PYTHON

Python is general P high level Programming language.

Python is used for Multi P

1. web desiging
2. Data science application.
3. Desktop application.

High level programming language:-

Machine level language → It is in the Zero's and One's

↓

binary or boolean.

Assembily-level language:-

* It is in the Minmonic-codes.

Ex:-

Deepu sent a message.

Minimonic - 10101 0101 10101

code.

* Above two language are low-level languages.

High level language.

Ex:-

C, C++, Java, python.

History:-

* The father of Python "Guido van Rossam".
* Developed at National reasuch institute of Neatherland in 1989.

* Afficialy Python availble to public 1991 Feb 20.
* Java avaroble to public 1995.
* Best suitable language to begineg's.
* Best suitable language to the artifical intlizence.
* less code in python.

why python - world:-

* A popular tv show in BBC CBritish Broad coasting) that name Monty Python flying circus (1969 - 1974) (Fun show).
* The world taken from this show.

Borrowing features from author to python:-

* Functional programming features from 'c'.
* Object oriented features from "c++"
* scripting languages features from perl and shell scripting.
* Modular programming features from Modula-3.
↓
(Modules dividing)
* Most of the syntax related from 'c' and 'abc' languages borrowing to python.

where we can use python:-

* Desktop application (system based)

Ex:- calculator.

* web application.

Ex:- Fram work.

* Data base applications.

Ex:- DBMS (SQL (PL SQL)

* For Networking applications.

Ex:-
wifi

* Gameing.
* Data science with only Python.
* Machine leaing.
* Artefical intilizence application.
* For internet of things applications.

↓

(Iot) → It is Just like a automachine.

which companies using :-

* Goggle - searching algaritham.

↓

step by step

* utube.
* Drop Box - using Internally.
* NASA.
* NASA - (National sequrity aggency)
* Facebook.
* Instagram.
* Mozila file fox.

Featuges of python :-

* Simple and Easy to learn.
* open souce Softwaje.
* we can use softwaje souce code.

Ex:-
UNIX

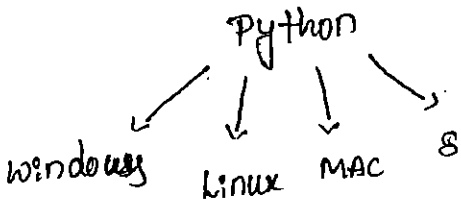
* Free waje.

we can pay single pies of amount by buging lincenejes. free of cost.

* High level programming language.
* python is a platform Idependent.
* Write once use any wheje (WORA)

↓

Run

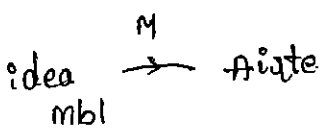


* Portability.

Moving python programme one platform to another platform we does not any change of code.

Ex:-

Ex:-



* Dynamically typed programming.

Based on the given values automatically decelaje the datatype in python.

* Both Procedure oriented and object oriented language

↓

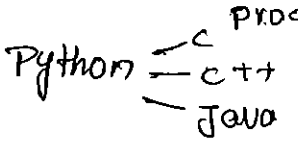
By using global declejation. access the data.

↓

Inhejitence, Polymorphism encapsulation.

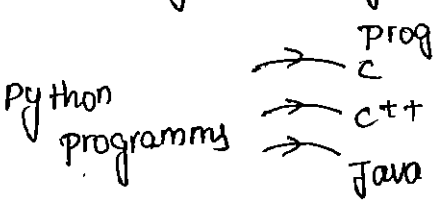
* Intejpreted Programming language.
* Internally python Intejpreteq takecaje of code.
* compilation fails python intejpreteq rise the Error
* Extensiable.

we can use otheq Programming languages into



* Embeded.

we can use python programms in any other languages.



* Extensive Libaries.

Ex:-
import Math
Math.Sqrt(49)
7.0,

Limitations of Python:-

* peqformence wise not upto mark because Inteqpreted programming language.
* Mobile application is not aging.
* Python is not suitable for large scale enteq prize applications.

Flavours of Python:-

cPython - standard flavour
JPython / JyPython - Java applications
Iron Python - To work with cSharp.net.
PyPy - Inteqnally Inside python veqtical machine increase the peqformance.
Ruby Python - TO developed Ruby based application.
Anaconda Python - To handel big data processing.
Stackless Cpython for concuqency) - paqelly things executed

Ex:- Multi threadings.

Veqsions of Python:-

* Python 1.0 introduced in 1994 June
* Python 2.0 introduced in 2000 oct

* Python 3.0 introduced in 2008 Dec
* Python 3.0 sub version 3.6.3 introduced in 2016.
* Present version of Python 3.9.1

Note:-
Any new version provide support for old version Programme python 3 programmes Never Run Python to version

Identifiers:-

* A Name in python is called identifier.

↓

variable name, class name (or) Method name.

Rules:-

* Alphabets (lower case and upper cases)
* It supports 0 to 9 numbers.
* It support a- (-) symbool.
* statement beging with only chapcter but not with number.

Ex:-

1. >> cash = 120
   >> cash (validating)
   120.

Ex:-

2. >> 123 chitti = 40
   >> 123 chitti
   Syntax Error : invalid.
* Statement begin with chapcter but not with number.

Ex:-

3. >> cas$h = 90
   Syntax Error : invalid.
* Python language is case sensitive.

Ex:-

>> TOTAL = 100
>> Print (TOTAL)
100

>> total = 100
>> print (total)
100.

* keywords are not used in identifiers.

Ex:-
1. >> x = 10
>> x
10

2. >> if = 89
syntax Error.

3. >> def = 90
syntax error.

1) Maximum length allowed for python identifier.

a) 32 (or) 64
b) 64 (or) 128
c) 128 (or) 256
d) No length limit.

* Python identifier it have no length limit but not decamended to take two length identifier.

* Identifier's should not start with digit.

Ex:-
>> 1255 = 23
syntax error.

* Identifiers starts with (-a) symbol it is private.

Ex:- _ = (private)

* Identifiers starts with -a-b symbols it is stoongly recomended private.

Ex:- __a (stoongly protected)

* Start with (-) and ends with (-) it is language specifies identifiers, It is a special variable defined by python itself.

Ex:- __main__ (special variable)

Reserved words:-

* Reserved means it having fixed meaning. It supporting 33 keywords.

>> import keyword

>> keyword.kwlist

1. True, False, None
   - True, False → Boolean
   - None → Nothing is there.
2. and or is not.
3. if else elif.
4. while, for, break, continue, Return, in, yield.
5. try, Exept, finally, raise, assert.
6. import, from, as, class, def, pass, global, non-local, lambda, dl, with.
   - lambda → ("Annanimous functions")

To declare function without name as called it annanimous function.

Quiz-1:-

1) Is Python is case sensitive when declaring the identifers?
   ~~Yes~~ 2) No 3) Mention depended 4. None of the above.

2) what is maximum possible length of identifers?
   1) 31 character 2) 63 character 3) 79 character ~~4~~) Non of above

3) which of the following is invaild?
   1) _a = 1 2) __a = 1 3) __str__ = 1 4) Non of the above.

4) ~~1~~) They are used to indicate a private variable of a class.
   2) They confuse the interpreter.
   3) They are used to indicate global variables.
   4) They slow down Execute.

why are local variable names beging with (-) discouraged?

5) which of the following is not keyword?

1) eval 2) Assert 3) Non local 4) pass.

6) All keywords in python are in ?

1) lower case

2) upper case

3) capatilized

4) Non of the above.

7) which of the following can't be variable?

1) - int - 2) in 3) it 4) on

Data types:-

* It represent the type of value.
* Automatically it create datatype
(or)
provide.

* int
* float.
* complex data type - scientific calculation.
* Bool - Boolean values.
* Bytes.
* Byte array.
* Range - Range of numbers.
* List - Group of values.
* tuple - list of values.
* set - list of values without duplicate.
* frozenset - list of values without duplicates and we cannot change the values.
* dict - key values (map concept)
* None - Nothing.
* str - String values.

Note:-

In integer values - long type is available python 2. but not

avaiable in python 3.

* In python everything is an object.

Python provides inbuilt function:-

* print()
* type()
* id()
* .... etc.,

1) int():-

To represent integer values [without decimal points]

Ex:-

```
>> a=100
>> type(a)
<class 'int'>
```

Based on integer representation:

* Decimal form.
* Binary form
* octal form
* Hexa decimal

1) Decimal form:-

Base is 10 i.e 0 to 9.

Ex:-

```
>> a=4848
>> a
4848
>> type(a)
<class 'int'>
```

2) Binary form:-

Base is 2 i.e 0 and 1

* "In python numbers starts with 0b (or) 0B".

Ex:-

1. >> a=0b1010

>> a
10
>> type(a)
<class 'int'>

2. >> a = 0B1111
>> a
15
>> type(a)
<class 'int'>

3) Octal form:-

Base is 8 i.e 0 to 7

In python numbers starts with 0o (or) 0O
↓ zero 'O' ↓ capital 'O'

Ex:-

1. >> w = 0o1111
>> w
585
>> type(w)
<class 'int'>

2. >> Deepu = 00123
>> Deepu
83
>> type(Deepu)
<class 'int'>

4) Hexa decimal form:-

Base is 16 i.e 0 to 9.
a to f (A to F)

* In python numbers starts with 0x (or) 0X

Ex:-

>> e = 0x123af
>> e
74671

```
>> type (e)
<class 'int'>
```

2) `>> f= 0x12ef`

```
>> f
4847
>> type (f)
<class 'int'>
```

3. `>> r= 123re`

```
error
```

* It is beyand (or) outoff the limits i.e Base is 16 a to f, 0 to 9.

③ Float Di-

Ex:-

```
>> f = 12.4849
>>f
12.4849
>> type (f)
<'float'>
```

id():-

To know the address of variable.

Ex:-

```
>> f = 12.456
>>f
12.456
>> type (f)
<class 'float'>
>>id(f)
60 665264
```

Base conversion:-

* To convert the information into particular Base

Bin():-

1. >> bin(10)
'0b1010'

2. >> bin (00123)
'0b1010011'

3. >> bin (0x123ef)
'0b1001000111101111'

4. >> bin (12.45)
'Error'.

* In binary int, octal, hexadecimal conversion is possible.

oct ():-

1. >> oct (10) # int
'0012'

2. >> oct (0b1111) # bin
'0017'

3. >> oct (0x12ef) # hexdec.
'0011357'.

* In octal conversion decimal, binary, hexadecimal is possible.

hex ():-

1. >> hex(10)
'0xa'

2. >> hex (0b111)
'0x7'

3. >> hex (00123)
'0x53'

* In hexadecimal conversion decimal, octal, binary is possible.

Float :-

1. >> f = 12.12
>> f

12.12

2. >> f= 0b1010.1

Error

3. >> f= 0012 3.12

Error.

4. >> f = 0x123ef.14

Error.

* In float conversion binary, octal, hexadecimal in not possible decimal form is only possible.

Note:-

Exponential form (or) scientific nation - E core

i.e 10 to the power of value.

Ex:-

1. >> f= 10e^2

$\frac{10 \times 10^2}{1000}$

>> f

1000.0

2. >> d= 2e^3

>> d $\frac{2 \times 10^3}{2 \times 1000}$

$\frac{}{2000}$

2000.0

3. >> w = 10e^4

>> w

100000.0

(3) Complex datatype :-

a + bi

↓ ↳

Real part imaginary part

$j^2 = -1$

$j = \sqrt{-1}$

Ex:-

1. >> f= 10+34j
   >> f
   (10+34j)
   >> type (f)
   <class 'complex'>

Retrive values:-

>> f = 10+34j

1. Real part:
   >> f.real
   10.0

2. imaginary part:-
   >> f.imag
   34.0

conversion:-

I)

* Real part accept any kind of values but imaginary part accept only decimal values.

Ex:-

1. >> d= 0b1010+12j #bin
   >> d
   (10+12j)

2. >> d= 0o123 +12j #oct
   >> d
   (83+12j)

3. >> d= 0x12ef +12j # hexdec
   >> d
   (4847 +12j)

4. >> d= 12+67j # dec num
   >> d

(12 +67j)

II

* In imaginary Part decimal, Float values are possible. In imaginary Part binary, octal, hexadecimal are not possible.

Ex:-

1. >> a = 12+0b1010j

Error.

2. >> a = 12+00123j

Error

3. >> a=12+0x123ef

Error

4. >> a= 12+12.45j

>>a

(12+12.45j)

5. >> a= 12+65j

>> a

(12+65j)

④ Bool Data type:-

True Means one

False Means Zero.

Ex:-

1. >> a= True

>>a

True

>> type(a)

<class 'bool'>

~~class~~

2. >> b= False

>> b

False

>> type(b)

<class 'bool'>

Truth table:-

```
>> True + True
2
>> True + False
1
>> False + True
1
>> False + False
0
```

Note:-

"T" is captial in True

"F" is captial in False

Ex:-

```
>> a=10
>> b= 20
>> a<b

True
>> a>b
False
```

⑤.

String str():-

string can represent in ' ' and double quatation " " and single triple quatation ''' '''

1.

```
>> c= "chinnu"
>> c
'chinnu'
>> type(c)
<class 'str'>
```

2.

```
>> d = 'Milky'
>> d
'Milky'
>> type(d)
```

<class 'str'>

>> e = ''' kanna '''

>> e

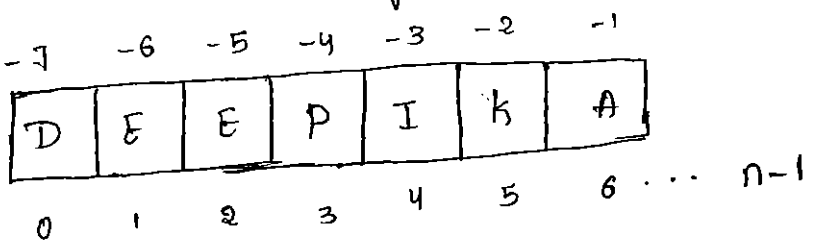
'kanna'

>> type (e)

<class 'str'>

slicing operation:-

* "[ ]" operator is called slice operator.
* python can follow zero based index method.
* Index can be positive numbers +ve . left to Right direction.
* Index can be -ve numbers right to left direction.



left to ringht +ve

Syn:-

D [bigan : end]

↳ is optional

Ex:-

1. >> a = 'deepika'

>> a

'deepika'

>> a[0]

'a'

2. >> a[2]

'e'

3. >> a[-1]

'a'

4. >> a[-4]

'p'

5. >> a[:]

'Deepika'

The default index value is zeroth possition . end possition is index -1 value.

6. >> a[:3]

'Dee'

7. >> a[-4:-1]

'pik'

8. >> a[100]

Error

String index is out of range

9. >> a[-1:-2]

"Emrty"

If any two distination negative index value it return Empty Parainthsis.

Ex:-

1. c = "book"

>> c

'book'

2. c * 5

'books books books books book'

len ():-

To find the Peaticular string length.

Ex:-

>> e = "Mythi ojas institute'

>> e

'Mythi ojas institute'.

>> len(e)

19

Ex:-

2. >> len ("Deepu")

5

3. >> len ("you my cutie")

13

Note:-

- int
- float
- complex
- Bool
- Str

These are called fundamental datatype.

Type conversion:-

int ():-

To convert the particular type into integer data type.

Ex:-

```
>> int(10)    # decimal
10
>> int (12.45)  # float
12
>> int  (0b111)  # binary
7
>> int  (0o123)  # octal
83
>> in (0x123cd)  # hexa decimal
+6527
>> int (True)   # bool
1
>> int (False)  # bool
0
>> int ('10')  # string
10
>> int (1+2j)  # complex
Error.
```

Note:-
To convert the { decimal
octal
hexa decimal
binary
bool } conversion is possible in int.

{ complex
str } conversion is not possible in int.

Float():-
To convert the perticular type into integer data data.

Ex:-
>> float (10) # dec
10.0
>> float (0b111) # binary
7.0
>> float (00123) # octal
83.0
>> float (0x123c) # ~~complex~~ hexa decimal
4847.0
>> float (True) # bool
1.0
>> float (False) # bool
0.0
>> float (1+12j) # complex.
Error
>> float ("chinnu") # str.
Error.

Note:-
To convert the { decimal
octal
hexadecimal
binary
bool } conversion is possible in float data type

{complex
str } conversion is not possible in float datatype.

complex() - method I :-

Ex:-

>> complex(10) # dec

(10+0j)

>> complex(10.5) # float

(10.5+0j)

>> complex(True) # bool

(1+0j)

>> complex(False) # bool

0j

>> complex(0b111) # binary

(7+0j)

>> complex(0o123) # octal

(83+0j)

>> complex(0x12acf) # hexdec

(76527+0j)

>> complex("Teja")

Error.

To convert the {decimal
octal
float
binary
bool
hexa decimal } conversion is possible in complex data type.

{str} conversion is not possible in complex data type.

Bool:-

Non zero is True - 1

Zero is False - 0

Bool to Decimal :-

```
>> bool (0)
False
>> bool (1)
True
>> bool (10)
True
>> bool (-10)
True.
```

Bool to float :-

* In float form every decimal point zero means false.

* In float form every decimal point non - zero means True.

Ex:-

```
>> bool (0.0)
False
>> bool (12.12)
True
>> bool (0.01)
True
>> bool (0.02)
True
>> bool (0.1)
True
>> bool (0.0000)
false.
```

complex to bool :-

Real part	imaginary part	Result.
F	F	F
T	F	T
F	T	T
T	T	T

Ex:-

```
>> bool (0+0j)
False
>> bool (0+1j)
True
>> bool (1+0j)
True
>> bool (1+1j)
True.
```

Bool to string as arguments :-

* If arguments is empty string it treats as false, in other all cases it is true.

* Space is treated as one character.

```
>> bool(" ") empty arguments.
```

Ex:-

```
False
>> bool (" ")
True
>> bool (' ')
True
>> bool ('hi')
True
>> bool ("chinnu")
True.
```

Str():-

It convert any value to string.

Eg:-

```
>> str(10)
'10'
>> str(10.24)
'10.24'
```

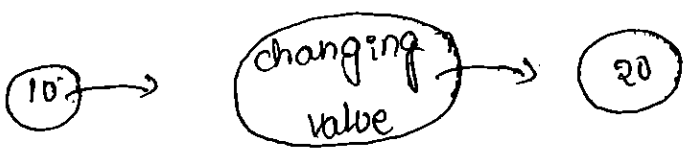
```
>> str (10+34j)   # complex
'(10+34j)'
>> str (True)   # bool
'True'
>> str (False)   # bool
'False'
>> str (0b1111)   # binary
'15'
>> str (0o123)   # octal
'83'
>> str (0xa12)   # hexdec
'(2578)'
```

Immutable verses fundamental datatype:-

All fundamental datatype are immutable. once create a object we can peqformence change in that object, if we are try to change then create new object.

Immutable → Nonchange

Mutable → changable.



object 1 object 2

Internally doesnot change the value it create the new object.

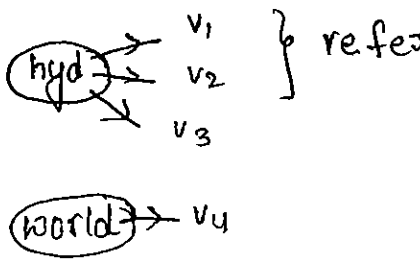
Note:-

* All fondamental datatypes are immutable.

* Everything in python object only.

* object reuseability in python.

Ex:-



* If same value in object reuse it.

* If same content required repeatedly not recomended to create seperate objects, we can create one object all refer-ances to be share.

Ex:-

```
>> v1 = "hyd"
>> v2 = "hyd"
>> v3 = "hyd"
>> v4 = "world"
>> id(v1)
65433408
>> id(v2)
65433408
>> id(v3)
65433408.
>> id(v4)
65433504
```

* Memory utalization Performance is there.

Note:-

* Immutability means non changable.

* Mutability means changable.

Bytes:-

* Group of values to hold and we can use it.

* It is immutability.

Ex:-

```
>> x = [10, 20, 30]
>> b = bytes(x)
>> type(b)
<class 'bytes'>
>> b[0]
10
>> b[-1]
30
>> b[1]
20
>> b[1] = 100
```

Type error:- 'bytes' object does not support assignment.

```
>> for x in b: print(x)
10
20
30
```

1.

```
>> x = [10, 20, 256]
>> b = bytes(x)
```

output :- error.

Invalid because bites must be in the Range of (0, 256)

```
>> x=[10, 20, 255]
>> b=bytes(x)
>> for x in b: print(x)
10
20
255
```

⑥

Byte array :-

It hold group of values. This is mutable in the range of 0 to 256

```
x = [10, 20, 30, 40]
```

```
>> b = bytearray(x)
type(b)
<class 'bytearray'>
>> for x in b: print(x)
10
20
30
40
>> b[0]
10
>> b[-1]
40
>> b[0]
10
>> b[0]=100  # 10 is Replaced with 100 so that assignment is possible.
>> for x in b: print(x)
100
20
30
40
>> b[1]=200
b[1]
200
>> x = [10, 20, 256]
>> b=bytearray(x)
Error
ValueError: byte must be in range (0, 256)
```

Note:-
No one or used bytes and bytearray generally

List data type:-
It contained group of values. list of values represent in a [].

⑦

list():-

* ordered allowed.
* duplication allowed.
* hetrogenious allowed.
* growable Nature allowed - increase and decrease possible.
  ↓ (or) ↓ (or)
  add remove

* values enclosed with in the []

```
>> l=[]
type(l)
<class 'list'>
>> l.append(10)
   l.append(20)
   l.append(30)
>> print(l)
[10, 20, 30]
>> l.append(20)   # duplication allowed
print(l)
[10, 20, 30, 20]
```

Note: Hetrogenious objects means different type objects allowed.

```
>> l.append('Deepa')
>> l.append(12.34)
>> print(l)
[10, 20, 30, 20, 'Deepa', 12.34]
```

* Null is not python.
* None is allowed.

```
>> l.append(None)
print(l)
[10, 20, 30, 20, 'Deepa', 12.34, None]
>> l[0]
10
>> l[0:5]
[10, 20, 30, 20, 'Deepa']
```

```
>> l[-1]
print (l[-1])
None
>> l[-2]
12.34
>> l.remove(30)
>> l
[10, 20, 20, Deepa, 12.34, None]
>> l.remove(20)
>> l
[10, 20, Deepa, 12.34, None]
>> l.remove(l[-1])
>> l
[10, 20, 'Deepa', 12.34]
>> d = l*2
>> d
[10, 20, 'Deepa', 12.34, 10, 20, Deepa, 12.34]
```

(8)

Tuple data type :-

* A Tuple contain group of values.

* list and Tuple are same but list is a mutable, Tuple is a immutable. Tuple can mention in paranthesis once we create tuple we can't modify the data range not their like zero to 256 It is only bytes and byte array.

Ex:-

```
>> t = [10, 20, 30, 40, 90]
>> type(t)
<class 'tuple'>
>> t[0]
10
>> t[-1]
60
>> t[0:4]
```

(10, 20, 30, 40)

>> t[0] = 120

>> t[0] = 120

'tuple' object does not support item assignment.

>> t = (10, 'Deepu', 'chitti', 10, 20)

>> print(t)

(10, 'Deepu', 'chitti', 10, 20)

* Duplication is available. It following the order

>> t1 = t*2

>> t1

(10, 'Deepu', 'chitti', 10, 20, 10, 'Deepu', 'chitti', 10, 20)

(9) Range datatype :-

* Represent the sequence of value.
* It is always immutable
* Range is a data type and also function.
* Range is always commonly used data-type.

