OBJECT ORIENTED PROGRAMMING STRUCTURE

EVERTHING IN PYTHON IS AN OBJECT

WHAT IS OOPS?

- OOPS STANDS FOR OBJECT ORIENTED PROGRAMMING STRUCTURE.
- IN PYTHON, OBJECT ORIENTED PROGRAMMING IS A PROGRAMMING PARADIGM THAT USES OBJECTS & CLASSES IN PROGRAMMING.

WHAT IS THE POWER OF OOP? / WHAT OOP DO?

- THE MAIN POWER OF OOP,THAT OOP GIVES A PROGRAMM TO CREATE THEIR OWN DATA TYPE.

WHY WE USE OOP?

- IT MAKES CODE MORE REUSABLE & EASIER TO WORK WITH LARGER PROGRAMMS.
- OOP PROGRAMS PREVENT YOU FROM REPEATING CODE BECAUSE A CLASS CAN BE DEFINED ONCE & REUSED MANY TIMES.
- **IN SHORT :**
  - REUSABILITY
    - INHERITANCE
    - POLYMEORPHISM
  - SECURITY
    - ENCAPSULATION
    - ABSTRACTION

FOUR MAJOR CONCEPTS OF OOP:

1. INHERITANCE
2. POLYMORPHISM
3. ENCAPSULATION
4. ABSTRACTION

CONCEPTS OF OOP:

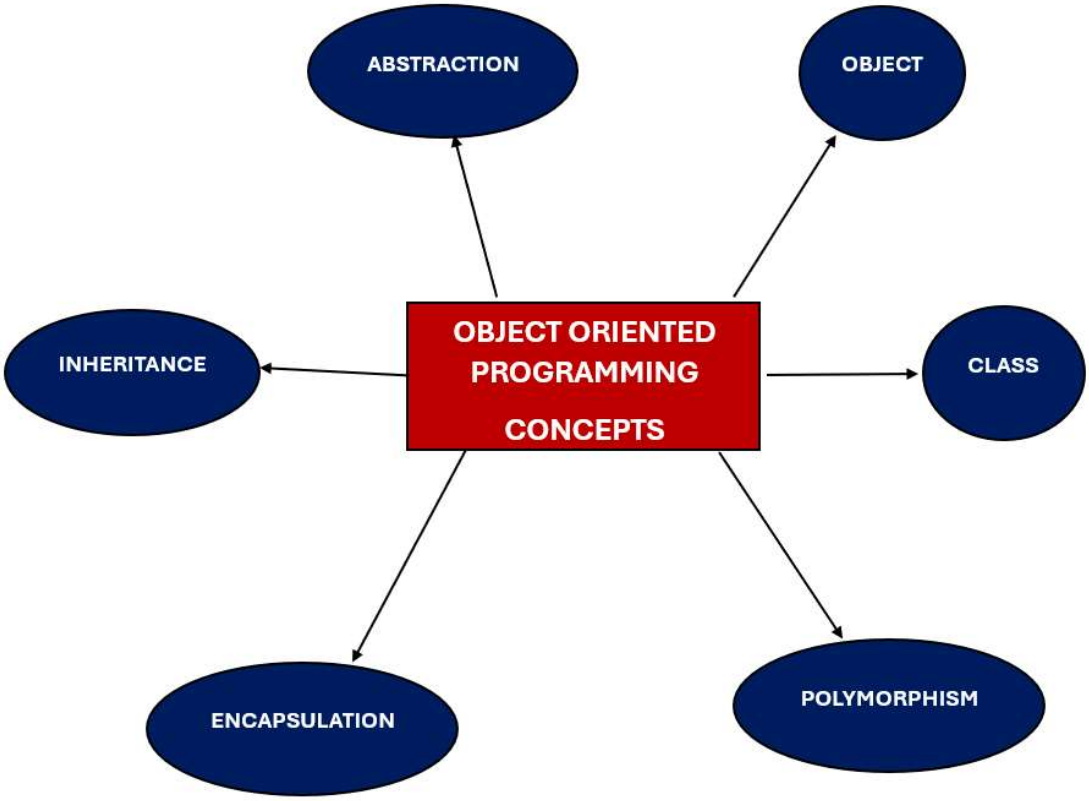
- CLASS
- OBJECT
- REFERENCE VARIABLE
- "self" KEYWORD
- TYPES OF VARIABLES

```
        * LOCAL
        * INSTANCE
        * STATIC
```

- TYPES OF METHODS

```
        * INSTANCE
        * CLASS
        * STATIC
```

- CONSTRUCTOR
- "super" KEYWORD
- NESTED CLASS



WHAT IS CLASS?

- IT IS A STRUCTURE WHICH CONSISTS OF `MEMBERS OF VARIABLES` & `MEMBERS OF METHODS` .
  - **VARIABLES** --> WHICH REFERS TO THE MEMORY LOCATION
  - **METHODS** --> IT'S A FUNCTION WITHIN THE CLASS.
- IT'S NOT A REAL WORLD ENTITY.
  - IT DOES NOT OCCUPY MEMORY.
- IT CAN BE CONSIDERED AS PROTOTYPE / TEMPLATE / BLUE-PRINT.

