-------- NOTES --------

What is Python??

Why python??

Top MNCs working on Python?? TCS, Infy, Google, IBM ….

Python Installation

==========

Q- What is python??
=> Python is General purpose Dynamic High Level Interpreted prog lang.

Python is Dynamic prog lang:->

- There are two types of prog langs

1- Static prog lang: -

-> At the time of variable definition if we specify the datatype of variable then that prog lang is static prog lang.

Q-> How to declare variable??

=> sty: -

datatype variable_name = value

eg:

int a = 10 ;

eg:-

c , cpp , java

2- Dynamic prog lang:-

-> At the time of variable definition if there is no need to specify the datatype of variable then that prog lang is dynamic.

-> as per the User provided value PVM will consider the datatype of variable.

Q- How define variable??

=> sty

variable_name = value

eg:

a = 10

eg:- Python.

Python High level prog lang :->

-> There are two types of prog langs

1-> High level :->

-> Human readable prog lang

eg:

for x in range (10):

print(x)

eg:-

c , cpp , java , PHP , ruby , python ,JS.

2-> Low level :->

-> Machine readable prog lang / binary prog lang

eg :

10100 10001 010 10001(1010):

1000(1010)

eg:

assembly

Python is Interpreted prog lang.

-> There are two types of prog lang

NOTE :->

Compiler and Interpreter is a prog That will convert High level prog to low level prog.

1-> Compiler.

-> It will scan all code at a time and compile.

-> compile time Error check.

-> more memory is required.

2-> Interpreter.

-> It will scan one line at a time and Interpret.

-> Run time Error check.

-> Less memory is required.

Python is General purpose prog lang.

-> We can use python anywhere.

1- Web development
2- IOT
3- testing
4- DS
5- MC
6- DL
7- RPA
8- games
9- GUI
10- Web scrapping.
11- Big data processing
12- Automation
13- DA
14- Desktop application
15- Hacking

Python Official website

-> www.python.org

-> google => psf => python software foundation.

C++ :-

```
#include<iostream>
int main(){

        int a , b , c;
        a = 10 ;
        b = 20 ;
        c = a + b ;
        cout << c << endl;

        return 0;
}
```

Java :->

```
public class AddTwoNumbers{
        public static void main(String [] args){
                int a , b , c;
                a = 10 ;
                b = 20 ;
                c = a + b ;
                System.out.println(c)
        }
}
```

---

python :->

```
a , b = 10 , 20
print(a + b)
```

Q- How to define variable in python??
=> sty

```
var_name = value
```

eg:

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

Q-> What is variable??
=> It is container that will hold some value.
=> Variable is named memory location.

Basic Python + Functional Prog + OOP + File Handling + Exception Handling + HTML + CSS + Bootstrap + SQL + Django + numpy + Pandas + …..

Variable :->
=> It is container that will hold some value.
=> Variable is named memory location.

# sty to define variable :->
variable_name = value

# eg :-
a = 10

==========

Predefine function in Python :->

1- print() :
use :->
To print msg or output on screen we can use print function.
sty :->
print("msg" , var_name)
eg :->
a = 10
print("Value of a is: " , a) # Value of a is: 10

2- type () :
use :->
type function will return which type of data is stored in my variable that data type will return by type function.
sty :->
type(var_name)
eg :->
a = 10
print(a) # 10
type(a) # <class "int">

f = 10.5
print(f) # 10.5
type(f) # <class "float">

3- id() :
use :->
Will return memory location of my variable in memory in which location/address my variable is stored that address will return by id function.

sty :->
id(var_name)

eg:->
a = 10
id(a) # 2005
b = 20

```
        id(b) # 1005
        c = 30
        id(c) # 3005
```

Input function in python :->

use :->

-> To accept dynamic data from keyboard.
-> To accept user input from keyboard.

sty :->

```
var_name = input("MSG to end user")
```

expl :->

input function will accept data from keyboard and it will store that data in a variable.

eg :->

```
a = input("Enter First number: ") # 10

print(a) # '10'
```

NOTE :->

=> Input function is always works on string datatype only. that's why typecasting is must.

Q-> what is typecasting??
=> converting one datatype to other datatype
eg:->

```
s = '10'
print(s) # '10'
type(s) # <class "str">
a = int(a)
print(a) # 10
type(a) # <class "int">
```

WAP that will accept two numbers from user and print addition??

