

Python:

What is python:

Python is a scripting and general purpose high level and interpreted programming language.

Programming: programming is a way to instruct computer to do some specific task

Features of python:

1)Functional programming language: we can write programs by using functions

2)object oriented programming language: we can write programs by using classes and objects

3)Dynamically typed:no need to declare any type of data we storing in a variable

4)interpreted language: it compiles the code line by line if it detects an error then control flow will be terminated

5)scripting language: python is used for scripting purpose also

Variable: variable is a name it is used to given to memory location

.py and .pyc: .py is python file that contains source code and source code can be duplicated and .pyc is compile python file that contains byte code and byte code can't be duplicated

Identifiers: identifiers is used to identify the class name,function name and variable name.

Rules of identifiers:

1. Can't start with an integers.
2. Special characters are not allowed except underscore(_).
3. Key words are not allowed(True,False,None,etc total 34 key words).

Data types: there are two data types are there

1. single value data types

2.collection data types

1.single value data types:(int,float,Boolean,complex):

one variable stores only one value those are single value data types.

Int: a number without decimal point it is called integer and default value of integer is zero(0).

```
a=0
print(a)
print(type(a))
```

Float: a number with decimal point it is called float and default value of float is 0.0

```
a=0.0
print(a)
print(type(a))
```

Boolean: Boolean data type have only two True and False
Default value of bool is False.

```
a=False
print(a)
print(type(a))
```

by default python consider True as 1 and False False has 0.

Complex: complex data type will have two parts.

1) real part

2) imaginary part

```
a=3+4j
```

```
print(a)
```

```
print(type(a))
```

Default value of complex is 0j

2.collection data types:(string,list,tuple,set,dictionary):

1)string:

- string is declared by using single coats(' ') double coats(" "),triple coats('' '').
- if we declare a single line string we use single coats and double coats and we declare multiline string we use triple coats.
- String is orderd data type it support indexing.
- String is immutable data type it can't be modified.

2)List:

- list is declared by using square brackets([]).
- List is ordered data type it support's indexing.
- List is mutable data type it can be modified.
- It allows duplicate.
- List is both homoginus and heteroginus
Homoginus means having similar data([1,2,3,4,5])and heteroginus means having different data([1,2.3,True,3+4j]).

3)Tuple:

- Tuple is declared by using round brackets().
- Tuple is ordered data type it support indexing.
- It is immutable data type it can't be modified but mutable data types in tuple can be modifies.
- It allows duplicates
- If we declare single value in tuple comma(,) is mandatory
- It is both homogenous and heterogenous

4)set:

- Set is declared by using curlie brackets{} but empty set is declared by using set() function.
- Set is unordered data type it doesn't support indexing.
- Set is mutable data type it can be modified.
- It does not allow duplicates.
- It is both homogenous and heterogenous but it allows only immutable data type.

5)dictionary:

- dictionary is declared by using curlie brackets {}and elements are in the form of key value pair.
- It ordered data type but it doesn't support indexing
- It is mutable data type it can be modified.
- Duplicate keys are not allowed if we add duplicate keys old value replaced with new value where as duplicate values are allowed

- Only immutable keys are allowed but values can be both mutable and immutable.

Operators:

1. arithamatic operators(+, - , * , % , ** , / , //).
2. Comparision operators(>, <, >=, <=, ==, !=).
3. Assignment operators(=, +=, -=, *=, /=, %=, //=).
4. Betwise operators(&, |, ^, ~, <<, >>).
5. Logical operators(and, or).
6. Membership operators(in, not in)
7. Identity operators(is, is not)
8. Terinary operators(it means we should write if and else in single line).

Inbuilt methods

1)string inbuilt methods:

Upper(): this method is used to convert all lower case charecters to upper case charecters.

Example : n='vinay'
 print(n.upper())

op: VINAY

Lower():this method is used to convert all upper case charecters to lower case charecters.

Example : n='VINAY'
 print(n.lower())

op: vinay

swapcase: this method is used to convert all uppercase characters to lower case and lower case characters to upper case.

Example:

```
n='vinayVicky'
print(n.swapcase())
op: VINAYvicky
```

title: this method is used to convert every word of first character into uppercase.

Example: i='hai vinay how are you'
 print(i.title())
 Op: Hai Vinay How Are You

Capitalize: this method is used to convert first letter of a string into uppercase remaining all lower case.

Example: j='python developer'
 print(j.capitalize())
 op: Python developer

index:

- it returns index position of first occurrence of a substring
- if substring is not available then it throws value error.

Example: h='hai good morning'
 print(h.index('o'))
 op:5

rindex:

- it returns index position of last occurrence of a substring
- if substring is not available then it throws value error.

Example: h='hai good morning'
 print(h.rindex('o'))
 op:8

find:

- it returns index position of first occurrence of a substring
- if substring is not available then it will not throw any error it returns -1

Example:
 h='hai good morning'
 print(h.find('x'))
 op:-1

rfind:

- it returns index position of last occurrence of a substring
- if substring is not available then it will not through any error it returns -1

Example:

```
h='hai good morning'
print(h.find('x'))
op:-1
```

count:

- it returns no.of times substring occurred in string.
- If substring is not available then it returns 0
-

Example:

```
h='engineering'
print(h.count('e'))
op:3
```

replace: this method is used to replace one substring with another substring

```
Example: a='vinay'
print(a.replace('v','s'))
op:sinay
```

split:

- this method is used to convert string into list.
- Providing argument is not given by default it takes space as a delimator

Example: n='vinay is a python developer'

```
print(n.split())
op:['vinay', 'is', 'a', 'python', 'developer']
```

format: this method is used for variable substitution in a specified place holder.

```
Example: i='vinay'
j=20
print(f'{i} is {j} years old')
op: vinay is 20 years old
```

slicing: slicing is a feature to access part of sequence from string list tuple.

Example:

```
x='vinay completed masters'
print(x[:]) op: vinay completed masters
print(x[0:]) op: vinay completed masters
print(x[2:5]) op: nay
```

```
print(x[:2]) op: vnycm lae atr  
print(x[-1:-6:-1]) op: srets  
print(x[-1:-6:-2]) op: ses  
print(x[::-1]) op: sretsam detaelpmoc yaniv
```

list inbuilt methods

add values into the list:

append:

- this method is used to add one value at time
- value is added at end of the list

```
Example: l=[12,13,14,15]  
l.append(16)  
print(l)  
Op: [12, 13, 14, 15, 16]
```

Insert:

- this method is used to add one value at a time.
- Value is added at given index position.

```
Example: l=[11,13,14,15]  
l.insert(1,12)  
print(l)  
op:[11, 12, 13, 14, 15]
```

extend:

- this method is used to add multiple values at time.
- It takes only one argument which is of collection data type.

Remove values into the list :

Remove:

- It is used to remove values into the list.
- If value occurs more than once then it returns first occurrence of value.
- If value is not available then it throughs value error.

```
Example: x=[11,12,13,14,15,12]  
x.remove(12)  
print(x)  
op:[11, 13, 14, 15, 12]
```


pop:

- It takes index as an argument.
- It returns removed value.
- Index is not given by default it takes index as -1.

Example: a=[20,30,40,50]

```
x=a.pop(1)
n=a.pop()
print(x) op:30
print(n) op:50
print(a) op:[20,40]
```

clear: this method is used to remove all values into the list.

Example: x=[1,2,3,4,5,6]

```
x.clear()
print(x) op: [ ]
```

index:

Example: n=[22,33,44,55]

```
print(n.index(33))
op:1
```

count:

Example: n=[22,33,44,55,22,33,22]

```
print(n.count(22))
op:3
```

sort:

- it works only on homogenous list.
- it is used to arrange the data from ascending and descending.

Example: # ascending order

descending order

```
x=[4,3,5,6,2,7,1,8,9]
x.sort()
print(x)
op:[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
x=[4,3,5,6,2,7,1,8,9]
x.sort(reverse=True)
print(x)
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

join: this method is used to convert list of substring into a string.

Example:

```
l=['hai','hello','how are you']
n='@'.join(l)
print(n)
```

op:hai@hello@how are you

set inbuilt methods:

add values into the set:

add: this method is used to add values into the set.

Example: `i={1,2,3}`
`i.add(4)`
`print(i)`
op:{1, 2, 3, 4}

```
# empty set
n=set()
n.add(1)
n.add('vinay')
n.add(True)
n.add(2.0)
n.add(2+2j)
print(n)
op:{(2+2j), 1, 2.0, 'vinay'}
```

Update: this method is used to add multiple values at a time.

Example: `n={22,33,44}`
`n.update('vinay',[1,33,44,1])`
`print(n)`
op:{'v', 33, 1, 'a', 44, 'y', 'n', 'i', 22}

```
n={22,33,44}
n.update('vinay',[1,33,44,1])
print(n)
op:{'v', 33, 1, 'a', 44, 'y', 'n', 'i', 22}
```

remove values into the set:

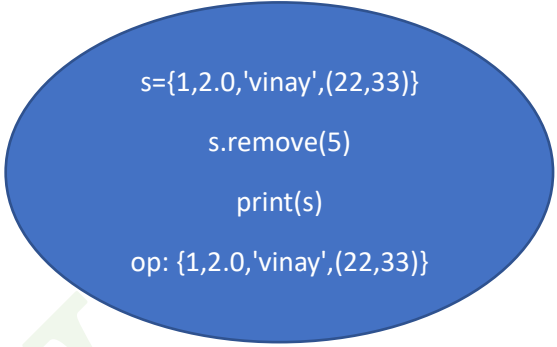
remove: this method is used to remove values in set if value is not available in set it will through key error.

Example: `s={1,2.0,'vinay',(22,33)}`
`s.remove('vinay')`
`print(s)`
op:{1, 2.0, (22, 33)}

```
s={1,2.0,'vinay',(22,33)}
s.remove(5)
print(s)
op: KeyError
```

Discard: this method is used to remove values in set if value is not available in set it will not through any error.

Example: `s={1,2.0,'vinay',(22,33)}`
`s.remove('vinay')`
`print(s)`
`op:{1, 2.0, (22, 33)}`



```
s={1,2.0,'vinay',(22,33)}  
s.remove(5)  
print(s)  
op: {1,2.0,'vinay',(22,33)}
```

pop:

- it returns the removed value and it will not accept any argument.
- it removes the random value from set.

Example: `x={1,2.0,'vinay',(22,33)}`
`x.pop()`
`print(x)`
`op:{2.0, (22, 33), 'vinay'}`

clear: this method is used to remove all values in set.

Example: `s={1,2.0,'vinay',(22,33)}`
`s.clear()`
`print(s)`
`op:set()`

dictionary inbuilt methods:

add key value pair into the dictionary:

variable name[key]=value

Example: `n={}`
`n['a']=10`
`n['b']=20`
`n[1]=20`
`n[(22,33)]=30`
`n[2.0]=40`
`n['a']=20`
`print(n)`

```
op: {'a': 20, 'b': 20, 1: 20, (22, 33): 30, 2.0: 40}
```

update: this method is used to add multiple key value pair to the dictionary.

```
Example: x={'a':10,'b':20}
         y={'c':30,'d':40}
         x.update(y)
         print(x)
         print(y)
         op: {'a': 10, 'b': 20, 'c': 30, 'd': 40}
         op: {'c': 30, 'd': 40}
```

remove key value pair into the dictionary:

pop:

- pop takes key as an argument
- it returns the value of key provided, if key is not in dictionary it throws key error.

```
Example: n={'a':10,'b':20,'c':30}
         m=n.pop('a')
         print(m)
         print(n)
         x=n.pop('d')
         print(x) op: key error
         op: 10
         op: {'b': 20, 'c': 30}
```

get:

- get method is used to return the values of given key.
- If key is not in dictionary it default value.
- If default value is not given it returns none.

