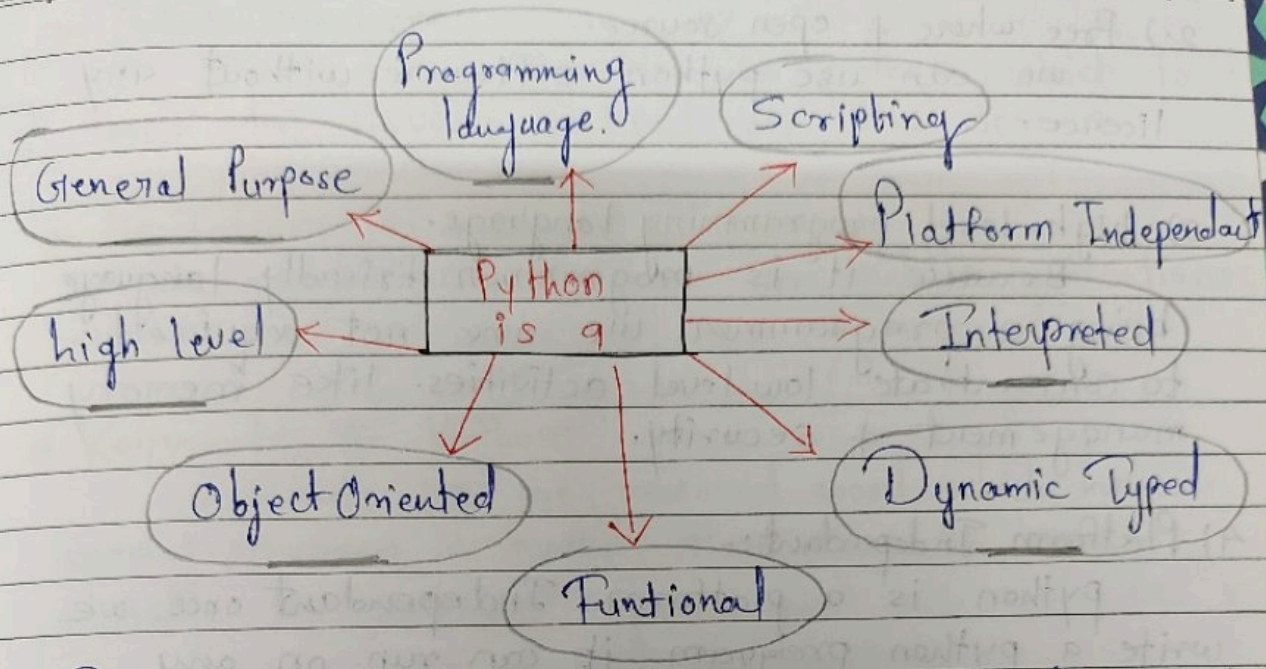
PYTHON

18/01/2024
59



* Python is a General Purpose, high level, Object Oriented Dynamic typed, Interpreted, platform Independant, programming language.
* Which is develop by Guido Van Rossum in the year of 1989 at NRI at Netherland. NRI (National Research Institute).
* The first version of python in 20th Feb 1991. Is called 0.9.

* Application of Python.
For develop web application
For develop desktop application.
For develop database application.
For develop network application
For develop Data science application
For develop ML/AI Robotics application.

* Features of python:
1. Simple & easy to learn.
* When we read python program we can feel like reading english statement.

2.) Free where & open source:
we can use python software without any licence.

3) high-level programming language.
Because it is programmer friendly language being a programmer we are not required to contsantrate low-level activities. like memory management & security.

4) Platform Independant:-
python is a platform Independant once we write a python program it can run on any platform without rewriting once again. Internally PVM is responsible ~~without~~ ~~now~~ to convert into machine understandable form.

5) Dynamically Typed:-
There are no need to define type of data that's why its dynamically typed.

6) Procedure & Object Oriented Programming language:
Python language support procedure oriented language like c programming & object oriented like c++

7) Interpreted Programming language:
Compiler & interpreter. its execute code line thats why is called interpreted language.

8) Extensive Library:-
Python has a rich inbuilt library.

* Print Function:
Print is a function in python which is used to display output on the screen.

* limitations of python:
We cannot use python for mobile applications. python slow as other c, c++ & java.

* Keywords In Python:
keywords are reserved word which have special meaning in python there are 86 keywords available in python.

* How to print keywords in python.
* There are 35 keyword in python currently.
Impot keyword
Print (keyword.kwlist)

- Variables :-
Variables are the memory location which is used to store data. starts with only alPhabates -(undersco) & $ dollar sign.

Python Interview Questions:

Q.1 o What is python
Q.2 o features of python
Q.3 o limitation of python
Q.4 o Applications areas or application of python
Q.5 o is python dynamic typed? if yes how?
Q.6 o Is python interpreted? How?
Q.7 o What are the keyword in python?
Q.8 o How many keywords are available in python.
Q.9 o What are the current version in python. 3.10
Q.10 o What is variable in python?
Q.11 o Who develop python.

* Write a print even Number between 0 to 100. in python.

```
for num in range (0,101):
    if num % 2 == 0:
        print (num)
```

range (start, end, start)
(1, 11)
(1, 11 - - - - - -),

* Int:- We can use int datatype to represent whole number. Int function is use to convert other data type into integer.

- Ex:-
```
a = 10
print (type (a))
```

- For binary:-
```
print (bin(a))
```

- For hexadecimal:-
```
print (bin (a))
```

- for octal:-
```
print (oct (a))
```

* Float:- It is used to store decimal value. we can also floating point value by using exponential form

- Ex:-
```
a = 10.3
prit (type(a))
```

* Complex:- It is used to store imaginary value.

Ex:-

```
a = 2+3j
print (type(a))
```

* Boolean:- Booleans represent any of two values. True or False.

* Comments:-

* Type Casting:-

We can comment any type of values into another type, this type is called type casting.

→ Int:- We can comment another data type into int datatype except complex datatype. if we want to comment str data type into int type then string should be specified in base 10 (decimal value).

→ float:- we can convert other data type into float datatype except complex datatype.

* Complex Function:-

We can use complex function to convert other datatype into complex datatype.

Example:-

```
Print (complex (10))
```

output

```
10+0j
```

* Bool :-

We can use to convert other datatype to bool data type.

```
a = 10
if a=10:
    print (bool (a))
```

of total number of value = 0

- If both real or imaginary part are zero then it is false other wise true.

- If s- contain same string then it is true, otherwise it display false.

In python if a new object is required then PVM (Python virtual machine)

Self :-
Practice


```
For i in range (1,6):
    for j in range (1, i+1):
        print (j, end="")
    print ("\n")
```

```
1
12
123
1234
12345
```

```
For i in range (1,6):
    For j in range (1, i+1):
        print ("*", end="")
    print ("\n")
```

```
*
**
***
****
*****
```

```
for a in range (4,0,-1):
    For b in range (1,8):
        if (b>=5-a and b<=3+a):
            print ('*',end="")
        else:
            print (' ',end="")
    print ()
```

```
* * * * *
 * * * *
  * * *
   * *
    *
```

```
For a in range (1,5):
    For b in range (1,8):
        if (b>=-a and b<=3+a):
            print ('*', end="")
        else:
            print (' ',end")
    print()
```

```
*
* *
* * *
* * * *
* * * * *
```

```
no_row=int (input("enter the row number :"))
for row in range (0,no_row+1):
    For b in range (1,no_row+1):
        print (" ",end="")
    For column in range (1,(2*row-1)+1):
        print("*", end="")
    print ('\n')
```

```
*
* *
*   *
*     *
*       *
```

* Operators in python:

1] Arithematic Operation:-

→ +, -, *, /, //, %, **

2] Comparison operators:/ Relational Operators:

→ <, >, =<, >=

3] Logical Operators:

→ And, OR, Not.

* Assignment Operators in python:

1) =
2) += a+=1 → a=a+1 a+=2 → a=a+2
3) -= a-=1 → a=a-1 a-=2 → a=a-2
4) *= a*=2 → a=a*2 a*=1 → a=a*1
5) == a==1
6) /=

* Special Operators:

1] Identity Operators:

Identity operators are used for addressed comparison. There are two identity operator in pyth

1) Is
2) Is Not

- Is = It returns true both variable are pointing to same object.