=>

```
>>> a = int(input("Enter first number: "))
Enter first number: 10
>>>
>>> b = int(input("Enter second number: "))
Enter second number: 20
>>>
>>> type(a)
<class 'int'>
>>>
>>> type(b)
<class 'int'>
>>>
>>> print(a + b)
30
>>>
>>> a = int(input("Enter first number: "))
Enter first number: 10
```

```
>>> b = int(input("Enter second number: "))
Enter second number: 20
>>>
>>> print(a * b)
200
>>>
>>> a = int(input("Enter first number: "))
Enter first number: 10
>>> b = int(input("Enter second number: "))
Enter second number: 20
>>>
>>> print(a / b)
0.5
>>>
>>>
```

WAP To calculate are circle??

NOTE :->

input radius of circle from user.

prog :->

```
>>> r = int(input("Enter radius of circle: "))
Enter radius of circle: 10
>>>
>>> area = 3.14 * r * r
>>>
>>> print(area)
314.0
>>>
>>> print("Area of circle is: " , area)
Area of circle is:  314.0
>>>
>>> r = float(input("Enter radius of circle: "))
Enter radius of circle: 10.5
>>>
>>> area = 3.14 * r * r
>>>
>>> print(area)
346.185
>>>
```

==========

Basic Python :->

1-> Identifiers
2-> Keywords
3-> datatypes
4-> operators
5-> flow control statements.
6-> project (Student Management system)

==========

1-> Identifiers:-

Q- What is Identifier??

=> The name which is used for Identification purpose that name is by default consider as Identifier.

eg:-

variable name , class name , function name , method name

eg:

```
price = 45.22
a     = 10
area  = 310
```

rules to define Identifiers:-

1- It can contain Alphanumeric characters.
   -means 0-9 , a-z , A-Z
2- It should not starts with digit.
3- _ is the only spl symbol used in Identifiers.
4- Python Identifiers are case sensitive.
5- keywords cannot be a Identifiers.
6- There is no length limit to define Identifier.

==========

2-> Keywords in python:-

- Keywords are the reserved words wich is having it's own meaning.

Q-> How to print list of keywords??

=> prog

```
import keyword

print(keyword.kwlist)

# list of keywords
```

```
>>> import keyword
>>>
>>> print(keyword.kwlist)

['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import',
'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

==========

3-> datatypes :-

- To represent data we need datatypes.

1- Integer
2- float
3- string

4- bool
5- complex

6- bytes
7- bytearry
8- list
9- tuple
10- set
11- dict
12- range
13- frozenset
14- None

1- Integer datatype :->

use :- To represent Integer value we can use Integer datatype.

Q- What is Integer value ??
=> value without floating point.

eg :

```
a = 10
type(a) # <class "int">

b = -10
type(b) # <class "int">
```

2- Float datatype :->

use :- To represent floting point value.

Q- what is float value ??
=> value with floating point.

eg:

```
f       = 10.5
type(f) # <class "float">

a       = -10.5
type(a) # <class "float">

price   = 100.50
type(price) # <class "float">
```

3- bool datatype :-

use:- For logical representation we can use bool datatype.

Q- What is logical representation ??
=> Output of any condition is always in True and False that is nothing but logical representation.

NOTE :-

Boolean values are following
1-> True

- Internal value of True is 1.

2-> False

- Internal value of False is 0.

proof :-

```
1 + 1 = 2       True + True = 2
1 + 0 = 1       True + False = 1
0 + 1 = 1       False + True = 1
0 + 0 = 0       False + False = 0
```

proof :-

```
int(True) # 1
int(False) # 0
```

proof :

```
float(True) # 1.0
float(False) # 0.0
```

eg:

```
b = True
type(b) # <class "bool">
```

eg:

```
a = 10
b = 20

print( a > b ) # False
print( a < b ) # True
```

4-> Complex datatype :->

use :->

For Math application we can use complex datatype.

What is the sty to define complex datatype ??

=> sty :->

```
var_name = a + bj
```

expl :-

-> a means real part

-> b means imag part

eg :-

```
c = 10 + 20j
type(c) # <class "complex">
```

NOTES :->

1-> Real part can be Integer or float

2-> Imag part must be decimal or float

Integer :->

Q- what is various ways to represent integer number ??

=> There are four ways

1- Decimal :-

-> Means number between 0 to 9

eg :

