if the condition is true then control will Execute the statement block else it will through an assertion Error by Printing the msg.

Eg:
```
a = 100
b = 20
assert a>b, 'b is greater'
Print('a is greater')
```

O/p: a is greater

if
```
a = 10
b = 20
```

O/p:

AssertionError : b is greater [Error]

Wap to find len of the collection without using length fondion and for loop.

```
try:
    st = input('Enter the string') / C
    c = 0
    i = 0
    while True:
        res = st[i]
        c += 1
        i += 1
except:
    Print('Len of the given coll is :', c)
```

O/P: Enter the string : hello
Len of the given coll is : 5

Set comprehension :

Whenever we want to create / store the output data in the form of set then we will make use of set comprehension method.

→ with the help of this method we can increase the Efficiency of the code and we can decrease the no : of instruction.

Syntax :

Var = { Val_to_be_stored for var in col if <cond> }

Wap to find the square of all the numbers between the range 1 to 10 and store the result in the form of set

~~A = { res for i in range (1, 10) type(i)~~

A = { i * i for i in range (1, 11) }

Print (n)

Wap to Extract all the integer numbers present inside a List collection and store them in the form of set. n = [10, 2.3, 'kai', 5, 2]

num = { i for i in n if type(i) == int }

Print (num)

## Dictionary comprehension :

zip() :
It is a function which is used to take two collections in for loop at a time by considering two variables

→ If length of both the collection are not same then it will consider the collection which is having lesser than and same number of values will get printed from both the collection.

eg: for i, j in 'hai', 'he' :
    Print(i, j)

→ whenever we want to create output in the form of dictionary then we will make use of dictionary comprehension. with the help of this method we can increase the Efficency of code by decreasing the no: of instructions.

Syntax :

var = {k : v for k v in zip (col1, col2)
    if <cond> }

WAP map to string in the form of dict

a = {k : v for k, v in zip ('hai', 'hai')}
Print(a)

WAP create dict by considering a heterogeneous list collection only if the key is of string type

Print({k : v for k, v in zip ('abc', 'mno') type(k) == str})

(or)

Print({k : v for k v in zip ('1', 23, 3+2j, 'hello', [1, 2, 3, 4, 5])
    if type(k) == str})

WAP to create dict where key should be an integer and value should be a collection which is having more than or equal to 3 has a length by considering a heterogeneous tuple collection

```
Print({K:V for K V in zip((1,2,3,4,5,6),
      ('hello', [1,2,3], (1,), 'hai')) if type(K)== int
      and type(V) in [list, tuple, set, dict]) and len(K)>=3})
```

WAP create dict where value should be the square of two original value by considering to hemogeneous list

```
Print({K:V**V for K, V in zip([1,2,3,4,5,6],[4,5,6,7,
                                8,9]})
```

WAP to get following o/p by considering the given i/p

i/p : 111000110011 00

o/p : True

```
a = 11100011 00 11 00
res1 = st.split('0')
res2 = st.split('1')
res1 = rem(res1)
res2 = rem(res2)
def is-t(a,b):
    if len(a) != len(b):
        return False
    for i in range(len(a)):
        if len(a[i]) != len(b[i]):
            return False
    return True
Print(is-t(res1, res2))
```

Decorators :

Decorators is a function which is use to additional feature to the main task

→ [operation before the task and operation after the main task]

→ there are two types of decorators

1. In-Built decorator
2. user define decorator

1. In-Built decorator :

These are the decorators whose task is predefined by a developers.

Eg: @classmethod, staticmethod, abstractmethod.

2. user define Decorators :

These are the decorative functions which will get created based on the user requirment.

Syntax :

```
def decorate(fun):
    def inner(args):
        [ Operation before main task ]
        fun()
        [ operation after main task ]
    return inner
```

```
@decorate
def frame(args):
    ____
    ____
    ____
frame(args)
```

WAP to calculate the total taken to find the square of all the numbers between range (1,200)

```
import time
st = time.time()
for i in range(0, 201):
    i = i * i
End = time.time()
Print('the total time taken is :' End - st)
```