- Is not = Returns true if both variables are the same object.

2] Membership Operator:- In , Not in

We can use membership operators weather the given object present in the given ~~condition~~ collection.

```
s = 'hello'
print ('e' in s)  # True
print ('e' not in s) # False.
```

In = In return true if given object present in specified collection.

Not in = if given object is not present in specified collection it returns true.

* Bitwise Operators:

1) Bitwise and - (&)

if both bits are one then result is one otherwise zero.

→ Print (10 & 8)

8

→ Print (14 & 13)

12

2) Bitwise OR - (|)

if atleast one bit is one result will be one otherwise zero.

→ Print (8 | 10)

10

3) Bitwise XOR (^)

if both bits are different then result is one otherwise zero.

→ Print (8 ^ 10)

2

* Not (~) :- Invert all the bits.

* Bitwise left shift (<<) :-
shift left by pushing zeros in from the right and left the leftmost bits fall of

This operators shifts the bits of a number to the left by a specified number of operators

→ Print (10 << 2) # 40

* Bitwise Right shift (>>) :-
This operators shifts the bits of a number to the right by a spacified number of position.

→ Print (10 >> 2) #

* Input Function :-
It is used to take input from user.

Q. Write a program to take input from user and perform addition on that number.

→ a = input ('enter value of a')
b = input ('enter value of b')
print (a, b)

Ans = enter value of a

* String

* list

list is ordered, hetrogenious, mutable collection of items defined in [] square brackets.

```
→ 1 = [1, 2.0, 'hello']
  Print (1)
```

* Nested forloop:-

forloop inside foorloop is called nested forlop.

```
ex:- for i in range (4):
        For j in range (4):
            print (i);

end:- For i in range (3):
        For j in rang (3):
            print (i,j)
```

```
* For i in range (6):
      For j in range (6):
          if (j<=i):
              print ("*", end = ")
      print()
```

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

```
* n = 1
  for i in range (5):
      For j in range (i):
          print (n, end = '')
          n += 1
  print ()
```

* While loop:-

Write a program to fetch 1 to 100 numbers using while loop.

Ex:-
```
num = 1
while num <= 100:
    print (num)
    num += 1
```

* Write a program to fetch Even

Ex:-
```
num = 2
while num <= 100:
    print (num)
    num += 2
```

* Controlling Statement / Transfer Statement:

1. Break
2. Pass
3. Continue

1. Break:- We can use break statement inside loops to break loop execution based on some condition. Break statement used to terminate current look.

Ex:-
```
for i in range (11):
    if i == 6:
        break
    print (i)
```

*
```
n = 2
for in range (4):
    for j in range (i):
        print (n*n, end = '')
        n = n+1
    Print ('')
```

*
```
4
9 16
25 36 49
```

```
n = 2
for i in range (5):
    for j in range (i):
        print (n, end = '')
        n+=2
    print ()
```

2
4 6
8 10 12
14 16 18 20

* Transfer / Controll Statement
* Continue Statement :-

Continue statement it is used to skip the current itteration and move to next itteration.

* Pass Statement :-

Pass is a null statement in python. which is used to define empty block.

```
x = 1
if x > 10:
    pass
Print (x)
```

* Practice :-

1.
```
n = 5
for i in range (n):
    for j in range (n):
        print ('*', end='')
    Print ()
```

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

2.
```
n = 5
for i in range (n):
    for j in range (i+1):
        print ('*', end='')
    print ()
```

```
*
* *
* * *
* * * *
* * * * *
```

3. n=5

```
for i in range (n):
    for j in range (i,n):
        print ('*', end= '')
    print ()
```

```
* * * * *
* * * *
* * *
* *
*
```

4. n=5

```
i=1; j=0
while (i<=n):
    while (j<=i-1):
        print ("* ", end= " ")
        j+=1
    print ("\r")
    j=0; i+=1
```

```
*
* *
* * *
* * * *
* * * * *
```

(In print statement after * take space for space in each star print in result statement).

5. n=5

```
for i in range (n):
    for j in range (i,n):
        print (' ', end= '')
    For j in range (i):
        print ('*', end= '')
    for j in range (i+1):
        print ('*', end= '')
    print()
```

```
1  *
3  * * *
5  * * * * *
7  * * * * * * *
9  * * * * * * * * *
```

6. n=5

```
for i in range (n):
    for j in range (i+1):
        print (' ', end= '')
    for j in range (i,n-1):
        print ('*', end= '')
    For j in range (i,n):
        print ('*', end= '')
    print ()
```

```
* * * * * * * * *
* * * * * * *
* * * * *
* * *
*
```

String:-

String is a sequance of characters which is enclosed in single cot (' ') & double (" ").
("" "") → Multiline - docstring.

Ex: s= 'hello'
print (type(s))

For indexing value:
s1 = 'Nahida'
print (s1[1])
a

H	E	L	L	O
1	2	3	4	5 end (n-1)

- Indexing start in zero [start : end : step]
indexing [1]
Slicing - Piece

- s= "python is programming language."
print (s[0:6])
python.

- length:- length is used for count length of data when we using indexing type (len-1).

* Negative Indexing:-
Negative indexing start with -1.
s= 'hello'
print (s[-5:1-1])
print (s[-5::])
print (s[::,-1])

-5	-4	-3	-2	-1
H	E	L	L	O
-4	-3	-2	-1	-0

s= Python is a programming language.
print (s[15:23]) — gramming
print (s[8:11]) — ~~Possitive~~ s a
print (s[22:32]) — g language
print (s[2:11]) — thon is a

```
print (s[-17:-9])        gramming
print (s[-24:21])        s a
print (s[-30:-21])       thon is a
print (s[-10::])         g language
```

Write a program to reverse string ~~using~~ without using slicing & indexing.

```
s = ''                      s = ' '
s1 = 'Nahida'               s1 = 'Nahida'
for i in s:                 for i in s:
    s1 = i + s1                 s1 = s1 + i
    print (s1)                  print (s1)
```

* Reverse

Accessing Element using forloop:-

```
s = 'python is a programming language.
print (s[0:5:2])        } Positive
print (s[15:22:3])
print (s[-32:-27:2])    } Negative.
print (s[-17:-10:3])
```

```
*  a = 'hello'        #  h
   for i in a:           e
       print (i)         l
                         l
                         o
```

1.

* Length Function:

Length function is used to calculate length of sequance

```
s = 'python'
print (len (s))    #6
```

② * Reverse Statement:-
It is used to reverse string.
s= 'python'
print (reversed (s))

* s1= 'python'
s1= reversed (s1)

for i in s1:
print i

n
o
h
t
y
p

* Mathematical Operations in string: (+)
1) Concatination (+) of string
⇒ Print ('hello' + 'world')
helloworld

To spacing
⇒ Print ('hello' + ' ' + 'world')
Hello world

* Multiplication (*)
In multiplication we need one int & one str.
print ('hello' * 4)

* Comparison of String:-
print ('hello' > 'apple') # 'hello' True
print ('Apple' > 'apple') # 'apple' False

A to Z - 65 - 91
a to Z - 97 - 122 } Ass value.

8. Cheking membership member.

s= 'hello'
print ('h' in s) True

* Print built in string.

③ 1] Find :- Find function returns index of first occurances of the given substring. if it is not available written -1.

Syntax:

S. Fine (substring)

Ex:-

```
s='python'
print (s.Find('p'))
# 0
print (s.find ('a'))
#  -1
```

④ * 2.] Index function:

Index function is exactely same as find function. but difference is that if substring is not available it will give value error.

Ex:-

```
s='Apple'
print (s. index ('p'))
print (s. index ('b'))
```

⓪

* Count Function

We can find number of occurances of substring present inthe string.

Syntax:

S. count (substr)

Ex:

```
s='itvedant'
print (s. count ('t'))
```

* using without count function:

```
• s='aaabbbccab'
  n= 0
  s1 = ' '
  for s1 in s:
      if s1 == 'a':
          n+ =1
  Print (n)
# 4
```

```
• s='aaabbbccab'
  n=0
  for i in s:
      if (i=='a'):
          n+=1
  Print (n)
```

5.