```
Example: g={'a':10,'b':20,'c':30,'d':40}
         print(g.get('a'))
         print(g.get('f',100))
         print(g.get('g'))

         op: 10
         op: 100
         op: None
```

setdefault:

- it returns the value of given key.
- if key is not in dictionary it returns the default value and update the dictionary with key value pair.

Example: `s={'a':10,'b':20,'c':30,'d':40}`
`print(s.setdefault('d'))`
`print(s.setdefault('f',100))`
`print(s)`
op:40
op:100
op:{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'f': 100}

fromkeys:

- from keys is used to create a dictionary with same values for all given keys.
- If value is not given then dictionary of keys are updated with None key word.

Example: `d={}`
`x=d.fromkeys('abc',[1,2,3])`
`print(x)`
op:{'a': [1, 2, 3], 'b': [1, 2, 3], 'c': [1, 2, 3]}

type casting: convert one data type to another data type it is called type casting.

boolean

Int

`Int(15.5)` op:15
`Int(True)` op:1
`Int(3+2j)` op: type error
`Int(11.8)` op:11
`Int('10')` op: 10
`Int('11.8')` op: type error

float

`float(11)` op: 11.0
`float(2+4j)` op: type error
`float(True)` op:1.0
`float('12.5')` op: 12.5
`float('12')` op: 12.0

complex:

`A=10 print(bool(A))` op: true
`A=0 print(bool(A))` op: False
`A=0.0001 print(bool(A))` op: True
`B=0.1 print(bool(B))` op: True
`C=0.0 print(bool(c))` op: False
`C=3+4j print(bool(C))` op: True
`D='_' print(bool(D))` op: True
`A=[1,2,3,4] print(bool(A))` op: True
`A=[0] print(bool(A))` op: False
`A=(0) print(bool(A))` op: False
`A=(0,) print(bool(A))` op: True
`B={11,2,3,4,5} print(bool(B))` op: True
`d=set() print(bool(d))` op: False

```
complex(2.5,5.7) op:(2.5+5.7j)
complex(True,True) op: (1+1j)
complex(6) op: (6+0j)
complex('2') op:(2+0j)
complex('2', '5') op: type error
```

string: we will convert all data types into string.

List: we can't convert single value data types into list.

Example: a=10
 Print(list(a))
 Op: type error

We will convert all collection data types into list.

```
String to list
a='vinay'
print(list(a))
print(len(a))
op:['v','l','n','a','y']
op: 5
```

```
Tuple to list
n=(1,2,3,4,5)
print(list(n))
print(len(n))
op:[1, 2, 3, 4, 5]
op:5
```

```
Set to list
x={1,2,3,4,5}
print(list(x))
print(len(x))
op:[1, 2, 3, 4, 5]
op:5
```

```
Dictionary to list
y={'a':10, 'b':20, 'c':30}
print (list (y) )
print (len (y) )
op:['a', 'b', 'c']
op:3
```

tuple: we can't convert single value data types into tuple.

Example: a=10
 Print(tuple(a))
 Op: type error

We will convert all collection data types into tuple.

```
String to tuple
a='vinay'
print(tuple(a))
print(len(a))
op>('v','l','n','a','y')
op: 5
```

```
list to tuple
n=[1,2,3,4,5]
print(tuple(n))
print(len(n))
op:(1, 2, 3, 4, 5)
op:5
```

```
Set to tuple
x={1,2,3,4,5}
print(tuple(x))
print(len(x))
op:(1, 2, 3, 4, 5)
op:5
```

set: we can convert dictionary into set.
Example:

```
Dictionary to set
y={'a':10, 'b':20, 'c':30}
print(tuple(y))
print(len(y))
op:('a', 'b', 'c')
op:3
```

We will convert all collection data types into set.

String to set
a='vinay'
print(set(a))
print(len(a))
op: {'v', 'i', 'n', 'a', 'y'}
op: 5

Tuple to set
n=(1,2,3,4,5)
print(set(n))
print(len(n))
op: {1, 2, 3, 4, 5}
op: 5

list to set
x=[1,2,3,4,5]
print(set(x))
print(len(x))
op: {1, 2, 3, 4, 5}
op: 5

```
Dictionary to list
y={'a':10, 'b':20, 'c':30}
print(set(y))
print(len(y))
op: {'a', 'b', 'c'}
op: 3
```

dictionary: we can't convert single value data types into dictionary.

Example: a=10
Print(dict(a))
Op: type error

- We can't convert string to dictionary

List to dictionary
a=[(1,2),['a',4]]
print(dict(a))
print(len(a))
op: {1: 2, 'a': 4}
Op: 2

tuple to dictionary
a=((1,2),['a',4])
print(dict(a))
print(len(a))
op: {1: 2, 'a': 4}
Op: 2

set to dictionary
a={(1,2), 'ab'}
print(dict(a))
print(len(a))
op: {1: 2, 'a': 'b'}
Op: 2

Conditional statements

In python there are 3 conditional statements are there **if,elif,else.**

If:

- if given condition is true then control goes to if block otherwise control goes outside block.
- for if block else block is not mandatory

Else:

- if given condition is false then control goes to else block.
- For else block if block is mandatory.

Elif:

- If previous condition is false then control goes to elif block.
- For elif block if block is mandatory and else block is not mandatory.
- For one if block we can write multiple elif blocks.

input method:

- This method is used to take input at run time.
- Input method accept values in the form of string.

X=input('enter the value of x')

Print(type(of(x)) op: str

To convert int

X=int(input('enter the value of x'))

Print(type(x)) op: int

Range inbuilt function:

- Range function returns a sequence of numbers starting from 0 by default and increments one by one and stops before a specified number.
- Range is work only in list and tuple.

Ex: `x=list(range(1,10))`

Print(x) op:[1,2,3,4,5,6,7,8,9]

Ex: `x=tuple(range(1,10))`

Print(x) op:(1,2,3,4,5,6,7,8,9)

Ex: `x=list(range(-1,-5,-1))`

Print(x) op: [-1,-2,-3,-4]

Ex: `y=list(range(-10,-1,-1))`

Print(x) op: [-10,-9,-8,-7,-6,-5,-4,-3,-2]

Loops in python:

In python there are two types of loops are there

1)for ,2)while

1)for loop:

- if we know the iterations then we use for loop.
- Starting point of execution and ending point execution both has to be same.
- No.of iterations is same as the sequence.

for i in range(1,5):

print(i,end=' ') op:1,2,3,4

```
for b in range(1,5):  
    print('vinay')
```

```
op:  
vinay  
vinay  
vinay  
vinay
```

```
for a in range(1,5):  
    print(a)
```

```
op:  
1  
2  
3  
4
```

```
for i in range(5):  
    print(i)
```

```
op:  
0  
1  
2  
3  
4
```

```
X='vicky'  
for a in x:  
    print(a)
```

```
op:  
v  
i  
c  
k  
y
```

```
n=[22,33,44,55]  
for i in n:  
    print(i)
```

```
op:  
22  
33  
44  
55
```

```
X={'a':10,'b':20,'c':30}  
for a in x:  
    print(a)
```

```
a  
b  
c
```

```
X={'a':10,'b':20,'c':30}
```

```
for a in x.values():  
    print(a)
```

```
op:  
10  
20  
30
```

```
for k,v in x.items():  
    print(f' {k}={v}')
```

```
(a,10)  
(b,20)  
(c,30)
```

2)while loop:

- If we don't know the iterations then we use while loop.
- If condition is true then control enters the loop otherwise control goes outside the loop.

```
n=1
while n<=5:
    print(n)
    n+=1
op:
1
2
3
4
```

```
n=5
while n>=1:
    print(n)
    n-=1
op:
5
4
3
2
```

```
n=1
while n<=5:
    print('vinay')
    n+=1
op:
vinay
vinay
vinay
vinay
```

Control statements:

In python there are 3 control statements are there.

1)break 2)continue 3)pass

1)**break**: if given condition is satisfied then control goes outside the loop.

```
for a in range(1,6):
    if a==4:
        break
    print(a)
op:
1
2
3
```

2.continue: if control reaches continue it will skip the current iteration and control goes next iteration.

```
for a in range(1,6):  
    if a==4:  
        continue  
    print(a)  
op:  
1  
2  
3  
5
```

Pass: pass will not affect the control flow of a program. it is used to create an empty block otherwise it will through syntax error.

```
for a in range(1,6):  
    if a==4:  
        pass  
    print(a)  
op:  
1  
2  
3  
4  
5
```

programs

wap to print hieghest among 4 numbers.

a,b,c,d=10,20,30,40

if a>b:

 if a>c:

 if a>d:

 print('a is highest')

 else:

 print('d is highest')

else:

 if b>c:

 if b>d:

 print('b is highest')

 else:

 print('d is highest')

else:

 if c>d:

 print('c is highest')

 else:

 print('d is highest')

op: d is highest

```
n=int(input('enter the value:'))
```

```
if n%3==0 and n%5==0:
```

```
    print('fizz buzz')
```

```
elif n%3==0:
```

```
    print('fizz')
```

```
elif n%5==0:
```

```
    print('buzz')
```

```
year=int(input('eter the year:'))
```

```
if year%4==0 and year%100!=0 or year%400==0:
```

```
    print('it is a leap year')
```

```
else:
```

```
    print('is not leap year')
```

```
s=int(input('enter value:'))
```

```
st=1
```

```
for x in range(s):
```

```
    for y in range(st):
```

```
        print("*",end=" ")
```

```
    print()
```

```
    if x<s//2:
```

```
        st=st+1
```

```
    else:
```

```
        st-=1
```

```
enter value: 5
```

```
*
```

```
**
```

```
***
```

```
**
```

```
*
```

pattern programs

```
s=int(input('enter value:'))
st=1
for x in range(s):
    for y in range(st):
        print("*",end=" ")
    print()
    if x<s//2:
        st=st+1
    else:
        st-=1
```

```
enter value: 5
*
* *
* * *
* *
*
```

```
s=int(input('enter value:'))
st=1
sp=s//2
for x in range(s):
    for y in range(sp):
        print(" ",end=" ")
    for z in range(st):
        print("*",end=" ")
    print()
    if x<s//2:
        st=st+1
        sp=sp-1
    else:
        st-=1
        sp+=1
```

```
enter value: 5
*
* *
* * *
* *
*
```

```
for x in range(6):
    for y in range(6):
        print('*',end=' ')
    print()
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

```
n=int(input('enter the value:'))
st=n
for a in range(n):
    for b in range(st):
        print('*',end=' ')
    print()
    st-=1
enter the value: 6
* * * * *
* * * * *
* * * *
* * *
* *
*
```

```
n=int(input('enter the value:'))
st=1
for a in range(n):
    for b in range(st):
        print('*',end=' ')
    print()
    st+=1
enter the value: 7
*
* *
* * *
* * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

```

st=1
n=5
while n!=0:
    print(' *(n-1)+'* '*st)
    st+=1
    n-=1
    *
    * *
    * * *
    * * * *
    * * * * *

sp=0
n=5
while n!=0:
    print(' '*sp,'* '*((n*2)-1))
    sp+=1
    n-=1

* * * * *
* * * * *
* * * *
* * *
*

sp=0
n=5
while n!=0:
    print(' '*sp,'* '*((n*2)-1))
    sp+=1
    n-=1

* * * * *
* * * * *
* * * *
* * *
*

n=1
while n<=5:
    print('* '*5)
    n=n+1
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *

```

