types of applications:

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

the following are the applications, we have the in the real-time:

==

1.stand-alone application

2.web application

3.enterprise application

4.mobile application

5.distributed application

1.stand-alone application:

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

stand-alone application is also called as "desktop application" if we say any application is "stand-alone", then the application is does not require any internet to access the application and does not require any server to run the application

example: calculator,clock,billing application

the main drawback of the application is "We can anot able to send the data from one device to another device ,because no internet

2.web application:

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

web application is also called "internet-based application" if we say any application is "web" application,then the application will uses internet to access the application and it will also uses server to run the application

example: gmail,bookmyshow,amazon,college website,....

in this application,we can able to send the data from one device to another device

3.enterprise application:

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

enterprise application is a "Web application" (it means it also uses internet for accessing and server to run the application) but in enterprise application, we can able to make the server can communicate with other servers, which not available in web application all enterprise applications can be called as "web application", but not vice versa enterprise application is also called as "business specific/business oriented application"

example: amazon,flipkart,mynthra,zomato,..........

in the case of enterprise application,we will have two types of

server-server communication:

===

1.third-party server communication:

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

in this server-server communication,the two servers will communicate with each other via API's and not related to same company/entity

2.native server communication:

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

in this server-server communication, the two servers will communicate with each other via API's and related to same company/entity

4.mobile application:

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

mobile application can be a "stand-alone/web application/enterprise application", but it runs in the mobile based on the specific operating system the famous mobile operating systems are

"android/ios/windows/black berry/symbois" when we develop the following like:

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

android mobile application using java/kotlin/python(kivy)/dart(flutter) ios mobile application using swift+objective C windows mobile application using xamrin(C#)

5.distributed application:

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

distributed application means "the application can able to run in the multiple devices at a time in the real-time over a network"

what is python and why python?

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

python is a "high-level, object-oriented, strongly typed,dynamic typed , programming language”

why python is a high-level language?

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

in general ,in computers we will have three types of languages:

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

high-level:

========

if a language is understand by the "programmers or developers", then the language is high-level language

example: c, c++, java, python, java script,.....

low-level:

=======

if a language is understand by the machine/computer, then the language is "low-level" language

this language is also called as "binary/machine" language

this language is composed of " 0's and 1's" when any programmer or developer write any code, first the programmer or developer will give the code to the translator,once the translator takes the code from the programmer,it will convert the given code into binary,once it is in binary,then

we will the binary code to the machine to process and once machine process it will give the result.

code ====>translator ===>binary ===>machine===>output

when we want to convert the given code into machine-level code,we will use the following translators:

===

1.compiler:

compiler is a language translator and using compiler we can convert the given code/program into binary compiler will translate the given whole code at a time into binary the translation process of the compiler is called as "Compilation" generally compiler used by all programming languages and programming languages means "the language which uses compiler"

example: c,c++,java,python,typescript,..........

in case of compiler,

de-bugging is not very easy the performance is very high when compare with interpreter

2.interpreter:

interpreter is a language translator and using interpreter we can convert the given ode/program into binary interpreter will translate the given code in line by line into binary the translation process of the interpreter is called as "Interpretation" generally interpreter used by all scripting languages and scripting languages means "the language which uses interpreter only"

example: java, python, java script, sql, php, ruby..........

in case of interpreter,

de-bugging is very easy the performance is very low when compare with compiler

why python is object-oriented language:

python is object-oriented programming language object-oriented is a python programming paradigm(model) when we say any language is "object-oriented", the language must follow the following concepts:

1.classes and objects

2.inheritance

3.polymorphism

4.data encapsulation

5.data abstraction

python is often called as "multi-programming paradigm language", because python supports procedural/functional -oriented language

why python is dynamic typed language?

python is a dynamic typed language and because in python, when we are storing any data into the variable, we need to specify the datatype to the variable, the variable can be particular type in python ,based on the data what we store in the variable.

why python is strongly typed language?

python is a strongly typed language and this language can allow the programmers or developers, perform the operation on similar/same type operands, we can not able perform operation on two different type of the operands

why python?

python is a platform independent language:

platform=processor + operating system when we write any python program, first the program will given to the translators, when we want give the python code to translators, generally it happens as follows:

==

step-1:

first we will give the python code to the python compiler,once python compiler takes the python code,it will generates a intermediate code called "byte code"(its file extension is .pyc) sample.py ====>python compiler ===>sample.pyc(byte code).

this byte code will run in any platform,that is the reason we are calling python as "platform independent" language once we got the byte code,this byte code will given to "PVM" PVM(python virtual machine) only can understand "byte code" here PVM is platform dependent sample.pyc ===>PVM(interpreter) ====>object/binary code python syntax very simple to understand than other langauges python having a rich set of libraries python having a big community and because of this we will get huge number of updates and new versions from python

python can use any where,like follows:

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

application development

gaming development

testing

Artifical intellingence

prompt engineering

data science

data analytics

IOT

networking

Cloud and Devops...........

core python:

1.python character set:

in python,we will have the following types characters:

1.letters:

1.upper case chracters/letters:

A,B,C,D,E,F,...........Z

2.lower case characters/letters:

a,b,c,d,e,f,g.....z

2.digits:

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

3.special symbols/characters:

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

python is not a case-insensitive language(it treats both upper case and lower case same)

python is a case sensitive language(it treats both upper ase and lower case differently)

example:

=======

abc and ABC are different in python

2.python keywords/reserved words:

these words are given by the "python" language using these words, we can able to perform a specific task in the program the following are some of the important keywords in python:

or,not,in,is,and <==== operator related keywords in python

if,elif,else,case,match<=== control flow related keywords

while,for<=== looping/iterative keywords

continue,break<=== un-conditional control flow keywords

def,return,lambda<=== functions related keywords

global,nonlocal<=== scope related keywords

import,from,as <=== modules/packages related keywords

True,False<=== Boolean values related keywords

try,except,finally,assert,raise <=== exception handling related keywords

class,self<=== class related keywords

with,yield,other keywords in python

3.python comments:

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

comments are "describe the code" with help of comments "any programmer or developer can

understand the code" comments will give better "Readability and understandability" of the code

comments will ignore by the compiler while processing comments will never effect the ouput/result of the program" in python, we can able to write the comments in 2 ways:

==

1.single line comments:

in python, we can write the comment in a single line using "#" symbol

example:

#author:Ram

#program for adding two numbers x,y

2.multi-line comments:

when we want to write the comments in the multiple lines, we can use "triple double/single quotes" in python
example:

```
"""
author : ram
program for adding two number x,y
"""
or
'''
author : ram
program for adding two number x,y
'''
```

4.python identifiers:

identifier means "any name in the program " the name may be "variable name, list name, tuple name, function name, class name or any name in the program" are called as "identifier" to create the identifier in python, the python uses the following

Rules:

1.every identifier name must starts with either letter or underscore

2.the identifier name may have the following chracters only:

a-z/A-Z , 0-9

only one special symbol (_)

3.the name of the identifier in python may be "alphanumeric"

4.indentifier name never be "keyword / reserved word" name in python

5.identifier name length can be any thing, use short names for identifiers

example:

abc123(valid)

abc_123(valid)

abc123$(invalid)

12abc(invalid)

abc#(invalid)

#abc(valid,but it is comment)

5.python data types:

data means "what we can able to store in the computer memory"data type " it will represent the type of the literal ,which is present in the variable"literal means "value" in python,we will have the following built-in data types:

1.numeric type:

this type is related to number in python, we will three types of numbers:

1.integer number:

these number are may be +ve or -ve

example:

1234,-1234,-100,9876,..........

how negative numbers can be stored in memory?

all negative numbers can be stored in memory, using 2's complement form

2.real/floating-point number:

these numbers are having decimal point and these are may be +ve or -ve

example:

1.234,5.678,9.0123,-0.934,...........

how a floating-point number can be stored in the memory?

Any floating-point number can be stored in the memory,using IEEE 754 format

3.complex number:

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

Complex number is in the form of "a+ib" where "a is real number and b is imaginary number"

in python,we store the complex number in the form of "a+bj"

example:

=======

10+5j,-5j,11-20j,..............

can complex number be the float or not?

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

yes, complex number can be real or float

example:

========

1.2+10.3j

2.sequence type:

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

sequence types in python, we have the following:

==

1.list===>it can able to store any number values or any number of any type of values under single name

2.tuple===>it can able to store any number of values or any number of any types values under single name

3.range()<=== it is a function and it can generate a group of values based on the given start and end value

note:

====

in python, list is mutable type(when we say any data type is mutable in python,it can allow insert/update/delete operations) in python ,list, set , array and dictionary are mutable types in python, tuple is immutable type(when we say any data type is immutable type,then we can not able to perform operations like insert/update/delete) in python, tuple,string and range() are immutable types when we say any data is sequence type, then it will follow both "indexing and slicing" when we say any data is sequence type, then it will can able to store or manipulate a group of values at a time when we say any data is sequence type, then it will called as "ordered-collection"(because the way we give data,it stores the data in the same way)

3.boolean type:

in this type, we will have only two types of litreals(values):

==

True(it will be in the number as "1")

False(it will be in the number as "0")

4.string type/character type:

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

any data which is enclosed in the "single/double/triple" quotes can be called as "string", in Python

for example:

'a'<=== string

"a"<===string

'''a'''<==== string or """a"""<===string

what is doc string in python?

doc string is a string in python, which is enclosed in triple quotes(''' or """) generally ,it is used for "comments" (multi-line comment)

5.set type:

set is used in python "to store multiple values under single,all value should be unique/distinct" set type will never allow duplicate or reduant dataset type is a "non-sequence type and un-ordered collectiontype in python" set type never follow indexing or slicing in python set type is mutable type set type is also can be "immutable" ,but when we convert theset into "frozenset"

6.map type:

Map type means "in python using map type, can able to store the data in the form of key and value pairs"in python map type, keys are always should be unique in python map type, values are can be duplicate in python, dictionary is map type in python, map type will never follow the indexing and slicing, that is the reason map type can be called as "non-sequence type"

7.Binary type:

In python, we can also represent the data as binary using the following:

Bytes, bytearray, memoryview

8.None type:

None type means "no value/nothing" in python when we do not know ,what type of value, we need to store in the variable, then we can assign "None" as the value python literals:

literals means "value what we can store in the name(identifier)" in python, we will have the following literals:

integer literals====>ex:10,20,-10,-100 floating-point number literals====> ex:1.234,-7.890,.....

complex literals===>ex:5+10j,6-10j,.......

list literals===>example:[1,2,3,4,5,6,7]

tuple literals===>ex: (1,2,3,4,5,6)

set literals===>ex:{1,2,3,4,5,6,7}

range() literals====>ex: range(1,10)

None type literals ===>ex: None

dictionary literals ====>ex: {1:2,3:4,5:6,......}

Boolean literals ===>True or False

6.python variables:

variable in python can have a value or can hold a value in python ,variable can act like container and it can store a value in python, the value of the variable can changes throughout the program variable can also called as "named memory location" to create the variable in python, we will use the following syntax:

variable_name=literal

example:

a=10

b=1.234

c="hello"

b1=True

c1=None

7.python operators:

operator means "symbol" in python using operator, we can able perform a specific operation on operands. Operand means "on which we will perform the operation" for example,

a=10,b=20

a+b ===>30 ,here a,b are operends and + is operator in python, we will have the following operators:

1.airthmetic operator

2.relational / comparison operators

3.logical operators

4.assignment operators

5.bitwise operator

6.membership operator

7.identity operator

8.conditional/ternary operator

9.warlus operator

1.airthmetic operator:

this operator is used to perform various arithmetic operations like addition, subtraction, multiplication, division, exponent and modulo division

symbol:

======

+ ===>addition example: a=10,b=20 ===>a+b===>30

- ===>subtraction   example: a=10,b=20 ===>a-b ===>-10

* ===>multiplication example: a=10,b=20 ===>a*b ===>200

/===>real division   example: a=10,b=5 ===>a/b ===>2.0

[note: in python "/" will give the result always "float/real number]

//===>floor division example: a=10,b=20 ===>a//b==>0

%==>modulo division example: a=10,b=20 ===>a%b===>10

[note:

    in python,

/ or // will give "quotient" as result

% will give "remainder" as result

if n1%n2 (n1<n2),then remainder is "n1"

if n1%n2(n1>n2),then remainder is "actual remainder"

example:

19%6 ===>1,78%12 ===>6,190%456===>190

** ===>exponenet(power)

example:

5**3 ===>125,5**0 ===>1

write a python program to demonstrate the "Airthmetic operators"

code:

```
"""
author:Ram
program to demonstrate "airthmetic operators"
"""
#take the two numbers a,b from user
a=int(input("Enter the a value:"))#10
b=int(input("Enter the b value:"))#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)#0

print("a%b:",a%b)#10

print("a**3:",a**3)#1000
```