In [1]:
```
# eg

l=[1,2,3]
print(type(l))


# HERE LIST IS CLASS & "l" IS THE OBJECT OF THE CLASS LIST.
```

```
<class 'list'>
```

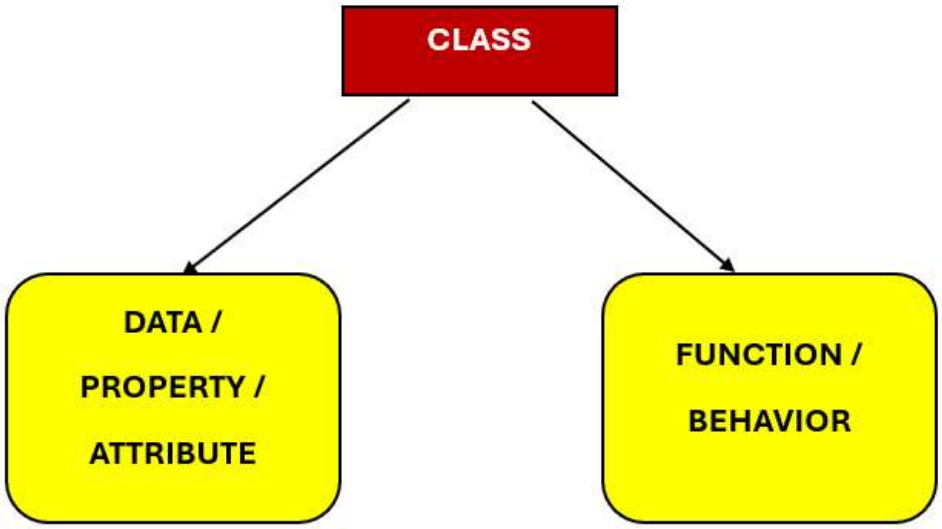
In [2]:
```
# eg

# THE STUDENTS IN THE MATH CLASS ROOM.
# --> HERE STUDENTS IS OBJECT & MATH CLASS ROOM IS CLASS.
```

In [3]:
```
# eg

# --> CARS IS CLASS
# --> MARUTI SUZUKI IS OBJECT.
```

In [4]:
```
# eg

# --> MOBILE PHONE IS CLASS.
# --> SAMSUNG SMARTPHONE IS OBJECT,
```

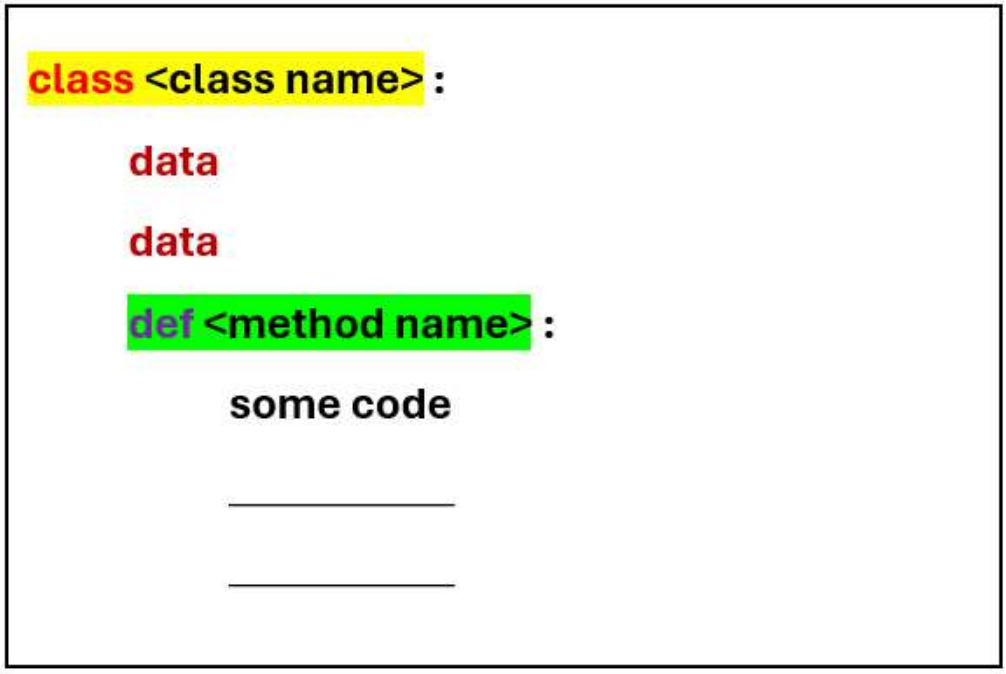
SUMMARY

- CLASS IS SET OF RULES THAT FOLLOWED BY OBJECTS.
- EVERY DATA TYPE IN PYTHON IS CLASS, WHEN WE CREATE A VARIABLE THEN THIS VARIABLE IS AN OBJECT.