```

n=11
st=1
sp=n//2
for a in range(n):
    for b in range(sp):
        print(' ',end=' ')
    for c in range(st):
        if c==0 or c==st-1:
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
    if a<n//2:
        st+=2
        sp-=1
    else:
        st-=2
        sp+=1
    *
    * *
    * * *
    * * * *
    * * * * *
    * * * * *
    * * * *
    * * *
    * *
    *

n=7
st=1
sp=n//2
for a in range(n):
    for b in range(sp):
        print(' ',end=' ')
    for c in range(st):
        if c==0 or c==st-1:
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
    if a<n//2:
        st+=1
        sp-=1
    else:
        st-=1
        sp+=1
    *
    * *
    * * *
    * * * *
    * * * *
    * * *
    *

```

```

n=5
st=1
for a in range(n):
    num=st
    for b in range(st):
        print(num,end=' ')
        num=num+1
    print()
    st=st+1

```

```

1
2 3
3 4 5
4 5 6 7
5 6 7 8 9

```

```

n=5
st=1
for a in range(n):
    num=st
    for b in range(st):
        print(num,end=' ')
        num=num+1
    print()
    st=st+1

```

```

1
2 3
3 4 5
4 5 6 7
5 6 7 8 9

```

```

n=4
st=1
sp=n-1
for a in range(n):
    num=st
    for b in range(sp):
        print(' ',end=' ')
    for c in range(st):
        print(num,end=' ')
        num=num-1
    print()
    num+=1
    st=st+2
    sp=sp-1
    1
    3 2 1
    5 4 3 2 1
    7 6 5 4 3 2 1

```

```

n=4
st=1
sp=n-1
for a in range(n):
    num=1
    for b in range(sp):
        print(' ',end=' ')
    for c in range(st):
        print(num,end=' ')
        num=num+1
    print()
    st=st+2
    sp=sp-1
    1
    1 2 3
    1 2 3 4 5
    1 2 3 4 5 6 7
n=5

```

```

n=5
st=5
for a in range(n):
    num=st
    for b in range(st):
        print(num,end=' ')
        num+=1
    print()
    st-=1
    5 6 7 8 9
    4 5 6 7
    3 4 5
    2 3
    1
n=5
num=1
for a in range(n):
    for b in range(n):
        print(num,end=' ')
    print()
    num+=1
    1 1 1 1 1
    2 2 2 2 2
    3 3 3 3 3
    4 4 4 4 4
    5 5 5 5 5

```



```

r=5
sp=r//2
st=1
for a in range(r):
    for b in range(sp):
        print(' ',end=' ')
    for c in range(st):
        print('*',end=' ')
    print()
    if a<r//2:
        st+=2
        sp-=1
    else:
        st-=2
        sp+=1
    *
    * * *
    * * * * *
    * * *
    *
r=5
st=1
sp=n//2
for a in range(r):
    num=st
    for b in range(sp):
        print(' ',end=' ')
    for c in range(st):
        print(num,end=' ')
    print()
    if a<r//2:
        st+=2
        sp-=1
    else:
        st-=2
        sp+=1
    1
    3 3 3
    5 5 5 5 5

```

Number programs:

write a program to check given number is prime number or not.

```
n=int(input('enter n value:'))
```

```
if n>1:
```

```
    for a in range(2,n//2+1):
```

```
        if n%a==0:
```

```
            print("is not prime")
```

```
            break
```

```
else:
```

```
    print('is prime')
```

```
else:
```

```
    print('is not prime')
```

write a program to check given number is composite number or not.

```
n=int(input('enter the value:'))
```

```
for x in range(2,n//2+1):
```

```
    if n%x==0:
```

```
        print(f"{n} is composite")
```

```
        break
```

```
else:
```

```
    print(f"{n} is not composite")
```

write a program to print factors of a given number.

```
n=int(input('enter the value:'))
```

```
for x in range(1,n+1):
```

```
    if n%x==0:
```

```
        print(x)
```

write a program to find sum of digits in a given number.

```
n=55677
```

```
num=n
```

```
res=0
```

```
while n!=0:
```

```
    rem=n%10
```

```
    res=res+rem
```

```
    n=n//10
```

```
print(f"{num} value is sum of {res}")
```

write a program to check given number is niven number or not.

```
n=8447
```

```
num=n
```

```
res=0
```

```
while n!=0:
```

```
    rem=n%10
```

```
    res=res+rem
```

```
    n=n//10
```

```
if n%res==0:
```

```
    print(f"{num} is niven")
```

```
else:
```

```
    print(f"{num} is not niven")
```

write a program to find product of digits in given a number.

```
a=2345
```

```
num=a
```

```
res=1
```

```
while a!=0:
```

```
    rem=a%10
```

```
    res=res*rem
```

```
    a=a//10
```

```
print(f"{num} value product is {res}")
```

write a program to print prime digits in a given number.

```
n=1245746
```

```
while n!=0:
```

```
    rem=n%10
```

```
    n=n//10
```

```
    if rem>1:
```

```
        for x in
```

```
            range(2,rem//2+1):
```

```
                if rem%x==0:
```

```
                    break
```

```
            else:
```

```
                print(rem)
```

```
## write a program to print
prime numbers from 20,30.
```

```
for a in range(20,31):
```

```
    if a>1:
```

```
        for v in range(2,a//2+1):
```

```
            if a%v==0:
```

```
                break
```

```
        else:
```

```
            print(a)
```

```
## write a program to check
given number is armstrong
number or not.
```

```
n= 371
```

```
p=len(str(n))
```

```
res=0
```

```
num=n
```

```
while n!=0:
```

```
    rem=n%10
```

```
    res=res+rem**p
```

```
    n=n//10
```

```
if res==num:
```

```
    print("is armstrong num")
```

```
else:
```

```
    print("is not arm strong num")
```

```
## write a program to odd even
digits and subtract odd digits in
a given number
```

```
n=2358746
```

```
num=n
```

```
res=0
```

```
while n!=0:
```

```
    rem=n%10
```

```
    n=n//10
```

```
    if rem%2==0:
```

```
        res=res+rem
```

```
    else:
```

```
        res=res-rem
```

```
print(res)
```

```
## wap to check given number is
disarum number or not.
```

```
n=135
```

```
p=len(str(n))
```

```
num=n
```

```
res=0
```

```
while n!=0:
```

```
    rem=n%10
```

```
    res=res+rem**p
```

```
    p=p-1
```

```
    n=n//10
```

```
if num==res:
```

```
    print("disarum number")
```

```
else:
```

```
    print("is not dissrum num")
```

```
## wap a program to riverse the given
number.
```

```
n=123
```

```
num=n
```

```
res=0
```

```
while n!=0:
```

```
    rem=n%10
```

```
    res=res*10+rem
```

```
    n=n//10
```

```
print(f" revers of{num} is {res}")
```

```
## wap to check given number is
palindrome number or not.
```

```
n=121
```

```
num=n
```

```
res=0
```

```
while n!=0:
```

```
    rem=n%10
```

```
    res=res*10+rem
```

```
    n=n//10
```

```
if num==res:
```

```
    print("is polindrom number")
```

```
else:
```

```
    print("is not polindrom number")
```

```
## wap to print factorial of a given
number
```

```
n=5
```

```
fact=1
```

```
for a in range(1,n+1):
```

```
    fact=fact*a
```

```
print(fact)
```

```
## wap to check given number is
special number or not.
```

```
n=145
```

```
num=n
```

```
res=0
```

```
while n!=0:
```

```
    rem=n%10
```

```
    n=n//10
```

```
    fact=1
```

```
    for a in range(1,rem+1):
```

```
        fact=fact*a
```

```
    res=res+fact
```

```
if num==res:
```

```
    print("is a special number")
```

```
else:
```

```
    print("is not special number")
```

```
## wap to check given number is
polyprime number or not.
```

```
n=151
```

```
num=n
```

```
res=0
```

```
while n!=0:
```

```
    rem=n%10
```

```
    res=res*10+rem
```

```
    n=n//10
```

```
if num==res:
```

```
    if num>1:
```

```
        for a in range(2,num//2+1):
```

```
            if num%a==0:
```

```
                print("is not poly prime")
```

```
                break
```

```
            else:
```

```
                print("is poly prime")
```

wap to print all 3 digit disrum numbers.

for a in range(100,1000):

```
    num=a
    res=0
    p=len(str(a))
    while a!=0:
        rem=a%10
        a=a//10
        res=res+rem**p
        p=p-1
    if num==res:
        print(res)
```

wap to check given number is strong number or not.

```
n=6
num=n
res=0
for a in range(1,n):
    if n%a==0:
        res=res+a
if num==res:
    print("strong")
else:
    print("not strong")
```

wap to convert decimal to binary.

```
n=14
res=0
x=1
while n!=0:
    rem=n%2
    n=n//2
    res=res+rem*x
    x=x*10
print(res)
```

wap to check given number is emirp or not.

```
n=13
num=n
res=0
while n!=0:
    rem=n%10
    res=res*10+rem
    n=n//10
if num!=res:
    if num>1:
        for a in range(2,num//2+1):
            if num%a==0:
                print("not emirp")
                break
            else:
                for b in range(2,num//2+1):
                    if num%b==0:
                        print('not emirp')
                        break
                    else:
                        print("is emirp")
else:
    print("is not emirp")
```

wap to convert binary to decimal

```
n=1110
res=0
p=0
while n!=0:
    rem=n%10
    n=n//10
    res=res+rem*2**p
    p=p+1
print(res)
```

wap to replace value

```
n=23537
res=0
p=1
while n!=0:
    rem=n%10
    n=n//10
    if rem==3:
        res=res+6*p
    else:
        res=res+rem*p
    p=p*10
print(res)

# WAP TO CHECK GIVEN NUMBER IS AUTOMORPHIC NUMBER OR NOT
a=5
s=a*a
l=len(str(s))
res=s%10**1
if res==a:
    print('automorphic')
else:
    print('not automorphic')
n=192
s=str(n*1)+str(n*2)+str(n*3)
check='123456789'
for a in s:
    if str(a) not in check:
        print('it is not fascinating number')
        break
else:
    print('it is fascinating number')
```

```
# wap to check given number is spy
number or not
```

```
n=6
```

```
res1=0
```

```
res2=1
```

```
while n!=0:
```

```
    rem=n%10
```

```
    res1+=rem
```

```
    res2*=rem
```

```
    n=n//10
```

```
if res1==res2:
```

```
    print('it is spy number')
```

```
else:
```

```
    print('it is not spy number')
```

```
x=5
```

```
st=1
```

```
for a in range(x):
```

```
    for b in range(st):
```

```
        if b==0 or a==x-1 or
```

```
a==b:
```

```
            print('*',end=' ')
```

```
        else:
```

```
            print(' ',end=' ')
```

```
    print()
```

```
    st+=1
```

```
*
```

```
**
```

```
* *
```

```
*  *
```

```
****
```

```
n=5
```

```
for a in range(n):
```

```
    for b in range(n):
```

```
        if a==0 or a==n-1 or
```

```
b==n-1 or a==n//2:
```

```
            print('*',end=' ')
```

```
    print()
```

```
*****
```

```
*
```

```
*****
```

```
*
```

```
*****
```

```
def fib(n):
```

```
    fib=[0,1]
```

```
    for a in range(2,n):
```

```
        fib.append(fib[a-2]+fib[a-1])
```

```
    return fib
```

```
a=fib(7)
```

```
f=a[::2]
```

```
s=a[1::2]
```

```
t=s[::-1]
```

```
z=f+t
```

```
n=7
```

```
space=0
```

```
for a in range(n):
```

```
    print(' '*space,end='')
```

```
    print(z[a])
```

```
    space=space+1 if a<n//2
```

```
else space-1
```

```
0
```

```
1
```

```
3
```

```
8
```

```
5
```

```
2
```

```
1
```

```
l=[4,2,6,3]
```

```
for x in range(max(l)):
```

```
    for y in range(len(l)):
```

```
        if l[y]>x:
```

```
            print('*',end=' ')
```

```
        else:
```

```
            print(' ',end=' ')
```

```
    print()
```

```
****
```

```
****
```

```
* **
```

```
* *
```

```
*
```

```
*
```

```
s=[{'service':'aws'},{'service':'azure'},{'os':'windows'},{'os':'linux'}]
```

```
d={}
```

```
for a in s:
```

```
    for k,v in a.items():
```

```
        if k not in d:
```

```
            d[k]=[v]
```

```
        else:
```

```
            d[k]+=[v]
```

```
print(d)
```

```
{'service': ['aws', 'azure'], 'os': ['windows', 'linux']}
```

list programs

wap to print even numbers in given a list

```
n=[11,22,14,15,17,18]
for a in range(0,len(n)):
    if n[a]%2==0:
        print(n[a])
```

wap to print even index position of a list

```
s=[11,12,14,15,18,22,19]
for a in range(0,len(s)):
    if a%2==0:
        print(s[a])
```

wap to print even num and odd in given a list

```
n=[11,22,33,44,66,14,17,19]
even=[]
odd=[]
for a in range(0,len(n)):
    if n[a]%2==0:
        even.append(n[a])
    else:
        odd.append(n[a])
```

print(even+odd)

wap to remove duplicates in given a list.