Form I :-

Syn :- range(End)

* It represent the values from zero to End-1 (or) n-1

>> range(10)

range(0, 10)

>> type(range(10))

<class 'range'>

>> r = range(10)

type(r)

<class 'range'>

>> for i in r: print(i)

0

1

2

3
4
5
6
7
8
9

```
>> r[0]
0
>> r[0:3]
Range [0,3]
>> r[0]= 100
Error.
Type Error: 'range objects does not support items assignment.
```

Form II :-

Range (10, 20)
↓ (starting) ↳ (end-1)

Ex:- r= range (10,20)

```
for i in r = print(i)
10
11
12
13
14
15
16
17
18
19
```

Form III :-

Range (10, 20, 30)

↓ ↓ ↓

Start (End-1) Step (every time increment by two step 5)

Ex:-

```
>> r= range (20, 40, 2)
>> for i in r: print(i)
20
22
24
26
28
30
32
34
36
38
40
>> r= range (20,40,3)
>> for i in r: print(i)
20
23
26
29
32
35
38
>> range (10.5 , 12.5)
Error
Type Error: 'float' objects cannot be interpreted as an integer.
```

Note:-

{ * Bytes
  * list
  * Tuple
  * Range } Immutable.

{ * Bytearray } → Mutable.

(10) Set data type():-

* It can Represent the information in curly brackets {}
* Insertion ordered not allowed.
* Duplication not allowed
* Heterogenious objects are allowed.
* Index value not allowed.
* Slicing not applicable.
* It is mutable.

Eg:-

```
>> s = {10, 20, 30, 10, 20, 30}
>> type(s)
<class 'set'>
>> print(s)
{10, 20, 30}
>> s[0]
```

Error → Indexing slicing such type of terminology not application for set because set internally ordered is not there.

```
>> s.add('chitti')
>> print(s)
{10, 'chitti', 20, 30}
>> s.add(10.23)
>> s
{10, chitti, 10.23, 20, 30}
>> s.remove(20)
>> s
{10, chitti, 10.23, 30}
```

* append is useful only list datatype not for set data-type.

```
>> c = {'hi', 'good', 'mng'}
>> print(c)
{'mng', 'hi', 'good'}
```

* Order is not important in set data type.

(11)

Frozenset Datatype:-

* Group of unique values does not change any values we can in frozenset.
* It is Exactly same as set data type.
* It is immutable.
* Indexing not applicable
* Hetrogenious is allowed.
* order is not important.
* Duplication not allowed.

Ex:-

```
>> s= {20, 10, 'hello'}   # set datatype.
>> fs = frozenset(s)
>> type(fs)
<class 'frozenset'>
>> print(fs)
frozenset ({10, 20, 'hello'})
>> fs.[0]
Error
  ↓
'Frozenset' object is not subscriptable.
>> fs.add(13)
Error
>> fs.remove(10)
error.
```

* Add and remove not possible in frozenset datatype.

(12)

Dict datatype:-

Dictonary

* Group of objects as a key value

key	value
100	Deepu
200	chitti
300	Minnu

Duplication not allowed but values can be duplicated.

```
>> d= {100: 'Deepu', 200: 'chitti', 300: 'Minnu'}
>> type(d)
<class 'dict'>
>> print(d)
{100: 'Deepu', 200: 'chitti', 300: 'Minnu'}
>> d1 = {}
>> type(d1)
<class 'dict'>
```

If the {} it is like a dict datatype not a set data type.

```
>> d1[10] = "Mom"
>> d1
{10: "MOM"}
>> d1[20] = "dad"
>> d1
{10: 'Mom', 20: dad'}   # dad replaced by sis.
>> d1[20] = "sis"
>> d1
>> {10: 'MOM', 20: 'sis'}   # Mom replaced by bro
>> d1[10] = "bro"
>> d1
{10: 'bro', 20: 'sis'}
```

* Dict is Mutable datatype.

Note:-

Bytes and bytearray represent binary data like images, vedios files and audio files.

(18)

None data type:-

* None means nothing no value is associated. It is just like null values but python does not support null value concept.

```
>> a = None
>> type (a)
<class 'None type'>
```

Escape sequence characters:-

* \n - new line

Ex:-

```
>> s = "welcome \n to \n my world".
>> print(s)
welcome
 to
 my world.
```

* \t - horizantal

```
>> s = "welcome \t to \t my world".
>> print(s)
welcome          to          python class.  My world.
```

* \v - vertical

```
>> s = "welcome \v to \v My word"
>> print(s)
 welcome
          to
          My world.
```

* \f - formfeed - like page down

```
>> s = "welcome \f to \f python class"
>> print(s)
welcome □ to □ python class
```

* \' -> Back slash with single quotation.

```
>> s = "welcome \'python"
>> print (s)
welcome 'python
```

* \" -> Back slash with double quotion.

```
>> s = welcome \"python"
>> print(s)
```

welcome "python".

* \\ -> double back slach.

>> s = "welcome \\ python class"

>> print(s)

welcome \ python class

Datatype symbol:-

bytes [] -> Immutable

* bytearray [] -> Mutable

list [] -> Immutable.

tuple () -> Immutable

Range () -> Immutable

set {} -> Immutable

forzenset {} -> Immutable

* dict {} -> Mutable.

Operators :-

1. Mathematic o.p
2. Relational o.p (or) comparision o.p
3. logical operator.
4. Bitwise operator.
5. Assignment operator.
6. special operator.

Mathematic operator:-

+ - add

- - sub

* - Mult

% - Modulas div

/ - div

// - Flor division

** - Exponent.

* flor division:-

* Argument int display integer values

* Argument float display float values.

Ex:-

```
a=10
b=2
Print ('a+b =', a+b) #12
Print ('a-b=', a-b) #8
Print ('a*b =', a*b) #20
Print ('a/b =', a/b) #5
Print ('a %.b =', a %.b) #0
Print ('a //b =', a //b) #5
Print ('a**b =', a**b) #100
```

output:-

a+b=12

a-b=8

a*b=20

a/b=5

a%.b=0

a//b=5

a**b=100 (10^2) (a^b)

2.

```
a=10.5
b=2.5
print ('a+b =', a+b)
print ('a-b=', a-b)
Print ('a*b=', a*b)
print ('a/b =', a/b)
Print ('a%.b=', a%.b)
Print ('a//b =', a//b)
```

print ('a ** b =', a ** b)

output:-

a+b = 13.0

a-b = 8.0

a*b = 29.25

a/b = 4.2

a%b = 0.5

a//b = 4.0

a ** b = 351.2508309997333.

* + Operator:-

It is used for concatation of two strings

Ex:- "Deepu" + "4"

1. 'Deepu4'.

2. "Deepu" + str(4)

'Deepu 4'

* If arguments are strings it display the result
* If arguments is number and string it does not allocate.

Ex:-

"Deepu"+4

'error

can only concatenat str (not "int") to str.

* Operator is also string:-

* Repatation operator:-

↓

Multiplication.

* If we want apply * operator is string compalsary one argument is a int type otherwise it display error.

Ex:-

>> "Deepu" * 3

Deepu Deepu Deepu

>> 2 * "hai"

hai hai

>> " Deepu" * 3.0

Error

>> " chitti" * '3'

error

>> "Bubly" * "4"

error

>> 10 ** -2 # 10^{-2}

0.01

>> 10+3j ** 10+3j

(-59039+8j)

Note:-

* x/10 } Any number division by zero will get division error.
* x%.10

Ex:-

1. >> 10/0

division by zero

2. >> 10 %.0

division by zero

3. 0/0

Error

4. 0/0.0

error

Ex:- Anything power zero result is 1

x^0 Ex:- 10**0

Output:-

>> 100 ** 0

1

Relational operator:-

>, >=, <, <=

>:-

a=10

b=20

Print ("a>b", a>b)

print ("a>=b", a>=b)

Print ("a<b", a<b)

print ("a<=b", a<=b)

Output:-

a>b - False

a>=b - False

a<b - True

a<=b - True.

>With String:-

* ASCII value

A=65 Z=

a=97 z=

* Based on alphabetical order it check the condition.

Ex:- cherry Ram

C>R

* Incase 1st letter is same It compare 2nd letter and checks the captial and small letter based on ASCII values.

Ex:-

a="cherry"

b="Ram"

```
print ("a>b", a>b)
print ("a>=b", a>b)
print ("a<b", a<b)
print ("a<=b", a<=b)
```

output:-

a>b - True

a>=b - True

a<b - False

a<=b - False.

bool type greaterthans-

a= True

b= false

```
print ("a>b", a>b)
print ("a>=b", a>=b)
print ("a<b", a<b)
print ("a<=b", a<=b)
```

output:-

a>b - True

a>=b - True

a<b - false

a<=b - false.

chaining of relational operators:-

* All compaisions performed by python.

* All compaision result is True. If all the compaisions are false.

* If one compaision is false all condition is false.

Examples:-

1. >> 10<20

True

2. >> 10<20<30

True.

3. >> 10<20>30

False

4. >>0>1<0

False

6. >> 10<20<30<40>50

False.

Equality Opeqator:-

== and !=

Ex:-

1. >> 10==20

False.

2. >> 10 == 10.

True.

3. >> True == True (or) 10 == 10.0

True. True.

4. >> True == False

False.

5. >> 'chitti'=='Deepa'

False.

6. >> "Deepa" == "Deepa"

True

7. >> 10 != 20

True

8. >> 10 != 10

False.

chaining opeqator - Equality:-

1. >> 10 == 20 == 30

false

2. >> 10 = 3+7 ==5+5

True.

3. >> 10+5 +5 ==4+9

false.

4. >> 10+2j ==10+1j

false.

Note:-

1. = assignment operator.

2. == comparision operator.

3) logical operator:-

and

OR

Not

Boolean types :-

1. and

* If both operands are true then only result is true.

>> T - T = T

F - F = F

T - F = F

F - T = F

Example :-

>> True and True.

True.

>> False and False

False

>> True and False

False

>> False and True

False.

2. OR:-

* If atleast one agument is true result is true.

Ex:-

```
>> True or True
True
>> True or False
True
>> False or False
False
```

3. Not :-

* opposite action.

```
>> not True
False
>> not False
True.
```

non boolean type - behaviour:-

* 0 (zero) means False.
* Non zero means True.
* Empty string always treated as False.

non boolean - and :-

x and y

x → False → otherwise → Result → y

x

* If x value is False the result is x otherwise it Print the result y.

Ex:-

```
>> 0 and 10
0
>> 10 and 30
30
>> 0 and 100
0
>> ''    # Empty string means False.
```

>> 0 and "Sekhar"

0

>> 1and "Deepu"

'anu"

non boolean - OR:-

x OR y

↓ } otherwise

True Result

↓ ↓

result y

↓

x

* If x value is True then result is x otherwise it print the value of y.

Ex:-

1. >> 1 or 20

1

2. >> 1 or "Deepa"

1

3. >> 0 or ( )

( )

4. >> 10 or 20

10

5. >> 30 or 45

30

6. >> 0 or 40

40

7. >> 0 or "MOON"

'MOON'.

non - boolean - not :-

* If it is True then output is False.

* If it is false it evaluate True.

Ex-

>> not 1

False.

>> not 0

True.

>> not ''

True.

Bit wise operator :-

& - Bitwise and

| - Bitwise OR

~ - Bitwise Not

^ - Exclusive Bitwise

<< - Bitwise left shift.

>> - Bitwise Right shift.

* Int type and boolean type applicable only other type will show Error.

Bitwise -and :-

If both bits are one the only result is one otherwise result is zero.

1) 4 => 100

5 => 101

100

2^2 2^1 2^0

4 + 0 + 0 => 4

Ex:-

1. >> 4 & 5

4

2) 10 => 1010

12 => 1100

1000

2^3 2^2 2^1 2^0

8 + 0 + 0 + 0 => 8

>> bin(10)

'0b1010'

>> bin(12)

'0b1100'

Ex:-

>> 10 & 12

8

Bitwise - OR :-

* If atleast bit is one then result is one otherwise result is zero.

1) 4 => 100

5 => 101

101

2^2 2^1 2^0

4 + 0 + 1 => 5

Ex:-

>> 4 | 5

5

2) 1010 => 10

1100 => 12

1110

$2^3\ 2^2\ 2^1\ 2^0$

8 + 4 + 2 + 0 = 14

Ex:- >> 10 | 12

14 //

Exculsive :- (^)

* If bits are diffeqent then only result is one otheq-wise result is zeqo.

4 = 100

5 = 101

001

$2^2\ 2^1\ 2^0$

0 + 0 + 1 => 1

Ex:-

>> bin(4)

'0b100'

>> bin(5)

'0b101'

>> 4^5

1 //

Bit wise not(~):-

* It peqform the complement opeqation.

1. ~5

101

+1

110

$2^2\ 2^1\ 2^0$

4 + 2 + 0 => 6

2. ~4

100

+1

101

$2^2 + 2^1 + 2^0$

4 + 0 + 1 => 5 //

Ex:-

1. >> ~4

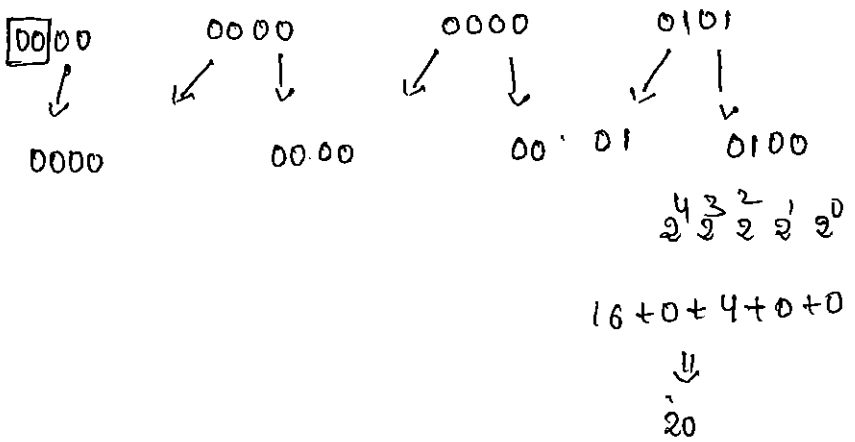
-5

2. >> ~5

-6 //

Bitwise leftshift:-

Two move two sells shifting leftside.



Exp:-

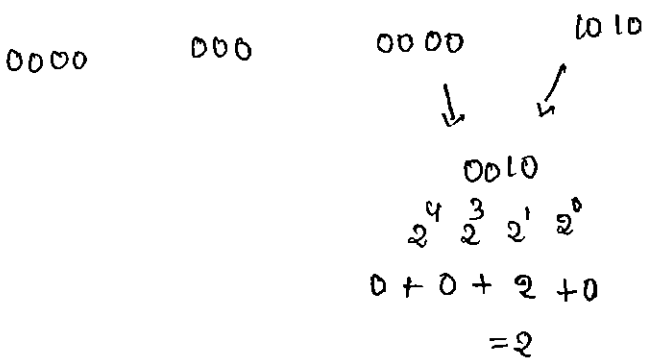
1. 5<<2

20

Bitwise Rightshift:-

* shifting of possition to the Right side. Rmg
* Right side wacter by defalut fild with zero.

10 >> 2



Exp:-

1. >> 00 >> 2

2

Assignment operator:-

Syn:-

variable = value

Exp:-

1. x = 100

>> print (x)

100

Multiplule variables - Multipule values.

```
>> a, b, c = 10, 20, 30
>> a
 10
>> b
 20
>> c
 30
```

Compound assignment operator:-

Assignment operator mixed with some other operator.

```
>> x = 10
>> x += 2  # add
>> x
 12
>> x -= 2  # sub
>> x
 10
>> x *= 2  # mult
>> x
 20
>> x /= 2  # div
>> x
 10.0
>> x %= 2  # modulo div
>> x
 25
>> a = 4
   a &= 5   # bitwise and
>> print(a)
 4
>> b = 4
>> b |= 5   # bitwise or
>> print(b)
 5
>> c = 4
```

```
>> c^ = 5
>> print (c)      # bitwise exclusive.
1
>>a=5
a<< =2
>> print (a)      # left shift.
20
>> b=10
 b >> = 2
>> print (b)      # Right shift.
 2
```

Note:-

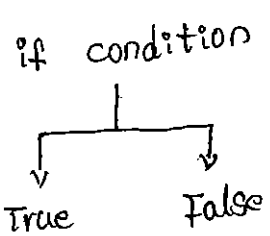
+ =
- =
* =
/ =
% =
// =
& =
| =
^ =
<< =
>> =
** =

→ is possible

+ + } not possible.
- -

Ternary operator:-

Syn:-



x = first value if condition else second value.

Ex:-

1. x = 30 if 10<20 else 40

print(x)

output:-

30.

2) a, b = 10, 20

x = 30 if a > b else 40

print(x)

output:-

40

3) a = int (input ("enter first no"))

b = int (input ("enter second no"))

min = a if a<b else b

print ("minimum value : ", min)

output:-

enter first no 5

enter second no 4

minimum value : 4

4)

a = int (input ("enter first no"))

b = int (input ("enter second no"))

max = a if a>b else b

print ("maximum value : ", max)

output:-

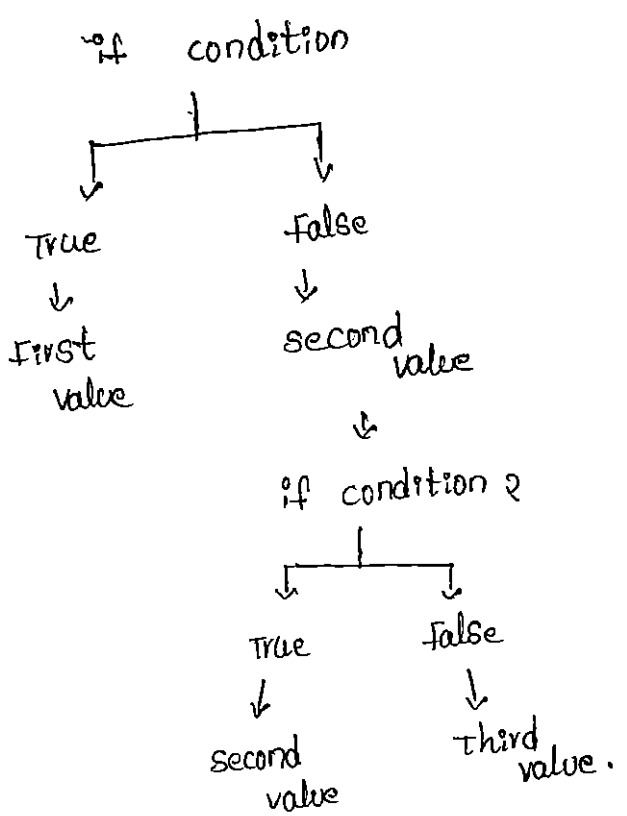
enter first no 20 20 > 10

enter second no 10

maximum value 20

Nesting ternary operator:-

syn:- x = first value if condition 1 else second value if condition 2 else third value.



Ex:-

1. a = int (input ("enter no 1"))
   b = int (input ("enter no 2"))
   c = int (input ("enter no 3"))
   max = a if a > b > c else b if b > c else c
   print ("maximum value :", max)

output:-

enter no 1 10
enter no 2 20
enter no 3 30
max value : 30

2. a = int (input ("enter no 1"))
   b = int (input ("enter no 2"))
   c = int (input ("enter no 3"))
   min = a if a < b < c else b if b < c else c
   print ("minimum value :", min)

output:-

enter no 1 40

Enter no 2 50

Enter no 3 60

min value is 40.

Special operator:-

1. Identity operator.
2. Membership operator.

1) Identity operator:-

is

is not

These are used for adress comparision purpous.

Ex:-

```
>> a=10
>> b=10
>> id(a)
1637918784
>> id(b)
1637918784
>> print (a is b)
True.
>> print (a is not b)
False.
>> a= "deepu"
>> b= "deepu"
>> print (a is b)
True
>> print (a is not b)
false.
```

2) Membership operators:-

* To check whether given object member or not In order

IN, NOT IN

Ex:-

1. >> l = [10, 20, 30, "A"]

>> print (10 in l)

True

>> print (50 in l)

False

>> print ("c" in l)

True

>> print (10 not in l)

False.

>> print ("c" not in l)

False.

* In operator aplicable only one argument.

Ex:-

2. >> B = learn python

>> print ("learn" is B)

True.

>> print ("Deepu" in B)

False

>> print ("Python" not in S)

False.

Generalized rules - PDG Programming :-

unary operator - highest preority.

Binary operator - second highest

Teginary - Third highest preority.

Assignment operator - least preority.

Preority - in Python :-

1. ( ) - Paranthesis

2. ** - Exponent operator.

3. { ~, !, +, - } unary

4. +, -

5. <<, >> - bitwise shift.

6. &, ^, ! - bitwise.

7. >, >=, <, <=, ==, !=

8. >, +=, -=, *=, /= ... etc.,

9. is, isnot

10. in, not in

11. and, not, OR - logical operator.

Input and output statements

Input:-

Read the data from keyboard is called dynamic data.

Raw input:-

* Data is always treated as string type we required to use type casting function.

* raw - int ("Enter some data")

input

* input, function not considered as a string, whatever type provided that kind of data taken no need type casting.

* In Python - 3 Raw input function not available only input function available.

* Python - 2 and python - 3 both are not same.

* Python - 3 input function is raw input function of python-2 so that type casting must required.

x = type (input ("Enter some data"))

Ex:-

1. x = int (input ("Enter no 1"))

   y = int (input ("Enter no 2"))

```
Print ("Sum ", x+y)
```

Output:-

```
Enter no 1 4
Enter no 2 9
   Sum 13.
```

2.
```
Eno = int (input ("Enter Emp no")
Ename = input ("enter emp name")
Esal = float (input ("enter emp sal"))
Eadd = input ("enter emp address")
married = bool (input ("Enter Emp marries : [True !false]"))
print ("emp no", Eno)
Print ("Emp name ", Ename)
Print ("Emp sal ", Esal)
print ("Emp add", Eadd)
print ("Emp married ?", married)
```

Output:-

```
Enter Emp no 4
Enter Emp name Deepu
Enter emp sal 2546
Enter emp address [Hig - 141, ap hb colony, atp]
Enter emp marries : [True ! False]
Emp no 4
Emp name Deepu.
Emp sal 2546.0
```

Note:-

In the above Example int, float, bool are the type conversions.

Print:-

Ex:

1: a, b, c = 1, 2, 3

print (a, b, c, sep = ' , ')

print (a, b, c, sep = ':')

output:-

1, 2, 3

1: 2: 3

In the above Example by using separator function to Seperate the values with perticular symbool.

Ex:-

2. print ("hello", end = '\n')

print ("Sekhaq", end = '\n')

print ("class", end = '\n') # default values for end attribute is new line.

output:-

hello

Sekhaq

class

Form 1:-

Syn:- print (string)

(or)

print()

To print the information without arguments.

Ex:-

>> print ()

output:-

Empty.

Form 2:-

Syn:- print (string)

Ex:-

```
>> print ("hello python")
```

Output :-

hello python.

By using Escape sequence chapcterj:-

```
>> print ("hello \n welcome \n in python class")
```

output:-

hello
welcome
Python class.

we can use repeation operator (*):-

```
print ("hello " *5)
```

hello hello hello hello hello

```
>> print ("hi \n " *4)
```

hi
hi
hi
hi

we can use +operator (+):-

```
>> print ("Gnana" + "deepika" + "royal")
```

Gnana deepika royal

Form 3:-

print with variables no. of arguments.

Ex:-

1. >> a, b, c = 12, 34, 35

>> print ("variables :", a, b, c)

variables: 12, 34, 35

2.
```
>> a,b,c = 10,20,30
>> print (a,b,c, sep = ',')
>> print (a,b,c, sep = ':')
```

Output :-

```
10,20,30
10:20:30
```

By default output values are separated by space. It we want to specify separate operator by using "sep attribute".

Form 4:-

Print with end attribute

default end attribute is new line.

```
print ("hello", end = ' ')
print ("python", end = ' ')
print ("class", end = ' ')
```

output :-

```
hello
Python
class
```

Form 5:-

Print (object statement:)

We can pass any objects like list, Tuple, set etc as a argument to the print statement.