* Replace Function :-

Syntax:

s.replace (old_str, New-str)

* Replace function is used to replace the substring.

Ex. *
```
s='aaabbbccab
print (s.replace ('a', 'z'))
```

• Replace without replace function:

```
s= 'aaabbbccab'

s1 = ''
for s1 in s:
    if s1 == 'a':
        print ('z', end= '')
    else:
        print ('s1, end = '')
```

6. Split Function:-

We can split the given string according to spacified seprater by using split method.

- The default seprater is space.

syntax:

s.split (seprator)

Ex:

```
s= 'itvedant private limited pune'
print (s.split ())
# ['itvedant', 'Private', 'limited', 'Pune']
print (s.split (t))
# ['i', 'vedan', 'Priva', 'e limi', 'ed pune']
```

Q1. Reverse internal content in string.

7. JOIN FUNCTION

We can a join group of strings (~~tits~~ list or tuple) with respect to given seprater.

- Syntax:

s= seprator.join (group of string)

- Ex:-

```
l= ['Itvedant', 'private', 'limited']
s=' '.join (l)
print (s)

# itvedantprivatelimited.
```

for space write

```
l= ['itvedant private limited']
```

06/02/2024.

Changing Case of substring:

1) Upper()
2) lower()
3) Swapcase()
4) title()
5) capitalise()

(8)

1) uppercase():

It returns or all characters convert into upper case

syntax:

s. upper()

(9)

2) lower():

It returns all character into lower case.

syntax:

s. lower()

Examples:

Q. s= Learning python is very easy.

⇒ print (s. upper()) # LEARNING PYTHON IS VERY EASY.

⇒ print (s. lower()) # learning python is very easy.

(10)

Imp.

3) Swapcase():

It converts all uppers character into lower.

Syntax:

s. swapcase()

(11)

4) title case():

To convert all character into title case. that is first character in every word should be upper.

Syntax:

s.title()

ex:- Learning Python Is Very Easy

(12)

5) Capitalize() :

only first character will be converted to upper case. and all remaining characters can be converted to lower case.

syntax:

s.capitalize()

Q.1. Write a program ~~swap~~ to swap 2 variables

```
* a=10
  b=20
  temp = a
  a=b
  b = temp
  print (a)
  print (b)
```

```
* a = 10
  b = 20
  a = a+b      # 30
  b = a - b    #
  a = a - b    # 20
  print (a) # 20
  print (b) # 10
```

* Checking Starting & ending part of the string:

* Checking :-

(13) 1.) starts with ()

(14) 2.) ends with()

Ex:-

```
s= "Learning python is very easy"
print (s. startswith ('Learning'))    # True
print (s. endswith ('M'))             # False.
```

* To check type of character present in string.

(15) 1] Isalpha():- Returns true if all characters are in alphabate symbol (A-Z, a-z)

(16)

2] Isdigit ():- Returns true if all number are in 0-9:

```
s = '123'                    # True
```

(17)

3] Isalnum(): It returns true if alphanumeric value in string. (both alpha & num).

```
s = 'learning 123'
print (s.Isalnum())        # True.
```

(18)

4] Istitle():- It returns first characters in string is capital case/uppercase.

```
s='Learning python is very easy'
print (s.title())
# Learning Python Is Very Easy.
```

(19)

5] isspace()

It returns to if string content only space.

```
ex:- s=' h'
     print (s.isspace())
# False
```

(20)

6] isupper(): It returns all character in string in capital case: True or False

```
s= 'Learning'          # False
print (s.isupper())    # LEARNING
```

(21)

7] islower(): It returns all characters in string is lower case: True or False.

```
s='LEARNING'           # True
print (s.islower())    # 'learning'
```

Q1. Write a program

```
s= '~~a4~~ a4 b3 c2'
output = aaaabbb cc
```

```
# string1 = 'a4b3c2'
  for i in string1:
      if (i.isalpha()):
          print(i)

x = " "
for j in string1:
    if (j.isalpha()):
        s1 = x + j
    elif (j.isdigit()):
        s2 = int(j)
        print(s1 * s2, end = " ")
```

* chr() :- Print (ord('a')) # 97

* ord() Print (chr('65') # A

65 to 91 = A to Z
97 to 122 = a to z

Task:

```
s = 'a4m3b2'
⇒ s1 = ' '
  for i in s:
      if i.isalpha():
          s1 = s1 + i
          p = i
      elif (i.isdigit()):
          s1 = s1 + chr(ord(p) + int(i))
Print(s1)
```

* List Datatype:-

If you want to represent group of individual objects as a single entity where insertion order pre-reserved and duplicates are allowed. then we should go for list.

list is a collection of element which content duplicates objects, hetrogenuous object (multiple type of data) ~~which~~ types ~~is~~ of ~~data~~ enclosed represent [ ].

* Accessing list element using forloop:

```
l= [1, 2, 3, 4, 5, 6]        #  1
for i in l:                     2
    print (i)                   3
                                4
                                5
                                6
```

To print odd number.

```
l=[1, 2, 3, 4, 5, 6]        # 2
For i in l:                   4
    if i % 2==0:              6
        print (i)
```

1. len Function:- It is used to return number of the element present in the list.

```
# Print (len (l))   #6
```

2. Count Function: We can find number of occurance of ~~list~~ spacified item inthe list.

```
# Print (l.count (a))   #4
```

3. Index Function: If spacified item not found

4. Append:

Appand function it is used to add element at the ~~list~~ end of the list

syntax:

list - append (element)

Ex:

```
l = [10, 20, 30, 40]
l. Append (50)
  print (l)
# [10, 20, 30, 40, 50]
```

5. Insert Function:

It is used to insert item at a spacific index position.

syntax:

list.insert(3, 45)

Ex:

```
l = [10, 20, 30, 40, 50]
l. insert (3, 60)
print (l)                # [10, 20, 30, 60, 40, 50]
```

6. Extend Function:

It is used to add all element of one list to another list.

Ex:

```
l1=[10, 20, 30, 40, 50, 60]
l2=[70, 80, 90]
l. extend (l1)
print (l1)

# [10, 20, 30, 40, 50, 60, 70, 80, 90]

# Mylist = [10, 20, 30, [5, 25, 55], 40, 50]
Print (mylist [3][0])        # 5
Print (mylist [3][1])        # 25
Print (mylist [3][2])        # 55
```

* Find maximum number
[10, 11, 12, 13, 15, 6, 8]

7. Remove function:

We can use remove function to remove spacified item from the list. if item present multiple time then only first occurance will be removed.

```
l = [10, 20, 30, 40, 50]
l = remove (30)
print (l)

# [10, 20, 40, 50]
```

8. Pop-function:()

It is used to remove last element from the list.

This is only function which manupulates lists and return some element.

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

Task

```
l = [10, 20, 30, 400, 200, 150, 180]
```

Q. wap to print highest element from l without using max() function:

```
m = l[0]
for i in l:
    if i > m:
        m = i
print (m)

# 400
```

Q. wap to print lowest element from l without using min() function:

```
m=l[0]
for i in l:
    if i<m:
        m=i
Print (m)
```

#10

9.] Reverse Function ():-

It is used to reversed order of ~~list~~. element of list.

Ex:-

```
l=[10,20,30,40,50]
l.reverse()
print (l)
```

[50,40,30,20,10]

Q. Write a program to reverse a list without using reverse function. using slicing function:

```
l=[10,20,30,40,50]
Print (l.[::-1])
```

[50,40,30,20,10]

```
l=[10,20,30,40,50]
l1=[]
For i in l:
    l1.insert (0,i)
print (l1)
```

[50,40,30,20,10]

10. Sort Function:

if you want to sort the element of list according to default natural sorting order then we should go for sort method.

It is used to sort list

by default assending

- for number = default natural sorting order is assending.
- For string = default natural sorting order is alphabetical order.

Ex:-

```
l=[10, 20, 30, 40, 5, 3, 100]
l.sort()
print(l)
# [3, 5, 10, 20, 30, 40, 100]
```

Reversed:-

```
l=[10, 20, 30, 40, 5, 3, 100]
l.sort(reversed=True)
print(l)
# [100, 40, 30, 20, 10, 5, 3]

l=['python', 'is', 'programming', 'language']
l.sort()
print(l)
# ['a', 'is', 'language', 'programming', 'lang']
```

Python

Note:- To use sort function compulsary list should contain only homogenious element.

