object oriented programming(oops)

- python is an object oriented programming language
- Almost everything in python is an object,with properites and methods

- Class:
  - "Blueprint" for creating an object
  - collection of variables and methods

In [2]:
```
a = [1,2,3]
print(type(a))
```

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

In [3]:
```
print(dir(list))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__form
at__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str_
_', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'rev
erse', 'sort']
```

In [6]:
```
class Person:
    age = 10
    def greet(self):
        print("Hello")
print(Person.age)
print(Person.greet)
```

```
10
<function Person.greet at 0x0000021905C70D08>
```

- object:
  - instance of the class

In [12]:
```
class Person:
    age = 10
    def greet(self):
        print("Hello")
p = Person()
p.greet()
print(p.age)
```

```
Hello
10
```

- Constructor:
  - Constructor in python class functions begins with double underscore __ are called special functions as they are having special meaning.
  - This special function gets called whenever a new object of the class is instantiated

In [15]:
```
class Srkit:
    y = "alekhya"
    def __init__(self):
        print("i am a constuctor")
x = Srkit()
x.y
```

```
i am a constuctor
```

Out[15]:
```
'alekhya'
```

In [20]:
```
class Srkit:
    y = "alekhya"
    def __init__(self,a,b=10):
        print("i am a constuctor")
        print(a,b)
x = Srkit(10,20)
```

```
i am a constuctor
10 20
```

In [27]:
```
class Srkit:

    def __init__(self,*a,b):
        print(a,b)
x = Srkit(2,b=45)
y = Srkit(2,1,3,4,5,b=89)
```

```
(2,) 45
(2, 1, 3, 4, 5) 89
```

In [29]:
```
class Apssdc:
    def __init__(self,name,number):
        self.name=name
        self.number=number
    def display(self):
        print(self.name)
        print(self.number)
x = Apssdc("alekhya",9876543210)
x.display()
```

```
alekhya
9876543210
```

- Inheritance
  - Inheritance allows us to define a class that inherits all the methods and properties from another class.

In [32]:
```
class A:#base class/parent class
    def details(self,name):
        self.name=name
        print(self.name)

class B(A): #derived class/child class
    def display(self,number):
        self.number=number
        print(self.number)


z = B()
z.display(908765432)
z.details("alekhya")
```

```
908765432
alekhya
```

In [35]:
```
class A:
    def __init__(self):
        print("i am a base class from constructor")
class B(A):
    def __init__(self):
        print("i am derived class from constructor")
        super().__init__()
x = B()
```

```
i am derived class from constructor
i am a base class from constructor
```

In [37]:
```
class A:
    def __init__(self):
        print("i am a base class from constructor")
    def display(self):
        print("i am a base class method")
class B(A):
    def __init__(self):
        print("i am derived class from constructor")
        super().__init__()
        super().display()
x = B()
```

```
i am derived class from constructor
i am a base class from constructor
i am a base class method
```

Multiple inheritance

- One class acquires the properties of one or more parent classes.

In [41]:
```
class A:
    def display(self):
        print("i am from class A")
class B:
    def display1(self):
        print("i am from class B")
class C(A,B):
    def display2(self):
        print("i am from class C")
s = C()
s.display2()
s.display1()
s.display()
```

```
i am from class C
i am from class B
i am from class A
```

- Multilevel inheritance

In [49]:
```
class A:
    def display1():
        print("i am from A class")
class B(A):
    def display2():
        print("i am from B class")
class C(B):
    def display3():
        print("i am from c class")
c = C
c.display3()
c.display2()
c.display1()
```

```
i am from c class
i am from B class
i am from A class
```

- Hierarchical inheritance
  - more than one derived classes are created from single base class