2.relational or comparison operators

these operators are used "to compare any two values/numbers" in python, relational operator will give the result as "Boolean"(either True or False) in python,we will have the following relational operators:

1.>(greter than) ==>ex: 10>20 ===>False ,20>10 ==>True

2.<(less than)===>ex: 10<20 ===>True, 20<10==>False

3.>=(greter than or equal to) ex: 10>=20 ==>False

4.<=(less than or equal to) ex: 10<=20 ===>True

5.==(equal to) ex:10==20 ===>False

6.!=(not equal to) ex: 10!=20 ==>True

write a python program to demonstrate the "Relational operators" in python:

```
#take the two numbers a,b from user

a=int(input("Enter the a value:"))#10

b=int(input("Enter the b value:"))#20

print("a>b:",a>b)#False

print("a>=b:",a>=b)#False

print("a<b:",a<b)#True

print("a<=b:",a<=b)#True

print("a==b:",a==b)#False

print("a!=b:",a!=b)#True
```

3.logical operators in python:

these operators are used "to compare any two conditions" these operators will give the result in "Boolean" as same as relational operators in python in python, we will have the following logical operators:

1.or:

rule:

in the case of logical or, if any one condition is "True", then the entire result is True

example:

=======

(10>20) or (10<20) ====>True

(10<20) or (10>20) ====>True

[logical or is "short-circuit" operator, when we are write any expression using logical or, when we are checking any condition and it got "True", then it will never the check the remaining onditions and simply return "True" as result, this behaviour is known as "short-circuit"]

2.and:

rule:

====

in the case of logical and, if any one condition is "False", then the entire result is False

example:

(10>20) and (10<20) ===>False

(10<20) and(10>20) ===>False

note:

[logical and is "short-circuit" operator, when we are write any expression using logical and, when we are checking any condition and it got "False", then it will never the check the remaining conditions and simply return "False" as result

3.not:

not result is always depends on the condition or expression if condition result is True, then the not result is False if condition result is False, then the not result is True

example:

=======

not((10>20)) ===>True

not((10<20)) ===>False

write a python program, to demonstrate the "logical operators" in python

==

code:

====

```
#take the two numbers a,b from user
a=int(input("Enter the a value:"))#10
b=int(input("Enter the b value:"))#20
print("(a>b) or (a<b):",(a>b) or (a<b))#True
print("(b>a) or (a>b):",(b>a) or (a>b))#True
print("(a>b) and (a<b):",(a>b) and (a<b))#False
print("(b>a) and (a>b):",(b>a) and (a>b))#False
print("not((a>b) or (a<b)):",not((a>b) or (a<b)))#False
print("not((b>a) or (a>b)):",not((b>a) or (a>b)))#False
```

4.assignment operators:

this operator is used to "assign a value to the variable" symbol: "="

example:

a=10 <=== here we assign value "10" to a

b=2.456<=== here we assign value "2.456" to b

in python, we will assign multiple values to multiple variables at a time, a, b , c=10,20,30 ====>a=10,b=20,c=30

here below, we assign same value to the multiple variables:

a=b=c=30 ====>a=30,b=30,c=30

using assignment operator, we can also perform compound operations(compound operation means performing more than one operation at a time)

example:

=======

a=100

a+=10 ===>a=a+10 =110<=== addition and assignment

a*=10 ===>a=a*10 =1100

a//=10 <===a=a//10 =110

a%=10<===a=a%10=0

write a python program to demonstrate the "assignment operators" in python

code:

====

```
#take the two numbers a,b from user

a=int(input("Enter the a value:"))#10

b=int(input("Enter the b value:"))#20

a+=10#a=a+10

print(a)#20

b*=100

print(b)#2000

a//=100

print(a)#0

b%=1000

print(b)#0
```

5.bitwise operator:

these operators are perform operation on "binary data of the given operands" when we give the data to this operators, these operators will give the result in normal data only ,but perform operation on the binary data of the given data.

data ===>binary ===>bitwise operator ===>result(in normal) in python ,we will have the following bitwise operators:

==

1.bitwise or (|(pipe)):

symbol: |

rule:

in the case of bitwise or, if we have one input is 1 ,then the entire result is "1"

example:

a=10 ====>01010

b=20====>10100

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

a=45===>0101101

b=67===>1000011

a|b===>1101111<===111

2.bitwise and(&(ampracend)):

symbol: &

rule:

in the case of bitwise and, if we have one input is 0 ,then the entire result is "0"

example:

a=24 ===> 11000

b=30 ===> 11110

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

1 1000

a=89===>1011001

b=76===>1001100

a&b ===> 1001000==>72

3.bitwise exclusive or (ex-or)(^(caret)):

symbol: ^

rule:

====

in the case of bitwise exclusive-or, if both inputs are same ,then the entire result is "0",otherwise output is "1"

a=32===>100000

b=39===>100111

a^b ====>000111<===7

a=54 ===>110110

b=56===>111000

a^b====>001110<===14

4.bitwise left shift(<<):
symbol:<<
formula:
=======
n<<s ==>n* 2 power s
example:
=======
a=10,s=2 ===>a<<2 ===>10*2 power 2 ==>10*4==>40

5.bitwise right shift(>>):
symbol:>>
formula:
=======
n>>s ==>n//2 power s
example:
=======
a=10,s=2 ===>a>>2 ===>10//2 power 2 ==>10//4==>2

6.bitwise one's complement(~(tlide)):

symbol: ~

formula: ~n=-(n+1)

example:

a=10

~a===>-11

a=-27

~a=-(-27+1)=-(-26)=26

write a python program to demonstrate the "bitwise operators" in python:

```
#take the two numbers a,b from user

a=int(input("Enter the a value:"))#10

b=int(input("Enter the b value:"))#20

print("a|b",a|b)#10

print("a&b",a&b)#0

print("a^b",a^b)#30

print("a<<2",a<<2)#40

print("a>>2",a>>2)#2

print("~a",a)#-11
```

6.membership operator:

this operator is used in python "to check the presence of the given value in the iterable (string/list/tuple/set/range())" these operators will return the result in "boolen form(True or False)" in python, we will have the following membership operators:

1.in(it checks for the value present)

syntax for "in":

value in "iterable_name"

2.not in(it checks for value for not present)

syntax for "not in":

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

value not in "iterable_name"

example:

=======

```
a="hello world"

print(a,type(a))

print("l" in a)

print(" " in a)

print("hello" in a)

print("L" in a)

print("123" in a)

print("1" in a)
```

python input and output statements:

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

python output statements:

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

when we want to display/show the any result to the programmer or developer after executing the python program, to display the any program result, in python we will use a function called "print()"

syntax:

======

```
print(data, sep=" ", end=" ")
```

sep is used to "separate the result with any specifier, what we give for sep in the print() function end is used to "display the result in the same line or new line", it is act like where the output has to display in the console the default value for "sep" in print() function is "space/whitespace"the default value for "end" in print function is "\n(new line)" due to the reason every print() function ouput will be displayedin the new line.

input statement in python:

to give any input to the program, in python we will have a function called "input()" function the input() function will take the "input" from the user, the input() function will always takes any input in the form of "String" only any data(number/string/any data)===>input()====>String
number ===>input()====>string
string/character ===>input()===>string
Boolean/complex/any other data===>input()===>string

any data we are giving for input(),it always consider the data as "string" format only, in order to convert the data into actual given format, in python we need to implement a process called "type conversion/type casting"type conversion is "process of converting one data type into another data type"in general, we will have two types of type conversion:

==

1.implicit type conversion:

This type conversion will perform by the python(pvm) and which is never done by programmer or developer in this conversion ,python convert the data from lower data type to higher data type, due to this ,implicit type conversion is also called as "widening"

example:

10+1.2===>11.2

2.explicit type conversion:

This type conversion will perform by the programmer or developer in python in this conversion ,python convert the data from higher data type to lower data type, due to this ,explicit type conversion is also called as "narrowing" write a python program to display "hello world" message:

code:

print('hello world')

write a python program, to demonstrate variables :

a=10

b=1.234

c=True

d="hello"

```
c1=10+5j

print(a)

print(b)

print(c)

print(d)

print(c1)

print(a,b,c,d,c1)

print(a,b,c,d,c1,sep=",")
```

note:

In python, to know the the data type of the given variable value we will use a function called "type()" function write a python program to check the data type of the variables in the given program:

code:

====

```
a,b,c,d,c1=10,1.234,True,"hello",10+5j

print(a,type(a))#10 <class 'int'>

print(b,type(b))#1.234 <class 'float'>

print(c,type(c))#True <class 'bool'>

print(d,type(d))#hello <class 'str'>

print(c1,type(c1))#10+5j <class 'complex'>
```

note:

In python, any value will considered itself as an "object" if there is an object, it will have a class in python all datatypes will be represented as "classes"

object	class name
integer object, ex:10,20,300,...	<class int>
float object	<class float>
string object	<class str>
Boolean object	<class bool>
complex object	<class complex>
list object	<class list>
tuple object	<class tuple>
range object	<class range>
set object	<class set>
frozen set	<class frozenset>
dictionary object	<class 'dict'>
None	<class NoneType>

write a python program to display the all python objects class names

```
#integer object
a=100
print(a,type(a))
#float object
b=1.234
print(b,type(b))
#string object
c="hello"
print(c,type(c))
#complex object
d=10+5j
print(d,type(d))
#None Object
e=None
print(e,type(e))
#list object
l1=[1,2,3,4,5]
print(l1,type(l1))
```

```
#tuple object
t1=(1,2,3,4,5)
print(t1,type(t1))
#set object
s1={1,2,3,4,5}
print(s1,type(s1))
#dictionary object
d1={1:2,3:4,4:5}
print(d1,type(d1))
```

write a python program to demonstrate "input()" function in python:

```
a=input("Enter the number for a:")

b=input("Enter the number for b:")

c=input("Enter the number for c:")

print(a,b,c)

print(type(a),type(b),type(c))
```

to convert the given data into a particular type, python provides the following functions:

1.int():

It is used to convert given "number string or floating-point number" into "integer" number

example:

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

print(a,type(a))

"""to convert the a number string into

integer,we will use int()"""

a=int(a)

print(a,type(a))

"""to convert any float number into integer

,we will use int() function"""

b=1.234

print(b,type(b))

b=int(b)
```

```
print(b,type(b))

c=int()

print(c)
```

example-2:

```
a=int("12_34")
print(a)
a=int("1_2_3_4")
print(a)
a=int("12_34")
print(a)
a=int("+123")
print(a)

a=int("-123")

print(a)
```

note:

int() function will allow in the number string following characters, to convert given number string into a integer number:

_ (underscrore) , only in between the digits

example: 1_23,1_2_3,12_34

+/- at starting of the string

example: +123,-123,....

2.float():

float() function is used to convert the "given number string or integer number into float"

example:

```
a=input("enter the number:")

print(a,type(a))

a=float(a)

print(a,type(a))

b=10
```

```
print(b,type(b))

b=float(b)

print(b,type(b))

print(float("1_2.3_4"))

print(float("+12.3_4"))

#print(float("-12._34"))

print(float("-12.34"))

print(float(".456"))

print(float("123."))
```

note:

float() function will allow in the number string following characters, to convert given number string into a float number:

_ (underscrore) , only in between the digits

example: 1_23.45,1_2_3.4_5_6,12_3.400

+/- at starting of the string

example: +123.0,-123.0,....

"." symbol any where in the number, except not before or after any symbol in the number.

3.bool():

this function is used to "convert the given number or given string or any data into boolean value(True/False)".

example:

```
a=input("enter the data:")

print(a,type(a))

a=bool(a)

print(a,type(a))

a=0
```

```
a=bool(a)

print(a,type(a))

a=0.0

a=bool(a)

print(a,type(a))

a=''#empty string

a=bool(a)

print(a,type(a))

a=None

a=bool(a)

print(a,type(a))

print(bool("Abc"))

print(bool(" "))
```

example-2:

```
a=int(True)
print(a)
a=int(False)
print(a)
a=float(True)
print(a)
a=float(False)
print(a)
a=True+100
print(a)
a=100-False
print(a)
```

4.complex():

complex function is used to convert the "given integer or floating-point number or any number string" into "complex number"

example:

```
a=input("enter the number:")
print(a,type(a))
```

```
a=complex(a)
print(a,type(a))
b=1.234
print(b,type(b))
b=complex(b)
print(b,type(b))
c=True
print(c,type(c))
c=complex(c)
print(c,type(c))
print(complex(int()))
print(complex(float()))
print(complex(bool()))
```

note:

in python, we can not convert a complex number into integer or float number,but we can onvert the complex number into string or Boolean value.

example:

```
a=10+5j
print(a,type(a))
#a=int(a)
#print(a,type(a))
#a=float(a)
#print(a,type(a))
a=bool(a)
print(a,type(a))
print(bool(0j))
print(str(10+5j))
```

6.membership operator:

this operator is used in python "to check the presence of the given value in the iterable (string/list/tuple/set/range())" these operators will return the result in "boolen form(True or False)" in python, we will have the following membership operators:

1.in(it checks for the value present)

syntax for "in":

value in "iterable_name"

2.not in(it checks for value for not present)

syntax for "not in":

value not in "iterable_name"

example:

```
a="hello world"
print(a,type(a))
print("l" in a)
print(" " in a)
print("hello" in a)
print("L" in a)
print("123" in a)
print("1" in a)
```

example-2:

```
s1="hello world"
print("e" not in s1)#False
print("E" not in s1)#True
print("hello" not in s1)#False
print("" not in s1)#False
print(" " not in s1)#False
print("rld" not in s1)#False
```

7.identity operator:

This operator is used to compare the "two objects memory locations are same or not" in Python in Python, to compare any two values(objects),we will use "==" operator these operators will give the result either "True or False" in python, we will have the following two identity operators:

1.is

2.is not

in python, if we store same "object" in two or more variables, then all variables will occupy same memory location in python, we will use a function called "id()",using "id()" function we can able to get the "object memory address or object id at memory"

example:

```
a=10
b=20
c=30
print(id(a))
print(id(b))
print(id(c))
print(id(10))
print(id(20))
print(id(30))
x=100
y=100
z=100
print(id(x),id(y),id(z))
```

example:

```
a=10
b=20
c=1.234
print(a is b)     #a and b are having same memory location
print(a is c)     #a and c are same memory location or not
x=y=z=100
print(id(x),id(y))
print(x is y)
print(id(y),id(c))
print(y is c)
print(y is z)
print(x is not c)
print(a is not a)
```

8.conditional operator:

this operator is also called "tenary operator" this operator will work like a "simple if-else" in python conditional operator syntax:

expression1 if condition else expression2

expresssion1 will be the result, if the condition is True

expression2 will be the result, if the condition is False

example:

```
a=10
b=20
print("a>b") if a>b else print("a<b")
print("a<b") if b>a else print("a>b")
result=a if a>b else b
print(result)#20
```

8.python control flow statements:

in python,we will have two types of control flow statements:

1.condition-based or conditional statements:

in python, we will have the following conditional statements:

1.simple if statement:

when we have a logic has to execute based on the condition, in python we will use "conditional statements" in python, when we have only one condition and it has to execute based on the condition, then in python we will use simple if statement.

syntax:

```
if condition:
    #logic/code for the if
```

note:

the logic what we write under the if will be executed, only when the condition has to be true, otherwise the logic will not be executed.

example:

```
a=int(input("Enter the a value:"))
if a>10 and a>0:
   print("the value of the a is more than 10")
if a<10:print("the value of a is less than 10")
```

how to write the empty if statement in Python:

Empty if statement means "if with no code" in python, if we do not know what code we need to write the inside if statement, we can simply write "pass" in Python, pass is used to represent there is no code in if or any other.

syntax:

```
if condition:
    pass
```

example:

```
a=int(input("Enter the a value:"))

if a>10 and a>0:

  pass

if a<10:pass
```

2.if-else statement:

in python,when we have to conditions,we will the logic using if-else statement if-statement is for "condition" True case else-statement is for "condition" False case else is often called as "counter part of the if" else is an alternative for "if" statement in python.

syntax:

```
if condition:
    #logic
else:
    #logic
```

note:

we will never write any condition for "else"

example:

```
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")
else:
    print("the value of the b is greter than a")
```

3.elif-if ladder:

in python, we will use else-if ladder ,when we have three or more than three conditions syntax for else-if ladder:

```
if condition:
   #logic
elif condition:
   #logic
elif condition:
   #logic
else:
  #logic
```

note:
else-if ladder will always starts with "if" condition
else-if ladder will no need to end with "else", in else-if ladder
"else" is always optional

example:

```
a=int(input("Enter the a value:"))#10
b=int(input("Enter the b value:"))#20
c=int(input("Enter the c value:"))#30
if a>b and a>c:
    print("a is maximum")
elif b>c:
    print("b is maximum")
elif c>a and c>b:
    print("c is maximum")
```

4.match statement:

this is available from python 3.10 version onwards it is same as "switch case" statement in java/c language.

syntax for match:

```
match choice_value:
case value1:  statements/logic
case value2:  statements/logic
case value3:  statements/logic
case value4:  statements/logic
case _: statements/logic
```

note:

match statement is used in python, to execute a particular logic based on the given value instead of condition this statement will be used "menu-driven program" using match statement, we can able to execute any one case based on the given choice value suppose, if the given is value not matches any case in the match statement,it will simply execute "case _" case in the match statement in match statement, case _ is called as "Default case"

example:

```
choice=int(input("enter the choice value:"))

match choice:

   case 1:print("you given choice 1")

   case 2:print("you given choice 2")

   case 3:print("you given choice 3")

   case 4:print("you given choice 4")

   case 5:print("you given choice 5")

   case _:print("invalid choice")
```

5.nested conditional statements:

when we write the conditional statements inside another conditional statement, this scenario we can say "nested conditional statements"

syntax:

```
if condition:

  if condition:

        if condition:

        else:

             if condition:

             else:

                 if condition:
```

example:

```
age=int(input("Enter the age:"))
```

```
if age>18:
   if age>18 and age<=30:
      print("enjoy the bacbhelors life")
   elif age>30 and age<=40:
      print("right age ,but check with uncles before marriage")
   elif age>40 and age<=50:
      print("already you are in heaven,enjoy the rest")
   elif age>50 and age<=70:
      print("you are already to hit the bucket")
   elif age>70:
      print("RIP")
else:
   print("it is not a video game")
```

2.un-condition-based or un-conditional statements:

in python,we will have the following un-conditional statements:

1.break:

this is only used inside the loop and using this we can break the execution of the loop or stop the loop execution

2.continue:

this is only used inside the loop and using this we can stop the current iteration of the loop execution

3.return :

this is used only inside the function and using this we can return the result from the function in python

9.python looping/iterative statements:

these statements are used to "Execute a logic for "n" of times until the condition become false" in python, we will have two types of loops:

1.while loop:

this loop we be created in the python, based on the condition that is reason in python "while loop is often called as "conditional loop" syntax for while loop in Python:

```
while condition:
     #logic
```

example:

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

```
    a+=1
print()
print(a)#11
```

example-2:

```
a=1
times=0
while a<=10:
    a+=3
    times+=1
    a+=2
print()
print("no of times is executed:",times)#2
print("the value of a is:",a)#11
```

example-3:

```
a=1
times=0
while a<=10:
    if a>=2 and a<=7:
        times+=1
print()
print("no of times is executed:",times)
print("the value of a is:",a)
"""
output:
    infinite loop
"""
```

example-4:

```
a=1
times=0
while a<=10:
    a+=2
    if a>=2 and a<=7:
        times+=1
    a+=1
print()
print("no of times is executed:",times)
print("the value of a is:",a)
"""
```

output:

```
    no of times is executed: 2
    the value of a is: 13
```

example-5:

```
a=1
times=0
while a<=10:
    a+=2
    if a>=2 and a<=7:
        times+=1
    a+=1
print()
print("no of times is executed:",times)
print("the value of a is:",a)
```

output:

```
    no of times is executed: 2

    the value of a is: 13
```

example-6:

```
a=1
times=0
while a<=10:
    a+=4
    if a>=2 and a<=10:
        times+=1
    a-=1
print()
print("no of times is executed:",times)
print("the value of a is:",a)
```

output:

```
    no of times is executed: 2

    the value of a is: 13
```

example-7:

```
a=1
times=0
while a<=10:
    a-=5
    if a<=1 and a<=10:
        times+=1
    a+=2

print()
print("no of times is executed:",times)
print("the value of a is:",a)
```

output:

```
    infinite loop
```

example-8:

nested loop/inner loop:

nested loop means "creating a loop inside another loop" using while loop, "we can able to create the nested loop"

syntax:

```
while condition:

       while condition:

            #logic <=== it is logic inner loop

       #logic <==== it is logic of outer loop
```

note:

when a nested/inner loop is executing ,the outer loop logic will never executed until the inner/nested loop execution complete

example-1:

```
a=1
b=1
times=0
while a<=10:
   while b<=5:
       times+=1
       b+=1
   a+=1
 print()
print("no of times is executed:",times)
print("the value of a is:",a)
print("the value of a is:",b)
"""
```

output:

```
 no of times is executed: 5
the value of a is: 11
the value of a is: 6
```

example-2:

```
a=1
b=1
times=0
while a<=10:
   while b<=5 and a<=5:
       times+=1
       b+=1
   a+=1
   b=1

print()
print("no of times is executed:",times)
print("the value of a is:",a)
print("the value of b is:",b)
```

output:

```
no of times is executed: 25
the value of a is: 11
the value of a is: 1
```

example-3:

```
a=1
b=1
times=0
while a<=10:
   while b<=5 and a<=5:
       if b==3 or b==5:
          times+=1
       b+=1
   a+=1
print()
print("no of times is executed:",times)
print("the value of a is:",a)
print("the value of b is:",b)
```

output:

```
    no of times is executed: 2
    the value of a is: 11
    the value of b is: 6
```

example-4:

```
a=1
b=1
times=0
while a<=10:
   while b<=10 and a%2==0:
          times+=1
          b+=1
   a+=1
   b=3
print()
print("no of times is executed:",times)
print("the value of a is:",a)
print("the value of b is:",b)
```

output:

```
   no of times is executed: 40
   the value of a is: 11
   the value of b is: 3
```

example-5:

```
a=1
b=1
times=0
while a<=10:
   while b<=10 and a!=(a+2):
          times+=1
          b+=1
   a+=1
   b=1
```

```
print()
print("no of times is executed:",times)
print("the value of a is:",a)
print("the value of b is:",b)
```

output:

```
 no of times is executed: 100
 the value of a is: 11
 the value of b is: 1
```

example-5:

```
a=1
b=1
c=1
times=0
while a<=10:
   while b<=10 and a<=5:
          times+=1#1
          while c<=5:
              times+=1#6
              c+=1
   a+=1
   b=1
print()
print("no of times is executed:",times)
print("the value of a is:",a)
print("the value of b is:",b)
```

output:

```
infinite loop
```

example-6:

```
a=1
b=1
c=1
times=0
while a<=10:
   while b<=10 and a<=5:
          times+=1#10
          while c<=5:
            times+=1#6
            c+=1
         b+=2
   a+=1
   b+=4
print()
print("no of times is executed:",times)
print("the value of a is:",a)
print("the value of b is:",b)
```

output:

```
no of times is executed: 10
the value of a is: 11
the value of b is: 51
```

example-7:

```
a=1
b=1
c=1
times=0
while a<=10:
   while b<=10 and a<=5:
         times+=1
         while c<=5:
            times+=1
            c+=1
         b+=2
   a=1
   b=4
   c=2
print()
print("no of times is executed:",times)
print("the value of a is:",a)
print("the value of b is:",b)
```

output:

```
infinite loop
```

example-8:

```
a=1
b=1
c=1
times=0
while a<=10:
  while b<=10 and a<=5:
        times+=1
        while c<=5:
            times+=1
            c+=1
        b+=2
  a+=1
  b=4
  c=2
 print()
print("no of times is executed:",times)
print("the value of a is:",a)
print("the value of b is:",b)
```
output:

```
no of times is executed: 42
the value of a is: 11
the value of b is: 4
```
2.for loop:

for loop is a loop in python and this loop will work only "iterables" in python in python, the iterabes are :

1.list

2.tuple

3.string

4.set

5.dictionary

6.range(),................................

to work with for loop, we are going to use "range()" range() is a "built-in function" and it used to generate the values from given range(start,end) syntax for range() function in python:

range(start,end,step)

in python range() function:

the default value for start is "0"

the default value for step is "1"

when we are working with range() function ,minimum we need to give one value as "end" value
range() will generate the values from "start to end-1" in range() function :
step value never be zero,can be +ve/-ve
start value can be "+ve/-ve"
end value can be "+ve /-ve" in python range() function can have also "indexing" in python
range() function, we can have two types of indexing:

1.positive indexing(the positive indexing will start from 0 to length-1) this indexing will start from start to end or left to right 'direction this is indexing is also called as "forward indexing" here length refers to "number of values/elements in the range" to find the length of the range(), we will use "len()" function in python syntax to access the range() value using indexing:
===
range_object_name[index]
2.negative indexing(the negative indexing will start from 1 to -(length)) this indexing will start from end to start or right to left direction this is indexing is also called as "backward indexing"
note:
the possible indexes of a range object is always lies in between "-length of the object to length of the range object-"

(or)

"-len(range_object_name) to len(range_object_name)" indexing can return maximum only one value at time slicing on range() function:
=====================
in python, when we apply the slicing on the range object it will generate the "zero or more values" ,but indexing maximum can give only one value when we want to apply the slicing on the range() object, we will use following syntax:
==

range_object_name[start:end:step]

here ":" can be called as "slice operator" in python
example-1:
```
for i in range(1,10):#exclude step
    print(i,end=" ")
print()
for i in range(10):#exclude start,step
    print(i,end=" ")
print()
#for i in range():#exclude start,end,step
    #print(i,end=" ")
result=range(1,10)
print(result,type(result))
```

example-2:

```
for i in range(1,10,2):
    print(i,end=" ")#1,3,5,7,9
print()
for i in range(1,20,5):
    print(i,end=" ")#1,6,11,16
print()
for i in range(1,30,10):
    print(i,end=" ")#1,11,21
```

example-3:

```
for i in range(-5,10,4):
    print(i,end=" ")#-5,-1,3,7
print()
for i in range(5,5,1):
    print(i,end=" ")#
print()
for i in range(10,5,1):
    print(i,end=" ")
print()
#for i in range(1,10,0):#ValueError
    #print(i,end=" ")
```

example-4:

```
##program to work "for with range() function"
for i in range(10,1,-1):
    print(i,end=" ")#
print()
```

```
for i in range(10,-2,-1):
    print(i,end=" ")
print()
for i in range(10,0,-1):
    print(i,end=" ")#
print()
```

example-5:

```
##program to work "for with range() function"
a=range(1,10)
length=len(a)
print("the length of the range is:",length)
print(a[0])#1
print(a[5])#6
print(a[8])#9
print(a[-4])#6
print(a[-9])#1
#print(a[-10])
```

example-6:

```
#program to work "for with range() function"
a=range(1,10)
length=len(a)
print("the length of the range is:",length)
#slicing on range() object called "a"
print(a[1:5])#1 to 4
print(a[1:7])#1 to 6
b=range(10,100,10)
print(b[3:9])#range(40,100,10)
print(b[3:6])#range(40,70,10)
```

example-7:

```
program to work "for with range() function"
a=range(1,10)
length=len(a)
print("the length of the range is:",length)
#slicing on range() object called "a"
print(a[1:4])
print(a[1:1000000])
print(a[2:8])
print(a[10:20])
print(a[10:1])
print(a[103:104])
```

eample-8:

```
a=range(1,10)
length=len(a)
print("the length of the range is:",length)
#slicing on range() object called "a"
print(a[1:])#range(2,10)
print(a[:9])#range(1,10)
print(a[:])#range(1,10)
print(a[1:10:3])#range(2,10,3)
print(a[1:10:4])#range(2,10,4)
print(a[3:8:3])#range(4,9,3)
print(a[3:10:4])#range(4,10,4)
print(a[1:8:2])#range(2,9,2)
```

example-9:

```
a=range(1,10)
length=len(a)
print("the length of the range is:",length)
#slicing on range() object called "a"
print(a[-8:-1])#range(2,9)
print(a[-4:0])#range(6,1)
print(a[-5:-2])#range(5,8)
print(a[-10:0])#range(1,1)
print(a[-6:])#range(4,1)
print(a[:-1])#range(1,9)
print(a[:-3])#range(1,7)
print(a[:-2])#range(1,8)
print(a[-9:])#range(1,10)
```

example-10:

```
a=range(1,10)
length=len(a)
print("the length of the range is:",length)
#slicing on range() object called "a"
print(a[-8:-1:2])#range(2,9,2)
print(a[::3])#range(1,10,3)
print(a[3::3])#range(4,10,3)
print(a[::-1])#range(9,1,-1)
print(a[::-2])#range(9,1,-2)
print(a[::-4])#range(9,0,-4)
print(a[:7:-2])#range(9,8,-2)
print(a[:4:-3])#range(9,5,-3)
```

programming with python:
=====================

1.write a python program "multiply the given number with

2 without using "*" operator:
==

code:
====

```
"""
author:Ram
python program "multiply the given number with 2 without using "*" operator
"""
a=int(input("enter the number:"))
print(a+a)#20
#or
print(a-(-a))#20
#or
print(a<<1)#20
```

2.write a python program to find the remainder of the given two numbers without using "%" operator:

```
a=int(input("enter the number1:"))#14
b=int(input("enter the number2:"))#5
if a<b:
    print("remainder:",a)
else:
    print("remainder:",(a-((a//b)*b)))
```

3.write a python program to find the maximum number for given three numbers:

```
a=int(input("Enter the number1:"))
```

```
b=int(input("Enter the number2:"))
c=int(input("Enter the number3:"))
if a>b and a>c:
    print("maximum:",a)
elif b>c:
    print("Maximum:",b)
else:
    print("Maximum:",c)
```

4.write a python program to display the minimum number from given three numbers:

```
a=int(input("Enter the number1:"))
b=int(input("Enter the number2:"))
c=int(input("Enter the number3:"))
if a<b and a<c:
    print("minimum:",a)
elif b<c:
    print("minimum:",b)
else:
    print("minimum:",c)
```

5.write a python program swap the given two numbers without using any third variable:

```
a=int(input("Enter the number1:"))
b=int(input("Enter the number2:"))
print("a:",a,"b:",b)
a,b=b,a
print("a:",a,"b:",b)
```

6.write a python program to print the even numbers from 1 to 100:

```
start=1
while start<=100:
    if start%2==0:
        print(start,end=" ")
    start+=1
print()
for i in range(1,101):
    if i%2==0:print(i,end=" ")
```

7.write a python program to print the maximum even number for given range:

```
start=int(input("Enter the start:"))#1
end=int(input("Enter the end:"))#10
if start<end:
    start=end
max=0
times=0
while start>=1:
    if start%2==0:
        if start>max:
            max=start
            break
    start-=1
    times+=1
print(max,times)
```

8.write a python program to print the minimum even number for given range:

```
start=int(input("Enter the start:"))#1
end=int(input("Enter the end:"))#10
if start>end:
    start=end
min=end
times=0
while start<=end:
    if start%2==0:
        if start<min:
            min=start
            break
    start+=1
    times+=1
print(min,times)
```

9.write a python program to count the how many even numbers for given range:

10.write a python program to find sum of the even numbers for given range:

```
start=int(input("enter the start:"))
end=int(input("enter the end:"))
sum=0
if start>end:
    start,end=end,start
for i in range(start,end+1):
    if i%2==0:
        sum+=i
print("the sum of the even numbers for given range:",sum)
```

11.write a python program to display the "digits of the given number":

```
==============================================
123 ===>1,2,3
12345 ===>1,2,3,4,5
code:
====
num=input("Enter the number:")#1234
for i in num:#1234
    print(i)
```

12.write a python program to find the sum of the digits of the

```
#given number:
#1234====>1+2+3+4====>sum=10
num=input("Enter the number:")#4567
sum=0
for i in num:#4567
   sum+=int(i)
print("sum of the digits of the given number:", sum)
```

13. program to find the maximum digit in the given number

```
5674 ===>maximum digit ===>7
1234 ===>maximum digit ===>4
code:
num=input("Enter the number:")#4567
max=0
for i in num:#4567
   if max<int(i):
       max=int(i)
print("maximum digit of the given number:", max)
```

14.program to find the minimum digit in the given number:

```
"""
program to minimum digit of the given number
"""
num=input("Enter the number:")#4567
min=9
for i in num:#4567
   if min>int(i):
       min=int(i)
print("minimum digit of the given number:", min)
```

15.program to average of the digits of the given number:

```
1234 ===>number of the digits and sum of the digits

                1+2+3+4=10,no.of.digits=4==>avg=10//4=2
"""
program to average of the digits of the given number
"""
num=input("Enter the number:")#4567
sum=0
for i in num:#4567
  sum+=int(i)
print("average of digits of the given number:", sum/len(num))
```

16.count the number of digits in the given number:

```
num=input("Enter the number:")
print("count the number of digits in the given number:",len(num))
```

17.python program remove the duplicate digits in the given number:

example:

```
1122338 ===>1238
7884456===>78456
```

code:

```
num=input("Enter the number:")
res=''
for i in num:
    if i not in res:
        res+=i
print("number:",res)
```

18.print the even digits of the given number
19.print the sum of the even digits of the given number
20.print the maximum even digits of the given number
21.print the maximum odd digit of the given number
22.print the minimum even digits of the given number
23.print the average of the even digits of the given number
24.print the odd digits of the given number
25.print the sum of the odd digits of the given number
26.print the prime digits of the given number

===

123768===>2,3,7'

code:

=====

```
num=input("Enter the number:")
for i in num:
    if i=='2' or i=='3' or i=='5' or i=='7':
        print(i)
print()
for i in num:
    if i in ['2','3','5','7']:
        print(i)
print()
for i in num:
    if i in "2357":
        print(i)
print()
for i in num:
    if int(i) in (2,3,5,7):
        print(i)
```

27.count number of prime digits in the given number
28.print the maximum prime digits in the given number
29.print the minimum prime digit in the given number
30.print the perfect digit in the given number:

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

example:

=======

16578 ====>6(1+2+3)===>6

code:

=====

```
"""
program to print perfect digits in the given number
"""
num=input("Enter the number:")
for i in num:
    if i=='6':
        print(i)
print()
```

31.print the reverse of the given number:

123 ====>321
1234====>4321

```
num=input("Enter the number:")
print("reverse of the given number:",num[::-1])
a=1234
print(str(a)[::-1])
```

32.check given number is palindrome or not:

121===>121(palindrome)
123===>321(not a palindrome)
code:

```
num=input("Enter the number:")
print("palindrome") if num==num[::-1] else print("not a palindrome")
```

33.print the factors of the given number:

12 ===>1,2,3,4,6
24===>1,2,3,4,6,8,12

code:

```
num=int(input("Enter the number:"))#24
fact=1
times=1
while fact<=(num//2):
    if num%fact==0:
        print(fact)
    fact+=1
    times+=1
print("no of times:",times)
```

34.count the number of factors of the given number:

12===>1,2,3,4,6===>count=5
36===>1,2,3,4,6,9,12,18===>count=8

code:

```
num=int(input("Enter the number:"))#24
fact=1
count=0
while fact<=(num//2):
    if num%fact==0:
      count+=1
    fact+=1
print("the no of factors of the given num:",count)
```

35.find the maximum factor of the given number:

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

36 ===>maximum factor=18

25===>maximum factor=5

96===>maximum factor=48

code:

```
num=int(input("Enter the number:"))#24
fact=(num//2)
times=0
while fact<=(num//2):
    if num%fact==0:
        print(fact)
        break
    fact-=1
    times+=1
print(times)
```

36.check given number is perfect number or not:

6===>1+2+6

28===>1+2+4+7+14=28

code:

```
"""
program to find maximum factor of the given number
"""
num=int(input("Enter the number:"))#24
fact=1
sum=0
while fact<=(num//2):
    if num%fact==0:
       sum+=fact
    fact+=1
print("perfect") if num==sum else print("Not a perfect")
```

37.find the gcd of the given two numbers

(Or)

find the hcf of the given two numbers:

12,24===>12

12,54===>6

27,81===>27

```
num1=int(input("Enter the number1:"))
num2=int(input("Enter the number2:"))
fact=num1 if num1<num2 else num2
times=0
while fact!=0:
    if num1%fact==0 and num2%fact==0:
        print(fact)
        break
    fact-=1
    times+=1
print(times)
```

38.find the lcm of the given two numbers:

3,6===>6
8,10===>40
5,12===>60

```
num1=int(input("Enter the number1:"))
num2=int(input("Enter the number2:"))
fact=num1 if num1<num2 else num2
while fact!=0:
    if num1%fact==0 and num2%fact==0:
        print((num1*num2)//fact)
        break
    fact-=1
```

39.check given number is prime or not:

2 ===>1,2
3===>1,3
29===>1,29
31===>1,31
36===>1,2,3,4,6,9,12,18,36,............

code:

```
num=int(input("Enter the number:"))
fact=1
count=0
```

```
while fact<=num:
    if num%fact==0:count+=1
    fact+=1
print("prime number") if count==2 else print("not a prime number")
```

for with else block:
when we write any else block along with "for loop", the else will always executed once the loop is end normally, but if the loop is terminated due to "break", then the else will not executed
example:

```
for i in range(1,10):
    print(i,end=" ")
else:
    print("i am always executed after loop execution")
print()
for i in range(1,10):
    if i==5:
        break
    print(i,end=" ")
else:
    print("i am always executed after loop execution")
```

code-2:(for prime number)

```
"""
program to check given number is prime or not
"""
num=int(input("enter the number:"))
for i in range(2,num//2):
    if num%i==0:
        print("not a prime")
        break
else:
    print("prime")
```

40.check given number is perfect number or not:

6===>1,2,3   ===>1+2+3
28===>1,2,4,7,14===>1+2+4+7+14=28
code:

```
num=int(input("enter the number:"))
sum=0
for i in range(1,(num//2)+1):
    if num%i==0:
        sum+=i
else:
  print("perfect") if sum==num else print("not a perfect")
```

41.check given number is Armstrong number or not:

example:

step-1: find the number length(use this as power)

step-2:calcuate the each digit with power sum

code:

```
num=int(input("enter the number:"))#153
sum=0
for i in str(num):
    sum+=int(i)**len(str(num))
print("armstrong") if sum==num else print("not a armstrong")
```

42.find the factorial of the given number:

5!===>120(5*4*3*2*1)

-2!===>2(not possible)

0!===>1

1!===>1

code:

```
num=int(input("enter the number:"))#153
fact=1
if num>=1:
  for i in range(1,num+1):
    fact*=i
  print(fact)
elif num==0:
    print(fact)
else:
    print("please give positive integer")
```

43.check given number is strong number or not:

145===>1!+4!+5!=1+24+120=145

code:

```
num=int(input("Enter the number:"))
fact=1
sum=0
```

```
for i in str(num):
    #factorial of the each digit
    for j in range(1,int(i)+1):
        fact*=j
    #sum of factorials of the each digit
    sum+=fact
    fact=1#for every new digit,fact has to start from 1 onwards
print("strong") if sum==num else print("not a strong")
```

44.print the maximum prime number for given range:

1 to 10 ====>7
20 to 30===>19
90 to 100===>97

code:

```
start=int(input("start:"))
end=int(input("end:"))
if start>end:start,end=end,start
count=0
for i in range(end,start,-1):
    for j in range(1,i+1):
        if i%j==0:
            count+=1
    if count==2:
        print(i)
        break
    count=0
```

45.print the even numbers for given range:

1 to 10 ====>2,4,6,8,10
24 to 30====>24,26,28,30

code:

```
start=int(input("start:"))
end=int(input("end:"))
if start>end:start,end=end,start
count=0
for i in range(start,end+1):
    if i%2==0:
        print(i,end=" ")
```

46.print the prime numbers for the given range:

code:

```
start=int(input("start:"))
end=int(input("end:"))
```

```
if start>end:start,end=end,start
count=0
for i in range(start,end+1):
    for j in range(2,i):
        if i%j==0:
            break
    else:
        if i!=1:
          print(i)
```

47.print the Armstrong numbers between the given range:

1 to 10 ===>1,2,3,4,5,6,7,8,9

code:

```
start=int(input("start:"))
end=int(input("end:"))
if start>end:start,end=end,start
sum=0
for i in range(start,end+1):
    for j in str(i):
      sum+=int(j)**len(str(i))
    if sum==i:
        print(i)
    sum=0
```

48.print the perfect numbers for given range:

code:

```
start=int(input("start:"))
end=int(input("end:"))
if start>end:start,end=end,start
sum=0
for i in range(start,end+1):
    for j in range(1,i):
        if i%j==0:
            sum+=j
    else:
        if sum==i:
            print(i)
    sum=0
```

49.program to print the strong numbers for given range:

code:

problems on patterns:

50. write a python program to print the following pattern:

```
1
12
123
1234
12345
```

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row+1):
        print(data,end="")
    print()
```

51.write a python program to print the following pattern:

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

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row+1):
        print(data+1,end="")
    print()
```

52.write a python program to display the following patteren:

```
0
2 4
6 8 10
12 14 16 18
20 22 24 26 28
```

code:

```
rows=int(input("enter the rows:"))
k=0
for row in range(1,rows+1):
    for data in range(1,row+1):
        print(k,end=" ")
        k+=2
    print()
```

53:write a python program to display the following pattern:

```
1
1 4
1 4 9
1 4 9 16
1 4 9 16 25
```
code:
```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row+1):
        print(data**2,end=" ")
    print()
```
54:write a python program to display the following patteren:
```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```
code:
```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row+1):
        print(row,end=" ")
    print()
```

55:write a python program to display the following pattern:
```
*
* *
* * *
* * * *
* * * * *
```
code:
```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row+1):
        print("*",end=" ")
    print()
```

(or)
```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
   print("* "*row)
```

56:write a python program to display the following patteren:

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

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    k=row
    for data in range(1,row+1):
        print(k,end=" ")
        k=k-1
    print()
```

57:write a python program to display the following patteren:

```
1
11
1 2 1
1 2 3 1
1 2 3 4 1
```

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row):
        print(data,end="")
    print("1",end="")
    print()
```

58:write a python program to display the following patteren:

```
12
1223
122334
12233445
```

```
1223344556
```
code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row+1):
        print(data,data+1,sep="" ,end="")
    print()
```

59:write a python program to display the following pattern:

```
111
111842
1118422793
111824279364164
1118242793641641252 55
```

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row+1):
        print(data**3,data**2,data,sep="" ,end="")
    print()
```

60:write a python program to display the following pattern:

```
*
1*
12*
123*
1234*
```

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row):
        print(data,end="")
    print("*",end="")
    print()
```

61:write a python program to display the following pattern:

```
1
*2
**3
***4
****5
```

code:
```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,row+1):
        if row==data:
            print(row,end="")
        else:
            print("*",end="")
    print()
```

62:write a python program to display the following pattern:
```
12345
1234
123
12
1
```
code:
```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,(rows-row)+2):
        print(data,end="")
    print()
```
63:write a python program to display the following pattern:
```
*****
****
***
**
*
```
code:
```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,(rows-row)+2):
        print("*",end="")
    print()
```
(or)

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
  print("*"*((rows-row)+1))
```

64:write a python program to display the following pattern:

```
2516941
16941
941
41
1
```

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range((rows-row)+1,0,-1):
        print(data**2,end="")
    print()
```

65:write a python program to display the following pattern:

```
1491625
14916
149
14
1
```

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    for data in range(1,(rows-row)+2):
        print(data**2,end="")
    print()
```

66.write a python program to display the following pattern:

```
1
12
123
1234
12345
1234
123
12
1
```

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    if row<=((rows//2)+1):
     for data in range(1,row+1):
        print(data,end="")
    else:
        for data in range(1,(rows-row)+2):
          print(data,end="")
    print()
```

67.write a python program to display the following pattern:

```
1
12
123
1234
12345
****
***
**
*
```

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    if row<=((rows//2)+1):
     for data in range(1,row+1):
        print(data,end="")
    else:
        for data in range(1,(rows-row)+2):
          print("*",end="")
    print()
```

68.write a python program to display the following pattern:

```
54321
4321
321
21
1
12
123
1234
12345
```

code:

```
rows=int(input("enter the rows:"))
for row in range(1,rows+1):
    if row<=((rows//2)+1):
     for data in range(((rows//2)+1)-(row-1),0,-1):
        print(data,end="")
    else:
        for data in range(1,(row-4)+1):
          print(data,end="")
    print()
```

working with continue in Python:

continue will "skip the current of the loop" in python

example-1:

```
i=0
while i<10:
    i+=1
    if i==5:
        continue
    print(i)
```

example-2:

```
for i in range(1,10):
    if i==5:
        continue
    print(i)
```

example-3:

```
times=0
for i in range(1,10):
  for j in range(1,5):
      if j==3:
          continue
      for k in range(1,5):
          if k==4:
              continue
          times+=1
      times+=1
  times+=1
print(times)
```

example-4:

```
times=1
```

```
for i in range(1,10):
  if i>5:
      continue
  else:
      for j in range(1,5):
          if i==3:
              continue
          else:
              times+=1
              for k in  range(1,5):
                  if j==3:
                      continue
                  else:
                      times+=1
  times+=1

print(times)
```

working with warlus operator in python:
walrus operator is a operator and this will "to do both assignment and comparison at a time"
symbol:
example-1:

```
a=10
if (a:=10+10)==20:
    print("the value of a is 20")
    print(a)
else:
    print("the value of a is not 20")
result=(a:=a+100)+200
print(result)#320
print(a)#120
a=0
while (a:=a+1)<=10:
    print(a)
print(a)
```

10.python functions(both user-defined and built-in functions):
function:
function is a block of code or statements, which are used to perform a specified task using functions, we can eliminate the "duplicate code or reduant code from the program" using

functions, we can divide the "entire program into "n" number of fucntions" using functions, we can enhance any code /logic in the program so easy.to create the function, in python we will use the following

syntax:

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

to create any function, we will use "def" is keyword in python any function may take the data as "arguments" or arguments of the function is nothing but "data to the function"when we create the function in python, the function code will be executed, when we call the function(function calling is nothing but executing the function) without calling the function, function will not be executed, even we create the function in python program function is always executed, only when we call the function to call the function in python, we will use the following syntax:

==

function_name(arg1,arg2,arg3....argn)

note:

while calling,the arguments in the function will always depends on the arguments in the function(if the function have arguments, then we will pass arguments while calling, otherwise we will not pass arguments)

example:

```
def display():#in this no arguments
   print("this is my first function in python")
#call the display() function
display()
```

ways to create the function in python:

in python, we create a function in 4 ways:

function without arguments and without return type:

in this model, function is not going take any arguments and function is not going to return any result

example:

```
#create a function without arguments and return type
def add_two_numbers():
   a=int(input("enter the value for a:"))
   b=int(input("enter the value for b:"))
   print(a+b)
#call the function
add_two_numbers()
```

2.function with arguments and without return type:

In this model, function is going take arguments and function is not going to return any result

example:

```
#create a function with arguments and return type
def add_two_numbers(x,y):#x=a,y=b
    print(x+y)
#call the function
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
add_two_numbers(a,b)
```

3.function with arguments and with return type:

in this model, function is going take arguments and function is not going to return any result when the function is executed, the result of the function can able to return from the function to function calling via "return" keyword every can able to send the result to function calling via "return" keyword when say "a function is return a value", then the function is having "return statement" when we are using "return" statement , it always "last statement" in any function(it is always optional) in the function, return can able to return more than one value also (it means return can return "multiple values")

example:

```
#create a function with arguments and return type
def add_two_numbers(x,y):#x=a,y=b
    print("the value of x is:",x)
    print("the value of y is:",y)
    return x+y
#call the function
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
result=add_two_numbers(a,b)
print(result)
```

4.function without arguments and with return type:

in this model, function is not going take any arguments and function is going to return result

example:

```
#create a function without arguments and return type
def add_two_numbers():
    a=int(input("enter the value for a:"))
    b=int(input("enter the value for b:"))
    return a+b
#call the function
print(add_two_numbers())
```

types of arguments in python function:
In python functions, we will have the following different types of arguments:
1.positional arguments:
positional arguments are "arguments of the function, these arguments will take the data ,based on the position of the arguments in the function call"

example:

```
#create a function with positional arguments
def display(x,y,z):
    print(x)
    print(y)
    print(z)
display(10,20,30)
display(30,20,10)#x=30,y=20,z=10
display(100,200,300)
```

2.default arguments:
Default arguments are arguments in the function and these arguments are define with some value while creating the function itself if any default argument is in the function, then while calling the function we can give value or even we can not give the also because, when we not given any value the function simply takes default value.

example:

```
#create a function with default arguments
def display(x=10,y=20,z=30):
    print(x)
    print(y)
    print(z)
display()
display(100)#x=100
display(100,200)#x=100,y=200
display(100,200,300)#x=100,y=200,z=300
```

3.keyword arguments:

keyword arguments are arguments in the function, these arguments will take the data from the function call, based on name instead of position

example:

```
#create a function with keyword arguments
def display(x,y,z):
    print(x)
    print(y)
    print(z)
display(z=10,x=20,y=30)
display(x=100,z=300,y=200)
display(x=100,z=500,y=200)
```

4.var-length arguments:

These arguments are again divided into twp types:

1.arbitary arguments(*):

it make function can able take to "multiple values from the function call into single argument(arbitrary argument) the arbitrary argument we define, with help of "*" in the function all arbitrary arguments will taken function as "tuple" format

example:

```
#create a function with arbitary arguments
def display(*x):
    print(x)
    print(type(x))
display()
display(10,20,30)
display(10,20,30,40,50,60,70)
display(1,2,3,4,5,6,7,8,9,10,11,12,13)
```