```
n=[11,33,11,44,22,11,22,33,44]
res=[]
for a in range(0,len(n)):
    if n[a] not in res:
        res.append(n[a])# or
        res=res+[n[a]]
print(L)
```

wap to print heighest vallue in a list

```
n=[9,14,3,6,7]
h=n[0]
for a in range(1,len(n)):
    if h<n[a]:
        h=n[a]
print(h)
```

wap to sort values in list without using sort method

```
s=[6,4,3,7,2,8,5]
for a in range(0,len(s)):
    for b in range(0,len(s)-1-a):
        if s[b]>s[b+1]:
            s[b],s[b+1]=s[b+1],s[b]
print(s)
```

wap to print second heighest value in given list

```
s=[11,14,15,17,5]
for a in range(0,2):
    for b in range(0,len(s)-1-a):
        if s[b]>s[b+1]:
            s[b],s[b+1]=s[b+1],s[b]
print(s[-2])
```

wap to print unique values in given a list

```
a=[1,2,3,1,3,4,5,6]
for x in range(0,len(a)):
    if a.count(a[x])<=1:
        print(a[x])
```

wap to count each charecter in a list

```
a=[1,2,3,1,3,4,5,6]
l=[]
for x in range(0,len(a)):
    if a[x] not in l:
        l.append(a[x])
    print(f'{a[x]}={a.count(a[x])}')
# wap to to sort list (selection sort)
```

```
n=[21,12,15,10,19,16]
for a in range(0,len(n)-1):
```

```
    m=a
    for b in range(a+1,len(n)):
        if n[m]>n[b]:
            m=b
    n[a],n[m]=n[m],n[a]
```

print(n)

wap to reverse the given string without using slicing .

```
s='vinay'
res=""
for a in range(-1,-(len(s)+1),-1):
    res=res+s[a]
print(res)
```

wap to check given string is palindrome or not(reverse ga chadivina same ga untey it is palindrome)

```
v='madam'
res=""
for i in range(-1,-(len(v)+1),-1):
    res=res+v[i]
if res==v:
    print('it is palindrome')
else:
    print('not palindrome')
```

wap to reverse the given string without using slicing .

```
s='vinay'
res=""
for a in range(-1,-(len(s)+1),-1):
    res=res+s[a]
print(res)

## wap to check given string is
palindrome or not(reverse ga
chadivina same ga untey it is
palindrome)
v='madam'
res=""
for i in range(-1,-(len(v)+1),-1):
    res=res+v[i]
if res==v:
    print('it is palindrome')
else:
    print('not palindrome')

## wap to print vowels in given
string by using index
s='engineering'
vow='aeiouAEIOU'
res=""
for i in range(0,len(s)):
    if s[i] in vow:
        res=res+s[i]
print(res)

## wap to print vowels in given
string without using index
s='engineering'
vow='aeiouAEIOU'
res=""
for i in s:
    if i in vow:
        res=res+i
print(res)
```

string programs

wap to remove duplicates in a given string

```
a='engineering'
res=""
for b in range(0,len(a)):
    if a[b] not in res:
        res=res+a[b]
print(res)

## wap to count vowels in given a
string
c='engineering'
count=0
vow='aeiouAEIOU'
for a in range(0,len(c)):
    if c[a] in vow:
        count=count+1
print(count)

## wap to convert given string into
uppercase
s='Vinay'
res=""
for i in range(0,len(s)):
    if 'a'<=s[i]<='z':
        res=res+chr(ord(s[i])-32)
    else:
        res=res+s[i]
print(res)
```

#wap to convert given string into all lower case charecters to uppercase and all uppercase to lowercase

```
x='vinay VICKY'
res=""
for y in range(0,len(x)):
    if 'a'<=x[y]<='z':
        res=res+chr(ord(x[y])-32)
    elif 'A'<=x[y]<='Z':
        res=res+chr(ord(x[y])+32)
    else:
        res=res+x[y]
print(res)

## WAP TO PRINT SUB
PALINDROMES IN GIVEN A SSTRIN.
n='ABCBA'
for a in range(0,len(n)):
    for b in range(a+1,len(n)+1):
        s=n[a:b]
        if s==s[::-1] and len(s)>1:
            print(s)

## wap to split given string without
using split method
s='we are groot'
L=[]
res=""
for a in range(0,len(s)):
    if s[a]!=' ':
        res+=s[a]
    else:
        L.append(res)
        res=""
L.append(res)
print(L)
```

WRITE A PROGRAM TO COUNT EACH CHARETER IN A GIVEN STRING.

```
s='ENGINEERING'
```

```
count=0
```

```
res=""
```

```
for a in range(0,len(s)):
```

```
    if s[a] not in res:
```

```
        res=res+s[a]
```

```
        print(f"{s[a]}={s.count(s[a])}")
```

pinadi idi same idi inko methods

```
s='engineering'
```

```
d={}
```

```
for a in range(0,len(s)):
```

```
    if s[a] not in d:
```

```
        d[s[a]]=1
```

```
    else:
```

```
        d[s[a]]+=1
```

```
for k,v in d.items():
```

```
    print(f"{k}={v}")
```

wap to count each charecter in given a string in dictionary

```
n='we want to attend class everyday because we want job'
```

```
m=n.split()
```

```
d={}
```

```
for a in range(0,len(m)):
```

```
    if m[a] not in d:
```

```
        d[m[a]]=m.count(m[a])
```

```
print(d)
```

wap to count special charecters in given a string

```
s="%^%#@^$^%@FYF*@YR@F"
```

```
count=0
```

```
ch="@#$$%^&*"
```

```
for a in range(0,len(s)):
```

```
    if s[a] in ch:
```

```
        count=count+1
```

```
print(count)
```

wap to print most freequently occured charecter in given a stirrng

```
n='engineering'
```

```
d={}
```

```
for a in range(0,len(n)):
```

```
    if n[a] not in d:
```

```
        d[n[a]]=1
```

```
    else:
```

```
        d[n[a]]+=1
```

```
val=list(d.values())
```

```
h=val[0]
```

```
for a in range(1,len(val)):
```

```
    if h<val[a]:
```

```
        h=val[a]
```

```
for k,v in d.items():
```

```
    if h==v:
```

```
        print(k)
```

wap to reverse the string without reversing wordds

```
s='vinay is good boy'
```

```
l=s.split()
```

```
for a in range(0,len(l)//2):
```

```
    l[a],l[len(l)-1-a]=l[len(l)-1-a],l[a]
```

```
res=' '.join(l)
```

```
print(res)
```

wap to print most freequently occured word in a string

```
n='we want to attend class everyday because we want job'
```

```
l=n.split()
```

```
d={}
```

```
for a in range(0,len(l)):
```

```
    if l[a] not in d:
```

```
        d[l[a]]=1
```

```
    else:
```

```
        d[l[a]]+=1
```

```
val=list(d.values())
```

```
h=val[0]
```

```
for a in range(1,len(val)):
```

```
    if h<val[a]:
```

```
        h=val[a]
```

```
for k,v in d.items():
```

```
    if h==v:
```

```
        print(k)
```

wap to print most freequently occured word in a string

```
n='we want to attend class everyday because we want job'
```

```
l=n.split()
```

```
d={}
```

```
for a in range(0,len(l)):
```

```
    if l[a] not in d:
```

```
        d[l[a]]=len(l[a])
```

```
val=list(d.values())
```

```
h=val[0]
```

```
for a in range(1,len(val)):
```

```
    if h<val[a]:
```

```
        h=val[a]
```

```
for k,v in d.items():
```

```
    if h==v:
```

```
        print(k)
```


wap to reverse the string without reversing wordds

s='vinay is good boy'

l=s.split()

for a in range(0,len(l)//2):

l[a],l[len(l)-1-a]=l[len(l)-1-a],l[a]

res=' '.join(l)

print(res)

wap to reverse each word in a string

s='vinay is good boy'

l=s.split()

for a in range(0,len(l)):

l[a]=l[a][::-1]

res=' '.join(l)

print(res)

s='aaabbccccbaaf'

res=""

count=1

for a in range(0,len(s)-1):

if s[a]==s[a+1]:

count+=1

else:

res+=str(count)+s[a]

count=1

res+=str(count)+s[a+1]

print(res)

wap to replace the charecteer without using replace method

s='abcdabcd'

res=""

for a in range(0,len(s)):

if s[a]=='a':

res+='@'

else:

res=res+s[a]

print(res)

wap to print all index position of given chareccter in a string

s='abcbasna'

ch='a'

for a in range(0,len(s)):

if s[a]==ch:

print(a)

t=int(input('enter table name:'))

for a in range(1,11):

print(t,'x',a,'=',t*a)

enter table name: 6

6 x 1 = 6

6 x 2 = 12

6 x 3 = 18

6 x 4 = 24

6 x 5 = 30

6 x 6 = 36

6 x 7 = 42

6 x 8 = 48

6 x 9 = 54

6 x 10 = 60

—

n='a3b5c7'

res=""

for a in range(0,len(n),2):

res+=n[a]+(chr(ord(n[a])+int(n[a+1])))

print(res)

adbgcj

l=[1,2,3,4,5,0,6,10,7,8,9]

e=[]

o=[]

d={}

for a in range(0,len(l)):

if l[a]%2==0:

e.append(l[a])

else:

o.append(l[a])

n=e+o

l=[]

for a in n:

if a>1:

for x in range(2,a//2+1):

if a%2==0:

break

else:

l.append(a)

d['even']=e

d['odd']=o

d['prime']=l

print(d)

output:

{'even': [2, 4, 0, 6, 10, 8], 'odd': [1, 3, 5, 7, 9], 'prime': [2, 3, 5, 7, 9]}

n='vinay'

res=""

l=[]

for a in range(-1,-(len(n)+1),-1):

res=res+n[a]

l.append(res)

print(l)

matrix

n=3

l=[[1,2,3],[4,5,6],[7,8,9]]

for i in range(n):

for j in range(n):

print(l[i][n-1-i],end=' ')

print()

3 6 9

2 5 8

1 4 7

```
n=3
l=[[1,2,3],[4,5,6],[7,8,9]]
for i in range(n):
    for j in range(n):
        print(l[n-1-j][i],end=' ')
    print()
7 4 1
8 5 2
9 6 3
```