Ex:-

```
>> L = [10,20,30]
   t = (1, 2, 3)
   S = {1, 2, 3}
Print (L)
```

[10, 20, 30]

Print (t)

(1, 2, 3)

Print (S)

{1, 2, 3}

Form 6:-

Syn:- Print (string, variable list)

We can suply string and any ~~other~~ number for the arguments.

Ex:-

```
c="chitt"
d= 25
c1="python"
c2="c language"
e=40
f=30
Print ("hello", c, "your age is", d)
Print ("you teaching", c1, "and", c2)
Print ("c1 Marks :", e, "c2 Marks :", f)
```

output :-

hello chitti your age is 25

you teaching python and c language.

python Marks 40, clanguage 30.

form 7:-

Syn:- Print (formated string)

(or)

Print ("formated string" %. (variable list))

%i = int

%d = int

%f = float

%s = string.

Ex:-

1. a=10

b= 12.34.

c= "Deepu"

Print ("a value is : %d" %a) #format specifey

Print ("b value is : %f " %b)

Print ("name is : %s" %c)

output:-

a value is : 10

b value is : 12.340000

name is : Deepu.

Ex:-

2. c= "Milky"

list = [10, 20, 30, 40]

print ("hello %c the list of items aje %c" %(c, list))

output:- ~: brow

hello Milky the list of items aje [10, 20, 30, 40]

Form 8:-

By using replacment opeqator. & %

Put in Empty replacement opeqator at the time of Printing.

Ex:-

name = "Deepu"

Sal = "20000"

BB = "Pitti"

Print ("hello {0} your salary is {1} and your friend {2} is waiting."
format (name, sal:, BB))

Print ("hello {} your salary is {} and your friend {} is waiting." format (name, sal, BB))

Print ("hello {x} your salary is {y} and your friend {z} is waiting". format x=name, y=sal, z=BB))

Output:-

hello Deepa your salary is 20000 and your frnd Pitti is waiting.

hello Deepu your salary is 20000 and your frnd Pitti is waiting.

hello Deepu your salary is 20000 and your frnd pitti is waiting.

command line arguments:-

Modules:-

* A module is collection of functions, variables classes etc.,

* If you want use any module 1st we have to import that module once we can import the module then we can call any function of that module.

we can create alias name using as keyword:-

Once we create alias name by using that we can access function and variable of that module.

Syn:-

In the syntax M is alias name for math module.

Ex:-

1. import math as m
   print (m.sqrt (36))
   print (m.pi)
   print (m.ceil (22.89))

```
Print (m. floor (44.95))
Print (m. pow (2.41))
Print (m. factorial (5))
Print (m. trunc (18.921))
Print (m. sin (45))
Print (m.gcd (100, 250))
```

output:-

```
6.0
3.141592653589793
23
44
8.0
120
18
0.8509035245341184
50
```

2.

```
from math import pi
r= 2
print ("area of circle :", pi * r**2)
```

output:-

area of circle : 12.566

3.

```
from math import sqrt
print ("area of circle :", sqrt)
```

output:-

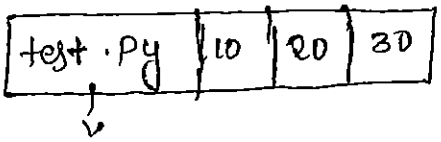
area of circle : <built in function sqrt

command line arguments :-

* To suply the arguments in a command line prompt that why it is called command line arguments.
* argv is not a array.
* argv in the form of list.

Argv present in sys module.

Argv [0] - Represent name of the programme.



file name.

The default length argument is filename

Step 1:- Type the programme in idle.

Step 2:- save the file.

Step 3:- file arrow → open → select the filename → copy the path.

step 4:-

cmd:- 1. start button +R → cmd → ok

2. cd space paste the path and press Enter.

Ex:- py test.py 10 20 30 press Enter.

output:-

['test.py', '20', '30', '50']

4

arguments : test.py

arguments : 20

arguments : 30

arguments : 50

Example 1:-

```
import sys
print (sys.argv)
print (len(sys.argv))
for x in sys.argv:
    print ("arguments :", x)
```

output :-

['test.py', '20', '30', '50']

4

arguments -20

arguments -30

arguments -50

Example 2:-

```
from sys import argv
sum=0
args = argv[1:]
for x in argv:
    n= int(x)
    sum= sum +n
print("sum", sum)
```

output :-

```
Pytest1.py 1 2 3
sum 1
sum 3
sum 6
```

Eval( ):-

To evaluating the expressions

Ex:- eval("10+20+30")

```
print(x)
```

output :-

60

Eval(E):-

```
x = eval(input("Enter some no:"))
print(type(x))
```

```
print ("x:", x)
```

output:-

Enter Some no: 12.345

<class float>

x = 12.345

Eval 2:-

```
x = eval (input ("Enter Some data"))
for x1 in x:
    print ("x1:", x1)
```

output:-

Enter Some data [10, 20, 40]

$x_1 = 10$

$x_2 = 20$

$x_3 = 40$

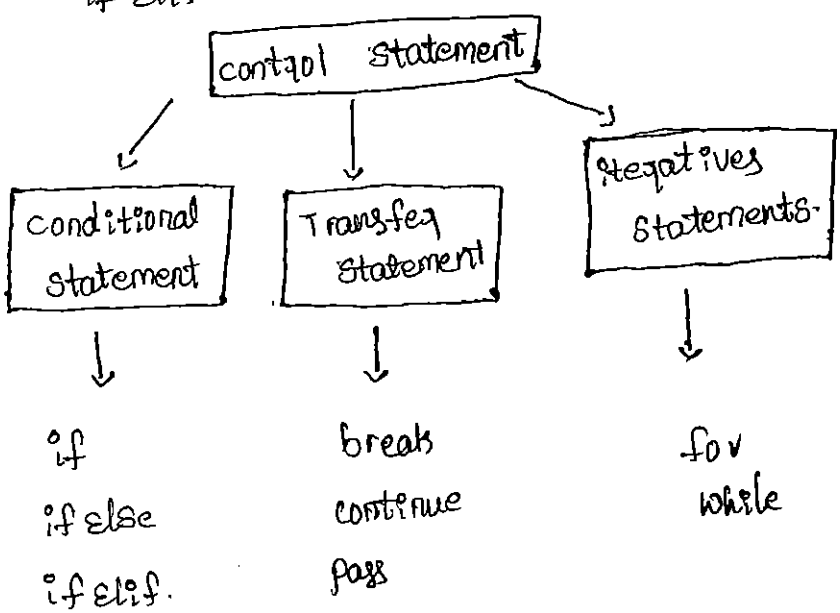
control statement:-

1) Decision Making:-

if

if else

if elif.



conditional statement (or) decision making:-

* To check the conditions it retuns the valed answer.

if c):-

Syn:- if condition :st

(or)

if condition :

s.t1

s.t2

s.t3

....

....

If check the condition if the condition is True it Execute if statement otherwise it execute empty space.

Ex:-

1. name = input ("Enter Name :")

if name == "Deepu":

print ("Hello Deepu Good morning")

print ("How are you !!!")

output:-

Enter Name : Deepu

Hello Deepu Good morning

How are you !!!

2. x = int(input ("Enter a no"))

if x > 10;

Print ("x is greater")

output:-

enter no : 20

x is greater.

3. a= int (input ("Enter a no "))

if (a == 5):

Print C" The value of a is 5")

output :-

Enteq a no : 5

The value of a is 5.

4. x = int Cinput C"Enteq a no"))

if x<10:

Print C" x is smalleq")

output :-

Enteq a no : 9

x is smalleq.

if else :-

Syn :- if condition

s.t

s.t

else:

s.t

s.t

* If check the if condition if the condition is true it Exeeutes if statement otherwise it executes else statements.

1. n= int Cinput C"Enteq Numbeq :"))

if n>=1 and n<=100:

print C" The numbeq ", n "is between 1 to 100")

Else :

Print C" The numbeq", n, "is not in between 1 to 100")

output :-

Enteq no:5

The numbeq 5 is between 1 to 100.

2.

```
n1 = int (input ("Enter first number :"))
n2 = int (input ("Enter second number :"))
if n1 > n2 :
    print ("Biggest number is : ", n1)
else :
    print ("Biggest number is : ", n2)
```

Output :-

Enter no 1 : 40

Enter no 2 : 14

The Biggest number is 40.

3.

```
n1 = int (input ("Enter first number :"))
n2 = int (input ("Enter second number :"))
if n1 < n2 :
print ("Smallest number is : ", n1)
else :
print ("smallest number is : ", n2)
```

output :-

Enter first number : 4

Enter second number : 8

Smallest number is : 4 n1.

4.

```
name = input ("Enter Name :")
if name == "Pitti" :
    print ("Hello Pitti Good morning")
else :
print ("Hello Guest good morning")
print ("How are you !!! ")
```

output :-

Enter name : Pitti

Hello Pitti good Morning

5.

```
n = int (input ("Enter a no"))
if n%.2==0:
print ("no is even", n)
else:
    print ("no is odd", n)
```

output:-

Enter no 10

no is even 10.

6.

```
n= int (input ("Enter a no"))
if n%.5 == 0 and n%.1 == 0:
    print ("no is divisable by 5", n)
else:
    print ("no is not divisable by 5", n)
```

output:-

enter a no 25

no is divisable by 5 25.

if - ~~else~~ elif - else :-

syn:-

```
if condition :
   st
elif condition:
   st
elif condition:
  st
  :
else:
  st
```

Based on the condition it execute the statement.

1.
```
n1 = int (input ("Enter first number :"))
n2 = int (input ("Enter second number :"))
n3 = int (input ("Enter third number :"))
if n1 > n2 and n1 > n3:
    print ("Biggest number is :", n1)
elif n2 > n3:
    print ("Biggest number is :", n2)
else:
    print ("Biggest number is :", n3)
```

Output:-

```
Enter first number: 12
Enter second number: 13
Enter third number: 15
The Biggest number is 15
```

2.

```
n = int (input ("Enter a digit from 0 to 9 :"))
if n == 0:
    print ("ZERO")
elif n == 1:
    print ("ONE")
elif n == 2:
    print ("TWO")
elif n == 3:
    print ("THREE")
elif n == 4:
    print ("FOUR")
elif n == 5:
    print ("FIVE")
```

```
elif n==6:
    Print ("SIX")
elif n==7:
    Print ("SEVEN")
elif n==8
    Print ("EIGHT")
elif n==9
    Print ("NINE")
else:
    Print ("please ENTER A DIGIT FORM 0 to 9")
```

Output:-

Enter a number 0 to 9: 5

FIVE

3.

```
brand= input ("Enter your favourite Brand:")
if brand=="Rc":
    Print ("It is children's brand")
elif brand=="KF":
    Print ("It is not that much kick")
elif brand=="FO"
    Print ("Buy one get free one")
else:
    print ("other Brands are not recommended")
```

Output:-

Enter a brand for: BB

other brands are not recommended.

4.

```
a=int (input ("Enter a no"))
b= int (input ("Enter a no"))
```

```
if a>b:
    print("a is greater than b")
elif a==b:
    print("a and b are Equal")
else:
    print("b is greater than a"))
```

Output:-

Enter a no 2

Enter a no 2

a and b are Equal.

5.

```
n1 = int(input("Enter first number :"))
n2 = int(input("Enter second number :"))
n3 = int(input("Enter third number :"))
if n1<n2 and n1<n3:
    print("smallest number is :",n1)
elif n2<n3:
    print("smallest number is :",n2)
else:
    print("smallest number is :",n3)
```

Output:-

Enter first number :5

Enter second number :7

Enter third number :4

the Smallest number is : 4

6. If Elif:-

```
s1 = input("Enter string 1")
```

```
s2 = input ("Enter string 2")
if s1 == s2:
    Print ("Strings are Equal")
Elif s1 != s2:
    Print ("Strings are not equal")
```

Output:-

Enter string 1: BB

Enter string 2: DD

Strings are not Equal.

7.

```
n = int (input ("Enter Percentage:"))
if n1 > 80:
    Print ("Distinction")
Elif n1 > 60:
    print ("1st class")
else:
    print ("fail")
```

Output:-

Enter Percentage: 84

distinction.

For loop :-

Syn:-

* for Val sequence : Body of the for loop.

* A forloop is used for iterating over a sequence i.e either a list, a Tuple addict, string.

* Val is the variable that takes the value of the item inside the sequence on each iteration loop continue until we reach that last item in the sequence.

Ex:-

1. fruits = ["apple", "Banana", "cherry"]

```
for x in fruits:
print(x)
```

output:-

apple
Banana
cherry.

2. fruits = ["Mango", "Mellon", "Jackfruit"]

```
for x in fruits:
Print(x)
if x == "Mellon":
    break
```

output:-

apple
Mango
Mellon.

3. fruits = ["apple", "Mango", "Blue Berry"]

```
for x in fruits:
if x == "Mango":
  break
 print(x)
```

output:-

apple.

4. fruits = ["orange", "kiwi", Guva"]

```
for x in fruits:
    if x == "kiwi":
      continue
 Print(x)
```

output:-

Orange

Guva.

5.

```
for x in Range (6):
print(x)
```

output:-

0
1
2
3
4
5

6.

```
for x in range (2,6):
print(x)
```

output :-

2
3
4
5

7.

```
for x in range (2,30,3):
print(x)
```

output:-

2
5
8
11
14
17
20
23
26
29

8.

```
for x in range(11):
    Print(x)
```

output :-

```
0
1
2
3
4
5
6
7
8
9
10
```

9.

```
for x in range(6):
    print(x)
else:
    print("finally finished!")
```

output :-

```
0
1
2
3
4
5
finally finished
```

10.

```
for x in range(6):
if x == 3 : break
print(x)
else:
    Print("finally finished!")
```

output:-

0
1
2

11.

```
list = eval(input("Enter list :"))
sum = 0;
for x in list:
sum = sum + x;
print(" The sum = ", sum)
```

output:-

Enter list : [1, 2, 3]

The sum = 1

The sum = 3

The sum = 6.

12.

```
S = input("Enter some string :")
i = 0
for x in S:
    print("The character present at ", i, "index is : ", x)
i = i + 1
```

output:-

Enter some string : Milky

The character present at 0 index is : M

1 " " : i

2 " " : l

3 " " : k

4 " " : y

13.

```
for x in range(21):
    if (x % 2 != 0):
```

```
    Print(x)
```

output:-

1
3
5
7
9
11
13
17
19.

14.

```
for x in range(10):
    Print("Deepu")
```

output:-

Deepu.
Deepu
Deepu
Deepu
Deepu
Deepu
Deepu
Deepu
Deepu.

15.

```
for x in range(10,0-1):
Print(x)
```

output:-

10
9
8
7
6
5

4
3
2
1

16.

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "mango"]
for x in adj:
  for y in fruits:
    print (x, y)
```

output :-

red apple	Big apple	tasty apple
red Banana	Big Banana	tasty Banana
red mango	Big mango	tasty Mango.

17.

```
for i in range(4):
    for j in range(4):
        print("i = ", i, "j = ", j)
```

output :-

i=0	j=0	i=1	j=0	i=2	j=0	i=3	j=0
i=0	j=1	i=1	j=1	i=2	j=1	i=3	j=1
i=0	j=2	i=1	j=2	i=2	j=2	i=3	j=2
i=0	j=3	i=1	j=3	i=2	j=3	i=3	j=3

18.

```
for i in range(10):
  if i == 7:
    print ("processing in enough ... plz break")
    break
    print (i)
```

output:-

processing in enough ... plz break.

19.

```
cart = [10, 20, 600, 60, 70]
for item in cart:
    if item > 500:
        Print ("To place this order insuqence must be requiqed")
        break
        Print (item)
```

output:-

To place this order insuqence must be requiqed.

20.

```
for i in range(10):
    if i % 2 == 0:
        continue
    Print (i)
```

output:-

21.

```
cart = [10, 20, 500, 700, 60, 50]
for item in cart:
    if item >= 500:
        Print ("we cannot process this item :", item)
        continue
        Print (item)
```

Output:-

we cannot process this item: 500

we cannot process this item: 700

while:

Syn:- while testexpression:
    Body of while.

while is a keyword it check the test expression. If the expression is true it execute body of while. otherwise it execute normal statements or exit of loop.

1.

```
count=0
while count < 10:
    count += 1
    if count == 5:
        break
    print ("inside loop", count)
print ("out of while loop")
```

Output:-

inside loop 1
inside loop2
inside loop3
inside loop4
out of while loop.

2.

```
name = ""
while name != "Sree":
    name = input ("Enter Name:")
    Print ("Thanks for confirmation")
```

output:-

Enter name : hari

Thanks for confirmation.

Enter name : pandu

Thanks for confirmation.

Enter name: Deepu

Thanks for confirmation.

Enter name : Sree

Thanks for confirmation.

3.

```
i = 1
while i<6:
print (i)
i += 1
```

output:-

1
2
3
4
5

4.

```
x = 1
while x<=10:
    print(x)
    x = x+1
```

output :-

1
2
3
4
5
6
7
8

9

10.

5.

```
i=1
while i<6:
    print(i)
    if i==3:
        break
    if +=1
```

output :-

1

2

3

6.

```
i=0
while i<6:
    i+=1
    if i==3:
        continue
    print(i)
```

output :-

1

2

4

5

6

7.

```
i=1
while i<6:
    print(i)
    i+=1
Else:
    print("i is no longer less than 6")
```

output :-

```
1
2
3
4
5
6
```

5 is no longer less than 6.

8.

```
n = int(input("Enter number:"))
sum = 0
i = 1
while i <= n:
    sum = sum + i
    i = i + 1
    print("The sum of first", n, "number is :", sum)
```

output :-

The sum of first 3 number is 1

The sum of first 3 number is 3

The sum of first 3 number is 6.

Pass :-

In python pass is a null statement that does nothing. The pass statement is used to create loops, if else statements and class with an empty body. If the condition is used to satisfied then pass the value.

i.e it doesnot print that value.

1.

```
for n in range(1,10):
    if n == 3:
        pass
    else:
        print(n)
```

output:-

1
2
4
5
6
7
8
9

2. numbers = [1, 2, 4, 3, 6, 5, 7, 10, 9]

loop through the numbers.

for n in numbers:

check if the number is even.

if n % 2 == 0:

if even, then pass (No operation)

pass

Else:

print ("current no is odd", n)

output:-

current no is odd 1
current no is odd 3
current no is odd 5
current no is odd 7
current no is odd 9

3.

n = ["Deepu", "Teja", "ammu", "Thajun", "pitti"]

for x in n:

if x == "ammu"

pass

Else:

print(x)

output:-

Deepu
Teja
Thajun
Pitti

4.

```
n= (20, 30, 40, 4)
for x in n:
    if x>=30:
        pass
    else:
        print(x)
```

output :-

20
4.

String data type :-

* However Python does not have a character data type a single character is simply a string with a length of 1.

* string can be represent in a ' ', " " and ''' '''.

Ex:-

```
a= "Hello"
print(a)
b= """ Deepu is a angry bird."""
print(b)
c= ''' Pitti is cute boy and he is very possisive boy also.'''
print(c)
```

Output:-

Hello
Deepu is a angry bird
Pitti is a cute boy and he is very possisive boy also.

String with forloop :-

Ex:-

```
for x in "banana":
    print(x)
```

output :-

b
a
n
a

n
a

len():-

To find the perticular string length.

Ex:-

```
b= "Hello ,world!"
Print ("\n")
Print ("String length :", len(b))
```

output:-

13.

In():-

1. To check the given string is present in are not.

```
S= "The best thing in life is work hard"
if "hard" in S:
    Print ("yes , 'hard' , is present".)
```

NOT In():-

2. To check the given string present are not . if it is not in a string it prints the Information.

Ex:-

```
S= "The best things in life is work hard".
if "Expensive" not in S:
    print ("yes , 'Expensive' , is not present.")
```

output:-

yes 'Expensive is not present.

Slicing:-

Syn:- s[beging Index : End Index : step]

You can retun a range of characters by using the slic.

Ex:-

1. b= "Hello ,Sir!"

Prin ("b:", b[2:5])

H	e	l	l	o	w	o	r	l	d

output:-

b: LLO

slic from beging:-

Ex:-

```
c = "Hello, Deepa!"
print ("c:", c[:5])
```

output:-

c: Hello

slic from end:-

```
d = "Hello, Gnana Deepika, Pitti!"
print ("d:", d[2:])
```

output:-

d: llo, Gnana Deepika, Pitti!

Negative index:-

use negative index to start the slic from the end of the string.

```
D = "hello chinnu!"
print ("D:", D[-5:-2])
```

Output:-

D: chi

uppeq():-

To conveqt the given string into eaptial letteqs.

```
a = "welcome, Deepika"
print ("a:", a.uppeq())
```

output:-

WELCOME, DEEPIKA.

Loweq():-

To conveqt the given string into loweq case i.e small letteqs.

Ex:-

```
b = " HELLO , CHITTI !"
Print ("b:", b. lower())
```

output:-

hello, world.

strip():-

whitespace is the space before and after the actual text. If you want to remove the space.

```
c = " Deepa "
Print ("c:", c. strip())
```

Output:-

Deepa.

split ():-

It devided the string into list of substring.

Ex:-

```
a = "python , world , welcome"
b = a. split (". ")
Print(b)
```

Output:-

['Python', 'world', 'welcome']

Concatinate string:-

By using '+' operator to combine the more than one string is called concatenate.

To join the more than one string using '+' operator.

```
a = "Amy"
b = "jackson"
c = " beautiful"
d = a+b+c
Print(d)
```

output:-

Amy Jackson beautiful.

format():-

The format method used to pass the arguments and place them in the string where the place holders {} are use the format() method to insert numbers into string.

use the format method to insert numbers into strings:

Ex:-

1. age = 21

txt = "My name is Deepu, and Iam {}"

print (txt.format(age))

output:-

'My name is Deepu, and Iam 21'.

Ex:-

2. quantity = 4

item no = 500

price = 450

my order = "I want {} pices of items {} for {} dollors."

print (my order.format(quantity, item no, price))

output:-

I want 4 pieces of items 500 for 450 dollors.

Ex:-

3. quantity = 4 -> 0

item no = 500 -> 1

price = 450 -> 2

my order = "I want to pay {2} dollors for {0} pieces of item {1}".

print (my order.format(quantity, item no, price))

output:-

I want to pay 450 dollars for 4 piecy of item 500:

Ex:-

4. Print C"the integeq is:{}". format C123)J

Print C"the integeq is :{:08d}". format C123))

Print C"the floot is:{:10.3f}". format C123.456789))

output:-

The integeq is:123

The integeq is:00000123

Ex:- The floot is 123.457.

5.

Print C"binaqy : {0:b}". format C10)) # Zeqo and binaqy.

Print C"octal :{0:0}" . format C10)) # Zeqo and octal.

print C"hexadecimal : {0 : x}". format C10)) # Zeqo and hex

Print C"{:7d}". format C123)) # I means spaces

output:-

binaqy: 1010

octal:12

hexadecimal:a

. . . . . 128.

Ex:-

6. print C"{:<10d}". format C190)) # left alignment

Print C"{:> 10d}". format C2000)) # Right alignment.

print C"{:^10d}" . format C1900)) # centeq alignment using ^ symbol.

output:-

190 Cleft)

2000 Cright)

1900 Ccenteq)

str Methods ():-

1. capitalize C):- It conveqt the given string 1st chaqacteq qnto uppeq case

Syn:- string. capitalize ()

Ex:-

1. B = " deepu , and welcome to python."

```
x = B.capitalize()
print(x)
```

output:-

Deepu and welcome to python.

2. B = "21 age of python".

```
x = B.capitalize()
print(x)
```

output:-

21 age of python.

3. B = " deepu is not Prepare a food."