In [5]:
```
# eg

# CARS

# DATA --> COLOR, TOP SPEED
# FUNCTION --> CALCULATE AVERAGE SPEED
```

SYNTAX

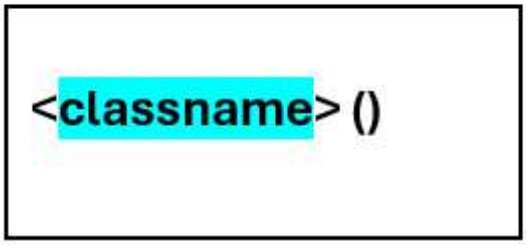


WHAT IS OBJECT?

- OBJECT ARE VARIABLE THAT CONTAIN DATA & FUNCTIONS THAT CAN BE USED TO MANIPULATE THE DATA.
- OBJECT IS AN INSTANCE OF THE CLASS.
- OBJECT IS A REAL WORLD ENTITY.
  - OBJECT OCCUPY MEMORY.
- FOR EACH OBJECT WE CAN DERIVE "n" NUMBER OF OBJECTS.
- OBJECT IS MANDATORY TO ACCESS MEMN=BERS OF THE CLASS, WITHOUT OBJECT WE CAN NOT ACCESS MEMBERS OF THE CLASS.
- IT IS A BUILT IN CLASS.

In [6]:
```
# eg

# car --> wagnor
# sport --> gilidanda
```

SYNTAX



In [7]:
```
# eg

# Create a Class

class my_class:
    city="Hyderabad"    # VARIABLE

    def greet(self):    # METHOD
        print("Welcome to first Class Method")
```

In [8]:
```
# Create a object og "my_class"

mcls=my_class()
```

In [9]:
```
# Call the Variable City
mcls.city
```

Out[9]:
```
'Hyderabad'
```

In [10]:
```
# Call the Method greet

mcls.greet  # This give the adress of the Method greet
```

Out[10]:
```
<bound method my_class.greet of <__main__.my_class object at 0x0000017F79A
89220>>
```

In [11]:
```
mcls.greet()
```

```
Welcome to first Class Method
```

MULTI VARIABLES & METHODS

In [12]:
```
# eg

# CREATE A CLASS

class my_class1:
    country="INDIA"  # VARIABLE-1
    state="ODISHA"   # VARIABLE-2
    city="BBSR"      # VARIABLE-3

    def m_add(self):   # METHOD-1
        num1=10
        num2=20
        a=num1+num2
        print(a)

    def m_mul(self):    # METHOD-2
        n1=20
        n2=30
        a1=n1*n2
        print(a1)

```

In [13]:
```
# CREATE AN OBJECT OF THAT CLASS

mycls=my_class1()
```

In [14]:
```
# CALL THE VARIABLE
mycls.city
```

Out[14]: 'BBSR'

In [15]:
```
mycls.state
```

Out[15]: 'ODISHA'

In [16]:
```
# CALL THE METHOD
mycls.m_add()
```

```
30
```

In [17]:
```
mycls.m_mul()
```

```
600
```

WHAT IS METHOD?

- METHOD IS A FUNCTIO THAT BELONG TO AN OBJECT / CLASS.
- THEY ARE ALWAYS CALL WITH AN OBJECT.
- IT'S ALWAYS HAVE A DEFAULT PARAMETER AS `"self"` .

WHAT IS FUNCTION?

- FUNCTIONS ARE INDEPENDENT.
- THEY ARE CALLING BY THEIR NAME.

IN SIMPLE LANGUAGE:

- IF YOU CREATE A FUNCTION INSIDE A CLASS THEN WE TELL THAT AS `METHOD` .
- IF YOU CREATE A FUNCTION OUTSIDE A CLASS THEN WE TELL THAT AS