11. Clear function():- is used to remove all element from the list. only contain deleted.

Ex:-

```
l=[1, 2, 3, 4, 5]
l.clear()
print(l)
```

Aliasing & Cloning:-

* Aliasing:-

A process of giving another reference variable to the existing list called aliasing.

Ex:-

```
l = [10, 20, 30]
l1 = l
print (l1)
# [10, 20, 30]
```

- The problem in this approach by using one reference variable if we are changing content/element. then those changes will be reflected to the other reference variable. to overcome this problem we should go for cloning.

* Clonning:- slicing & copy

We can empliment cloning by using slicing operator and copy method function.

```
l = [10, 20, 30, 40]
l1 = l[:]  → slicing
print (l1)
l1 [2] = 20
print (l, l1)
# [10, 20, 30, 40] [10, 20, 20, 40]
```

```
l = [10, 20, 30, 40]
l1 = l.copy()  → copy
print (l1)
l1[2] = 20
print (l, l1)
# [10, 20, 30, 40] [10, 20, 20, 40]
```

* list comprehension:

Creating list object from any itterable object.

- Syntax:

list = [expression for i in iterable-object if condition]

- Task :- Even number.

```
l=[1,2,3,4,5,6,7,8,9,10]
l1=[]
for i in l:
    if i%2==0:          i%2!=0  odd
        l1.append(i)
print(l1)
# [2,4,6,8,10]
```

Example 2:-

```
l=[1,2,3,4,5,6,7,8,9,10]
l1=[i for i in l]
print(l1)
# [1,2,3,4,5,6,7,8,9,10]
```

l1=[i for i in l if i%2==0] Even number.
short code.

* wap to display unique vowels present in the given word.

Write a program to chek if the number ambstorm or not.

Ans:-

```
n= 153
T=n
Sum=0
While T>0:
    digit = T % 10
    Sum + digit ** 3
    T=T // 10
if n==sum:
    print ('given number is armstrong')
else:
    print ('given number is not armstrong')
# given number is armstrong number)
```

```
l=['hello', ' ', 'python', ' ', 'java']
1.  for i in l:
        if i == ' ':
            l. remove (' ')
    print (l)
# ['hello', 'python', 'java']

2. l1 = [ i for i in l if i != ' ']
   print (l1)
# ['hello', 'python', 'java']

3.) l1=
   l2 =[string for string in l if string]
      print("original string", l)
      print ("filter string", l1)
```

* Remove all occurances of spacific item ~~of~~ from list

```
* l = [1, 2, 3, 1, 1, 1, 3, 4, 5, 6, 1]
  l1 = [i for i i l if i != 1]
  print (l1)
# [2, 3, 3, 4, 5, 6]
```

* Write a program to print factorial number of 6

* Tupple :-

Tupple is a collection of order hytrogenious element immutable, duplicates are ~~not~~ allowed data element. We can define tupple using ( ) round brackates.

```
Ex:- t = (1, 2, 3, 4, 5)
     print (type (t))
# <class 'tupple'>

     t = 10, 20, 30
     print(type(t))
# <class tuple>
```

```
* t1 = tuple()
  print (type (t1))
# <class tuple)
```

* Where we use tuple:-
  If your data is fixed and never changes then we should go for tuple.

* Accessing tuple element:
1. Indexing:-
2. Slicing:-
3. forloop:-
4. While loop:-

```
* while loop:-
  num=0
  while num<=20:
      print (num)
      num+=1
```

```
* i=0
  while (i< len (t)):
      print (t[i])
      i+=1
```

* Write a program to ~~given~~ print given tuple element without using sum.

```
t=(10, 20, 30, 40, 50)
Sum=0
for i in t:
    sum = sum+i
Print (sum)

# 150
```

- Maximum:

```
t = (10, 20, 80, 40, 50)
m = t[0]
for i in t:
    if i > m:
        m = i
Print (m)
#50
```

- Minimum:-

```
t = (10, 20, 30, 40, 50)
m = t[0]
for i in t:
    if i < m:
        m = i
Print (m)
#10
```

* Second highest:

```
t = (10, 20, 30, 40, 50)
m = t[0]
For i in t:
    if i > m:
        m = i
k = 0
for i in t:
    if i > k and i != m:
        k = i
Print (k)
#40
```

* Functions in tuple:-

1. len:- It is use to return length of tuple.

ex:- t = (1, 2, 3, 4, 5)
print (len (t))
5

2. Count:- It is used to count occurances of element.

t = (1, 2, 3, 4, 5
t.count

3. Index function:- It is used to return index of first occurance. if it is not available then give value error.

t = (1, 2, 3, 4, 5
Print (t.index (5) #5

4. Sort

It is used to return

Ex:- t = (40, 30, 20, 10)
print (sorted (t)) # [10, 20, 30, 40]

t = (40, 20, 30, 50, 60, 70)
print (sorted (t, reverse = True))
[70, 60, 50, 40, 30, 20]

5. Min & Max:-

- It is used to return maximum number of tuple.
- It is used to return minimum number of tuple.

6. Reverse:-

It is used to returen reversed order of tuple.

* Wap program tuple of number print its average

t = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
total = Sum (t)
len (t) # 55

* Cmp( ) :- It is used to compare two tuple.

ex:-

```
t1 = (1, 2, 3)
t2 = (1, 2, 3)
print (cmp (t1, t2))
# 0
```

* Tuple Comprahension:-

Creating tuple object from any itterable object.

Tuple comperanhsion not supported in Python.

* del ( ) :-

```
t = (1, 2, 3, 4, 5)
del t
print (t)                    #
```

* Tuple Packing & Unpacking:

1. Tuple Packing:- Tuple packing is a creating a tuple object from packing of variable.

Ex:-

```
a = 10
b = 20
c = 5
d = 15
t = (a, b, c, d)
print (t)
#(10, 20, 5, 15)
```

2. Tuple Unpacking:- Tuple unpacking is a reverse of tuple packing.

Ex:-

```
a, b, c, d = t
print (a)        # 10
print (b)        # 20
print (c)        # 15
print (d)        # 5
```

List	Tuple
• Comma Saprated value square brackate (mandatory)	• Comma Saprated values define round brackate
• list object are mutable. We can change the element of list	• Comma Tuple objects are immutable. we cannot change element of tuple.
• If the content is not fixed and keep on changing then we should go for list.	• If content is fixed then we should go for tuple
• list required more memory then tuple	• Tuple required less memory then list.
• More execution time then tuple.	• less execution type then tuple.

Bitwise ~~w~~ right shift.

$20 \times 4 = 80$

$2 \times 2 = 4$

Print (13<<3)

117.

Print (120<<2)

* Dictionary In Python:

If you want to represent a group of object as a key value pair then we should go for dictionary. Duplicate keys are not allowed in dictionary but value can be duplicated. hetrogenious objects are allowed for both key and value. Dictionary are mutable.

Indexing and slicing are not allowed in Dictionary. we can define dictionary (curly brackets) { } , dict () function.

value

- d= {'name': 'Ajit' , 'Roll_No'= 1 }

key

Seprator ,

- d= { }
  print (type (d))
  # <class 'dict'>

* Accessing Elements of Dictionary :-

We can access element of dictionary using key name.

