python language fundamentals:
========================

1.character set:
============

upper case letters(capital letters):
==========================

A,B,C,D,.......Z

lower case letters(small letters):
=========================

a,b,c,d,.....z

digits:
=====

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

special characters/symbols:
======================

+,-,/,*,[,],{,},(,),&,#,@,!,~,|,^,,$,%,>,<,:,",',........................

2.keywords/reserved words:
======================

these words are used to "do a spesific job/action in the python program"

in python,we will have the following keywords/reserved words:
===

if,else,elif,

for,while,def,

break,continue,

class,lambda,with,

as,import,True,False,

try,except,finally,assert,

del,......................

to know the all python keywords,we will use the following python code:
===

```
import keyword

print(keyword.kwlist)
```

python comments:
==============

python comments are written in python program,to give the description about python code

python comments will give better readability and understandability about the python

program or code

to write the python comments in python,we will use the following ways:
==

single line comments ===>to write single line comments,we will use a symbol called "#"

multi-line comments===>to write the multi-line comments,we will use """ """(triple
      quotes)

python data types:
==============

data ====>" what we can able to store in the computer memory"

information ===>"processed data /after the processing the data what we got,is known as

information"

example:
=======

a=10,b=20

a+b===>30(here a,b are data,+ is process or operation and 30 is information)

data type===>data type refers "what kind of data the variable has"

in python,we will have the following data types:
=====================================

numeric data type:
==============

integer data type:
==============

integer means "integer numbers(the number which are does not have any decimal/fractional

point)

example: 1000,100,9867,-123,-4589,...............

real number/floating-point number type:
===============================

these number will contain "decimal point/fractional point"

example:
========

1.234,7.8901,-0.9875,.................

complex number type:
==================

complex number is a number and which is in the form of "a+ib"

where a is "real number" and i is "imaginary number"

but in python,the complex number is in the form of "a+bj"

example:

10j

4+5j,9-2j,..................

character type:
============

in python,we will have three types of character data types:
==

1.single charcater

2.multi-character(String)===>string means "collection or group of characters"

3.doc string(multi-line string)

to store the single character / string,in python we will use "Single quotes or double quotes"

to store the doc string,we will use "triple quotes"

example:
=======

```
'h','r',"hello",'hai',"p"........  <=======single character/string

"""this is doc string,this is doc string
                                                            <======= doc
string(multi-line string)
  this is doc string,this is doc string"""
```

Boolean data type:
==============

this data type we are to store the boolean values(True/False)

those are

True(in numeric,boolean means other than zero)

False(in numeric,boolean means always zero)

in general,when we are doing any airthmetic operations/other operations,we will use true as 1 and false as 0

example:

a=True

b=False

a+b===>1

a*b===>0

a/b===>undefined

in python we will also have other data types like as follows:
==

sequence type(list,tuple amnd range())

[sequence type we are using in python,to store the multiple values in python]

set type(set,frozen set)

[set type is used to store the multiple values and which are unique(not duplicate)]

map type(dictionary)

[map type used to store the data in the form key and value pairs]

None type(empty value or no value)

python identifiers:
==============

in python,identifier means "a name in python program"

the name may be:
==============

variable name

class name

list name

function name

module name

tuple name ....

whenever we are creating python identifier(any name),we need to follow the following rules:
===

1.in python,any identifier must start with letter or underscore(_)

2.in python,any identifier may have the following characters:

```
        all upper case letters(A,B,C,D,...Z)

        all Lower Case Letters(a,b,c,d,.....z)

        all digits(0,1,2,3,4,5,6,7,8,9)

        one special symbol called underscore(_)

3.in python,any identifier may be alphanumeric(name may have both letters as well
as digits)

4.in python,any identifer length may be anything

5.in python,any identifier name never be keyword name or reserve word name

python operators:
==============

operator ===>operator is a "symbol" and which is used to perform a spesific
operation

operend ===>operend means on which we will perform operation

example:
=======

a=10,b=20

a+b===>30,here a,b are operends,+ is symbol(airthmetic operator),30 is result

a+b is called as "expression"

expression===>expression is a combination of "operators and operends"

in python,we will have the following operators:
====================================

1.airthmetic operator

2.relational/comparision operator

3.logical operator

4.assignment operator

5.identity operator

6.membership operator

7.bitwise operator

1.airthmetic operator:
=================

this operator is used to perform all airthmetic operations like

addition,subtraction,multiplication,division,.................
```

symbol ======	meaning ========	a=10,b=20 ==========	result ======	
+	addition	a+b	30	
-	subtraction	a-b	-10	
*	multiplication	a*b	200	
/	division	a/b	0.5	<==== quotient
//	floor division	a//b	0	<==== integer quotient
%	modulo division	a%b	10	<==== remainder
**	power	a**b	100000000000000000000	

2.relational operators/comparision operators:
===================================

relational operators are used "to compare any two values"

relational operator will give the result in the form of boolean values(True/False)

in python,we will have the following relational operators:
==

greter than:
=========

symbol:>

example:
=======

a=10

b=20

a>b====>False

b>a====>True

greter than or equal to:
==================

symbol:>=

example:
=======

a=10

b=20

a>=b====>False

b>=a====>True

less than:
=========

symbol:<

example:
=======

a=10

b=20

a<b====>True

b<a====>False

less than or equal to:
================

symbol:<=

example:
=======

a=10

b=20

a<=b====>True

b<=a====>False

equal to:
=======

symbol:==

example:
=======

a=10

b=20

a==b ===>False

b==a===>False

not equal to:
==========

symbol:!=

example:
=======

a=10

b=20

a!=b===>True

b!=a===>True

logical operator:
=============

this operator is used to "compare any two conditions"

this operator will give the result in the form of boolean(True/False)

in logical operator,we will have the following operators:
===

1.logical or

2.logical not

3.logical and

1.logical or:
=========

symbol: or

rule: if any one condition is true,then entire output is true in the case of logical or

example:
=======

a=10,b=20

(a>b) or (a<b) ===>True

(a>b) or (a<b) ===>True

(a>b) or (b<a) ===>False

2.logical and:
=========

symbol: and

rule: if any one condition is false,then entire output is fa   lse in the case of logical and

example:
=======

a=10,b=20

(a>b) and (a<b) ===>False

(a>b) and (a<b) ===>False

(a>b) and (b<a) ===>False

(a<b) and (b>a)===>True

3.logical not:
==========

it will make the result "True as false or false as true"

symbol: not

example:
=======

a=10,b=20

not((a>b) and (a<b))===>not(False)===>True

not((a>b) or (a<b))====>not(True)===>False

4.assignment operator:
==================

it is used to assign any value to the variable

symbol: =

example:
=======

a=10<=== 10 is assigned to varibale a

compound operator with "="
======================

a+=10===>a=a+10=20

a-=10===>a=a-10=20-10=10

a*=10===>a=a*10=10*10=100

5.identity operator:
===============

this operator is used to compare the values which are same or not

this operator also give the result in boolean(True/False)

in python,we will have the following identity operator:
==

```
is

is not

example:
=======

a=10

b=20

c=30


a is b===>false

b is a===>false

a is c===>false

a is not b===>True

a is not c===>True


6.membership operators:
===================

in python,membership operators are used to "check the presence of value in the
given

iterable(list/tuple/string/set......)"

these operators will give the result either True or False

in python,we will have the following membership operators:
==============================================

in

not in

example:
=======

x="i am from india"

y="am"

z="india"

result= y in x  ==>True

result=z in x ==>True

result= x not in y ==>True

result= z not in x ==>False
```

result= y not in x ==> False

bitwise operators:
==============

these operators will work on binary data

when we are using bitwise operators,first the data will be converted into
"binary",on that

we will peform bitwise operations

data===>binary ===>biwtwise operator===>result(in binary)===>normal data==>user

note:
====

bitwise operator result never be "floating-point/real number",but bitwise operator

result is always "integer"

in python we will have the following bitwise operators:
==

bitwise or:
========

symbol: |(pipe)

rule: in the case bitwise or,one input is "1",then the entire result is "1"
===

example:
=======
a=10 =====>01010

b=20 =====>10100
================
a|b=======>11110<=== 30

bitwise and:
=========

symbol: &(ampracend)

rule: in the case bitwise and,one input is "0",then the entire result is "0"
===

example:
=======
a=10 =====>01010

b=20 =====>10100
================
a&b======> 00000<=== 0

```
bitwise exclusive or
===============

symbol: ^(Caret)

rule: in the case of bitwise ex-or,if both inputs are same,then the result is
zero,otherwise "1"
===

example:
=======
a=10 =====>01010

b=20 =====>10100
================
a^b=======>11110<=== 30

bitwise left shift:
=============

symbol:<<

formula: n<<s=n*2 power s

example:

a=10

a<<2==>10<<2==>10* 2power 2=10*4=40

bitwise right shift:
==============

symbol:>>

formula: n>>s=n/2 power s

example:

a=10

a>>2==>10>>2==>10/ 2power 2=10/4=2

bitwise one's complement
====================

symbol:   ~(tilde)

formula:  ~n=-(n+1)

example:

a=10

~a=-(10+1)=-11

a=-20

~a=-(-20+1)=-(-19)=19
```

python variable:
=============

python variable is used to "Store a value and the value of the variable can changes throught

the program"

in python,the variable can able to only one value at a time

in python,the variable can able to any kind of data(python dynamic typed language)

in python,if we want to create any variable,we will use the following syntax:
==

```
variable_name=literal
```

here litreal means "value"

example:
=======

```
a=10

a=1.234

a="hello"

a=True

a=None
```

python litreals:
============

litreal means "value"

in python we will have the following litreals:
==================================

numerical litreals:
==============

1.integer

    example: 1000,1234,-5678,.................

2.floating-point numbers

    example: 1.234.0.98766,..............

3.complex numbers

    example: 1+5j,9-8j,...............

4.binary numbers/litreal

```
    in python,to store binary litreals we will use prefix "0b or 0B"

    in binary number we will only digits 0 and 1

    example:0b10101,0B1100011

5.octal numbers/litreal:
===================

    in python,to store octal litreals we will use prefix "0o or 0O"

    in octal numbers we will have digits 0 to 7(0,1,2,3,...7)

    example:0o127,0O754,..........

6.hexa decimal numbers/litreal:
=========================

    in python,to store hexa decimal litreals we will use prefix "0x or 0X"

    in hexa decimal number we will only digits 0 to 9,A,B,C,D,E,F

    example:0xabf,0X123fbd,.............


character litreals:
==============

single character litreal:

      example: 'a','b',"g","y",.............

string:  group of charcaters

      example: "abc",'bcd',"fgh",..............

doc string: it is a multi-line string

      example: """ here we write the string whcih is any number of lines"""

list litreal: it is used to give the group of data

              example: [1,2,3,4,5,6,7,8,9,10]

tuple litreal:it is used to give the group of data and data will never change again

              example: (1,2,3,4,5,6,7,8,9,10)

set litreal: it is used to store the data,and all data in the set litreal are
unique

              example: {1,2,3,4,5,6,7,8,9,10}

dictionary litreal: here we store the data in the form of key and value pairs

             example:{1:2,3:4,5:6,7:8,9:,10}

boolean litreal: these are two values,those are True or False
```

None litreals: it is used to represent nothing/no-value

example: None

conditional statements/control statements/flow control statements:
==

if statements are used to "execute any logic" based on certain condition,then those

statements are called as "conditional statements"

these statements,generally will change the "flow execution of the code",that is reason

these statements are also called as "flow-control statements"

in python,we will have the two types of conditional statements:
===

1.conditional statements(with condition):
================================

in python,we will have the following condtional statements:
==

if statement

if-else statement

else-if ladder

2.un-conditional statements(without condition):
=====================================

break

continue

return

if statement/simple if statement in python:
=================================

when we have only one condition and we want to execute "a piece of code" based on the

condition,in python,we will use "if statement or simple if statement"

to create the if statement,we will use a keyword called "if"

the code what we write under if,is executed only when the "condition what we take for

if has to be true".

syntax for if statement:

```
==================

if condition:

   logic

note:
====

python is a braces free language

insted of braces,python will use a concept called "indentation"

if-else statement:
=============

when we have two conditions,based on the condition we need to execute any logic, in
python

we will use the "if-else" statement

when the condition is true,the code under the if will be executed

when the condition is false,the code under the else will be excuted

syntax:

if conditon:

     logic for if(this will excuted when condition of if is true)

else:

     logic for else(this will executed when the condition is false)


else-if ladder:
==========

if we have three or more than three conditions and we need to execute any logic
based

on the condition,then in python we will use "else-if ladder"

when we are creating else-if ladder in python,it always start with "if" and ends
with "else"

and remaing all are "else-if" conditions

suppose we have 3 conditons,in else-if ladder,we will have 1-if,1-else,1-elif(else-
if)

suppose we have 5 conditions,in else-if ladder,we will have 1-if,1-lese,3-elif's

suppose we have n conditions,in else-if ladder,we will have 1-if,1-else,n-2-elif's

syntax:
======
if condition:

      logic

elif condition:

      logic

elif condition:

     logic

.
.
.
else:

   logic
```

nested conditonal statements:
=======================

writing the "conditional statements inside another conditional statements",then we say

it is "nested conditional statements"

example:
=======

```
if condition:

      if condition:

            logic

     else:

           if condtion:

                  logic

           else:

                 logic
```

points to be remember:
==================

1.without if,we can not write "else"

2.without else,we can not write else-if ladder

3.without if,we can not write else-if ladder

un-conditional statements:

=====================

break:

it is used to "stop the pre-maturly the any loop/ from the block"

syntax:

break

continue:
========

it used to "stop or skip the current iteration in the logic"

syntax:
======

continue

return:
======

it used in the function,to return any value

return is always last statement in the function

syntax:

return expression/value

looping statements in python:
=======================

looping statements are also called as "iterative statements"

these statements are used "to excute a logic for "n" number of times until the condition

what we taken hase to be false

once the condition what we taken for looping statements is false,the loop will not execute

other the loop will execute continously due to condition is True

in python,we will have two types of looping statements:
===

while loop

for loop

while loop:
=========

in python,while loop will have the following syntax:
==

```
while condition:

      =========

        logic

     ===========
```

here,when we write any while loop in python,first it will check the condition,if it true,then it

execute the logic what we write under the while,after the executing the logic,it again check

the condition,if it is true,then the repeat the process,otherwise loop will be terminated

example:
=======

```
a=1

while a<=10:

     displa a

     a=a+1
```

output:
=======

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

for loop:
======

for loop is used to "Execute any logic based on the given iterable/collection/sequence"

in python

in python,the for loop will use with "list/tuple/set/string/dictionary/range()"

syntax for for loop:
===============

```
for variable_name in sequence_name/iterable_name:

          #logic here
```

range():
======

range() function will give the data based on given range of values

syntax for range():
==============

```
range(start,end)

in the above,the data will come from range,start to end-1

example:
=======

range(1,10)====>1 ,2,3,4,5,6,7,8,9

range(1,5)=====>1,2,3,4,

range(-4,3)=====>-4,-3,-2,-1,0,1,2

range(2,2)======>no data

range(8,2)======>no data

note:
====

when we are working with range() function,the start value always less than end

other wise

if we given start>end,the result is nothing/no-result

if we  given start and end value both are same,the result is no data

range() function with for loop in python:
================================

syntax:

for variable_name in range(start,end):

          #logic

example:
=======

for i in range(1,10):

     display i


output:
======
1 2 3 4 5 6 7 8 9

when we are working with range() function, we also take "Step" size

syntax:
======

range(start,end,step)

example:
=======
range(1,10,1)===>1,2,3,4,5,6,7,8,9

range(1,10,2)===>1,3,5,7,9

range(0,10,2)===>0,2,4,6,8

range(0,10,3)===>0,3,6,9

range(12,1,5)===>no data

note:
====

when we using range(),start or end value never be float/real number


python input and output statements:
=============================

output statements:
===============

when we want to display "any output" of the program,in python we will use a
function

called "print()"

syntax:
======

print("write the string here")

or

print(variable_name/list_name/tuple_name.......)

example:
=======

a=10

print(a)

a=1.234

print(a)

a=[1,2,3,4,5] #list

print(a)

print("hello world")

print("hello")

input statements in python:
=====================
```

in python,input statements are used to "take any input from the user"

in python,to take any input we will use a function called "input()"

syntax for input() function in python:
============================

variable_name=input("write the prompt here")

example:
=======

a=input("enter the value for a:")

b=input("enter the value for b:")

but in python,when ever we are taking any data from the user ,with help of "input()" function

,the data what we got from "input()" function will be always in the form of string format

it means,

number(integer/float/real/complex number)=====>input()=====>string

boolean data=====>input()========>string

string data====>input()======>string

whenever,we are giving the data to input() function,the data will string,to take the

data as it is,we will implement a process called "type conversion/type casting"

type conversion/type casting means "convert the data from one data type to another data

type"

convert given string data into integer:
=============================

in python to convert the given string data into integer,we will use a function called

"int()"

convert given string data into float/real number:
=====================================

in python to convert the given string data into float/real number,we will use a function called

"float()"

convert given string data into complex number:
=====================================

in python to convert the given string data into complex number,we will use a function called

"complex()"

convert the given string data into boolean data:
=====================================

in pytohn,to convert the given string data into boolean data,we will use a function called

"bool()"

convert the given integer data into float data:
===================================

in python,to convert the given integer data into float data,we will use a function called

"float()"

convert the given float data into integer:
===================================

in python,to convert the given float data into integer data,we will use a function called

"int()"

convert the given integer data into complex data:
======================================

in python,to convert the given integer data into complex data,we will use a function called

"complex()"

convert the given float data into complex data:
======================================

in python,to convert the given float data into complex data,we will use a function called

"complex()"

note:
====

we can not convert the given complex number into integer data

we can not convert the given complex number into float/real number data

to convert the given integer data into string data,we wil use "str()" function

to convert the given float/real number into string data,we will use "str()" function

to convert the given integer data into boolean data,we will use "bool()" function to convert the given float/real number into boolean data,we will use "bool()" function

from	to	fucntion
string	integer	int()
string	floating	float()
string	boolean	bool()
string	complex	complex()
integer	floating	float()
floating	integer	int()
integer	boolean	bool
floating	boolean	bool

python programs:
==============

1.write a python program to display a message "hello world"
===

program:
=======

```
#program for displaying "hello world"
print("hello world")
```

2.write a python program to demonstrate different type of data in python:
===

program:
=======

```
#program for displaying different types of data
a=10# integer data
print(a)
a=1.234 #floating/real number data
print(a)
a="hello" #string data
print(a)
a=10+5j #complex number
print(a)
a=True  #boolean data
print(a)
a=None #none
print(a)
```

3.write a python program to demonstrate type of the data in python:
==

in python,to find the type of the data,we will use a function called "type()"

program:
=======

```
#program for displaying different types of data
a=10
print(type(a))#<class int>
a=1.234
print(type(a))#<class float>
a="hello"
print(type(a))#class str>
a=10+5j
print(type(a))#<class complex>
a=True
print(type(a))#<class bool>
a=None
print(type(a))#<class None>
```