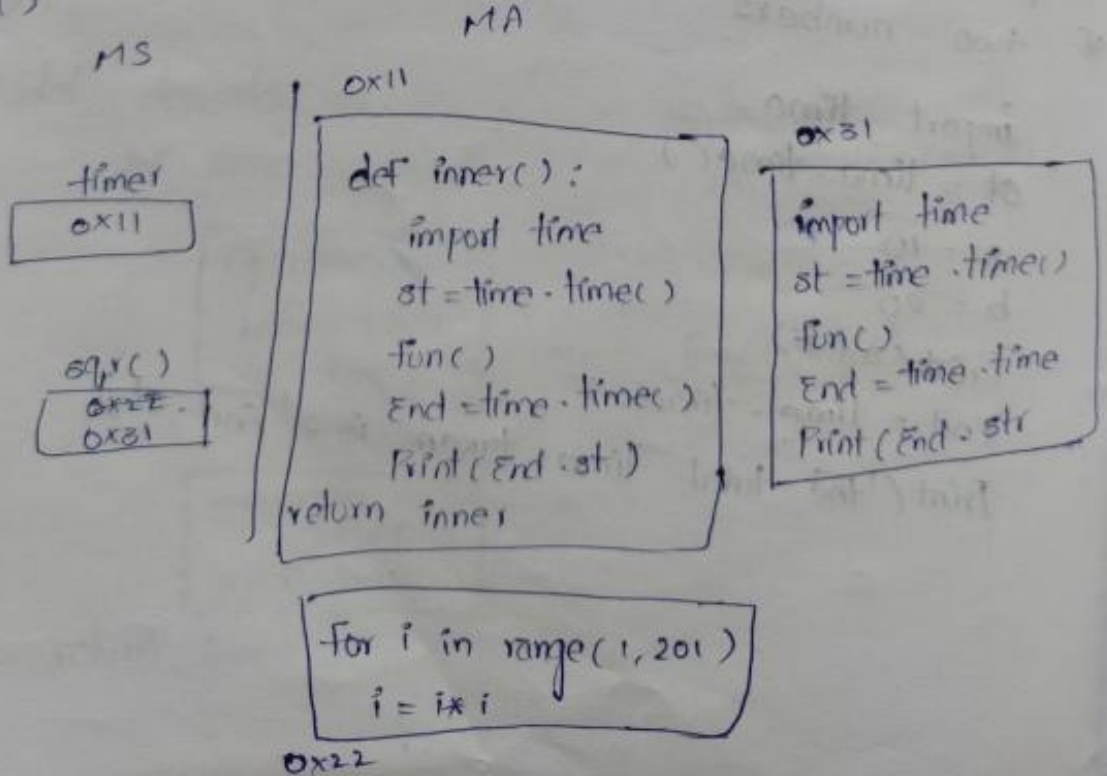
WAP find the total time taken to find the sum of all the numbers btw the range (1, 100)

```
import time
st = time.time(), sum = 0
for i in range(1, 101):
    sum += i
End = time.time()
Print('the total time taken is :' End - st)
```

WAP calculate the total time taken the sum of two numbers.

```
import time
st = time.time()
a = 10,
b = 20
Print(a + b)
End = time.time()
Print('the total time taken is :' End - st)
```

```
def timer(func):
    def inner():
        import time
        st = time.time()
        func()
        End = time.time()
        Print('total time :', End - st)
    return inner
@timer
def sqr():
    for i in range(1, 201):
        i = i * i
sqr()
@timer
def add()
    a = 100
    b = 200
    c = a + b
add()
```



```
def fb(func)
    def inner():
        Print('goto wwow.fb.com')
        Print('login')
        fun()
        Print('logout')
    return inner

@fb
defup:
  Print('photo got uploaded!')

up()
Print('-' * 50)
@fb
def fr()
    Print('request sent')

fr()
```

Regular Expression :

Regular Expression is a phenomenon of doing pattern matching.

→ whenever we want to Extract a particular data which is stored inside a raw data then we can Easily Extract that data with the help of regular Expression concept

→ [A-z] used to match only one char from A to z

→ [a-z] used to match only one char from a to z

→ [0-9] used to match only one char from 0 to 9 only

→ [A-z a-z] used to match only one from either A-z and a-z

→ [A-z a-z 0-9] used to match only one from A-z or a-z or 0-9 (or) /w

→ To match n char btw A-z ⇒ {n}

→ To match char btw [A-z] we can use {m,n}

→ It is used to match 0 or one char [?]

→ [+] it is used to match 1 to n char

→ [*] it is used to match 0 to n char

→ In Regular Expression '/' is used to remove the special meaning of any special char.

→ In Regular Expression highfen (-) is used for all act as special character.

Wap Expression to match the Indian phno number

91 - ----------

91 - [1-9][0-9]{9}

91 - [1-9]\d{9}

Wap an Expression to match university seat no

17JFIA0519

\d{2}[A-Z]{2}\d[A-Z]\d{4} (or) \w{10}

Write a Regular Expression to match IP address

---.---.---.---

\d{1-3}\.\d{1-3}\.\d{1-3}

Write a RegE to match Pan card no.

RAMAN1234G

[A-Z]{5}\d{4}[A-Z]

Write a RE to match date of birth dd-mm-yy

18-FEB-2001

\d{2}-[A-Z]{3}-[\d]{4}

or

15-08-1947

\d{2}-\d{2}-\d{4}

Write an RE to match Email id.

