Program :

collection of instructions that can be executed by the computer in order to solve a specific problem is called as program.

Programming :

It is the process of how we are creating/writing the program to instruct the computer to solve specific problem

Variable :

Variable is a name that is used to refer a memory location.

(or)

Variable is a name which can be used to hold a value during program execution.

Rules For Defining Name or Identifier :

1.name Can start with either alphabets(A-Z or a-z) or underscore (_)
2.Do not use Keyword qs a name
3.name should not start with digits or special characters.
4. Name can contain alphabets ,digits and underscore(_).
5.Names in Python are case-sensitive.
A and a are different names.

Valid Names (Identifier) :

---

a
name
_name
_
A
Age
a2
abc2
a123
name_2
a_b

Invalid Names (Identifier)

23
$name
name$
2a
&

Keywords :

False,None,True,and,as,assert,async,await,break,class,continue,def,del,elif,else,except,finally,for,from,global,if,import,in,is,lambda,nonlocal,not,or,pass,raise,return,try,while,with,yield

Examples for creating variables

```
a=2
name="Anand"
_age=20
roll_1=111
roll_2=123
roll_3=345
```

```
a , name =2 , "Anand"
a , b, c =2 , 3, 4
age , name=20 , "ram"

a=b=c=2
```

Examples :

```
age=20
Age=23
AGE=30
print(age,Age,AGE)
```

Output :

20 23 30

```
age=2
age=3
age=4
print(age)
```

Output :

4

```
name="Anand"
myname=name
print(name,myname)
```

output :

Anand Anand

Data type :

Data type in programming is a classification of data that specifies the type of data and which type of operations that can be applied to it without raising any error.

Data Types in python :

1. int (Integer)
2. float (Floating-point Number)
3. string
4. list
5. tuple
6. dict (Dictionary)
7. set
8. frozenset
9. bool (Boolean)

Others :

10. complex
11. bytes
12. bytearray

1.Integer (int)

It represents the numerical values in programming.
It can be positive,negative, or zero(0)
It can be any length.

Examples :
1
1000
20000000
-100000000
0

Program to find type of data :

```
a=2
print(type(a))
```

Output :

```
<class 'int'>
```

Program to check a given data is Integer or not

```
a=2
print(isinstance(a,int))

name="Anand"
print(isinstance(name,int))
```

output :
True
False

2.Floating-point Number (float)

-> It represents decimal point numbers in programming.
-> it can be positive or negative.
-> it must contains a decimal point.
-> it should contain only 0 to 9 digits and one decimal point (.)
-> It is accurate up to 15 decimal point.

Examples :

1.2
1.333
123.444
-1.34
0.0

Program to find the type of the data:

```
test=3.14
result=type(test)
print(result)
```

output:

```
<class 'float'>
```

Program to check a given data is Float or not

```
test=1.23
a=isinstance(test,float)
print(a)

te=2
print(isinstance(te,float))
```

output :

True

False

Specual Case Representation of float:

1.2e4 --->1.2 * 10**4

1.3E4 --->1,2 * 10**4

1.2-e4 -->1.2 * 10 **(-4)

1.5-E4 -->1.5 * 10 ** (-4)

Maximum value of float : 1.7* 10**308

if you give value more than that maximum value.it will print as infinite value.

```
print(1.79e308)
```

output :

1.79e308

```
print(1.8e308)
```

inf

Minimum value of float is 5.0 *10 **(-324)
if we give value less than that minimum value it will print as 0

```
print(5e-324)
```

output :

5e-324

```
print(5e-325)
```

output :

0.0

Strings :

String is a sequence (or) collection (or) array of character data.
Character is anything that you can type on keyboard with one keystroke, like a letter,number etc.
Every string must be surrounded by either single quotes or double quotes.
String is an immutable data type.

Examples :

```
"Anand"
"1233qeee"
"college"
"$wyw"
'abcd123'
```

Find type of the data:

```
a="anand"
print(type(a))
```

output :

```
<class 'str'>
```

Creating Strings:

```
a=""
print(a)
a="hello"
print(a)
a='hello'
print(a)
```

output:

```
hello
hello
```

Check the type of the data is string or not :

```
a="anand"
print(isinstance(a,str))
```

output :

```
True
```

```
a=1233
print(isinstance(a,str))
```

ouput:

```
False.
```

Type conversion from Int to str:

```
a=2
print(a,type(a))
a=str(a)
print(a,type(a))
```

ouput:

```
2 <class 'int'>
2 <class 'str'>
```

Type conversion from float to str:

```
a=10.4
print(a,type(a))
a=str(a)
print(a,type(a))
```

ouput:

```
10.4 <class 'float'>
10.4 <class 'str'>
```

String Indexing

In programming, individual elements in the sequence of data items can be accessed using the numerical index or key value is called the indexing.

We Know that, string is an ordered sequence of charcter data.So, Inorder to access the particular charcter over the sequnce, we have to use string indexing.

String indexing in Python is zero-based: the first character in the string has index 0, the next has index 1, and so on. The index of the last character will be the length of the string minus one.

a="COLLEGE"

" C	O	L	L	E	G "
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

```
print(a[0])
```

Ouput:

```
C
```

```
print(a[5])
```

output:

```
 G
```

```
print(a[10])
```

output:

```
IndexError: string index out of range
```

```
print(a[-1])
```

output:

```
G
```

```
print(a[-6])
```

output:

```
C
```

```
print(a[-10])
```

output:

```
IndexError: string index out of range
```

String Slicing :

Inorder to access a part of a string, we have to use string slicing.

Syntax :

```
variable_name [StartIndex : EndIndex : Step]
```

Default values:

StartIndex = 0
EndIndex =len(variable_name) - 1
Step =1

Examples:

```
a="mystring"
print(a[1:4])
print(a[:4])
print(a[4:])
print(a[1:5:1])
print(a[1:5:2])
print(a[1:5:3])
print(a[1:10000])
print(a[100000:200000])
```

Output:

```
yst
myst
ring
ystr
yt
yr
ystring
"" -->empty string
```

Negative Slicing :

```
a="mystring"
print(a[-3:-1])
print(a[-4:-1])
print(a[:-4])
print(a[-2:])
print(a[-6:-1:2])
print(a[::])
print(a[-1000000:])
```

output :

```
in
rin
myst
ng
srn
mystring
mystring
```

String Methods

1.upper()

It converts all the lower case letters of the string into uppercase.

Eg:-

```
a="Student"
a=a.upper()
print(a)
```

output:-

```
STUDENT
```

2. lower()

Converts all the uppercase letters of the string into lowercase letters.

Eg:-

```
a="STUdent"
a=a.lower()
print(a)
```

output:

```
student
```

3.swapcase()

It converts all the lower case letters of the string into uppercase and uppercase letters into the lowercase letters.

Eg :-

```
a="STUDent"
a=a.swapcase()
print(a)
```

output ;-

---

studENT

4.title()

Converts first charcter of each word which are seperated by spaces into uppercase.