* d= {'name': 'suprit', 'Roll: 1 ', 'batch': 'P422'}
  Print (d [nam])
  # Suprit

* WAP to enter nam & percentage in a dictionary and display.

*

```
d = {'name' : 'suprit',
     'Roll' : 1 }
d = ['Batch'] = 'P422'
print (d)
# {'name' : 'nahida', 'Roll' : 1 , 'Batch' : 'P422'}
```

we are directely assign key or values in dictionary

* WAP to enter name & Percentage in a dictionary and display.

```
d = { }
i = 1
n = 1
while i<=n:
    name = input ('enter name')
    mark = input ('enter ~~name~~ mark')
    d[name] = mark
    i+=1
Print (d)
# Enter name 'Nahida' enter
# Enter mark 87 enter
# {'nahida' : '78'}
```

* Deleting Dictionary Element:

1. How to delet key value pair:-

We can delete key value pair using del

* del d[key]
* Ex:- d = {'name' : 'nahida', 'Roll' : 4}
  [name]

```
d.
print(d)
# {'Roll' : 4}
```

2. d.clear()
d.clear() used to remove all elements from dictionary

Ex: d.clear()
print (d)
d={'name': 'nahida', 'Roll': 4}
d.clear
print (d) # { }

3. del d ():
3. To remove dict:
Ex: del d

* Important Function of dictionary:-

1. Dict(-):-
It is used to define & create dictionary.
Ex: d= dict ()
print (type())
<class 'dict'>

2. len ():- it is used to calculate len of dictionary:
Ex:- len ()
print (len(d))

Ex:- d={'name': 'nahida', 'Roll': 4}
print (len (d))
2

3. Get ():- It is used to return value associated with key.
* Get (key)
If the key is available then return corresponding value otherwise return none.

If you ~~want~~ don't want none value then define (key default value)

Ex:- d= {'name': 'nahida', 'Batch': 'P422'}
print (d.get ('nam'))
nahida.

4. Pop ():- It removes the entry associated with specified key and return corresponding value. If the specified key is not available then we will get key error. Pop (key)

Ex:- d= {'name': 'nahida', 'Age': 23, 'batch': 'P422'}
Print (d.pop ('batch')
P422

5. Pop item (): It is used to delete & return last inserted item from the dictionary. If the dictionary is empty it gives key error.

Ex:- d= {'name': 'nahida', 'from': 'Pune', 'batch': 'P422'}
Print (d.popitem ())
('batch': P422')

d= { }
print (d.popitem ())
key Error: 'popitem (): dictionary is empty.

6. key ():- It is used to return all the keys from Dictionary.

Ex:- d= {'name': 'nahida', 'Roll': 4, 'batch': 'P422'}
print (d.keys())

dict_keys (['name', 'from', 'batch')

7. Values :- It is used to return all the value from dictionary.

* d.value

ex:-

```
d={'name':'nahida','From':'Pune','batch':'P422'}
Print(d.values())
# dict_values(['nahida','Pune','P422'])
```

8. Items() :- It returns list of tuple represent key value pair.

Ex:-

```
d={'name':'nahida','Roll':4,'From':'Pune'}
print(d.items())
# dict_item([('nam':'nahida','Roll':4,'from':'Pune')])
```

* define valus using key function

```
Ex: d={'name':'nahida','Roll':4,'From':'Pune'}
For i in d.keys():
    print(d[i])
# nahida
Pune
P422
```

* define values using item function:

```
Ex: d={'name':'nahida','from':'mishwar','batch':'P422'}
for k,v in d.items():
    print(v)
# nahida
mishwar
P422
```

→ (Cloning)

9] Copy (): copy function is used to create exacte duplicate dictionary.

```
Ex: d={'name': 'itvedant', 'Branch': 'Pune'}
    d1 = d.copy()
    print (d1)

# {'name': 'itvedant', 'Branch': 'Pune'}
```

WAP to take dictionary and print sum of values

```
⇒ d={1: 34, 2: 76, 3: 78, 4: 56}
  S = 0
    For k,v in d.items():
        s= s+v;
    Print (s)
  # 244.
```

WAP to find number of occurances of each latter present in string & store in dictionary.

```
Q:- s='aabbcdefabced'
    d={}
    for i in s:
        d[i]=s.count(i)
    Print(d)
#
    {'a':3, 'b':3, 'c':2, 'd':2, 'e':2, 'f':1}

    s='aabbcdefabced'
    d={}
    for i in s:
        d[i]=s.count(i)
    Print(d)
```

→

```
    d[i]=d.get(i,0)+1
```

* Dictionary Comprehension :- Method.

Creating dictionary from object from any itterable object.
list, range, string, tuple.

```
# d= { i : i**2 for i in range (1,11) }
  print (d)
```

WAP A program to find number of occurances of each vowels present in given string

10 Update ():

```
d={'name': 'nahida', 'from': 'Pune', 'batch': 'DSE'}
d1={ }
d1.update (d)
print (d1)

# {'name': 'nahida', 'from': 'Pune', 'batch': 'DSE'}
```

11 Set-default ():

```
d.setdefault (key)
d={1:1 , 2:4 , 3:9 , 4:10 }
print (d.setdefault (3,27))        # 9
print (d.setdefault (5, 25)
#d{ 1:1 , 2:4 , 3:9 , 4:10 , 5, 25}
```

(key, value) In case key not defined.

12 Form keys () :- It is used to create new dictionary with keys from suplied itterable object and values all set to spacified value.

syntax:-

```
l = [1, 2, 3, 4]
d = fromkeys (iterative-object, default)
```

```
l = [1, 2, 3, 4, 5]
print (d.Fromkeys (1, 'number'))
# {1: 'number, 2 : 'number, 3: number', 4 : 'number'}
```

*. zip ()

```
l = [1, 2, 3, 4]
l1 = [1, 4, 9, 16]
d = dict (zip (l, l1))
print (d)
# {1: 1, 2:4, 3:9, 4 : 16}
```

Set:

* Set :- If you want to represent group of unique value as a single entity then we should go for set. duplicates are ~~all~~ not allowed, insertion order is not preserved. indexing & slicing are not allowed. hytrogenious are allowed. set object are mutable. we can represent set element curly braces. We can perform mathmatical operation on set like intersaction, differencial union. (Merge)

Ex:- set = { }
- print (type (set))

<class dict>

- set = {1, 2, 3, 4, 5}
  print (set)

[1, 2, 3, 4, 5]

- s = {1, 2, 3, 4, 5}
  for i in s:
  print (s)

< {1, 2, 3, 4, 5}
{1, 2, 3, 4, 5}

* Important function of Set:

1] Add ():

syntax:- Set.add (element)

Set1 = {1, 2, 3, 4, 5}
Set 1. add (4) (element)
print (set1) #

{1, 2, 3, 5, 4}

Imp

* Functions:-

Functions is a block of code define with name.

* We used function whenever need to perform same task multiple time: without writing same code again.

Python support two type of function
1.) Built-In function
2.) User Define function.

1.] Built-In :-

The function which are coming along with python are called as built-In function
Eg. Print(), input(), eval(), append(), chr(), Complex().

2.] User Define :-

The function which are developed by Programmer explicitly according to ther requirement called as user-define function.

Eg. Def

We can use Def keyword to define function

Syntax:-

```
def function-name (Argu1, Argu2....):
    return value
```

Eg.

```
def hello():
    print ('hello world')
hello()
```

* Write a function for addition of 2 variables or parameters

```
def addition (a,b):
    Return (a+b)
Print (addition (10,20))
```

* Parameters :- Are input to the function. If the function contains parameters then at the time of calling compulsory we should provide values otherwise we will get errors.

* Return Keyword :-
It is used to return output or value from the function.

* 4 Types of arguments are allowed in python.

1) Positional Argument
2) Keyword Argument
3) Default Argument
4) Variable length Argument

1) Positional Argument :
The number of argument and position of argument must be match if we change the order then result will be changed.

2) Keyword Argument :-
We can pass Argument by keyword that is parameter name.

3) Default Argument:

Sometimes we can provide default value for our positional argument.

eg.

```
def display (name = 'shivam'):
        print (name)
display ('suprit)
```

4) Variable length Argument:

Sometimes we can pass variable no of argument are called as variables length argument.

We can declare variable length argument using * symbol.

* Scope Of Variables:-

1) Global
2) local.

1) Global :- The variable which are declare outside the function called as global variable.

We can access global variable inside and outside the function.