4.write a python program to demonstrate python litreals:
===

program:
=======

```
#program for displaying different litreals in python
a=10
print(type(a))#integer lireal
a=1.234
print(a)#float litreal
a="hello"
print(a)#string litreal
a=10+5j
print(a)#complex litreal
a=True
print(a)#boolean litreal
a=None
print(a)#None litreal
a=0b10101
print(a)#binary litreal(it displays in decimal)
a=0o127
print(a)#octal litreal(it displays in decimal)
a=0xabcd
print(a)#hexadecimal litreal(it displays in decimal)
```

note:
====

in python,we will have the the following three number conversion functions:
===

bin()====>it is used to convert the given number into binary number

oct()====>it is used to convert the given number into octal number

hex()====>it is used to convert the given number into hexa decimal number

```
program:
========

#convert the given number into binary number
a=bin(123)
print(a)#0b1111011
#convert the given number into octal number
a=oct(127)
print(a)#0o177
#convert the given number into hexadecimal number
a=hex(100)
print(a)#0x64


write  a python to take  data from the user:
==================================

program:
=======

#program to take the data from the user
#take data using input()
a=input("enter the data for a:")
b=input("enter the data for b:")
print(a,b)
print("type of a is:",type(a))#string data
print("type of b is:",type(b))#string data
#to convert the given string data into integer
a=int(a)
b=int(b)
print("type of a is:",type(a))#integer data
print("type of b is:",type(b))#integer data

write a python to convert given integer data into float data:
==============================================

program:
=======

#convert the given integer data into float data
#take data using input()
a=int(input("enter the data for a:"))#10
b=int(input("enter the data for b:"))#20
print(a,b)
print("type of a is:",type(a))#intger data
print("type of b is:",type(b))#integer data
#to convert the given integer data into float
a=float(a)
b=float(b)
print(a,b)
print("type of a is:",type(a))# 10.0 float data
print("type of b is:",type(b))# 20.0 float data


write  a python to demonstrate airthmetic operators in python:
=================================================

program:
```

```
=======

#working with airthmetic operators
#take data using input()
a=int(input("enter the data for a:"))#10
b=int(input("enter the data for b:"))#20
print("a+b:",(a+b))#30
print("a-b:",(a-b))#-10
print("a*b:",(a*b))#200
print("a/b:",(a/b))#0.5
print("a%b:",(a%b))#10
print("a//b:",(a//b))#0
print("a**b:",(a**b))#100000000000000000000
```

write a python program to demonstrate relational operators:
===

program:
=======

```
#working with relational operators
#take data using input()
a=int(input("enter the data for a:"))#10
b=int(input("enter the data for b:"))#20
print("a>b:",(a>b))#False
print("a<b:",(a<b))#True
print("a>=b:",(a>=b))#False
print("a<=b:",(a<=b))#True
print("a==b:",(a==b))#False
print("a!=b:",(a!=b))#True
```

write a python to demonstrate logical operators:
=====================================

program:
=======

```
#working with logical operators
#take data using input()
a=int(input("enter the data for a:"))#10
b=int(input("enter the data for b:"))#20
print("(a>b) and (a<b):",(a>b) and (a<b))#False
print("(a<b) and (a>b):",(a<b) and (a>b))#False
print("(a>b) or (a<b):",(a>b) or (a<b))#True
print("(a<b) or (b>a):",(a<b) or (b>a))#True
print("not((a>b) or (a<b)):",not((a>b) or (a<b)))#False
```

write a python to demonstarate assignment operator:
==

```
#program for assignment operator
a=10# here we assign 10 value to variable a
print(a)#10
a+=10#a=a+10=20
print(a)#20
a-=10#a=a-10=20-10=10
```

```
print(a)#10
a*=10#a=a*10=100
print(a)#100
a/=10#a=a/10=100/10=10
print(a)#10
a%=10#a=a%10=10%10=0
print(a)#0
```

write a python program to demonstrate identity operator:
===

program:
=======

```
#program for identity operators
x=10
y=20
z=30
print(x is y)#False
print(x is not y)#True
print(x is z)#False
print(x is not z)#True
print(y is y)#True
print(x is x)#True
print(z is not z)#False
```

write a python program to demonstrate membership operators:
==

```
#program for membership operators
x="i am from hyderabad"
y="am"
z="hyderabad"
p="From"
print(y in x)#True
print(z in x)#True
print(p not in x)#True
print( z not in x)#False
print(x in y)#False
print( y in y)#True
print( z in z)#True
```

write a python program to demonstrate bitwise operators:
===

program:
=======

```
#program for bitwise operators
a=10
b=20
print(a|b)#30
print(a&b)#0
print(a^b)#30
print(a<<2)#40
print(a>>2)#2
print(~a)#-11
print(~b)#-21
```

write the programs for the following(using operators,take input from user for every program):
==

1.find the sum of the two numbers:
============================

```
#program for sum of two numbers
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
result=a+b
print("the sum of given a and b is:",result)
```

note:
====

```
#program for sum of two numbers without using "+"
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
result=a-(-b)
print("the sum of given a and b is:",result)
```

2.find the average of the given three numbers:
====================================

program:
=======

```
#program for average of three number
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
c=int(input("enter the value for b:"))
result=(a+b+c)/3
print("the sum of given a and b is:",result)
```

3.find the simple interest:(PTR/100)
============================

program:
=======

```
#program for simple interest
principle_amount=int(input("enter the value for principle:"))
rate_of_interest=int(input("enter the value for rate of interest:"))
time_period=int(input("enter the value for timeperiod:"))
result=(principle_amount*time_period*rate_of_interest)/100
print("the sum of given a and b is:",result)
```

4.find the area of triangle:
====================

5.find the area of cirlce

6.find the farenheit value for given celcius value(c*(9/5)+32)

7.convert the given meters into miles

9.convert given seconds into days

10.convert given hours into years

11.swap two numbers

12.swap two numbers without using temporary variable

===

1.write a python program to demonstrate "if statement":
==

program:
=======
```
#program to demonstrate "if statement"
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
if a>b:
    print("the value of a is greter than b")
if a<b:
    print("the value of b is  greter than a")
```

2.write a python program to demonstrate "if-else statement":
===

program:
=======

```
#program to demonstrate "if-else statement"
a=int(input("enter the value for a:"))#10
b=int(input("enter the value for b:"))#20
if a>b:#if condition is true,"if" will execute
    print("the value of a is greter than b")
else:#if condition is false,else will execute
    print("the value of b is  greter than a")
```

3.write a python program to demonstrate "else-if ladder":
==

program:
=======

```
#program to demonstrate "else-if ladder"
a=int(input("enter the value for a:"))#10
b=int(input("enter the value for b:"))#20
c=int(input("enter the value for c:"))#30
if a>b and a>c:
    print("A is maximum/largest")
elif b>c:
    print("b is maximum/largest")
else:
    print("c is maximum/largest")
```

4.write a python program to demonstrate while loop in python:
===

program:
=======

```
#program to demonstrate "while loop"
a=1
while a<=10:
    print(a,end=" ")#1 2 3 4 5 6 7 8 9 10
    a=a+1
```

5.write a python program to demonstrate for loop in python:
===

program:
=======

```
#program to demonstrate "for loop"
for i in range(1,10):#1 to 9
    print(i,end=" ")
print()#for new line
for i in range(5,10):#5 to 9
    print(i,end=" ")
print()#for new line
for i in range(-4,4):#-4 to 3
    print(i,end=" ")
print()#for new line
for i in range(10,4):#empty/no data
    print(i,end=" ")
```

write python programs for the following:(using conditional and looping)
==

1.check given number is positive or negative:

program:
=======

```
#program to check given number is positive or negative
a=int(input("enter the number:"))
if a>0:
    print("given number is positive")
elif a<0:
    print("the given number is negative")
else:
    print("the given number neither +ve or -ve")
```

2.check given number is end with zero or not:

program:
=======

```
#program to check given number ends with 0 or not
a=int(input("enter the number:"))
if a%10==0:
    print("given number is ends with zero")
else:
    print("the given number is not ends with zero")
```

3.check given number is divisible by 5 or not:

program:
=======

```
#program to check given number divisible by 5 or not
a=int(input("enter the number:"))
if a%5==0:
    print("given number divisible by 5 ")
else:
    print("given number not divisible by 5")
```

4.check given number is with in the given range or not:
===

program:
=======

```
#program to check given number with in the range or not
start=int(input("enter the start:"))
end=int(input("enter the end:"))
num=int(input("enter the number:"))
if num>=start and num<=end:
    print("given number is with in the range")
else:
    print("the given number is not in range")
```

5.check given number is even or odd:
==============================

program:
=======

```
#program to check given number even or odd
a=int(input("enter the number:"))
if a%2==0:
    print("given number is even ")
else:
    print("given number is odd")
```

6.check given number is not divisible 5 and 8:
====================================

program:
=======

```
#program to check given number not divisible by 5 and 8
a=int(input("enter the number:"))
if a%5!=0 and a%8!=0:
    print("given number not divisible by 5 and 8 ")
else:
    print("given number divisible by 5 and 8 ")
```

7.print the number between 1 to 100,which are even(write logic using two loops)

program:
=======

```
start=1
end=100
#using while loop
while start<=end:
    if start%2==0:
        print(start,end=" ")
    start=start+1
print()
start=1
#using for loop
for i in range(start,end+1):
    if i%2==0:
        print(i,end=" ")
```

8.print the numbers between 1 to 100,which are odd(" ")

9.print the number between 1 to 100,which are divisble by 2,4 and 6

10.print the number between 1to 100,which are divisible by 10 and display numbers

in reverse

11.find the maximum of given three numbers

12.find the minimum of given three numbers

13.find the given two numbers are equal or different
==

python functions:
==============

function is a "block of/collection of group statements and which are used to perform a

spesified task".

in python ,we will use functions,for the following:
======================================

1.to avoid the code duplication/code reduancy

2.we can increase the code maintainability in programming/project development

in python,we will have three types of functions:
=====================================

1.user defined functions/custom functions(the functions which are created by the programmer/developer)

2.lambda functions(these functions are also created by the programmer or developer)

3.built-in functions(these functions will given by the python language (modules,packeages))

to create the function in python,we will use the following syntax:
==

```
def function_name(arg1,arg2,arg3,.........argn):

        ===================================
            #here we write the logic/code for function
       ====================================
```

in the above syntax:

def is a keyword/reserve word and it is used to create the function

function_name refers to "name of the function" and it used to execute the function

it will give the identification from others function

every function name follows identifier rules

every function will take or recive the arguments/data from the calling/at execution

these are arguments are can be two types:
=================================

1.formal arguments(arguments which created in the function or arguments of function)

2.actual arguments(the arguments which are passed to function while execution)

arguments====>data or values

note:
====

in python,we can create any number of functions

every function must have a unique name

every function no need to have arguments(means it may /may not have arguments)

so,the arguments of function are optional)

how to execute the function in python:
==============================

in python,we will execute the the function,with help of "name of the function" and simply we

can call the funciton,function automatically executed,function may give the result based

on logic/code,what we written inside the function

```
syntax:
======

function_name(Arg1,arg2........argn) <=== here we are calling the function with
arguments

or

function_name()<==== here we are calling the function without arguments

note:
====

the arguments what we use in the function calling are known as "actual arguments"

example:
=======

#creating the function
def display():#here display is the function name
    print("this is my first function in python")
display()#here it is function calling

how manys we will have to write the/create the function in python:
====================================================

in python,we will have four ways ,we can write/create the function:
====================================================

1.function without arguments and without return type:
==========================================

in this type,function will not take any argument and will not given any result as
return

simple execute the code,what programmer or developer writes in the code of the
function

example:
=======

#function without arguments and without return type
def display():
    a=int(input("enter the value for a:"))
    b=int(input("enter the value for b:"))
    print("the sum is:",a+b)
#here we are calling function "display()"
display()

2.function with arguments and with out return type:
=========================================

in this type,function will  take  arguments and will not given any result as return

simple execute the code,what programmer or developer writes in the code of the
function

example:
```

=======

```
#creating the function with arguments and without return type
def display(a,b):#a=x,y=b
    print("the sum is:",a+b)#300
#take the data for x and y
x=int(input("enter the data for x:"))#100
y=int(input("enter the data for y:"))#200
#calling the function display()
display(x,y)
```

3.function with arguments and with return type:
======================================

in this type,function will take any argument and will give result as return to the

where we are calling the function in the program

example:
=======

```
#creating the function with arguments and with return type
def display(a,b):#a=x,y=b
    return a+b
#take the data for x and y
x=int(input("enter the data for x:"))#100
y=int(input("enter the data for y:"))#200
#calling the function display()
result=display(x,y)
print("the sum is:",result)#300
```

4.function without argument and with return type:
=======================================

in this type,function will not take any argument and will give result as return to the

where we are calling the function in the program

example:
=======

```
#creating the function with out arguments and with return type
def display():
    #take the data for x and y
    x=int(input("enter the data for x:"))#100
    y=int(input("enter the data for y:"))#200
    return x+y
#calling the function display()
result=display()
print("the sum is:",result)#300
```

note:
====

after execution of any function,function may return the result

to return any result by the function,function will return the result with help of "return" keyword

this return will send the result,where the function is calling

in functions,the return statement will always last statement in the function,after return

we never write any code

any function may have only one return statement and we can say return is itself exit for

any function

lambda functions:
==============

lambda function is also a function and it is also called as "anonymous function"

this function will not have any name like normal function

this function by default "function with arguments and return type"

this function will take any number of arguments,but it have only line as code

to create this function in python,we will use a keyword called "lambda"

to create lambda function in python,we will use the following syntax:
===

```
lambda arg1,arg2,arg3,...argn:expression/code
```

example:
=======

```
#create the lambda function
result=lambda x:x+10
print(result(10))#x===>10,result=10+10=20
result=lambda x,y:x+y
print(result(100,200))#x==>100,y==>200,result==>300
result=lambda x,y,z:x+y+z
print(result(100,200,300))#x=>100,y=>200,z=>300,result=600
```

check given number is even or odd using functions(all 4 types):
===

1.function without arguments and without return type:
===

```
def even_or_odd():
    number=int(input("enter the number:"))
    if number%2==0:
        print("the given number is even")
    else:
        print("the given number is odd")
even_or_odd()
```

2.function without arguments and with return type:
===

```
def even_or_odd():
    number=int(input("enter the number:"))
    if number%2==0:
        return "the given number is even"
    else:
        return "the given number is odd"
result=even_or_odd()
print(result)
```

3.function with arguments and with return type:
=====================================

```
def even_or_odd(number):
    if number%2==0:
        return "the given number is even"
    else:
        return "the given number is odd"
num=int(input("enter the number:"))
result=even_or_odd(num)
print(result)
```

4.function with arguments and with out return type:
==

```
def even_or_odd(number):
    if number%2==0:
        print("the given number is even")
    else:
        print("the given number is odd")
num=int(input("enter the number:"))
even_or_odd(num)
```

write the programs for the following using functions(with all 4 types):
==

1.check given number is positive or negative

2.find the area of triangle

3.check given character is vowel or not

4.check given number is with in the given range or nor

5.find the simple interest

6.find the average of the given 5 numbers

7.find the percentage of the given 6 subjects

8.print the numbers from 1 to 100

9.print the all squares of the number between the 1 to 50

10.check given two numbers or same or not

11.check how many numbers are even between 1 to 100

12.check how many numbers are odd between 1 to 100

13.check given how many numbers are 5 multiples between 1 to 100

14.check given how many numbers are end with 5 in between 1 to 100

15 check given how many numbers are ends with zero between 1 to
100SSSSSSSSSSSSSSSS

python modules :
==============

module is a collection of "data and functions"

the data in the module may be variable,list,tuple,set,string,.....................

the functions in module are user defined functions

with help of module,we can able to use data or functions of one python file , in another

python file

module is nothing but "python file with data or methods"

to work with modules,we will use the following steps:
==

step-1:create a python with some data or functions

step-2:save the file with "some_name.py"

step-3:create another python file and access the data/functions of another python using

```
          import keyword

          syntax for importing the module:

          import module_name( here we will give the python file name)
```

step-4: we can access the module data or module function with help of the following syntax:

```
          module_name.data_name or module_name.function_name
```

note:
====

the python file "what is giving the data or functions,is known as module"

example:
=======

step-1:
======

creating python file with data and functions and saving the file with name as

```
"mymodule.py":
==========================================================================

mymodule.py
===========

#create the data
x=100
y=200
z=300
def sum(a,b):
    print("the sum is :",a+b)
def mul(a,b):
    print("the multiplication is:",a*b)
def div(a,b):
    print("the division is:",a//b)

step-2:
=====

create the another python file with name "test.py":
========================================

test.py
======
import mymodule
print(mymodule.x)
print(mymodule.y)
print(mymodule.z)
mymodule.sum(10,20)
mymodule.mul(10,20)
mymodule.div(10,20)

in python when  we short-hand the name of the module, in where we are importing,but
when

we short-hand the name of the module,it not change the original name of the module

short-hand the name of the module is nothing but "aliasing the name of the module"

we will do aliasing in python,with help of "as" keyword in python

syntax for aliasing in python using as keyword:
=====================================

import module_name as short-hand_name or alias_name

example:
=======

import mymodule as mm
print(mm.x)
print(mm.y)
print(mm.z)
mm.sum(10,20)
mm.mul(10,20)
mm.div(10,20)

if we want to access only spesific data or methods from the module,we will use a
```

keyword

called "from"

example:
=======

```
from mymodule import x,y,z,mul
print(x)
print(y)
print(z)
mul(10,20)
```

note:
====

from keyword allow the programmer or developer can get a spesific data or spesific functions

from the module

when we use from keyword,we can access with direct name of the data or function,in where

we are importing

if the functions which are created by the programmer or developer,then we can say the

functions are user-defined functions or custom functions

if the functions which are given by the python language,then we can say the functions are

buit-in functions

if the modules which are created by the programmer or developer,then we can say the

modules are user-defined modules or custom modules

if the modules which are given by the python language,then we can say the modules are

buit-in modules

buit-in functions:
=============

print()===>it is used to display any output of the program in python

input()===>it is used to take the any input from the user in python

int()====>it is used to convert the given data into integer

example:
=======

```
a=int(input("enter the value for a:"))
print(type(a))
print(a)
#convert the given data into binary
b=bin(a)
print(b)
#convert the given data into octal
c=oct(a)
print(c)
#convert the given data into hexadecimal
d=hex(a)
print(d)
```

float()===>it is used to convert the given data into floating-point number/real number

example:
=======

```
a=float(input("enter the value for a:"))
print(type(a))
print(a)
```

note:
====

float data can not be converted into "binary data"

float data can not be converted into "octal data"

float data can not be converted into "hexadecimal data"