Eg:-

```
a="hello world"
a=a.title()
print(a)
```

output:

Hello World.

5.capitalize()

Converts the first character of given string into the uppercase.

Eg;-

```
a="hello world"
a=a.capitalize()
print(a)
```

output:

Hello world.

6.isupper():

It checks all the characters of the sting are upper case are not.
it returns True if all are uppercase
it returns False if not all are uppercase.

Eg:

```
a="student"
result=a.isupper()
print(result)
```

output :-

False

```
a="STUDENT"
result=a.isupper()
print(result)
```

output :-

True

7.islower()

It checks all the characters of the sting are lower case are not.
it returns True if all are lowercase
it returns False if not all are lowercase.

Eg:

```
a="student"
result=a.islower()
print(result)
```

output :-

True

```
a="Student"
result=a.islower()
print(result)
```

output :-

False

8.isupper()

It checks all the characters of the sting are upper case are not.
it returns True if all are uppercase
it returns False if not all are uppercase.

Eg:

```
a="ALL"
result=a.isupper()
print(result)
```

output :-

True

```
a="Student"
result=a.isupper()
print(result)
```

output :-

False

9.isalpha()

It checks all the charcters in a string are alphabets only.
it returns True if all charcters are alphabets otherwise it returns False.

Eg:-

```
a="abced"
result=a.isalpha()
print(result)
```

output:-

True

```
a="Hello123"
result=a.isalpha()
print(result)
```

output:-

False

10.isdigit() or isnumeric() or isdecimal()

It checks all the charcters in a string are numbers only.
it returns True if all charcters are numbers otherwise it returns False.

Eg:-

```
a="abced"
result=a.isdigit()
print(result)
res=a.isnumeric()
print(res)
result=a.isdecimal()
print(result)
```

output:-

```
False
False
False
```

```
a="123"
result=a.isdigit()
print(result)
res=a.isnumeric()
print(res)
result=a.isdecimal()
print(result)
```

output:-

```
True
True
True
```

11.isalnum()

It checks all the charcters in a string are numbers and alphabets only.
it returns True if all charcters are numbers and alphabets otherwise it returns False.

Eg:-

```
a="abced"
result=a.isalnum()
print(result)
```

output:-

```
True
```

```
a="Hello123$$"
result=a.isalnum()
print(result)
```

output:-

```
False
```

12. strip() and lstrip() and rstrip()

strip() ->It removes the spaces at beginning and end of the string if any.
lstrip() ->It removes the spaces at beginning or left side of string only.
rstrip() ->It removes the spaces at end or right side of the string only.

Eg :

```
a="    car    "
a=a.strip()
print(a)
```

output :

```
"car"
```

```
a="    car    "
a=a.lstrip()
print(a)
```

output :

```
"car    "
```

```
a="    car    "
a=a.rstrip()
print(a)
```

output:

```
"    car"
```

Special Case :

By default these methods will remove the spaces, we can also remove the characters by mentioning them between parenthesis.

Eg:

```
a="abc1234cba"
a=a.strip("ab")
print(a)
```

output:

```
c1234c
```

```
a="abc123cba"
a=a.lstrip("ab")
print(a)
```

output:

```
c123cba
```

```
a="abc123cba"
a=a.rstrip("ab")
print(a)
```

output:

```
abc123c
```

13.count()

It returns number that how many times a spcified string appeared in the original string.

Eg:

```
a="anand"
out=a.count("an")
print(out)
```

output:

```
2
```

```
temp="abababab"
c=temp.count('a')
print(c)
```

output:

```
4
```

Special Case :

we can also mention the range of string, on which we need to count.

eg :

```
a="abababab"
c=a.count('a',3,6)
print(c)
```

output:-

```
1
```

14.join()

It is used to join the collection of strings by the given value to form new string.
it is opposite to the split.

eg:

```
a=['1','2','3']
result="ab".join(a)
print(result)
```

output:

```
1ab2ab3
```

```
a=['1','2','3','4']
result="".join(a)
print(result)
```

output:

```
1234
```

15.replace()

It can be used to replace the characters of the given string by given required value.

Eg :

```
a="abababab"
a=a.replace('a','c')
print(a)
```

output :

```
cbcbcbcb
```

```
a="one one two"
a=a.replace('one','two')
print(a)
```

output:

```
two two two
```

Special Case:

We can also mention how many times we need to replace the value.

Eg:

```
a="one one two"
a=a.replace("one","two",1)
print(a)
```

output:

```
two one two
```

16.find() and rfind() and index()

These methods are used to search an string in the given original string.
find()->it returns index if it found the string otherwise it returns -1.it starts searching from left to right.
rfind() ->it returns index if it found the string otherwise it returns -1.it starts searching from right to left
index()->it returns index if it found the string otherwise it will throw error.it starts searchinf from left to right.

Eg :-

```
a="abcdefab"
res=a.find('a')
print(res)
```

output:

0

```
a="abcdfeab"
res=a.rfind('a')
print(res)
```

output:

6

```
a="abcdef"
res=a.index('a')
print(res)
```

output:

0

```
a="myprog"
r=a.find('an')
print(r)
```

output:

0

```
a="myprogr"
r=a.rfind('an')
print(r)
```

output:

6

```
a="myprog"
r=a.index('an')
print(r)
```

output:

0

String Formatting

It is the process of inserting the custom values or variables into the string.

1.By Using format()

Eg:

```
name="Anand"
print("my name is {}",format(name))
```

output:

```
my name is Anand
```

Eg:

```
a=int(input())
b=int(input())
sum=a+b
print("sum of {} and {} is {}".format(a,b,sum))
```

input:

```
2
3
```

output:

```
sum of 2 and 3 is 5
```

Eg:

```
a=int(input())
b=int(input())
sum=a+b
print("sum of {2} and {1} is {0}".format(sum,b,a))
```

input:

```
2
3
```

output:

```
sum of 2 and 3 is 5
```

Eg:

```
a=int(input())
b=int(input())
sum=a+b
print("sum of {n1} and {n2} is {s}".format(s=sum,n1=b,n2=a))
```

input:

```
2
3
```

output:

```
sum of 2 and 3 is 5
```

2. Using f-strings;

Eg:

```
a=3
b=5
s=a+b
print(f"sum of {a} and {b} is {s}")
```

output:

```
sum of 3 and 5 is 8
```

Mutability

Mutability is a property that states that, the internal state (value) of the object can be modified after its creation.
simply we can make changes to object.
eg:- list, set, dictionary

->changing value is cheap.

Immutability

Immutability is a property that states that, the internal state of the object can't be modified after its creation.
simply we can't make changes to object.
if we want make change, we have to create new value.
Eg : int , float, string, tuple, frozenset
->changing values is expensive.

List :-

List is a built-in data type which is used to store and represent the collection of data items of any type.

