methods we can't use instance varibales**  
 **class Employee:  
 '''This class give details about employee'''  
 def \_\_init\_\_(self, eid, name, sal):  
 self.eid = eid  
 self.name = name  
 self.sal = sal  
  
 def get\_data(self):  
 print("Employee data : ", self.eid, self.name, self.sal)  
  
# Built in class attributes**   
**print("Employee.\_\_dict\_\_:", Employee.\_\_dict\_\_)  
print("Employee.\_\_doc\_\_:", Employee.\_\_doc\_\_)  
print("Employee.\_\_name\_\_:", Employee.\_\_name\_\_)  
print("Employee.\_\_module\_\_:", Employee.\_\_module\_\_)  
print("Employee.\_\_bases\_\_:", Employee.\_\_bases\_\_)**

***# CRUD - Create Retrieve Update Delete*madhu = Employee(100, "Madhu N", 10000) *# CREATE***

**setattr : To set IV value inside object # UPDATE  
getattr : To retrieve IV value # RETRIEVE   
hasattr : To check IV has specific value # RETRIEVE  
delattr : To delete IV # DELETE**   
  
***# getter - RETRIEVE*print("Madhu name :", getattr(madhu, "name"))  
print("Madhu eid :", getattr(madhu, "eid"))  
*# print("Madhu addr :", getattr(madhu, "addr"))  
  
# setter - UPDATE*print("Setting name :", setattr(madhu, "name", "MAD"))  
print("Setting eid :", setattr(madhu, "eid", 200))  
print("Setting addr :", setattr(madhu, "address", 'Bangalore'))**  
**print("Get name:", getattr(madhu, "name"))  
print("Get eid :", getattr(madhu, "eid"))  
print("Get addr:", getattr(madhu, "address"))  
  
print("Has attr sal :", hasattr(madhu, "sal"))  
print("Has attr addr:", hasattr(madhu, "address"))  
  
print("Delete attr :", delattr(madhu, "sal"))  
print(getattr(madhu, "sal"))**