```
n=3
l=[[1,2,3],[4,5,6],[7,8,9]]
t=0
for i in range(n):
    for j in range(n):
        t+=l[j][i]
print(t)

45
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    for b in range(3):
        print(l[a][b],end=' ')
    print()

1 2 3
4 5 6
7 8 9
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[a][0])
1
4
7
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[2][a])
7
8
9
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[0][a])
1
2
3
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[a][a])
1
5
9
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[3-a-1][3-a-1])
9
5
1
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[a][1])
2
5
8
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[a][2])
3
6
9
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[3-1-a][3-1-a])
9
5
1
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[a][3-1-a])
3
5
7
```

```
l=[[1,2,3],[4,5,6],[7,8,9]]
for a in range(3):
    print(l[3-1-a][a])
7
5
3
```

```
n=3
l=[[1,2,3],[4,5,6],[7,8,9]]
for i in range(n):
    for j in range(n):
        print(l[j][n-1-i],end=' ')
    print()
3 6 9
2 5 8
1 4 7
```

```
n=3
l=[[1,2,3],[4,5,6],[7,8,9]]
for i in range(n):
    for j in range(n):
        print(l[n-1-j][i],end=' ')
    print()
```

```
7 4 1
8 5 2
9 6 3
```

```
l1=[1,2,3,4,5]
l2=[1,2,3,4,5]
x=[]
for a in range(len(l1)):
    x.append(l1[a]+l2[a])
print(x)
```

```
[2, 4, 6, 8, 10]
```

```
n=4
st=4
for a in range(n):
    for b in range(st):
        if a==0 or a==3 or b==0
or b==3:
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
* * * *
*   *
*   *
* * * *
```

```
s1=input('enter string1')
s2=input('enter string2')
if sorted(s1)==sorted(s2):
    print(f'{s1} and {s2} are
anagrams')
else:
    print('not anagrams')
```

```
enter string1 rajuharivinay
enter string2 vinayhariraju
rajuharivinay and vinayharir
aju are anagrams
```

Functions



There are 2 types of functions are there.

- Inbuilt functions
- User defined functions

Function:

- It is a block of organized and reused code that we can executed when function is called.
- Function call is given after function declaration.
- Function is declared by using def keyword.

Ex: `def sample():`
 `print('vinay')`
`sample()`
op: vinay

return:

- Return is a key word it is used to return the values to the function call.
- By using return keyword we can return multiple values to the function call.
- It takes control flow from fuctional space to main space.

Ex: `def sample():`
 `print('vinay')`
 `return 100`
`sample()`
op: vinay
 100

```
def sum(a,b,c):
    return a+b+c
x=sum(10,20,30)
print(x)
y=sum(1,2,3)
print(y)
# or
def sum(a,b,c):
    return a+b+c
print(sum(10,20,30))
print(sum(1,2,3))
60
6
60
6
```

Types of arguments:

1. Positional arguments.
2. Keyword arguments
3. Default arguments
4. Variable length non keyword arguments
5. Variable length key word arguments

1.Positional arguments.

- In positional arguments we pass values to the functional call without name.
- While passing values we need to follow the order.
- no.o arguments in function call and function declaration both has to be same.

Ex : **def** details(name,age,gender):
 print(f'name : {name}')

```
print(f'age : {age}')
```

```
print(f'gender :{gender}')
```

details('vinay',25,'male')

```
Op:name : vinay
```

```
age : 25
```

```
gender :male
```

2.keyword arguments:

- in keyword arguments we pass values to the function call with specific variable name.
- no.of arguments in function call and function declaration both has to be same.

Ex: **def** details(name,age,gender):
 print(f'name : {name}')

```
print(f'age : {age}')
```

```
print(f'gender :{gender}')
```

details(gender='female',name='vinay',age=25)

```
Op:name : vinay
```

```
age : 25
```

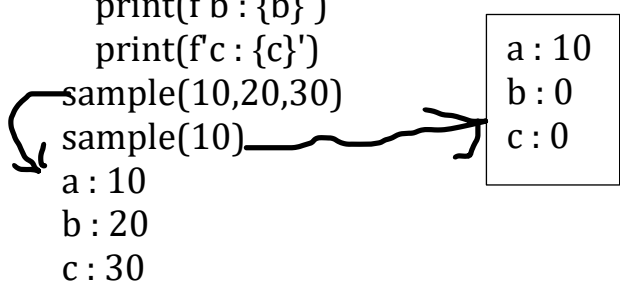
```
gender :female
```

3.default arguments:

- it takes default arguments when we are not pass any values to the function call.
- No.of arguments in functional call and function declaration no need to be same.

Ex: `def sample(a=0,b=0,c=0):`

```
    print(f'a : {a}')
    print(f'b : {b}')
    print(f'c : {c}')
sample(10,20,30)
sample(10)
```



a : 10
b : 0
c : 0

4.variable length keyword arguments:

- It allows a function to accept any no.of values to the function call.
- Variable name is prefixed with single star.

Ex: `def sample(*args):`
 `print(f'args : {args}')`
 `sample(10,20,30)`
 `sample()`
 `args : (10, 20, 30)`
 `args : ()`

5.variable length keyword arguments.

- It allows a function to accept any no.of key word arguments.
- Keyword arguments are stored in dictionary.
- Variable name is prefixed with double star(**)

Ex: `def star(**kwargs):`
 `print(f'kwargs : {kwargs}')`
 `star(a=10,b=20,c=30)`
 `star()`
 `kwargs : {'a': 10, 'b': 20, 'c': 30}`
 `kwargs : {}`

WAP TO PRINT GIVEN NUMBER IS PRIME OR NOT

```
def prime(n):  
    if n<=1:  
        return False  
    for a in range(2,n//2+1):  
        if n%a==0:  
            return False  
    return True
```

n=17

```
if prime(n):  
    print('it is prime')  
else:  
    print('not prime')
```

wap to print prime numbers in given range by using function

```
def prime(n):  
    if n<=1:  
        return False  
    for a in range(2,n//2+1):  
        if n%a==0:  
            return False  
    return True
```

for y in range(1,21):

```
    if prime(y):  
        print(y)  
  
    # wap to reverse the number by  
    using function
```

```
def reverse(n):  
    res=0  
    while n!=0:  
        rem=n%10  
        res=res*10+rem  
        n=n//10  
    return res  
print(reverse(12345))
```

wap to check given number is polyndrome or not by using function

```
def reverse(n):  
    res=0  
    while n!=0:  
        rem=n%10  
        res=res*10+rem  
        n=n//10  
    return res
```

n=121

```
num=n  
if reverse(n==num):  
    print('polyndrome')  
else:  
    print('not polyndrome')
```

wap to check the given number is polyprime or not

```
def prime(n):  
    if n<=1:  
        return False  
    for a in range(2,n//2+1):  
        if n%a==0:  
            return False  
    return True
```

```
def poly(n):  
    res=0  
    while n!=0:  
        rem=n%10  
        res=res*10+rem  
        n=n//10  
    return res
```

x=11

```
r=poly(x)  
if x!=r and prime(x):  
    print('polyprime')  
else:  
    print('not polyprime')
```

wap to print factorial of given number number

```
def factorial(n):  
    fact=1  
    for a in range(1,n+1):  
        fact=fact*a  
    return fact
```

print(factorial(5))

wap to check the given number is emrip number or not

```
def prime(n):  
    if n<=1:  
        return False  
    for a in range(2,n//2+1):  
        if n%a==0:  
            return False  
    return True
```

```
def poly(n):  
    res=0  
    while n!=0:  
        rem=n%10  
        res=res*10+rem  
        n=n//10  
    return res
```

x=11

```
r=poly(x)  
if x!=r and prime(n) and prime(r):  
    print('emrip number')  
else:  
    print('not emrip number')
```

wap to check given number is special number or not

```
def factorial(n):
```

```
    fact=1
```

```
    for a in range(1,n+1):
```

```
        fact=fact*a
```

```
    return fact
```

```
def special(n):
```

```
    res=0
```

```
    while n!=0:
```

```
        rem=n%10
```

```
        res=res+factorial(rem)
```

```
        n=n//10
```

```
    return res
```

```
s=40585
```

```
num=s
```

```
if special(num==s):
```

```
    print('special num')
```

```
else:
```

```
    print('not special')
```

wap to print 10th number fibnoccie series

```
def fib(n):
```

```
    f,s=0,1
```

```
    if n==1 or n==2:
```

```
        return n-1
```

```
    for a in range(0,n-2):
```

```
        t=f+s
```

```
        f=s
```

```
        s=t
```

```
    return t
```

```
print(fib(5))
```

wap to print first 10 fibnoccie numbers

```
def fib(x):
```

```
    f,s=0,1
```

```
    if x==1 or x==2:
```

```
        return x-1
```

```
    for a in range(0,x-2):
```

```
        t=f+s
```

```
        f=s
```

```
        s=t
```

```
    return t
```

```
for a in range(1,11):
```

```
    print(fib(a))
```

Recursion

Function call inside same function or function calling itself it is called recursion.

advantages

by using recursion we can avoid loops.

no. of lines of code can be reduced.

disadvantages

it takes lot of memory space while execution.

wap to print numbers from 1 to 10 by using recursion

```
def fun(n):
    if n==11:
        return
    print(n,end=' ')
    fun(n+1)
fun(1)
1 2 3 4 5 6 7 8 9 10
```

wap to print numbers from -1 to -11 by using recursion

```
def fun(n):
    if n== -11:
        return
    print(n,end=' ')
    fun(n-1)
fun(-1)
-1 -2 -3 -4 -5 -6 -7 -8 -9 -10
```

wap to find sum of digits in a number by using recursion

```
def sumdig(n):
    if n==0:
        return 0
    return n%10+sumdig(n//10)
sumdig(12345)
```

15

wap to reverse the given string.

```
def revstr(n):
    if n==-(len(s)+1):
        return ""
    return s[n]+revstr(n-1)
s='vinay'
print(revstr(-1))
yaniv
```

wap to print factorial of a given number

```
def factorial(n):
    if n==0 or n==1:
        return 1
    return n*factorial(n-1)
print(factorial(5))
120
```

wap to print 10th number fibonocci series

```
def fib(n):
    if n==1 or n==2:
        return n-1
    return fib(n-1)+fib(n-2)
print(fib(6))
```

```
def armstrong(n,p):
    if n==0:
        return 0
    return
(n%10)**p+armstrong(n//10,p)
n=153
p=len(str(n))
if armstrong(n,p)==n:
    print('armstrong number')
else:
    print('not armstrong number')
```

Armstrong number

```
def disarum(n,p):
    if n==0:
        return 0
    return
(n%10)**p+disarum(n//10,p-1)
n=135
p=len(str(n))
if disarum(n,p)==n:
    print('disarum')
else:
    print('not')
```

disarum

Name space

Global variables:

- Global variables are declared as global space or main space.
- Global variables can be accessed both global space as well as local space.

```
def sample():  
    print(f' local space : {z}')  
    z=500
```

```
sample()  
print(f' global space : {z}')  
    local space : 500  
    global space : 500
```

local variables:

- Local variables are declared as local space or functional space.
- Local variables can be access only local variables.

```
def sample():  
    a=100  
    print(f' local space : {a}')
```

```
sample()  
print(f' global space : {a}')
```

- Local variables can be used in global space by declaring local variables as global by using global keyword.
- Global declaration has to be done before assigning the value.

```
def sample():
    global a
    a=100
    print(f' local space : {a}')
    a=200
    sample()
    print(f' global space : {a}')
```

local space : 100
global space : 100

non local variables:

these variables are declared in nested function.

Non local variables are not bound to local are global space.