2.keyword arbitrary arguments(**):

when we define the arguments in the function, as "keyword arbitrary arguments", the arguments we will defined with help of "**" and with help of keyword arbitrary arguments we can any number arguments to the function as "key and value" format, where key is act like a keyword arguments the keyword arbitrary arguments taken by the function as "dictionary" format.

example:

```
#create a function with keyword arbitary arguments
def display(**x):
    print(x)
    print(type(x))
display()
display(name="ram",location="hyderabad")
display(a=1,b=2,c=3,d=4,e=5)
```

lambda functions in python:
lambda function is a function, which is going to take "any" number of arguments and this will have only one line as "Code" lambda function is also called as "anonymous function" lambda is not having "any name" like normal function to create the lambda function in python, we will use a keyword called "lambda" lambda function is also called as "single line function" in Python lambda function is also termed as " lambda expression"

syntax for lambda function in python:
lambda arg1,arg2,arg3,....argn: code for lambda function in order to call the lambda, we will keep the entire lambda function in a "name" like a "value in variable". using that name ,we can execute the "lambda function" when we write any code except print statement, then lambda behave like "function with return type"

example:

```
result=lambda: print("this is first lambda fucntion")
print(type(result))
result()
result=lambda x: print(x)
result(10)
result=lambda x,y: x**y #2**3
print(result(2,3))
result=lambda x,y,z:x+y+z
print(result(10,20,30))
```

writing the lambda function with "normal function"
example-1:

```
def display(x):
    return lambda y:x*y
result=display(100)
```

```
print(type(result))
print(result(20))
```

example-2:

```
def display(x,y):
    return x*y
result=lambda y:y*display(int(input("x:")),int(input("y:")))
print(result(int(input("give value for lambda y:"))))
```

recursion

recursion means "Calling by itself" when a function is calling by itself, then we say the function is "recursive function"

example:

```
def display(x):
    if x<=5:
        print(x)
        x=x+1
        display(x)
display(1)
```

write a python program to find the sum of the number 1 to for given number using recursion:

for example,

n=10 ===>1+2+3+4+5+6+7+8+9+10=55

code:

```
def display(num,sum):
    if num!=0 and num>0:
        sum=sum+num
        num=num-1
        display(num,sum)
    else:
        print(sum)
num=int(input("num:"))
sum=0
display(num,sum)
```

write a python program to print the even number from 2 to given number using recursion:

10 ===>2,4,6,8,10

====

```
def display(start,num):
    if start<=num:
        print(start)
        start=start+2
        display(start,num)
num=int(input("num:"))
start=2
display(start,num)
```

write a python program to print the prime number from 2 to given number:

2 to 10 ===> 2,3,5,7

2 to 20===>2,3,5,7,11,13,17,19

code:

```
def outer(start,num):
    def check_prime(start,fact):
        #logic
        for i in range(1,start+1):
            if start%i==0:
                fact+=1
        if fact==2:print(start)
        start+=1
        if start<=num:
            check_prime(start,0)
    check_prime(start,0)

num=int(input("enter the num:"))
outer(2,num)
```

inner functions

inner function is a function, which is create inside another function inner function is also called as "nested function" inner function is also termed as "closures" in python when we create inner functions, to execute inner function we need to call the inner function inside the outer function, but we can not call the "inner function" outside the "outer" function when we are working with inner functions, the inner function can access the "outer function" data, but not vice versa (the outer function can not access inner function data)outer function can access only "it's data" or "outer function data" only in a outer function ,we can create any number of "inner functions".

example:

```
def outer():
    def inner():
        print("this is my first inner function")
    inner()
```

```
outer()
```

example:

```
def outer(a,b):#outer function data a,b
    def inner(x,y):#inner function data x,y
        print("this is my first inner function")
        print("a:",a)
        print("b:",b)
        print("x",x)
        print("y:",y)
    inner(100,200)
outer(10,20)
```

example:

```
def outer():
    def inner1():
        print("inner function-1")
    def inner2():
        inner1()
        print("inner function-2")
    def inner3():
        inner2()
        print("inner function-3")
    inner3()
outer()
```

python scope
scope refers to "where we can able to access an data(variable)" in python, we will have two types of scopes:
1.global scope:
when we define any data outside the function, the data will gets global scope any global scope data, can access any where in the the program, but we can not able to manipulate in any function, but we can manipulate any where outside function in the entire program
2.local scope:
when we define any data inside the function, the data will gets local scope any local scope data, we can access any where in the function but not in outside the function and it's manipulation can be done only where we define.

example:

```
x=100 #global
def display():
  y=200 #local data
  print(x) # <== x is global data
  y=x+100
  print(y)# <== y is local data,y=200
display()
print(x)
#print(y)  #<== here y is local data
x+=100
print(x)#<==x=x+100=100+100=200
```

how to make any local data into global data in python:
to make any local data into global data in python, we will use a keyword called "global" keyword when we use global "on" any data, we just we give only name not with any value

syntax:
global local_vairable_name
note:
when we assign any data, while using global keyword is not allowed in python.

example:
```
def display():
    global x
    x=100 #<== here x is local data
    print(x)#100
display()
print(x)#100
x+=200
def display2():
    print(x)#300
display2()
```

example-2:
```
x=10#global data
def display():
    global x
    print(x)#10
    x+=10
    print(x)#20
```

```
display()
```

example-3:

```
x=10
def display():
    global x
    print(x)#10
    #x+=30 ==>x=40
    x+=30
    print(x)#40
    def inner_display():
        global y
        y=100#local variable
        print(x)#40
        print(y)#100
    def inner_display2():
        print(y)
    inner_display()
    inner_display2()
display()
print(y) #100
```

example-4:

```
x=100#global data
print(x)#100
def display():
    global x
    print(x)#100
    x+=100#x=200
    print(x)#200
    def inner_display():
        global x
        x+=100 #x=200+100=300
        print(x)#300
    inner_display()
    x+=400
    print(x)#700
display()
```

example-5:

```
x=10#global data
print(x)#10
def display():
    x=100
    print(x)
    x+=100#x=200
    def inner_display():
        nonlocal x
        print(x)
        x+=200 #400
    inner_display()
    print(x)#400
display()
```

example-6:

```
x=100#global data
print(x)#100
def display():
    global x
    print(x)#100
    x+=100#x=200
    def inner_display():
        global x
        print(x)#200
        x+=200 #400
    inner_display()
    print(x)#400

display()
print(x)#400
```

example-7:

```
x=10#global data x=10
def display():
    x=100#local data
    print(x)
print(x)#x=10
display()
```

example-8:

```
x=10#global data x=10
```

```
def display():
    print(x)
    def inner_display():
        print(x)
    inner_display()
display()
print(x)
```

11.python modules(both user-defined and built-in modules):

module in python:

module is "a python file and which may contain data and methods(functions)" using python module, we can able to "share the one python file data/methods(functions) to another python file" using python modules "we can able to get re-useability" to create the module, in python we will use the following steps:

step-1: create a python file with some data and functions

example:

```
"""create the data here"""
x=100
y=200
z=300
"""create the methods here"""
def add(x,y):
    print(x+y)
def sub(x,y):
    print(x-y)
def mul(x,y):
    print(x*y)
```

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

example:

mymodule.py

```
"""create the data here"""
x=100
y=200
z=300
"""create the methods here"""
def add(x,y):
    print(x+y)
```

```
def sub(x,y):
    print(x-y)
def mul(x,y):
    print(x*y)
```

step-3: use the same python file in another python file as "module" , here name of the python file is itself module name.

example:

```
import  mymodule
```

step-4: to use the any python module data or functions, we will use following syntax:

```
import module_name
```

step-5: to access any module data/ function in another file ,we will use following syntax:

```
module_name. data_name
(or)
module_name.method_name/function_name
```

example:

```
import mymodule
#access the data
print(mymodule.x)
print(mymodule.y)
print(mymodule.z)
#access the fucntions
mymodule.add(10,20)
mymodule.sub(10,20)
mymodule.mul(10,20)
```

how to alias the module name in python:

to alias the module name in python, we will use a keyword called "as"

syntax:

```
import module_name as alias_name
```

example:

```
import mymodule as mm
#access the data
print(mm.x)
print(mm.y)
print(mm.z)
```

```
#access the fucntions
mm.add(10,20)
mm.sub(10,20)
mm.mul(10,20)
```

how to access the only specific data or methods from the python module:
to access the only specific data or methods from the python module, in python we will use a keyword called "from"

syntax:

```
from module_name import data_name1,data_name2,....
```

example:

```
from mymodule import x,y,mul
#access the data
print(x)
print(y)
#access the fucntions
mul(10,20)
```

note:
when we are accessing the "any data or functions" using from keyword, we no need to use "module_name" before to any data or function while using in the python program if we want to access the all data and methods of a module, using from keyword, we will use the following syntax:

from module_name import *
example:

```
from mymodule import *
#access the data
print(x)
print(y)
print(z)
#access the fucntions
add(10,20)
sub(10,20)
mul(10,20)
```

12.python packages:
package in python is a "folder and in this we can create "n" number modules" python package is a "group of modules" when we are working with any package, we need to follow the following steps:

step-1: create a folder in the system with some_name and it is act as "package name"
step-2: once we create the folder(package) in the system, in that folder, we need create an empty python file called "__init__.py"
step-3: once we create the empty python file(__init__.py), we need the all python modules here
step-4: to access the any data or method from package module we will use the following
syntax:

```
     import package_name.module_name  as alias_name
       or
    from package_name.module_name  import data1,data2...
```

example:

```
import mypackage.mymodule as mm
print(mm.x)
print(mm.y)
mm.add(10,20)
mm.sub(10,20)
mm.mul(10,20)
```

example:

```
from mypackage.mymodule import x,mul
print(x)
mul(10,20)
```

example:

```
from mypackage.mymodule import *
print(x)
mul(10,20)
```

python sub-packages:
sub-package means "create a package inside another package/ package"
how to create the sub-package in package:
step-1: create a sub-package/package inside another package
stpe-2 : in the sub-package ,create an empty python file with name "__init__.py"
step-3: create modules inside the sub-package

step-4: access the sub-package module data or functions as follows:
import package_name.sub_package_name.module_name as alias_name
example:
```
import mypackage.mysubpackage.mysubmodule as mmm
mmm.display()
```
example-2:
```
from mypackage.mysubpackage.mysubmodule import *
display()
```

python built-in functions and built-in modules:
built-in functions:
built-in function is a "function" and which given by python in python, we will have two types of function:
1.user-defined/custom function:
a function which is created by the "programmer or developer" in python, is called as "user-defined or custom function" there two types of the user-defined functions:
1.function which is created using "def" keyword
2.function which is created using "lambda"

2.built-in function:
built-in function is a "function" and which given by python the following are the most commonly used built-in functions in Python:
1.print():
it is used to "display any result or ouput of the any python program"
syntax:
```
print(data,sep=" ",end=" ")
```

example:
```
a,b,c=10,20,30
print(a,b,sep=",")#10,20
print(a,b)#10 20
print(a,b,c,end=" ")
print(a,b)
print(a,b,c,end="\n")
print(a,b)
```

2.input():
it is used to take "input from user via keyboard and it will always takes any data in the form of string"

syntax:

```
a=input("enter the number for a:")
```

example:

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

3.int():

it is used to convert the given data into integer type

example:

```
a=int(10)
print(a)
a=int(1.234)
print(a)
a=int("100")
print(a)
a=int()
print(a)
a=int("+100")
print(a)
a=int("-100")
print(a)
a=int("1_23")
print(a)
a=int("12_34")
print(a)
a=int("12.34")
print(a)
```

note:

when we give any number as "string" to the "int()" function, it allow the following characters:

+/- (at begin only)

_(between the digits only)

4.float():

it is used to convert the given data into float type

example:

```
a=float(10)
```

```
print(a)
a=float(1.234)
print(a)
a=float(+12.34)
print(a)
a=float(1_2.)
print(a)
a=float("12.34")
print(a)
a=float("+12.")
print(a)
a=float("-1_2.3_4")
print(a)
a=float()
print(a)
```

note:
when we give any number as "string" to the "float()" function, it allow the following characters:

+/- (at begin only)

_(between the digits only)

.(between the digits or at end)