complex():===>it used to convert given number into complex type
========
example:
=======

```
a=10#integer type
#converting a into complex type
a=complex(a)
print(a)#10+0j
print(type(a))#complex
b=1.234
print(type(a))#float
#converting the float value into complex type
b=complex(b)
print(b)
print(type(b))
c=0b101010
#convert the given binary value into complex value
c=complex(c)
print(type(c))
print(a)
d=0o107
#convert the given octal value into complex
d=complex(d)
print(d)
```

```
x=0xabf
#copnvert the given hexadecimal value into complex
x=complex(x)
print(x)

example:
=======

a=10+40j#the structure of the complex number will be a+bj
print(a)
#top convert the given complex number into integer
a=int(a)
print(a)


note:
====

any complex number can not be converted into integer or real/floating numbers

any integer/real-float number can be converted into complex number

binary/octal/hexa decimal number can be converted into complex number(these
converted

first into decimal integer,and then complex number)

str():====>it is used to convert the given data into string
====
a=10
print(type(a))
#convert the given integer data into string data
a=str(a)
print(type(a))
b=1.234
#convert the given float data into string data
print(type(b))
b=str(b)
print(type(b))
c=True
#convert the given boolean data into string data
print(type(c))
c=str(c)
print(type(c))
d=None
#convert the given none data into string data
print(type(d))
d=str(d)
print(type(d))

when we are working with string data,if we apply "+" on the string data

string are concatenated

example:
=======

a="hello"
b="world"
```

```
print(a+" "+b)#helloworld
a='10'
b='20'
print(a+b)#1020
a="hello"
b=10
print(a+b)#error
```

bin()===>it is used to convert the given data into binary

example:
=======

```
x=10
print(x)#it is in decimal form
x=bin(x)
print(x)
y=1.234
#y=bin(y)
#print(y)
z="10"
#z=bin(z)
#print(z)
p=True
p=bin(p)
print(p)
```

note:
====

when we working with bin() function,the data has to be "integer"

float data or string data we are not allowed to give as data to the bin() function

we can boolean data also as data to convert given data into binary

True===>1

False===>0

oct()====>it is used to convert the given data into octal number

example:
=======

```
x=10
print(x)#it is in decimal form
x=oct(x)
print(x)#it is in octal form
y=1.234
#y=oct(y)
#print(y)
z="10"
#z=oct(z)
#print(z)
p=True
```

```
p=oct(p)
print(p)
```

note:
====

when we working with oct() function,the data has to be "integer"

float data or string data we are not allowed to give as data to the oct() function

we can boolean data also as data to convert given data into octal

True===>1

False===>0

hex()===>it is used to convert the given data into hexadecimal form

example:
=======

```
x=10
print(x)#it is in decimal form
x=hex(x)
print(x)#it is in hexa decimal form
y=True
y=hex(y)
print(y)
z=1.234
z=hex(z)
#print(z)
```

note:
====

note:
====

when we working with hex() function,the data has to be "integer"

float data or string data we are not allowed to give as data to the hex() function

we can boolean data also as data to convert given data into hexa decimal

True===>1

False===>0

list()===>it is used to convert given iterable into list(where iterable means

              tuple/set/dictionary..............)

tuple()===>it is used to convert given iterable into tuple(where iterable means

              list/set/string/dictionary..............)

```
set()===>it is used to convert given iterable into set (where iterable means

              tuple/list/string/dictionary..............)


frozenset()===>it is used to convert given iterable into forzenset(where iterable
means

              tuple/set/dictionary/list..............)


bool()====>it is used to convert the given data into boolean

example:
=======

x=10
print(x)#integer
x=bool(x)
print(x)#True
y=1.234
y=bool(x)
print(y)
z="hello"
z=bool(x)
print(z)
p=10+5j
p=bool(p)
print(p)
l=0
print(bool(l))
print(bool(10-20))
print(bool(10-10))


dict()===>it is used to convert the given iterable data into dictionary

type()===>it is used to check the "what type of data the variable has"

example:
=======

x=10#integer
print(type(x))#<class int>
x=1.234
print(type(y))#<class float>
x="hello"
print(type(x))#<class str>
x=True
print(type(x))#<class bool>


len()===>it is used to "find the number of elements present in the given iterable"

             where iterable means "list/tuple/string/set/.............."

example:
=======
x="hello"
length=len(x)#5
print(length)

chr()===>it is used to give the cheracter for given number

example:
=======

x=65
print(chr(x))
x=97
print(chr(x))
x=59
print(chr(x))
x=63
print(chr(x))

note:
====

upper case letters starts 65 onwards(A-65,B-66,c-67,......)

lower case letters starts from 97 onwards(a-97,b-98,c-99,d-100)


ord():===>it will give a number for given character

example:
=======

x='a'
print(ord(x))
x='A'
print(ord(x))
x=';'
print(ord(x))


abs():====>it will give the result always positive for any given data
====
                   it will convert even if we give negative data,it will simply
convert the given data

                   into positive

                  abs stands for "Absoulte"

example:
=======

x=-10
print(abs(x))
x=10
print(abs(x))
x=-1.234
print(abs(x))

sum():===> it will give the sum of the given iterable(list/tuple/set....)
```

```
example:
=======

x=[1,2,3,4,5,6,7,8,9,10]#list
print(sum(x))#55
x=(10,20,30,40,50,60,70,80,100)#tuple
print(sum(x))#460
x={1,2,3,4,5,6,7,8,9,100,200,300,400}#set
print(sum(x))#1045

note:
====

given data must be "iterable" for sum() function


max():===> it will give the maximum of the given iterable(list/tuple/set...)

example:
=======

x=[1,2,3,4,5,6,7,8,9,10]#list
print(max(x))
x=(10,20,30,40,50,60,70,80,100)#tuple
print(max(x))
x={1,2,3,4,5,6,7,8,9,100,200,300,400}#set
print(max(x))

note:
====

given data must be "iterable" for max() function

min()===>it will give the minimum of the given iterable(list/tuple/set....)

example:
=======

x=[1,2,3,4,5,6,7,8,9,10]#list
print(min(x))
x=(10,20,30,40,50,60,70,80,100)#tuple
print(min(x))
x={1,2,3,4,5,6,7,8,9,100,200,300,400}#set
print(min(x))

note:
====

given data must be "iterable" for min() function


divmod()====>it will give the both quotient and remainder as result for given two
numbers

example:
=======

x=divmod(10,20)#===>(10//20,10%20)===>(0,10)
```

```
print(x)
x=divmod(20,10)#===>(20//10,20%10)===>(2,0)
print(x)

pow()===>it is used to find the power for given base and exponent

example:
=======

x=pow(2,3)#power(base,exponent)====>8
print(x)
x=pow(20,4)#pow(base,exponent)=====>160000
print(x)

any()===>it is used to find ,in the given iterable,any one of the value is
true,then any()

                function will return "True",otherwise "False"(no value is true in
the iterable)

                this function also takes data as iterable

example:
=======

x=[1,2,3,4,5,6,7,8]
print(any(x))#True
x=[0,0,0,0,0,0,0,1]
print(any(x))
x=[0,0,0,0]
print(any(x))

all()===>it is used to find ,in the given iterable,all values are true in the given
itretable is

             ,then all() function will return "True",otherwise "False"(any one
value is false in the

             given iterable)

              this function also takes data as iterable


example:
=======

x=[1,2,3,4,5,6,7,8]
print(all(x))#True
x=[1,2,3,4,5,0]
print(all(x))#True
x=[1,2,3,4,5-10,-30,-100]
print(all(x))#True

exec():

this function is used to execute given code in the form of string

example:
========
exec('a=100')
exec('b=200')
exec('c=300')
exec('print(a+100)')#200
exec('print(a+b)')#300
exec('print(a+b+c)')#600
```

eval():

this function will used to execute given expression in the form string

example:
=======

```
a=10
b=20
c=30
print(eval('a+b'))
print(eval('a*b'))
print(eval('a//b'))
```

round():

this function will used to round of the given float or real number

example:
=======

```
x=12.5
print(round(x))
x=12.6
print(round(x))
x=12.345
print(round(x,2))
x=12.45678
print(round(x,1))
x=12.45678
print(round(x,3))#12.457
print(round(x,4))#12.4568
```

built-in modules:
=============

math module:
===========

this module we will use ,to perform all mathematical operations

in this module,we will have the following functions:
===

1.sqrt()<=== it is used to find the "square root of the given number"

 example:
========

```
import math as mt
print(mt.sqrt(100))#10.0
print(mt.sqrt(16))#4.0
print(mt.sqrt(1296))#36.0
print(mt.sqrt(625))#25.0
```

2.ceil()<=== it will take a real number as input and it will give "next highest integer number

as result"

example:
========

```
import math as mt
print(mt.ceil(5.67))#6
print(mt.ceil(6.78))#7
print(mt.ceil(9.1))#10
print(mt.ceil(4.0))#4
```

3.floor()<=== it will take a real number as input and it will give "before lowest integer number

as result"

example:
=======

```
import math as mt
print(mt.floor(5.67))#5
print(mt.floor(6.78))#6
print(mt.floor(9.1))#9
print(mt.floor(4.0))#4
```

4.trunc()<===it is used to remove the real/fractional part from the given real/floating-point

number

example:
=======

```
import math as mt
print(mt.trunc(5.6756789))#5
print(mt.trunc(6.78))#6
print(mt.trunc(9.1))#9
print(mt.trunc(4.0))#4
```

5.fsum()<==== it is used to sum the given data

example:
=======

```
import math as mt
print(mt.fsum([1,2,3,4,5,6,7,8,9,10]))
print(mt.fsum((1,123,456,789,345,678)))
print(mt.fsum({100,200,300,400,500,600}))
```

6.fabs()<=== it used to get given number in the form positive number

fabs() will convert given number into positive number

example:
=======

```
import math as mt
print(mt.fabs(-11))
print(mt.fabs(0))
print(mt.fabs(1))
```

7.prod()<=== it is sued to find the product of the given numbers

example:
=======

```
import math as mt
print(mt.prod([1,2,3,4,5]))
print(mt.prod((100,200,300,400)))
print(mt.prod({1,2,3,4,5,6,7}))
```

8.pow()<=== it is used to find the power of the given base and exponent

example:
=======

```
import math as mt
print(mt.pow(2,3))
print(mt.pow(1,8))
print(mt.pow(2,-3))
```

9.exp()<=== it is used to find the exponential of the given number

example:
=======

```
import math as mt
print(mt.exp(2))
print(mt.exp(-2))
print(mt.exp(1))
```

10.factorial()<=== it is used to give the factorial of the given number

example:
=======

```
import math as mt
print(mt.factorial(5))
print(mt.factorial(6))
print(mt.factorial(10))
```

11.gcd()<=== it is used to find the gcd of the given two numbers

example:
=======

```
import math as mt
print(mt.gcd(100,200))
print(mt.gcd(10,20))
print(mt.gcd(4,64))
```

```
12.perm()<=== it is used to find the permutation for given two numbers

example:
=======
import math as mt
print(mt.perm(10,4))

13.comb()<=== it is used to perform the combination of the given two numbers

example:
=======

import math as mt
print(mt.comb(10,4))


in math module,we can also have the functions which are related to "trignometry"
are like:
====================================================================

sin()

cos()

tan()

sinh()

cosh()

tanh()

in math module,we will also have logerthmetic fucntions:
============================================

log()

log2()

log10()


in this module,we will also have the following constants:
============================================

example:
=======
import math as mt
print(mt.pi)
print(mt.tau)
print(mt.inf)

example:
=======
import math as mt
radius=10
print(mt.pi*radius*radius)
```

Random module:
=============

this module will give the random number for a particular range

in python,if we want to work with "random numbers",then we will use "random"

module

to use the random module,we will follow the following syntax:
===

```
import random
```

in random module we will have the following functions:
==

1.random():
=========

this function will give the random number between 0 to 1

this fucntion may generate random number as 0,but not 1

example:
=======

```
import random
print(random.random())
```

example-2:
========

```
import random
#print(random.random())
#using random(),generate integer random number between 1 to 10
print(int((random.random())*10))
#using random(),generate integer random number between 1 to 100
print(int((random.random())*100))
#using random(),generate integer random number between 1 to 1000
print(int((random.random())*1000))
#using random(),generate integer random number between 1 to 10000
print(int((random.random())*10000))
#using random(),generate integer random number between 1 to 100000
print(int((random.random())*100000))
```

randint(start,end):
==============

randint() fucntion will generate random number in a integer format for given range

syntax for randint():
===============

```
import random as rd

rd.randint(start,end)
```

```
example:
=======
import random as rd
print(rd.randint(1,10))
print(rd.randint(1,100))
print(rd.randint(1,1000))
print(rd.randint(1,10000))
print(rd.randint(1,100000))


note:

when we are using randint(), it will give both start and end also as random number

randint(1,10)===>it may give give random number 1,2,3,4,5,6,7,8,9,10


randrange(start,end,step):
====================

it will give the random number for given range as well as step size  given in the
range

syntax for randrange():
=================

import random as rd

rd.randrange(start,end,step)

example:
=======
import random as rd
print(rd.randrange(1,100,4))
print(rd.randrange(0,10,2))
print(rd.randrange(1,1000,3))

example:
=======

import random as rd
print(rd.randrange(1,10,2))
print(rd.randrange(0,10,2))
print(rd.randrange(1,100000,6))


note:
====

randrange() will give start number as random number,but not end number

randrange(1,10,1)===>it may give the random number as 1,2,3,4,5,6,7,8,9


choice():
=======

this fucntion will give the random number for given sequence type of
data(list,tuple or string)
```

```
if we want to get a random number,for given set of numbers only,then in python we
will use

choice() function

syntax:

import random as rd

rd.choice(list_name/tuple_name/string_name)


example:
=======

import random as rd
c1=[1,2,3,4]
print(rd.choice(c1))
c2=[1,2,3,4,"hello","hai","raj",1.234]
print(rd.choice(c2))

shuffle():
======

this function is going to shuffle the "given sequqnce of data (it may be
list/tuple/string)"

syntax:

import random as rd

rd.shuffle()

example:
=======
import random as rd
l1=[1,2,34,56,78,90,100]
rd.shuffle(l1)
print(l1)#[90, 34, 2, 100, 1, 78, 56]
rd.shuffle(l1)
print(l1)#[1, 56, 2, 100, 90, 34, 78]
rd.shuffle(l1)
print(l1)#[1, 78, 100, 56, 90, 2, 34]

to know all functions of the "Random" module we will use the following code:
============================================================

code:
====
import random
functions_info=dir(random)
print(functions_info)

['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST',
'SystemRandom', 'TWOPI', '_Sequence', '_Set', '__all__', '__builtins__',
'__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', '_accumulate', '_acos', '_bisect', '_ceil', '_cos', '_e', '_exp',
'_inst', '_log', '_os', '_pi', '_random', '_repeat', '_sha512', '_sin', '_sqrt',
'_test', '_test_generator', '_urandom', '_warn', 'betavariate', 'choice',
'choices', 'expovariate', 'gammavariate', 'gauss', 'getrandbits', 'getstate',
'lognormvariate', 'normalvariate', 'paretovariate', 'randint', 'random',
'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular', 'uniform',
'vonmisesvariate', 'weibullvariate']
```

time module in python:
==================

this module is related to time related information or to process with any time in the python

program,we will use the "time module"

syntax for time module:
===================

```
import time as t
```

in this module,we will have the following functions:
==

time():
=====

it will give the cureent time "time in the seconds"

syntax:
======

```
import time as t

current_time=t.time()
```

code:
====

```
import time as t
current_time=t.time()
print(current_time)
```

ctime():<=== it is used to convert the given time into human readable form

code:
====

```
import time as t
current_time=t.time()#here we will get the time in seconds
#here we will get the time in human readbale form
current_time=t.ctime(current_time)
print(current_time)
```

ctime() will return the "timestamp" as result (timestamp means both date and time)

sleep()<=== it is used to supend the execution of a program for a particular time

```
slice

syntax:
======

import time

time.sleep(time_slice)

example:
=======

import time as t
for i in range(1,10):
    t.sleep(2)
    print(i)


strftime()<=== it is used to format the given time

to display time and date,it will uses the following format spesifiers:
====================================================

%Y===>YEAR

%m==> MONTH

%d==>  DATE

%H===>HOUR

%M===>MINTUTE

% S===>SECONDS

syntax:
======

import time as t

t.strftime(format,time)

note:
====

when we working with strftime() function,we need to convert the timestamp into
local time

to convert the timestamp into localtime,we will use a fucntion called "localtime()"

example:
=======
import time as t
#get the timestamp in seconds using time()
timestamp=t.time()
#convert the timestamp in seconds to localtime using localtime()
local_time=t.localtime(timestamp)
#format the local time using strftime()
date_time=t.strftime("%Y-%m-%d %H:%M:%S",local_time)
```

```
print(date_time)

perf_counter()(performance counter):

to measure the a piece of code or a code  which is taking how much time,in python
we will

use "perf_counter()"

syntax:

import time as t

start=t.perf_counter()

#write the code here

end=t.perf_counter()

=========================================================================

total_time=end-start


example:
=======
import time as t
#start time
start=t.perf_counter()
for i in range(1,5):
    print(i)
#end time
end=t.perf_counter()
#total time takes for execute the for loop
total_time=end-start
print("the total taken by the for loop:",total_time)

gmtime():<=== this function will give the time in the "UTC" format for given
timestamp

code:
====

import time as t
#get the time stamp
timestamp=t.time()
#convert the timetsmp into utc format
utc_time=t.gmtime(timestamp)
print(utc_time)

datetime module:
==============

this module will work with " both date and time" related information in python

in this we will have various functions which are deals with "date and time"

syntax:
```

```
import datetime as dt

now():
=====

this function will  give the current date and time(timestamp)

example:
=======

import datetime as dt
print(dt.datetime.now())


we can also give our own date and time to the "datetime() " fucntion

example:
=======

import datetime as dt
date_time=dt.datetime(2023,7,8,12,34,56)
print(date_time)


how to work with only date using datetime module:
========================================

in datetime module,we can work with only "date" also

we work with only "date" in datetime module,as follows:
===========================================

from datetime import date
#below function will display today date
date=date.today()
print(date)
#we are display to today day in a month
day_in_month=date.day
print(day_in_month)


how to work with only time using datetime module:
========================================

in datetime module,we can work with only "time" also

we work with only "time" in datetime module,as follows:
===========================================

from datetime import time,date
#create the time using time class
t=time(12,58,45)
print(t)
d=date(2021,8,20)
print(d)
#start_date
sd=date(2023,7,26)
#end_date
ed=date(1998,8,21)
```

```
difference=sd-ed
print(difference)


how to find the difference between two dates:
===================================

from datetime import date
sd='2023-07-26'
ed='1998-08-21'
#convert the date into native format
sd=date.fromisoformat(sd)
ed=date.fromisoformat(ed)
#find the difference
diff_in_days=sd-ed
#print(type(diff_in_days))
data=str(diff_in_days)
data=data[:4]
#print(type(data))
data=int(data)//365.25
print(int(data))


example-2:
=========
from datetime import date,timedelta
sd='2023-07-26'
ed='1998-08-21'
#convert the date into native format
sd=date.fromisoformat(sd)
ed=date.fromisoformat(ed)
#find the difference
diff=sd-ed
days=diff.days
print(int(days//365.25))#years
print(int(days*24))#hours
print(int(days//7))#weeks
print(int(days//30))#months

working with timedelta:
==================

from datetime import timedelta,date
delta=timedelta(4)
print(delta)
date=date.today()
print(date)
#after 4 days,what is date
print(date+delta)
#from today date,what 4 days before date
print(date-delta)

example-2:
========

from datetime import timedelta,date
delta=timedelta(weeks=52,days=1,hours=12,minutes=13)
print(delta)
date=date.today()
```

```
print(date)
print(date+delta)
print(date-delta)
```

format the dattime module's date and time:
=================================

to format the datetime module's date and time,we will use a function called "strftime()"

```
import datetime as dt
date_time=dt.datetime.now()
print(date_time.strftime("%Y"))#year
print(date_time.strftime("%m"))#month
print(date_time.strftime("%d"))#day
print(date_time.strftime("%H"))#hour
print(date_time.strftime("%M"))#minitue
print(date_time.strftime("%S"))#seconds
print(date_time.strftime("%Y-%m-%d"))#date
print(date_time.strftime("%H:%M:%S"))#time
```

python data structures:
==================

in python,if we want to store the data we will use "variable"

one variable can store only one value in python

"n" variables can store "n" values

for example,if i want to store "100" values ,in python program we need to create

the "100" variables

data structure provides a "Way to organize the data in python"

using data structures,we can able to handle the data vary easily and all data items/elements

we can maintain under single name/reference

using data structures,we can able to perform the following vary easily:
==

insertion

traverse/display/retrive the data

updation

deletion

search and other operations

in python,we will have the following elementary data structures/fundamental data structures:
==

1.list

2.tuple

3.set

4.string

5.dictionary

List:
===

==>list is used to store "a collection or group of data under single name"

===>list can store any kind of data

===>list can store integer data/float data/string data/charcater data................