Properties:

-> It is mutable (or) changeable
  -> Add ✅
  -> Remove ✅
  -> update ✅
->It is ordered.
  ->It remembers position of each data item.
  ->Indexing ✅
  ->Slicing ✅
->It allows duplicate values.
->It can store any data of any type.

Tuple :-

Tuple is a built-in data type which is used to store and represent the collection of data items of any type.

Properties:

-> It is immutable (or) unchangeable
  -> Add ❌
  -> Remove ❌
  -> update ❌
->It is ordered.
  ->It remembers position of each data item.
  ->Indexing ✅
  ->Slicing ✅
->It allows duplicate values.
->It can store any data of any type.

Set :-

Set is a built-in data type which is used to store and represent the collection of data items of immutable type.

Properties:

-> It is mutable (or) changeable
  -> Add ✅
  -> Remove ✅
  -> update ❌
->It is unordered.
  ->It doesn't remembers position of each data item
  ->Indexing ❌
  ->Slicing ❌
->It doesn't allows duplicate values.
->It can store only immutable data.

frozenSet :-

---

frozenset is a built-in data type which is used to store and represent the collection of data items of immutable type.

Properties:

-> It is immutable (or) unchangeable
  -> Add ❌
  -> Remove ❌
  -> update ❌
->It is unordered.
  ->It doesn't remembers position of each data item
  ->Indexing ❌
  ->Slicing ❌
->It doesn't allows duplicate values.
->It can store only immutable data.

Operations on list :

1. Adding elements ✅
2. accessing elements ✅
3. updating elements ✅
4. deleting elements. ✅

Operations on Tuple :

1. Adding elements ❌
2. accessing elements ✅
3. updating elements ❌
4. deleting elements. ❌

Operations on set :

---

1. Adding elements ✅
2. accessing elements ❌
3. updating elements ❌
4. deleting elements. ✅

Operations on frozenset :

---

1. Adding elements ❌
2. accessing elements ❌
3. updating elements ❌
4. deleting elements. ❌

creating list

creating empty list

```
a=list()
print(a)
```

output:

```
[]
```

creating list with elements

```
a=[1,2,"abc"]
print(a)
```

output:

```
[1,2,"abc"]
```

creating Tuple

creating empty Tuple

```
a=tuple()
print(a)
```

output:

```
()
```

creating Tuple with elements

```
a=(1,2,"abc")
print(a)
```

output:

```
(1,2,"abc")
```

creating set

creating empty set

```
a=set()
print(a)
```

output:

```
set()
```

creating set with elements

```
a={1,2,3,4}
print(a)
```

output:

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

creating frozenset

creating empty frozenset

```
a=frozenset()
print(a)
```

output:

```
frozenset()
```

creating fozenset with elements

```
a=frozenset([1,2,3])
print(a)
```

output:

```
frozenset({1,2,3,4})
```

List

A list is a built-in data type (or) data structure in python, which represents the collection of data items and it has features such as,

1.Mutable (or) changeable
2.Ordered
3.Allows duplicates.
4.It can store any type of data.
5.It can be any length

Eg:-

```
[1,3,4]
[1,"anand"]
[[1,2,3],"abc",3]
```

Operations On list:

1.creating list
2.Adding elements of list
3.accessing elements to list
4.updating elements of list
5.deleting elements of list

1.creating list

creating empty list

```
a=[]
print(a)
```

output:

```
[]
```

creating list with elements

```
a=[1,2,"abc"]
 print(a)
```

output:

```
[1,2,"abc"]
```

2. Adding elements to list

append()

It is a bulit-in method which is used to add a element at the end of list.

return type : None

Eg :-

```
a=[]
a.append(2)
a.append(3)
print(a)
```

output:

```
[2,3]
```

```
temp=[1,2,3]
temp.append("abc")
temp.append(1.6)
print(temp)
```

output:

```
[1,2,3,"abc",1.6]
```

Error Case:

We can add only one element at a time using append method.

```
a=[1,2]
a.append(5,6)
print(a)
```

output:

```
TypeError: list.append() takes exactly one argument
```

Insert()

It is a built-in method which is used to add element to list at required position.

return type : None

Syntax :

```
ListVariable.insert(position,value)
```

Eg :-

```
a=[1,3,4]
a.insert(0,10)
print(a)
```

output:

```
[10,1,3,4]
```

```
a=[1,2,3]
a.insert(-2,15)
print(a)
```

output:

```
[1,15,2,3]
```

extend()

It is a built-in method used for adding collection of elements to a list at a time.
return type : None

Eg:

```
a=[1,2]
b=[1,2,3]
a.extend(b)
print(a)
```

output:

```
[1,2,1,2,3]
```

```
b=[1]
e=(2,3,4)
b.extend(e)
print(b)
```

output:

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

```
s=[]
k={1,2,3}
s.extend(k)
print(s)
```

output:

```
[1,2,3]
```

Error case:

we can add only collection data types(iterable) in extend method and we can extend only one collection at a time.

Eg:-

```
a=[1,3]
a.extend(1)
print(a)
```

output:

```
Error
```

```
a=[1,2]
a.extend(1,2,3)
print(a)
```

output:

```
Error
```

```
a=[1,2]
a.extend("anand")
print(a)
```

output:

```
[1,3,'a','n','a','n','d']
```

"+"

It is used to concatenate two lists

eg:

```
a=[1,2,3]
b=[2,3,4]
c=a+b
print(c)
```

output:

[1,2,3,2,3,4]

```
a=[1,2,3]
b=(2,3,4)
c=a+b
print(c)
```

output:

---

error.

3.Accessing elements of list

---

list indexing

---

a=[1,2,3,"abc"]
a[0] ---->1
a[3] ---->"abc"
a[-1] ---->"abc"
a[10] ---->IndexError
a[-22] ---->IndexError

list slicing

---

a=[10,20,30,40]
a[1:3] -->[20,30]
a[0:3] -->[10,20,30]
a[:2] --->[10,20]
a[3:] -->[40]
a[10:20] -->[]
a[0:3:2] -->[10,30]

4.Updating elements of a list

---

we can update single element by using list indexing
we can update multiple elements by using list slicing

Updating single elements:

Eg:

```
a=[1,2,3]
a[2]=5
print(a)
```

output:

[1,2,5]

```
a=["a","b"]
a[0]=[1,2,3]
print(a)
```

output:

[[1,2,3],"b"]

```
a=[1,2,3]
a[2]="and"
print(a)
```

output:

[1,2,"and"]

updating multiple elements:

Here step size should be always equal to one.