```
d = 16374547207
type(d) # <class 'int'>
```

NOTE :- PVM will treat any number as a decimal number only.

2- binary :-

-> means 0 and 1

eg:

```
b = 1010 # decimal number
type(b) # <class "int">
print(b) # 1010
```

Q-> What is sty to repersent binary number ??

=> sty

```
var_name = 0b/0Bvalue
```

eg:

```
b = 0b1010
print(b) # 10
type(b) # <class "int">
```

3- octal :->

means base 8 and number's between 0 to 7.

eg:

```
o = 133625
type(o) # <class "int">
```

Q-> What is sty to reperesent octal value ??

=> sty:

```
var_name = 0ovalue
```

eg:

```
>>> o = 0o576471
>>> o
        195897
```

4- Hexa- Decimal :-

base 16 value :- the number's between 0 to 9 and A/a to F/f

eg :

```
h = face
NameError : name 'face' is not defined.
```

Q-> What is sty to represent hexa decimal number ?

=> sty

var_name = 0xvalue

eg:

h = 0xface

```
>>> h = 0xface
>>> h
64206

>>> h = 0xFACE
>>> h
64206
>>>
>>> type(h)
<class 'int'>
```

eg:

```
c = 0b1010 + 20.5j # valid
c = 20.5 + ob1010j # in-valid
```

Complex function ??

Q-> How to use complex function ??

=> sty :->

```
var_name = complex(real_value , imag_value)
```

eg:

```
c = complex(20.5 , 0b1010)
print(c) # 20.5 + 10j
type(c) # <class "complex">
```

 # String Datatype :->

Q-> What is string datatype ??

=> Anything if we specify in ' ' or " " that is nothing but string.

NOTE :->

- There is no char datatype in python.
- If we specify character in a ' ' or " " it is also string only.

eg:

```
c = "A"

type(c) # <class "str">

a = 10
type(a) # <class "int">

a = "10"
type(a) # <class "str">
```

```
s = "om sai ram"
type(s) # <class "str">
```

Q-> Memory representation of string ??
=> by using array

eg :->

```
s = "santosh"
type(s) # <class "str">
print(s) # santosh
```

expl :->

1-> Python supports 0 base indexing.
2-> In python there are two types of indexing
    1-> +ve / Forword indexing
    2-> -ve / reverse indexing
3-> +ve indexing starts with 0 and from left to right.
4-> -ve indexing starts with -1 and from right to left.

Q-> How to define string object / string variable??
=> sty :

```
var_name = "value"
type(var_name) # <class 'str'>
```

Q-> How to access string data??
=> by using index and slice
=> we can access string data using index and slice

Q-> When to use index??
=> To fetch single element from string we can use index.
sty:

```
var_name[index_value]
```

eg:

```
s = "santosh"
s[3] # 't'
s[-4] # 't'

s[6] # 'h'
s[-1] # 'h'
```

Q-> what is slice and how to use??
=> If we want to fetch more than one character from the given sequence then we can use slice.

sty :->

```
var_name [ start_index : end_index ]
```

expl :->
This operator will return start to end-1 index value.

eg:
```
s = "santosh"
s[1] # 'a'
s[1 : 5] # "anto"
s[3 : 6] # 'tos'
s[1 : 3] # 'an'
s[-5 : -2] # 'nto'
s[ : ]
```

EG :->
```
s = 'instagram'
```

egs:

+ve index 10

-ve index 10

PM on slice :->
sty :->
```
var_name[start_index : end_index]
```

1 : start_index is optaional and default value of start_index is 0.
sty :
```
var_name[ : end_index]
```
expl :
This slice will return 0 to end_index - 1 value.
eg:
```
s = "instagram"
s[ : 7] # "instagr"
```

2 : end_index is also optional and default value is len(string) -1 index value.

sty:
```
var_name[start_index : ]
```

expl :-
This will return start_index to len(string)- 1 index value.
eg:
```
s[3 : ] # 'tagram'
```

3 : start and end index both are optional.
sty:
```
var_name[ : ]
```
expl:
This slice will return 0 to len(string)-1 .
menas it will return hole string.

eg:

```
s = "Instagram"
s[ : ] # 'Instagram'
```

==========

NOTE :->

-> All Five fundamental datatypes are Immutable.

-> All Five fundamental datatypes : int , float , str , complex , bool

Q : what is the meaning of Immutable ??

=> non - changeable.

=> Once we create an object there is no way to change in that object. if we are trying to change in that object then by default new object got created with those changes such type of thing is nothing but Immutable.

eg:

```
a = 10
type(a) # <class 'int'>
a = 11
a = 12
```

Object Reusing concet :->

-> If the value of object is same then plz don't create new object instead assign reference to the same object.

eg:

```
a = 10
b = 10
c = 10
d = 10
```

```
>>>
>>> a = 10
>>> b = 10
>>> c = 10
>>> d = 10
>>>
>>>
>>> id(a)
1876456860240
>>> id(b)
1876456860240
>>> id(c)
1876456860240
>>> id(d)
1876456860240
>>>
>>> a
```

```
10
>>> b
10
>>> c
10
>>> d
10
>>>
```

==========

real life example on Immutability :

    # Voter registration system for pune (25lacks voter):

```
v1 = 'pune'
v2 = 'pune'
v3 = 'pune' # F
v4 = 'pune'

v25lacks = 'pune'

v3 = 'mumbai'
```

==========

 # List datatype :->

Q- What is list ??
=> It is ordered mutable heterogenous collection of elements where insertion order is preserved and duplicates are allowed.

meaning of some imp words :
    1- ordered : means index is present
    2- mutable : means changeable
    3- heterogenous collection : we can add different types of data in list object.
    4- insertion order is preserved : data will be added as per index value.
    5- duplicates are allowed

Q- How to represent list object ??
=> sty:

```
    var_name = []
    type(var_name) # <class "list">
```

Q- How to add data to the list ??
=> by using append method

sty:

```
    var_name = []
    var_name.append(value)
```

eg:

```
l = []
l.append(10) # [10]
l.append(20.5) # [10 , 20.5]
l.append("sai") # [10 , 20.5 , "sai"]
l.append(True) # [10 , 20.5 , "sai" , True]
l.append('sai') # [10 , 20.5 , "sai" , True , 'sai']
```

Q- Memory representation of list object??
=>

```
l = [10 , 20.5 , "sai" , True , 'sai']
```

Q- How to access list data??
-> using index and slice

index :->
 # sty

```
var_name[index_value]
```

 # eg:

```
l = [10 , 20.5 , "sai" , True , 'sai']
l[3] # True
l[-4] # 20.5
```

 # Slice :->
 # sty

```
var_name[start_index : stop_index : step]
```

 # eg:

```
l = [10 , 20.5 , "sai" , True , 'sai']
l[ : : ] # [10 , 20.5 , "sai" , True , 'sai']
l[ 1 :-2 : 1] # [20.5 , 'sai']
```

Q- List object is mutable ??
=> proof :

If we update in list object data and if it will update in the same list object then that object is mutable.
In mutable object we can add update and delete the data from list object.

prog proof:

```
l = []
l.append(10) # add data to list object
l.append(20)
l.append(True)
l.append('sai')

print(l) # [10,20,True , 'sai']

print(type(l)) # <class 'list'>
```

```
print(id(l))  # 2021

l.remove(20) # [10, True , 'sai'] # remove method to remove data from list object

l[0] # 10
l[0] = 100 # Update list object data

print(l) # [100 , True , 'sai']

print(id(l)) # 2021
```

==========

```
# To add data to list object methods :
        append(value) , insert(inedx , value)
# To remove data ??
        remove(value) , pop(index)
```

##########

 # Tuple datatype :->

-> Tuple datatype is exactly same as list datatype the only difference is list object is mutable and tuple object is Immutable.

Q- What is Tuple??
=> It is ordered Immutable heterogenous collection of elements where insertion order is preserved and duplicates are allowed.

NOTE :->
-> Tuple object is immutable that's why we can not add , update and delete the data from tuple object.
-> Tuple object is non-changeable

Q-> How to define tuple object??
=> sty :