===>list can store any number of data(it means any number of values...)

===>list can be duplicate/list can be have duplicate elements/data

===>list can be empty

===>list can chnage the data at any time in the program

to create the list,in python we will use the following syntax:
==

list_name=[value1,value2,value3,value4,............]

the values/data in the list is known as "list elements"

on list,we can able to perform the following operations:
===

insertion

traverse

update

delete

search

copy......................

if list contain all elements are similar type,then we can say list is "homogeneous"

if list contain all eleemnts are not similar type,then we can say list is "hetrogeneous"

example:
=======

#working with list

```
l1=[1,2,3,4,5,6,7,8,9,10]#integer list
print(l1)
l2=[1.2,3.4,5.6,7.8,9.0]#float/real number list
print(l2)
l3=["abc","bca","cba","def"]#string list
print(l3)
l4=[1,2,3,4,1.2,3.4,5.6,7.8,9.0,"abc","bca","cab"]
print(l4)
l5=[]#empty list
print(l5)
l6=[1,2,3,4,1,2,3,4]#duplicate list
print(l6)
```

how to find the length of the list in python:
==================================

length of the list means "the number of elements present in the list"

to find the length of the list in python,we will use a function called "len()"

len() function is a built-in function

syntax for len() function:
===================

length of the list=len(list_name)

example:
=======

```
#working with list
l1=[1,2,3,4,5,6,7,8,9,10]#integer list
l2=[1.2,3.4,5.6,7.8,9.0]#float/real number list
l3=["abc","bca","cba","def"]#string list
l4=[1,2,3,4,1.2,3.4,5.6,7.8,9.0,"abc","bca","cab"]
l5=[]#empty list
l6=[1,2,3,4,1,2,3,4]#duplicate list
print("length of the list l1 is :",len(l1))#10
print("length of the list l2 is :",len(l2))#5
print("length of the list l3 is :",len(l3))#4
print("length of the list l4 is :",len(l4))#12
print("length of the list l5 is :",len(l5))#0
print("length of the list l6 is :",len(l6))#8
```

how to access the list element one by one or individual elements from the list:
===

in python,if we want to acess the elements from list one by one/individual elements,

we will use a concept called "indexing"

in python,we will have two types of indexing:
===================================

1.positive indexing

2.negative indexing

in positive indexing,the indexing will start from 0 to n-1 [where n is length of the list]

in positive indexing it will take the indexing from left to right

in negative indexing,the indexing will start from -1 to -n [where n is length of the list]

in negative indexing it will take the indexing from right to left

example on positive indexing:
========================

```
#working with list
l1=[10,20,30,40,50,60,70,80,90,100]#integer list
print(l1[0])#1
print(l1[5])#60
print(l1[8])#90
print(l1[9])#100
print(l1[4])#50
print(l1[2])#30
print(l1[1])#20
```

example on negative indexing:
========================

```
#working with list
l1=[10,20,30,40,50,60,70,80,90,100]#integer list
print(l1[-1])#100
print(l1[-4])#70
print(l1[-9])#20
print(l1[-6])#50
print(l1[-3])#80
print(l1[-10])#10
print(l1[-2])#90
print(l1[-5])#60
```

how to display the data in list using while loop:
=====================================

```
#working with list
l1=[10,20,30,40,50,60,70,80,90,100]#integer list
index=0
length=len(l1)#10
while index<length:
    print("l1[",index,"]:",l1[index])#10 20 30 40 50 60 70 80 90 100
    index=index+1
```

how to display the data in list using for loop:
====================================

```
#working with list
l1=[10,20,30,40,50,60,70,80,90,100]#integer list
for data in l1:
    print(data)
```

slicing on the list:
==============

slicing means "to get the data from the list for a particular range"

when we apply slicing,we may one or more elements from the list

slicing will be performed on list,using following syntax:
===

list_name[start:end:step]

here step is how many element we need skip,while slicing

suppose,if write like list_name[start:end],we will get the data from start to end-1

if start is greter than end,the result is nothing

if start and end both are same ,the result is nothing

example:
=======

```
#working with list
l1=[10,20,30,40,50,60,70,80,90,100]#integer list
#positive slicing  the list
print(l1[0:6])#10,20,30,40,50,60
print(l1[0:7])#10,20,30,40,50,60,70
print(l1[1:8])#20,30,40,50,60,70,80
print(l1[5:])#60,70,80,90,100
print(l1[:9])#10,20,30,40,50,60,70,80,90
print(l1[:])#10,20,30,40,50,60,70,80,90,100
```

example-2:
=========

```
#working with list
l1=[10,20,30,40,50,60,70,80,90,100]#integer list
#positive slicing  the list
print(l1[0:6:1])#10,20,30,40,50,60
print(l1[0:7:2])#10,30,50,70
print(l1[1:8:3])#20,50,80
print(l1[5:9:4])#60
print(l1[:9:5])#10,60
print(l1[::4])#10,50,90
```

example-3:
=========

```
#working with list
l1=[10,20,30,40,50,60,70,80,90,100]#integer list
#negative  slicing  the list
print(l1[-6:-1])#50,60,70,80,90
print(l1[-8:-2])#30,40,50,60,70,80
print(l1[-4:-8])#nothing
print(l1[-9:])#20,30,40,50,60,70,80,90,100
print(l1[:-1])#10,20,30,40,50,60,70,80,90
print(l1[::-1])#100,90,80,70,60,50,40,30,20,10
print(l1[::-2])#100,80,60,40,20


example-4:
=========

#working with list
l1=[10,20,30,40,50,60,70,80,90,100]#integer list
#negative  slicing  the list
print(l1[-6:-1:1])#50,60,70,80,90
print(l1[-8:-2:2])#30,50,70
print(l1[-4:-8:3])#nothing
print(l1[-9::4])#20,60,100


operations on the list:
=================

on list,we will perform the following operations:
=====================================

1.insertion

2.updation

3.deletion

4.copying

5.search and other operations


1.insertion operation on the list:
=========================

on list,we can insert the data,in the following ways:
========================================

1.insert at end

2.insert at any position

insert at end:
==========

to add the any element into the list at the end,we will use a function called
"append()"

syntax:
======

list_name.append(element)

example:
=======

#create the list with empty
l1=[]
#add the elements into the list using append()
```

```
l1.append(10)#[10]
l1.append(20)#[10,20]
l1.append(30)#[10,20,30]
l1.append(40)#[10,20,30,40]
l1.append(50)#[10,20,30,40,50]
l1.append(60)#[10,20,30,40,50,60]
print(l1)
```

2.insert at any position:
==================

to insert the element at any position,in python we will use a function called "insert()"

syntax for insert():
==============

```
list_name.insert(index,element_to_insert)
```

example:
=======

```
#create the list
l1=[10,20,30,40,50]
#add the elements into the list using insert()
l1.insert(0,100)#[100,10,20,30,40,50]
print(l1)
l1.insert(3,300)#[100,10,20,300,30,40,50]
print(l1)
l1.insert(5,600)#[100,10,20,300,30,600,40,50]
print(l1)
l1.insert(7,900)#[100,10,20,300,30,600,40,900,50]
print(l1)
```

updation on the list:
================

to update any element in the list,we will use "index"

syntax:
======

```
list_name[index]=new element
```

example:
=======

```
l1=[10,20,30,40,50]
#update the list using index
l1[0]=100
print(l1)#[100,20,30,40,50]
l1[1]=200
print(l1)#[100,200,30,40,50]
l1[4]=600
print(l1)#[100,200,30,40,600]
l1[3]=300
print(l1)#[100,200,30,300,600]
```

```
delete operation on the list:
======================

to perform the delete operation on the list,we will use the following ways:
==========================================================

1.delete the element from end:
=========================

to delete the element from the end,we will use a function called "pop()"

syntax:
======

list_name.pop()

example:
=======

l1=[10,20,30,40,50]
#delete the element from the list at end of the list
l1.pop()#[10,20,30,40]
print(l1)
l1.pop()#[10,20,30]
print(l1)
l1.pop()#[10,20]
print(l1)
l1.pop()#[10]
print(l1)

2.delete the elements from the list based on index:
=========================================

to delete the any element from the list,based on index,we will use "Del" keyword

syntax:

del list_name[index]

example:
=======

l1=[10,20,30,40,50]
#delete the element from the list based on the index
del l1[0]
print(l1)#[20,30,40,50]
del l1[3]
print(l1)#[20,30,40]
del l1[2]
print(l1)#[20,30]



3.delete the element based on the element wise/value wise from the list:
=========================================================

to delete the elements from the list,based on the element wise/value wise ,we will
use
```

```
"remove()"


using remove(),we can delete the values based on the given element/value

syntax:

list_name.remove(element)

example:
========

l1=[10,20,30,40,50]
#delete the element from the list based on element
l1.remove(10)
print(l1)#[20,30,40,50]
l1.remove(40)
print(l1)#[20,30,50]
l1.remove(50)
print(l1)#[20,30]


delete the all elements from the list:
=============================

to delete the all elements from the list,we will use the "clear()" function

syntax:
======

list_name.clear()

example:
=======

l1=[10,20,30,40,50]
l1.clear()#it will remove the all elements
print(len(l1))


to remove the list permenently from the memory,we will use "Del" keyword

syntax:
======

del list_name

example:
=======

l1=[10,20,30,40,50]
del l1 #here we are remove the list premently
print(l1)

how to copy the list in python:
========================

we can copy the list in python using "=" (assignment) operator
```

example:
=======

```
l1=[10,20,30,40,50]
l2=l1
print(l2)
```

how to reverse the list in python:
==========================

to reverse the list in python,we will use a function called "reverse()"

syntax:
======

```
list_name.reverse()
```

example:
=======

```
l1=[10,20,30,40,50]
print(l1[::-1])
l1.reverse()
print(l1)
```

how to find the index of the given element in the list:
===

to find the index of the given element in the list,we will a function called "index()"

syntax:

```
list_name.index(element)
```

example:
=======

```
l1=[10,20,30,40,50]
print(l1.index(10))#0
print(l1.index(50))#4
print(l1.index(30))#2
print(l1.index(20))#1
```

how to find the frequency of the given element in the list:
===

frequency means "how many times the given element is repeated in the complete list"

to find the frequency of the given element in the list,we will use "count()"

syntax:
======

```
list_name.count()
```

example:

```
=======

l1=[10,20,30,40,50,50,60,70,10,20,30,50,70,10,20,30]
print(l1.count(10))#3
print(l1.count(50))#3
print(l1.count(30))#3
print(l1.count(20))#3

how to sort the data in the given list:
=============================

to sort the data in the given list,we will use the "sort()" function


syntax:
======
list_name.sort() ===>Ascedning order

or

list_name .sort(reverse=True) ===>descending order


example:
=======

l1=[10,20,30,40,50,50,60,70,10,20,30,50,70,10,20,30]
l1.sort()
print(l1)
l1.sort(reverse=True)
print(l1)


how to combine the lists in python:
============================

to combine the lists in python,we will use the following ways:
================================================

1.using "+" operator:
================

example:
=======

l1=[10,20,30,40,50]
print(l1)
l2=[60,70,80,90,100]
print(l2)
print(l1+l2)
l3=l1+l2
print(l3)


2.using extend() :
==============

l1=[10,20,30,40,50]
l2=[60,70,80,90,100]
```

```
l3=[110,120,130,140,150]
l1.extend(l2)
l1.extend(l3)
print(l1)
```

nested lists in python:
==================

creating list inside another list is known as "nested list"

in the nested list,every list is an element to main list

example:
=======

```
l1=[[10,20],[30,40],[50,60],[70,80]]
print(l1)
print("length of l1:",len(l1))#4
print(l1[0])#[10,20]
print(l1[2])#[50,60]
print(l1[3])#[70,80]
print(l1[1])#[30,40]
print(l1[1][0])#30
print(l1[2][1])#60
print(l1[0][1])#20
print(l1[3][1])#80
```

example-2:
=========

```
l1=[[10,20],[30,40],[50,60],[70,80]]
print(l1)
print("length of l1:",len(l1))#4
l2=[[[10,20,30]]]
print(len(l1))#4
print(len(l2[0]))#1
print(len(l2[0][0]))#3
l3=[[[[[10,20,30,40,50,60,70]]]]]
print(len(l1))#4
print(len(l3))#1
print(len(l3[0][0]))#1
print(len(l3[0][0][0][0]))#7
```

example-3:
=========

```
l3=[[[[[10,20,30,40,50,60,70]]]]]
print(l3[0][0][0][0][6])#70
print(l3[0][0][0][0][2])#30
print(l3[0][0][0][0][1])#20
```

len() =====>to find the length of the list append() ===>to add the elements at the end of the list

insert() ===>to insert the element at given position

pop()===>to delete the element from the end of the list

remove()===>it remove the given element from the list

clear()===>to clear the all elements from the list

reverse()===>it reverse the given list

sort()===>it will sort the given list

index()===>it will find the index of the given element in the list

count()===>it will found the how many times the given is repeated in the given list

extend()===>it used to combined given two lists

operators on list:
=============

```
l1=[1,2,3,4,5]
l2=l1*3# it copy the list for 3 times in the l2
print(l2)
l3=l1+l2 #combing the lists
print(l3)
```

tuples in python:
=============

tuple is going store the data "one or more values" same like list

tuple can be empty

tuple can be store "duplicate values"

tuple onece created on tuple we can not able to perform "insertion/update/deletion"

tuple will uses indexing same like list

tuple will uses slicing same like list

tuple is a "immutable"data type (immutable means "it can not change/modify once we create")

but list is a "mutable type" (mutuable means once we create,we can change any time)

to create the tuple,in python we will use the following syntax:
===

```
tuple_name=(val1,val2,val3,val4,.............valn)
```

example:
=======

```
t1=(1,2,3,4,5,6,7,8,9,10)
```

```
t2=()
t3=(1.2,3.4,5.6,7.8,9.0)
t4=(1,2,3,4,1.2,3.4,5.6,7.8,9.0,"abc",bca","cab")
```

to find the length of the tuple,we will use "len()" function

example:
=======

```
t1=(1,2,3,4,5,6,7,8,9,10)
t2=()
t3=(1.2,3.4,5.6,7.8,9.0)
t4=(1,2,3,4,1.2,3.4,5.6,7.8,9.0,"abc","bca","cab")
t5=(1,2,1,2,1,2)
print("length of the t1:",len(t1))#10
print("the length of t2 is:",len(t2))#0
print("the length of the t3 is:",len(t3))#5
print("the length of the t4 is:",len(t4))#12
print("the length of the t5 is:",len(t5))#6
```

indexing and slicing on tuple:
=======================

indexing and slicing same like "list"

example:
=======

```
t1=(1,2,3,4,5,6,7,8,9,10)
print(t1[0])#1
print(t1[9])#10
print(t1[5])#6
print(t1[-6])#5
print(t1[-10])#1
print(t1[-2])#9
print(t1[1:4])#(2,3,4)
print(t1[:8])#(1,2,3,4,5,6,7,8)
print(t1[:-6])#(1,2,3,4)
print(t1[-6:])#(5,6,7,8,9,10)
```

we can not able to use the following functions on tuple:
===

append()

insert()

pop()

remove()

sort()

clear()

extend()

reverse()

we can apply the following function on tuple:
===================================

len()

index()

count()

list()

list() is the function is used to convert the tuple into list

whenever we want to change the tuple elements,we need to convert the tuple into list

change the list,then convert the list into tuple using "tuple()" function

example:
=======

```
t1=(1,2,3,4,5,6,7,8,9,10)
#converting the tuple into list
l1=list(t1)
print(l1)
#convert the list into tuple
t1=tuple(l1)
print(t1)
```

how to display the tuple usig loops:
============================

```
#using while loop

t1=(1,2,3,4,5,6,7,8,9,10)
index=0
length=len(t1)
while index<length:
    print(t1[index])
    index=index+1

#using for loop

t1=(1,2,3,4,5,6,7,8,9,10)
for i in t1:
    print(i)
```

how to revese the tuple in python:
===========================

```
reversed_tuple_name=tuple_name[::-1]
```

example:
=======

```
t1=(1,2,3,4,5,6,7,8,9,10)
t2=t1[::-1]
print(t2)
```

questions on list and tuple:
=====================

1.display the list using while loop

2.display the tuple using for loop

3.display the list using for loop

4.display the tuple using while loop

5.display the list in reverse order

6.display the tuple in reverse order

7.print the maximum element in the list

8.print the minimum element in the list

9.print the maximum element in the tuple

10.print the minimum element in the tuple

11.print the even elements in the list

12.print the odd eleemnts in the list

13.print the even index elements in the list

14.print the odd index elements in the list

15.print the sum of the elements in the list

16.print he product of the elements in the list

17.print the sum of the even elements in the list

18.print the sum of the odd elements in the list

19.print the count of the number elements in the list

20.print the count of the even elements in the list

21.prin the count the of odd elements in the list

22.find the average of the list elements

23.find the average of the even elements in the list

24.find the average of the odd elements in the list

25.sort the given list

how to combine the two tuples in python:

================================

```
t1=(1,2,3,4,5,6,7,8,9,10)
t2=(10,20,30,40)
t3=t1+t2
print(t3)
```

list vs tuple:
=========

list is mutable

tuple is immutable

list can be empty

tuple can be empty

list can store duplicate elements

tuple can store duplicate elements

list can have indexing and slicing

tuple can have indexing and slicing

list can have operations like insert,update and delete

tuple can not have operation like insert,update and delete

set:
===

set is used to "store a group or collection of unique elements"

set can not store "duplicate" elements

set can be "empty"

set can store "any kind of data"

set can not uses "indexing like list or tuple"

set can not have "any nested set like list or tuple"

in python,we can create set as follows:
==============================

```
set_name={element1,element2,element3,.............elementn}
```

note:
====

in python,to create the set,we will use "{}" (Curly braces)

example:
=======

```
s1={1,2,3,4}
```

```
s2={1.2,3.4,5.6,7.8,9.0}

s3={"abc","bca","cab","ghj"}

s4={1,2,3,4,1.2,3.4,5.6,7,8,"abc","bca"}

s5={}

example:
=======

s1={1,2,3,4}
print(s1)
s2={1.2,3.4,5.6,7.8,9.0}
print(s2)
s3={"abc","bca","cab","ghj"}
print(s3)
s4={1,2,3,4,1.2,3.4,5.6,7,8,"abc","bca"}
print(s4)
s5={}
print(s5)
s6={1,2,3,1,2,3,1,2,3}
print(s6)#{1,2,3}

on sets in python,we will perform the following operations:
==============================================

insertion

updation

deletion

retriving

coping the data

union operations ons ets

intersection on sets

difference or complement of sets

symmetric difference of sets

comparision of sets

sub sets

super sets

disjoints sets

how to find the length of the set:
==========================

to find the length of the set,we will use a function called "len()"
```

example:
=======

```
length of the set=len(set_name)
```

example:
=======

```
s1={1,2,3,4,5,6,7,8,9,10}
print(len(s1))#10
```

example:
=======

```
s1={1,2,3,4}
print(len(s1))#4
s2={1.2,3.4,5.6,7.8,9.0}
print(len(s2))#
s3={"abc","bca","cab","ghj"}
print(len(s3))#4
s4={1,2,3,4,1.2,3.4,5.6,7,8,"abc","bca"}
print(len(s4))#11
s5={}
print(len(s5))#
s6={1,2,3,1,2,3,1,2,3}
print(len(s6))#3
```

how to eleminate duplcates from the list:
================================

if we want to delete the duplicate elements from the list,we will convert the list into

"Set"

to convert the list into set,we will use a function called "set()"

syntax:
======

```
set_name=set(list_name)
```

example:
=======

```
l1=[1,2,3,4,1,2,3,4,1,2,3,4]
#eliminate duplicate elements from the list
#convert l1 list into set s1 using set() function
s1=set(l1)
print(s1)
```

how to eliminate duplicate elements from the tuple:
==

to eleminate duplicate elements from the tuple,we will use a function called "set()"

syntax:

```
======

set_name=set(tuple_name)

example:
=======

t1=(1,2,3,4,1,2,3,4,1,2,3,4)
#eliminate duplicate elements from the list
#convert l1 list into set s1 using set() function
s1=set(t1)
print(s1)
t1=tuple(s1)
print(t1)


how to display the set using for loop:
=============================

syntax:

for variable_name in set:

          #write the logic here

example:
=======

s1={1,2,3,4,5,6,7,8,9,10}
#using for loop
for i in s1:
    print(i)


how to insert the data into set:
========================

to insert the data in set,we will use  a function called "add()"

example:
=======

s1={1,2,3,4,5,6,7,8,9,10}
s1.add(200)
print(s1)
s1.add(300)
print(s1)
s1.add(500)
print(s1)
s1.add(600)
print(s1)

note:

in sets the data "will be stored in any form,that is reson set can be called as "

unorderd type"

list and tuple are "ordered type"
```

how to remove the data from the set:
=============================

to remove the data from the set,we will use a function called "discard()"

syntax:
======

```
set_name.discard()
```

example:
=======

```
s1={1,2,3,4,5,6,7,8,9,10}
s1.discard(6)
print(s1)
s1.discard(10)
print(s1)
s1.discard(123)
print(s1)
s1.discard(3)
print(s1)
```

we can also use "remove()" function,to remove the data from the set

syntax:
======

```
set_name.remove()
```

example:
=======

```
s1={1,2,3,4,5,6,7,8,9,10}
s1.remove(1)
print(s1)
s1.remove(10)
print(s1)
#s1.remove(123)
#print(s1)
```

note:
====

discard() ====>it is used to remove the element from the set,when we use discard() if the

element is not there,it never throws any error

remove()====>it is also used to remove the element from the set,when we use remove() if

the element is not there,it will throw the error

how to remove the all elements from the set:
===================================

```
to remove the all elements from the set,we will use "clear()"

syntax:
======

set_name.clear()

example:
=======

s1={1,2,3,4,5,6,7,8,9,10}
print("the length of the set s1:",len(s1))#10
s1.clear()
print("the length of the set s1:",len(s1))#0


how to delete the set with both data as well as  in python:
============================================

to remove the set,permenently,we will use a keyword called "del"

syntax:
======

del set_name

example:
=======

s1={1,2,3,4,5,6,7,8,9,10}
print("the length of the set s1:",len(s1))#10
s1.clear()
print("the length of the set s1:",len(s1))#0

how to update the set in python:
==========================

to update the set in python,we will use a function called "update()"

syntax:
=====

set_name.update()

example:
=======

s1={1,2,3,4,5,6,7,8,9,10}
s1.update({100,200,300,400})
print(s1)
s1.update([4,-5,9,0])
print(s1)
s1.update((11,122,33,44,55))
print(s1)
s1.update({1000})
print(s1)

note:
```

when we use update() function ,"the data has to be iterable(list/tuple/set)"

example:
=======

```
s1={1,2,3,4,5,6,7,8,9,10}
s1.update({100,200,300,400})
print(s1)
s1.update([4,-5,9,0])
print(s1)
s1.update((11,122,33,44,55))
print(s1)
s1.update({1000})
print(s1)
s1.update((103,))
print(s1)
```

how to copy the data from the set:
===========================

to copy the data from the set,we will use a function called "copy()"

syntax:

```
set_name=set_name.copy()
```

example:
=======

```
s1={1,2,3,4,5,6,7,8,9,10}
s2=s1.copy()
print(s2)
```

example:
=======

```
s1={1,2,3,4,5,6,7,8,9,10}
s2={100,200,300,400}
s3={1}
s3.update(s1)
s3.update(s2)
print(s3)
```

union on sets:
===========

union means "combine the two sets into single set"

using union() function also,we can perform "union" operation on sets

example:
=======

```
s1={1,2,3,4,5,6,7,8,9,10}
s2={100,200,300,400}
s5={11,22,33,44}
#we perform union using "|" operator
```

```
s3=s1|s2
print(s3)
s4=s1|s2|s5
print(s4)
#we can also perform union using "union()" function
s6=s1.union(s2)
print(s6)
```

intersection on sets:
================

intersection means "getting or finding the common elements from the sets"

to find the intersection,we will use the following ways:
==

1.using "&" operator

2.using intersection() function

3.using intersection_update() function:
===============================

example:
=======

```
s1={1,2,3,4,5}
s2={1,2,3,8}
#we are finding the intersection using "&"
result=s1&s2
print(result)#{1,2,3}
#we are finding the intersection using "intersection()"
result=s1.intersection(s2)
print(result)#{1,2,3}
#finding intersection using intersection_update()
s1.intersection_update(s2)
print(s1)
```

difference on sets:
===============

diference means "the unique elements from one set,which are not in another

set"

in python,we can find the difference of two sets as follows:
===

1.using "-" operator

2.using "difference()" function

===

example:
=======

```
s1={1,2,3,4,5}
```

```
s2={1,2,3,8,8,9,10}
#we are finding the difference using "-" operator
result=s1-s2
print(result)#{4,5}
result=s2-s1
print(result)#{8,9,10}
#we are performing the difference using "difference()"
result=s1.difference(s2)
print(result)
result=s2.difference(s1)
print(result)
#using difference_update
s1.difference_update(s2)
print(s1)#{4,5}
s2.difference_update(s1)
print(s2)#{1,2,3,8,9,10}
```

symmetric difference on sets in python:
================================

symmetric difference means "unique elements from the both sets"

to perfrom the symmetric difference ,we will use the following ways:
==

1.using ^

2.using symmetric_difference() function

3.using symmetric_difference_update() function
===

example:
=======

```
s1={1,2,3,4,5}
s2={1,2,3,8,8,9,10}
#we are finding the symmetric difference using "-" operator
result=s1^s2
print(result)#{4,5,8,9,10}
#we finding the symmetric difference using symmetric_difference()
result=s1.symmetric_difference(s2)
print(result)
#we are finding symmetric difference using symmetric_difference_update()
s1.symmetric_difference_update(s2)
print(s1)
```

disjoint sets:
==========

disjoints sets means "the sets intersection is empty or the sets does not have

any common element"

example:
=======

```
s1={1,2,3,4,5}
```

```
s2={1,2,3,8,8,9,10}
s3={100,200,300}
result=len(s1&s3)#0
if result==0:
    print("both sets are disjoint")
else:
    print("both sets are not disjoint")
```

how to take elements into set, dynamically from the user:
==

example:
=======

```
s1=set({})
length=int(input("enter how many elements to insert:"))
while length!=0:
    s1.add(int(input(("enter the element:"))))
    length=length-1
print(s1)
```

comparision on sets:
=================

sets are equal or not:
=================

```
s1={1,2,3,4,5,6,7}
s2={1,2,3}
s3={1,2,3}
if s1==s2:
    print("s1 and s2 are equal sets")
else:
    print("s1 and s2 are not equal sets")
if s2==s3:
    print("s2 and s3 are equal sets")
else:
    print("s2 and s3 are not equal sets")
```

check given set is subset or not:
==========================

example:
========

```
s1={1,2,3,4,5,6,7}
s2={1,2,3}
if s2<s1:
    print("the given set s2 is subset")
else:
    print("the given set s2 is not subset")
if s1<s2:
    print("the given set s1 is subset")
else:
    print("the given set s1 is not subset")
#we can also check given set is subset or not
print(s1.issubset(s2))
print(s2.issubset(s1))
```

how to check given set is superset ot not:
===========================

example:
=======

```
s1={1,2,3,4,5,6,7}
s2={1,2,3}
if s2>s1:
    print("the given set s2 is superset")
else:
    print("the given set s2 is not superset")
if s1>s2:
    print("the given set s1 is superset")
else:
    print("the given set s1 is not superset")
#we can also check given set is superset or not
print(s1.issuperset(s2))
print(s2.issuperset(s1))
```

how to check two sets are disjoint or not:
=================================

```
s1={1,2,3,4,5,6,7}
s2={1,2,3}
s3={100,200,300}
print(s1.isdisjoint(s2))
print(s1.isdisjoint(s3))
```

frozen sets:
=========

frozen set is a "set and which does not allow any changes once it is created"

to create the frozen set in python,we will use a function called "frozenset()"

example:
=======

```
s1={1,2,3,4,5,6,7}
print(type(s1))
s1.add(100)
s1.add(200)
print(s1)
s1=frozenset(s1)
print(type(s1))
```

note:
====

once we create the frozen set,it can not change(we can not insert/update/delete the set)

list vs tuple vs set in python:
=======================

list can store "any number of elements and any type of eleements" tuple can store "any number of elements and any type of eleements"

set can store "any number of elements and any type of elements"

list and tuple can store "duplicate elements"

set can not store "duplicate elements"

list and tuple allows "indexing and slicing"

set not allows "indexing and slicing"

in list and tuple "elements order preserved"

in set "the elements order is not preserved"

both list and set can allow "insertion/updation/deletion"

both tuple and frozenset can not allow "insertion/updation/deletion",once

created

list,tuple and set are can be empty

empty set in python,is termed as "dictionary" by default

list can converted or interchanged as tuple using tuple() or set using set()

tuple can converted or interchanged as set using set() or list using list()

set can converted or interchanged as list using list() and tuple using tuple()

Strings in python:
==============

strings means "sequence of chracters or collection of characters" in python

in python,we have three type of characters litreals or string litreals:
==

1.single character litreal

2.string litreal

3.multi-line string or doc string

single character or string can be stored in python,with help of

single or double quotes

multi-line string can be stored with help of "triple quotes"

example:
=======

```
s1='a'
print(s1)
print(type(s1))
s2="abc"
```

```
print(s2)
print(type(s2))
s3="""i am from usa"""
print(s3)
print(type(s3))
```

how to find the length of the string:
============================

to find the length of the string in python,we will use "len()" function

syntax:
======

```
len(string_name)
```

example:
=======

```
s1='a'
print(len(s1))#1
s2="abc bca"
print(len(s2))#7
s3="""i am from usa"""
print(len(s3))#13
```

strings with indexing:
=================

in strings also we can "apply indexing as same as list or tuple"

on strings also,we can have both "positive indexing(0 to n-1) and negative

indexing(-1 to -n),where n is length of the string"

example:
=======

```
s1="hello world"
print(s1[0])#h
print(s1[6])#w
print(s1[9])#l
print(s1[10])#d
print(s1[-1])#d
print(s1[-11])#h
print(s1[-8])#l
```

strings with slicing:
===============

example:
=======

```
s1="hello world"
print(s1[0:])#hello world
print(s1[:6])#hello
print(s1[4:10])#o worl
print(s1[2:-1])#llo worl
print(s1[9:1])#no data

how to compare two strings in python:
==============================

example:
=======

s1="hello world"
s2="hello world"
if s1==s2:
    print("the given two strings are same")
else:
    print("the given two strings are not same")

how to compare two strings without using "==":
=====================================

s1="hello world"
s2="hello world"
print(id(s1))
print(id(s2))
if s1 is s2:
    print("the given two strings are same")
else:
    print("the given two strings are not same")


how to print the only vowels from the given string:
====================================

example:
=======

s1="hello world"
for i in s1:
    if i in {'a','e','i','o','u'}:
        print(i,end="")


how to print only consodents from the given string:
========================================

example:
=======

s1="hello world"
for i in s1:
    if i not in {'a','e','i','o','u'}:
        print(i,end="")


how to convert the string into list:
===========================

example:
=======
s1="hello world"
l1=list(s1)
print(l1)
```

how to convert the string into tuple:
=============================

example:
=======

```
s1="hello world"
t1=tuple(s1)
print(t1)
```

how to convert the list into string:
===========================

example:
=======

```
s1=['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
s2=""
for i in s1:
    s2+=i
print(s2)
```

how to remove the duplciates from string:
=================================

example:
=======

```
s1="hello world"
s1=set(s1)
print(s1)
```

how to add the character or string into the given string in python:
==

example:
=======

```
str1=input("enter the string:")
print("the given string:",str1)
ch=input("enter the character:")
print(ch)
#insert at end
print(str1+ch)
#insert at begin
print(ch+str1)
#insert at any position
position=int(input("enter the position:"))
if position<len(str1):
    print(str1[0:position]+ch+str1[position:])
else:
    print("the given position is more than the given string length")
```

```
how to update the given character in the given string in python:
===================================================

str1=input("enter the string:")
print("the given string:",str1)
ch=input("enter the character:")
print(ch)
#update the first character in thge string
str1=ch+str1[1:]
print(str1)
#update the last character of the string
str1=str1[:-1]+ch
print(str1)
#update the any character the string at any position
position=int(input("enter the position:"))
if position<len(str1):
    str1=str1[:position]+ch+str1[position+1:]
    print(str1)

how to delete the given character in the given string in python:
==================================================
example:
=======
str1=input("enter the string:")
print("the given string:",str1)
ch=input("enter the character:")
print(ch)
index=0
position=0
for i in str1:
    if i==ch:
        position=index
    index=index+1
if position==0:
    print("the given character not found")
else:
    str1=str1[:position]+str1[position+1:]
    print(str1)

String functions on python:
======================

upper():
======

this function will be used to convert the given into "uppercase"

syntax :
======

string_name.upper()

example:
=======
s1="hello"
result=s1.upper()
print(result)
```

```
lower():
======

this function will used "To convert the given string into lower case"

syntax:
======

string_name.lower()

example:
=======

s1="HELLO"
result=s1.lower()
print(result)


casefold():
========

this function is also used to convert the "given string characters into lower case

only"

syntax:
======

string_name.casefold()

example:
========

s1="HELLO"
result=s1.casefold()
print(result)


count():
======

this function is used "count the given charcater/string is how many times is

repeated in the string"

syntax:
======

string_name.count()

example:
=======

s1="hello world"
print(s1.count("l"))#3
print(s1.count("o"))#2
print(s1.count("hello"))#1
```

index():
======

index() is a function and it used to "find the given character index ot

given string index in the string"

syntax:
======

string_name.index()

example:
=======

```
s1="hello world"
print(s1.index("hello"))
print(s1.index("o"))
#print(s1.index("Z"))
```

capitalize():
=========

capitalize() means "it will makes the first character of the string as capital

or uppercase letter"

syntax:
======

string_name.capitalize()

example:
========

```
s1="hello world"
print(s1.capitalize())
```

startswith():
=========

this function will be used to "to check given string is start with particular
string

ot not"

syntax:
======

string_name.startswith("string to search")

example:
========

```
s1="hello world"
result=s1.startswith("hello")
print(result)
result=s1.startswith("world")
```

```
print(result)
result=s1.startswith("hello",0,3)
print(result)
result=s1.startswith("hello",0,6)
print(result)
result=s1.startswith("hello",6,10)
print(result)
```

endswith():
=========

this function will be used to "to check given string is end with particular string

ot not"

syntax:
======

```
string_name.endswith("string to search")
```

example:
========

```
s1="hello world"
result=s1.endswith("hello")
print(result)
result=s1.endswith("world")
print(result)
result=s1.endswith("hello",0,3)
print(result)
result=s1.endswith("hello",0,6)
print(result)
result=s1.endswith("hello",6,10)
print(result)
```

find():
=====

this string is used to check the given string is there or not in the string

if given string is present in the string,it will return the "index of first character

of the string"

if the given string is not in the string,it will return the "-1"

syntax:
======

```
string_name.find("string_name")
```

example:
=======

```
s1="hello world"
result=s1.find("hello")
```

```
print(result)#0
result=s1.find("world")
print(result)
result=s1.find("Z")
print(result)
result=s1.find("he",6,10)
print(result)
result=s1.find("he",0,5)
print(result)#0
```

swapcase():
==========

this function will convert the given "uppercase letters to lowercase or lowercase

letters to uppercase"

syntax:
======

string_name.swapcase()

example:
=======

```
s1="heLLo WOrld"
result=s1.swapcase()
print(result)
```

isdigit():
======

it will check given charcater is "digit or not"

syntax:
======

string_name.isdigit()

example:
=======

```
s1="1"
print(s1.isdigit())
s1="9"
print(s1.isdigit())
s1="a"
print(s1.isdigit())
```

isalpha():
=======

it is used to check given characters in the string are alphabets or not

syntax:
======

```
string_name.isalpha()

example:
=======

s1="123"
print(s1.isalpha())
s1="ab9"
print(s1.isalpha())
s1="abc"
print(s1.isalpha())


isalnum():
========

it is used to check the given string "contains either alphabets or digits"

syntax:
======

string_name.isalnum()

example:
=======

s1="abc123"
print(s1.isalnum())
s1="abc"
print(s1.isalnum())
s1="123"
print(s1.isalnum())
s1="abc#12"
print(s1.isalnum())

isprintable():
==========

it will check the given string characters are printable or not

printable characters means ===>A to Z,a to z,,all special charcaters

syntax:
======

string_name.isprintable()

example:
=======

s1="abc+-*"
print(s1.isprintable())
s1="abc#%&^}{:"
print(s1.isprintable())
s1="\n"
print(s1.isprintable())


isascii():
```