Pyspiders@gmail.com

\w+\@\w+\.[A-Za-z]{2,3}

Pyspiders.new@python.gmail.com

\w+\.\w\@\w+\.\w+\.[A-Za-z]{2-3}

\w+\?\w*\@\w*\?\w+\.[A-Za-z]{2-3}

## Accessing the Required data from the Original (or) Raw Data.

→ Whenever we want to access the data based on the regular Expression then we need to use a function call 'findall'

→ This function is not directly available so that we need to import regular Expression then we can use 'findall'

Syntax:

```
import re
var = re.findall(Exp, data)
```

Eg:

```
fo = open(r'c:\users\@sp\regexp.txt', 'r')
res = fo.read()
fo.close()
import re
out = re.findall('[6-9]\d{9}', res)
Print(out)
```

(or)

```
('\w+\.?\w+@\w+\.?\w+\.[a-z A-Z]{2,3}')
Out = re.findall('@yspiders', res)
Print(len(out))
```

# Multi Threading:

Process:

Process is a Program which is in Execution

Thread:

Thread is a smallest part of Process

→ Multithreading is a phenomenon of Executing 2 or more task parallely.

→ we can run 2 task parallely only if the task are independently Each other

→ To Execute independent functions we need to create them as threads

→ We can't create threads directly so that we will make use of syntax.

Syntax:

```
import threading
var = threading.Thread(target = fun, args, (v1, v2))
```

Eg:

```
import threading
t1 = threading.Thread(target = add, args = (12, 34))
t2 = threading.Thread(target = sub, args = (50, 25))
t1.start()
t2.start()
t2.join()
t2.join()
```

17/12/2021

SQL connection / Backend connectivity :

Whenever we want communicate with database file by being their in Python file, we will make use of SQL connection / Backend connection.

→ To Establish the connection between ~~data~~ database file and Python file we are going to import a module called 'sqlite3' and we will import the functionalities of that module into Python file.

Some important functions used in SQL connection :

1. connect() :
   It is a function which is use to Establish the connection between Python file and database file

2. cursor() :
   It is a function which is use to point to the Particular database



3. Execute() :
   It is a function which is use to Execute the Queries

4. commit() :
   It is a function which is use to save all the changes.

5. close() :
   The function which is use to close the Open file.

* These functions are use to insert the data

Syntax:

```
import sqlite3
Var = Sqlite3.connect('File.DB')
Var1 = Var.cursor()
Var1.Execute('Query')
Var.commit()
Var.close()
```

To Extract the data from the data base we will make use of two functions.

1. fetchone():

```
Var.fetchone()
```

This function is use to fetch the first record present in the data base.

2. fetchall():

```
Var.fetchall()
```

This function is use to fect the all the records present in the data base.

To insert the data:

```
import sqlite3
a=sqlite3.connect('file1.bal')
b=a.cursor()
b.Execute('create table Python(sname char, sid char)')
b.Execute('insert into Python values("A", "S15")')
b.Execute('insert into python values("B", "S14")')
a.commit()
a.close()
```

Fetch / Extract the data

```
c=sqlite3.connect('file1.db')
d=c.cursor()
d.Execute('select * from Python')
res=d.fetchall()
Print(res)
c.commit()
c.close()
```

Package Architecutre:

Package Architecture is a phenamenon of of diviaing the packages into n no:of Packages

Project:

Project is a goal or aim which is sent by the company based on the requirments from clients.

Package :

Package is a folder which consist of n no: of files init

import Keyword :

import is use to drag the functionalies of one package to another package.

from Keyword :

From is use to point the particular location

→ whenever we want to use any inbuilt functions from any of the modules we will make use of two keywords

1. From Keyword :

It is a keyword which is use to make the control to point to the given location.

2. import Keyword :

It is a keyword which is used to import as grap the functionalites of from the inbuilt modules.

In math we have so many inbuilt functions but we are using factorial(), pow(), pi.

Eg:

```
import math
n=int(input())
a = math.factorial(n)
Print(a)

m=int(input('Enter the num:'))
p=int(input('Enter the num:'))
Print(math.pow(m,p))
Print(math.pi)
```

Random Module:

It is an inbuilt package whenever we want to generate any values randomly then we will make use of this module.

Syntax:

```
import random
Var = random.randint(x, y)
```

Eg:

1. 
```
import random
Print(chr(random.randint(ord('A'), ord('z'))))
```

2. 
```
import random
Print(random.randint(1,4))
```

file1

```
Print('ctrl is in first file')
class sam():
    Print('sam func')
class a:
    a=10
Print(A.a)
```

o/p:

```
ctrl is in first file
sam func
10
```

file2

```
from file1 import sam
Print('ctrl is in 2nd file2')
sam()
```

o/p: ctrl is in first file
sam func
10
ctrl is in 2nd file2

→ If we execute file1 it will display regarding output.

→ if we want to execute one file in the another file we have to use from keyword to pointout and import to grap the data from the file1.