Eg:

```
a=[1,2,3,4]
a[1:3]=[5,6]
print(a)
```

output:

[1,5,6,4]

```
a=[1,3,4,5,5]
a[1:4]=[1,2,3,4,5,6]
print(a)
```

output:

[1,1,2,3,4,5,6,5]

5.Deleting elements of list

remove()
pop()
clear()
del()

remove()

It is built-in method which is used remove an element from given list.
if the element is exist in the list it will remove it, otherwise it will throw error.
It can remove only one element at a time.
return type : None

Eg:

```
a=[1,3,4]
a.remove(3)
print(a)
```

output:

[1,4]

```
a=[1,2,3,4,4]
a.remove(4)
print(a)
```

output:

[1,2,3,4]

```
a=[1,3]
a.remove(5)
print(a)
```

output:

ValueError

pop()

It is built-method which is used to delete the element based on given index from list.

return type : value

Eg :

```
a=[1,2,3,4]
a.pop()
print(a)
```

output:

```
[1,2,3]
```

```
a=[1,2,3]
a.pop(0)
print(a)
```

output:

```
[2,3]
```

```
a=[1,2,3,5,6]
b=a.pop(4)
print(b)
print(a)
```

output:

```
6
[1,2,3,5]
```

del

It is a keyword which is used to delete a value during execution of program.

Eg:

```
a=[1,3,4]
del a[2]
print(a)
```

output:

```
[1,3]
```

```
a=[1,2]
del a[0]
print(a)
```

output:

```
[2]
```

```
a=[1,2,3]
del a[10]
print(a)
```

output:

```
IndexError
```

Clear() :-

It is a built-in function which is used to remove all elements from the list.

return type : None

Eg:-

```
a=[1,3,4,5]
a.clear()
print(a)
```

output:

```
[]
```

```
a=["ana","bcd"]
a.clear()
print(a)
```

output:

```
[]
```

Other Methods in List:

Count()
index()
reverse()
copy()
deepcopy()
sort()

count()

It returns a number , that represents how many times the given value appears in the list.

return type : Int

Eg:

```
a=[1,3,4,5,3]
c=a.count(3)
print(c)
```

output:

2

```
a=["ab",3,4,"ab","ab"]
c=a.count("ab")
print(c)
```

output:

3

index()

It is used to find the position of given value in the list.

return type : Int(index)

Eg:

```
a=[1,2,3,4]
i=a.index(2)
print(i)
```

output:

```
1
```

```
a=[1,"a","c",2]
ind=a.index(4)
print(ind)
```

output:

```
ValueError
```

Special Case:

index(value,start,end)

eg:

```
a=[1,3,4,4,5,4,4]
ind=a.index(4,4,6)
print(ind)
```

output:

```
5
```

```
a=[1,2,3,3,3,3,3,5]
ind=a.index(3,-4,-1)
print(ind)
```

output:

```
4
```

reverse()

It is used to reverse the given list.

return type : None

eg:

```
a=[1,2,3,4]
a.reverse()
print(a)
```

output:

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

```
a=["a","b","c"]
a.reverse()
print(a)
```

output:

```
["c","b","a"]
```

Sort()

It is used to arrange the elements either increasing order or decreasing order.

Note: Every element in the list must be of same type or comparable.

Ascending or increasing :

```
a=[3,2,10]
a.sort()
print(a)
```

output:

```
[2,3,10]
```

```
a=["x","b","d"]
a.sort()
print(a)
```

output:

```
["b","d","x"]
```

```
a=[[1],[4],[0]]
a.sort()
print(a)
```

output:

```
[[0],[1],[4]]
```

descending or decreasing :

```
a=[3,2,10]
a.sort(reverse=True)
print(a)
```

output:

```
[10,3,2]
```

```
a=["x","b","d"]
a.sort(reverse=True)
print(a)
```

output:

```
["x","d","b"]
```

copy() and deepcopy()

Basically there are two types:

1.shallow copy
2.deep copy

1.shallow copy:

In this method, It creates a new list and it copies the references of all the values of this original list into new list.

return type: list

Eg:

```
temp=[1,2,3,4,5,6]
c=temp.copy()
print(temp,c)
```

output:

[1,2,3,4,5,6] [1,2,3,4,5,6]

Problem with copy:
_____

Eg:
___

```
a=[1,2,[1,2,3]]
b=a.copy()
print(a,b)
b[2].append(5)
print(a,b)
```

output:
_____

[1,2,[1,2,3]] [1,2,[1,2,3]]
[1,2,[1,2,3,5]] [1,2,[1,2,3,5]]

In the above example we can observe that, the change in list b reflects the list a.
To avoid that problem , we have to use deepcopy

deepcopy()
_____

In this method, It creates new list and also recursively creates each element in the original list, then it adds each element to the new list

eg:
___

```
import copy
a=[1,3,[1,2,3]]
b=copy.deepcopy(a)
print(a,b)
b[2].append(5)
print(a,b)
```

output:
_____

[1,3,[1,2,3]] [1,3,[1,2,3]]
[1,3,[1,2,3]] [1,3,[1,2,3,5]]

Special Methods :

len() ->returns the length of list
max() ->returns the maximum element of list
min() ->returns the minimum element of list
sorted() ->returns the sorted list

Eg:

```
a=[1,2,3]
length=len(a)
print(length)
```

Output:

3

```
a=[2,3,4]
s=max(a)
print(s)
```

output:

4

```
a=[1,2,4]
s=min(a)
print(s)
```

output:

1

Eg:

```
a=[2,1,3]
temp=sorted(a)
temp_r=sorted(a,reverse=True)
print(temp,temp_r)
```

output:

[1,2,3] [3,2,1]

Tuple Methods

count()
index()

creating Tuple

creating empty Tuple

```
a=tuple()
print(a)
```

output:

```
()
```

creating Tuple with elements

```
a=(1,2,"abc")
print(a)
```

output:

```
(1,2,"abc")
```

count()

It returns a number , that represents how many times the given value appears in the tuple.

return type : Int

Eg:

```
a=(1,3,4,5,3)
c=a.count(3)
print(c)
```

output:

```
2
```

```
a=("ab",3,4,"ab","ab")
c=a.count("ab")
print(c)
```

output:

```
3
```

index()

It is used to find the position of given value in the tuple.

return type : Int(index)

Eg:

```
a=(1,2,3,4)
i=a.index(2)
print(i)
```

output:

1

```
a=(1,"a","c",2)
ind=a.index(4)
print(ind)
```

output:

ValueError

Special Case:

index(value,start,end)

eg:

```
a=(1,3,4,4,5,4,4)
ind=a.index(4,4,6)
print(ind)
```

output:

5

```
a=(1,2,3,3,3,3,3,5)
ind=a.index(3,-4,-1)
print(ind)
```

output:

4

Special Methods :
---

len() ->returns the length of tuple
max() ->returns the maximum element of tuple
min() ->returns the minimum element of tuple
sorted() ->returns the sorted list

Eg:

```
a=(1,2,3)          a=(2,3,4)      a=(1,2,4)
length=len(a)      s=max(a)       s=min(a)
print(length)      print(s)       print(s)
```

```
Output:            output:        output:
---------          ----------     ---------
3                    4              1
```

Eg:

```
a=(2,1,3)
temp=sorted(a)
temp_r=sorted(a,reverse=True)
print(temp,temp_r)
```

output:

[1,2,3] [3,2,1]

Set Methods

Adding elements to set

---

add()
update()

add()

It is used to add one element at a time to the set.

eg:-

```
a=set()              b={1,2,3}
a.add(2)              b.add(3)
a.add(3)              b.add(5)

output:-             output:
---------            ---------
{2,3}                {1,23,5}
```

update()

It used to add multiple elements to the set at a time.
Simply, it is used to add to sets.