=======

this function will check "the all characters in the string are ascii characters

or not"

syntax:
======

string_name.isascii()

example:
========

```
s1="abc+-*"
print(s1.isascii())
s1="abc#%&^}{:"
print(s1.isascii())
s1="\n"
print(s1.isascii())
```

isnumeric():
=========

it will check all characters in the string are "numeric or not"

syntax:
======

string_name.isnumeric()

example:
=======

```
s1="1234"
print(s1.isnumeric())
s1="0.5"
print(s1.isnumeric())
s1="1/2"
print(s1.isnumeric())
s1="1234a"
print(s1.isnumeric())
```

isspace():
========

it checks "all characters in the string are space or not"

syntax:
======
string_name.isspace()

example:
========

```
s1="   12"
print(s1.isspace())
```

```
s1="                 "
print(s1.isspace())

islower():
=======

it will check "all the characters of the string in lowercase or not"

syntax:
======

string_name.islower()

example:
=======

s1="abc"
print(s1.islower())
s1="abc123"
print(s1.islower())
s1="ABC123abc"
print(s1.islower())
s1="1234"
print(s1.islower())

isupper():
=======

it will check "all the characters of the string in uppercase or not"

syntax:
======

string_name.isupper()

example:
=======

s1="abc"
print(s1.isupper())
s1="abc123"
print(s1.isupper())
s1="ABC123abc"
print(s1.isupper())
s1="1234"
print(s1.isupper())
s1="ABC"
print(s1.isupper())

title():
=====

this function will make the "every first character of the word string will be in

upper case"


syntax:
======
string_name.title()

example:
========

s1="i am from india"
s1=s1.title()
print(s1)#I Am From India

istitle():
======

to check the given string is in the form of title or not

syntax:
======

string_name.istitle()


example:
=======

s1="i am from india"
s1=s1.istitle()
print(s1)
s1="I Am From India"
print(s1.istitle())


write a program to find the following for given string:
===========================================

no of characters in the string

no of words in the string

no of vowls in the string

no of consodents in the string

code:
====

s1="i am from india"
print("no of characters in the string is:",len(s1))
space=0
vowel=0
consodent=0
for i in s1:
    if i=='a' or i=='A' or i=='e' or i=='E' or i=='i' or i=='I' or i=='o' or i=='O'
or i=='u' or i=='U':
        vowel+=1
    elif i==" ":
        space=space+1
    else:
        consodent+=1
```

```
print("the number words in the string is:",space+1)
print("the number of vowels in the string are:",vowel)
print("the number of consodents are in the string are:",consodent)
```

write a python program to print the duplicate characters of the string:
==

input: hello world

output: l,o

input: aabcdeabcfg

output:a,b,c

program:
=======

```
s1="aabcdeabcfg"
s2=set(s1)
print(s2)
count=0
for i in s2:
    for j in s1:
        if j==i:
            count=count+1
    if count>1:
        print(i)
    count=0
```

isidentifier():
==========

this function will be used to check given "String" is identifier or not

syntax:
======

```
string_name.isidentifier()
```

example:
=======

```
s1="hello"
print(s1.isidentifier())#True
s1="123hello"
print(s1.isidentifier())#False
s1="#if"
print(s1.isidentifier())#False
s1="_if"
print(s1.isidentifier())#True
s1="while$"
print(s1.isidentifier())#
```

isdecimal():
=========

it will check all charcaters of the string are decimals or not

```
all characters of the string must be in the range of 0,1,2,3,...9

syntax:
======

string_name.isdecimal()

example:
=======

s1="1234"
print(s1.isdecimal())
s1="12390123"
print(s1.isdecimal())
s1="abf"
print(s1.isdecimal())


format():
=======

this function will be used to format the string

syntax:
======

string_name.format()


example:
=======

location=input("enter the location:")
s1="i am from {}"
print(s1.format(location))
pincode=int(input("enter the pincode is:"))
s2="i  am  from  {0} and my pincode is {1}"
print(s2.format(location,pincode))

example-2:
=========

lang=input("enter the language:")
type=input("enter the type:")
str1=" in {lang},{type} is store group of data"
print(str1.format(lang=lang,type=type))

example-3:
=========

s1="the binary number for {0} is: {0:b}"
number=int(input("enter the number:"))
print(s1.format(number))
s1="the octal number for {0} is: {0:o}"
print(s1.format(number))
s1="the hexadecimal number for {0} is: {0:x}"
print(s1.format(number))

:b===> binary
```

```
:o===>octal

:x===>hexadecimal

example:
=======

s1="the given {0} in percentage is {0:%}"
print(s1.format(12.4))

split():
=====

it is used to divide the given string into "list of strings"

syntax:
======

string_name.split(" delimeter")

example:
=======

s1="i am from india"
print(s1.split(" "))
print(s1.split("m"))#[i a, fro, india]
print(s1.split("i"))#[ am from ,nd,a]
s2="hello world"
print(s2.split(" "))#['hello','world']
print(s2.split("l"))#[he,'',o wor',d]
print(s2.split("o"))#['hell', 'w','rld']


example:
=======

a,b,c=input("enter the numbers:").split(" ")
print(a,b,c)
a,b,c=input("enter the numbers:").split(",")
print(a,b,c)

replace():
========

it is used to replace given string in another string in a string

syntax:
======

string_name.replace(source_String,replace_string)

example:
=======

str1="hello world hello world"
str1=str1.replace("hello","hi")
print(str1)
str2="hello hi hello hi"
```

```
str2=str2.replace("h","H",2)
print(str2)
str2=str2.replace("l","L",3)
print(str2)
str2=str2.replace("i","I",0)
print(str2)
str2=str2.replace("i","I",3)
print(str2)
```

dictionary in python:
=================

dictionary is used to "Store the data in the form of key and value pairs"

in dictionary,the data can be indexed via "keys"

in dictionary,the data/value can be duplicate

in dictionary,the key always unique

in dictionary,the data/value can be any type

in dictionary,the key can be any type

in python,dictionary can be empty

to create the dictionary,in python we will use " { }"

syntax for dictionary:
=================

```
dictionary_name={key1_name:value_1,key2_name:value2,................}
```

here befor ":",we will write the keys

here after ":",we will write the values

note:
====

empty set in python,is also known as "dictionary"

example:
=======

```
d1={1:2,3:4,5:6,7:8}#here keys:1,3,5,7 values:2,4,6,8
print(d1)
d2={1:1.2,3:4.5,6:7.8,9:12.3}#here keys:1,3,6,9 values:1.2,4.5,7.8,12.3
print(d2)
d3={"name":"Ram","location":"malkajgiri","pincode":500047}
print(d3)
d4={}
print(d4)
d5={1:2,1:2,1:3,1:4,1:5,1:10}
print(d5)
```

example:
=======

```
d1={}
print(d1)
d1[1]=2
d1[3]=4
d1[5]=6
d1[7]=8
d1[9]=11
d1[10]=100
print(d1)
d1[100]=2345
print(d1)
```

how to update the key value in the dictionary:
=====================================

example:
=======

```
d1={"name":"kiran","location":"hyderabad","salary":10000}
print(d1)
#here we are update the keys
d1["name"]="suraj"
d1["location"]="delhi"
d1["salary"]=10000
print(d1)
#here we are add the keys
d1["email"]="suraj@gmail.com"
d1["role"]="developer"
print(d1)
```

how to find the length of the dictionary:
================================

to find the length of the dictionary,in python we will use "len()" function

syntax:
======

```
length=len(dictonary_name)
```

here length of the dictionary= number of key and values pairs

one key and value in a dictionary can be termed as "item"

item===>key and value

example:
=======

```
d1={"name":"kiran","location":"hyderabad","salary":10000}
print(len(d1))#3
#here we are add the keys
d1["email"]="suraj@gmail.com"
d1["role"]="developer"
print(len(d1))#5
```

how to display the only keys of the dictionary:
=====================================

to know the dictionary keys in python,we will use a function called "keys()"

syntax:
======

```
keys=dictionary_name.keys()
```

example:
=======

```
d1={"name":"kiran","location":"hyderabad","salary":10000}
keys=d1.keys()
print(keys)
```

how to know the values of the dictionary in python:
===

to know the values of the dictionary in python,we will use "values()" function

syntax:
======

```
values=d1.values()
```

example:
=======

```
d1={"name":"kiran","location":"hyderabad","salary":10000}
keys=d1.values()
print(keys)
```

how to know the items in the dictionary in python:
===

to know the both key and value pairs in dictionary,in python we will use

items()" function

syntax:
======

```
items=dictionary_name.items()
```

example:
=======

```
d1={"name":"kiran","location":"hyderabad","salary":10000}
items=d1.items()
print(items)
```

how to display the dictionary using for loop:
====================================

```
example:
=======

d1={"name":"kiran","location":"hyderabad","salary":10000}
for i in d1:
    print(i,":",d1[i])

how to check given value associted key in the dictionary:
=============================================

code:
====

d1={"name":"kiran","location":"hyderabad","salary":10000}
value=input("enter the value:")
for i in d1:
    if str(d1[i])==value:
        print(i)


how to add the key and value to dictionary dynamically in python:
====================================================

code:
=====

for one key and pair:
=================

d1={}
key=input("enter the key:")
value=input("enter the value:")
d1[key]=value
print(d1)

for n kay and value pairs:
====================

d1={}
count=int(input("enter the how many key and value need to insert:"))
while count!=0:
    key=input("enter the key:")
    value=input("enter the value:")
    d1[key]=value
    count=count-1
print(d1)


how to remove the dictionary key and value pair in python:
==============================================

in python,we can remove the key and value pair in the following ways:
======================================================

1.using pop():
===========

when we use pop() function on dictionary,it will remove the based on the given
```

key in the pop() function

syntax:
======

```
dictionary_name.pop("key_name");
```

example:
=======

```
d1={"name":"kiran","location":"hyderbad","salary":20000}
print(d1)
d1.pop("salary")
print(d1)
#d1.pop("salary")
```

note:
====

when we use pop() function,if we give any key as wrong,it will generate the an error

called "KeyError"

2.using popitem():
==============

popitem() is used to remove the "the key and value pair from the dictionary at the end"

or

it will remove the "latest new key which is added into dictionary"

syntax:
======

```
dictionary_name.popitem()
```

example:
=======

```
d1={"name":"kiran","location":"hyderbad","salary":20000}
print(d1)
d1.popitem()
print(d1)
d1.popitem()
print(d1)
```

3.using del:
=========

using "del",we can remove the item from the dictionary based on the given "key" value

syntax:

======

```
del dictionary_name[key];
```

example:
=======

```
d1={"name":"kiran","location":"hyderbad","salary":20000}
del d1["name"]
print(d1)
del d1["salary"]
print(d1)
del d1["location"]
print(d1)
```

4.using clear():
============

this function will be used to remove "all key and value pairs or items of the dictionary"

syntax:
======

```
dictionary_name.clear()
```

example:
=======

```
d1={"name":"kiran","location":"hyderbad","salary":20000}
d1.clear()
print(d1)
```

how to update the key and value pairs in the dictionary:
==

to change the key and value pair in the dictionary,we will use the following ways:
===

1.using key name

2.update() function

example:
=======

```
d1={"name":"kiran","location":"hyderbad","salary":20000}
#update the key and value pairs using "key name"
d1["salary"]=45000
d1["name"]="bhuvan"
d1["location"]="delhi"
print(d1)
d1.update({"name":"karan"})
print(d1)
#using update(),we can add the new key and value pair also
d1.update({"Email":"karan@gmail.com"})
print(d1)
```

how to copy the dictionary in the python:
================================

to copy the dictionary in python,we will use the following ways:
===

1.using "=" operator

2.using copy() function

3.using dict() function

example:
=======
```
d1={"name":"kiran","location":"hyderbad","salary":20000}

#copy the dictionary using "=" operator
d2=d1
print(d2)
d2["pincode"]=500047
print(d1)
del d1["salary"]
print(d2)


#using copy() function
d3=d1.copy()
print(d3)
d3["salary"]=50000
print(d1)
print(d3)

d1={"name":"kiran","location":"hyderbad","salary":20000}
#using dict() function
d2=dict(d1)
print(d2)
d1["email"]="kiran@gmail.com"
print(d2)
print(d1)
```

list comprehension:
===============

list comprehension means "creating the list,based on the another list/iterable or using

range()"

syntax:
======

```
list_name=[variable_name for variable in list_name/iterable_name condition]
```

example:
=======

```
l1=[1,2,3,4,5,6,7,8,9,10]
l2=[i for i in l1 if i%2==0]
print(l2)
l3=[i for i in l1 if i%2!=0]
print(l3)
l4=[i for i in l1 if i>5]
print(l4)
l5=[i for i in l1 if i<5]
print(l5)
```

tuple comprehension:
=================

tuple comprehension means "creating the tuple,based on the another tuple/iterable or using

range()"

syntax:
======

```
tuple_name=[variable_name for variable in tuple_name/iterable_name condition]
```

example:
=======

```
t1=(1,2,3,4,5,6,7,8,9,10)
print(t1)
t2=tuple(list((i for i in t1 if i%2==0)))
t3=tuple(list((i for i in t1 if i%2!=0)))
t4=tuple(list((i for i in t1 if i>5)))
t5=tuple(list((i for i in t1 if i<5)))
print(t2)
print(t3)
print(t4)
print(t5)
```

tuple comprehension also called as "generator comprehension"

set comprehension:
================

set comprehension means "creating the set,based on the another set/iterable or using

range()"

syntax:
======

```
set_name={variable_name for variable in set_name/iterable_name condition}
```

example:
=======

```
l1=[10,20,30,40,50,60,70,80,90,100]
s1={i for i in l1 if i%2==0}
print(s1)
```

```
s2={i for i in l1 if i%2!=0}
print(s2)
s3={i for i in l1 if i>50}
print(s3)
s4={i for i in l1 if i<50}
print(s4)
```

dictionary comprehension:
=====================

dictionary comprehension means "creating the dictionary,based on the another .list/tuple/set or using

range()"

syntax:
======

```
dictionary_name={key_name:value_name for variable in set_name/iterable_name condition}
```

example:
========

```
l1=[10,20,30,40,50,60,70,80,90,100]
d1={i:i+4 for i in l1 if i%2==0}
print(d1)
d2={i:i-4 for i in l1 if i%3==0}
print(d2)
d3={i:i-4 for i in l1 if i>50}
print(d3)
d4={i:i-4 for i in l1 if i<50}
print(d4)
```

example:
=======

```
l1=[i for i in range(1,10,1)]
print(l1)
l2=[i for i in range(10,1,-1)]
print(l2)
l3=[i for i in range(1,10,-2)]
print(l3)#[]
l4=[i for i in range(1,10,4)]
print(l4)
```

example:
========

print the numbers from 1 to 10 without using range() and loops

```
data=1
def display():
    global data
    print(data)
    data=data+1
    if data<=10:
        display()
display()

iterators in python:
===============

iterators are used "to get the data from the any iterable(list/tuple/set/string/..)

one by one item"

to work with iterators,in python we will use the following functions:
=====================================================

iter()===>it is used to create the  "iterator" for given iterable

next()===>it is used to traverse the data one by one from the given iterable

example-1:
=========

l1=[10,20,30,40,50,60,70,80,90,100]#here l1 is list
#create the iterator using iter()
i1=iter(l1)#here this function will data of l1
#retrive the data from given list via iterator
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))

example-2:(iterator with for loop):
===========================

l1=[10,20,30,40,50,60,70,80,90,100]#here l1 is list
#create the iterator using iter()
i1=iter(l1)#here this function will data of l1
#retrive the data from given list via iterator
for i in i1:
    print(i)

example-3:
=========

t1=(10,20,30,40,50,60,70,80,90,100)#here t1 is tuple
#create the iterator using iter()
i1=iter(t1)#here this function will data of l1
#retrive the data from given list via iterator
for i in i1:
    print(i)

example-4:
=========

s1="hello world"
#create the iterator
i1=iter(s1)
#retrive the data from iterator
print(next(i1))#h
print(next(i1))#e
print(next(i1))#l
print(next(i1))#l
#using for loop
for i in i1:
    print(i)

example-5:
=========

s1="hello world"
#create the iterator
i1=iter(s1)
print(type(i1))#<class 'str_ascii_iterator'>

example-6:
=========

s1=123
i1=iter(s1)#here s1 is not iterable
print(next(i1))#error

example-7:
=========

s1=range(1,10)
i1=iter(s1)
print(next(i1))
print(next(i1))
for i in i1:
    print(i)#3 4 5 6 7 8 9

example-9:
=========

s1=range(1,10)
print(type(s1))#range


generators in python:
=================

generator is a "custom or user-defined iterators"

to create the generators,in python we will use  a keyword called "yield"

example:
=======

#here i create the data as a function
def display():
    yield 1
    yield 2
    yield 3
    yield 4
    yield 5
#create the iterator
i1=display()
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
```

example-2:
=========

```
#here i create the data as a function
def display():
    yield 1
    yield 2
    yield 3
    yield 4
    yield 5
#create the iterator
i1=display()#<class 'generator'>
print(type(i1))
for i in i1:
    print(i)
```

example-3:
=========

```
#here i create the data as a function
def display():
    yield 1
    yield 2
    yield 3
    yield 4
    yield 5
for i in display():
    print(i)
```

example-4:
=========

```
#here i create the data as a function
def display():
    yield 1
    yield "hi"
    yield 3
    yield "hello"
    yield 5
for i in display():
    print(i)
```

what is "StopIteration" state in Python:
===============================

when we try to print the any data from iterator or generator,more than it's size

or length,python will show the error called "StopIteration"

example:
=======

```
#here i create the data as a function
def display():
    yield 1
    yield "hi"
    yield 3
    yield "hello"
    yield 5
i1=display()
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
```

generator as expression:
====================

 to create the generator as expression,we will use the concept called

"iterable comprehension"

example:
=======

```
t1=(i for i in range(1,10) if i%2==0)
i1=iter(t1)
print(next(i1))#2
print(next(i1))#4
print(next(i1))#6
print(next(i1))#8
```

example:
========

```
t1=(i for i in range(1,10) if i%2==0)
for i in t1:
    print(i)
```

example:
=======

```
t1=tuple((i for i in range(1,10) if i%2==0))
print(t1)
```

copy the data using "=" operator:
==========================

```
l1=[1,2,3,4,5,6,7]
#copy the data using "=" operator
l2=l1
l1.append(100)
l1.append(200)
print(l2)
l2.append(400)
l2.append(500)
print(l1)
l1.clear()
print(l2)
#l1 and l2 are interdependent
#if we change the l1,there is change  in l2 as l1
```

shallow copy and deep copy:
=======================

to perform the shallow copy and deep copy ,in python we will use a module

called "copy" module

shallow copy:
===========

example:
=======

```
import copy
#take the l1
l1=[1,2,3,4,5,6,7,8,9]
#on l2 we are applying the shallow copy
l2=copy.copy(l1)
print(l2)
l2.append(100)
l2.append(200)
print(l2)
l3=[[1,2,3]]
l4=copy.copy(l3)
l4[0].append(100)
print(l3)
```

note:
====

in the case of shallow copy,

when we copy the any original object without any nested objects,the original

object and copied object independent each other

when we copy the any original object with any nexted objects,only the

nested objects of both original and copied objects depend with each other

deep copy:
=========

example:
========

```
import copy
#take the l1
l1=[1,2,3,4,5,6,7,8,9]
#on l2 we are applying the shallow copy
l2=copy.deepcopy(l1)
print(l2)
l2.append(100)
l2.append(200)
print(l2)
print(l1)
```

```
l3=[[1,2,3]]
l4=copy.deepcopy(l3)
print(l4)
l3.append(100)
print(l4)
```

note:
====

note:
====

in the case of deep copy,

when we copy the any original object without any nested objects,the original

object and copied object independent each other

when we copy the any original object with any nested objects,even

nested objects of both original and copied objects also independent with each

other

OOPS in python:
=============

OOPS stands for "Object oriend programming System"

we can say python is a "object oriented programming language"

object-oriented is a "programming padigm of the python language"

programming paradigm will tells "how the programming will be in the language"

in general we will have the following programming paradigms:
===

1.procedure-oriented programming paradigm

2.function-oriented programming paradigm

3.structure oriented programming paradigm

4.object based programming paradigm

5.object oriented programming paradigm

1.procedure-oriented programming paradigm:
=====================================

in this model,

the program is divided into "n" number of prcedures

in this model,problem will decomposed into "procedures"

here procedure is "building blocks of this programming paradigm" procedure will consists "a group or collection of statements",these statements

will perform a spesified task

example:
=======

frotron(formula transition)