5.complex():
it is used to convert the given data into complex type
example:

```
a=complex()
print(a)
a=complex(10)
print(a)
a=complex(1.234)
print(a)
a=complex(1+10j)
print(a)
a=complex(1+10J)
print(a)
a=complex(1+2j)
print(a)
```

```
a=complex("1.")
print(a)
a=complex("0.")
print(a)
a=complex("1_2.3_4")
print(a)
a=complex("12+0j")
print(a)
```

addition of two complex numbers(take input from the user):
code:

```
num1=complex(input("enter the number1:"))
num2=complex(input("enter the number2:"))
print(num1+num2)
```

subtraction of two complex numbers:

```
num1=complex(input("enter the number1:"))
num2=complex(input("enter the number2:"))
print(num1+num2)
```

multiplication of two complex numbers:

```
num1=complex(input("enter the number1:"))
num2=complex(input("enter the number2:"))
print(num1*num2)
```

example:

```
a=100
b=200
num1=complex(a,b)
print(num1)
print(num1.real)
print(num1.imag)
```

6.str():
it is used to "any data into string type"

string(data)===>string type

example:

```
a=str(10)
```

```
print(a,type(a))
a=str(10.56)
print(a,type(a))
a=str(True)
print(a,type(a))
a=str(None)
print(a,type(a))
a=str([1,2,3,4,5])
print(a,type(a))
a=str((1,2,3,4))
print(a,type(a))
a=str(set())
print(a,type(a))
a=str({1:2,3:4})
print(a,type(a))
```

7.bool():
it is used to convert the given data into "Boolean data" bool(data)===>Boolean data
example:

```
a=bool(100)
print(a)
a=bool(-100)
print(a)
a=bool(10-10)
print(a)
a=bool(None)
print(a)
a=bool('')
print(a)
a=bool([])
print(a)
a=bool(())
print(a)
a=bool({})
print(a)
a=bool(set())
print(a)
a=bool([1,2,3])
a=bool(range(5,5))
print(a)
```

```
a=bool(range(5,5))
print(a)
a=bool(-987654)
print(a)
a=list(range(5,5))
print(a)
```

when bool() will return false:
when data is "zero"
when data is "None"
when data is "Empty string"('')
when data is "empty list"
when data is "empty tuple"
when data is "empty set"
when data is "empty dictionary"
when data is "empty range()"

write a python program ,take two values from user input, take only one value which is a non-zero number only from given two numbers:
code:

```
a=int(input("a:"))
b=int(input("b:"))
print(a) if a!=0 else print(b)
```

code:

```
print(int(input("a:")) or int(input("b:")))
```

note:
in python, to represent any binary number, we will use "0b" or "0B" as prefix
in python, to represent any octal number, we will use "0o" or "0O" as prefix
in python, to represent any hexadecimal number, we will use "0x" or "0X" as prefix
every integer in python, is a decimal number

8.bin():
it is used to convert the given number into binary:

bin(data)===>binary number
===

code:

```
x=bin(0o127)#octal number into binary
print(x)
x=bin(0xabc)#hexa decimal into binary
print(x)
x=bin(100)#decimal into binary
print(x)
```

write a python program to convert the given decimal number into binary without using any built-in function:

code:

```
x=int(input("number:"))
rem=''
while x!=0:
    rem+=str(x%2)
    x=x//2
print(rem[::-1])
```

write a python program to convert the given octal number into binary without using any built-in function

code:

```
#convert given octal number into binary number
onum=int(input("Enter Octal Number:"))
res=''
temp=''
if '8' in str(onum) or '9' in str(onum):
    print("given number is not a valid octal number")
else:
    for i in str(onum):
        if i in "123":
            res+='0'
        while int(i)!=0:
            temp+=str(int(i)%2)
            i=int(i)//2
        res+=temp[::-1]
        temp=''
print(res)
```

write a python program to convert the given hexa decimal number into binary without using any built-in function

code:

```
#convert given octal number into binary number
hnum=input("Enter hexadecimal Number:")#abc
res=''
temp=''
for i in hnum:
    if i in "aA":i=10
    elif i in 'bB':i=11
    elif i in 'cC':i=12
    elif i in "dD":i=13
    elif i in 'eE':i=14
    elif i in 'fF':i=15
    elif i not in "ABCDEFabcdef" and i not in "0123456789":
        print("given number is not a valid hexa decimal number")
        break
    else:i=int(i)
    if str(i) in "1":
        res+='000'
    elif str(i) in "2 3":res+="00"
    elif str(i) in "4567":res+='0'
    while int(i)!=0:
      temp+=str(int(i)%2)
      i=int(i)//2
    res+=temp[::-1]
    temp=''
print(res)
```

9.oct():

this function will convert any given number(Decimal/binary/ hexadecimal) into octal number

oct(num)===>octal number

example:

```
#converting binary number into octal number
num=oct(0b101010)
print(num)
#converting decimal number into octal number
num=oct(129)
print(num)
#converting hexa decimal number into octal number
```

```
num=oct(0xabc)
print(num)
```

write a python program to convert the given binary number into octal number without using any built-in function:

```
#take a number from user
a=10101101
#convert the given number into decimal
pow=0
res=0
for i in str(a)[::-1]:
    res+=int(i)*(2**pow)
    pow+=1
a,res=res,''
#convert the given number into octal
while a!=0:
    res+=str(a%8) #
    print(a%8)
    a=a//8
print(res[::-1])
```

write a python program to convert the given hexadecimal number into octal number without using any built-in function:

```
#take a number from user
a='AB'
#convert the given number into decimal
pow=0
res=0
for i in a[::-1]:
    if i in "aA":i=10
    elif i in 'bB':i=11
    elif i in 'cC':i=12
    elif i in 'dD':i=13
    elif i in 'eE':i=14
    elif i in 'fF':i=15
    elif i in "0123456789":i=int(i)
    else:
        print("given number is invalid")
        res=0
        break
    res+=(i)*(16**pow)
    pow+=1
if res!=0:
    a,res=res,''
    #convert the given number into octal
    while a!=0:
      res+=str(a%8)
      a=a//8
    print(res[::-1])
```

write a python program to convert the given decimal number into octal number without using any built-in function:
code:

```
#take a number from user
a=49
#convert the given decimal number into octal
pow=0
res=''
while a!=0:
      res+=str(a%8)
      a=a//8
print(res[::-1])
```

10.hex():
this function will convert the given number into hexadecimal number
hex(number) ===>hexadecimal number

example:

```
num=hex(0b1010101)
print(num)
num=hex(0o127)
print(num)
num=hex(49)
print(num)
```

write a python program to convert the given decimal number into hexadecimal number without using any built-in function:
code:

```
#take a number from user
a=int(input("enter the number:"))
#convert the given decimal number into hexadecimal
```

```
pow=0
res=''
while a!=0:
      if a%16==10:res+='a'
      elif a%16==11:res+='b'
      elif a%16==12:res+='c'
      elif a%16==13:res+='d'
      elif a%16==14:res+='e'
      elif a%16==15:res+='f'
      else:
          res+=str(a%16)
      a=a//16
print(res[::-1])
```

write a python program to convert the given octal number into hexadecimal number without using any built-in function:
code:

```
#take a number from user
a=127
#convert the given octal number into decimal
pow=0
res=0
for i in str(a)[::-1]:
    res+=int(i)*(8**pow)
    pow+=1
print(res)
a,res=res,''
#convert the given decimal number into hexadecimal
res=''
while a!=0:
      if a%16==10:res+='a'
      elif a%16==11:res+='b'
      elif a%16==12:res+='c'
      elif a%16==13:res+='d'
      elif a%16==14:res+='e'
      elif a%16==15:res+='f'
      else:
          res+=str(a%16)
      a=a//16
print(res[::-1])
```

write a python program to convert the given binary number into hexadecimal number without using any built-in function:
code:

```
#take a number from user
a=10101011
#convert the given bianry number into decimal
pow=0
res=0
for i in str(a)[::-1]:
    res+=int(i)*(2**pow)
    pow+=1
print(res)
a,res=res,''
#convert the given decimal number into hexadecimal
res=''
while a!=0:
      if a%16==10:res+='a'
      elif a%16==11:res+='b'
      elif a%16==12:res+='c'
      elif a%16==13:res+='d'
      elif a%16==14:res+='e'
      elif a%16==15:res+='f'
      else:
          res+=str(a%16)
      a=a//16
print(res[::-1])
```

how to convert binary, octal, hexadecimal into decimal:
code:

```
#convert the given binary into decimal
print(int(0b10101))

#convert the given octal into decimal
print(int(0o127))
```

```
#convert thegiven hexadecimal into decimal
print(int(0xabc))
```

11.chr():
in python, all characters are represented in the form of "Unicode" Unicode format "to represent all native language characters" chr() will return "character(Unicode character) for given number"
chr(number)===>character
example:

```
print(chr(97))
print(chr(122))
print(chr(65))
print(chr(90))
print(chr(48))
print(chr(49))
print(chr(57))
print(chr(58))
print(chr(59))
```

12.ord() :
this function will convert the given character into Unicode number
ord(character) ===>Unicode number
example:

```
print(ord('a'))
print(ord('b'))
print(ord('A'))
print(ord('Z'))
print(ord(';'))
```

example-1:

```
A
B C
D E F
G H I J
K L M N O
```

code:
=====

```
rows=int(input("enter the number of rows:"))
charnum=65
for rownum in range(1,rows+1):
    for data in range(1,rownum+1):
        print(chr(charnum),end=" ")
        charnum+=1
    print()
```

example-2:

```
A
B
C
D
E F G H I
```

code:

```
rows=int(input("enter the number of rows:"))
charnum=65
for rownum in range(1,rows+1):
    if rownum==rows:
        for data in range(1,rownum+1):
            print(chr(charnum),end=" ")
            charnum+=1
    else:
        print(chr(charnum),end=" ")
        charnum+=1
    print()
```

exmple-3:

```
A B C D E
B
C
D
E F G H I
```

code:

```
rows=int(input("enter the number of rows:"))
charnum=65
temp=charnum#65
for rownum in range(1,rows+1):
    if rownum==rows or rownum==1:
        for data in range(1,rows+1):
            print(chr(charnum),end=" ")
            charnum+=1
    else:
        print(chr(charnum),end=" ")

    print()
    temp=temp+1
    charnum=temp
```

example-4:

```
A b C d E
b
C
d
E f G h I
```

code:

```
rows=int(input("enter the number of rows:"))
charnum=65
temp=charnum#65
for rownum in range(1,rows+1):
    if rownum==rows or rownum==1:
        for data in range(1,rows+1):
            if data%2==0:print(chr(charnum+32),end=" ")
            else:print(chr(charnum),end=" ")
            charnum+=1
    else:
        if rownum%2==0:print(chr(charnum+32),end=" ")
        else:print(chr(charnum),end=" ")

    print()
    temp=temp+1
    charnum=temp
```

example-5:

```
A  C  E  G  I

C           D

E           F

G  I  K  M  O
```

code:
```
rows=int(input("enter the number of rows: "))
charnum=65
temp=charnum
for rownum in range(1,rows+1):
    if rownum==1 or rownum==rows:
        for data in range(1,6):
            print(chr(charnum),end=" ")
            charnum+=2
    else:
        print(chr(charnum),chr(charnum+1),sep="        ",end="")
    temp=temp+2
    charnum=temp
    print()
```

example-6
```
A
  B
    C
      D
        E
```

code:
```
rows=int(input("enter the number of rows: "))
charnum=65
print()
if rows>0:
   for rownum in range(1,rows+1):
      print(((" ")*rownum)+chr(charnum))
      charnum+=1
else:
    print("rows never be and never be negative")
```

example-7:
```
*
**
***
****
*****
```
code:

```
rows=int(input("enter the number of rows: "))
for rownum in range(1,rows+1):
    print("*"*rownum)
```

example-8:

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

code:

```
rows=int(input("enter the number of rows: "))
star=rows
for rownum in range(1,rows+1):
    print("*"*star)
    star-=1
```

example-9:

```
*
 *
  *
   *
    *
```

code:

```
rows=int(input("enter the number of rows: "))
for rownum in range(1,rows+1):
    print((" "*rownum)+"*")
```

example-10:

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

code:

```
rows=int(input("enter the number of rows: "))
for rownum in range(1,rows+1):
   if rownum==1 or rownum==rows:
        print("* "*rows)
    else:
        print("*"+(" "*(rows*2-3))+"*")
```

example-11:

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

code:

```
rows=int(input("enter the number of rows: "))
for rownum in range(1,rows+1):
    if rownum==1 or rownum==rows:
        print("* "*(rows*2))
    else:
        print(" "*((rows*2)-2)+"*")
```

example-12:

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

code:

```
rows=int(input("enter the number of rows: "))
for rownum in range(1,rows+1):
    if rownum==1 or rownum==rows:
        print(" "*(rownum)+"* "*rows)
    else:
        print(" "*(rownum)+"*"+(" "*(rows*2-3))+"*")
```

example-13:

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

code:

```
rows=int(input("enter the number of rows: "))
for rownum in range(1,rows+1):
   if rownum==1 or rownum==rows:
       print("* "*rows)
   else:
       print(" "*(rows-rownum)*2+"*")
```

example-14:

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

code:

```
rows=int(input("enter the number of rows: "))
space=rows
for rownum in range(1,rows+1):
   print(" "*(space)+"* "*rownum)
   space-=1
```

example-15:

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

code:

```
rows=int(input("enter the number of rows: "))
space=rows
for rownum in range(1,rows+1):
  if rownum==1:
       print(" "*(space)+"* "*rownum)
  else:
      print(" "*(space)+"*"+(" "*((rownum*2)-3))+"*")
  space-=1
```

example-16:

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

example-17:

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

13.list():
this function is used to "convert the given iterable into list" iterable means "list, tuple, set, string, dictionary, range()..........."
list(iterable) ====>list

example:

```
print(list((1,2,3,4)))#[1, 2, 3, 4]
print(list(range(1,10)))
print(list({1,2,3,4,5}))
print(list({1:2,3:4,5:6}))
print(list([1,2,3,4,5,6]))
```

```
t1=(10,20,30,40)
print(type(t1))
t1=list(t1)
print(t1)
print(type(t1))
```

14.tuple():
this function is used to "convert the given iterable into tuple" iterable means "list, tuple, set, string, dictionary, range()..........."
tuple(iterable) ====>tuple

example:

```
print(tuple([1,2,3,4,5]))
print(tuple({1,2,3,4,5,6,7,8,9,10}))
print(tuple({1:2,3:4,5:6}))
print(tuple(range(1,10,2)))
```

15.set():
this function is used to "convert the given iterable into set" iterable means "list, tuple, set, string, dictionary, range()..........."
set(iterable) ====>set

example:

```
print(set([1,2,3,4,5,6,7,8,9,10,1,2,3]))
print(set((1,2,3,1.2,3.4,5.6,"hello","hai")))
print(set(range(1,10,3)))
print(set({1,2,3,4,5,6}))
print(set({"name":"suraj","location":"hyderabad"}))
```

16.dict():
this function will convert the given data into "dictionary" this function will take data in the form of "keyword arguments" each keyword act like a "key" in dictionary and value of the given keyword is "value" for the key in the dictionary
example:

```
print(dict(a=10,b=20,c=40))
print(dict(a=2,c=4,d=6,e=8))
print(dict(name="suraj",location="hyderabad",salary=10000))
print(dict(keys="a",values=10))
```

17.sum():
this function is used "to find the sum of the values of the given iterable"
sum(iterable_name) ===> sum of the values of the given iterable

example:
```
print(sum([1,2,3,4,5,6,7,8,9,10]))
print(sum((-10,2,3,5,10)))
print(sum(range(1,11,2)))
print(sum({1,2,3,4,5}))
print(sum({1:2,3:4,5:6}))
```

18.max():
this function is used "to find the maximum value of the given iterable"
max(iterable_name) ===> maximum value of the given iterable

example:
```
print(max(1,2,3,4,5))
print(max(-10,-2,-3,-5,-6))
print(max([1,2,3,4,5,6,7,100]))
print(max(range(1,100,5)))
print(max({1,2,3,4,5,6,7,8,9,10}))
```

19.min():
this function is used "to find the minimum value of the given iterable"
miniterable_name) ===> minimum value of the given iterable

example:
```
print(min(10,-1,2,3,-99,-1000))
print(min(range(2,10,2)))
print(min({1,2,3,4,5,6,7,8,9,10}))
print(min({1:2,3:4,5:6,7:8,9:10}))
```

20.abs():
this function makes every given value always as "positive"
example:

```
print(abs(-0))
print(abs(-1.2))
print(abs(-1000))
print(abs(1000))
try:
    number=int(input("enter the number:"))
except:
    print("invalid input")
else:
    print(-number) if number<0 else print(number)
```

21.enumerate()

22.eval():
this function is used to "evaluate the given expression into result"here we will give the an expression in the form of string eval("expression") ===>result
example:

```
print(eval("10+20"))
print(eval("10==20"))
a=10
b=20
print(eval("a+b"))
a="hello"
b="world"
print(eval("a+b"))
print(eval('10+20'))
print(eval("a=10"))
```

23.exec():
this function is used to "execute the given code ,which is in the form of string, it will execute any operation, which is given in string"

example:

```
exec("a=10")#a=10
exec("b=20")#b=20
exec("print(a)")#10
exec("result=a+b")
exec("print(result)")
exec("print(a)")
```

24.sorted():
this function is used to "Sort the given data either in the ascending or descending order" this function take the data in the form of "iterable" and return the sorted result always in the "list" format sorted(iterable_name,reverse=True or False)

note:
if we take "reverse=True" , the data will in the form of "descending order" if we take "reverse=False" ,the data will be in form of "ascending order" by default, sorted() function will sort the data in "ascending order" and give the result in the form of "list".

example:
```
print(sorted([10,20,30,4,5,6,7,-100,-300,-789]))
print(sorted([10,20,30,4,5,6,7,-100,-300,-789],reverse=True))
print(sorted([10,20,30,4,5,6,7,-100,-300,-789],reverse=False))
print(sorted({"abc","aac","baa","cab","aaa"}))
print(sorted((1,2,3,4,1.2,3.4,5.6)))
```

25.open():

26.zip()

27.len():
it will give the "number of elements(values) present in the given iterable(list, tuple, set, string, dictionary) len(iterable_name) ====>length of the given iterable

example:
```
l1=[1,2,3,4,5,6,7,8,9,10]
print(len(l1))
print(len((1,2,3,4,5,6,7,8,9,10)))
print(len("hello"))
print(len({1,2,3,4,5,6}))
print(len({1:2,3:4,5:6}))
```

28.range():
this function is used to give the values for the given range

syntax:
range(start, end, step)

example:

```
print(list(range(1,10)))
print(list(range(100)))
print(list(range(10,1,-1)))
```

29.map()

30.filter()

31.isinstance():
this function is used to compare the "given object" is related to given class or not it will give the result either " True or False "

syntax:

```
isinstance(object, class_name)
```

32.divmod():
this function will return "both quotient and remainder"
syntax:

```
divmod(number1,number2) ===>(number1//number2,number%number2)
```

example:

```
print(divmod(10,20))#(0,10)
print(divmod(2,4))#(0,2)
print(divmod(20,10))#(2,0)
```

note:
divmod() will give the result in the form of "tuple" format

built-in modules:
in general, in python we will have the two types of modules:
==
1.user defined modules:
this module will created by the "programmer or developers" using python

2. built-in modules:
these modules are given by the "python" we are going to work with following built-in modules:
1.math module:

this module will be used in python , to peform all mathematical operations in this module, we will have the following important functions:

1. ceil():
it will take any real number or integer as input, and it will return "integer" as result it will always give the "up-value" for given any real number
syntax:
```
import math as mt
mt.ceil(number) ===> integer number
```

example:
```
import math as mt
print(mt.ceil(5))
print(mt.ceil(4.1))
print(mt.ceil(5.0))
print(mt.ceil(5.01))
print(mt.ceil(-5.2))
print(mt.ceil(-3.8))
```

2.floor():
it will take any real number or integer as input, and it will return "integer" as result it will always give the "down-value" for given any real number
syntax:
```
import math as mt
mt.floor(number) ===> integer number
```

example:
```
import math as mt
print(mt.floor(4.5))
print(mt.floor(-3.8))
print(mt.floor(-2.9))
```

3.trunc():
it will take any real number or integer as input, and it will return "integer" as result it will simply truncate the "fractional part/integral part".

syntax:
```
import math as mt
mt.trunc(number) ===> integer number
```

example:

```
import math as mt
print(mt.trunc(12.356))
print(mt.trunc(4.56789))
print(mt.trunc(-3.4567))
print(mt.trunc(-2.456))
```

4.fsum():

this function will be used "to find the sum of the values of the given iterable"

note:

sum() ===>this function will give the result based on the data in the iterable it is a built-in fucntion

fsum()===>this function will give the result always in float value it is a function of math module

example:

```
import math as mt
print(mt.fsum([10,20,30,40,50]))
print(mt.fsum((1,2,3,4,5)))
print(mt.fsum(range(1,10,3)))#1,4,7 ===>12.0
```

5.fmod():

it is used to find the "reminder of the given two numbers" fmod() function will always return "float value" as result

example:

```
import math as mt
print(mt.fmod(10,20))
print(mt.fmod(20,10))
print(mt.fmod(1.2,3.4))
print(mt.fmod(3.4,1.2))
```

6.prod():

this function will be used "to calculate the product of the numbers of the given iterable"

syntax:

```
import math as mt
mt.prod(iterable_name)
```

example:

```
import math as mt
print(mt.prod([1,2,3,4,5,6,7,8,9,10]))
print(mt.prod((10,20,30,40,50)))
print(mt.prod(range(1,5)))
print(mt.prod({1,2,3,4,5,6,7,8}))
```

7.factorial():

0!=1
1!=1
2!=2
6!=720

this function is used to calculate the "factorial of the given number"

example:

```
import math as mt
print(mt.factorial(3))#3*2*1=6
print(mt.factorial(4))
print(mt.factorial(0))
print(mt.factorial(10))
```

write a python program to calculate the factorial of the given number without using any built-in function or method of the any built-in module:
code:

```
fact=1
try:
    number=int(input("Enter the number:"))
except:
    print("invalid input")
else:
    if number==0 or number==1:
        print(1)
    elif number>0:
        while number!=0:
            fact*=number
            number-=1
        print(fact)
    else:
        print("invalid input")
```

write a python program to calculate the factorial of the given number without using any built-in function or any method of the any built-in module and also without "*" operator:
code:
```
fact=0
try:
    number=int(input("Enter the number:"))
except:
    print("invalid input")
else:
    if number==0 or number==1:
        print(1)
    elif number>0:
        temp=number
        while number!=0:
            while temp!=0:
                fact+=number
                temp-=1
            print(fact)
            number-=1
            temp=number

        print(fact)
    else:
        print("invalid input")
```

write a python program to find the multiplication of the given two numbers without using "*" operator:
code:
```
result=0
try:
   num1=int(input("enter the number1:"))
   num2=int(input("enter the number2:"))
except:
    print("Invalid Input")
else:
  while num2!=0:
      result+=num1
    num2-=1
  print(result)
```

8.fabs():
this function will give the result always in "real number or floating-point number"
example:

```
import math as mt
print(mt.fabs(-10))
print(abs(-10))
print(mt.fabs(-1.234))
print(abs(-1.234))
```

9.comb():
it will give the output as same as "nCk"

example:

```
import math as mt
print(mt.comb(10,3))
print(mt.comb(10,10))
print(mt.comb(10,20))
print(mt.comb(20,10))
```

10.perm():
it will give the output as same as "nPk"

example:

```
import math as mt
print(mt.perm(10,3))
print(mt.perm(10,6))
print(mt.perm(10,20))
```

11.distance():
it is used to find the Euclidian distance here the data will given the inform of "iterables"
example:

```
import math as mt
p1=[10]
p2=[20]
print(mt.dist(p1, p2))
print(mt.dist([1,2],[3,4]))
```

12.sqrt():
this function is used to find the squre root of the given value
example:
```
import math as mt
print(mt.sqrt(10))
print(mt.sqrt(20))
print((10)**(0.5))
print((20)**(0.5))
```

13.cbrt():
this function is used to "find the cube root of the given number"

example:
```
import math as mt
print(mt.cbrt(25))
print(mt.cbrt(125))
print((25)**(1/3))
print((125)**(1/3))
```

logerthemic fucntions:
log2()
log10()
log()

example:
```
import math as mt
print(mt.log2(1))
print(mt.log10(1))
print(mt.log2(4))
```

write a python program, to find the given number length using log() function:
code:
```
import math as mt
number=int(input("enter the number:"))
if number>0:
   print(mt.ceil(mt.log10(number)))
elif number<0:
    print("invalid input")
```

exponential fucntions:

exp()<=== it used to find eluers number for given number
pow()<=== it used to find the power for given base and exponent

example:

```
import math as mt
print(mt.pow(2,3))
print(2**3)
print(mt.exp(1))
```

trigonometric functions:

sin(), cos() , tan()
asin(), acos(), atan()
sinh(), cosh(), tanh()

2. sys module
3. os module
4. time module
5. datetime module
6. threading module
7.abc module
8.pickle module
9.json module

10.numpy

11.pandas

12.calender
13.python file handling

14.python exception handling