In [18]:
```
# eg

l=[1,2,3,4]
```

In [19]:
```
len(l)
```

Out[19]: 4

In [20]:
```
l.append(5)
```

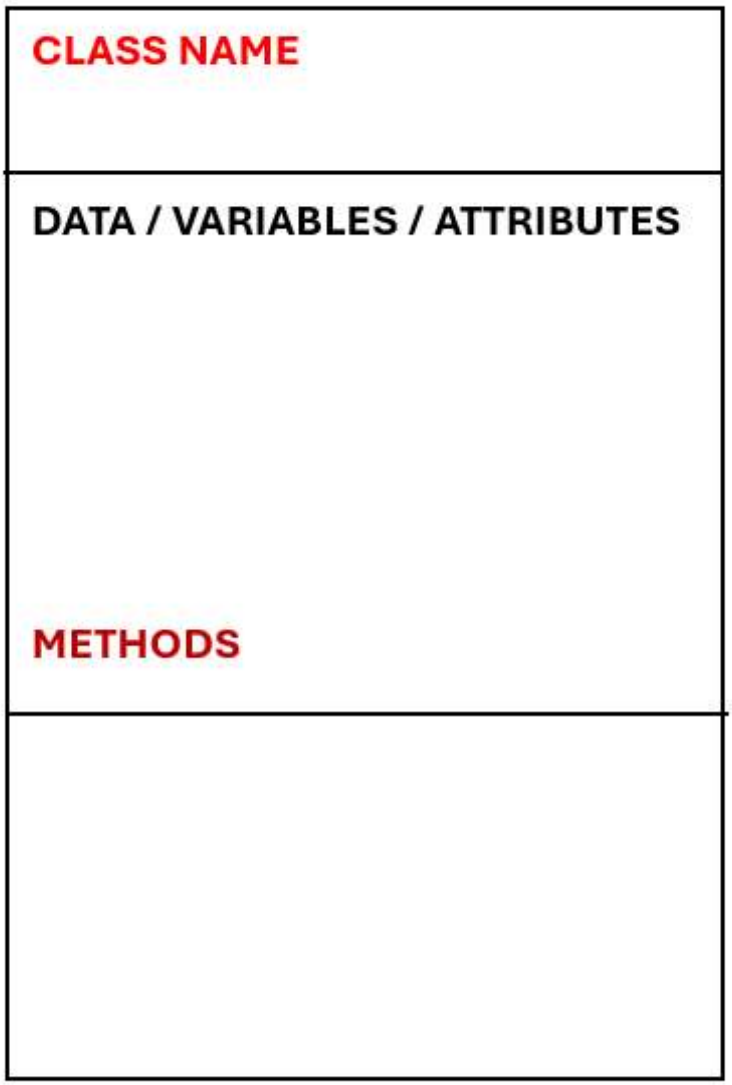
In [21]:
```
l
```

Out[21]: [1, 2, 3, 4, 5]

HERE "len" IS A FUNCTION & "append" IS A METHOD.

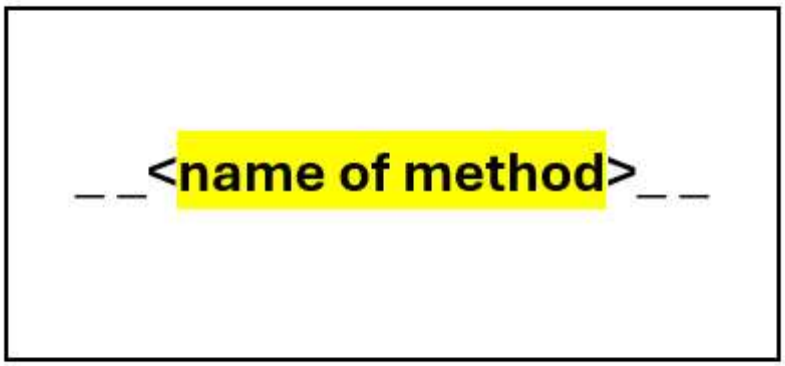
"len()" IS OUTSIDE THE LIST CLASS & "append()" IS INSIDE THE LIST CLASS.

CLASS DIAGRAM



MAGIC METHOD (A.K.A DUNDER METHOD)

- IT IS ALSO KNOWN AS DUNDER METHOD.
- IT IS A SPECIAL KIND OF METHOD.
- IT HAVE THEIR OWN POWER.
- BY USING MAGIC METHOD, WE CREATE OUR OWN DATA TYPES.



WHAT IS CONSTRUCTOR?

- IT IS A SPECIAL METHOD.
- IT AUTOMATICALLY CALL WHEN THE OBJECT IS CREATED.
- A METHOD WHICH CAN BE EXECUTED WITHOUT EXPLICITLY CALLING.
- CONSTRUCTOR IS A SPECIAL TYPE OF METHOD WHICH IS EXECUTED WHENEVER AN OBJECT IS CREATED.
- THESE CONSTRUCTORS ARE USED TO INITIALIZE THE VARIABLES WITH VALUES.
- NAME OF CONSTRUCTOR IS : `_ _ init _ _ ()`
- CONSTRUCTOR COMES UNDER THE INSTANCE METHOD CATEGORY, i.e. DEFAULT ARGUMENT IS `self` .
- ANY NUMBER OF CONSTRUCTOR CAN BE DEFINED WITHIN A CLASS BUT ONLY ONE IS EXECUTED i.e. **`LATEST ONE`** .

TRUE BENIFIT OF CONSTRUCTOR:

- TO INITIALIZE THE OBJECT.
- TO WRITE CONFIGURATION RELATED CODE.
- THAT MEANS IF WE WOULD LIKE TO CONNECT TO OUR DATABASE, TO OUR BACKEND SERVICE, WE WRITE THIS CODE INSIDE THE CONSTRUCTOR.
- IN SIMPLE, WE DO NOT GIVE THE CODE CONTROL BY USER. IN A CLASS EVERY FUNCTION IS CONTROLLED BY THE USER BUT THE CONSTRUCTOR IS NOT CONTROLLED BY THE USER.

- eg:
  - **QUES**:

    IF GOD IS THE PROGRAMMER, EARTH IS THE CLASS, HUMANE BEINGS IS THE OBJECT THEN WHAT IS THE CONSTRUCTOR?
  - **ANS**:
    - THE CONSTRUCTOR IS DEATH.
    - GOD DO NOT WANT TO GIVE THE DEATH TO CONTROL BY THE HUMAN.

