



```
In [1]: 1  ## oops-concept
        2  ## (1) class and objects
        3
        4  # class(template)
        5  '''
        6  1.we will define class by using 'class'
        7  2.blue print to create a objects
        8  3.collection of objects is called class
        9  '''
       10  #ex fruits
       11
       12  # object
       13  # physical entity(real)
       14  '''
       15  1.an instance of a class
       16  2.memory is created when it declared
       17  '''
       18  #ex apple,orange,mango
       19
       20  # attribute (variable) data members
       21  '''
       22  age=20
       23  color='blue'
       24  '''
       25
       26
       27  # method(behaviour) or functions
       28  '''
       29  eat()
       30  sleep()
       31  '''
```



```
26
27 # method(behaviour) or functions
28 '''
29 eat()
30 sleep()
31 '''
32
33 # self keyword
34 '''
35 we can access the attributes and methods of the class(current class only)
36 '''
37 # class Class_name:
38 #     #constructor
39 #     #attributes
40 #     #methods
41
42
43
44
45
46 class Apple:                # class , class name
47     print('jai kisan')      # attributes/data members
48     def display(self):      #method(self)
49         a=55
50         b=66
51         print(a,b)
52 obj=Apple()                 #object name,method name()
53 obj.display()               #declaring object
```

jai kisan

55 66



```
In [26]: 1 class Apple:
          2     c=88
          3     def display(self):
          4         a=55
          5         print(a)
          6 ob=Apple()
          7 obj.display()
          8 obj.c
```

55

Out[26]: 88

```
In [12]: 1 class Apple:
          2     a=22
          3     def display(self):
          4         print(self.a)
          5 obj=Apple()
          6 obj.display()
```

22

```
In [31]: 1
          2 # __init__
          3
          4 '''
          5 Constructors are generally used for instantiating an object.
          6 The task of constructors is to initialize(assign values)
          7 to the data members of the class when an object of the class
          8 is created.
          9 In Python the __init__ () method is called the constructor
```



Run



Code



In [31]:

```
1  # __init__
2
3
4  '''
5  Constructors are generally used for instantiating an object.
6  The task of constructors is to initialize(assign values)
7  to the data members of the class when an object of the class
8  is created.
9  In Python the __init__() method is called the constructor
10 and is always called when an object is created
11
12 doesn't support multiple constructor
13 '''
14
15 class Name:                                # class declaration
16     def __init__(self,a,b,c):               #constructor
17         self.l=a        # variables
18         self.m=b
19         self.n=c
20     def display(self):
21         print(self.l)
22         print(self.m)
23         print(self.n)
24
25 obj=Name(25,26,27)
26 obj.display()
```

25

26

27

In [10]:

```
1  ## inheritance
2  ## single parent child
3  ## multiple
4  ## multilevel
5  ## hierarchical
6
7
8  # single-level-inheritance
9  class Parent:
10     def display(self):
11         print('this is a parent class')
12
13  class Child(Parent):
14     def display1(self):
15         print('this is a child class')
16  obj=Child()
17  obj.display()
18  obj.display1()
```

this is a parent class

this is a child class

In [3]:

```
1  ### multilevel-inheritance
2  class Grandfather:
3      def display(self):
4          print('this is a grand father class')
5  class Father(Grandfather):
6      def display1(self):
7          print('this is a father class')
8  class Child(Father):
9      def display2(self):
10         print('this is a child class')
11
12  obj=Child()
13  obj.display()
14  obj.display1()
15  obj.display2()
```

```
this is a grand father class
this is a father class
this is a child class
```

In [11]:

```
1  #multiple-inheritance
2
3  class Father:
4      def display(self):
5          print('this is a father class')
6  class Mother:
7      def display1(self):
8          print('this is a mother class')
9  class Child(Father,Mother):
10     def display2(self):
11         print('this is a child class')
12 obj=Child()
13 obj.display()
14 obj.display1()
15 obj.display2()
16
```

```
this is a father class
this is a mother class
this is a child class
```




In [15]:

```
1  # heirchical-inheritance
2  class Father:
3      def display(self):
4          print('this is a father class')
5  class Child1(Father):
6      def display1(self):
7          print('this is a child 1 class')
8  class Child2(Father):
9      def display2(self):
10         print('this is a child 2 class')
11  obj=Child1()
12  obj=Child2()
13  obj.display()
14  obj.display()
```

```
this is a father class
this is a father class
```




In [12]:

```
1  # polymorphism##
2  #method- overloading ::same class,same methods,different parame
3
4  class Method_overloading:
5      def display(self,a,b=0,c=0,d=0):
6          print(a,b,c,d)
7
8  obj=Method_overloading()
9  obj.display(10,14,15,18)
10 obj.display(10,14,15)
11 obj.display(10,14)
12 obj.display(10)
```

```
10 14 15 18
10 14 15 0
10 14 0 0
10 0 0 0
```



```
In [45]: 1 # method-overriding
          2 # different class,different parameters,same methods
          3 class Methodoverriding:
          4     def display(self):
          5         print('this is method overriding class')
          6
          7 class Child(Methodoverriding):
          8     def display(self):
          9         print('this is child class')
         10         super().display()
         11
         12 obj=Child()
         13 obj.display()
         14
```

```
this is child class
this is method overriding class
```

[56]:

```
1
2 #encapsulation
3
4 # binding of class (methods and variables(attributes))
5 # public
6 # and
7 # private __
8 # protected _
9
10
11 ## _ is protected,they belongs to only family
12
13 class GrandFather:
14     def __init__(self,a):
15         self._x=a
16
17 class Father(GrandFather):
18     def display1(self):
19         print(self._x)
20
21 class Child(Father):
22     def display2(self):
23         print("child",self._x)
24
25 obj=Child('500 cr')
26 obj.display2()
```

child 500 cr

```
58]: 1  ## __private ,they dont belong to anyone
      2  class GrandFather:
      3      def __init__(self,a):
      4          self.__x=a
      5
      6  class Father(GrandFather):
      7      def display1(self):
      8          print(self.__x)
      9
     10  class Child(Father):
     11      def display2(self):
     12          print("child",self.__x)
     13
     14  obj=Child('500 cr')
     15  obj.display2()
```

AttributeError Traceback (most recent call last)

<ipython-input-58-8787a65aadc9> in <module>

```
13
14 obj=Child('500 cr')
--> 15 obj.display2()
```

<ipython-input-58-8787a65aadc9> in display2(self)

```
10 class Child(Father):
11     def display2(self):
--> 12         print("child",self.__x)
13
14 obj=Child('500 cr')
```

AttributeError: 'Child' object has no attribute '_Child__x'

In [6]:

```
1  #abstraction
2  #abs method there is no body
3  #abs class can not create object
4  #a class contain one or more abstract methods then it said to be a abc
5
6
7
8  from abc import ABC,abstractmethod
9
10 class Parent(ABC):
11
12     @abstractmethod
13
14     def display(self):
15         pass
16     def display1(self):
17         print('this is parent')
18
19 class Child(Parent):
20     def display(self,a):
21         print('this is child',a)
22
23 obj=Child()
24 obj.display(1200)
25 obj.display1()
```

this is child 1200

this is parent