classes and objects in this the class is the blue print of the objects that means if we have a class we can have any number of objects and those ojcets have properties

- lets have an example : teddy bear and the mold here the mold is the class by that mold we can create n number of objects like teddy bears and this teddy is object and that teddy will have properties like weight, color, and taste this are about class and obejcts

now about meathods

a meathod is actually like a function which manipulate and changes the attributess of the specific object. this is about the meathod creating the class and defining the class attributes n now we need to create a class and the object to them and we need to create attributes to them to accesss class teddy:

In [2]:
```
class Teddy:
    quantity=200
teddy1=Teddy()
teddy2=Teddy()
print(teddy1.quantity)
print(teddy2.quantity)
```

```
200
200
```

to change the attributes we need to use instance meathod

we should have to create a constructor that is init meathod and self is a key word in it that means its a reference to that perticular instance

In [3]:
```
class Teddy:
    quantity=200

    def __init__(self,name,color):
        self.name=name
        self.color=color
teddy1=Teddy('abhi','white')
print(teddy1.name)
print(teddy1.color)
```

```
abhi
white
```

In [5]:
```
teddy2=Teddy('sonu','black')
print(teddy2.name)
print(teddy2.color)
```

```
sonu
black
```

now in this we are going to crate the class meathod and we will going to use that method so that we need to create in the same class to acess the meathod

In [7]:
```
class Teddy:
    quantity=200

    def __init__(self,name,color):
        self.name=name
        self.color=color
    def change_name(self):
        self.color='white'
teddy1=Teddy('abhi','sonu')
print(teddy1.name)
print(teddy1.color)

teddy1.change_name()
print(teddy1.name)
print(teddy1.color)
```

```
abhi
sonu
abhi
white
```

In [8]:
```
class Teddy:
    quantity=200

    def __init__(self,name,color):
        self.name=name
        self.color=color
    def change_name(self,color):
        self.color=color
teddy1=Teddy('abhi','sonu')
print(teddy1.name)
print(teddy1.color)

teddy1.change_name('Orange')
print(teddy1.name)
print(teddy1.color)
```

```
abhi
sonu
abhi
Orange
```

function way vs oops way of writing the code now we are going to create a student class for that we need to initialize the data next we need to add instance and then get the data and then display the data from taking the input from the user

In [9]:
```
### Functional way
name=input('enter name')
age=input('enter age')
print(name)
print(age)
```

```
enter nameAbhilash
enter age25
Abhilash
25
```

In [13]:
```
### OOPS ways
class Student:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def get_data(self):
        self.name=input('enter name')
        self.age=input('enter age')
    def put_data(self):
        print(self.name)
        print(self.age)
student1=Student("","")
student1.get_data()
student1.put_data()
```

```
enter nameAbhilash
enter age25
Abhilash
25
```

inheritance is where one class can accept the properties and meathods of other class

In [16]:
```
### Single Inheritance
class Student:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def get_data(self):
        self.name=input('enter name')
        self.age=input('enter age')
    def put_data(self):
        print(self.name)
        print(self.age)
class Science_student(Student):
    def science(self):
        print('this is a science meathod')
a=Science_student("","")
a.get_data()
a.put_data()
```

```
enter nameAbhilash
enter age25
Abhilash
25
```

multiple inheritance we can inherit from one class

In [19]:
```
### Multiple Inheritance
class A:
    def a_meathod(self):
        print("this is the a method")
class B:
    def b_meathod(self):
        print("this is the B method")
class C(A,B):
    def c_meathod(self):
        print("this is the C method")
```

In [24]:
```
c_object=C()
c_object.a_meathod()
c_object.b_meathod()
c_object.c_meathod()
```

```
this is the a method
this is the B method
this is the C method
```

multi level inheritance is that we can inherit the class which is already inherited by a method

In [26]:
```
### Multilevel Inheritance
class A:
    def a_meathod(self):
        print("this is the a method")
class B(A):
    def b_meathod(self):
        print("this is the B method")
class C(B):
    def c_meathod(self):
        print("this is the C method")
c_object=C()
c_object.a_meathod()
c_object.b_meathod()
c_object.c_meathod()
```

```
this is the a method
this is the B method
this is the C method
```

Recursion in python

- that means the function calling itself

In [28]:
```
def factorial(x):
    if x==1:
        return 1
    else:
        return x*(factorial(x-1))
result=factorial(5)
print(result)
```

```
120
```

sets

- it consists of the unique numbers not like list

In [29]:
```
numbers={1,2,3,4,5}
print(5 in numbers)
```

```
True
```

In [30]:
```
numbers.add(9)
```

In [31]:
```
print(numbers)
```

```
{1, 2, 3, 4, 5, 9}
```

In [33]:
```
numbers.remove(4)
```

In [34]:
```
numbers
```

Out[34]:
```
{1, 2, 3, 5, 9}
```

In [ ]:
```
### Union
```

In [38]:
```
seta={1,2, 3, 4, 5}
setb={4, 5, 6, 7, 8}
print(seta | setb)
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [ ]:
```
### Intersection
```

In [39]:
```
seta={1,2, 3, 4, 5}
setb={4, 5, 6, 7, 8}
print(seta & setb)
```

```
{4, 5}
```

In [41]:
```
### Difference
seta={1,2, 3, 4, 5}
setb={4, 5, 6, 7, 8}
print(setb - seta)
```

```
{8, 6, 7}
```

itertools

- it helps us to do some of the functional programming
- the count function accepts only single function

In [42]:
```
from itertools import count
for i in count(3):
    print(i)

    if i ==20:
        break
```

```
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

In [63]:
```
### accumulate function it keeps on accumalting the numbers from  a list

from itertools import accumulate
numbers = list(accumulate(range(0)))
print(numbers)
```

```
[0, 1, 3, 6, 10, 15, 21, 28]
```

In [53]:
```
def from_iterable(iterables):
    # chain.from_iterable(['ABC', 'DEF']) --> A B C D E F
    for it in iterables:
        for element in it:
            yield element
```

In [56]:
```
from_iterable('ABC')
```

Out[56]:
```
<generator object from_iterable at 0x1132f09e8>
```

In [57]:
```
### Takewhile in itertools
```

In [66]:
```
from itertools import accumulate,takewhile
numbers = list(accumulate(range(0)))
print(numbers)
print(list(takewhile(lambda x: x<=10,numbers)))
```

```
[0, 1, 3, 6, 10, 15, 21, 28]
[0, 1, 3, 6, 10]
```

In [68]:
```
### Operator Overloading
class Point:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def __add__(self,other):
        x=self.x+other.x
        y=self.y+other.y
        return Point(x,y)
    def __str__(self):
        return "({},{})".format(self.x,self.y)
point1=Point(1,4)
point2=Point(2,6)
print(point1 + point2)
```

```
3,10
```

Data Hiding- Encapsulation

- it allows the features like data hiding
- only certain amount of code is accessable to the data or visible
- we cannot able to access the data out side of the class

In [75]:
```
class Myclass:
    __hiddenvariable=0

    def add(self,increament):
        self.__hiddenvariable+=increament
        print(self.__hiddenvariable)
object1=Myclass()
object1.add(2)
print(object1._hiddenvariable)
### here we cannot access the inside variable
```

```
5

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-75-512402aa4504> in <module>
      7 object1=Myclass()
      8 object1.add(2)
----> 9 print(object1._hiddenvariable)
     10 ### here we cannot access the inside variable

AttributeError: 'Myclass' object has no attribute '_hiddenvariable'
```