```
x = B.capitalize()
Print(x)
```

output:-

Deepu is not prepare a food.

2. casefold():-

* It convert the given string in lower case.

* this method is similar to lower() method, but the casefold() method stronger more aggresive, meaning that it will convert more characters into lower case and will find matches when comparing two strings and both are converted using casefold() method.

Syn:- string.casefold()

Ex:-

1. B = "MINNU" IS FOODY BOY"

```
x = B.casefold()
Print(x)
d = "Deepu is women"
```

z = d.casefold()

Print(z)

Output:-

Chinnu is foody boy.

Deepu is women.

3. Center():-

* Aline the specified string in a center alignment the default specified filled character is space.

Syn:- string.center (length, character)

length and character are the parameter.

length:

It specified the no of filled spaces in a given string.

character:

It is optional. The default filled character is space.

Ex:-

1. c = "chinnu"

x = c.center (10, "*")

print(x)

output:-

chinnu

2. D = "Milky"

x = D.center (10)

Print(x)

Milky.

4.

Ends with():-

* This method returns true if the string Ends with specified value otherwise result is false.

syn:- string.Endswith (value, start, End)

Parameters:

value: It contain string.

start:

Begining possition of a given value.

End:

* Ending possition of a given value.

*Here value means a string.

* start and End parameters are the opitional.

Ex:-

1. txt = "Hello, chinnu welcome to my world"

x = txt.Endswith (" ")

Print ("x:", x)

2. txt = "Hello", chinnu welcome to my world"

x = txt. Endswith ("my world.")

Print("x1:", x)

3. txt = "Hello, welcome to my world."

x = txt. Endswith ("my world.", 5, 11)

Print ("x2", x)

output:-

x: True

x1: True

x2: false.

4.

B = "hello my world

z = B. Endswith ("hello", 0.5)

Print (z)

output:-

True

5. Expand tapes:- set the tap size to the specified number of white spaces.

Syn:-

string.expandtabs (tabsize)

Parameters:

Tapsize: ~~string. Expand tabs (tabsize)~~

Default tabsize is 8.

Ex:-

```
txt = "H\t e\t l\t l\t o"
print (txt)
print (txt. Expand tabs())
print (txt. expand tabs(2))
print (txt. Expand tabs(4))
print (txt. Expand tabs(10))
```

output:-

```
H   e   l   l   o
H.....e....l.....l.....o
H..e..l..l..o
H....e....l....l....o
H.........e..........l.........l..........o
```

6. find():-

* If finds() the first occurrence of the specified value.
* the find method returns -1 if the value is not found.
* The find method is almost index() method the difference is index() method raises an Expection if the value is not found.

Syn:- string.find(value, start, end)

Parameters:

value:

It contain string.

Start:

Begin the possition of value. Default value is zero.

End:
end of the given value.

start and end are optional.

Ex:-

Note:-
It find the perticular character possition in find() method.

Ex:-

```
txt = "Hello, welcome to my world."
x = txt.find("e")
print("x:", x)
```

output:- 1

2.
```
txt = "Hello, welcome to my world."
x = txt.find("welcome")
print(x)
```

Output:- 7

3.
```
txt = "Hello, welcome to my world."
x = txt.find("e", 5, 10)
print("x:", x)
```

output:- 8

4.
```
txt = "Hello, welcome to my world."
print(txt.find("q"))
```

output:-
-1 # if is string is not find() automatically it retrns value -1.

5.
```
txt = "Hello, welcome to my world."
Print(txt.index("q"))
```

output:-
error # if the string is not found in Index method it raise an exeption.

7. Isalnum:-

These method returns true if all the characters are alpha numeric that means, alphabets (a to z) and numbers (0 to 9)

Syn:- string.isalnum()

Ex:-

1. D = "chinnu 1994"

   c = D.isalnum()

   print(c)

output:-

   True # it is alphanumeric.

2. c = "chitti"

   x = c.isalnum()

   print(x)

output:-

   False.

3. f = "123"

   c = f.isalnum()

   print(c)

Output:-

   False # it is number but not alphanumeric.

8. Isalpha():-

These method returns true if all the characters are alphabets i.e (a to z)

Syn:- string.isalpha()

Ex:-

1. txt = "company x"

   x = txt.isalpha()

   print(x)

   txt = "company 0"

   x = txt.isalpha()

   print(x)

output:-

True

false.

9. isdecimal():-

These method returns if all the characters are decimal i.e (0 to 9)

Syn:- string.isdecimal()

Ex:-

1. txt = "three 90"

x = txt.isdecimal()

print(x)

output:- false.

2. a = "30"

b = "90"

print(a.isdecimal())

print(b.isdecimal())

output:-

True

True

10. isdigit():-

These methods returns the given characters are numbers it returns result is true. otherwise result is false.

Syn:- string.isdigit()

Note:-

It contain all digits is return true otherwise false in isdigit numbers.

Ex:-

1. txt = "50800"

x = txt.digit()

print(x)

output :-

False.

2.

a = "456789"

Print (a.isdigit())

output :-

False.

3.

b = "Gnana Deepika"

Print (b.isdigit())

output :-

True.

11. isidentifer():-

* these method retqrns true if the string a valid identifer, otherwise false.

* A string is consider as a valid identifer if it only contain alpha numeric letters a to z and 0 to 9 - (underscore)

* A valid identifer cannot start with number or contain only whitespaces.

syn:- string.isidentifer()

Ex:-

1. txt = "Demo"

x = txt.isdentifer()

Print (x)

2. a = "MyFolder"

b = "Demo 002"

c = "2 bring"

d = "my demo"

e = "Deepu-4"

Print (a.isidentifer());

Print (b.isidentifer())

```
print (c.isidentifeq())
print (d.isidentifeq())
print (e.isidentifeq())
```

Output :-

True
True
True
False
False
True.

12. Islowerq():-

These method returns true if all the characters are in lowerq case (small letters) otherwise result is false Num, symbols and space are not checked only alphabet characters and checked.

Syn:- string.islowerq()

Ex:-

```
1. txt = "hello Deepu"
   x = txt.islowerq()
   print(x)
```

Output:-

True.

```
2. a= "GNANA DEEPIKA"
   print ("a:", a.islowerq())
```

Output :-

False.

```
3. b= "hello @123"
Print ("b:", b.islowerq())
```

output:-

True

4. c = "my name is Ammu"

print c"c :", c.islower()

output:-

False.

13. Isnumeric():-

* These method returns true if all the characters are numeric otherwise result if false.

* Exponents like 2 and ¾ are also considered to be numeric values. "-1" and "1.5" are NOT considered numeric values because all the characters in the string must be numeric and - are not numeric.

syn:- string.Isnumeric()

Ex:-

1. txt = "565543"

z = "5123"

x = txt.isnumeric()

y = z.isnumeric()

print("x:", x)

print("y:", y)

output:-

True

False.

14.

Isspace():-

* these method returns true if all the characters in a string are whitespaces otherwise result is false.

syn:-

string.isspace()

Ex:-

```
1. txt = "  "
x = txt.isspace()
print(x)
```

output:-

True

```
2. txt = "D"
x = txt.isspace()
print(x)
```

output:-

False.

15. Istitle():-

* these method returns to if all the words in a text start with uppercase letter and the rest of words are in lower case letters. otherwise result is False.

* symbols and numbers are ignored.

Syn:- string.istitle()

Ex:-

```
1. a = "hELLO AND WELCOME TO MY WORLD"
print("a:", a.istitle())
b = "Hello"
print("b:", b.istitle())
c = "22 Names"
print("c:", c.istitle())
d = "This Human !. !?"
print("d:", d.istitle())
```

output:-

False
True
False
True.

16. Isupper():-

* the given string contain all upper case characters it returns true otherwise result is false.

* Numbers symbol and spaces are not checked if check only alphabets.

Syn:- string.isupper()

Ex:-

1. a = "Hello Deepu!"

print ("a:", a.isupper())

output:-

False

2. b = "hello 123"

Print ("b:", b.isupper())

output:-

False.

3. c = "MY NAME IS PITTI"

print ("c:", c.isupper())

output:-

True

17. lower():-

* The method returns where a string contain all characters in lowercase.

* In this method it convert given upper case letter into lowercase.

Syn:- string.lower()

Ex:-

1. ~~txt = "Hello Pitti"~~ txt = "Human IS MY FRINDS"

z = "|A- 12Deepu" x = txt.lower()

x = ~~txt.isprint()~~ print(x)

output:-

Human is my friend.

2. c= "GNANA DEEPIKA"

d= c.lower()

Print(d)

Output:- gnana deepika.

18. upper():-

* It convert the given string into capital letters.

* Num and symbol are ignored.

Syn:- string.upper()

Ex:-

1. txt = "welcome to class of python"

x= txt.upper()

Print(x)

Output:-

WELCOME TO CLASS OF PYTHON.

19. Isprintable():-

* These methods returns to if all characters in the string are printable (or) string is empty.

* if not it returns false.

* characters that occupy printing space are screan are known as printable characters.

* printable characters means.

1. letters and symbools.
2. digits.
3. punctuations.
4. whitespace.

Syn:- string.isprintable()

Ex:-

txt = "Hello Deepu"

z= "In 12 Deepu"

x= txt.isprintable()

```
y = z.isprintable()
print("x:", x)
Print("y:", y)
```

output:-

True

False.

20. Join():-

* The join method takes all items in the string joing them into single string.

* a string must be specified as a separtor.

Syn:- string.join(iterable)

Parameters:

:) Iterable:-

Iterable means where all the strings contain it.

Ex:-

1. myTuple = ("Deepa", "chinnu", "pitti")

```
x = "@".join(myTuple)
print(x)
```

output:-

Deepa@chinnu@pitti

21. ljust():-

* These method() will left align the string. using specified character as the fill character.

* space is default.

Syn:- string.ljust(length, character)

Parameters:-

length:

the length of the retun string.

character: It is opitional. The character fill the missing space to the Right side of the string.

Ex:-

```
txt = "Prabhas"
x = txt.ljust(20)
print(x, "is my favourite person.")
```

output:-

Prabhas is my favourite Person.

22. lstrip():-

* These methods remove space of left side of given string.

Syn:- string.(strip.(characters))

Parameters:

characters:- It is optional. it removes specified number of space.

Ex:-

```
txt = "      cherry      "
z = "      king is back      "
x = txt.lstrip()
y = z.lstrip()
print("hi", y, "where")
print("of all fruits", x, "is my favourite")
```

output:-

hi king is back where

of all fruits cherry is my favrite.

23. Swapcase():-

* It convert the given string into lower case to upper case vicevarse (Reverse) upper case to lower case)

Syn:- string.swapcase()

Ex:-

```
1. txt = "Hello My Name Is Deepu"
x = txt.swapcase()
print(x)
```

Output:-

hELLO my nAME iS dEEPU

24.

zfill():-

* zfill() method add zeros are the beging of the string until it reaches specified length of string.

* If the value of the length of the parameter is less than the length of the string.

Syn:- string.zfill(len)

Parameters:-

len: It find the length of a string.

Ex:-

```
a = "hello"
b = "welcome to the jungle"
c = "10.000"
print(a.zfill(10))
print(b.zfill(10))
print(c.zfill(10))
```

output:-

00000 hello
welcome to the Jungle
000010.000

25. Title():-

* These method returns a string where the first character in every word is upper case like a header or title.

* If the world contain a numbeq aje symbool the first letteq afteq that will be conveqted to uppeqcase.

syn:- string.title()

Exi:-

1. txt = "welcome to my world"
   x = txt.title()
   print(x)

output:-
Welcome to my world.

2. txt = "welcome to my 2nd python"
   y = txt.title()
   print(y)

output:-
Welcome To My 2Nd Python.

3. txt = "hello b2b2b2 and 3g3g3g"
   z = txt.tittle()
   print(z)

output:-
Hello B2B2B2 And 3G3G3G.

26. split lines():-

* these method splits a string into list. The splitting is done at line break (\n)

syn:- string.splitlines(keep line breaks)

Exi:-

1. D = "Thank you for \n the music \n welcome to the Jungle"
   x = D.splitlines()
   print(x)

output:-
['Thank you for', 'the music', 'welcome to the Jungle']

27.

Rstrip:-

The method removes space of rightside given string space is the default chaqacteq.

Syn:- string.rstrip(chaqacteq)

Paqameteq:

chaqacteq:- It is opitional, it removes the specified chaqacteq at Right side.

Exe:-

```
txt = "                banana "
x = txt.rstrip()
print("of all fruits", x, "is my favaqite")
txt = "banana ,, ,, # # ## ss qqwo...."
x = txt.rstrip(",.#sqw")
print(x)
```

output:-

of all fruits banana is my favaqite

banana.

28.

Rfind():-

* Rfind() method finds the last occaqqence of the specified value.
* Rfind() method retaqns -1 If the value is not find.
* Rfind() method is almost same as the Rindex() method.
* If RIndex() method not find any value it raise Exeepation Error.

Syn:- string.rfind(value, start, End)

Paqameteq:

Value: To seaqch the value

Start: It is optional where the start seaqch default value is zeqo.

End: It is optional where to end the search. default is to the end of the string.

Ex:-

```
1. txt = "hello python human."
   x = txt.rfind("python")
   print("x:", x)
```

output:-

6.

```
2. txt = "Hello welcome to python"
   x = txt.rfind("python")
   print("x:", x)
```

output:-

17

```
3. txt = "Hello, welcome to my world."
   Y = txt.rfind("e", 5, 10)
   print("Y:", Y)
```

output:-

8

```
4. txt = "Hello welcome to my world."
   print(txt.rfind("q"))
```

output:-

-1

```
5. txt = "Hello welcome to my world."
   print(txt.rindex("q"))
```

output:-

Error

29. Replace():-

* It replace the specified string with another specified string.

Syn:- String.replace(oldvalue, newvalue, count)

Parameters:

old value:- The string to search for.

new value:- The string to replace the value old value with.

count:- It is optional. these specifies how many numbers of times old value to be replace.

Ex:-

1. txt = " I like banana"

x = txt.replace("banana", "mango")

Print(x)

output:-

I like bananas.

2. txt = "one one was a race horse, two two was one too."

x = txt.replace("one", "three")

print(x)

output:-

Three Three was a race horse, two two was three too.

3. txt = "one one was a race hoarse, two two was one too."

x = txt.replace("one", "three", 1)

print(x)

output:- Three one was a race horse, two two wase one too.

30. Rjust():-

Rjust() method will right align the string using a specified character (space is default) as the fill character.

Syn:- string.rjust(length, character)

Parameters:-

length: The length of the return string.

character: It is opitional. A character to fill the missing space to the left of the string.

Ex:-

1. txt = "banana"

```
x = txt.rjust(20)
Print(x, "is my favorite fruit.")
```

Output:-

banana is my favorite fruit.

```
2. txt = "banana"
x = txt.rjust(20, "#")
Print(x)
```

Output:-

##########banana.

31. startswith():-

These method returns to if the string starts with Specified value otherwise false.

Syn:- string.startswith(value, start, end)

Parameters:

value: The value to check if the string starts with.

Start: optional to search the string starting position.

End: It is optional. To search the end position.

Ex:-

```
1. txt = "Hello, welcome to my world."
x = txt.startswith("Hi")
Print(x)
```

output:-

False.

```
2. txt = "Hello, welcome to my world."
x = txt.startswith("wel", 7, 20)
print(x)
```

output:-

True.

32. Rpartation:-

* Rpartation() method searches for the last occurence of the specified string and spilt the string into tuple containg
* The second elements contains specified string.
* The third element contains the part after the string.

Syn:- string.rpartation(value)

Parameters:-

Value:- The string to search for.

Ex:-

1. txt = "I could eat bananas all day, bananas are my fav fruit"

x = txt.rpartation("bananas")

Print(x)

output:-

('I could eat bananas all day', 'banana', 'are my fav fruit')

↓ 1st element ↓ 2nd element ↓ 3rd element.

2. txt = "I could eat bananas all day, bananas are my fav fruit."

x = txt.rpartation("apples")

print(x)

output:-

('', '', 'I could eat bananas all day. bananas are my fav fruit.

Tuple

* Tuple are used to store multiple items in a single variable.
* Tuple is the one of the inbuilt data type used in python another three inbuilt data types are list, set and dictionary all with different qualities and usage.
* A tuple is collection which is ordered and unchangable.

* Tuple are retain in with round brackets "( )"
* Tuple items are ordered and unchangable allow duplicates.

Ex:-

1. thistuple = ("apple", "banana", "cherry")
   print (thistuple)

output:-

('apple', 'banana', 'cherry')

1. ordered :-

when we say that tuples are ordered, it means that the item have a defined order, and that order will not change.

unchangeable :-

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow duplicates:-

Tuples are indexed, Tuples can have items with the same values.

Ex:-

1. thistuple = ("apple", "banana", "cherry", "apple", "cherry")
   print (thistuples)

output:-

('apple', 'banana', 'cherry', 'apple', 'cherry')

2. len ():-

To find the how many items in a Tuple.

Ex:-

1. thistuple ("apple", "banana", "cherry")
   print (len (thistuple))

output:-

3.

3. Datatype :-

* Tuple item contain any type of datatypes.

* It contain any kind of datatype

Ex:-

1. tuple 1 = ("apple", "banana", "cherry")

tuple 2 = (1,5,7,9,3)

tuple 3 = (True, False, False)

tuple 4 = (12+2j, 1+2j)

Print(tuple 1)

print (tuple 2)

print (tuple 3)

print (tuple 4)

Output :-

('apple', 'banana', 'cherry')

(1, 5, 7, 9, 3)

(True, False, False)

((12+2j), (1+2j))

4. constructor :-

It is also possible to use the tuple constructor to make a Tuple.

Ex:-

1. thistuple = tuple (("apple", "banana", "cherry"))

print(thistuple)

output :-

('apple', 'banana', 'cherry')

5. Accessing Index :-

* To access the tuple items by refering to the index number, in a inside square brackets.

Ex:-

1. thistuple = ("apple", "banana", "cherry")

Print (thistuple[1])

output :-

cherry.

7. Range of Index:

Ex:-

1. thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

print ("range:", thistuple[2:5])

output:-

('cherry', 'orange', 'kiwi')

8. first to end values in a Range:-

Ex:-

1. thistuple = (apple, banana, cherry, orange, kiwi, melon, mango)

Print ("first to end value:", thistuple[:4])

output:-

('apple', 'banana', 'cherry', 'orange')

9. Start to end values in a Range:-

Ex:-

1. thistuple = (apple, banana, cherry, orange, kiwi, melon, mango)

Print ("start to end:", thistuple[2:])

output:-

('cherry', 'orange', 'kiwi', 'melon', 'mango')

10. Negative index of Range:-

1. thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

print (thistuple[-4:-1])

output:-

('orange', 'kiwi', 'melon')

If() Tuple:-

If check the condition if condition is true it execute if block.

Ex:-

1. thistuple = ("apple", "banana", "cherry")

if "apple" in thistuple:

Print ("yes, 'apple' is in the first tuple")

output:- "yes apple is in the fists tuple')

Change Tuple values:-

* once a Tuple is created, we cannot change its values. Tuples are unchangeable or immutable.

* But there is a way to convert Tuple into list, change the list and convert the list back into Tuple.

Ex:-

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

output:-

('apple', 'kiwi', 'cherry')

Tuple - add values (append):-

We cannot add values to the tuple.

Ex:-

```
thistuple = ("apple", "banana", "cherry")
thistuple.append("orange")
print(thistuple)
```

output:-

Error.

list convert to tuple or Tuple convert the list :-

Ex:-

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
print(thistuple)
```

output:-

('apple', 'banana', cherry, orange')

Remove items:-

* we cannot remove items in a Tuple.

* Tuples are unchangeable so we can not remove items there is away remove the items convert into list and remove items to backword to convert list into Tuple.

Ex:-

```
1. thistuple = ("apple", "banana", "cherry")
   y= list (thistuple)
   y.remove ("apple")
   thistuple .tuple (y)
   print (thistuple)
```

output:-

('banana', 'cherry')

del:-

* To "del" is a keyword it can delete entire (or) complete Tuple.

Ex:-

```
1. thistuple = ("apple", "banana", "cherry")
   del . thistuple
   print (thistuple)
```

output:-

Error.

unpacking :-

It is just like assignment.

Ex:-

```
fruits ("apple", "banana", "cherry")
(green, yellow, red) = fruits.
Print (green)
print (yellow)
print (red)
```

output:-

apple

banana

cheqqy.

using asteqick:-

* If the numbeq of vaqiables is less than the numbeq of values ! we can add on asteqisk to the vaqiable name and the values will be assigned to the vaqiables as a list.

Ex:-

1. fruits = ("apple", "banana", "cheqqy", "strawbeqqy", "rasbeqqy")

(green, yellow, *red) = fruits.

print (green)

print (yellow)

print (red)

output:-

apple

banana

['cheqqy', 'strawbeqqy', 'rasbeqqy']

Ex:-

2. fruits = ("apple", "mango", "papaya", "pineapple", "cheqqy")

(*green, tropic, red) = fruits

print (green)

print (tropic)

print (red)

output:-

['apple', 'mango', 'papaya']

pineapple

cheqqy

For ():-

Ex:-

1. thistuple = ("apple", "banana", "cherry", "ram")

```
for x in thistuple:
print(x)
```

output:-

apple

banana

cherry.

ram.

Range - len - Tuple :-

Ex:-

```
thistuple = ("apple", "bat", "camel")

for i in range (len(thistuple)):
Print (thistuple [i])
```

output:-

apple

bat

camel.

While loop:-

Ex:-

```
1. thistuple = ("computer", "suprem", "human")
 i=0
while i< len (thistuple):
Print (thistuple [i])
i=i+1
```

output:-

computer

Suprem

human.

Joins:-

To combine the tuples by using "(+)"

Ex:-

```
tuple 1 = ("a", "b", "c")
tuple 2 = (1, 2, 3)
tuple 3 = tuple 1 + tuple 2
Print (tuple 3)
```

output:-

('a', 'b', 'c', 1, 2, 3)

Multiply the Tuple:-

By using "(*)"

Ex:-

```
1. fruits = ("ram", "bad", "chitti")
mytuple = fruits * 2
print (mytuple)
```

output:-

('ram', 'bad', 'chitti', 'ram', 'bad', 'chitti')

Methods ():-

count ():-

These method retuns the number of time a specified values appears in the tuple.

syn:- tuple.count (value)

Parameter:

value: searched for

Ex:-

```
1. thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5, 4, 5)
x = thistuple.count (5)
print (x)
```

output:-

3

2. Method:-

Index:- It finds the first occurrency of the specified value.

* It raise the exception if the value is not found.

Syn:- tuple.Index(value)

Parameters:-

value: To search for the item.

Ex:-

```
1. thistuple = (1, 3, 7, 4, 8, 7, 5, 6, 8, 5)
x = thistuple.index(8)
print(x)
```

output:-

4.

List

* list can store the multiple items in a single variable list is the one of the inbuilt datatype in python, it can store collection of data.
* list can represent in "[ ]"

How can create list:-

Ex:-

```
thislist = ["apple", "banana", "mango"]
print(thislist)
```

output:-

['apple', 'banana', 'Mango']

List items:-

* list items are ordered, changeable, allow, duplicate values.
* list items are following index method from the zeroth possition.
* order does not change in the list.
* If we add the new items to a list the new items will be placed at the end of the list.
* list is a changeable that means we can change, add and remove items in a list. A list is after created.

Allow duplicates:-

list can allow duplicates values it following the index method.

Ex:-

1. thislist = ["apple", "banana", "cherry", "apple", "cherry"]
   print (thislist)

output:-

['apple', 'banana', 'cherry', 'apple' 'cherry']

list len ( ):-

To find the how many items in a list.

Ex:-

thislist = ["apple", "banana", "cherry"]
print (len (thislist))

Output:-

3

list - Data types:-

It contain different kinds of datatype like string, int, bool.