Eg.

```
a=40
def func1 ():
    print (a)
func 1 ()
print (a)
```

40
40

2) Local :- Local variable which are declare in side the function called as local variable local variable are accessible only for the function or only within the function we cannot access local variable outside of function.

eg.

```
def func1 ():
    b= 20
    print (b)
func1 ()                    # 20
```

* Global Keyword:

We can use global keyword for to declare global variable inside function to make global variables to the function so that we can perform require modification.

eg.

```
c= 100
def see ():
    global c
    c=77
    print (c)
def look ():
    print (c)
see ()
look ()
```

output # 77
77

* Recursive function: A function call itself called as Recursive function.

o Advantages of Recursive function:
1) We can reduce the length of code.
2) Improve preadibility.
eg:-

```
def factorial (n)
    if n==0:
        fact=1
    else:
        Fact = n * factorial (n-1)
        return fact.
fact = 3 * factorial (2)
     = 3 * 2 * factorial (1)
     = 3 * 2 × 1 × factorial (0)
     = 3×2×1×1
     = 6
```

* Mathematical operations on set:
1. Union :- It is used to return all element present in both set

2. Intersection:- It is used to return common elements present in set
v. Intersection (v)

3. Difference ():- It is used to return element present in first set but not in set 2.

4. Symmetry. difference ():-
It is used to return element present in either set 1 or set 2 but not in both.

* Anonymous Function:-

The nameless function known as anonymous function. Sometimes we can declare a function without any name this type of ~~func~~ name less function known as anonymous function. The main purpose of anonymouse function is just for instant use for that is one time uses. We can define anonymouse function using lambda keyword. Anonymous function also called as lambda function.

* Syntax:-

```
lambda argument : expression
```

Eg:-

```
def sqr (n):
    return n**2
print (sqr (5))
```

Eg: Lambda.

```
s= lambda n: n**2
print (s(5))
# 25
```

* Write a lambda function to find some of two numbers

```
b= lambda a,b : a+b
print (b (10,20))    # 30
```

* Write a lambda function

```
a= lambda a,b : a if a>b else b
print (10, 30)
# 30
```

* Filter Function:

We can use filter function to filter of values from given sequance based on some condition.

* Syntax:-

```
filter (function, Sequ
```

```
* def iseven (n):
      if n%2==0:
          return True
      else:
          return False
  l=[1,2,3,4,5,6,7,8,9,10]
  even= list (filter (iseven, l))
  print (even)
#[2, 4, 6, 8, 10]
```

* Using lambda function.

```
l=[1,2,3,4,5,6,7,8,9,10]
even=list (filter (lambda n: n%2==0, l))
print (even)
#[2, 4, 6, 8, 10]
```

* Map function ():-

for every element present in the given sequance apply some functionality and generate new element with required modification. For this requirement we should go for map function.

* Syntax

map (Function, Sequance)

```
l=[1,2,3,4,5,6,7]
def sqr (n):
    return n*n
l1=list (map (sqr, l))
print (l1)
#[1, 4, 9, 16, 25, 36, 49]
```

using lambda function.

```
l= list (map (lambda n: n*n ))
print (l1)
```

* Reduce Function:

Reduce function reduces sequance of elements into a single element by applying specified function.

Syntax:-

Reduce (Function, sequance)

Reduce function reduce in functools modules. hence we should write import statement.

Eg:-

```
From functools import *
l= [1, 2, 3, 4, 5, 6, 7]
add= reduce (lambda x,y : x+y , 1)
print (add)
# 28
```

• Modules in Python:

* Modules:-

- Every python file acts as a module.
- A Group of function, variables and classes save to file which is nothing but a module.
- syntax 1:-

  import module-name
- Syntax 2:

  From module-name import class

Using import

File1 x=10 → test.py

```
def show ():
    print ('hello this is my test module')
```

File2 import test → module.py

```
print (test.x)

# 10
Print (test.show())
# hello this is my test module
None → None because we use print.
```

Using from.

```
From test import *
print (x)
show ()
# 10
hello this is my test module.

def add (a,b):
    return a+b
def sub (a,b)
    return a-b
def mul (a,b)
    return a*b
def div (a,b)
    return a/b

From add-module import *
print (add (5,6))
print (sub (5,6))
# 11
# -1
# 30
# 0.833333333333334
```

* Package :-

Package is a collections of module. Any directory or folder which content __init__.py file. A package content sub-packages. __init__.py it will be a empty file.

* File handling:-

1) File:- File is a storage area to store data.
There are two types of file
1) Text file
2) Binary file

Text → txt, doc, html, py
Binary → image, png, jpg, video, audio.

* Opening file:-

Open():- used to create new file and open existing file.

Syntax:-

Open ('filename', 'mode')

* 12 mode → There are main three
- 'w' → used to write a file
- 'r' → used to read a file
- 'a' → us

1. w → it is used to open file in write mode (only write). if the file already content some data then it will be overriden. if the spacified file is not available then new file will created.

* Open ('File name', 'mode')

close ():- used to close file.
write ():- use to write data

eg.
```
f = open ('Text.txt', 'w')
f.write ('first line')
f.close ()
print ('File created')
```

f.name → It is used to show name of file
f.mode → It is used to show mode of file.
writelines ():- It is used to write list of lines.

eg:-
```
l = ['first line \n', 'secondline \n', 'thirdline \n']
F = open ('Filename.txt', 'w')
f.writelines (l)
f.close()
```

1) read (): Used to read total data from the file.

2) read (n): It is used to read n characters from the file.

3) readline ():- It is used to read only one line.

4) readlines ():- It is used to read all lines in the list.

Read File

1.) r :- It opens an existing file to read-only mode. The file pointer exists at the begining

2.) rb:- It opens the file to read only in binary format. The file pointer exists at the begining.

3.) r+ :- It opens the file to read and write both. The file pointer exists at the begining.

4.) rb+ :- It opens the file to read and write both in binary format. The file pointer exists at the begining of the file.

5.) w :- It opens the file to write only. It overwrites the file if previousely exists or creates a new one if no file exists with the same name.

6.) wb :- It opens the file to write only in binary format. It overwrites the file if it exists previousely or creates a new one if no file exists.

7.) w+ :- It opens the file to write and read data. It will override existing data.

8.) wb+ :- It opens the file to write and read both in binary format.

9.) a :- It opens the file in the append mode. It will not override existing data. It creates a new file if no file exists with the same name.

10.) ab :- It opens the file in the append mode in binary format.

11.) a+ :- It opens a file to append and read both.

12.) ab+ :- It opens a file to append and read both in binary format.

* With Statement:-

With statement is used to open a file. The Advantages of with statement is it will take care of closing file.

Syntax:-

```
with open ('file_name', 'mode') as f:
```

No need to close() statement.

Eg:-

```
with open ('test.txt', 'a') as f:
    f.write ('using with statement')
```

1) Tell():- Tell function is used to check current position of cursor in file. from the begining

syntax:- Print (f.tell)

2) Seek():- We can seek method to move cursor to some spacific position.

syntax:-

```
f.seek (offset) → where to place
```

eg.

```
f.seek (14)
print (f.tell())
# 14
```

OS Module:- Operating System:-

```
os
```

WAP to check given file available or not if it availab print its containt.

```
⇒ import os
print (os.path.itself ("with.txt"))
if os.path
```

Write a program to print number of lines & character present in the given file.

* Working with Directory:-

1) To know C.W.D. Current Working Directory:-
~~os. cwd~~
os.getcwd()

2) To Creat Subdirectory / folder.
os. mkdir (directory names)
→ import os
os. mkdir (ajit)

3) To Create multiple subdirectories:
syntax:-
os. mkdirs(directory /)

→ os.mkdirs (ndhida / madhavi)

4) To remove directory
syntax:-
os. remove ('Div-name')

5) To Rename Directories:-
syntax:
os. rename (oldname , newname)

* Oops:- Object Oriented programming language.

Object oriented programming is programming which is completely based on object. in oop everything is treated as object.

* In python following are the oops concept.
1) Class
2) Object
3) Inheritence
4) Polymorphism
5) Abstraction
6) Encapsulation

1.) Object :- Object is an instance of a class. It is a collection of atributes and methods. we use object of a class to perform action. object has two characteristics, state and behaviour.

2.) Class :- Class is blueprint of object. class is used to define data structure. that binds data members (means variable) & methods into single unit. we can define class using class keyword.

- Syntax to define class:
  class class-name

(self is a keyword and it is compulsory).