Eg :-

```
a={1,2,3}
b={5,6}
a.update(b)
```

output:

{1,2,3,5,6}

Removing elements from set :
---

remove()
discard()
pop()
clear()

remove()

It is used to remove a particular element from the set.
if element not found in the set, then it will raise an error.

return type : None

Eg:

```
a={1,2,3}
a.remove(2)
print(a)
```

output:

{1,3}

```
b={1,2,3}
b.remove(10)
print(b)
```

output:
---
keyerror

discard()

It is also used to remove particular element from the set.

But, unlike remove, it won't raise any error if element not existed in the set.

return type : None

Eg:

```
a={1,2,3}
a.discard(2)
print(a)
```

output:

```
{1,3}
```

```
b={1,2,3}
b.discard(10)
print(b)
```

output:

```
{1,2,3}
```

pop()

In general, pop method removes the last element from the collection.
since, set is unordered, it will remove random element from the
set.

return type : popped element

Eg: -

```
a={1,2,3}
print(a.pop())
print(a)
```

output:

```
1
{1,2,3}
```

```
a={}
print(a.pop())
print(a)
```

output:

```
KeyError: 'pop
from an empty set
```

clear()

It is used to remove all the elements from the set at a time.

return type : None

Eg:

```
a={1,2,3}
a.clear()
print(a)
```

output:

---

set()

Special Operators & methods of set

---

union() (or) |
intersection() (or) &
difference() (or) -
symmetric_difference() (or) ^
isdisjoint()
issubset() (or) <=
issuperset() (or) >=

union() or |

---

method : union()
operator : |

It is used to combine the two or more sets to form a new set.

return type : New combined Set

eg:

```
a={1,2,3}
b={2,4,5}
temp=a.union(b)
print(temp)
```

output:

{1,2,3,4,5}

```
a={1,2,3}
b={4,5,3}
c={7,8,2}
d=a.union(b,c)
print(d)
```

output:

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

using operator :

---

eg:

```
a={1,2,3}
b={4,5}
print(a | b)
```

output :

{1,2,3,4,5}

eg:

```
a={1,2,3}
b={2,3}
c={5,6,7}
print(a | b | c)
```

output:

{1,2,3,5,6,7}

intersection()

method : intersection()
operator : &

It is used to get the common elements from two or more sets.

return type : set

Eg :

```
a={1,2,3}
b={3,15,6}
temp=a.intersection(b)
print(temp)
```

output:

{3}

```
a={1,2,3}
b={4,6,3}
c={1,2,3}
s=a.intersection(b,c)
print(s)
```

output:

{3}

using operator (&)

Eg:

```
a={1,2}
b={2,3}
print(a&b)
```

output:

{2}

```
a={1,2,3}
b={1,2}
c={6,7}
print(a&b&c)
```

output:

set()

isdisjoint()

It is used to check whether two sets contains common elements (or) not.

if it contains common elements it returns False

if both are different, it return True

return type : Boolean

Eg:

```
a={1,2,3}
b={4,5}
print(a.disjoint(b))
```

output:

```
True
```

```
a={1,2,3}
b={3,4}
print(a.disjoint(b))
```

output:

```
False
```

difference()

---

method : difference()
operator : -

It is used find the difference between two or more sets.

it return set of elements which are present in set1 but not in set2

return type : set

Eg :

```
a={1,2,3}
b={2,3,5}
print(a.difference(b))
```

output:

{1}

```
a={1,2,3}
b={2,9,8}
c={3,4,5}
print(a.difference(b,c))
```

output:

{1}

using operator(-)

Eg:

```
a={1,2,3}
b={3}
print(a-b)
```

output:

{1,2}

```
a={1,2,4}
b={1}
c={6,7}
```

output:

{2,4}

Symmetric_difference()

method : symmetric_difference()
operator : ^

It is used to get the elements which are not common the two or more sets.

simply, it is opposite to the intersection.

Eg:

```
a={1,2,3}
b={3,4,5}
print(a.symmetric_difference(b))
```

output :

{1,2,4,5}

```
a={1,2,3,4}
b={2,3,6,7}
c={7,8,9}

print(a.symmetric_difference(b,c))
```

output:

{1,4,6,8,9}

Using Operator(^)

```
a={1,2,3}
b={3,4,5}
print(a^b)
```

output:

{1,2,4,5}

```
a={1,2}
b={4,5}
c={5,6}
print(a^b^c)
```

output:

{1,2,4,6}

issuperset()

---

method : issuperset()

operator : >=

It used to check one set is supetset to another set.

That means, a set_1 is superset when all elements of set_2 are contained in set_1 .

return type : Boolean

Eg:

```
a={1,2,3,4,5}
b={1,2}
print(a.issuperset(b))
```

output:

True

```
a={1,2,3,4,5}
b={1,2,4,5,6}
print(a.issuperset(b))
print(a.issuperset(a))
```

output:

False

using operator (>=)

Eg:

```
a={1,2,3,4,5}
b={1,2,3}
print(a>=b)
```

output:

True

```
a={1,2,3,4,5}
b={4,5,6}
print(a>=b)
print(a>=a)
```

output:

False

True

special case :

operator (>)

It return False, when both sets are equal.
It is used to check proper subset

A proper subset is the same as a subset, except that the sets can't be identical. A set x1 is considered a proper subset of another set x2 if every element of x1 is in x2, and x1 and x2 are not equal.

eg:

```
a={1,2,3,4,5}
b={1,2,3}
print(a>b)
print(a>a)
```

output:

```
True
False
```

issubset()

method : issubset()
operator : <=

It used to check one set is subset to another set.

That means, a set_1 is subset when all elements of set_1 is contained in set_2 .

return type : Boolean

Eg:

```
a={1,2,3}
b={3,4,5,1,2}
print(a.issubset(b))
```

output:

```
True
```

```
a={1,2,3}
b={1,2,4,5}
print(a.issubset(b))
print(a.issubset(a))
```

output:

```
False
```

using operator (<=)

Eg:

```
a={1,2,3}
b={5,6,1,2,3}
print(a<=b)
```

output:

```
True
```

```
a={1,2,3}
b={4,5,6}
print(a<=b)
print(a<=a)
```

output:

```
False
True
```

special case :

operator (<)

It return False, when both sets are equal.
It is used to check proper subset

A proper subset is the same as a subset, except that the sets can't be identical. A set x1 is considered a proper subset of another set x2 if every element of x1 is in x2, and x1 and x2 are not equal.

eg:

```
a={1,2,3}
b={1,2,3,4,5}
print(a<b)
print(a<a)
```

output:

```
True
False
```

Frozenset methods:

Since it is completly immutable, it supports the following methods only.