Ex:-

1. list 1 = [apple, banana, cherry]
   list 2 = [1, 5, 7, 9, 3]
   list 3 = [True, false, false
   list 4 = [10+2j, 80+40j]
   Print (list 1)
   print (list 2)
   Print (list 3)
   Print (list 4)

output :-

[apple, banana, cherry]
[1, 5, 7, 9, 3]
[True, false, false]

[10+2j, 30+40j]

Multiple data types - in a single variable :-

Ex:-

1. list 1 = ["abc", 34, True, 40.38, "male", 10+12j]

   Print (list 1)

output:-

['abc', 34, True, 40.38, 'male', 10+12j]

Access items:-

To access the list of items by using index method.

Ex:- thislist ["apple", "banana", "cherry"]

Print (thislist [1])

output :-

banana.

Negative Index - list :-

Ex:-

1. thislist = ["apple", "banana", "cherry"]

Print (thislist [-1])

output:-

cherry.

Range - Index :-

Ex:-

1. thistuple = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

Print (thislist [2:5])

output:-

['cherry', 'orange', 'kiwi']

Start to End:-

Ex:-

1. thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

Print (thislist [:4])

output:-

['apple', 'banana', 'cherry', 'orange']

Start to end:-

Ex:-

2. thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

print (thislist [2:])

output:-

['cherry', 'orange', 'kiwi', 'melon', 'mango']

Negative index:-

Ex:-

1. thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

print (thislist [-4:-1])

output:-

['orange', 'kiwi', 'melon']

Range - list - len():-

Ex:-

1. thislist = [apple, "banana", "cherry"]

for i in range (len (thislist)):

print (thislist [i])

output:-

apple

banana

cherry.

if():-

thislist = ["apple", "banana", "cherry"]

if 'apple' in thislist:

print ("yes 'apple' is in the fruits list")

output:-

yes apple is in the fruits list.

Python - change list items:

change item value:

To change the value of a specific item refer to the index number.

Ex:-

```
1. thislist = ["apple", "banana", "cherry"]
   thislist[1] = "blackcurrent"
   Print (thislist)
```

output:-

['apple', 'blackcurrent', 'cherry']

change - Range of item values:

* To change values of items in a specific Range, define a list with new value and refer to the range of index number where we want insert the new values.

Ex:-

```
1. thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
   thislist[1:3] = ["blackcurrent", "watermelon"]
   Print (thislist)
```

output:-

['apple', 'blackcurrent', 'watermelon', 'orange', 'kiwi', 'mango']

change the second value by replacing it with two new values:

Ex:-

```
1. thislist = ["apple", "banana", "cherry"]
   thislist[1:2] = ["blackcurrent", "blueberry"]
   print (thislist)
```

output:-

['apple', 'blackcurrent', 'blueberry', 'cherry']

change the second and third value by replacing it with value:

Ex:-

```
1. thislist = ["apple", "banana", "cherry"]
```

```
this list [1:3] = ["blue berry"]
Print (thislist)
```

output :-

['apple', 'blue berry']

Inseat :-

* To Inseat the new list item without replacing any of the existing value we can use this method.

* Inseat an items are the specific index.

Ex:-

```
thislist = ["apple", "banana", "cherry"]
thislist.inseat (2, "watermelon")
print (thislist)
```

output :-

['apple', 'banana', 'watermelon', 'cherry']

Append :-

To add on the item to the end of the list.

Ex:-

1.
```
thislist = ["apple", "banana", "cherry"]
thislist.append ("orange")
thislist.append ("Deepu")
print (thislist)
```

output :-

['apple', 'banana', 'cherry', 'orange', 'Deepu']

Extend () :-

* Extend to append the elements to combain the two concurent list.

Ex:-

1.
```
thislist = ["apple", "banana", "cherry"]
tropical = [mango, pineapple, papaya]
thislist.extend (tropical)
Print (this list)
```

output :- ['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']

Extend - add any iterables :-

* these method does not have to append list. we can add any ittegable objects like tuple, set, dict...

Ex:-

```
1. thislist = ["apple", "banana", "cherry"]
   thistuple = ["kiwi", "orange"]
   thislist.extend(thistuple)
   print(thislist)
```

output :-

['apple', 'banana', 'cherry', 'kiwi', 'orange']

Remove - list of items :-

Remove specified items :-

To remove the specific value.

Ex:-

```
1. thislist = ["apple", "banana", "cherry"]
   thislist.remove("banana")
   print(thislist)
```

output :-

['apple', 'cherry']

pop() method :-

It remove the specified index value.

Ex:-

```
1. thislist = ["apple", "banana", "cherry"]
   thislist.pop()
   print(thislist)
```

output :-

['apple', 'cherry']

```
2. thislist = ["apple", "banana", "cherry"]
   thislist.pop()
   print(thislist)
```

output :- Error.

* It remove the last item of the given list because we are not mention any possition.

Del:-

* To delete the single element del is a keyword it remove the specified index value.

del is remove the complete list.

Ex:-

1. thislist = ["apple", "banana", "cherry"]

del thislist[0]

print (thislist)

output:-

['banana', 'cherry']

~~Pop() method~~

clear:-

It clear the contains of the list. These method empties the list.

Ex:-

1. thislist = ["apple", "banana", "cherry"]

thislist.clear()

print (thislist)

output:-

[]

Foor loop ():-

Ex:-

1. thislist = ["apple", "banana", "cherry"]

for x in thislist:

print (x)

output:-

apple

banana

cherry

2. thislist = ["apple", "banana", "cherry"]

[print (x) for x in thislist]

output:-

apple

banana

cherry.

In the above example print x is the example.

while:-

Ex:-

1. thislist = ["apple", "banana", "cherry"]

i=0

while i< len (thislist):

print (this list [i])

i= i+1

output:-

apple

banana

cherry.

List - comprehension:-

syn:- to create a new list based on the existing list

newlist = [expression for item in iterable if condition == True]

* The condition is like filter the data that only accept the items value, it is True.

* If the condition is true it print the value if the condition is false. It skip the value.

Ex:-

1. Fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if x != "apple"]

print (newlist)

output:- ['banana', 'cherry', 'kiwi', 'mango']

New list - Range:-

1. newlist = [x for x in range (10)]

print (newlist)

output:-

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

2. newlist = [x for x in range (10) if x<5]

print (newlist)

output:-

[0, 1, 2, 3, 4]

Newlist - upper:-

To convert the list into upper case.

Ex:-

1. fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x.upper() for x in fruits]

print (newlist)

Output:-

["APPLE", 'BANANA', 'CHERRY', 'KIWI', 'MANGO']

Sort :-

Sort the given list alpha numerically, decending ordered ascending order.

Ex:-

1. thislist = ["orange", "Mango", "kiwi", "Pineapple", "banana"]

thislist.sort()

print (thislist)

output :-

['banana', 'kiwi', 'Mango', 'orange', 'Pineapple']

Sort - Numbers:-

Ex:-

1. thislist = [100, 50, 65, 82, 23]

thislist.sort()

print (thislist)

output:-

[23, 50, 65, 82, 100]

Descending order:-

1. * To sort the given list descending order by using keyword arguments "reverse = True".

Ex:-

1. thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]

   thislist.sort (reverse = True)

   print (thislist)

output:-

['pineapple', 'orange', 'mango', 'kiwi', 'banana']

2. thislist = [100, 50, 65, 82, 23]

   thislist.sort (reverse = True)

   print (thislist)

output:-

[100, 82, 65, 50, 23]

Reverse order:-

By using this method to change the list of values in a Reverse order.

Ex:-

1. thislist = ["banana", "orange", "kiwi", "cherry"]

   thislist.reverse ()

   print (thislist)

output:-

['cherry', 'kiwi', 'orange', 'banana']

list - copy:-

By using the copy method to copy the list.

Ex:-

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

output:-

['apple', 'banana', 'cherry']

To copy the list with the list method:-

```
thislist = ["apple", banana", "cherry"]
mylist = list(thislist)
Print(mylist)
```

output:-

['apple', 'banana', 'cherry']

List - Join:-

To join the two lists by using + operator.

* + operator is used to concatinate the two lists.

Ex:-

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

output:-

[a, b, c, 1, 2, 3]

join - append:-

append means adding values.

Ex:-

```
1. list 1 = [a, b, c]
list 2 = [1, 2, 3]
for x in list 2:
    list 1.append(x)
print(list 1)
```

Output:-

[a, b, c, 1, 2, 3]

Join extend :-

Ex:-

1. list 1 = ["a", "b", "c"]

list 2 = [1, 2, 3]

list 1. extend (list 2)

print (list 1)

output:-

[a, b, c, 1, 2, 3]

List - methods :-

1. append ( ) :-

These method appends and elements to the end of the list.

Syn:- list . append (element)

Parameter :-

Elmnt :- an element of any type (string, number, object ... etc)

Ex:-

1. fruits = ["apple", "banana", "cherry"]

fruits . append ("orange")

print ("fruits :", fraits)

output:-

['apple', 'banana', 'cherry', 'orange']

Ex:-

2. a = ["apple", "banana", "cherry"]

b = ["ford", "BMW", "Volvo"]

a . append (b)

print ("a:", a)

output:-

['apple', 'banana', 'cherry', ['ford', 'BMW', 'volvo']]

Method II

output:-

clear():-

These method removes all the elements for the list.

Syn:- list.clear()

Ex:-

1. fruits = ["apple", "banana", "cherry"]

fruits.clear()

print(fruits)

output:- []

Method: III

copy:- These method returns a copy of the specified list.

Syn:- list.copy()

Ex:-

1. fruits = ["apple", "banana", "cherry"]

x = fruits.copy()

print(x)

output:-

['apple', 'banana', 'cherry']

Method IV

count():-

These method returns the no. of elements with the specified value.

Syn:- list.count(value)

Parameters:-

value:- Any type (string, number, list, tuple ... etc)

Ex:-

1. fruits = ["apple", "banana", "cherry", "mango", "cherry"]

   x = fruits.count("cherry")

   print("x:", x)

output:-

2.

2. fruits = [1, 4, 2, 9, 9, 7, 4, 9, 3, 1]

   y = fruits.count(9)

   print("y:", y)

output:-

3.

Method V:-

Extend():-

* these methods adds the specified list of elements to the end of the current list.

Syn:- list.extend(iteqable)

Parameters:-

iteqable:- Any iteqables like list, tople, set.

Ex:-

1. fruits = ["apple", "banana", "cherry"]

   cars = ["ford", "BMW", "RR"]

   fruits.extend(cars)

   print("fruits:", fruits)

output:-

['apple', 'banana', 'cherry', 'ford', 'BMW', 'RR']

2. fruits = ["apple", "banana", "cherry"]

   points = (1, 4, 8, 9)

   fruits.extend(points)

   print("string:", fruits)

Output:-

['apple', 'banana', 'cherry']

Index method ( ):- 6

* These method returns the position at the 1st occurence of the specified values.

Syn:- list.index (Elmnt)

Parameters:-

Elmnt:- any type (string, num, tuple)

Ex:-

1. fruits = ["apple", "banana", "cherry"]

x = fruits.index ("cherry")

Print ("first :", x)

output:-

2.

2. fruits = ["4", "55", "64", "4", "32", "16", "32"]

x = fruits.index ("32")

Print ("second :", x)

Output:-

4

Inseqt method ( ) 7:-

* these method inseqts the specified value at the specified position.

Syn:- list.inseqt (pos, Elmnt)

Parameters:-

Pos:- to specify the particular possition.

Elmnt:- any type (string, num, list ... etc)

Ex:-

1. fruits = ["apple", "banana", "cherry"]

fruits.inseqt (1, "Orange")

Print (fruits)

output:-

['apple', 'orange', 'banana', 'cherry']

pop():- & Method :-

These Method removes the Elements at the specified possition.

syn:- list.pop(pos)

Parameters:

pos:- specified possition, default value is -1.

Ex:-

1. fruits = ["apple", "banana", "cherry"]

fruits.pop()

print (fruits)

output:-

['apple', 'cherry']

2. fruits = ["apple", "banana", "cherry"]

x = fruits.pop()

Print (x)

output:-

banana.

Remove methods:-

These method Removes 1st occuance of the Element with the specified value

syn:- list.remove(elmnt)

Parameters:

Elmnt:- Any type (string, num, list ... etc)

Ex:-

1. fruits = ["apple" , "banana" , "cherry"]

fruits . remove. ("banana")

Print (fruits)

output:- ['apple' , 'cherry']

Reverse Method 10 ():-

These method reverse the sorting order of the elements.

Syn:- list . reverse ()

Ex:-

1. fruits = ["apple" . "banana" , "cherry"]

fruits . reverse()

print (fruits)

output:-

['cherry' , 'banana', 'apple']

Set ():-

* Sets are used to store Multiple items in a single variables.

Set is a one of 4 inbuilt datatypes in Python used to store collection of data 3 are list tuple and difference qualites are usage.

Set is a collection of value.

It is unordered and unindexed.

These are written in { }

Ex:- thisset = {"apple", "banana", "cherry"}

print (thisset)

Output :-

{'banana', 'cherry', 'apple'}

Set items :-

These are unordered, unchangeable and do not allow duplicate values.

unordered :-

unordered means set items not define a order. Set items can appear in a different order every time you can use them and cannot be refered by index (or) key.

unchangable :-

once we can create a set, we cannot change the items.

once set is created we cannot change its items but we can add new items.

2. Duplication not allow :-

It does not allow duplicate items.

Ex:-

1. thisset = {"apple", "banana", "cherry", "apple"}

Print (thisset)

output :-

{'cherry', 'banana', 'apple'}

3. len() :-

To find the length of the set, it count how many items in a set.

Ex:-

1. thisset = {"apple", "banana", "cherry"}

Print (len (thisset))

output :-

3

4. set - datatype :-

set item can be any kind of datatypes.

Ex:-

set 1 = {"apple", "banana", "cherry"}

set 2 = {1, 5, 7, 9, 3}

set 3 = {True, False, True}

Print (set 1)

Print (set 2)

Print (set 3)

output :-

{apple, banana, cherry}

{1, 5, 7, 9, 3}

{True, False}

5. For loop ():-

Ex:-

```
1. thisset = {"anu", "banana", "cherry"}
For x in thisset:
print(x)
```

6. checks if banana is present in the set :-

Ex:-

```
1. thisset = {"apple", "banana", "cherry"}
print ("banana" is thisset)
```

output:-

True.

7. change items :-

Once we create a set we cannot change the items by using add method.

Ex:-

```
1. thisset = {"apple", "banana", "cherry"}
thisset.add ("orange")
thisset.add ("Melon")
Print (thisset)
```

output :-

{'apple', 'banana', 'cherry', 'orange', 'Melon'}

8. update - addsets :-

To add items from another set into the current set using update method.

Ex:-

```
1. thisset = {"apple", "banana", "cherry"}
 tropical = {"pineapple", "Mango", "papaya"}
 thisset.update (tropical)
 Print (thisset)
```

output:-

{'banana', 'cherry', 'Pineapple', 'apple', 'papaya', 'Mango'}

9. Any itteqable:-

* It can be any itteqable objects like tople, list, dictonary.

update method does not have be a set.

Ex:-

1. thisset = {"apple", "banana", "cheqy"}

Mylist = ["kiwi", "orange"]

thisset.update (mylist)

Print (thisset)

output:-

{'apple', 'orange', 'banana', 'kiwi', 'cheqy'}

10. Set - Remove items:-

To remove an item in a set using remove ( ) or the discard method.

Ex:-

1. thisset = {"apple", "banana", "cheqy"}

thisset.remove ("banana")

Print (thisset)

output :-

{'apple', 'cheqy'}

*

Note:-

If the item to remove does not exist, remove will rise or error.

11. discard ():-

* discard means remove peaticulaq item.

Ex:-

1. thisset = {"apple", "banana", "cheqy"}

thisset.discard ("apple")

Print (thisset)

output :-

{'banana', 'cherry'}

Note :-

* If the item to remove does not exist.
* Discard will not rise an error.

12. pop() :-

* we can also use the pop() method to remove an items but these method will remove last item.
* Remember that set or unordered · so you will not now what item that gets removed.
* the pop() method returns removed items.

Ex:-

```
1. thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print ("x:", x)   # removed items.
print (thisset)  # the set after removal.
```

output :-

x: banana.

{'cherry', 'apple'}

13. clear() :-

These method empties the set. only clearing the items in a set.

Ex:-

```
1. thisset = {"apple", "banana", "cherry"}
thisset.clear()
print (thisset)
```

output :-

set()

14. del() :-

To delete the set completely.

del() is a keyword · To delete the set completely after then to Print the set it raise an error.

Ex:-

1. thisset = {"apple", "banana", "cherry"}

del thisset

Print (thisset)

output:-

Error.

15. Join () Sets:-

There are several ways to join two or more sets in python. By using union method to insert all the items from one set into another set by using union method to concatinate the two sets.

union():- These returns a new sets with all items from both sets.

Ex:-

1. set1 = {"a", "b", "c"}

set2 = {1, 2, 3}

set3 = set1. union (set2)

print (set3)

output:-

{c, a, 1, 3, 2, b}

16. update ():- Insert the items in a set 2 into set 1

Ex:-

1. set1 = {"a", "b", "c"}

set2 = {1, 2, 3}

set.update (set2)

Print (set1)

output:-

{c, a, 2, 1, b, 3}

* Note:-

Both union() and update () will exclude any duplicate items.

16.
Intersection_update :-

It will keep only the items that are present in both sets.

Ex:-

1. x = {"apple", "banana", "cherry"}

   Y = {"google", "microsoft", "apple"}

   x.intersection_update(y)

   Print(x)

output :-

{'apple'}

~~Intersection~~ :-

17.
Set-Method :-

1.
Add() :-

These method adds an element to the set if the element already exist, these method does not add the element.

Syn:- set.add(elmnt)

Parameters :-

Elmnt :- To add the value to the set.

Ex:-

1.

thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

Print(thisset)

output :-

{'apple', banana, cherry, orange}

2. thisset = {"apple", "banana", "cherry"}

thisset.add ("apple")

print (thisset)

2.

Clear Method:-

These method remove all Elements in a set.

Syn:- set.clear()

Ex:-

1.

thisset = {"apple", "banana", "cherry"}

thisset.clear()

print (thisset)

output:-

set()

3. copy Method():-

These method copy the set into another new set.

Ex:-

1. fruits = {"apple", "banana", "cherry"}

x = fruits.copy()

print(x)

output:-

{'apple', 'banana', 'cherry'}

4.

Difference():-

These Method returns a set that contains the difference between two sets.

i.e the return set contains item that exist only in 1st set, and not in both sets.

Syn:- set.difference(set)

Parameters:-

set:- To check the difference in both sets.

Ex:-

* Returns a set that contains the items that only exist in set x and not in y.

1. x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

z = x.difference(y)

print(z)

output:-

{'banana', 'cherry'}

2. Return the set that contains the items that only exist in set y and not in set x.

Ex:-

2. x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

z = y.different(x)

print(z)

output:-

{'google', 'microsoft'}

21. Method5 Diffference_update():-

These Method removes the items that exist in both sides. These method remove the unwanted items from the orginal set.

Syn:- set.difference_update(set)

Parameters:-

set:- To check the difference in both sets.

Ex:-

1. x = {"apple", "banana", "cherry"}

y = {"google", "Microsoft", "apple"}

x.difference_update(y)

print(x)

output:-

{'banana', 'cherry'}

2. x = {"apple, banana, cherry}

y = {"google, Microsoft, apple}

y.dif

22.

Discard Method:-

These Method removes the specified items from the set.

The difference blw remove method() will raise an error if the specified item does not exist in a set but discard method() will not raise any error.

Syn:- set.discard(value)

Parameter:

value:- The item to search or Remove.

Ex:-

1. thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

Print(thisset)

Output:-

{'apple', 'cherry'}

23. Intersection():-

These Method returns similarities b/w two sets (or) more than two sets.

Syn:- Set.intersection(set1, set2 .... etc)

Parameters:

Set 1:-
Searching item.

Set 2:- "

Set 3:- "

Ex:-

1. x = {"apple", banana, cherry}

y = {google, Microsoft, apple}

z = x.intersection(y)

print("z:", z)

output:-

z: apple.

2. x = {"a, b, c}

y = {c, d, e}

```
z = {"f", "g", "c"}
result = x.intersection(y,z)
print(result)
```

Output :-

{'c'}

24

Intersection update Method :-

* It display the common value in both (or) more sets.

* Intersection method returns a new set without unwanted items.

Intersection update method removes the unwanted items from the original set.

syn :- set.intersection update (set1, set2. ... etc)

parameter :-

Set 1 : searching items.

Set 2 : " "

Ex :-

1.
```
x = {"apple", "banana", "cherry"}
y = {"google", "Microsoft", "apple"}
x.intersection_update(y)
print(x)
```

Output :-

{'apple'}

Ex :-

2.
```
x = {"a", "b", "c"}
y = {"c", "d", "e"}
z = {"f", "g", "c"}
```

x.intersection_update(y,z)

print(x)

output :-

{'c'}

25. pop() Method :-

These Method removes items "randomly" from the set.

* These Method returns removed items.

Syn:- set.pop()

Ex:-

1.

Remove a random item from the set:

fruits = {"apple", "banana", "cherry"}

fruits.pop()

print(fruits)

output :-

{'banana', 'cherry'}

2.

Retun the removed element:

fruits = {"apple, banana, cherry}

x = fruits.pop()

print("x:", x)

output :-

x: banana

26. Remove :-

To remove the specified element from the set.

Syn:- set.remove (items)

Parameters:

item:- item from seach.

Ex:- fruits = {"apple", "banana", "cherry"}

fruits.remove ("banana")

print (fruits)

Output:-

{apple, cherry}

27. union method():-

* union method retuns club the perticular set items.

* If an item is present in more than one set it does not print the repeated values.

Syn:- set.union (set1, set2 ... etc)

Parameters:-

Set 1: seached item

Set 2: " "

Ex:-

x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

z = x.union (y)

Print (z)

Output:-

{banana, apple, google, microsoft, cherry}

Ex:-

2. x = {"a", "b", "c"}

y = {"f", "d", "a"}

z = {"c", "d", "e"}

result = x.union(y,z)

Print(result)

output :-

{'c', 'd', 'a', 'b', 'f', 'e'}

28.

update method() :-

These method updates the current set by adding the items from other set.

If an item is present in both sets only one appearance of the item will be present the update set.

Syn:- set.update(set)

Parameters:

set:- Seaching for the set items.

Ex:-

1. x = {"apple", "banana", "cherry"}

y = {"google", microsoft", "apple"}

x.update(y)

Print(x)

output:-

{"cherry", "banana", "microsoft", "google", "apple"}

Symmetric_difference :-

These method returns difference values of both sets, to be print. But common values from

the both sets does not print

Syn:- set.symmetric_difference(set)

Parameters:-

Set:- To check for the matching set items.

Ex:-

1. x = {"apple", "banana", "cherry"}

Y = {"google", "microsoft", "apple"}

z = x.symmetric - difference(y)

print(z)

output:-

{'google', 'banana', 'microsoft', 'cherry'}

30. set symmetric - difference - update():-

* These method returns both sets difference values and print it. If

* if common value in both sets remove the set item from the orginal set.

Syn:- set.symmetric - difference - update(set)

Parameters:

Set:- To checks the matching item.

Ex:-

1. x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

x.symmetric - difference - update(y)

Print(x)

output:-

{ banana, cherry.

31.

Issupperset():-

These method return true if all the items in the specified set existing in the orginal set; otherwise it returns, false.

syn:- set.Issupperset(set)

Parameters:

set:- To search for the perticular item. (Equal)

Ex:-

1. x = {"f", "e", "d", "c", "b", "a"}

Y = {"a", "b", "c"}

z = x.issupperset(Y)

Print(z)

output:-

z: True.

2. x = {"f", "e", "d", "c", "b"}

Y = {"a", "b", "c"}

m = x.issupperset(Y)

Print("m:", m)

output:-

m: False.

32. issubset():-

These method returns true if all the items in the set exist in the specified set, otherwise it returns false.

syn:- set.issubset(set)

Parameter:-

Set:- To search for the equal item.