```
var_name = ()
type(var_name) # <class "tuple">
```

==========

Q- when to use list object ??
=> If data is in Growable then use list

Q- When to use tuple object ??
=> If the data is static fixed then use tuple.

==========

 # set datatype :->

Q- What is set datatype ??
=> It Is unordered mutable Heterogenous collection of elements where insertion order is not preserved and duplicates are not allowed.

unordered : index is not present

Q-> How to create empty set ??
=> by using set function.

sty:
```
var_name = set(sequence)
```

eg:
```
s = set('santosh')
print(s) # {'h','a', 's','n','t','o'}
type(s) # <class "set">
```

Q- What is sequence ??
=> In any datatype more than item is nothing but sequence.

NOTE :->
1-> WE can not use index and slice on set the reason is set is unordered.

SET IS Mutable so we can add and remove some data from set object .

Q-> How to add data to set object ??
=> by using add method
sty:
```
var_name = set()
var_name.add(value)
```

Q-> How to remove data from set object ??
=> by using remove method
sty:
```
var_name = set()
var_name.remove(value)
```

Q-How to access set object data ??
=> by using for loop

```
for temp_var in sequence:
    print(temp_var)
```

eg:
```
s = set()
s.add(10)
s.add(20.5)
s.add("sai")
s.add(True)
s.add("sai")
```

eg of for loop:
```
for x in s:
```

```
        print(x)

    10
    "sai"
    True
    20.5
```

 # Frozenset Datatype :->

It is Exactly same as set the only differnce is set is mutable and Frozenset is Immutable.

Q- What is frozenset object ??
=> It Is unordered immutable Heterogenous collection of elements where insertion order is not preserved and duplicates are not allowed.

Q-> How to define frozenset object ??
=> sty:

```
var_name = frozenset(sequnec)
```

NOTE:

-> Index and slice in not present for frozenset datatype.

-> Frozenset is a immutable object that's why we can only read frozenset object we can not add update and remove data from frozenset object.

==========

 # Range datatype :->

This datatype will represent a sequnce of hole numbers

There are three ways to represent range datatype

1- range(stop_value)

Exp : This datatype will return sequnce of 0 to stop_value - 1 hole numbers.

2- range(start ,stop)

exp: This datatype will return sequnce of start to stop -1 hole numbers

3- range(start ,stop , step)

exp: This datatype will return start to stop-1 hole numbers with specified step value.

eg:

```
r = range(1 , 10)
for x in r:
    print(x) # 1,2,3,4,5,6,7,8,9
```

eg: WAP That will return table of 5

```
r = range(5 , 51 , 5)
```

eg: WAP That will accept user input and will print table of that number.

```
>>>
>>> n = int(input("Enter the number so will print table of the number: "))
Enter the number so will print table of the number: 28
>>>
>>> for x in r:
        print(x * n)
```

==========

 # Dict :->

Q- What is Dict ?
=> It is unoreded mutable Heterogenous collection of elements where data is represented as a key value.

NOTE :
1 : Dict key must be unique and Immutable.
2 : Dict value can be duplicate as well as it can be mutable or Immutable.

Q- How to represent Dict ??
=> sty:

```
var_name = {}

type(var_name) # <class 'dict'>
```

Q- How to add data to the dict ??
=> sty:

```
var = {}
var['key'] = value
```

Q- How to fecth dict data ??
=> sty:

```
var_name = {key : value}
```

1st way:

```
var_name[key]
```

2nd way:

```
var_name.get(key)
```

Q- How to Initilize a dict ??
=> sty:

```
var_name = {'key1':value1 , 'key2':value2 , key>>n : value>>n}
```

eg:

```
d = {'jay' : 100 , 'kiran' : 200 , 'pavan': 300 , 'nayan' : 400}
```

Q- When to use dict ??
=> If we have reletional data then go for dict

eg:

word : meaning

student : roll number

There are three methods we have for dict.

1: keys()

-sty :

var_name.keys()

-This method will return list of all keys.

2: values()

-sty :

var_name.values()

-This method will return list of all values()

3: items()

-sty :

var_name.items()

- This method will retrun list of key and value

Q- How Iterate dict : (how to use for loop on dict)

1:

```
for temp_var in dict_var :
    print(temp_var) # will get all dict key's

---------------------------------------------------

for temp_var in dict_var.keys():
    print(temp_var) # will get all dict key's
```

2:

```
for temp_var in dict_var :
    print(dict_var[temp_var]) # will get all values

---------------------------------------------------

for temp_var in dict_var.values():
    print(temp_var) # will get all values
```

3:

```
for temp_var in dict_var :
    print(temp_var , '------>' , dict_var[temp_var])