In [22]:
```
# eg

# CREATE A CLASS
class temp:
    def __init__(self): # CONSTRUCTOR / FIRST MAGIC METHOD
        print("Hello")

```

In [23]:
```
# CREATE A OBJECT OF THAT CLASS
p=temp()

# WE DO NOT CALL THE CONSTRUCTOR,
# IT EXECUTED WHEN WE CREATED THE OBJECT.
```

```
Hello
```

TWO TYPES OF CONSTRUCTOR:

a. DEFAULT CONSTRUCTOR

- A CONSTUCTOR WITHOUT ANY ARGUMENTS.

In [24]:
```
# eg

# Create a Class

class con_class:

    def __init__(self):   # Constructor --> Default
        print("This is from constructor method")
        self.n1()  # we call the method "n1" inside the constructor

    def n1(self):  # Method
        print("This is from n1 method")
        num=5

        for i in range(1,11):
            print(num, "*", i, "==", (num*i))

```

In [25]:
```
# Create an Object
q=con_class()
```

```
This is from constructor method
This is from n1 method
5 * 1 == 5
5 * 2 == 10
5 * 3 == 15
5 * 4 == 20
5 * 5 == 25
5 * 6 == 30
5 * 7 == 35
5 * 8 == 40
5 * 9 == 45
5 * 10 == 50
```

b. PARAMETERIZED CONSTRUCTOR

- A CONSTRUCTOR WITH ARGUMENT.

In [26]:
```
# eg

# Create a Class
class byclass:

    def __init__(self,z):    # Constructor --> Parameterized
        print("This is from Constructor of class ayclass")
        self.n2(z)


    def n2(self,num):    # Method
        print("This is from N2 Instance Method")

        for i in range(1,11):
            print(num,"*",i,"==",num*i)

```

In [27]:
```
# Call the class
byclass(10)
```

```
This is from Constructor of class ayclass
This is from N2 Instance Method
10 * 1 == 10
10 * 2 == 20
10 * 3 == 30
10 * 4 == 40
10 * 5 == 50
10 * 6 == 60
10 * 7 == 70
10 * 8 == 80
10 * 9 == 90
10 * 10 == 100
```

Out[27]:
```
<__main__.byclass at 0x17f79ab3ee0>
```

In [28]:

```
# eg

class byclass:

    def __init__(self,z):
        print("This is from Constructor of class ayclass")
# The instance Varible "self.num" is not same as z so we assign it.
        self.num=z
        self.n2(z)


    def n2(self,num):
        print("This is from N1 Instance Method")

        for i in range(1,11):
            print(self.num,"*",i,"==",self.num*i)


byclass(12)
```

```
This is from Constructor of class ayclass
This is from N1 Instance Method
12 * 1 == 12
12 * 2 == 24
12 * 3 == 36
12 * 4 == 48
12 * 5 == 60
12 * 6 == 72
12 * 7 == 84
12 * 8 == 96
12 * 9 == 108
12 * 10 == 120
```

Out[28]:

```
<__main__.byclass at 0x17f79b44850>
```

In [29]:

```
# eg

# Latest one will be executed.

class zclass:

    def __init__(self):
        print("This is a Constructor")

    def __init__(self,u):
        print("This is a parameterized constructor")
        res=u*u
        print(res)

    def __init__(self,a,b):
        print("This is a parameterized constructor")
        res=a*b
        print(res)


```

In [30]:
```
1 obj=zclass()
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_3996\2938015046.py in <module>
----> 1 obj=zclass()

TypeError: __init__() missing 2 required positional arguments: 'a' and 'b'
```

In [ ]:
```
1 obj=zclass(5,6)
```

GOLDEN RULE OF OOP:
EVERY ATTRIBUTES & METHOD OF CLASS CAN ONLY ACCESSED BY ONLY THE OBJECT OF CLASS.

"self" KEYWOORD

- **self** REPRESENTS THE INSTANCE OF CLASS.
- IT ALLOWS US TO ACESS VARIABLES, ATTRIBUTES, & METHODS OF A DEFINED CLASS.
- IT REFERS TO CUREENT OBJECT.
- IT IS AN IDENTIFIER WHICH HOLDS OBJECT OF THE CLASS.
- IT HELPS TO IDENTIFY THE LOCAL & INSTANCE VARIABLES.
- IT GIVE THE POWER THAT IF WE WANT TO CALL A METHOD INSIDE ANOTHER METHOD IN THAT TIME WE USE **self** .
- **self** IS NOTHING BUT THE CURRENT OBJECT.

In [ ]:
```python
# eg

# Create a Class

class con_class:

    def __init__(self):   # Constructor --> Default
        print("This is from constructor method")
        self.n1()


    def n1(self):  # Method
        print("This is from n1 method")
        num=5

        for i in range(1,11):
            print(num, "*", i, "==", (num*i))
```

In [ ]:
```python
# Create an Object
w=con_class()
```

In [31]:
```python
# Create a Class

class con_class:

    def __init__(self):   # Constructor --> Default
        print("This is from constructor method")
#         self.n1()
        print(id(self))

    def n1(self):  # Method
        print("This is from n1 method")
        num=5

        for i in range(1,11):
            print(num, "*", i, "==", (num*i))
```

In [32]:
```python
w=con_class()
```

```
This is from constructor method
1647013723056
```

In [33]:
```python
id(w)
```

Out[33]:
```
1647013723056
```

In [34]:
```python
# memory location of "self" & object is same.
# that means the "self" is nothing but the object
```

2nd MAGIC METHOD:

_ _ str _ _ ()

- IF YOU PRINT THE OBJECT OF YOUR CLASS, THEN AUTOMATICALLY THIS IS EXECUTED.