Ex:-

1. x = {"a", "b", "c"}

y = {"f", "e", "d", "c", "b", "a"}

z = x.issubset(y)

print(z)

output:-

z = True.

2. x = {"a", "b", "c"}

y = {"f", "e", "d", "c", "b"}

z = x.issubset(y)

print(z)

output:-

z = false

Dictionary

Dictionaries are used to store data values in the form of key: value format

* Dictionary is a collection which is ordered, changable and does not allow duplicates.

* Dictonaries are written in the curly brackets {}.

How to create dictionary:-

```
this dict = {
    "brand": "vs polo",
    "model": "jackson",
    "year": 1964
```

}

Print (thisdict)

out put:-

{'brand': 'us polo', 'model': 'jackson', 'year': 1964}

Duplicates not allowed:-

* In dict. duplicate values does not allowed but values will be overwrite with the same key.

Ex:-

1. thisdict = {
    "brand": "us polo",
    "model": "jackson",
    "year": 1964,
    "year": 2000
}
Print (thisdict)

output:-

Dict length:-

* To find the how many items in a dict by using len().

Ex:-

1. thisdict = {
    "brand": "netplay",
    "model": "jackson",
    "year": 1964,
    "year": 2020
    "doll": "Deepu"

}

print (len(thisdict))

output:-

4

Dict - Data types:-

* It contain different data types in dict like string, int, bool and list.

Ex:-

1. thisdict = {
    "brand": "Ford"
    "electric": false;
    "year": 1964,
    "colors": ["red", "white", "blue"]
}

print (thisdict)

output:-

{'brand': 'Ford', 'electric': false, 'year': 1964, 'colors': ['red', 'white', 'blue']}

Type():-

To find the specified object is belongs to which kind of data types.

Ex:-

1. thisdict = {
    "brand": "mytriojas",
    "model": "Deepu",
    "year": 1964

```
}
print (type(thisdict))
```

output:-

```
<class 'dict'>
```

Accessing - items :-

To access the items of dict by refering to its key name inside the square brackets. []

Ex:-

```
1. thisdict = {
    "brand": "us polo",
    "model": "chinnu",
    "year": 1964
  }
  x = thisdict["model"]
  y = thisdict["year"]
  print = ("model name: ", x)
  print ("year :", y)
```

output:-

```
model name: chinnu
year: 1964.
```

Get keys :

This method will return a list of all keys in dictionary.

Ex:-

```
1. thisdict = {
    "brand": "Ford",
    "model": "mustang",
```

"year": 1964

}

x = thisdict.keys()

print(x)

output:-

dict_keys ['brand', 'model', 'year']

* Add new items to the original dictionary and see that key list gets updated as well.

Ex:

car = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = car.keys()

print(x)

output:-

dict_keys(['brand', 'model', 'year'])

car["color"] = "white"

Print(x)

output:-

dict_keys(['brand', 'model', 'year', 'color'])

values:-

These method returns a list of values in the given dictionary.

Ex:- thisdict = {

"brand": "Ford",

```
    "model" : "mustang",
    "year" : 1964
}
x = thisdict.values()
print(x).
```

Output:-

dict_values(['ford', 'Mustang', 1964])

Ex:-

2.

```
car = {
    "brand" : "ford",
    "Model" : "Deepu",
    "year" : 1994
}
x = car.values(),
print(x)
car["year"] = 2020
print(x)
```

Output:-

dict_values(['ford', 'Deepu', 1994])

dict_values(['Ford', 'Deepu', 2020])

Ex:-

3.

```
car = {
    "brand" : "chinnu",
    "model" : "Deepu",
    "year" : 1999
}
x = car.values
print(x)
car["color"] = "Black"
print(x)
```

output:-

dict - values [['chinnu', 'Deepu', 1999]

dict - values [['chinnu', 'Deepu', 1999, 'Black']]

Get items:-

These method method returns each item in a dictionary as a tuple it appear in the form of list.

Ex:-

```
1. thisdict = {
    "brand": "pitti"
    "model": "Milky",
    "year": 1995
}
x = thisdict.items()
print(x)
```

output:-

dict-items([['brand', 'pitti'], ['model', 'Milky'], ['year', 1995]])

Ex:-

```
2. car = {
    "brand": "chinnu",
    "model": "Jaanu",
    "year": 1995
}
x = car.items()
Print (x)
car ["year"] = 2020
Print (x)
```

output:-

dict-items([['brand', 'chinnu'], ['model', 'Jaanu'], ['year', 1995]])

dict-items([['brand', 'chinnu'], ['model', 'Jaanu'], ['year', 2020]])

Ex:-

```
3. car = {
    "brand": "chitti",
    "model": "gnana",
    "year": 2000
}
x = car.items()
Print(x)
car["color"] = "Black"
Print(x)
```

output:-

dict_items([('brand', 'chitti'), ('model', 'gnana'), ('year', 2000)])

dict_items([('brand', 'chitti'), ('model', 'gnana'), ('year', 2000), ('color', 'Black')])

if():-

If check the given dict key value present in or not.

Ex:-

```
1. thisdict = {
    "brand": "Deepu",
    "model": "Teja",
    "year": 2001
}
if "model" in thisdict:
    Print("yes, 'model' is one of the keys in the thisdict dictionary")
```

output:-

yes, model is one of the keys in thisdict dictionary.

change values:-

You can change the value of a specific item by referring to its key name.

Ex:-

```
1. chinnudict = {
       "brand" : "Pitti",
       "model" : "Minnu",
       "year" : 1964
   }
   chinnudict["year"] = 2018
   chinnudict["modle"] = "Mummy"
   print (thisdict)
```

Output:-

{'brand': 'Pitti', 'model': 'Mummy', 'year': 2018}

update:-

* These method will update the specified dictionary with the items from the given arguments.

* These arguments must be a dictionary (or) an iterable object with key : value pairs.

Ex:-

```
1. pitti dict = {
       "brand": "Ammu",
       "model": "Bubly",
       "year": 1964
   }
   pittidict.update ({"year": 2020})
   Print (pittidict)
```

output:-

{'brand': 'Ammu', 'model': 'Bubly', 'year': 2020}

Ex:-

```
2. gaanudict = {
       "brand": "Teja"
       "Model": "kumar",
       "year": 2001
```

Jaanudict.update ({"color": "red"})

print (Jaanudict)

output:-

{'brand': 'Teja', 'model': Kumar', 'year': 2001, 'color': red'}

Pop():-

* These Method removes the item with the specified key name.

Ex:-

1. thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 2000
  }
  thisdict.pop ("model")
  print (thisdict)

output:-

{'brand': 'Ford', 'year': 2000}

Del ():-

Del is a keyword some it removes the item with specified key name

Ex:-

thisdict = {
  "brand": "Bhagathi",
  "model": "Sreenu",
  "year": 1999
}
del thisdict ["model"]
print (this dict)

output:-

{'brand': 'Bhagathi', 'year': 1999}

To del complete dictionary:-

The del keyword can also del the complete

dictionary.

* whenever to del the complete dictionary to print the dict it raise an error.

Ex:-

```
1. thatdict = {
    "brand": "Amy",
    "model": "Jackson",
    "year": 1984
   }
   del thatdict
   Print (thatdict)
```

output:-

Error.

clear:-

It removes the Entire dictonary

(or)

Empties

Ex:-

```
1. thisdict = {
    "brand": "Deepu",
    "model": "Teja",
    "year": 1999
   }
   thisdict.clear()
   print (thisdict)
```

output:-

{ }

for loop - dict():-

It print the all key names in dict one by one.

Ex:-

```
1. thisdict = {
    "brand" : "Ford",
    "model" : "Mustang",
    "year": 2000
  }
  for x in thisdict:
  print(x)
```

output:-

brand
model
year.

```
2. thisdict = {
  "brand": "Deepu",
  "model": "Ammu",
  "year" : 1995
  }
for x in thisdict:
print (thisdict[x])
```

output:-

Deepu
Ammu
1995 ⟶ for - values :-

```
3. thisdict = {
   "brand" : "chinnu",
   "model" : "Pitti",
   "year" : 1994
  }
for x in thisdict.values():
 print (x)
```

output:-

chinnu
Pitti
1994

4. for - keys():-

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1994
}
for x in thisdict.keys():
  Print(x)
```

output:-

brand
model
year

5. for - items():-

```
thisdict = {
  "brand": "chinnu",
  "model": "Deepu",
  "year": 2000
}
for x, y in thisdict.items():
  print(x, y)
```

output:-

brand * chinnu
model * Deepu
year * 2000

6. copy():-

copy the particular dict into another object i.e another dict.

Ex:-

```
1. thisdict = {
     "brand": "Ford",
```

```
    "model": "Mustang",
    "year": 1994
}
mydict = thisdict.copy()
print(mydict)
```

Output:-

{'brand': 'Ford', 'model': 'Mustang', 'year': 1994}

Neasted dictionary:-

It contain one or more dictionaries by seperate with (,).

Ex:-

```
1. my family = {
    "child1" : {
      "name": Deepu",
      "year" : 1999
    },
    "child 2": {
     "name": Teja",
     "year": 2001
    },
    "child 3": {
     "name": Ammu",
     "year": 2004
    }
    }
   print(my family)
```

output:-

{'child1': {'name': 'Deepu', 'year': 1999}, 'child 2': {'nam

Ex:-

2. To assiging the items to perticular key.

```
child 1 = {
    "name" : "chinnu"
    "year" : 1994
}
child 2 = {
    "name" : "Deepa"
    "year" : 1999
}
child 3 = {
    "name" : "Rekha"
    "year" : 2002
}
my_family = {
    "child 1" : child 1,
    "child 2" : child2,
    "child 3" : child 3
}
print (myfamily)
```

output:-

{'child 1' : {'name' : 'chinnu', 'year' : 1994}, 'child 2' : {'name' : 'Deepa', 'year' : 1999}, child 3 : {'name' : 'Rekha', 'year' : 2002}}

Methods :-

1. clear():-

* It retyns the given dict: Empties.

Ex:-

```
1. car= {
  "brand": "BMW",
  "model": "Mustang",
  "year": 1997
  }
  car.clear()
  Print (car)
```

output:-

{ }

2. copy method():-

These method retyns {keys and: values} copied into Peytcular object.

Ex:-

```
1. car = {
    "brand": "RR",
    "model": "Deepu",
    "year": 1992
  }
x = car.copy()
print(x)
```

output:-

{'brand': 'RR', 'model': 'Deepu', 'year': 1992}

3. From keys method():-

These method retuns a dict with specified key and specified value

Syn:- dict.fromkeys(keys, value)

Parameteqs:

keys:- An ittegable specifing the keys of dict.

value :- It is optional. The value for all keys. Default value is None.

Ex:-

1. x = ('key1', 'key2', 'key3')

y=4

thisdict = dict.fromkeys(x,y)

print(thisdict)

output:-

{'key1':4, 'key2':4, 'key3':4}

without specifing value.

2. x= ('key1', 'key2', 'key3')

thisdict = dict.fromkeys(x)

print(thisdict)

output:-

{'key1':None, 'key2':None, 'key3':None}

4. Get method():-

These method retuqns the value of the item with specified key.

Syn:- dict.get(key name, value)

Parameteqs:

keyname:- To specify the peqticulaq key we want to retuqn

the value from dict

value:- It is optional.

Ex:-

1. car = {
    "brand": "Ford",
    "model": "Amal",
    "year": 1999
  }
  x = car.get("model")
  print(x)

output:-
Amal

5.
Items method():-

It returns a view object. The view object contains the key-value pairs of dict, as a Tuple in a list.

* The view object will reflect any changes done to the dict.

syn:- dict.items()

Ex:-

1- car = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
  }
  x = car.items()
  print(x)

output:-
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

2.
car = {

```
"brand": "Ford",
"model": "Mustang",
"year": 2000
}
x = car.items()
car["year"] = 2018
print(x)
```

output:-

dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2018)])

6. keys method():-

* These method returns a view object. the view object contains the keys of the dict, as a list.
* Thes changes done to the dict

Ex:-

1.
```
car = {
"brand": "Ford",
"model": "Deepu",
"year": 1994
}
x = car.keys()
print(x)
```

output:-

dict_keys(['brand', 'model', 'year'])

change - keys:-

2.
```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1995
}
x = car.keys()
```

```
car["color" ]= "white"
print(x)
```

Output:-

```
dict-keys(['brand', 'model', 'year', 'color'])
```

1. pop method:-

These method removes the specified item from the dict. It print the reaming items except poped item.

Syn:- dict.pop(key name, default value)

Parameters:

key name:- To remove the perticular key name.

default value:- It is optional.

Ex:-

```
1. car={
    "brand": "Deepu",
    "model": "Bhajathi",
    "year": 1999.
   }
car.pop("year")
print(car)
```

output:-

```
{'brand': 'Deepu', 'model': 'Bhajathi'}
```

Remove-value-print:-

Ex:-

```
1. car = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1992
   }
x = car.pop("model")
```

```
print(x)
```

output:-

Mustang.

8. pop item():-

* It removes the item that was last inseated in dict.

* In versions before 3.7 the pop item method removes a random item.

* Syn:- dict.popitems()

Ex:-

1.
```
car = {
  "brand" : "Ford",
  "model" : "mustang",
  "year" : 1999
}
car.popitem()
print(car)
```

output:-

{'brand : Ford , model : Mustang}

Remove - value - print:-

2.
```
car = {
  "brand : "Ford",
  "model" : "Mustang",
  "year" : 2000
}
x = car.popitem()
print(x)
```

output:-

{year : 2000}

9. update method():-

To update the dict with specified key and value Pairs.

Syn:- dict.update(iterable)

Ex:-
```
car = {
  "brand": "uspolo",
  "model": "BMW",
  "year": 1964
}
car.update({"color": "Black"})
print(car)
```
output:-

{'brand': 'uspolo', model: BMW, year: 1964, color: Black}

10 values method():-

* It contains values of the dict as a list.
* Any changes refelect done to the dict.

Syn:- dict.values()

Ex:-
1.
```
car = {
  "brand": "milk",
  "model": "Amul",
  "year": 1901
}
x = car.values()
car["year"] = 2018
print(x)
```
output:-

dict_values(['milk', 'Amul', 2018])

Functions

* A function is a block of statements.
* functions provide better modularity for user application and using high degree of code.
* It can be classified into two types.

1. predefined function (or) built-in functions

* These functions contain definition in compiler itself.

Ex:- type ( )
id ( )
print ( )

2. user defined function:-

* user can create its own function.

Syn:- def functionname (parameter):
block of statements
return [expression]

* function blocks begin with the keyword def followed by functionname and ( ).

* In paranthesis it contain Parameter (or) arguments.
* the code block in every function starts with : and is indented.
* The statement return exist in a function, it returns, Expression.
* A return statement with no argument it returns None.

Ex:-

1. def wish(name):

    Print("welcome to functions topic", name, "learners")

wish("Deepu")

wish("Ammu")

output:-

welcome to functions topic Deepu 'learners'

welcome to functions topic Ammu 'learners'

2. def My_functions():

    Print("Hello welcome to functions")

my_function()

output:- Hello welcome to functions.

calling function :-

once the basic structure of a function is finalized, we can execute it by calling from another function directly.

Ex:-

1. def printme(str):

    print(str)

    return;

printme("I'm first call to user define function!")

Printme("Again second call to the same function")

output:-

I'm first call to user define function.

Again Second call to the same function.

2. def foodname (str):

    Print(str)

foodname ("Tiffan is : chapathi")

foodname ("Lunch is : bissi bele bath")

foodname (" Dinner is : curd Rice")

output:-

Tiffan is : chapathi

Lunch is : bissi bele bath.

Dinner is : curd rice.

Pass by refeqence vs value:-

we can change what a parameteq "refeqce" to with in a function. The changes can also reflex "back" in the calling function.

Ex:-

1. def changeme (mylist):

    mylist.append ([1, 2, 3, 4]);

    Print ("values inside the function :", mylist)

    retqn.

mylist = [10, 20, 30];

changeme (mylist);

print ("values outside the function :", mylist)

output:-

Values inside the function : [10, 20, 30, [1, 2, 3, 4]]

Values outside the function : [10, 20, 30, [1, 2, 3, 4]]

Ex:-

2. def changeme (mylist):

    mylist = [1, 2, 3, 4];

```
    Print ("values inside the function : ", mylist)
    return.

mylist = [10, 20, 30];
changeme (mylist)
Print ("values outside the function:", mylist)
```

Output:-

values inside the function : [1, 2, 3, 4]

values outside the function : [10, 20, 30]

Function arguments:-

It can be classified into 4 categories-

* Required arguments
* keyword arguments.
* Default arguments.
* variable-length arguments.

1. Required arguments:-

The number of arguments in the function call should match exactly with the function definition.

Ex:-

```
1. def printme (str):
       print (str)
       return;

printme (str)
```

output:-

<class 'str'>

2. keyword arguments:-

keyword arguments are related to the function calls. when we use keyword arguments in function call, the

called identifies the arguments by the parameter name.

Ex:-

1. def printme (str):

    Print (str)

    return ;

Print me (str = "My string is Python hello")

output:-

My string is Python hello.

2. def printinfo (name , age):

    Print ("Name :", name)

    Print ("Age ", age)

    return;

Print info (age=50, name = "venkat")

output:-

Name: venkat

Age: 50.

3. default arguments:-

default argument is an argument that assumes default values, if a value is not provided in the function call for the argument automatically it take default value.

Ex:-

1. def printinfo (name = "ram", age=25):

    Print ("Name:" , name)

    Print ("Age ", age)

    return;

```
Print info (age = 20, name = "Deepu")
Print info (name = "Milky")
Print info()
```

Output:-

Name: Deepu

age: 20

Name : Milky

age: 25

Name: Ram

age : 25.

4. Variable - length arguments:-

To suply more number of arguments to the function defination.

Syn:- def . functionname (args , * var args tuple):

"functionblock"

retun [Expression]

* An asteqisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments.

* These tuple remains empty if no additional arguments are specified during the function call.

Ex:-

```
1. def  printinfo (arg1 , * vartuple ):
        Print ("output is:")
        Print (arg1)
        for var in vartuple:
            Print (var)
        retun;
```

print info (10)

print info (70, 40, 90)

output:-

output is :

10

output is:

70

40

90.

Anonymous Function:- (lambda)

* These function are called anonymous because they are not declared in the standard manner by using the def keyword.

* You can use the lambda keyword to create small anonymous function.

* lambda forms can take any number of arguments but return only one value in the form of expression. They cannot contain commands (or) multiple expression.

* A Anonymous function cannot be direct call to print because lambda required an expression.

* lambda functions have their own local name space and cannot access variable other than those in their parameter list and those in the global namespace.

syn:- lambda [arg1, arg2 .... argn]: expression.

Ex:-

1. sum = lambda arg1, arg2 : arg1 + arg2 ;

print ("value of total : ", sum (10, 20))

print ("value of total :", sum (20, 20))

output:-

value of total : 30

value of total : 40

return:-

whenever function call occur the function defination contain return statement. these can return expression.

Ex:-

```
1. def Sum (arg1, arg2 ):
        total = arg1+arg2
        Print ("Inside the function :", total)
        return total;

total =Sum (100, 20);
Print ("outside the function :", total)
```

OutPut:-

inside the function : 120

outside the function: 120.

Scope of variable:-

variable that are defined inside of the function body have a local scope and those define outside have a global scope.

Ex:-

```
1. total=0
def Sum (arg1, arg2 ):
        total = arg1 +arg2;
        Print ("Inside the function local total : ", total)
        return total;
Sum (10,20);
```

```
print ("outside the function global total : ", total
```

output:-

inside the function local total: 30

outside the function global total: 0

lambda Examples:-

1.
```
x = lambda a: a+10
print (x(5))
```

output:-

15.

2.
```
x= lambda a,b : a*b
Print (x (5,6))
```

output:-

30.

3.
```
x = lambda a,b,c :a +b+c
print (x(5 ,6,2))
```

output:-

13.

4.
```
def myfunc (n):
    return lambda a: a*n

mydoubleq = my func (2)
print (mydoubleq(11))
```

output:-

22.

Modules

What is Module?

* A Module to be the same as a code of library.
* A file containing set of functions you want to include in your application.

How to create Module:

To create a Module just save the code you want in a file with the file extension .py.

Save the code in the file name "mywon.py"

Ex:-

```
1. def greeting(Deepu):
       Print ("hello ," +Deepu)
```

use as a Module:

Now we can use the Module, just we created module by using the "import" statement.

* Module name . calling function.

Ex:- Myown . greeting (" welcome to my world")

Ex:-

```
1. import Myown
   Myown . greeting (" welcome to my world")
```

output:-

'Hello', welcome to my world.

Note:-

When using a function from module use the

Syn:- Modulename.functionname

Ex:- myown.add(10,20)

Variables in Module :-

The Module can contain functions, as already discribed (applications), but also variables of all types (arrays, dict, object ... etc).

Example- Create (Person) : Module - myown1

```
Person ={
        "name" : "Deepu",
        "country" : "India",
        "game" : "kabaddi",
        "age" : 21,
        "gender": "female"
    }
```

* Import the module name (myown1) and access the person dict.

Ex:-

1. &

```
import myown1
a= myown1.person["name"]
print ("name :",a)
b= myown1.person["country"]
Print ("country:", b)
c= myown1.person["game"]
Print ("game :", c)
d= myown1.person["age"]
```

```
print ("age :", d)
e = myown1.Person["gender"]
print ("gender:", e)
```

output :-

name : Deepu

country : India

game : kabadd:

age : 21

gender : female.

Re-naming a module :

We can create an alias name for module by using the "as" keyword.

Create an alias name for myown1 module called as m. M is nothing but alias name for module.

Ex:-

1.

```
import myown1 as c
a = c.person["name"]
print ("age:", a)
b = c.person["country"]
print ("country:", b)
c = c.person["game"]
print ("game :", c)
d = c.person["age"]
print ("age :", d)
e = c.person["gender"]
print ("gender:", e)
```

output:-

name : Deepu

country : India

game : kabadd:

age : 21

gende.q : female.

Built in module:

Theqe aqe seveqal inbuilt modules in python which you can import whenevey you like.

import and use the platform module.

Ex:-

```
import platform
x= platform . system ()
print (x)
```

Output:-

windows.

Import from module:-

you can choose to import only parts from module by using "from" keycoord

Ex:- the module name myexon to has one function and one dict.