Ex:- class A:
pass To define empty block

- syntax for object creation
  object_name = classname ()

- How to call method in class
  obj name . method name ()

Example: class A:
def show (self):
print ('hello this is my show')
def display (self):
print ('display')
obj = A ()
obj. show ()
obj. display ()

* Constructor:-

Constructor is a special method in python which is used to initialized object of class.

- Constructor will call automatically at the time of object creation.
- We can define constructor using __init__()
  There are three type of constructor

1.) Default Constructor
2.) Parameterized constructor
3.) Non- Parameterized Constructor

1.) Default Constructor:-

Python will provide default constructor if no constructor is not defined.
It does not perform any class but initialize the object.

2) Parameterized Constructor:-
A constructor with define parameters or arguments called as parameterized constructor.

3) Non-Parameterized Constructor:-
A constructor without any argument is called as Non-Parameterized constructor.

*. Self keyword:-
Self is a first argument of class whenever we define instance method for a class then we use self as first parameter.
- using self we can access instance variables and instance method of the object or class.

eg:-
```
Class A:
    def __init__ (self, name, roll):Parameters
        self.name = name
        self.roll = roll
    def show (self):
        print (self.name)
        print (self.roll)
obj = A ('xyz', 1)
obj.show()
```

* Destructor:-
Destructor is special method that is called where object get destroyed.
- We can use __del__() for define destructor

- Destructor is used for clean code.
Clean means it will delete the parameters inside code.

- when we execute del object name destructor get call automatically & object get garbage collected.

- --del-- is used to perform cleanup task before deleting the object.

3-) Inheritance :-

Inheritance is the process of inheriting all the properties and behaviour from parent to the child class called as inheritance.

- Parent class called as base class.
- child class called as sub-class or derived class.

Syntax:-

Class class name (base_classname)

Eg.

```
class A:
    pass
class B(A):
    pass.
```

* Inheritance Types.
1) Single
2) multiple
3) multi~~valued~~level.
4) Hierachial
5) Hybrid.

eg.
```
Class A:
        def show (self):
            print ('class A')
Class B (A):
        def display (self):
            print ('class B')
Obj = (B)
obj.show()                    # Class A
obj.display()                 # Class B
```

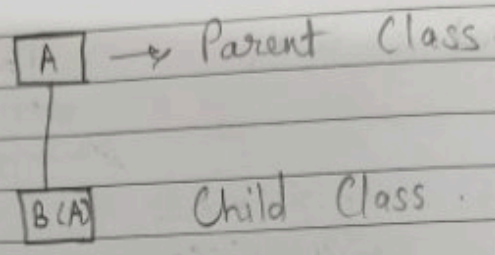
WAP for inheritance in python.

WAP to check given number is ~~[illegible]~~ armstrong number is not.

```
n= 153
T= n
sum=0
while T>0:
    digit = T % 10
    sum + digit ** 3
    T=T // 10
if n == sum:
    print ('given number is armstrong')
else:
    print ('given number is not armstrong')
```

1) Single Inheritance :-

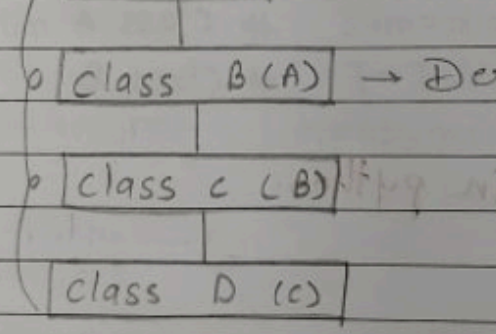
A child class is inheri~~taritance~~ a single parent class called as 'single inheritance'.



2) Multi-level :-

A child class which is inherit from derived class called as multi-level class. "Chaining of Classes".

Eg.

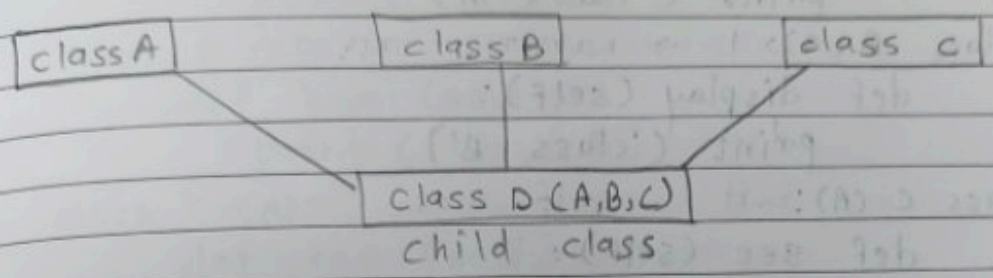


Ex:-

```
class A:
    def show (self):
        print ('class A')
class B (A):
    def display (self):
        print ('class B')
class C (B):
    def see (self):
        print ('class C')
class D (C):
    def dee (self):
        print ('class D')
obj = D ()
```

3) Multiple Inheritance:-

A child class which is derived from multiple parent class called as multiple inheritance.

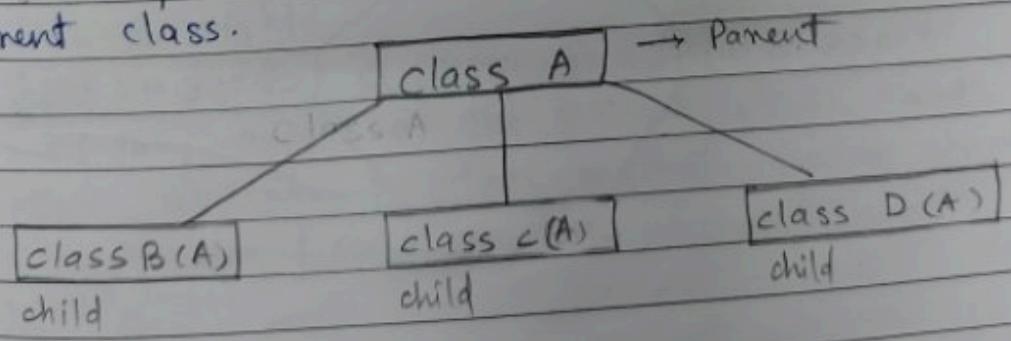



```
class A:
    def show (self):
        print ('class 'A')
class B:
    def display (self):
        print ('class B')
class C:
    def see (self)
        print ('class C')
class D (A, B, C)
    def dee (self)
        print ('class D')
```

```
obj = D ()
obj. show ()
obj. display ()
obj. see ()
obj. dee ()
```

4) Hierachial Inheritance:-

Multiple child class derived from single Parent class.

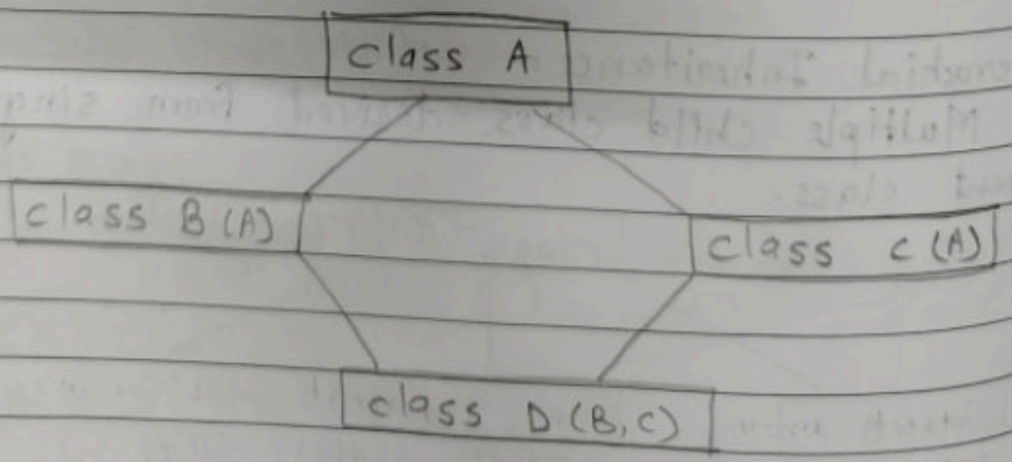


Ex:- class A:
```
    def show (self):
        print ('class A')
class B (A):
    def display (self):
        print ('class B')
class C (A):
    def see (self):
        print ('class C')
class D (A):
    def dee (self):
        print ('class D')

obj = B ()
obj = C ()
obj = D ()
obj. Display ()
obj. see ()
obj. dee ()
```

5) Hybrid:-

Hybrid inheritance is the combination of multiple types of inheritance.