1.len
2.max
3.min

special methods:

---

intersection()
union()
difference()
symmetric_difference()
issubset()
issuperset()

Dictionary :

Dictionary is a built-in data type which is used to store and represent the collection of key-value pairs.

simply we are assigning the label to exah

Properties:

-> It is mutable (or) changeable
  -> Add ✔
  -> Remove ✔
  -> update ✔
->It is unordered.
  ->It doesn't remembers position of each data item
  ->Indexing ✘
  ->Slicing ✘
->It doesn't allows duplicate values.
->key should be always immutable
->value can be mutable or immutable

Operations on dictionary :

1. Adding elements
2. accessing elements
3. updating elements
4. deleting elements.

creating dictionary

---

creating empty dictionary

```
a=dict()
print(a)
```

output:

```
{}
```

```
a={}
print(a)
```

output:

```
{}
```

Creating dictionary with elements:

```
a={"name":"Anand",1:"abc"}
print(a)
```

output:

```
{"name":"Anand",1:"abc"}
```

```
a=dict([("name","Anand"),(1,"abc")])
print(a)
```

output:

```
{"name":"Anand",1:"abc"}
```

Accessing elements of dictionary

---

using key over square brackets
using get() method

using key over square brackets

```
a={1:"a",2:"b",3:"c"}
print(a[1])
print(a[2])
print(a[3])
```

output:

---

a
b
c

error case:

---

if provided key doesn't exist in the given dictionary, then it will raise the key error

eg:

```
a={"name":"Anand"}
print(a["age"])
```

output:

---

keyerror

using get method:

---

syntax:

```
dict_variable.get(key,None)
```

eg:

```
a={1:"a", 2:"b", 3:"c"}
temp=a.get(1)
print(temp)
```

output:

"a"

```
a={1:"a", 2:"b", 3:"c"}
temp=a.get(5)
print(temp)
```

output:

None

special case:

```
a={1:"a", 2:"b", 3:"c"}
temp=a.get(10,"Not existed")
print(temp)
```

output:

```
"Not existed"
```

Adding elements to dictionary:

eg:

```
a={}
a[1]="a"
a[2]="b"
print(a)
```

output:

```
{1:"a".2:"b"}
```

```
a={}
a.update([('a',1),('b',2)])
print(a)
```

output:

```
{a:1, b:2}
```

Deleting elements from dictionary

pop()
popitem()
del
clear()

pop()

it is used to delete the element from dictionary based on the given key.

syntax :

```
dict_variable.pop(key)
```

eg:

```
a={"name":"Anand"}
a.pop("name")
print(a)
```

output:

```
{}
```

```
a={"a":2}
a.pop("b")
print(a)
```

output:

```
KeyError
```

del

eg:

```
a={"a":2,"b":2}
del a["a"]
print(a)
```

output :

```
{"b":2}
```

```
a={"a":5,"b":6}
del a["c"]
print(a)
```

output:

```
KeyError
```

popitem()

It is used to delete random element from the dictionary.

```
a={"name":"Anand","abc":2}
a.popitem()
print(a)
```

output:

```
{"name":"Anand"}
```

clear()
---

it is used to remove all the elements from the dictionary.

eg:

```
a={"a":1,"b":2}
a.clear()
print(a)
```

output:

```
{}
```

updating elements of the dictionary
---

eg:

```
a={"name":"Anand"}
a["name"]="sai"
print(a)
```

output:

```
{"name":"sai"}
```

```
a={"name":"sai"}
a.update([("name","Anand")])
print(a)
```

output:

```
{"name":"Anand"}
```

Other methods:

---

len()

---

it used to find the number of elements present in the dictionary

eg:

```
a={"a":"2","b":"cd","d":"123"}
print(len(a))
```

output:

3

keys()

It is used to get all keys of given dictionary.

eg:

```
a={"a":"123","b":"5","c":123}
print(a.keys())
print(list(a.keys()))
```

output:

```
dict_keys(['a', 'b', 'c'])
['a', 'b', 'c']
```

values()

It is used to get all keys of given dictionary.

eg:

```
a={"a":"123","b":"5","c":123}
print(a.values())
print(list(a.values()))
```

output:

```
dict_values(['123', '5', 123])
['123', '5', 123]
```

items()

It is used to get all key-value pairs of given dictionary.

eg:

```
a={"a":"123","b":"5","c":123}
print(a.items())
print(list(a.items()))
```

output:

```
dict_items([('a', "123"), ('b', "5"), ('c', 123)])
[('a', "123"), ('b', "5"), ('c', 123)]
```

Operators:

Operators are the special symbols that are used to perform specific computation on the values and variable.

Operanads

Operands are the values or variables where operator acts on.

Expression:

Sequence of operators and operands is called as expression

Eg:

2+3
Here "+" is operator
2,3 are operands.
2+3 is an expression

Types of operators

1.Arithmetic operators
2.Comparison operators
3.Logical operators
4.Identity operators
5.Membership operators
6.Bitwise operators
7.Assignment operators

1.Airthmatic Operators.

These operators generally used for performing mathematical operations on the numerical values.

Eg:

+	addition	2+3	-> 5
-	subtraction	5-2	-> 3
*	multiplication	2*3	-> 6
/	division	5/2	->2.5
//	floor division	5//2	->2
%	modulo division	5%2	->1
**	exponential	2**3	->8

Eg:

```
a=int(input())
b=int(input())
print(f"sum of {a} and {b} is {a+b}")
print(f"subtraction of {a} and {b} is {a-b}")
print(f"multiplication of {a} and {b} is {a*b}")
print(f"divison of {a} and {b} is {a/b}")
print(f"floor division of {a} and {b} is {a//b}")
print(f"modulo division of {a} and {b} is {a+b}")
print(f"exponention of {a} and {b} is {a**b}")
```

input :

3
2

Output:

---

sum of 3 and 2 is 5
subtraction of 3 and 2 is 1
multiplication of 3 and 2 is 6
divison of 3 and 2 is 1.5
floor division of 3 and 2 is 1
modulo division of 3 and 2 is 5
exponention of 3 and 2 is 9

2.Comparision Operators:

Comparison operators are used to compare values.
It returns either True or False according to the condition.