2.function-oriented programming paradigm:
===================================

in this model,

the program is divided into "n" number of functions

in this model,problem will decomposed into "functions"

here function is "building blocks of this programming paradigm"

functions will consists "a group or collection of statements",these statements

will perform a spesified task

procedure will never return any value

function will return a value

example:
=======

c,c++,java,python,javascript...........

3.structure oriented programming paradigm:
====================================

in this model,we will follow the rules to write any program

in this,any program whatever we write,it will have a rigid/fixed structure

when we need to write program,we need to follow the it's defined of the

language

example:
=======

c,java,c#,c++.............

4.object based programming paradigm:
===============================

in this model,

the data will shared in the form of obejcts in model,the programming will done in the form objects

here object will have "data as well as methods(functions)"

here we can able communicate object to object

example:
=======

javascript,

in this model,we do not have "inheritance",but we have inheritance in object

oriented

5.object oriented programming paradigm:
=================================

in this model,

the complete program is divided into "n" number of classes and it's objects

in this model,we will have the following several concepts like:

1.classes

2.objects

3.inheritance

4.polymorphism

5.data abstraction

6.data encapsulation

7.message passing

8.dynamic method dispatch model,.........................

example:
========

c,c++,java,python,php,...................

"small talk" is the first object oriented programming langauge

python follows:
=============

function oriented

object oriented

object based

classes and objects:

================

class:
====

class is collection of "data and method",where data or methods are called as

members of the class

or

class will represent the "attributes and behaviour of the object"

class is a logical entity

class is a "collection of objects or group of objects",but each will share the

same data or method

object:

object is an entity and which is exists physically in the real world

or

object is a instance of a class

or

object is a class type variable

example:
========

class
====

furniture <====== chair,sofa,table,........(objects)

fruit<==========apple,banana,............(objects)

vehicle<=======lorry,bus,bike,............(objects)

polygon<====== triangle,rectangle,square,.............(objects)

how to create the class in python:
===========================

to create the class in python,we will use "class" keyword

syntax:
======

```
class classs_name:

        #data

        #methods
```

```
in python class,

we can create the following:
======================

1.data ====>variable,list,tuple,string,set,dictionary

2.methods ===>functions


the both data and methods are called as "members of the class"

we can not access the any data or method of the class,directly

if we want to access the any member of the class,we will create the

"object" to the class

with help of object,we can access the any data or method in the class,

to create the object to class in python,we will use the following syntax:
=========================================================

object_name=class_name()

to acess the any member of the class with object,we will use the following syntax:
===========================================================

object_name.member_name

example:
=======

#create the class
class A:
    #data
    a=10
    l1=[10,20,30]
    t1=(100,200)
    s1={1,2,3,4}
    d1={1:2,3:4,5:6}
#create the object for class "A"
a1=A()
print(a1.a)
print(a1.l1)
print(a1.t1)
print(a1.s1)
print(a1.d1)


example-2:
=========

#create the class
class A:
   #method
   def display1(self):
        print("this display1 method")
    def display2(self):
        print("this is display2 method")
    def display3(self):
        print("this is display3 method")

#create the object for class "A"
a1=A()
a1.display1()
a1.display2()
a1.display3()
```

note:
====

methods are always created only in the class

methods are always called via object only

method are not created outside the class like functions

methods are always associated with objects and classes

methods can return result same like functions

methods can take arguments same like functions

we can create class only with "Data" in python

we can create class only with "methods" in python

we can create the class with "data and methods" in python

we can create the class with out "data and methods" in python (empty class)

if class does not have any data or method,the class can be called as

"empty class"

to create the empty class in python,we will use "pass" keyword

example:
========

```
#empty class
class A:
    pass
#object creation for A class
a1=A()
```

example:
=======

```
class A:
    #data
    a=10
    b=20
    c=30
#object creation for A class
a1=A()
```

```
print(type(a1))
#here we create the object using another object of same class
a2=a1
print(type(a2))
print(a2.a)
print(a2.b)
```

object cloning:
============

in python,we will create the object with help of the another object of same

class,then this process is known as "object cloning"

example:
=======

```
a1=A()

a2=A()

a3=a1

a4=a2
```

here a1,a2 are objects which are created with help of class

here a3,a4 are objects which are created with help of a1 and a2 objects

here a3 a4 are called as clone objects,because both will refer the same id's of

a1 and a2

a3 id and a1 id are same

a4 id and a2 id are same

example:
========

```
a1=A()
print(type(a1))
#here we create the object using another object of same class
a2=a1
print(type(a2))
print(id(a1))
print(id(a2))
```

in above example,if we see both a1 and a2 are having same id,a2 can be called

as clone object of a1

how to call the a method inside another method in the same class:
===

whenever we want to call any method inside the another method in the same

class.

we will use "self" keyword

example:
========
```
class A:
    #create two methods in the class
    def display(self):
        print("this is display method")
    def display2(self):
        self.display()
        print("this is diplay2 method")
#how to call method outside the class
#by using object
a1=A()
a1.display2()
```

how to call/access the attributes or data of class in the methods of the same class:
==

to call/access the attributes or data of class in the methods of the same class,we

will use "self" keyword

example:
=======

```
class A:
    #data/attributes
    a=10
    b=20
    c=30
    #method
    def display(self):
        print(self.a)
        print(self.b)
        print(self.c)
a1=A()
a1.display()
```

note:
====

self is a "keyword or reserved word " in python

self is represents "current class object"

if we want to access the any data or method in the same class method,we will

use "self" keyword

whenever we create any method,we take "self" as first argument to the method

in the class

constructors in python classes:
=========================

what is constructor?
================

constructor is a method in python class

constructor is used to "initilize the data of an object"

constructor can be called "during the object creation"

to call the constructor,we need just "create the object for constructor associate class"

constructor never call by object,constructor can be called automatically when we create the object to class

how to create the constructor in the python:
===================================

to create the constructor in pytthon we will use the following syntax:
===

```
def __init__(self):

      #write the logic for constructor
```

in python,we will have two types of constructors:
=======================================

1.default constructor:
==================

a constructor which not having any argument or constrctor with no paramerts can be called as "default constructor"

syntax:
======

```
def __init__(self):

      #write the logic for constructor
```

2.paramterized constructor:
=======================

if constructor takes atleast one argument or parameter or constructor with paramters can be called as "parameterized constructor"

syntax:
======

```
def __init__(self,arg1,arg2,arg3,........argn):

      #write the logic for constructor
```

example-1: (using default constructor)

```
==============================

class Sample:
    def __init__(self):
        print("this is default constructor")
s1=Sample()

example-2: (using paramterized constructor)
====================================

class Sample:
    def __init__(self,a,b):
        print("a:",a)
        print("b:",b)
s1=Sample(100,200)


how to initilize the object using constructor:
===================================

example-3:
=========

#here we initilize  the data for object using "default constructor"
class Sample:
    def __init__(self):
        a=int(input("enter the data for a:"))
        b=int(input("enter the data for b:"))
        self.a=a
        self.b=b
s1=Sample()
print(s1.a)
print(s1.b)


example-4:
=========
#here we initilize the object via using parameterized constructor
class Sample:
    def __init__(self,a,b):
        self.a=a
        self.b=b
a=int(input("enter the data for a:"))
b=int(input("enter the data for b:"))
s1=Sample(a,b)
print(s1.a)
print(s1.b)


example-5:
=========

#here we initilize the object via using constructor
class Sample:
    x=100
    y=200
    def __init__(self):
        self.x=self.x+100
        self.y=self.y+100
```

```
s1=Sample()
print("x value:",s1.x)
print("y value:",s1.y)
```

example-6:
=========

```
#here we initilize the object via using constructor
class Sample:
    x=100
    y=200
    def __init__(self):
        print(self.x)
        print(self.y)
    def __init__(self):
        self.x=self.x+100
        self.y=self.y+100
    def __init__(self):
         self.x=self.x-50
         self.y=self.y-50
s1=Sample()
print("x value:",s1.x)
print("y value:",s1.y)
```

note:
====

we can create as many as constructors in class,but all are should be similar type

inheritance in python:
==================

inheritance means "giving one class properties(data and methods) to another

class"

the main important feature of inheritance is "code reuseability"

because of the code reuseability,

the amount time taken to do the project may be reduced(productivity will increases)

the amount of cost incuurred will be reduced on the project

inheritance terminology:
====================

parent class/base class/super class:
============================

the class "which is giving the properties" is called as "parent class or

base class or super class"

child class/derived class/sub class:
============================

the class "which is taking the properties from another class" is called as

"child class or derived class or sub class"

super class data:
=============
the data which are used by the child class or the data which are derived

by the child class are known as "super class data or parent class data"

super class method:
================

the methods which are used by the child class or the methods which are derived

by the child class are known as "super class methods or parent class methods"

type of inheritance:
================

single inheritance:
===============

in this inheritance,only we have two classes,those are parent and child

multiple inheritance:
=================

multiple inheritance means "one class can derive the properties from one or

more parent classes"

multi-level inheritance:
===================

in inheritance one class will give the properties another class,the same another

will properties to another class and so on.........'

syntax for inheritance on classes:
===========================

```
class A: <======= parent class/base class/super class

     #write the logic for A class

class B(A): <========  child class/sub class/derived class

    #write the logic for B class
```

example-1:
=========

```
#single inheritance
```

```
class A:#parent class
    a=100
    b=200
    c=300
class B(A):#child class
    d=1000
b1=B()
print(b1.a)
print(b1.b)
print(b1.c)
print(b1.d)

example-II:
=========

#single inheritance
class A:#parent class
    a=100
    b=200
    c=300
    def display(self):
        print(self.a)
        print(self.b)
        print(self.c)
class B(A):#child class
    def display2(self):
        print(self.a)
        print(self.b)
        print(self.c)
b1=B()
b1.display()
b1.display2()

example-III:
=========

#single inheritance
class A:#parent class
    a=100
    b=200
    c=300
    def display(self):
        print(self.a)
        print(self.b)
        print(self.c)
class B(A):#child class
    d=1000
    def display2(self):
        self.display()
        print(self.d)
b1=B()
b1.display2()

example-IV:
==========

class A:# A is parent class to C
    #data
    a=10
    #method
    def display(self):
        print("this is class A")
class B:# B is parent class to C
    #data
    b=20
    #method
    def display2(self):
        print("this is class B")
#multiple inheritance
class C(A,B):
    pass
c1=C()
print(c1.a)
print(c1.b)
c1.display()
c1.display2()


example-V:
=========

class A:# A is parent class to B
    #data
    a=10
    #method
    def display(self):
        print("this is class A")
class B(A):# B is parent class to C
    #data
    b=20
    #method
    def display2(self):
        print("this is class B")
#multi-level inheritance
class C(B):
    pass
c1=C()
print(c1.a)
print(c1.b)
c1.display()
c1.display2()

example-VI:
=========

class A:
    #A'class deafult constructor
    def __init__(self):
        print("this is default constructor of A class")
class B(A):
    #B'class default constructor
    def __init__(self):
       super().__init__()
       print("this is default constructor of B class")
b1=B()

note:
=====
```

in python,we will use super() method,for the following:
==

1.to call or access the super class methods

2.to call or access the super class constructors

example-VII:
==========

```
class A:
    def method1(self):
        print("this A class method")
class B(A):
    def method2(self):
        super().method1()
        print("this is B class Method")
class C(B):
    def method3(self):
        super().method2()
        print("this is C class method")
c1=C()
c1.method3()
```

Data Encapsulation and Data Abstraction:
=================================

Data Encapsulation:
================

"wrapping the both data and methods together as single unit"

example:
=======

implementing the class

Data Abstraction:
==============

"hiding the the result or information,which not need by the user,show the

details what user need"

example:
========

while inheritance,if we make any member of the class can not able to access

by it's child classes,then we are following data abstaraction in the parent class

or super class

access modifiers/specifiers in the python:
=================================

```
in python,we will have three types of access modifiers/access spesfiers:
=========================================================

public:
======

when we make any member of the class as "public",then those members can accessed
any where in the class,outside the class,in other classes too

in python,by default all members of the class are "public"

private:
======

when we make any member of the class as "private",then those members can

be accessed only that class

in python,to create any member as "private",we will use "__"

we will give "__" at starting name of member of the class

example:
========

class A:

    __a=100  #private

   __b=200  #private

def  __display(self):#private method

note:
====

if all memebers of the class are private,then the class is called as "Selaed class"


private:

protected:
========

when we make any member of the class as "protected",then those members can

be accessed only that class and it's subclasses

in python,to create any member as "protected",we will use "_"

we will give "_" at starting name of member of the class

example:
=======

class A:

     _a=100
     _b=200

def _display(self):

      #code here
```

example-1:
=========

```
class A:
    #here a,b,c are public data
    a=100
    b=200
    c=300
    #here display() is public method
    def display(self):
        print(self.a)
        print(self.b)
        print(self.c)
a1=A()
print(a1.a)
print(a1.b)
print(a1.c)
a1.display()
```

note:
====

public data can access anywhere outside the class in python

public data can access even it's child classes too

to declare any public data,we no need to create any prefix with "underscore"

like private or protected

by default all members in the class are "public" in python

example-II
=========

```
class A:
    #here a,b,c are private data
    __a=100
    __b=200
    __c=300
    #here display() is private method
    def __display(self):
        print(self.__a)
        print(self.__b)
        print(self.__c)
    def display2(self):
       self.__display()
a1=A()
a1.display2()
```

note:
====

private members can not be accessed in it's sub classes

private members not be accessed out side the classs

private members can be accessed only in same class where we define the

"private members"

when we are dealing with private members,we should use those members

via public or protected members in outside the class or it's subclasses

```
example-III:
=========
class A:
    #here x,y are protected data
    _x=100
    _y=200
    #here z is private data
    __z=300
    #here p is public data
    p=500
class B(A):
    #here display is protected method
    def _display(self):
        print(self._x)
        print(self._y)
b1=B()
print(b1._x)
print(b1._y)
#print(b1.__z)# here __z is private data
print(b1.p)
```

note:
====

generally "protected" data can be accessed only in the "same class or

its sub classes"

but in python,we have convention like "protected can be also accessed outside

the class as well"

static method:
===========

generally methods are bounded to the "object of the class"

we can make the method of class bounded to "class" itself

it means "when we make any method as static,then those methods are can

accessed via class name itself,without creating any object like normal method"

the static methods are created in Python,with help of "@staticmethod" decorator

```
syntax:
@staticmethod
def method_name(arg1,arg2,....argn):

        #here the code

static methods will not take "self" as an argument

syntax to call static method:
======================

class_name.method_name(val1,val2,val3,val4,..................)

example:
========

class A:
    @staticmethod
    def display():
        print("this is static method")
    #instance method
    def display2(self):
        print("this is normal method")
A.display()
a1=A()
a1.display()
a1.display2()
#A.display2()

example-II:
=========

class A:
    #instance data
    a=10
    b=20
    c=30
    @staticmethod
    def display():
        print("this is static method")
        a=100
        b=200
        c=300
        print(a)
        print(b)
        print(c)
    #instance method
    def display2(self):
        print(self.a)
        print(self.b)
        self.display()

a1=A()
a1.display2()

note:
====
```

static method can called or accessed in the instance methods

instance data or methods can not called in "static methods" in python

there are four types of methods in general,those are:
==

1.instance method:
================

instace method is "a method which is bounds with object"

example:
=======

```
class A:
   #instance data
   a=10
   b=20
   c=30
   #instance method
   def display(self):
       print(self.a)
       print(self.b)
       print(self.c)
#create the object for A class
a1=A()
a1.display()
```

2.static method:
=============
static method is a "method which is bound with class"

example:
=======

```
class A:
  @staticmethod
  def display():
      print("this is static method")
A.display()
```

3.concrete method:
================

concrete method which is method with "fully implementation" in the class

concrete method can be "instance method or static method"

example:
========

```
class A:
  #instance method
  def display(self):
      print("this is instance method")
  @staticmethod
  def display2():
      print("this is static method")
A.display2()
a1=A()
a1.display()
```