In [50]:
```
class A:
    def display1():
        print(" i am from A")
class B(A):
    def display2():
        print(" i am from B")
class C(A):
    def display3():
        print(" i am from C")
x = B
x.display2()
x.display1()
y = C
y.display3()
y.display1()
```

```
 i am from B
 i am from A
 i am from C
 i am from A
```

hybrid inheritance

In [55]:
```
class A:
    def display1():
        print("i am from A")
class B(A):
    def display2():
        print("i am from B")
class C(B):
    def display3():
        print("i am from C")
class D(A):
    def display4():
        print("i am from D")

x = D
x.display4()
x.display1()
y = C
y.display3()
y.display2()
y.display1()
```

```
i am from D
i am from A
i am from C
i am from B
i am from A
```

In [ ]:

In [ ]:

Numpy

In [56]:
```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\alekhya\anaconda3\lib\site-packages (1.16.2)
Note: you may need to restart the kernel to use updated packages.

WARNING: You are using pip version 20.2.3; however, version 20.3.3 is available.
You should consider upgrading via the 'C:\Users\Alekhya\Anaconda3\python.exe -m pip install --upgrade pip' com
mand.
```

In [69]:
```
import numpy as np
```

In [58]:
```
np.__version__
```

Out[58]: '1.16.2'

In [70]:
```
a = np.array([1,2,4])
print(type(a))
```

```
<class 'numpy.ndarray'>
```

In [63]:
```
b = [1,2,3]
print(type(b))
```

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

In [64]:
```
a.ndim
```

Out[64]: 1

In [65]:
```
a.dtype
```

Out[65]: dtype('int32')

In [71]:
```
arr = np.array([1,2,"alekhya",8.9])
arr.dtype
```

Out[71]: dtype('<U11')

In [72]:
```
s = np.array("23-12-2020")
s.dtype
```

Out[72]: dtype('<U10')

In [73]:
```
a = np.array(3+2j)
a.dtype
```

Out[73]:
```
dtype('complex128')
```

In [74]:
```
range(1,10)
```

Out[74]:
```
range(1, 10)
```

In [76]:
```
range(1,10,2.5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-76-ed5239e802cc> in <module>
----> 1 range(1,10,2.5)

TypeError: 'float' object cannot be interpreted as an integer
```

In [79]:
```
np.arange(1,11,1.5)
```

Out[79]:
```
array([ 1. ,  2.5,  4. ,  5.5,  7. ,  8.5, 10. ])
```

In [81]:
```
a = np.arange(1,10.5,1.2)
a
```

Out[81]:
```
array([1. , 2.2, 3.4, 4.6, 5.8, 7. , 8.2, 9.4])
```

In [82]:
```
a.size
```

Out[82]:
```
8
```

In [83]:
```
a.shape
```

Out[83]:
```
(8,)
```

In [84]:
```
s = np.array([[1,2,3],[4,5,6]])
s.shape
```

Out[84]:
```
(2, 3)
```

In [85]:
```
s
```

Out[85]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [86]:
```
i=int(input())
Multiplication=[i*n for n in range(1,11)]
print(Multiplication)
```

```
6
[6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
```

In [ ]:
```
Task-1(frequency of characters in a string)


Task-2(to print even values in a dictionary using DC)

D1={"chandu" :20,"rani":17,"saidivya":16}
Even={key:value for (key, value) in D1.items() if value%2==0}
print(Even)

Task-3(to print only alphabets in string using list comprehension)

S="apssdc12345srkit67134"
A=[char for char in S if char.isalpha()]
```

In [91]:
```
S="python programming"
D={char:S.count(char) for char in S}
print(D)
```

```
{'p': 2, 'y': 1, 't': 1, 'h': 1, 'o': 2, 'n': 2, ' ': 1, 'r': 2, 'g': 2, 'a': 1, 'm': 2, 'i': 1}
```

In [88]:
```
D1={"chandu" :20,"rani":17,"saidivya":16}
Even={key:value for (key, value) in D1.items() if value%2==0}
print(Even)
```

```
{'chandu': 20, 'saidivya': 16}
```

In [89]:
```
S="apssdc12345srkit67134"
A=[char for char in S if char.isalpha()]
A
```

Out[89]:
```
['a', 'p', 's', 's', 'd', 'c', 's', 'r', 'k', 'i', 't']
```

In [ ]:
```

```