3rd MAGIC METHOD:

__ add __ () --> "+"

- IT AUTOMATICALLY TRIGGER, WHEN YOU GIVE OPERAND "+" IN BETWEEN THE TWO OBJECTS OF THE CLASS.

__ sub __ () --> "-"

__ mul __ () --> "*"

__ true div __ () --> "/"

In [ ]:
```
1
```

In [35]:
```
class fraction:

    def __init__(self,x,y):
        self.num=x
        self.den=y

    def __str__(self):
        print( "idhar")

    def __add__(self,other):

        new_num = (self.num * other.den) + (other.num * self.den)
        new_den = self.den + other.den

        return "{}/{}".format(new_num,new_den)

```

In [36]:
```
f1=fraction(2,3)
```

In [37]:
```
f2=fraction(3,4)
```

In [38]:
```
f2.den
```

Out[38]: 4

In [39]:
```
f1.den
```

Out[39]: 3

In [40]:
```
f1+f2
```

Out[40]: '17/7'

In [1]:
```
class Person:
    def __init__(self,name_input,country_input):
        self.name=name_input
        self.country=country_input

    def great(self):
        if self.country == "India":
            print("Namaste" ,self.name)
        else:
            print("Hello" ,self.name)
```

In [2]:
```
p=Person("Mitra","India")
```

In [3]:
```
# HOW TO ACCESS ATTRIBUTES
p.country
```

Out[3]:
```
'India'
```

In [4]:
```
p.name
```

Out[4]:
```
'Mitra'
```

In [5]:
```
# HOW TO ACCESS METHODS
p.great()
```

```
Namaste Mitra
```

In [ ]:
```
# You just write OBJECT NAME >> dot(.) >> [CLICK TAB OPTION]
# You will see all the Attributes & Methods of that class.
```

In [6]:
```
# WHAT IF I TRY TO ACCESS NON-EXISTENT ATTRIBUTES?
p.gender   # IT WILL GIVE ERROR
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_11736\2416981676.py in <module>
      1 # WHAT IF I TRY TO ACCESS NON-EXISTENT ATTRIBUTES?
----> 2 p.gender   # IT WILL GIVE ERROR

AttributeError: 'Person' object has no attribute 'gender'
```

In [7]:
```
# ATTRIBUTE CREATION FROM OUTSIDE OF THE CLASS.
p.gender = "Male"
```

In [8]:
```
p.gender
```

Out[8]:
```
'Male'
```

REFERENCE VARIABLE:

- REFERENCE VARIABLES HOLD THE OBJECTS.
- WE CAN CREATE OBJECTS WITHOUT REFERENCE VARIABLE AS WELL.
- AN OBJECT CAN HAVE MULTIPLE REFERENCE VARIABLES.
- ASSIGNING A NEW REFERENCE VARIABLE TO AN EXISTING OBJECT DOES NOT CREATE A NEW OBJECT.
- A REFERENCE VARIABLE IS A VARIABLE THAT POINTS TO AN OBJECT OF A GIVEN CLASS, LETTING YOU ACCESS HE VALUE OF AN OBJECT.

In [14]:
```
# eg

class Person:
    def __init__(self):
        self.name="Mitra"
        self.country="India"

```

In [15]:
```
Person()  # Object is Created
```

Out[15]:
```
<__main__.Person at 0x1a4a8772c40>
```

In [16]:
```
p=Person()
```

In [17]:
```
p
```

Out[17]:
```
<__main__.Person at 0x1a4a8772190>
```

In [18]:
```
id(Person())
```

Out[18]:
```
1806691070784
```

In [19]:
```
id(p)
```

Out[19]:
```
1806712643984
```

In [ ]:
```
# That means "p" is not the object it contain the address of the Object
# So "p" is the REFERENCE VARIABLE which hold the Address of the Object
```

PASS BY REFERENCE

In [20]:
```
# eg
class Person:
    def __init__(self,name,gender):
        self.name=name
        self.gender=gender


# OUTSIDE OF THE CLASS
def greet(person):
    print("Hii my name is", person.name, "and I am a", person.gender)

```

In [22]:
```
p=Person("Mitra","Male")
greet(p)
```

```
Hii my name is Mitra and I am a Male
```

In [ ]:
```
# That means we can Execute Class Object inside a Indepenedet Function.
```

In [25]:
```
# eg

class Person:
    def __init__(self,name,gender):
        self.name=name
        self.gender=gender


# OUTSIDE OF THE CLASS
def greet(person):
    print("Hii my name is", person.name, "and I am a", person.gender)
    p1=Person("Ankit","Male")
    return p1
```

In [26]:
```
p=Person("Mitra","Male")
x=greet(p)
print(x.name)
print(x.gender)
```

```
Hii my name is Mitra and I am a Male
Ankit
Male
```

In [ ]:
```
# We can also Execute Class object inside a Independent Function.
```

TYPES OF VARIABLES:

a. LOCAL VARIABLES:

- THESE VARIABLES ARE DECLARED & ACCESSED WITHIN THE SAME METHOD OF A PARTICULAR CLASS.