<	less than	2<3 -> True	5<3 ->False
>	greater than	6>3 ->True	3>10 ->False
==	equal to	2==2 ->True	3==5 ->False
!=	not equal to	4!=2 ->True	3!=3 ->False
<=	less than or equal	2<=2 -> True	3<=1 ->False
>=	greater than or equal	3>=1 ->True	5>=2 ->False

Eg :

```
a=int(input())
b=int(input())
print(f"{a} less than {b} result is {a<b}")
print(f"{a} greater than {b} result is {a>b}")
print(f"{a} equal to {b} result is {a==b}")
print(f"{a} not equal to {b} result is {a!=b}")
print(f"{a} less than or equal to {b} result is {a<=b}")
print(f"{a} greater than or equal to {b} result is {a>=b}")
```

input:

```
3
5
```

Output:

```
3 less than 5 result is True
3 greater than 5 result is False
3 equal to 5 result is False
3 not equal to 5 result is True
3 less than or equal to 5 result is True
3 greater than or equal to 5 result is False
```

3.Logical Operators :

These are generally used to compare and check the two or more conditional expressions.

and True if both expressions are True, otherwise False

or True if any one of the expression is True . Otherwise False

not True if expression is False, False if expression is True

A	B	A and B	A or B	not A	not B
False	False	False	False	True	True
False	True	False	True	True	False
True	False	False	True	False	True
True	True	True	True	False	False

(2<3) and (3>1) ->True (2==2) or (3==3) ->True not (2>3) ->True
(3<1) and (2!=3) ->False (2<=5) or (5>=2) ->True not (2==2) ->False
(2!=2) and (3>5) ->False (3<1) or (2>0) ->False not (5!=5) ->True

4.Identity Operators:

These operators are used to check whether given two operands refers the same memory location or not.
Simply it compares the memory addresses instead of values.

is	True, if both operands have same memory location, otherwise False
is not	True, if both operands have different memory locations, otherwise False

Eg:

```
a=1000
b=999+1
print(id(a),id(b))
print(a is b)
print(a is not b)
```

```
a=10
b=10
print(id(a),id(b))
print(a is b)
print(a is not b)
```

output:

```
1896488300976 1896488300688
False
True
```

```
1896452325904 1896452325904
True
False
```

5. Membership Operators:

These are used to check the presence of given value in collection or sequence.

in	True if value present in sequence	"a" in "anand" -> True "ba" in "anand" ->False
not in	True if value not present in sequence	2 not in [1,3] ->True 1 not in [1,3] ->False

eg:

```
a="abcd"
temp="ab" in "abcd"
print(temp)
```

output:

True

```
b=[1,2,3,4]
temp= 2 not in b
print(temp)
```

output:

False

```
a="abcdef"
print("xy" not in a)
```

output:

True

6.Bitwise Operators

These are used perform bit level operation on numerical value.

convert decimal integer to binary value;

```
a=2
print(bin(a))
```

output:

```
0b10
```

```
a=10
print(bin(a))
```

output:

```
0b1010
```

convert binary value to decimal integer

```
a="10"
temp=int(a,2)
print(temp)
```

output:

```
2
```

```
a="1010"
temp=int(a,2)
print(temp)
```

output:

```
10
```

steps involved

-> convert decimal values to binary
->perform operations on binary values
->convert resulted binary values into decimal value

Types:

1.Bitwise And (&)
2.Bitwise Or (|)
3.Bitwise Xor (^)
4.Bitwise Not (~)
5.Bitwise left shift (<<)
6.Bitwise right shift (>>)

Bitwise And (&)

if both bits are 1 then output is 1, otherwise output is 0

AND Table:

```
0       0       -> 0
0       1       -> 0
1       0       -> 0
1       1       -> 1
```

Eg:-

```
a=3                 3 ->   011
b=5                 5 ->   101
print( a & b )    3 & 5 -> 001 -> 1
```

output :

```
1
```

Bitwise Or (|)

if both bits are 0 then output is 0, otherwise output is 1

Or Table:

```
0       0       -> 0
0       1       -> 1
1       0       -> 1
1       1       -> 1
```

Eg:-

```
a=3                 3 ->   011
b=5                 5 ->   101
print( a | b )    3 & 5 -> 111 -> 7
```

output :

```
7
```

Bitwise Xor (^)

if both bits are different then output is 1, otherwise output is 0

Xor Table:

```
0       0       -> 0
0       1       -> 1
1       0       -> 1
1       1       -> 0
```

Eg:-

```
a=3                 3 ->   011
b=5                 5 ->   101
print( a ^ b )    3 & 5 -> 110 -> 6
```

output :

```
6
```

Bitwise Not (~)

if bit is 1 then output is 0 , if bit is 0 else 1

NOt Table:

```
0    ->    1
1    ->     0
```

Eg:-

```
a=3                  3 ->    0 1 1
print( ~a )         ~3->    1 0 0  -> -4
```

output :

```
-4
```

Bitwise left shift (<<)

it moves the bits of its first operand to the left by the number of places specified in its second operand. It also takes care of inserting enough zero bits to fill the gap that arises on the right edge of the new bit pattern.

Eg:-

```
a=3                  3 ->    0 1 1
print( a <<2 )      3 << 2 -> 01100  -> 12
```

output :

```
12
```

Bitwise right shift (>>)

it removes the bits of its first operand from right by the number specified in its second operand.

Eg:-

```
a=3                    3 ->    011
print( a >> 2 )        3 >> 2 -> 0 ->  0
```

output :

```
0
```

7. Assignment Operator

These are used to assign the values to variables .

In General, right operand is assigns to left operand.

we can also specific operation on right operand before assigning to left operand by mentioning its symbol.

Eg :-

```
a=3
b=6
c=10
name="Anand"
```

Special Case:

operand (symbol)=operand ->operand=operand (symbol) operand

```
a+=b    -> a=a+b
a-=b    -> a=a-b
a*=b    -> a=a*b
a/=b    -> a=a/b
a//=b    -> a=a//b
a%=b    -> a=a%b
a&=b    -> a=a&b
a|=b    -> a=a|b
a^=b    -> a=a^b
a>>=b    -> a=a>>b
a<<=b    -> a=a<<b
```

Control-flow statements:

---

statement:-

In python programming, any instruction that python interpreter can execute is called as statement.

control-flow:

The order of execution of program is called as control flow.

Control-flow statements:

---

statements which can regulate/change the normal execution flow of a program are called as control flow statements.

Types of control flow statements:

---

1. conditional (or) decision (or) selective (or) branching
2. iterative (or) Looping (or) repetetive
3. transfer

conditional Statements

These statements are used to execute a block of statements based up on the result of certain condition.

Types Of Conditional Statements :-

1.if
2.if else
3.elif
4.nested if
5.match

Truthy Values	Falsy Values
True	False
Any Integer except 0	0
any float except 0.0	0.0
string with length at least 1	empty string
list with length at least 1	empty list
tuple with length at least 1	empty tuple
set with length at least 1	empty set
dict with length at least 1	empty dict
	None

1.if

Syntax:

```
if  (<expression>):
    <statement>
    <statement>
```

If the expression returns Truthy value,
then the statements will execute, otherwise those statements inside if block will not executed (or) skipped in execution flow.

```
if (<expression> is True):
     statements will execute
if(<expression> is False):
      statements will be skipped
```

Eg :

```
a=2
if(a>1):
   print("a is greater than 1")
if (a>5):
   print("a us greater than 5")
```

output:

```
a is greater than 1
```

if - else:

The if..else statement evaluates expression and will execute the body of if only when the test condition is Truthy.
If the condition is Falsy, the body of else is executed.

```
if <expression>:
    <Body of if>
else:
    <Body of else>
```

eg:

```
num=10
if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

output:

---

positive or zero

if-elif-else:

---

the elif is short for else if. It allows us to check for multiple expressions.

If the condition for if is False, it checks the condition of the next elif block and so on.

If all the conditions are False, the body of else is executed.

Only one block among the several if...elif...else blocks is executed according to the condition.

The if block can have only one else block. But it can have multiple elif blocks.

syntax :

```
if <expression>:
    <Body of if>
elif <expressions>:
    <Body of elif>
else:
    <Body of else>
```

Eg:

```
num=10
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

output:

```
Positive number
```

Nested - if

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. They can get confusing, so they must be avoided unless necessary.