```
def outer():
    print('one')
    def inner():
        print('two')
        print('three')
    outer()
    one
    three
```

```
def outer():
    print('one')
    def inner():
        print('two')
        print('three')
    inner()
    outer()
    one
    three
    two
```

```
def outer():
    x=200
    print(f'non local:{x}')
    def inner():
        x=300
        print(f'local:{x}')
    inner()
    print(f'non local:{x}')
x=500
outer()
print(f'global:{x}')
non local:200
local:300
non local:200
global:500
```

-

```
def outer():
    x=200
    print(f'non local:{x}')
    def inner():
        x=300
        print(f'local:{x}')
    inner()
    print(f'non local:{x}')
x=500
outer()
print(f'global:{x}')
```

Lambda function:

lambda function is a function without name and it is also called as an anonymous function.

it takes more than one argument and returns only one value.

lambda function is used to execute simple mathematical operation.

```
n=lambda x,y:x+y
print(n(5,6))
11
```

```
n=lambda x,y:x+y
print(n('vinay','vicky'))
vinayvicky
```

```
n=lambda x,y:x//y
print(n(10,2))
5
```

```
n=lambda x,y:x%y
print(n(10,2))
0
```

```
n=lambda x,y:x*y
print(n(10,2))
20
```

```
n=lambda x,y:x**y
print(n(2,3))
8
```

```
from functools import reduce
l=[20,10,30,4]
n=reduce(lambda x,y:x-y,l)
print(n)
-24
```

```
l=[10,20,15,18,25,19]
x=reduce(lambda x,y:x if x>y
else y,l)
print(x)
25
```

Map function:

map function is used to access and modify the values in sequence or iterable data type.

map function returns always map object.

by using map function we can avoid loops.

no. of inputs and outputs both are same.

```
l=[2,3,4,5,6]
n=list(map(lambda n:n**2,l))
print(n)
[4, 9, 16, 25, 36]
```

```
def square(n):
    return n**2
l=[2,3,4,5,6]
n=list(map(square,l))
print(n)
[4, 9, 16, 25, 36]
```

Filter function:

filter function is used to filter the values based on user requirement.

filter function returns always filter object.

by using filter function we can avoid loops.

no. of input values and output values are not same.

```
l=[10,11,12,13,14,15]
n=list(filter(lambda n:n%2==0,l))
print(n)
[10, 12, 14]
```

```
def mod(n):
    return n%2==0
l=[10,11,12,13,14,15]
n=list(filter(mod,l))
print(n)
[10, 12, 14]
```

```
def prime(n):
    if n<=1:
        return False
    for a in range(2,n//2+1):
        if n%a==0:
            return False
    return True
l=[11,12,13,14,15]
x=list(filter(prime,l))
print(x)
[11, 13]
```

Module creation

a file

```
def fact(n):  
    t=1  
    if n==0 or n==1:  
        return 1  
    return n*fact(n-1)  
  
def fib(n):  
    if n==1 or n==2:  
        return n-1  
    return fib(n-1)+fib(n-2)  
  
def prime(n):  
    if n<=1:  
        return False  
    for a in range(2,n//2+1):  
        if n%a==0:  
            return False  
    return True  
  
if __name__=='__main__':  
    print(prime(5))  
    print(fib(6))  
    print(prime(3))
```

b file

```
import a  
  
x=a.fact(5)  
  
print(x)  
  
n=a.prime(4)
```

print(n)

oops

- Object oriented programming language(oops)
- We can write programs by using classes and objects.

Class: class is a blue print that is used to create an object.

Object:

- Object is real time entity that has state and behaviour.
- State is defined by using variables and behaviour is defined by using methods.
- Object is created by using class name.
- We can create any no.of objects by using class name.

```
class sample():  
    pass  
n=sample()  
print(n)  
print(type(n))  
op:<__main__.sample object at 0x00000274BA187610>  
op:<class '__main__.sample'
```

advantages of oops:

- Bottom to approach
- Data security(by using encapsulation)
- Code reusability(by using inheritance)
- Hiding the data(by using abstraction)

Types of variables:

➤ There are two types of variables are there.

1)class variables

2)object variables

1)class variables:

- Class variables are declared inside the class.
- We can access class variables by using class name as well object name.
- If we modify class variables by using class name it affect all the objects that belongs to a class
- If we modify class variables by using object name that affect only that particular object.

```
class jspyders():
    cname='python fullstac dev'
    tname='vinay'
ob1=jspyders()
ob2=jspyders()
print(ob1.cname,ob1.tname)
print(ob2.cname,ob2.tname)
print(jspyders.cname,jspyders.tname)

python fullstac dev vinay
python fullstac dev vinay
```

```
class jspyders():
    cname='python'
    tname='santhosh'
ob1=jspyders()
ob2=jspyders()
ob3=jspyders()

jspyders.cname='java'
ob1.tname='vinay'

print(ob1.cname,ob1.tname)
print(ob2.cname,ob2.tname)
print(ob3.cname,ob3.tname)

java vinay
java santhosh
java santhosh
```

```
class jspyders():
    cname='python'
    tname='santhosh'
ob1=jspyders()
ob2=jspyders()
ob3=jspyders()

ob1.cname='java'
ob1.tname='vinay'

print(ob1.cname,ob1.tname)
print(ob2.cname,ob2.tname)
print(ob3.cname,ob3.tname)

java vinay
python santhosh
python santhosh
```

```

class pypiders():
    cname='python development'
    tname='vinay'
ob1=pypiders()
ob2=pypiders()
def details(obj,name,age,gender,mno):
    obj.name=name
    obj.age=age
    obj.gender=gender
    obj.mno=mno
details(ob1,'vinay',21,'male',91193783892)
details(ob2,'madhu',22,'male',98798786787)
print(ob1.name,ob1.age,ob1.gender,ob1.mno)
print(ob2.name,ob2.age,ob2.gender,ob2.mno)
vinay 21 male 91193783892
madhu 22 male 98798786787

```

1)object variables:

- Object variables are declared inside the constructor
- We can access object variables only by using object name.

Constructor:

- Constructor is a special method that is called at the time of object creation.
- Constructor is used to initialize the object variables(members or attributes).
- It takes mandatory argument self that holds address of the object.

```

class a():
    c='vinay'
    def __init__(self):
        print('constructor called')
ob1=a()
ob2=a()
constructor called
constructor called

```



```
class pyspiders():
    cname='python development'
    tname='vinay'
    def __init__(self,name,age,gender,mno):
        self.name=name
        self.age=age
        self.gender=gender
        self.mno=mno
ob1=pyspiders('vinay',21,'male',91193783892)
ob2=pyspiders('madhu',22,'male',98798786787)
print(ob1.name,ob1.age,ob1.gender,ob1.mno)
print(ob2.name,ob2.age,ob2.gender,ob2.mno)

vinay 21 male 91193783892
madhu 22 male 98798786787
```

types of methods:

1. Object method
2. Class method
3. Static method

1)object method:

- object method is used to access and modify object variables(members or attributes).
- Object method is called(access) by using object name.
- It takes mandatory argument self that holds address of the object.

```
class bank():
    bname='SBI'
    ROI=0.07
    def __init__(self,name,mob,acno,bal):
        self.name=name
        self.mob=mob
        self.acno=acno
        self.bal=bal
    def checkbalance(self):
        print(f'{self.name}current account balance is {self.bal}')
    def deposit(self,amt):
        self.bal+=amt
        print('amount deposited')
    def withdrawl(self,amt):
        if self.bal>=amt:
            self.bal-=amt
            print('amount debit successfullly')
        else:
            print('insufficient')
ob1=bank('vinay',9133119384,69676479237923779,15000)
ob1.checkbalance()
ob1.deposit(10000)
ob1.withdrawl(2000)
vinaycurrent account balance is 15000
amount deposited
amount debit successfullly
```

2)class method:

- Class method is declared by using @classmethod decarator.
- class method is used to access and modify the class variables.
- Class method is called(access) by using class name as well object name.
- It takes mandatory argument cls that holds address of the class.

```
class bank():
    bname='ANDHARA BANK'
    ROI=0.07
    def __init__(self,name,mobn,accn,bal):
        self.name=name
        self.mobn=mobn
        self.accn=accn
        self.bal=bal
    @classmethod
    def changeROI(cls,n):
        cls.ROI=n
ob1=bank('amar',99434333,34333403430000,50000)
ob2=bank('vinay',88434333,3433340344564,40000)
ob3=bank('hari',77434333,35555403430000,30000)
ob4=bank('shiva',66434333,876633403430000,20000)
print(ob1.ROI,ob2.ROI,ob3.ROI,ob4.ROI)
ob1.changeROI(0.09)
print(ob1.ROI,ob2.ROI,ob3.ROI,ob4.ROI)
0.07 0.07 0.07 0.07
0.09 0.09 0.09 0.09
```

3.static method:

- Static method is declared by using @staticmethod decarator.
- In static method we can't access class attributes or object attributes.
- It is access by using class name and object name.
- It will not accept any mandatory argument.

class bank:

 bname='RBI'

 ROI=0.09

def __init__(self,name,mbn,accn,bal,pin):

 self.name=name

 self.mbn=mbn

 self.accn=accn

 self.bal=bal

 self.pin=pin

 @classmethod

def changeROI(cls,n):

 cls.ROI=n

def withdraw(self):

if self.pin==self.getpin():

 amt=int(input('enter amount:'))

if self.bal>=amt:

 self.bal-=amt

 print('amount debited sucessfully')

else:

 print('insufficient balence')

```

    else:
        print('incorrect pin')

    def checkbalance(self):
        if self.pin==self.getpin():
            print(f'{self.name} current balance is {self.bal}')
        else:
            print('incorrect pin')
    @staticmethod
    def getpin():
        return int(input('get pin:'))
ob1=bank('vinay',9133119385,8946234724779832749,50000,913
3)
ob1.checkbalance()
ob1.withdraw()
get pin: 9133
vinay current balance is 50000
get pin: 9133
enter amount: 50000
amount debited sucessfull

```

there are 3 access modifiers in python.

1)public access modifiers: by default all public variables are declared as public.

- We can access public variables to any where in the program.

2)protected variables:

- Protected variables are declared by using single underscore
ex: `_a, _b`
- We can access protected variables only with in same package.

3)private variables:

- private variables are declared by using double underscore ex: `__a,__b`.
- we can access private variables only with in a same class.

Getter method: this method is used to get the values of private data members out side the class.

Setter method: this method is used to modify the values of private data members outside the class

```
Class a()  
    def __init__(self):  
        self.__x=20  
    def getter(self):  
        return self.__x  
    def setter(self,n):  
        self.__x=n  
  
obj=a()  
print(obj.getter()) op:20  
obj.setter(50)  
print(obj.getter()) op:50
```

There are 4 pillers of concept

1.encaptulation

2.inheritance

3.polymorphism

4.abstraction

1.encapsulation:

- Binding data members and functionalities in a single unit it is called encapsulation.

- it is used to secure the data by declaring datamembers private to avoid accidental modification from unauthorized users.

2.inheritance: inheriting properties from one class to another class is called inheritance.

Types of inheritance:

1. single level inheritance.
2. Multilevel inheritance.
3. Multiple inheritance.
4. Hyrarichal inheritance
5. Hybrid inheritance.

1.single level inheritance: inheriting properties from parent class to child class is called single level inheritance.