In [27]:

```
# eg

class empclass:
    def m_input(self):

        print("Employee input method")
        name1=input("Enter Employee Name: ")
        deg1=input("Enter Employee Designation: ")
        tech1=input("Enter Employee Technology: ")

    def m_display(self):
        print("Display Method")
        print(name1)
        print(deg1)
        print(tech1)

obj7=empclass()
obj7.m_input()
obj7.m_display()
```

```
Employee input method
Enter Employee Name: Mitra
Enter Employee Designation: Data Analyst
Enter Employee Technology: Python
Display Method

---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_11736\4167071036.py in <module>
     15 obj7=empclass()
     16 obj7.m_input()
---> 17 obj7.m_display()

~\AppData\Local\Temp\ipykernel_11736\4167071036.py in m_display(self)
      9     def m_display(self):
     10         print("Display Method")
---> 11         print(name1)
     12         print(deg1)
     13         print(tech1)

NameError: name 'name1' is not defined
```

b. INSTANCE VARIABLE:

- IT IS A SPECIAL KIND OF VARIABLE THAT DEPENDS ON OBJECT OF THE VARIABLE.
- FOR EVERY OBJECT THE VALUE OF INSTANCE VARIABLE IS DIFFERENT.
- eg:-->
  - Student's Name, Age, CGPA
  - Bank's Account Number, Account Holder Name, Account Balance
- OBJECT LEVEL VARIABLE, THE VALUES ARE DIFFERENT FOR EVERY OBJECT.

> - THESE VARIABLES ARE CLASS LEVEL GLOBAL VARIABLES.
> - THESE VARIABLES ARE DECLARED WITHIN A METHOD WHICH CAN BE ACCESSED IN ANY METHOD OF SAME CLASS.
> - INSTANCE IS PREFIXED WITH `"self"` KEYWORD.
> - INSTANCE VARIABLE CAN BE ACCESSED USING `"slef"` KEYWORD.

In [28]:
```
class empclass:
    def m_input(self):

        print("Employee input method")
        self.name=input("Enter Employee Name: ")
        self.deg=input("Enter Employee Designation: ")
        self.tech=input("Enter Employee Technology: ")
        print()

    def m_display(self):
        print("Display Method")
        print(self.name)
        print(self.deg)
        print(self.tech)

obj1=empclass()
obj1.m_input()
obj1.m_display()
```

```
Employee input method
Enter Employee Name: mITRA
Enter Employee Designation: Data Analyst
Enter Employee Technology: Pythn

Display Method
mITRA
Data Analyst
Pythn
```

c. STATIC VARIABLES:

- THESE VARIABLES ARE CONSIDERED AS A GLOBAL VARIABLE WHICH CAN BE ACCESSED OUTSIDE THE CLASS ALSO.

- THESE VARIABLES ARE DECLARED OUTSIDE ALL THE METHOD & INSIDE THE CLASS.

> **ACCESSIBILITY**:
>
> > - THESE VARIABLES ARE ACCESSIBLE WITHIN THE METHOD OF SAME CLASS USING CLASS NAME.
> > - THESE VARIABLES ARE ACCESSIBLE OUTSIDE THE CLASS USING CLASS NAME or CLASS OBJECT.

In [29]:

```
# eg

class myclass:
    city="Hydrabad"  # Static Variable

    def m_add(self):
        print("Add Method")
        self.num1=int(input("Enter first number: "))  # Instance Variab
        self.num2=int(input("Enter second number: ")) # Instance Variab
        a=self.num1+self.num2 # Local Variable
        print(self.num1)
        print(self.num2)
        print(a)
        print(myclass.city) # Call city inside the method


    def m_mul(self):
        print("Multiplication Method")
        m=self.num1*self.num2 # Local Variable
        print(self.num1)
        print(self.num2)
        print(m)
        print(myclass.city) # Call city inside the method

obj2=myclass()
obj2.m_add()
print()
obj2.m_mul()
print()
print("Outside the class")
print("Using object: ",obj2.city)
print()
print("Using class: ",myclass.city)
```

```
Add Method
Enter first number: 12
Enter second number: 15
12
15
27
Hydrabad

Multiplication Method
12
15
180
Hydrabad

Outside the class
Using object:  Hydrabad

Using class:  Hydrabad
```

TYPES OF METHODS:

a. INSTANCE METHOD:

- THIS IS DEFAULT METHOD.
- A METHOD WITH `"self"` AS DEFAULT ARGUMENT IS CONSIDERED AS INSTANCE METHOD.

ACCESSIBILITY:

- THESE METHODS CAN BE ACCESSED WITHIN THE CLASS & FROM OUTSIDE THE CLASS.