4.abstract method:
===============

this is opposite to "Concrete method"

abstract method is a method with no "implementation" and it's implementation

will given by it's subclasses

if any class having atleast one abstract method,then the class is called as

"abstract class"

to deal with abstract method,we will use a "module" called "abc"

in this module we will have a decorator called "@abstractmethod"

syntax:
======
```
from abd import ABC,abstractmethod
@abstarctmethod
def abstract_method_name(self):
     pass
```

note:
====

when we create any abstarct method in the class,the method will be created

with help of "@abstractmethod" decorator

when we create any abstractmethod,the class become "abstract class"

the abstarct class or class which is having abstract method,must take a parent

class called "ABC"(abstract base class)

this "ABC" class is "superclass" for all abstract classes

in python, we can not create the object for "abstract class"

example:
========

```
from abc import ABC,abstractmethod
class A(ABC): #here "A" is abstarct class
    @abstractmethod
    def display(self):
        pass
    @abstractmethod
    def display2(self):
        pass
class B(A):
     def display(self):
         print("this abstract method of A")
     def display2(self):
        print("this display 2 abstract method of A")
b1=B()
b1.display()
b1.display2()
```

if a class does not have any abstact method,then the class is known as

"concrete class"

in the concreate class,all methods are fully implemented or empty method,but

not abstarct method

polymorphism:
============

poly means "many"

morph means " form"

ism means "exhibiting"

polymorphism "exhibiting multiple forms based on the scenario"

in python,we can implement "polymorphism",in the following ways:
===

1.compile time polymorphism:
=========================

if the polymorphism exhibits at "compile time",then the polymorphism is called

as "compile time polymorphism"

to implement compile time polymorphism,we will use "method overloading"

2.run time polymorphism:
=====================

if the polymorphism exhibits at "run time",then the polymorphism is called

as "run time polymorphism"

to implement run time polymorphism,we will use "method overriding"

working with compile time polymorphism:
==================================

to implement compile time polymorphism,we will use "method overloading"

method overloading means "define the method with same name but different

in number of arguments or number of type of arguments" in Python,to implement method overloading,we will use "default arguments with

method"

python advance keywords:
=====================

1.break:
=======

it will stop or break the loop pre-maturely

break is also called as "unconditional statement"

break is also called as "flow-control" statement,beacause it will change the

flow of execution

syntax:
======

break;

example:
========

```
a=int(input("enter the number:"))#1
while a>0:
    if a>10:
        break #stop the execution of loop
        print(a)
    else:
        print(a) #1 2 3 4 5 6 7 8 9 10
    a=a+1
```

note:
====

at a time,break will come from "one loop at a time"

break always used with "loop" statements only

2.continue:
=========

continue will skip the current iteration in the loop
continue will used with loops
the code what we write after "continue", will not executed

example:
```
a=1
while a<=10:
    if a==5:#infinite loop
        continue
    print(a)
    a=a+1
#output:1 2 3 4 6 7 8 9 10
```

```
example-2:
=========

a=1
while a<=10:
    if a==5:
      a=a+1
      continue
    else:
      print(a)
      a=a+1
#output:1 2 3 4 6 7 8 9 10

example-3: (print numbers from 1 to 10 ,except 5,9,3)
=========

a=1
while a<=10:
    if a==5 or a==3 or a==9:
      a=a+1
      continue
    else:
      print(a)
      a=a+1
#output:1 2 3 4 7 8 10

3.pass:
======

pass is used in python "to make any block as empty"

we will use pass in the following cases in python:
=======================================

1.with conditional statements:
========================

if condition: #empty if statement

    pass

or

else: # empty else statement

    pass



2.loop statements:
===============

while condition: #empty while loop

        pass

for variable_name in iterable_name:#empty for loop

       pass
```

3.functions:
==========

```
def function_name(arg1,arg2,arg3,....argn):#empty function

      pass #when we create any function with "pass",then function is empty
                fucntion
```

4.classes:
========

```
class class_name:#empty class

      pass
```

4.global keyword:
==============

it is used to make the local data as "global data"

local data "the data which is created inside the function"

global data "the data which is create outside the function"

example:
=======

```
x=1#here x is global data
def display():
    global x
    x+=1#x=x+1
    print(x)
    global y
    y=10#local data
    print("inside the function:",y)
display()#call the display function
print("outside the function:",y)
```

5.nonlocal:
=========

this keyword is used to access the data of nearest function

when we are working with "inner functions",we can not access the "inner

function data inside the parent/outside function"

example:
=======

```
def display():
    x=10
    y=20
    z=100
    def inner():
        nonlocal z
        z=200#here z is inner function data
        print(z)
        print(x)# x is parent/outside fucntion  data
        print(y)# y is parent/outisde function data
    inner()
    print(z)
display()
```

6.match:
=======

match is used to excute a "piece of code based on the given value at match"

syntax:
======

```
match data/object:

    case  value:

                        code

    case value2:

                        code

    case  value3:

                        code

    case  value4:

                       code


       .

       .

   case  _:     code
```

example:
========

```
data=int(input("enter the data:"))
match data:

       case 1: print("given data is 1")
       case 1: print("given data is 2")
       case 1: print("given data is 3")
       case _:print("no case statement is match with given data")
```

example-2:
=========

```
data=int(input("enter the data:"))
data=(data%2==0)
match data:

       case True: print("given data is even")
       case False: print("given data is odd")
       case _:print("no case statement is match with given data")
```

example-3:
=========

```
data=int(input("enter the data:"))
match data:

       case 1|2|3: print("given data is 1or 2 or 3")
       case 4|5|6: print("given data is 4 or 5 or 6")
       case 7|8|9: print("given data is 7 or 8 or 9")
       case _:print("no case statement is match with given data")
```

note:
====

here case _ is represents the "default case(when no case value matched with

given match data,then default case will execute)

7.try,except,finally,raise,assert(error handleing or exception handling):
==

exception:
========

exception means "error"

exception means "an runtime error which causes stops the normal flow of

execution"

when we have exception in the program or code,the program never give

desired output

exceptions are not "syntax error"

exceptions are related to "logic related"

to handling the exceptions in python program,we are using "exception handling"

exception handling will always done by "programmer or developer"

python will provide the exception handlers,those are follows:
===

try:

try is used "to write the code which will give the exception"

in try,we will write the code which will give the exception

or

try will have "risky code"

syntax:

```
======

try:

    #write the code here


except:
======

in this,we will write "the code which will handle the exception"

or

the code what give "Details about exception",we will write inside the

except block




finally:
======

in python,finally we will use "to write the code which has execute always

weather there is exception or there is no exception in the code"

in finally,we will write the "safe code or code has to execute always"

raise:
=====

raise is used "to raise the exception or to raise the a particular exception"


assert:
=====

assert is used to "check the each and every line is as per the given condition

or not"

statement level wise "if we need to de-bugging in the code,we will use "Assert"

in python"

in python we will have the following exceptions:
=======================================

1.TypeError:
==========

example-1:
=========
a=10
b="a"
```

```
print(a+b)


example-2:
=========

a=10
b="a"
print(a+b)#TypeError
a=10,20,30
print(a)
b=100
print(a+b)#TypeError

2.ValueError:
===========

a=int(input(" enter the a value: "))#0x10
b=int(input("enter the b value: "))
print(a,b)#ValueError


3.keyerror:
=========

l1={1,2,3,4,5,6,7,8,9,10}
#l1.remove(100)
d1={1:2,3:4,5:6}
#print(d1[7])

4.IndexError:
==========

example:
=======

l1=[1,2,3,4]
print(l1[6])

5.NameError:
===========

x=10
print(X)

6.ModuleNotFoundError:(when the module is not found)
====================

import mathe as mt
import numpy as np
import django as dj
#print(mt.floor(10.5))#ModuleNotFoundError
print(np.ndim)

7.AttributeError:
=============

l1=[1,2,3,4]
l1.discard()
```

```
exception handling template:
=======================

try:

    #here we write the code (which causes the exception)

except:

    #here we will write to handle the exception or it will details about exception


finally:

  #this code will execute always

example-1:
=========

try:
    a=int(input("enter the code for a:"))
    b=int(input("enter the code for b:"))
    print(a,b)
except ValueError:#it will work only for ValueError
    print("please only decimal number as input")
except:#it will work any exception
    print("please check the data")
finally:
    print("i am always executed")

note:
====

when we create "try" block in the python program,we must create try along with

either "except or finally"

when we write "Exception handling code "we must use try and except only" "


example-2:
=========

try:
    a=int(input("enter the value for a:"))
    b=int(input("enter the value for b:"))
    print(a/b)
except ValueError:
    print("plese give valid data for a or b")
except ZeroDivisionError:
    print("b never be Zero")
except:
    print("plese check your logic")

example-3:
=========
try:
    a=int(input("enter the value for a:"))
    b=int(input("enter the value for b:"))
    if a>=1 and a<=1000:
        print(a)
    else:
        raise ValueError("value must be in between 1 and 1000")
finally:
    print(a+b)
```

example-4:
=========

```
try:
    a=int(input("enter the value for a:"))
    b=int(input("enter the value for b:"))
    if a>=1 and a<=1000:
        print(a)
    else:
        raise Exception("value must be in between 1 and 1000")
finally:
    print(a+b)
```

note:
=====

here ,we will use "Exception",to raise any exception/we will this when we do not

know what exception has to raise

example:
========

```
try:
    a=int(input("enter the value for a:"))
    b=int(input("enter the value for b:"))
    if a>=1 and a<=1000:
        print(a)
    else:
        raise
finally:
    print(a+b)
```

try with else block:
===============

try with else template code:
======================

```
try: # in this we will write the code which will give exception

    #here write the code

except: #in this we will write code related to exception

   #here we write the code

else: #in this we will write the code (this will execute always when there no
                                                                exception)

  #here we write the code

finally:#in this we will write the code (this will execute always either exception

                                                                    or no exception)

   #here we write the code
```

example:
=======

```
try:
    a=int(input("enter the value for a:"))
    b=int(input("enter the value for b:"))
    if a>=1 and a<=1000:
        pass
    else:
        raise
except:
     print("value must be in 1 to 1000")
else:
    print(a+b)
```

note:
====

here else will execute only "when there is no exception in the code"

custom exception:
===============

custom exception also called as "user -defined exception"

custom exceptions will created by the "programmers or developers"

when we are creating the "custom exceptions",we need to inherit/extend/

use the class called "Exception"

in python for all exceptions "Exception" is super class

in python,every exception is a class,even our custom exceptions also classes

in python,any custom exception we will create with help of class

syntax:
======

```
class  custom_exception_name(Exception):

    pass
```

create a program which deals with custom exceptions:

```
===========================================

class InvalidData(Exception):
    pass
try:
    a=int(input("enter the data for a:"))
    if a<=10:
        raise InvalidData("data is less than 10 or 10")
except InvalidData:
    print("please check yout value(<10 or 10)")
else:
    print(a)
```

create a custom exception "user input always even number for x":
==

```
class NotEvenNumberError(Exception):
    pass
try:
    x=int(input("enter the data for x:"))
    if x%2==0:
        pass
    else:
        raise NotEvenNumberError()
except NotEvenNumberError:
    print("please enter always even number only")
except:
    print("please enter valid data")
else:
    print(x)
```

working with assert:
================

assert will check given statement is true or flase(for given condition)

if condition what we given for assert is True,then it will go to next statement

if condition what we given for assert is False,then it will given an error

"AssertionError"

example:
=======

```
x=int(input("enter the value for x:"))
assert x%2==0
print(x)
```

file handling in python:
===================

directory/folder:
=============

directory means "folder" in system

in system/ machine,the Operating System os will create the "directory or folder" directory means "collection of files"

file:
===

file is a "a location name in the auxilary memory(hard disk)"

in python,we can able to work with files

the process of working with files (to create,read,write,append,remove files) is

known as "file handling"

in python,to work with files we will have a module called "os"

with help of "os" module and other functions,we will do the following operations:
==

1.we can create the directory:
=========================

by writing the simple python program,we can create the directory in python:
===

to create the directory in python,we will have a function called "mkdir()"

in os module

example:
=======

```
import os
os.mkdir("myfiles")
```

how to know the current directory in system using python:
===

to know the current working directory,in python we will use a function called

"getcwd()"

getcwd===>get current working directoty

example:
========

```
import os
print(os.getcwd())
```

2.we can remove the directory:
=========================

to remove the directory in python,we will use a function called "rmdir()"

rmdir===>remove directory

example:
=======

```
import os
```

```
os.rmdir("myfiles")
```

3.we change the directory:
=====================

to change the directory in python,we will use "chdir()" function

chdir===>change directory

example:
=======

```
import os
print(os.getcwd())
os.chdir("D:/training/360digrii")
print(os.getcwd())
```

4.we can open a file:
================

to open the file,in python we will have a function called "open()"

syntax:
======

```
open("file_name.file_extension",mode of the file)
```

here mode represents "what kind of operation we are doing with file"

generally we will have two types of modes:
==================================

r ===>read mode

w===>write mode

a==>we will this mode for "append the file"

r+ ===>we can do both read and write

w+ ===>We can do both read and write

a+ ====>we can do both read and write(Append)

5.we can read the file:
==================

example-1:
=========

```
#open the file
fp=open('sample.txt',"r")
print(type(fp))
print(fp)
#to read the data from the file
#in python,we will use "read()" function
print(fp.read())
#close the file
fp.close()
```

```
#print(fp.read())

example-2:
=========

#open the file
fp=open('sample.txt',"r")
print(fp.read(10))#it will first 10 characters
#close the file
print(fp.read(2))#it will read first 2 characters
fp.close()
#print(fp.read())

example-3:
=========

#open the file
fp=open('sample.txt',"r")
print(fp.readline())
print(fp.readline())
print(fp.readline())
#close the file
fp.close()

read() <===this function will used to"read the data character by character from

                   file"

readline() <=== this function will be used "To read data from file line by line"

example:
=======

#open the file
fp=open('sample.txt',"r")
print(fp.readline(3))
print(fp.readline(4))
print(fp.readline(5))
#close the file
fp.close()

6.we can write the content into the files:
=================================

to write the content into the file,in python we will use a function called

"write()"

syntax:
======

file_pointer_name.write("here we give the data")

example:
=======

#open the file
fp=open('sample2.txt',"w")
fp.write("this is  added")
fp.close()
```

note:
====

when we open the file with "write" mode,if the file is not there in the folder

or directory,python will create " a file with that and write the content into the

file"

otherwise,it will override the previous content,with content what we are writing

with "write()" function

example-2:
=========

```
#open the file
fp=open('sample.txt',"w")
fp.write("this is  added")
fp.close()
```

7.we can append the content into the files:
==================================

in python,write means "remove the previous content of the file,

keep the new content into the file"

in python,append means "add the content into the file at end of the file"

when we are doing append,the mode is "a"

for appending,we will use write() function only

example:
=======

```
#open the file
fp=open('sample.txt',"a")
fp.write("this is  added")
fp.write("this is  added")
fp.write("this is  added")
fp.close()
```

example:
=======

```
#open the file
fp=open('sample.txt',"r+")
print(fp.read())
fp.write("this is  added")
fp.write("this is  added")
fp.write("this is  added")
print("end")
print(fp.read())
fp.close()
```

8.we can remove the files:
====================

to remove the any file from the folder or directory,we will use "remove()"

function,this function will avaliable from "os" module

example:
=======
```
import os
os.remove("Sample2.txt")
```

with keyword in python:
==================

with keyword in python,is used to manage the resources (files,databases,network

connections,.................)

generally when we open any file with open() function,we need to close the

file with close() function

with is used to "Create the context managers in python"

syntax:
======

```
with open() as filepointer_name:

          #write the logic or code inside the with
```

example:
=======

```
with open("sample.txt","r") as fp:
    print(fp.read())
print(fp.read())
```

higher order functions in python:
===========================

with help of higher order fucntions,we will take "function as argument in the

other functions"

in python,we will have the following three main important higher order functions:
===

1.map()

2.filter()

3.reduce() (this function from itertools module)

map():
=====

```
in python,map() function will take "function as argument,and take data as

iterable","peform the changes  in iterable as per logic in the function",return

the result as "map object"

syntax:
======

map(function_name,iterable_name)

example:
========

l1=[10,20,30,40,50]
result=map(lambda x:x+10,l1)
l1=list(result)
print(l1)
l2=[100,200,300,400,500,600,700,800,900]
result=list(map(lambda x:x//100,l2))
print(result)

example-2:
=========
l1=["hello","hi","world","bye","python"]
length_list=list(map(len,l1))
print(length_list)#[5,2,5,3,6]
reverse_list=list(map(lambda str1 :str1[::-1],l1))
print(reverse_list)

example-3:
=========

salaries=[10000,20000,30000,50000,60000,70000]
l2=[10000,20000,30000,50000,60000,70000]
#increase the salary by 30%
#without map()
index=0
while index<len(salaries):
    salaries[index]+=salaries[index]*(0.3)
    index=index+1
print(salaries)
#with map
l2=list(map(lambda x:x+x*(0.3),l2))
print(l2)

example-4:
=========
salaries=[10000,20000,30000,50000,60000,70000]
#increase the salary by 30%
#print the 2nd largest salary from the list
salaries=list(map(lambda x:x+x*0.3,salaries))
salaries.sort(reverse=True)
print(salaries[1])

example-5:
=========

salaries=[10000,20000,30000,50000,60000,70000]
```

```
#increase the salary by 30%
#print the difference of max and min salary in python
#difference always positive
salaries=list(map(lambda x:x+x*0.3,salaries))
salaries.sort(reverse=True)
print(salaries[0]-salaries[len(salaries)-1])
```

example-6:
=========

```
l1=["hello","hi","bye","python"]
#in place of each string remove second letter
#keep letter "h" in the second letter
#output :[hhllo,hh,bhe,phthon]
def change_char(str1):
    return str1[0]+'h'+str1[2:]
result=list(map(change_char,l1))
print(result)
result2=list(map(lambda x:x[0]+'h'+x[2:],l1))
print(result2)
```

2.filter():
=======

in python,filter function will "return the data based on the

condition in the function"

in python,filter function also can take the "function as a

argument" and iterable as a data

syntax:
======

```
result=filter(function_name,iterable_name)
```

example:
=======

```
l1=[10,20,30,40,50,60,70,80,90]
result=list(filter(lambda x :x>10,l1))
print(result)
l2=[10,20,30,40,50,60,70,80,90]
result=list(filter(lambda x :x==10,l2))
print(result)
l3=[10,20,30,40,50,60,70,80,90]
result=list(map(lambda x :x>10,l3))
print(result)
```

write a python program to "return all strings which are having

atleast one vowel as a character" in the given list:

```
code:
====

l1=["abc","bca","cab","efg","ghg"]
def check_vowel(x):
    count=0
    for i in x:
        if i in "aeiouAEIOU":
            count+=1
    if count>=1:
        return True
    else:
        return False
l1=list(filter(check_vowel,l1))
print(l1)


write a python program to "return the numbers from the list

which are not having any prime digit or zero"

==================================================

l1=[456,789,123,567,896,789,468]
def check_prime(x):
    count=0
    x=str(x)#here number converted into string
    for i in x:
        if int(i)  in [1,4,6,8,9]:
            count+=1
    if count==len(x):
        return True
    else:
        return False
l1=list(filter(check_prime,l1))
print(l1)


reduce():
=======

example:
=======

from functools import reduce
l1=[1,2,3,4,5,6,7,8,9,10]
sum=reduce(lambda x,y:x+y,l1)
print(sum)
```