```
class p():
    x=20
    y=30
class c(p):
    x=40
    z=30
obj1=c()
obj2=p()
print(obj1.z,obj1.x)
print(obj2.y,obj2.x)
30 40
30 20
```

```
class parent():
    def method1(self):
        print('m1 of parent class')
    def method2(self):
        print('m2 of parent class')
class child(parent):
    def method1(self):
        print('m1 of child class')
ob1=child()
ob1.method1()
ob1.method2()
m1 of child class
m2 of parent class
```

```
class bankv1():
    def __init__(self,name,mno,acno,bal):
        self.name=name
        self.mno=mno
        self.acno=acno
        self.bal=bal
    def details(self):
        print(f'name: {self.name}')
        print(f'mobile num: {self.mno}')
        print(f'account num: {self.acno}')
        print(f'balance: {self.bal}')
class bankv2(bankv1):
    def __init__(self,name,mno,acno,bal,mail,aadhar):
        bankv1.__init__(self,name,mno,acno,bal)
        self.mail=mail
        self.aadhar=aadhar
    def details(self):
        super().details()
        print(f'mail id :{self.mail}')
        print(f'aadhar no:{self.aadhar}')
obj=bankv2('vinay',913311,12339894949,50000,'vina
y@gmail.com',7285860)
obj.details()
name: vinay
mobile num: 913311
account num: 12339894949
balance: 50000
mail id :vinay@gmail.com
aadhar no:7285860
```


2.multilevel inheritance: inheriting properties from one child class to another child class is called multilevel inheritance.

```
class bank1():
    def __init__(self,name,mbn,acn,bal):
        self.name=name
        self.mbn=mbn
        self.acn=acn
        self.bal=bal
    def details(self):
        print(f'name:{self.name}')
        print(f'mbn:{self.mbn}')
        print(f'acno:{self.acn}')
        print(f'balance:{self.bal}')
class bank2(bank1):
    def __init__(self,name,mbn,acno,bal,aadhar,mail):
        bank1.__init__(self,name,mbn,acno,bal)
        self.aadhar=aadhar
        self.mail=mail
    def details(self):
        super().details()
        print(f'mail:{self.mail}')
        print(f'aadhar:{self.aadhar}')
class bank3(bank2):
    def __init__(self,name,mbn,acno,bal,aadhar,mail,pan):
        super().__init__(name,mbn,acno,bal,aadhar,mail)
        self.pan=pan
    def details(self):
        super().details()
        print(f'pan id:{self.pan}')
ob=bank3('vinay',9133119,98344499489,500000,285860
79397,'vinay@123','jjpj12435')
ob.details()
name:vinay
mbn:9133119
acno:98344499489
balance:500000
pan id:jjpj12435
```

3.multiple inheritance: inheriting properties from one child class to more than one parent class is called multiple inheritance.

```
class a():
    x=100
class b(a):
    z=100
class c(a):
    d=100
class d(b,c):
    pass
obj=d()
print(obj.x)
```

100

```
class parent1():
    def __init__(self):
        print('constructor of class parent1')
class parent2():
    def __init__(self):
        print('constructor of class parent2')
class child(parent1,parent2):
    def __init__(self):
        parent1.__init__(self)
        parent2.__init__(self)
        print('constructor child class')
```

obj=child()

constructor of class parent1

constructor of class parent2

constructor child class

4.hirarichal inheritance: inheriting properties from one Parent class to more than one child class is called hirarichal inheritance.

```
class a():
    x=100
class b(a):
    z=100
    y=200
class c(a):
    pass
obj=c()
print(obj.x)
100
```

5.hybrid inheritance: combination of more than one type of inheritance is called hybrid inheritance.

```
class a():
    x=100
class b(a):
    z=100
class c(a):
    d=100
class d(b,c):
    pass
obj=d()
print(obj.x)
100
```

method resolution order(MRO): the way of resolving methods and attributes is known as method resolution order.

```
class a():
    pass
class b():
    pass
class c(b,a):
    pass
print(c.mro())
```

Op: [<class '__main__.c'>, <class '__main__.b'>, <class '__main__.a'>, <class 'object'>]

3.polymorphism:

- One object behaving many forms is called polymorphism.
- By using both overloading and overriding we can achieve polymorphism.

Overloading:

- Overloading is also known as compile time polymorphism.
- Having same method names with different method signatures in a same class.
- In python overloading is not possible because it takes latest defined method.

```
class sample():
    def sum(self,a,b):
        return a+b
    def sum(self,a,b,c):
        return a+b+c
    def sum(self,a,b,c,d):
        return a+b+c+d
obj=sample()
print(obj.sum(2,3,4,4))
print(obj.sum(1,2,3))
```

op:13
Op: type error

```
from functools import
reduce
class sample():
    def sum(self,*args):
        return reduce(lambda
a,b:a+b,args)
obj=sample()
print(obj.sum(1,2,3,4))
print(obj.sum(1,2,3,4,3,4))
10
17
```

we can achieve overloading by using default argument.

```
class sample():
    def
sum(self,a=0,b=0,c=0,d=
0):
        return a+b+c+d
obj=sample()
print(obj.sum(1,2,3))
print(obj.sum(1,2,3,4))
print(obj.sum(1,2,3,4,))
6
10
10
```

Without using default argument we can achieve overloading in python below method.

```
class sample():
    def sum(self,*args):
        t=0
        for i in args:
            t+=1
        return t
ob=sample()
print(ob.sum(1,2,3,4))
print(ob.sum(1,2,3,4,6,5)
)
print(ob.sum(1,2,3))
4
6
3
```

Overriding:

- Overriding is also known as runtime polymorphism.
- In overriding inheritance is mandatory.
- Having same method names with same method signatures with in both parent and child class then parent class is overridden with child class.

```
class a():
    def __init__(self):
        print('inside class a')
class b(a):
    def __init__(self):
        print('inside class b')
        a.__init__(self)
obj=b()
inside class b
inside class a
```

magic methods:

```
class a():
    def __init__(self,x):
        self.x=x
    def __add__(self,other):
        return self.x+other.x
    def __sub__(self,other):
        return self.x-other.x
    def __mul__(self,other):
        return self.x*other.x
    def __pow__(self,other):
        return self.x**other.x
    def __le__(self,other):
        return self.x<=other
ob1=a(2)
ob2=a(3)
print(ob1+ob2) op:5
print(ob1*ob2) op:6
print(ob1**ob2) op:8
```

4.abstraction:

- In abstraction we will hide internal functionalities and show only basic implementation to the user.
- Abstract class is created by inheriting properties from ABC(abstract base class) from abc module.
- Abstract class can have both abstract method and concret method
- Abstract method can't have any implementation
- Abstract method is created by using @abstract method decarator from abc module.
- We can't create an object for abstract classss.
- All abstract method implementation we have to give child class

```
from abc import ABC ,abstractmethod
class animal(ABC):
    @abstractmethod
    def speek(self):
        pass
    def move(self):
        print('walking')
class dog(animal):
    def speek(self):
        print('bow')
class cat(animal):
    def speek(self):
        print('meow')
class cow(animal):
    def speek(self):
        print('OYE')
ob1=dog()
ob2=cat()
ob3=cow()
```

```
ob1.speek()
ob2.speek()
ob3.speek()
bow
meow
OYE
```

Function alising: it is a process of calling one function without using actual name of the function.

copying the address of function to new variable.

Decarator: decarator is a designed pattern in python that allows new functionalities to a function or class without modifying it.

```
def outer(arg):
    def inner():
        arg()
        print('inside sample funcrion')
    return inner
@outer
def sample():
    print('outside sample function')
sample()
outside sample function
inside sample funcrion
```

```
def outer(arg):
    def inner(a,b):
        arg(a,b) if a>b else arg(b,a)
    return inner
@outer
def sample(x,y):
    print(x/y)
sample(10,2)
sample(2,10)
5.0
5.0
```

Singleton class: a class which can create only one object in its life time it is called singleton class.

```
def singleton(arg):  
    d={}  
    def inner():  
        if arg not in d:  
            d[arg]=arg()  
        return d[arg]  
    return inner  
@singleton  
class sample():  
    def __init__(self):  
        print('constructor called')  
        print('object create')  
obj=sample()  
obj=sample()  
constructor called  
object create
```

```
def singleton(arg):  
    d={}  
    def inner():  
        if arg not in d:  
            d[arg]=arg()  
        return d[arg]  
    return inner  
@singleton  
class dhamaka():  
    def __init__(self):  
        self.tickets=250  
    def booking(self,n):  
        if self.tickets>=n:  
            self.tickets-=n  
            print('tickets booked')  
            print(f' {self.tickets} tickets are available')  
        else:  
            print('tickets completed')  
    def bookmyshow():  
        obj=dhamaka()  
        obj.booking(int(input('enter tickets')))  
bookmyshow()  
bookmyshow()  
enter tickets 33  
tickets booked  
remaining 217 tickets are available  
enter tickets 3  
tickets booked  
remaining 214 tickets are available
```


Exception handling

there are 3 types of errors

1. Syntax error
2. Exception
3. Logical error

1.syntax error: this error occurs when interpreter encounters invalid syntax in the python code.

2.exception:

- These are the errors that occurs while executing the program.
- When exception occurs control flow will be terminated.

3.logical error: this errors occurs when program is syntactically correct but it will not give desire output.

Base exception errors:

1)exception

2)keyboardInterrupt

1)exception: it is a base class of all types of exceptions

2)keyboard interrupt: this exception is raised when we pressed cnt+c

Types of exception:

1. AttributeError
2. ArithamaticError
 - I. ZeroDivisionError
 - II. FloatingPointError
 - III. OverviewError
- 3.EoFError
- 4.NameError
- 5.LookupError
 - i)IndexError ii)KeyError

6)stopIterationError

7)OSError

i)FileExistsError

ii)PermissionError

8)ValueError

1)AttributeError: it is raised when we try to print the attribute that doesnot belongs to class or object then we will get attribute Error.

2)ArithamaticError:

i)ZeroDivisionError: it is raised when we perform True division or FloorDivision or modular division with 0

ii)FloatingPointError: it is raised when Floating point caluculation Fails.

iii)OverFloawError: it is raised when caluculation exceeds for numeric type.

3)EOFError: it is raised when input method or raw input method fails.

4)NameError: it is raised when variable is used out of its scope.

5)LookUpError:

i)indexError: it is raised when index is not found in the sequence

ii)keyError: it is raised when key is not found in dictionary.

6)StopIterationError: it is raised when next method does not point to any object in the iterator

7)OSError:

i)FileExiastsError: it is raised when user try to create the file that is already created.

ii)PermissionError: it is raised when we try to open the file that is protected with password.

8)valueError: it is raised when we try to find index of value that is not in sequence.

try:

- in this try block we can write the code to test
- when error is raises in try block then control goes to inside except block.
- For try block either except block or finally block need to be declared.

```
try:
    x=10
    print('inside block')
    print(x)
except:
    print('inside except block')
```

inside block
10

```
try:
    print('inside try block')
    print(y)
except:
    print('inside except block')
```

inside try block
inside except block

```
try:
    l=[1,3,4,0,5,7]
    for i in l:
        print(l[i])
except:
    print('inside except block')
```

3
0
5
1
7
inside except block

```
try:
    num=120
    l=[2,5,6,'a',0]
    for i in l:
        print(num//i)
except ZeroDivisionError:
    print('ZeroDivision error occured')
except IndexError:
    print('index error occured')
except:
    print('unknown error occured')
```

60
24
20
unknown error occured

```
try:
    l=[10,20,30,0,'a']
    x=60
    for i in range(len(l)):
        print(x//l[i])
except ZeroDivisionError as msg:
    print(msg)
except TypeError as msg:
    print(msg)
except:
    print('unexcepted error')
```

6
3
2
integer division or modulo by zero

```
try:
    num=120
    l=[2,5,6,0,'a']
    for i in l:
        print(num//i)
except ZeroDivisionError:
    print('ZeroDivision error occured')
except IndexError:
    print('index error occured')
except:
    print('unknown error occured')
```

60
24
20
ZeroDivision error occured

```
try:
    l=[10,20,30,0,'a']
    x=60
    print(l[100])
    for i in range(len(l)):
        print(x//l[i])
except ZeroDivisionError as msg:
    print(msg)
except TypeError as msg:
    print(msg)
except IndexError:
    print('index error occured')
```

index error occured

except:

- If any error raises in try block then except block will be executed.
- For one try block we can write multiple except blocks.