WITHIN CLASS:

- IT IS ACCESSED USING `"self"` KEYWORD.
  - `self.methodname()`

OUTSIDE CLASS:

- IT CAN BE ACCESSED USING OBJECT.
  - `object.methodname()`

ALL THE VARIABLES ARE ACCESSIBLE

In [30]:
```
class myclass:
    def m_add(self):
        print("This is an Instance Method")

obj=myclass()
obj.m_add()
```

```
This is an Instance Method
```

b. CLASS METHOD:

- A METHOD WITH "cls" AS DEFAULT ARGUMENT IS CONSIDERED AS CLASS METHOD.
- THESE CLASS METHODS ARE DEFINED USING `DECORATOR / ANNOTATION .`
  `@classmethod`

ACCESSIBILTY:

OUTSIDE CLASS:

- THESE METHODS ARE ACCESSED USING TWO WAYS,
  - `classname.methodname()`
  - `obj.methodname()`

VARIABLES:

- LOCAL VARIABLES
- STATIC VARIABLES
  - `classname.staticvariable`
  - `cls.staticvariable`

- **INSTANCE VARIABLES CAN NOT BE ACCESSED.**

In [31]:

```
class myclass:
    def greet(self):
        print("This is an Instance Method")

    @classmethod
    def method1(cls):
        print("This is from CLASS METHOD")



obj=myclass()
obj.greet()
print()
obj.method1()
print()
myclass.method1()
```

```
This is an Instance Method

This is from CLASS METHOD

This is from CLASS METHOD
```

c. STATIC METHOD:

- A METHOD WITH NO DEFAULT ARGUMENTS IS CALLED AS STATIC METHOD.
- THESE STATIC METHODS ARE DEFINED USING `DECORATOR / ANNOTATION` . `@staticmethod`

ACCESSIBILITY:

OUTSIDE CLASS:

- THESE METHOD ARE ACCESSED USING TWO WAYS,
  - `classname.methodname()`
  - `obj.methodname()`

VARIABLE:

- LOCAL VARIABLES
- STATIC VARIABLES
  - `classname.staticvariable`
  - `cls.staticvariable`

INSTANCE VARIABLES CAN NOT BE ACCESSED.

In [32]:

```
# eg

class myclass:
    def greet(self):
        print("This is an Instance Method")

    @classmethod
    def method1(cls):
        print("This is from CLASS METHOD")

    @staticmethod
    def method2():
        print("This is from Static Method")


obj=myclass()
obj.greet()
print()
obj.method1()
print()
myclass.method1()
print()
obj.method2()
print()
myclass.method2()

```

```
This is an Instance Method

This is from CLASS METHOD

This is from CLASS METHOD

This is from Static Method

This is from Static Method
```

In [33]:

```python
# eg

class skyclass:
    city="HYDRABAD"

    def m1(self,n1,n2):
        self.num1=n1
        self.num2=n2

        a=self.num1+self.num2

        print(self.num1)
        print(self.num2)
        print(a)
        print(skyclass.city)
        print("**********************************")

    @classmethod
    def cmethod(cls):
        print("This is from Class Method")
        n1=100
        n2=200
        m=n1+n2
        print(m)
        print(skyclass.city)
        print("---------------------------------------")
        print(cls.city)


    @staticmethod
    def smethod():
        print("This is from Static Method")
        x=50
        y=90
        b=x+y
        print(b)
        print(skyclass.city)




obj1=skyclass()
obj1.m1(400,500)
print("**********************************")
obj1.cmethod()
print("*************************")
skyclass.cmethod()

obj1.smethod()
print("*****************************")
skyclass.smethod()
```

```
400
500
900
HYDRABAD
**********
**********
This is from Class Method
300
HYDRABAD
----------
HYDRABAD
**********
This is from Class Method
300
HYDRABAD
----------
HYDRABAD
This is from Static Method
140
HYDRABAD
**********
This is from Static Method
140
HYDRABAD
```

NESTED CLASS

In [34]:

```python
class parent:
    country="INDIA"

    def pa1(self):
        print("This is from Parent Class Method 1")

    class child:
        state="ODISHA"

        def c1(self):
            print("This is from Child Class Method 1")

obj_p=parent()
print(obj_p.country)
obj_p.pa1()

```

```
INDIA
This is from Parent Class Method 1
```

In [35]:

```
class parent:
    country="INDIA"

    def pa1(self):
        print("This is from Parent Class Method 1")

    class child:
        state="ODISHA"

        def c1(self):
            print("This is from Child Class Method ")

obj_p=parent()
print(obj_p.country)
obj_p.pa1()

obj_c=obj_p.child()
print(obj_c.state)
obj_c.c1
```

```
INDIA
This is from Parent Class Method 1
ODISHA
```

Out[35]:

```
<bound method parent.child.c1 of <__main__.parent.child object at 0x000001
A4A731AFD0>>
```

In [36]:

```
class parent:
    country="INDIA"

    def pa1(self):
        print("This is from Parent Class Method 1")

        obj_c= self.child()
        print("State: ",obj_c.state)
        obj_c.c1()

    class child:
        state="ODISHA"

        def c1(self):
            print("This is from Child Class Method ")

obj_p=parent()
print(obj_p.country)
obj_p.pa1()
```

```
INDIA
This is from Parent Class Method 1
State:  ODISHA
This is from Child Class Method
```