Eg:

```
num=0
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

output:

```
zero
```

Iterative Statements;

Iteration :

The process of repetitive execution of block of ststements over and over again is called as iteration.

types:

1.definite iteration
2.indefinite iteration

definite iteration :

The process of iteration, in which the number of iterations are explicitly declared in advance is known as definite iteration.
Eg :- for loop

indefinite iteration :

It is the process of iteration in which the block of statements will be executes untill some condition met.

Eg : - while loop

For loop :

Syntax :

```
for <iterator> in   <iterable> :
        <block of statements>
```

Iterable :

It is an object through which we can loop over.

An object is said to be iterable , if and only if it can form iterator from iter() function.

Eg:

```
a="1233"
print(iter(a))
```

output:

```
<str_iterator >
```

```
a=[1,2,3]
print(iter(a))
```

output:

```
<list_iterator>
```

Valid iterables:	invalid iterables
string	Integer
list	Float
tuple	complex
set	Boolean
frozenset	
dictionary	
range	

Iterator :

Iterator is an object which can be used to take the value one by one from the iterable.

It remembers the position during execution of for loop.

It consists of next function, which will be called to get the value from iterable,

Eg:

```
temp=iter("Anand")
print(next(temp))
print(next(temp))
print(next(temp))
print(next(temp))
print(next(temp))
```

output:

```
A
n
a
n
d
```

flow of execution of forloop

->Generates iterable by using iter function
->calls next function to get current item
->this process is continuos untill it gets stop iteration error

Types of forloop

1.range based
2.iterable based

range

It is a built in function which can be used generate sequence of itergers as per given constranits.

syntax:

range(stop)
range(start,stop)
range(start,stop,step)

start -> starting value of sequence, default is 0
stop -> ending value of sequence
step -> size of incrementation during for each value, default is 1

eg:

---

range(2) ->[0,1]
range(4) ->[0,1,2,3]
range(3) ->[0,1,2]
range(1,4) ->[1,2,3]
range(4,8) ->[4,6,7]
range(0,2) ->[0,1]
range(3,3) ->[]
range(1,5,2) ->[1,3]
range(1,6,3) ->[1,4]

range based for loops:

Eg;

```
for i in range(2):
    print(i)
```

output:

0
1

```
for j in range(1,8,2):
    print(j)
```

output

1
3
5
7

collection (or) iterable based

---

eg:

```
a=[1,2,3]
for i in a:
    print(i)
```

output:

```
1
2
3
```

```
a=(10,20,30)
for j in a:
    print(j)
```

output:

```
10
20
30
```

```
a="abcd"
for j in a:
    print(j)
```

output:

```
a
b
c
d
```

Nested for loop

---

if one for loop written inside another for loop ,
then it is called as nested for loop.

eg;

```
for i in range(3):
    for j in range(2):
        print(i,j)
```

output:

```
0 0
0 1
1 0
1 1
2 0
2 1
```

while loop

-> It is completely based on condition
-> untill the given condition returns falsy value, the while block will be executed continuously.

syntax :

```
while <condition>:
    <statement>
    <statement>
    <statement>
```

Eg:

```
a=5
while a>1:
    print(a)
    a-=1
```

output:

```
5
4
3
2
```

```
a=[1,2,3]
while a:
    print(a.pop())
```

output:

```
3
2
1
```

Functions

It is defined as self contained block of statements that is used to perform specific task.

it allows us to divide the program into smaller basic blocks called function.

A function can be called multiple times, so that it provides code reusability and modularity.

we know that, variable is a name that used to hold a value.

similarly, function is a name which is used to hold the collection of statements.

Syntax :

```
def <function_name>([<parameters>]):
    <statement(s)>
```

creating empty function

```
def myfunc():
   pass
myfunc()
```

creating function with statments:

```
def  myfunc():
     print("Anand")
myfunc()
```

output:

Anand

```
def myfunc(a,b):
     print(a+b)
myfunc(2,3)
```

output:

5

return statement:

A return statement is a keyword which is used to end the execution of the function call and "returns" the result to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned.

default value : None

return statement can used inside function only.

eg:

```
def myfun():
    pass
print(myfunc())
```

output:

```
None
```

```
def myfunc():
    return 10
print(myfunc())
```

output:

```
10
```

sum of two numbers using functions:

```
def sum(a,b):
    return a+b

a=int(input())
b=int(input())
print(sum(a,b))
```

input:

```
2
3
```

output:

```
5
```

```
def sum(a,b):
    s=a+b
    print(s)

a=int(input())
b=int(input())
sum(a,b)
```

input:

```
2
3
```

output:

```
5
```

Types of Arguments of function

1. positional arguments
2. default arguments
3. keyword arguments
4.variable length arguments

positional arguments

it is based on position, in this it assigns the ith (first, second, and so on) argument to the ith (first, second, and so on) function parameter is called positional parameter passing, while arguments passed in this way are named positional arguments.

Eg:

```
def myfunc(a, b, c):
    print(a, b, c)
myfunc(1, 2, 3)
```

output:

```
1 2 3
```

default arguments:

---

these arguments can be used to give the default value to the arguments in function, so that if we won't pass that argument in function call, it will take the default value.

eg

```
def myfunc(a,b=10):
    print(a,b)

myfunc(5)

myfunc(10,5)
```

output:

---

5 10

10 5

keyword arguments:

we can also pass key value pairs as arguments while calling a function, so that we don't need to worry about the order of arguments.

eg:

```
def myfunc(a,b):
    print("a is ",a)
    print("b is ",b)
myfunc(a=3,b=5)
myfunc(b=5,a=3)
```

output:

```
a is 3
b is 5
a is 3
b is 5
```

variable length parameters;

These are used to pass multiple argumets at a time.
we can also pass multiple keyword arguments at a time.

```
def myfunc(*args):
    print(args)
myfunc(1,2,3)
myfunc(1,2,3,4,5)
```

output;

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

```
def myfunc(**args):
    prints(args)
myfunc(a=2,b=3)
myfunc(a=2,b=4,c=5)
```

output:

```
{"a":2,"b":3}
{"a":2,"b":4,"c":5}
```