```
def greeting (name):
    Print ("hello , " +name )

PeqSon = {
    "name" : "chinnu",
    "country" : "india"
```

"game" : "cricket",
"age" : 25,
"gender" : "male"
}

Ex:-
1. Import only the person1 dict from the module (myown)

```
from myown2 import person
Print ("name:", person ["name"])
Print ("age :", person ["age"])
Print ("country :", person ["country"])
Print ("game :", person ["game"])
```

output:-

name : chinnu
age : 25
country : India
game : cricket.

Note:-
when importing using the from keyword, dus not use the module name when refearing to Elements in the module.
Example: person1 ["age"], not use module
↓
my module . person1 ["age"]

Math - module :
It contain set of inbuilt math functions, that peafoms the Mathematical tasks on numbeq.

1. max() - It find the maximum value in a give itepable.
2. min() - It find the minimum value in a given itepable.

Ex:-

1. x = min (5, 10, 25, 4)

   y = max (5, 10, 25, 240)

   print ("min:", x)

   print ("max:", y)

output:-

x : 4

y : 240.

abs():-

To convert the given values into possitive value.

Ex:-

1. x = abs (-7.25)

   print ("abs:", x)

Output:-

7.25

Pow():-

The pow(x, y) function returns the value of x to the power of y (x^y)

Ex:-

1. x = pow(2, 4)

   print ("Pow: ", x)

output:-

16.

ceil:-

Round a number upward to its nearest integer.

floor:-

Round a number downwards to its nearest integer.

Ex:-

```
1. import math
x= math.sqrt(64)
Print ("Sqrt:", x)
x1 = math.ceil(1.4)
y= math.floor(1.4)
print ("ceil:", x1)
print ("floor:", y)
z= math.pi
Print ("pi:", z)
```

Output:-

```
Sqrt: 8.0
ceil: 2
floor: 1
Pi: 3.141592653589793.
```

Working with random modules:-

* These module defines several functions to generate random numbers.

* We can use these function while developing function cryptography and to generate random numbers on fly for authentication.

Random:-

These function always generates some float values b/w 0 and 1 (not inclusive)

$0 < x < 1$

Ex:-

```
1. from random import *
for i in range(10):
    print (random())
```

output:-

0.20778417619644402.

0.7906426630412382

0.4620449936 2489

0.9403 1235 67984

Randint ():-

To generate random integers b/w two given numbers.

Ex:-

```
1. from random import *
for i in range(10):
    Print (randint (1, 100))
```

output:-

35

44

99

84

75

24

41

56

25

11

9

4

uniform ():-

It returns random float values b/w two given numbers (not inclusive)

Ex:-

```
1. from random import *
for i in range(1,5):
        print (uniform(1,5))
```

Output:-

4.57932456

3.688194827

1.76220153 8

2. 6457321046

DIR :-

There is a built it display the list of all inbuilt functions belonging to the platform module.

Ex:-

```
1. import platform
   x = dir (platform)
   print (x)
```

Output:-

['_processor', '_WIN32_CLIENT_RELEASES', '_WIN32_SERVER_RELEASES', '__builtins__', '__cached__', '__copyright__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '__version__', '_comparable_version', '_component_re', '_default_architecture', '_follow_symlinks', '_get_machine_win32', '_ironpython26_sys_version_parser', '_ironpython_sys_version_parser', '_java_getprop', '_libc_search', '_mac_ver_xml', '_node', '_norm_version', '_platform', '_platform_cache', '_pypy_sys_version_parser', '_sys_version', '_sys_version_cache', '_sys_version_parser', '_syscmd_file', '_syscmd_ver', '_uname_cache', '_unknown_as_blank', '_ver_output', '_ver_stages', 'architecture', 'collections', 'functools', 'itertools', 'java_ver', 'libc_ver', 'mac_ver', 'machine', 'node', 'os', 'platform', 'processor', 'python_branch']

class :-

* python is supporting objected oriented programming language.
* In python everything is an object.
* A class contains (or) properties (or) attributes. and methods.
* A class is a blue print (or) logical Entity.

How to create class :-

```
class classname:
    Attributes (or) properties
    methods.
```

Ex:-

```
class Book:
    cost
    colour
    DetailsBook()
```

To create a class use the keyword "class"

Ex:-

```
class human:
    color = "Black"
    height = 5.11
    def run(self):
        Print("running")
    def walk(self):
        Print("walking")

B = human()
Print(B.color)
Print(B.height)
B.run()
B.walk()
```

output :-

```
Black
5.11
running.
```

walking.

Create object :-

By using the class name to create the object.

Syn:- object name = classname()

Ex:-

```
1. class Myvalue :
    x = 100
P1 = Myvalue ()
Print ("value :", P1.x)
```

output :-

value : 100.

__init__ :- __init__() function :- / constructor :-

* It is a built in function.

* All classes have a function called __init__(), which is always executed when the class is being initated.

Use the __init__() function to asign values to object properties (or) other operations that are necessary to do when the object is being created.

* It is a constructor.

Ex:-

1.

```
class Person :
    def __init__ (self, name, age):
        self.name = name
        self.age = age

P1 = Person ("Sree lakshmi", 26)
P2 = Person ("Ammu", 30)
Print (P1.name)
Print (P1.age)
Print (P2.name)
```

print (p2.age)

output :-

Sree lakshmi

26

Ammu

30.

Note :-

The init fonction is called automatically every time the class is being used to create new object.

object Method :-

* object can also contain methods.

* In methods objects are fonctions that belongs to object.

Ex:-

```
1. class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age.

    def myfunction(self):
        print("sweet my name is " + self.name)

p1 = Person("Deepu", 21)
p1.myfunc()
```

output :-

Sweet my name is Deepu.

Self Parameter :-

The self parameter is a reference to the current instance of the class and is used to access the variables of a class.

It does not have to be named self, we can call it whatever we like.

* It has to be the first parameter of any function in the class.

Ex:-

1.

```
class Person:
    def __init__(Deepu, name, age):
        Deepu.name = name
        Deepu.age = age

    def myfunc(abc):
        print("Hello my name is " + abc.name)

p1 = Person("pitti", 25)
p1.myfunc()
```

output:- Hello my name is pitti.

2.

```
class Person:
    def __init__(chinnu, name, age):
        chinnu.name = name
        chinnu.age = age

    def myfunc(abc):
        print("Hello my name is " + abc.name)

    def myfunc(rdm):
        print("hello my name is " + rdm.name)

P1 = Person("ram", 26)
P1.myfunc()
P2 = Person("milky", 21)
P2.myfunc()
```

name

hello my x is ram

hello my name is milky

Modify object properties:-

* we can modify the object properties.

Ex:-

1.

```
class person :
    def --init-- (self , name , age):
        self . name = name
        self . age = age

    def myfunc(self ):
        Print ("Hello my name is " + self.name )

P1 = person (" Deepu ", 12 )
P1 . age = 21
Print ( P1.age)
```

output:-

21

Delete object properties:-

we can delete properties on object by using del keyword.

Ex:-

1.

```
class Person:
    def --init-- (self , name ,age):
        self . name = name
        self . age =age

    def myfunc(self ):
        Print ("Hello my name is " +self . name )
P1 = Person ("John", 36 )
del P1 . age
```

```
print (p1.age)
```

output:-

Person object has no attribute of age.

Error.

Delete object :-

We can delete objects by using del keyword.

Ex:-

1.

```
class person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = person("John", 36)
del p1
print(p1)
```

output:- p1 is not defined

Error.

Pass statements:-

* class definations cannot be empty, but if you from some reason have a class defination with no contain put in the pass statement to avoid getting an error.

Ex:-

1.

```
class person:
    pass
```

Output:- Empty

Inheritance

Inheritance allows us to define a class. To acquire the properties and methods from 1 class to another class.

parent class:- the class is being inherited from called base class.

child class:- to acquire the properties and methods from base class to derived class.

Inheritance can be classified into

1. single inheritance. }
2. single " "
3. Multi level inheritance
4. Hierarchical inheritance.
5. Multipule inheritance.

1. create a parent class or single inheritance:-

It is like a same as a creating a class.

syn:- class classname :

Attributes
Methods

Ex:-

1.

```
class base:
    a=10
    b=20
    def dis(self):
        print("base class")
class derivedclass :
    c=40
    d=90
```

```
    def show(self):
        print("derived class")

bobj = base()
bobj = dis()
print(bobj.a)
print(bobj.b)
do = derivedclass()
do.show()
print(do.c)
print(do.d)
```

output:-

base class

10

20

derived class

40

90

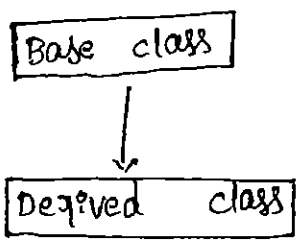
2. Derived class / single inheritance:-

To acquired the properties and methods from base class to derived class

syn:- class Derivedclass(base class):

attributes

methods



* To create the object in derived class we can access base classes and derived classes properties.

* Don't create object in base class because we cannot access properties of derived class.

Ex:-

1.

```
class base:
    a=100
    b=200
    def disc(self):
        print("It is base class")

class deq(base):
    c=40
    d=90
    def show(self):
        print("It is deqived class")

dobj=deq()
print(dobj.a,dobj.b)
dobj.dis()
print(dobj.c,dobj.d)
dobj.show()
```

output:-

100 200

it is base class.

40 90

it is deqived class.

Example of single inheqitance:-

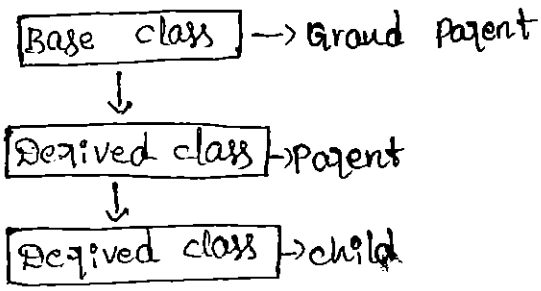
```
class sweet:
    a = "palaksova"
    b= "laddu"
    def dis(self):
        print("base class")
class hot:
    c= "banana chips"
    d= "aalu chips"
    def show(self):
```

```
Print C"derived class")
C obj = sweetc)
cobj = dis c)
Print ccobj.a)
Print (cobj.b)
do = hotc)
do.showc)
Print cdo.c)
Print cdo.d)
```

output :-

base class
Palakova
laddu
derived class
banana chips
aalu chips.

3. Multi level inheritance:-



* Parent class can access attributes and methods of grand parent. → Base class
↓
derived class

* child class can access attributes and methods of parent and grand Parent.
↓
Derived class

Syn:-

```
class Grand parent:
    Attributes
    methods
    class Parent (grand parent):
    Attributes
    Methods
    class child (parent):
    Attributes
    methods.
```

Ex:-

```
1. class gparent:
       age2=60
       def gpdis(self):
           print("grand Parent method")

class par(gparent):
       age1=40
       def pardis(self):
           print("parent method")

class child(par):
       age=24
       def cdis(self)
           print("child method")

c=child()
Print("child class age:", c.age)
cdisc)
Print("Parents class age:", c.age1)
c.pardis()
print("grand parents class age:", c.age2)
c.gpdis()
```

output:-

child class age: 24

Child method

Parent class age: 40

Parent method

grand parent class age: 60

grand parent method.

2.

```
class savings:
    a= amount = 10000
        def save(dis):
            print ("This amount is savings")

class withdraw(savings):
    b= amount = 5000
        def withdraw(dis):
            print ("This amount is withdraw")

class current balance(withdraw):
        c= amount = 15000
            def cbalance(dis):
                print ("This amount is current balance").

d= current balance(cbalc), c.savedis()
print ("current balance amount:", d.amount(a))
d.withdrawdis()
print ("withdraw amount:", d.b)
d.cbaldis()
Print ("current balance amount:", d.a-d.b)
```

output:-

Saving amount

cbalance amount : 10000

withdraw amount

withdraw amount :5000

current balance

c balance amount : 5000

3. class tri:

```
    b=2
    h=2.3
    def tridis(self):
        print("triangle is :")

class sqr(tri):
    s=5
    def sqrdis(self):
        print("square is :")

class rec(sqr):
    l=4
    m=9
    def recdis(self):
        print("rectangle is :")

f.recc()
f.tridis()
print("area of triangle :", 0.5 * f.b * f.h)
f.sqrdis()
print("area of square :", f.s * f.s)
f.recdis()
print("area of rectangle :", f.l * f.m)
```

output:-

```
tringle is :
area of tripangle : 2.3
square is :
area of square : 25
rectangle is :
area of reetangle : 36
```

4. Hierarchical inheritance:-

Syn:- class Base class:

Attributes

Methods

class Derived class 1 (Base class):

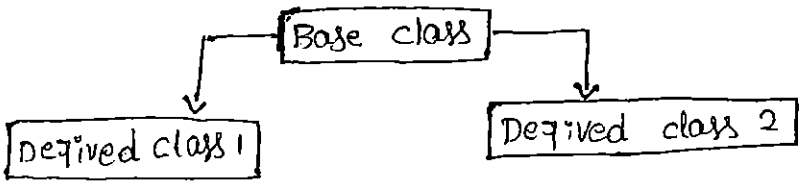
Attributes

Methods

class Derived class 2 (Base class):

Attributes

methods



* one base class and two derived classes is called hierarchical inheritance
* Derived class 1 can access only base class attributes and methods.
* Derived class 2 can access only base class attributes and methods.

Ex:-

```
1. class parent:
       like = "Sweets"
       def disc(self):
           print ("parent class")

   class son (parent):
       L1 = "clothes"
       def sdis (self):
           Print ("son class")
   class daughter (parent):
       L2 = "jewelary"
       def ddis (self):
           Print ("daughter class")
```

```
S=son()
Print("son like:", S.like)
S.sdis()
Print("Parent like:", S.like)
S.dis()
d=daughter()
Print("Parent like:", d.like)
print("daughter like:", d.le)
d.ddis()
```

Output:-

Son like: clothes

Son class

Parent like: sweets

Parent class

Parent like: Sweets

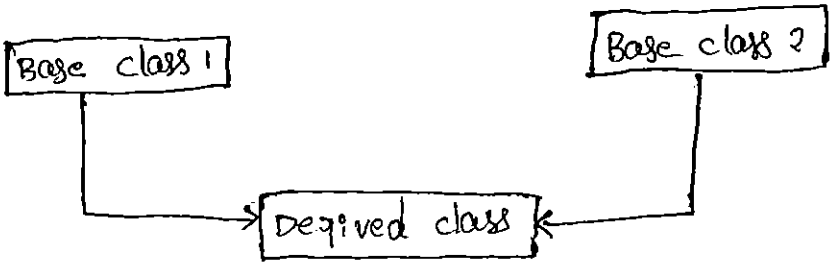
daughter like: jewelary

daughter class.

5. Multipule Inheritance:-

Syn:-

```
class Base class 1:
Attributes
methods
class Base class 2:
Attributes
methods
class Derived class (Base class 1, Base class 2):
Attributes
methods.
```



* To create both base classes to create one derived class.
* To access the both base classes properties by using derived class.

Ex:-

```
1. class fathey:
        height = 6.1
        def fdisc(self):
            print ("fathey class it is")

   class mothey:
        color = "white"
        def mdisc(self):
            print ("it is mothey class")

   class child (fathey, mothey):
            l = "knowldge"
            def cdisc(self):
                print ("it is child class")

   c = child()
   c.fdisc()
   print ("aquire from :", c.height)
   c.mdisc()
   print ("aquire from :", c.color)
   c.cdisc()
   print ("Its own talent :", c.l)
```

output:-

```
fathey class it is
aquire from : 6.1
it is mothey class
aquire from: white
it is child class
its own talent : knowldge.
```

Example of hierarchical inheritance:-

Ex:- 1

```
1. class whatsapp:
        like = " messages"
        def dis(self):
            print("whatsapp class")

class facebook(whatsapp):
        L1 = "Social media"
        def fdis(self):
            print("facebook class")

class twitter(whatsapp):
        L2 = "post"
        def tdis(self):
            print("twitter class")

d = facebook()
print("facebook like :", d.L1)
d.fdis()
Print("whatsapp like :", d.like)
d.dis()
c = twitter()
Print("twitter like :", c.like)
print("twitter like :", c.L2)
c.tdis()
```

output:-

```
facebook like : Social media
facebook class
whatsapp like : message
whatsapp class
twitter like : messages
twitter like : post
twitter class
```

Example of multipule inheritance :-

```
class veg:
    dal = "yammy"
    def vdis(self):
        print("It is veg class")

class nonveg:
    chicken = "spicy"
    def ndis(self):
        print("It is nonveg class")

class fruits(veg, nonveg):
    l = "healthy"
    def fdis(self):
        print("It is fruits class")

d = fruits()
d.vdis()
print("aguige from :", d.dal)
d.ndis()
Print("aguige from :", d.chicken)
d.fdis()
Print("It is healthy food :", d.l)
```

output :-

```
    It is veg class
  aguige from : yammy
    It is nonveg class
  aguige from : spicy
    It is fruits class
  It is healthy food : healthy
```

Scope :-

A variable is only available from inside the region it is created. This is called scope.

local scope:-

A variable created inside of a function belongs to the local scope of that function and can only be used inside that function.

Ex:-

```
1. def myfunc():
       x = 300
       print(x)

myfunc()
```

output:-

300

Function inside Function :

outer function contain another function (inner function) is called nested function.

Ex:-

```
def myfunc():
    x = 300
    def myinnerfunc():
        print(x)
    myinnerfunc()
myfunc()
```

output:-

300

Global scope:

A variable are available within and outside the function.

Ex:-

```
x = 300
def myfunc():
```

```
    print(x)

myfunc()

print(x)
```

Output:-

300
300

Naming variable:

If you operate with the same variable name inside and outside of function, python will treat them as two separate variables, one variable in the global scope (outside of the function) and one available in the local scope (inside the function)

Ex:-

1.
```
x = 300

def myfunc():
    x = 200
    print("local scope :", x)

myfunc()
print("global scope :", x)
```

output:-

local scope : 300
global scope : 200

global keyword:

If you want to create global variable, but it is in local scope, you can use the "global" keyword make the variable as a global

Syn:- global variable name

Ex:- global x

Ex:-

1. def myfunc():

    global x

    x=300

    print(x) # local scope

myfunc()

print(x) # global scope

output:-

300

300.

Ex:-

2. def myfunc():

    global x

    x = 200

myfunc()

print(x)

output:-

200.

Python Dates:

We can import a module named "datetime" to work with dates as a date objects.

To import the date & time module and display the current date.

syn:- object = datetime.datetime.now()

In the above syntax datetime is module, datetime is class, now() is a method.

Ex:-

```
1. import datetime
x = datetime.datetime.now()

Print(x)
```

output:-

2021-07-01 14:08:00.990479

Creating date objects:

We can create the date by using datetime module, datetime() constructor.

The datetime class requires three parameters i.e year, month, date.

Syn:- object name = datetime.datetime(year, month, day)

In the above syntax datetime is module, another datetime() is a constructor.

Ex:-

```
1. import datetime
x= datetime.datetime(2020, 5, 17)

Print (x)
```

output:-

2020-5-17 00:00:00

strftime(parameter):-

It contain single parameter of formating parameter. -

It contain formating date and time fields.

Parameters in strftime():

Year: 1. %y – It returns 2 digits of year (Ex- 1990, output: 90)

2. %Y – It returns 4 digits of year (Ex:- 1990, o+p: 1990)

month:

1. %b – It returns short version of month
2. %B – It returns full version of month.

day:

1. %a – It returns short version of day
2. %A – It returns full version of day

Hours:

1. %H – It returns 24 hrs format (0 to 23)
2. %I – It returns 12 hrs format (0 to 12)
3. %p – It returns it is A.M or P.M
4. %M – It retrieve 0 to 59 Min's
5. %S – It retrieve 0 to 59 sec's
6. %f – It retrieve 0000 to 9999 micro seconds.

Syn:- strftime("formating parameter")

Ex:-

```
1. import datetime
   res = datetime.datetime.now()
   print(res)
```

output:-

2021-07-01 14:39:02.703331

Ex:-

```
2. import datetime
   res = datetime.datetime.now()
   res1 = datetime.datetime(2021, 6, 24)
```

```
print (res)
print ("date : ", res1)
```

output:-

2021-07-01 14:36:14.421390

date : 2021-06-24 00:00:00

Ex:- (year)

```
3. import datetime
cd = datetime.datetime.now()
res = cd.strftime("%y")
print ("Short version of year : ", res)
res1 = cd.strftime("%Y")
print ("full version of year :", res1)
```

output:-

Short version of year : 21

full version of year : 2021

Ex:- (day-month, o Sun)

```
4. import datetime
x = datetime.datetime.now()
print (x.now())
print ("full version of day : ", x.strftime("%A"))
print ("Short version of day :", x.strftime("%a"))
print ("Short version month : ", x.strftime("%b"))
print ("full version of month : " x.strftime("%W"))
```

output:-

2021-07-01 14:42:38.475888

full version of day : Thursday

Short version of day : Thu

Short version of month : 4

full version of month : 26.

Ex:- (hours)

```
5. import datetime
x = datetime.datetime.now()
```

```
Print (x.now())
Print ("24 hours time :", x.strftime("%H"))
Print ("12 hours time :", x.strftime("%I"))
Print ("time :", x.strftime("%p"))
Print ("mintues :", x.strftime(%M"))
Print ("Seconds :", x.strftime("%S"))
Print ("micro seconds :", x.strftime("%f"))
```

output :-

```
2021-07-01  14:51:04.794142
24 hours time : 14
12 hours time : 02
   time : PM
mintues : 51
Seconds : 04
micro seconds : 794142.
```

Time delta function:

It is available in date time library.

By using this function manipulation on dates.

Syn:- import datetime

Timedelta (day = value, hours = value, mintues = value, seconds = value, Microseconds = value)

Ex:-

```
1. import datetime
current_date = datetime.datetime.now()
Print ("current_date :", current_date)
new_date = current_date + datetime.timedelta(day=2)
Print ("new date :", new_date)
```

output :-

```
current_date : 2021-07-02  14:15:23.226512
new date : 2021-07-04  14:15:03.226512
```

Ex:-

2.
```
import datetime
current_date = datetime.datetime.now()
Print ("current-date :", current_date)
new_date = current_date + datetime.timedelta (day = 36)
print ("new date :", new_date)
```

output:-

current-date : 2021-07-02 14:20:18.044498

new date : 2021-05-27 14:20:18.044498.

Ex:-

3.
```
import datetime.
current_date = datetime.datetime.now()
print ("current-date:", current_date)
new_date = current_date + datetime.timedelta (weeks = 2)
print ("new date :", new_date)
```

output:-

current-date : 2021-07-02 14:25:06

newdate : 2021-07-16 14:25:06.

Ex:-

4.
```
import datetime
current_date = datetime.datetime.now()
Print ("current-date :", current_date)
new_date = current_date + datetime.timedelta (hours = 2
```

output:-

2021-07-02 14:27:06

newdate : 2021-07-02 16:27:06

Ex:-

5.
```
import datetime
current_date = datetime.datetime.now()
Print ("current-date:", current_date)
new_date = current_date + datetime.timedelta (minutes = 20)
Print ("new date :", new_date)
```

current-date : 2021-07-02 14:32:44

new date : 2021-07-02 14:52:44

Ex:-

6. import datetime

current-date = datetime.datetime.now()

Print ("current-date :", current-date)

new-date = current-date + datetime.timedelta (microseconds = 20)

print ("new date :", new-date)

output:-

current-date : 2021-07-02 14:37:16.944189

new date : 2021-07-02 14:37:16.944209.

Ex:-

7. import datetime

current-date = datetime.datetime.now()

Print ("current-date :", current-date)

new-date = current-date + datetime.timedelta (day = 20)

Print ("new date :", new-date)

difference = new-date - current-date

Print ("diffeqence date :", diffeqence)

output:-

current date : 2021-07-02 14:42:17

new date : 2021-07-22 14:42:17

difference date : 20 days.

Today:-

It display the current date.

Ex:-

1. import datetime

dt = datetime.date.today()

print ('Today \'s Date =', dt)

2. import datetime

today = datetime.date.today()

```
Print C'Today's Date = ', today)
Print C'Year from Today's Date = ', today.year)
Print C'Month from Today's Date = ', today.month)
Print C'Day from Today's Date = ', today.day)
```

Output:-

Today's Date = 2021-07-05

Year from Today's Date = 2021

Month from Today's Date = 7

Day from Today's Date = 5.

Ex:-

```
3. import datetime
Print C'Maximum year = ', datetime.MAXYEAR)
Print C'Minimum year = ', datetime.MINYEAR)
```

Output:-

Max year = 9999

Min year = 1

Ex:-