Eg:-

```
class A :
    def aaa (self):
        print ('class A')
class B (A):
    def bee (self):
        print ('class B')
class C (A):
    def cee (self):
        print ('class C')
class D ('B,C) :
    def dee (self):
        print ('class D')

obj = D ()
obj . aaa ()
obj . bee ()
obj . cee ()
obj . dee ()
```

* MRO (Method Resolution Order):

MRO Plays an accential role of hybrid inheritance. ~~first~~

1) It search in current parent class if not available then search in a parent class from left to right.

* Super method:- A super function return a temporary object of the parent class that allow us parent class method inside child class.

When we have to convert child class parameters into parent class then we use this super method.

eg:-

```
class Parent:
    def __init__(self, name, mark)
        self.name = name
        self.mark = mark

class child (Parent):
    def __init__(self, name, mark, roll)
        self roll = roll
        super().__init__(name, mark)
    def show (self):
        print (self.name, self.mark, self.roll)

obj = child ('xyz', 100, 1)
obj.show ()
    # xyz 100 1
```

4.

* Polymorphism:-

many Forms.

Polymorphism is an ability of an object to take many forms. Polymorphism allow us to perform same action in many different way.

* Example :- len () function.

string = 'hello' →
list = [1, 2, 3] →

* Overloading:-

There are three types of overloading.

1. Operator Overloading
2. Method Overloading
3. Constructor Overloading

1. Operator Overloading:-

We can use same operator for multiple purpose called as operator overloading.

• Example:-

Print (10 + 20) = 30 ⟶ Addition

Print ('10' + '20') = 10, 20 ⟶ Concatination

* Magic Method:-

following is the list of operators.

+ 1. __add__ (self, other)
- 2. __sub__ (self, other)
* 3. __mul__ (self, other)
/ 4. __div__ (self, other)
// 5. __floordiv__ (self, other)
% 6. __mod__ (self, other)
** 7. __pow__ (self, other)

* Example:-

```
Class A:
    def __init__ (self, mark):
        self.mark = mark
        print ('calling')
    def __add__ (self, other)
        return self.mark + other.mark

obj1 = A(100)
obj2 = A(90)
print (obj1 + obj2)

# Calling
# Calling
# 190
```

2.) Method Overloading:-

A process of calling same method with different parameters called as method overloading.

Methodoverloading not applicable in python.

Ex:-
```
class A:
    def __init__(self, mark):
        self.mark = mark
        Print ('calling')
    def __add__(self, other):
        return self.mark + other.mark
obj1 = 100
obj2 = 90
Print (obj1+obj2)
```

```
# calling
# calling
190.
```

3.) Constructor Overloading:

Calling constructor with the different parameters called as constructor overloading.

```
Class A:
    def __init__(self, name):
        self.name = name
        print ('calling first')

    def __init__(self, name, mark):
        self.name = name
        self.mark = mark
        print ('calling second')

obj = A ('nahida', 100)

# Calling Second.
```

4.) Method Overriding :-

Whatever members available in parent class are by default available in child class through inheritance. If the child class not satisfied with parent class implimentation then child class is allowed to redefine that method in the child class based on this requirement. This concept is called as method overriding.

Re-implimenting the parent class method into child class is called as method overriding.

• Ex:-

```
Class Parent:
    def show (self):
        print ('calling A')
Class child:
    def show (self):
        print ('calling child')
obj= child()
obj. show()
# calling child
```

4. Encapsulation :- Class is a example of Encapsulation

Encapsulation is the process of binding data members into single unit.

eg:-

```
class A:
    def __init__ (self, name, mark, other):
        self self. name = name
            self. mark = mark
    def show (self):
        print (self. name)
    def display (self):
        print (self. mark)
obj = A (xyz, 100)
```

Python provide three types of acess modifier.

1. Public access modifier
2. Private access modifier
3. Protected access modifier.

1. Public access modifier:-

Public data member are accessible within class and outside of class.

Ex:-

```
obj = A ('nahida', 100)
Print (obj. name)
# nahida
```

2. Private data member:-

Private data member accessible only within class. And we cant access them directly outside of the class. to define private variable add two -- as a prefics at the start of variable name

eg:

```
self.--name = name
```

- We can access private variable outside the class

_classname__datamember	name mangling method.

3. Protected data member:-

protected data member is accessible within class and its sub class.

eg:-

```
class A:
    def __init__(self,name,mark):
        self.mark = mark
class B(A):
    def __init__(self,name,mark):
        super().__init__(name,mark)
    def show(self):
        print(self._mark)
obj = B('xyz', 90)
obj.show()
```

90

* Exception handeling:-

In python there are mainly two types of error

1. Syntax error
2. Runtime error

1. Syntax error:- Spelling mistak ; missing then

The error which are occure because of invalid syntax called as syntax error.

2. Runtime Error:-

Runtime error also called as exception. while executing the program if something goes wrong because of end user input or programming logic or memory management then we will get runtime error.

- only runtime program handle by programmer.
- Wrong logic by programmer

ex:-

```
a = int(input('a'))
b = int(input('b'))
print(a/b)
```

Note:- Exception handling concept aplicable for runtime error not for syntax error. The main advantage of exception handling is graceful termination of the program or code cleaner.

* Exception handling does not mean repairing we have to define alternative way to continue rest of the program.

→ Syntax:-

- try:
  # code which occures exception
- exception:
  # handle exception

- To handle exception we need to use try and exception block.

* What is try block?
  It is used to define risky code that can occures exception.

* What is exception block?
  It is used to handle error raised in try block. Except block executed when occuring try block.

ex:- try:

```
    a= int (input ('enter A'))
    b= int (input ('enter B'))
    c= a/b
    print (c)
except:
    print ('Please enter valid value of B')
```

#

enter value A = 10
enter value B = 0
please enter valid value of B

* try - except - else:-

• else:- we can use else statement with try block to check if try block executed with without any exception. else blocks will executed if there is no exception occure. if we have error then execute except or we have no error then else block.

```
try:
    a= int ( input ('enter A'))
    b= int (input ('enter B'))
    c= a/b
    print (c)
except:
    print ('Please enter valid value of B')
else:
    print ('else block')

# enter A =10
# enter B =2
  5.0
```

* try - except - else - Finally :

It is used to write block of code that must execute whether exception raised or not.

ex.
```
try:-
    a= int (input ('enter A'))
    b= int (input ('enter B'))
    c = a/b
    print (c)
• except: print ('Please enter valid value of B')
• else: print ('else block')
• finally:-
    print ('its Finally block')
```

* Pickling & Unpickling:-

* Pickling:-
  pickling is the process of converting python object into other file supported form.

* Unpickling / Desierialization / unmarshling.
  It is a process of ~~other~~ converting other file supported form into python object. called as unpickling desierialization

- We can implement pickling & unpickling using pickle module. Pickle module content dump function to perform pickling & pickle module content load function to perform unpickling

* Syntax of Dump ()
  [pickle. dump (object, file)]

* Syntax of load ():-
  [obj = pickle. load (file)]

Ex:-

```
Import pickle

Class student:
    def __init__(self, name, mark, roll)
        self. name = name
        self. mark = mark
        self. roll = roll
    def show(self):
        print (self.name, self.mark, self, roll)
with open ('student.dat', 'wb') as f:
    obj = student ('ABC' 'mark, roll)
    Pickle. dump (obj, f)
    print ("pickling done")
```

output:- Pickling done

* Unpickling :-

```
import pickle

class A:
    def __init__(self, roll, name, subject, mark):
        self.roll = roll
        self.name = name
        self.subject = subject
        self.mark = mark

    def show(self):
        print(self.name, self.roll)

with open("student.dat", "wb+") as f:
    obj = A(1, "nahida", "python", 80)
    pickle.dump(obj, f)
    print("pickling done")

with open("student.dat", "rb") as f:
    obj = pickle.load(f)
    obj.show()
    print("unpickling done")
```

output # :

```
# pickling done
# nahida 1
# unpickling done
```