```
try:
    x='vinay'
    print(y)
except:
    print('inside except block')
inside except block
```

else: if there is no errors in try block then else block will be executed.

```
try:
    l=[10,20,30,]
    x=60
    for i in range(len(l)):
        print(x//l[i])
except (ZeroDivisionError,TypeError) as msg:
    print(msg)
except:
    print('unexcepted error')
else:
    print('no errors are there')
```

6

3

2

no errors are there

finally: it is always executed.

```
try:
    ad=0
    l=[1,2,3,'a']
    for i in l:
        ad+=i
except:
    print('error occurred')
else:
    print('no errors are there')
finally:
    print(ad)
6
error occurred
6
```

raise: raise is a keyword it is used to through an exception whenever user need of it.

```
try:
    n=int(input('enter the value:'))
    if n%2!=0:
        raise Exception ('not even')
except exception as msg:
    print(msg)
else:
    print('even')
```

enter the value: 8
even

```
class even(Exception):
    def __init__(self,msg):
        self.msg=msg
    def __str__(self):
        return self.msg
try:
    n=int(input('enter number:'))
    if n%2!=0:
        raise even ('not even')
except even as msg:
    print(msg)
else:
    print('even')
```

enter number: 9
not even

```
class prime(Exception):
    def __init__(self,msg):
        self.msg=msg
    def __str__(self):
        return self.msg
try:
    n=int(input('enter number:'))
    for a in range(2,n//2+1):
        if n%a==0:
            raise prime ('not prime')
except prime as msg:
    print(msg)
else:
    print('prime')
```

enter number: 5
prime

assert: assert is a keyword it is used to check the condition if the condition is false it will through assertion error.

```
try:
    n=4
    assert n%2==0,'it is not even'
except AssertionError as msg:
    print(msg)
else:
    print('even')
even
```

iterator: iterator is an object it is used to iterate over iterable data type.

```
l=[1,2,3,4,5]
print(next(l))
print(next(l))
print(next(l))
print(next(l))
OP: Type Error
```

```
l=[1,2,3,4,5]
ob=iter(l)
print(next(ob))
print(next(ob))
print(next(ob))
print(next(ob))
print(next(ob))
1
2
3
4
5
StopIteration Error
```

```
try:
    l=[1,2,3,4,5]
    ob=iter(l)
    print(next(ob))
    print(next(ob))
    print(next(ob))
    print(next(ob))
    print(next(ob))
    print(next(ob))
except:
    print('completed iteration')

1
2
3
4
5
completed iteration
```

```
class seq():
    def __init__(self,s,e):
        self.s=s
        self.e=e
    def __iter__(self):
        return self
    def __next__(self):
        n=self.s
        if self.s>=self.e:
            raise StopIteration
        self.s+=1
        return n
ob=seq(1,6)
x=iter(ob)
for i in x:
    print(i)
```

```
1
2
3
4
5
```

Generators:

- Generator is a normal function with yield keyword.
- Output of generator function is generator object.
- Generator object are used either by calling next method on generator object or for using for loop on generator object.

```
def sample():  
    yield 1  
    yield 2  
x=sample()  
print(next(x))  
print(next(x))
```

1
2

```
def sample():  
    yield 1  
    yield 2  
    yield 3  
x=sample()  
for a in x:  
    print(a)
```

1
2
3

```
def fact(n):  
    t=1  
    if n==0 or n==1:  
        return 1  
    for i in range(1,n+1):  
        t=t*i  
    yield t
```

```
x=fact(5)  
for i in x:  
    print(i)
```

1
2
6
24
120

```
def fib(n):  
    f=0  
    s=1  
    for i in range(n):  
        t=f+s  
        yield f  
        f,s=s,t
```

```
x=fib(6)  
for i in x:  
    print(i)
```

0
1
1
2
3
5

Difference between yield and return:

- return returns the control flow from functional space to main space.
- Yield doesnot return control flow from functional space to main space.
- Return returns values to the function call where as yield returns generator object to the functiona call.

File handling:

- file is a collection of data the data the data is like video file, audio file ,text file etc.
- we have two types of files.
1)text file 2)binary file
1)text file: it is used to store charecters.
2)binary file: binary files have byte code it is used to store audio files,,video files,image fules etc.

File handling: file handling is nothing but opening a file and doing some operations like **write,read,append** access modes and **closing a file**

Syntax:

Fileobject=open('filename.txt', 'access mode')

Write:

- by using write access mode we can write the content in a file
- the provided file is not there then it will create a new file with provided file name.
- if the provided file is already exists then old content is overridden with new content.

- In write access mode file pointer always starts at beginning.

```
fo=open('vinay file.txt' , 'w')  
fo.write('hai how are you')  
fo.close()
```

read:

- By using read accessmode we can read a content in a file.
- If the provided file is not there then it will through file not found error.
- If the provided file is already exists then it will print content in a file
- In read access mode file pointer always starts at beginning.

append:

- By using append access mode we can add the additional content to the file.
- If the provided file is not there then it will create a new file with provided file name
- If the file is already exists then it will add additional content to that file
- In append access mode file pointer is always starts at end position.

1) way to print how many characters are found in a file.

```
fo=open('vinay.txt', 'r')  
x=fo.read()  
print(len(x))
```

2) way to print how many words are found in a file

```
fo=open('vinay.txt', 'r')
x=fo.read()
n=x.split()
print(len(n))
```

3) way to print how many lines are present in a file.

```
fo=open('hello.txt', 'r')
fo.readlines()
fo.seek()
l=fo.readlines()
print(len(l))
```

seek(): it is used for taking the cursor to specified index position.

Tell(): tell method is used for specifying the current position of the cursor.

4) way to print all the words which are starting with 'ha'

```
fo=open('hello.txt', 'r')
x=fo.read()
y=x.split()
for i in y:
    if i.startswith('ha'):
        print(i)
```

5)wap to print how many words are present in a file with 5 charecters.

```
fo=open('hello.txt', 'r')
x=fo.read()
y=x.split()
c=0
for i in y:
    if len(i)==5:
        c+=1
print(c)
```

regular expression

- regular expression is sequence of charecters that form a search patterns.
- Regular expression can be used to check if a string contains the specifies search patterns.
- In oreder to achive regular expression we have to import re module

1)match:

- It returns a match object if there is a match at starting of the string.
- It returns none if there is no matches are found.

```
import re
s='vinay vicky'
x=re.match('v',s)
y=re.match('i',s)
z=re.match('\w{4}',s)
print(x) op: <re.Match object; span=(0, 1), match='v'>
print(y) op:None
print(z) op: <re.Match object; span=(0, 4), match='vina'>
```

2)search:

- it returns a match object if there is a match anywhere in the string.
- It returns none if no matches are found.

```
x='vinay is a python developer'
a=re.search('y',x)
print(a)
<re.Match object; span=(4, 5), match='y'>
```

3)findall:

- it returns a list containing all matches are and returns an empty list if no matches are found.

```
x='vinay is a python developer'
b=re.findall('y',x)
print(b)
['y', 'y']
```

```
a='vinay'
x=a.startswith('v')
y=a.endswith('x')
print(x)
print(y)
True
False
```

Split:

- returns a list where the string has been split at each match

- by specifying the maxsplit we can control splitting of parent string.

```
import re
x='hai how are you'
n=re.split('\s',x)
print(n)
['hai', 'how', 'are', 'you']
```

5)sub: replaces one or many matches with a string.

```
import re
x='how many members'
n=re.sub('\s','#',x)
print(n)
how#many#members
```

6.subn:

- by specifying count we can restrict the replacing character for not
- more than n times even though string is present.

```
import re
x='hai hello how are'
n=re.subn('\s','#',x)
print(n)
('hai#hello#how#are', 3)
```

\d: returns a match where the string contains(numbers from 0_9)

\D: returns a match where the string does not contain digits.

\s: returns a match where the string contain a white space character.

\S: returns a match where the string does not contain a white space character.

\W: returns a match where the string contains any word character(characters from a to z,digits from 0-9 and the underscore character).

\w: returns a match where the string does not contain any word characters

[abc]: to find any of a or b or c in source

[a-c]: to find any of a,b,c,in source

[^abc]: to find all charecters except abc

[0-9]: to find non digit character.

comprehensions

- comprehensions are used for creating new sequences from iterables or another sequences.
- Comprehensions are single line statements.
- Based on expression if we want to define sequences.
- In python there are 3 types of comprehensions are there.
 - 1)List comprehension
 - 2)Set comprehension
 - 3)dictionary comprehension

1)list comprehension:

One value and one for loop:

```
l=[i for i in range(1,10)]  
print(l)  
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Adding the squares of numbers into list by using list comprehension approach.

```
l=[i**2 for i in range(1,10)]  
print(l)  
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

One value one for loop and one if condition:

```
l=[i for i in range(1,10) if i%2==0]  
print(l)  
[2, 4, 6, 8]
```

One value one for loop and multiple if condition:

```
m=(11,28,34,99,23,67,90,70)
l=[i for i in m if i%2!=0 and i>50]
print(l)
[99, 67]
```

one value and multiple for loops:

```
l=[[i]*3 for i in range(1,11)]
print(l)
[[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4], [5, 5, 5], [6, 6, 6], [7, 7, 7], [8, 8, 8], [9, 9, 9], [10, 10, 10]]
```

```
m=(11,28,34,99,23,67,90,70)
l=[i for i in m if i%2==1]
print(l)
[11, 99, 23, 67]
```

```
l=[1 if i%2==0 else 0 for i in range(1,11)]
print(l)
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
```

2)dictionary comprehension:

```
n='abcd'
d={i.upper():i.lower()*3 for i in n}
print(d)
{'A': 'aaa', 'B': 'bbb', 'C': 'ccc', 'D': 'ddd'}
```

```
n='abcd'
d={i.upper():i.lower()*3 for i in n}
print(d)
{'A': 'aaa', 'B': 'bbb', 'C': 'ccc', 'D': 'ddd'}
```

```
s='pqrs'
d={i:s[i] for i in range(len(s))}
print(d)
{0: 'p', 1: 'q', 2: 'r', 3: 's'}
```

Enumerate function: enumerate function is used for returning enumerate object which consists of index positions and its respective values as pairs.

```
s='pqrs'
list(enumerate(s))
[(0, 'p'), (1, 'q'), (2, 'r'), (3, 's')]
```

```
s='pqrs'
d={k:v for k,v in enumerate(s)}
print(d)
{0: 'p', 1: 'q', 2: 'r', 3: 's'}
```

```
s='pqrs'
l=[10,20,30,40]
d={s[i]:l[i] for i in range(len(s))}
print(d)
{'p': 10, 'q': 20, 'r': 30, 's': 40}
```

Zip function:

```
s='pqrs'
l=[100,200,300,400]
list(zip(s,l))

[('p', 100), ('q', 200), ('r', 300), ('s', 400)]
```

```
s='pqrs'
l=[100,200,300,400]
d={k:v for k,v in zip(s,l)}
print(d)
{'p': 100, 'q': 200, 'r': 300, 's': 400}
```

```
s='pqrs'
l='abcdefg'
list(zip(s,l))

[('p', 'a'), ('q', 'b'), ('r', 'c'), ('s', 'd')]
```