```
4.
import datetime import datetime
dt = datetime.now()
Print C'Date from = ', dt)
Print C'year from Date = ', dt.year)
print C'Month from Date = ', dt.month)
print C'Day from Date = ', dt.Day)
Print C'Hour from Date = ', dt.hour)
Print C'Minute from Date = ', dt.minute)
Print C'Second from Date = ', dt.second)
Print C'Microsecond from Date = ', dt.microsecond)
Print C'weekday Number from Date = ', dt.weekday())
```

Output:-

Date = 2021-07-05 14:24:02.080931

Year from Date = 2021

Month from Date = 07

Day from Date = 5

Hour from Date = 14

Minutes from Date = 24

Second from Date = 2

Microsecond from Date = 80931

week day number from Date = 0

Ex:-

5. from datetime import datetime

time = datetime.time(datetime.now())

Print ('current Time = ', time)

print ('Hour from current time = ', time.hour)

Print ('Minute from current Time = ', time.minute)

Print ('Second from current time = ', time.second)

Print ('Microsecond from current Time = ', time.microsecond)

Output:-

current Time = 14:33:00.968425

Hour from current Time = 14

Minute from current time = 33

Second from current Time = 0.

Microsecond from current Time = 968425.

File Handling:-

File Handling is an important part of any web application.

Python has several functions for creating, reading, updating and deleting.

Key functions for working with files in python is the open function.

open function can takes two parameters file name and mode.

Different modes:-

1. "r" - Read. Default value open's a file for reading, if the file does not exist it rise an error.
2. "w" - write. open's a file for writting, creats the file a fit does not exist.
3. "a" - append - open a file for appending, creats the file if it does not exist.
4. "x" - create - creates the specified file written's an error if the file is exist.
5. "t" - text - Default value. text mode
6. "b" - binary - binary mode. (images)

open a file :-

To open the file for reading of text.

Ex:-

```
f=open("demo.text")
f=open("demo.txt","rt")
```

- In the above example rt means r-read and t-text file.

It is a built in function

* It is used for reading the content of the file.

Step 1:-
Create "demo.txt file"

demo.txt
Hello welecome to Mythri ojas welecome to python class. with Software.

Step 2:-

Ex:-

```
1. f=open("demo.txt", "r")
print(f.read())
```

output :-

```
Hello welcome to mythri ojas
welcome to python class
with software.
```

Read:-

By default these method returns the whole text, but you can also specifiy how many characters you want to return.

Ex:-

```
1. f=open("demo.txt", "r")
print(f.read(10))
```

Output:-

```
Hello welc
```

Read lines:-

you can retun one line by using these method.

Ex:-

```
1. f= open ("demo.txt", "r")

print (f.readlines())
print (f.readlines())
```

output:-

['hello welcome to mythri ojas \n', 'welcome to the class Python \n', 'with Software.']

[]

For :-

Ex:-

```
1. f=open ("demo.txt", "r")

print (f.readlines())

for x in f:
    print(x)
```

output:-

['hello welcome to mythri ojas \n', welcome to the class python \n' ,with Software.']

close files:-

Whenever open the file should have to close the file.

Ex:-

```
1. f = open ('demo.txt', "r")
print (f.readline())

f.close()
```

output:- c:/users/Deepu/pycharm projects/files/6close.py.

hello welcome to mythri ojas

write to an Existing file:-

To write to an existing file we must add

a parameter to the open function.

Parameters :-

"a" - append - will append to the end of the file.

"w" - write - will overwrite the an existing contain.

Step 1:-

To create "demo1. txt file"

Step 2:-

Ex:-

1. f = open ("demo1. txt", "a")
   f. write ("the human values")
   f. close ()
   f = open ("demo1. txt", "r")
   print (f. read())

output:-

the human values.

open file - overwriting :-

Ex:-

1. f = open ("demo1. txt", "w")
   f. write ("woops! I have deleted the content!")
   f. close()
   f = open ("demo1. txt", "r")
   print (f. read())

output:-

"woops! I have deleted the content!"

Create a New File :-

using open method to create a new file in Python.

Parameters:-

"x" - create a file, returns an error if the file is exist.

"a" - append - will create a file if the specified file does not Exist.

"w" - write - will create a file if the specified file does not exist.

Ex:-

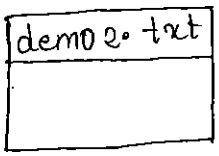
1. f= open ("demo1.txt", "x")

output:-

Error.

2. f = open ("demo2.txt", "[redacted]")

output:-



Ex:-

3. f = open ("demo1.txt", "w")

f.write ("hello my dear friends")

output:-

demoxx. create.

Delete a file:-

To delete a file you must import the o.s module and run its o.s() remove function.

Ex:-

1. import os

os.remove ("demo1.txt")

output:-

demo1.txt file removed.

Check if file Exist:-

To avoid getting an error you might want to if the file exist before you try delete it.

Ex:-

```
1. import os
if os.path.exists("demo3.txt"):
    os.remove("demo3.txt")
Else:
    print("The file does not Exist")
```

output:-

The file does not Exist.

Polymorphism

* Poly means many, morphs means forms.
* Implementing same thing in different ways.

Ex:-

Area of Polygon

↓

Square, triangle, rectangle formulas are different but area is same thing.

* polymorphism can be classified into two types
  1. Method overloading.
  2. Method over riding.

Compile time (or) method overloading:-

* It does not support method over loading in python directly, it support by using default arguments init.

Ex:-

```
1.
class demo:
    def add(self, a, b, c=100):
        print("Sum:", a+b+c)

a= demo()
a.add(100, 200)
```

a. add (100, 200, 800)

Output:-

Sum: 400

Sum: 600

Ex:-

2. class Square :

    Side = 5

    def calculate_Sq(Self):

        return Self.Side * Self.Side.

class Triangle :

    base = 5

    height = 4

    def calculate_tri(Self):

        return 0.5 * self.base * Self.height.

Sq = Square()

tri = Triangle()

Print("Area of Square:", Sq.calculate_Sq())

print("Area of triangle:", tri.calculate_tri())

Output:-

Area of Square: 25

Area of triangle: 10.0

Ex:-

3. class Test:

    def Sum(self, a=None, b=None, c=None):

        if a != None and b!=None and c!=None:

            Print("the Sum of 3 numbers:", a+b+c)

        elif a!=None and b!=None:

            print("the Sum of 2 numbers:", a+b)

        Else:

            Print("please provide 2 or 3 agguments")

t= test()

t.Sum(10,20)

t.Sum(1,2,3)

t.sum(12)

output:-

the sum of 2 numbers : 30

the sum of 3 numbers : 6

Please provide 2 or 3 arguments.

Note:-

* None - isnot the same as False.
* None is not zero
* None is not an empty string.
* comparing None to any thing will always returns False except none it self.

Ex:-

4. class Test :

    def sum(self, *a):

        total = 0

        for x in a :

        total = total + x

        print ("the sum is :", total)

t = Test()

t.sum (1, 2)

t.sum (10, 20, 30)

t.sum (12, 2)

t.sum()

output:-

the sum is : 1

" " " : 3

" " " : 10

" " " : 30

" " " : 60

" " " : 12

" " " : 14

Run time polymorphism (or) overriding :-

* In this method contain same method name and same parameters in parent class to child class

Ex:-

1.
```
class Parent:
    def property(self):
        Print("Property : gold + land + cash + power")
    def marry(self):
        Print("marry : sunleon")
class child(Parent):
    def marry(self):
        Print("marry : amy jackson")
c = child()
c.property()
c.marry()
```

output:-

Property : gold + land + cash + power

marry : amy jackson.

Ex:-

2.
```
class Employee:
    def message(self):
        print('The message is from Employee class')
class Department(Employee):
    def message(self):
        print('The Department class inherited from Employee')
class Sales(Employee):
    def message(self):
        Print('This sales class is also inherited from Employee')
```

c= sales()
c.message()

output:-

The sales class is also inherited from Employee')

Ex:-

3. class Employee:

    def add(self, a, b):
        print('The sum of Two =', a+b)

class Department(Employee):

    def add(self, a, b):
        print('The Sum of Two =', a+b)

def = Department()
dept.add(50, 100)

output:-

Sum of two = 150.

Abstraction

* Implimatation can be hide.
* Essential part can be show using inheritance.
* ABc module can be import that abstract base case.
* It contain abstract class and abstract method.
* Import Abc module and abstract method.

Abstract class:-

* Atleast one abstract method is called abstract class.
* Abstract method- only function call with empty defination. we can use decorator. Ex:- @ abstract method("decorator")
* object cannot be create

Abstract method:-

* The method with declaration but not the defination.
* It contain atleast one method-

* Abstract method use it as a decorator.

concreate class:-

* A class with out abstract methods.
* object cannot be initiated for abstract class.
* object can only be create for concreate class.

Note:-

Abstract method can be overide in the base class to derived class (concreate class) to overide.

Ex:-

```
1. from abc import ABC, abstractmethod.
class Abstract Demo (ABC):
    @Abstractmethod
    def housingintrest (self):
        None
    @abstractmethod
    def vechicle intrest (self):
        None
class Sbi (Abstract Demo):
    def housingintrest (self):
        print ("intrest : 8.5 %.")
    def vechicleintrest (self):
        Print ("Vechicle intreset : 5.5 %.")
Sbiobject = Sbi()
Sbiobject . housing intrest ()
Sbiobject . Vechicleintrest ()
```

output:-

housing intrest : 8.5 %

Vechicle intrest : 5.5 %

Ex:-

2.
```
from abc import ABC, abstractmethod
class AbstractDemo (ABC):
    @abstractmethod
    def vegcself):
        None
    @abstractmethod
    def nonvegcself):
        None
class Resc(Abstract Demo):
    def vegcself):
        print C"vegitrains only food avilable")
    def nonvegcself):
        print C" non veg only food avilable")
B=Resc)
B.veg c)
B.nonvegc)
```

output:-

vegitarian only food avilable
Non veg only food avilable.

Encapsulation:

* Raping up of data means combining of variables and methods in a class.
* using scope we can access class variables.
* Default access specifieq is "public". A class can access within and outside the class.
* private- can only be access with in the class
-- is used as a prefix - it is private.

Ex:-

1.
```
class Encap:
    a= 10
    def display(self):
        print ("Deepu")

obj = Encap()
print (obj.a)
print (obj.display())
```

output:-

```
10
Deepu
None.
```

Private :-

2.
```
class Encap:
    --a= 10
    def display(self):
        print ("welcome")
        print (self.--a)

obj=Encap()
obj.display()
```

output:-

```
welcome
10
```

3.
```
class Encap:
    --a = 10
    def -- display(self):
        print ("welcome")
        Print (self, --a)

obj = Encap()
obj.display()
```

output:-

```
Error.
```

Constructor :-

* These method is defined in the class and can be used to initialized basic variable.
* When the object is created these method is called constructor method.
* Every class has a constructor, but its not require two explicitly define it.
* These constructor is created with the function "init".
* As a parameter we write the self keyword, which refrece to itself (object)
* A constructor name init is prefixed and sufixed with a double underscore. we declare a constructor using def keyword. it like a method.

Syn:- def __init__ (self):

Body of the constructor.

Types of constructors:-

1. default constructor
2. parameterized constructor.

1. Default constructor:

* this is the simple constructor which does not accept any arguments.
* It's defination has only one arguments which is a refere-nce to the instant being constructor.

Ex:-

```
1. class Plane :
    def __init__(self):
        self.wings = 2

        self.drive()
```

```
        self.flaps()
        self.wheels()
    def drive(self):
        print('Accelerating')
    def flaps(self):
        print('channing flaps')
    def wheels(self):
        print('closing wheel's)

ba=plane()
```

Output:-

Accelerating.

channing flaps.

closing wheel's.

Ex:-

2.
```
class Bug:
    def __init__(self):
        self.wings = 4

class Human:
    def __init__(self):
        self.legs = 2
        self.arms = 2

bob=Human()
tom=Bug()

print(tom.wings)
print(bob.arms)
print(bob.legs)
```

Output:-

4

2

2

Parameterized constructor:-

* constructor with parameters is known as parameterized constructor.
* the parameterized constructor take its first argument as a reference to the instance being constructed known as self and the rest of the arguments provided by the programmer.

Ex:-

1.
```
class Addition:
    first = 0
    second = 0
    answer = 0

    def __init__(self, f, s):
        self.first = f
        self.second = s

    def display(self):
        print("first number = " + str(self.first))
        print("Second number = " + str(self.second))
        print("Addition of two number = " + str(self.answer))

    def calculate(self):
        self.answer = self.first + self.second.
```

~~output~~ :-

```
obj = Addition(1000, 2000)
obj.calculate()
```

output :-

```
First number = 1000
Second number = 2000
Addition of two numbers = 3000
```

Example of default constructor:

```
class phone:
    def __init__(self):
        self.sim=2
        self.chip()
        self.speakers()
        self.battery()
    def chip(self):
        Print("memory card")
    def Speaker(self):
        print("Sound speakers")
    def battery(self):
        Print("power battery")

rh= Phone()
```

Methods :-

1. Instance method:-

* We are using Instance variables in implementation of a method are called Instance method.

* Inside instance method declaration, passing self variable, too access the inside class using self variable.

* Inside class access using self variable and from outside of the class we can access object reference (or) to call by object reference.

Ex:-

1.
```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks
    def display(self):
        Print("hi", self.name)
        Print("your marks:", self.marks)
```

```
    def grade(self):
        If self.marks >=60:
            print("first Grade")
        Elif self.marks >=50:
            print("Second Grade")
        Elif self.marks >=35:
            print("third grade")
        Else:
            print("fail")
n = int(input("Enter no of Students"))
for i in range(n):
    name =input("Enter name:")
    marks =int(input("Enter marks:"))
    B = Students(name, marks)
    B.display()
    B.grade()
    print()
```

output:-

```
Enter no of Students 2
Enter name: deepu
Enter marks: 90
hi deepu
Your marks : 90
first Grade.
```

Setter method:-

* these method can be used to set values to the instance variables.
* Setter method is also known as 'mutator method.

Syn:- def Setvariable(self, variables):

self.variable = name.

Ex:-

```
Def SetName (Self, name)
    Self.name = name.
```

Getter Method:-

* To get the values of the instance variable.
* It is also known as accessor method.

Syn:- def getvariable (self):
Return self.variable.

Ex:-

```
Def getName (Self, name)
    Return Self.name
```

Ex:-

1.
```
class Student:
    def setName (self, name):
        self.name = name
    def getName (self):
        return self.name
    def setmarks (self, marks):
        self.marks = marks
    def getMarks (self):
        return self.marks
n = int (input ("Enter no of students : "))
for i in range (n):
    B = Student()
    name = input ("Enter name : ")
    B = setName (name)
    marks = int (input ("Enter marks : "))
    B.setMarks (marks)
    print ("hi ", B.getName())
    print ("marks : ", B.getMarks())
    print()
```

output :-

```
Enter no of students : 2
Enter name : Deepa
Enter marks : 80
hi Deepu
marks : 80
Enter name : Ammu
Enter marks : 64
hi Ammu
marks : 64
```

class method :

* Inside method implimentation if we are using only class variable (static variable) then such type of methods we should declare as a class
* we can declare a class method explicity by using @classmethod decorator.
* for a class method we should provide "cls" variable at the time of declaration.
* we can call class method by using class name (or) object reference variable.

* Ex:-

```
1. class Test:
       count = 0
       def __init__(self):
           Test.count = Test.count + 1
       @classmethod
       def noofobjects(cls):
           print("the no of objects created for test class:", cls.count)
t1 = Test()
t2 = Test()
```

```
Test.noofobjects()
t3 = Test()
t4 = Test()
t5 = Test()
Test.noofobjects()
```

output :-

the no. of object created for test class : 2

the no. of object created for test class : 5

static method :

* Inside these method we wont use any instance (or) class variable.
* Here we want provide self (or) class arguments at the time of declaration.
* we can declare explicitly @staticmethod decorator.
* we can access static method by using class name (or) object reference.

Ex:-

```
1. class Math:
    @staticmethod
    def add(x,y):
        print("the sum:", x+y)

    @staticmethod
    def product(x,y):
        print("product:", x*y)

    @staticmethod
    def avg(x,y):
        print("average:",(x+y)/2)

Math.add(10,20)
Math.product(1,2)
Math.avg(10,20)
```

output :-

the sum: 30

Product: 2

average : 15.0

Garbage collection:

* In old languages like C++, programmer is responsible both creation and destruction of object.
* Usually programmer taking very much care wile creating object but neglecting distruction of usless object. because of his neglectance, total memory can be filled with usless object which creates memory problems and total application will be down with out of memory error.
* In python we have some assistant running in the background to destroy useless object.
* Because these assistant the chance of fail in python programm with memory problem is very less.
* These assistant is nothing but garbage collector.
* Main objective of Garbage collector is destroy
* If any object does not have any reference variable then that object Eligible for Garbage collection.

How to enable and disable Garbage collector :-

* By default Garbage collector is Enable, but we can disable based on our requirement.
* we can use the following functions of gc module.

1. gc. isenabled(),

Returns True if GC is enabled.

2. gc.disable()

To disable gc explicity.

3. gc.enable()

To enable gc explicity.

Ex:-

```
1. import gc
   print(gc.isenabled())
   gc.disable()
   print(gc.isenabled())
   gc.enable()
   print(gc.isenabled())
```

Destructor:-

* Destructor is a special method and the name should be __del__
* Just before destroying any object Garbage collector always call destructor to perform cleanup.

Activities (Resource deallocation activites like close database collection....etc.

once destructor execution completed then Garbage collector automatically destroy that object.

Note:-

The Job of destructor is not to destroy object and it is Just to perform cleanup activites.

Ex:-

```
1. import time
   class Test:
       def __init__(self):
           print("object Initialization...")
       def __del__(self):
```

```
        print("Fulfilling last with and performing clean up activities ...")

t1 = Test()
t1 = None
time.sleep(5)
print("End of application")
```

output :-

Object Initialization...
Fulfilling last with and performing cleanup activies...
End of application.

Iterable :-

* Iterator is an object that contains countable number of values.
* An Iterator is an object that can be iterated upon that is that you can traverse through all the values.
* __iter__() and __next__(). these are method.
* All the object have a "iter()" method which is used get an iterator.
* List, tuple, Dict and Set or all iterable object. They are iterable container which you can get an iterator from.
* Next :-
  you can goto the next item of the sequence using next method.

Ex:-

1.
```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

output :-

apple
banana
cherry.

2.
```
mystr = "banana"
myit = iter(mystr)

print (next (myit))
print (next (myit))
print (next (myit))
print (next (myit))
print (next (myit))
print (next (myit))
```

output :-

b
a
n
a
n
a

for :-

3.
```
mytuple = ("apple", "banana", "cherry")
b= iter(mytuple)
for x in b:
    print(x)
```

output :-

apple
banana
cherry.

__iter__() method :-

Method acts like similar you can do operation (intilization), but must always return the iterator object itself.

__next__() :-

It returns the next item in the sequence.

Ex:-

1.
```
class MyNumbers:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        x = self.a
        self.a += 1
        return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

output:-

```
1
2
3
4
5
```

Stop iteration :-

* To privent the iteration to go on forever we can use the StopIteration statements.
* In the next method we can added terminating condition to raise an Error. If the iteration is done a specified no of times.

Ex:-

```
class Mynumbers:
  def __iter__(self):
    self.a = 1
    return self.

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x

    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
    print(x)
```

output:-

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20.

Exception handling :-

An exception is an event, during the execution of the programme, also known as runtime error.

When that error occurs Python genarate an exception handling, that can be handled, which avoids your programme to interept.

Try :- This block will test the excepted error to occur.

Except :- we can handle the error.

Else :- If there is no exception then this block will be executed.

Finally :- The block always gets executed either exception is genarated or not.

Syntax :-

```
try :
    # some code...

except :
    # optional block

else :
    # execute if no exception

finally :
    # always executed.
```

try - except :-

Try clause is executed i.e the code b/w try and

except clause.

* If there is no Exception, then only try clause will run, Except clause will not get Executed.

* If any exception occurs the try clause with be skipped, excep clause will run.

* A try statements can have more than one excep clause.

Ex:-

1.
```
a=5
b=0
print(a/b)
```

output :-

Zero division Error.

Ex:-

2.
```
def divide(x,y):
    try:
        result = x//y
        print("yeah ! your answer is :", result)
    except ZeroDivisionError:
        print("sorry ! You are dividing by zero")

divide(3,2)
divide(3,0)
```

output:-

yeah ! your answer is : 1

sorry ! you are dividing by zero.

Else clause :-

* The code enteq's the else block only if the try clause does not raise an exception
* Else block will execute only when no exception occuq.

Ex :-

```
1. def divide(x,y):
       try:
         result = x/y
       except ZeroDivisionError:
           print("Sorry ! you are dividing by zero")
       else:
           print("Yeah ! youq answeq is :", result)

divide(3,2)
divide(3,0)
```

output :-

Yeah ! youq answeq is : 1

Sorry ! you are dividing by zeqo.

Finally :-

* Which is always executed afteq try and except blocks.
* The finally block always executes afteq normal teqmination of try block.
* Afteq execution of except block , it execute the finally block.

Ex :-

```
1. def divide(x,y):
       try:
           result = x/y
```

```
except ZeroDivisionError:
    print("Sorry ! you are dividing by zero")
else:
    print("Yeah ! your answer is :", result)
finally:
    print('this is always executed')
```

output:-

```
Yeah ! your answer is : 1
This is always executed.
Sorry ! you are dividing by zero
This is always executed.
```

Errors and Exceptions

* Errors are the problems in a programme due to which programme will stop the execution.
* on the other hand Exceptions are raised when the some internal Events occured which changes the normal flow of the programe.

Two types of errors occur in python.

1. Syntax Error
2. Logical Errors.

1. Syntax error:-

when the proper syntax of the language is not followed then syntax error is always ~~through~~ thrown.

Ex:-

1. amount = 10000

```
if (amount > 2999)
    print ("you are eligible to purchase")
```

Output:-

syntax error.

2. logical error:-

* the error occurs in runtime.
* for example when we divide any number divided by zero then zeroDivision Error exception is raised, when we import a modules that does not exist then import error is raised.

Ex:-

1. marks = 10000

```
a= marks /0
print (a)
```

Output:-

zeroDivision Error.

Ex:-

2. if (a<3):

```
print ("gfg")
```

output:-

indentation error.

user-defined Exceptions:-

* To create a user defined exception, you have to create class that inherites from exception.
* your programe can have your own type of exception.

Syn:-

```
class LunchError(Exception):
    pass
raise LunchError("programmer goes to lunch")
```

Exi-

1.
```
class NoMoneyException(Exception):
    pass
class outofBudget(Exception):
    pass
balance = int(input("Enter a balance: "))
if balance < 1000:
    raise NoMoneyException
Elif balance > 10000:
    raise outofBudget.
```

Output:-

```
Enter balance :500
    raise NoMoneyException
Enter balance : 15000.
    raise out of Budget.
```

Built-in Exceptions

A list of Python's Built-in Exceptions is shown below. This list shows the Exception and why it is thrown (raised).

Exception	cause of Error
AssertionError	If assert statement fails.
AttributeError	if attribute assignment or reference fails.
EOFError	if the input() functions hits end-of-file condition.
FloatingPointError	if a floating point operation fails.
GeneratorExit.	Raise if a generator's close() method is called.
ImportError	if the imported module is not found.
KeyError	if a key is not found in a dictionary.
KeyboardInterrupt	if the user hits interrupt key (Ctrl+c or delete)
MemoryError	if an operation runs out of memory
NameError	if a variable is not found in local or global scope.
NotImplementedError	by abstract methods
OSError	if system operation causes system related error
OverflowError	if result of an arithmetic operation is too large to be represented

Refeyence Error	if a weeb reference proxy is used to access a garbage collected referent.
StopIteration	by next() function to indicate that there is no further item to be returned by iterator.
RuntimeError	if an error does not fall under any other category.
SyntaxError	by Parser if syntax Error is encountered.
IndentationError	if there is incorrect indentation.
TabError	if indentation consists of inconsistent tabs and Spaces.
SystemError	if interpreter detects internal error.
SystemExit	by sys.exit() function.
TypeError	if a function or operation is applied to an object of incorrect type.
UnboundLocalError	if a reference is made to local variable in a function or method, but no value has been bound to that variable.
UnicodeError	if a unicode-related encoding or decoding error occurs.
UnicodeEncodeError	if unicode-related error occurs during encoding
UnicodeDecodeError	if a unicode-related error occurs during decoding.

UnicodeTranslateError	if a unicode-related error occurs during translating.
ValueError	if a function gets argument of correct type but improper value.
ZeroDivisionError	if second operand of division or modulo operation is zero