---------------------------------------------------------

for key , value in dict.items():
    print(key , '-------->' , value)
```

==========

 # OPERATORS :->

- Types of operators

1: Arithmetic :

+ , - , * , / NOTE :-> Division operator always return floating point result.

** => Power operator => sty :-> number ** power

// => Floor Division => sty : number // number :-> expl => It will return result as a round-up value.

NOTE :

+ we can use on string :

sty :->

str + str # use : to combine two strings

* we can use on string

sty:

str * int # use : to repeat given string string to n number of times.

% : It will return reminder

2: Logical :->

and : If both operands are True then the result is True else result is False.

or : If at least one operand is True then the result is True.

not : It is a complement operator.

use :-> To Check more than one condition and result in the form of Boolean.

3: relational :->

> , >= , < , <=

equality operators:

== : equals to equals :-> To compare both sides we can use == operator

eg:

```
10 == 10 # True
10 == 10.5 # False
```

!= : Not equals to

Assignment Operator :->

= # assignment operator

eg:

```
a = 10
```

composite Assignment:->

```
a = 10
a += 1   or   a = a + 1
```

sty :->

-= , *= , /= , **= , //=

NOTE :

Output of relational operator and equality operator is always in Boolean(True , False).

4: spl operators :->

in operator :-> It is called as Membership operator

To check given element is a member of given sequence or not if found then return True else False.

sty :

ele in sequence

eg:

's' in 'santosh' # True

'z' in 'santosh' # False

is operator :

sty :

ref_var1 is ref_var2

eg:

```
a = 10
b = 10

print(id(a)) # 2021
print(id(b)) # 2021

a is b # True

a == b # It will check value
```

expl:

for address comparision puprose we can use is operator

==========

 # Flow control statements:

1 : conditional statement :

if , if - elif - else , if -else

2 : Iterative statement :

for , while

3 : Transfer statement :

break , continue and pass

---

1 : conditional statement :

if , if - elif - else , if -else

Use :

According to some condition if i want to execute some block of code then we can use conditional statements.

if statement :

sty :

if condition :

if-block of code

expl :

-> if is a keyword and it is used to start a if statement.

-> condition can be anything and output of condition is always in boolean (True / False)

-> : means if statement header part is complete and if block will starts with 4 space Indentation.

-> if-block of code always writeen in 4 space Indentation and that is standard.

eg:

```
        if True :
            print("This is if-block")

        if False :
            print("this is if-block")

        if 'jay' == 'jay':
            print('Hello jay welcome to python class')
```

if - else :

use : When we want to check one condition.

sty :

```
        if condition :
            if-block body
        else :
            else-block body
```

expl:

if the output of condition is True then execte the if-block else execute else-block

if - elif - else :

use : To check more than one condition.

elif : else-if(condition)

sty :

```
        if condition1 :
            condition1-block

        elif condition2 :
            condition2-block

        elif condition3 :
            condition3-block

        else:
            else-block
```

eg:

```
# This is demo prog for conditional statements

print("*"*70)

print()

print("********** Welcome To Python Calculator **********")
```

```
print('''

    + : Addition
    - : Substarction
    * : Multiplication
    / : Division

''')

print("*"*70)

n1 = int(input("Enter First Number: "))

ch = input("Enter your choice: ")

n2 = int(input("Enter Second Number: "))

if ch == '+' :
    print(n1,ch,n2 ,"=" , n1 + n2)

elif ch == '-' :
    print(n1,ch,n2 ,"=" ,n1 - n2)

elif ch == '*' :
    print(n1,ch,n2 ,"=" ,n1 * n2)

elif ch == '/' :
    print(n1,ch,n2 ,"=" ,n1 / n2)

else:
    print('Plz Enter valid choice')
```

---

2 : Iterative statement :

1: for :->

use :- If we know number of Iteration in Advance then we can use For Loop

sty :->

```
for temp_var in sequence :
    block_of_for_loop
```

expl :->

sequence :- means more than one element :-> range , str , list , tuple , set , dict .. etc

-> if the element's present in the sequence plz Iterate my for_loop_body when elements reaches to 0 then terminate for loop.

```
-> if len(s) > 0 :
    Iterate_for_loop_bod
  elif len(s) == 0 :
    Terminate for-loop
```

2: while :->

use :- If we don't know number Iteration the use While loop.

sty : ICU

initialization # start value

while condition : # end value
while_loop_body
update # increment / decrement

-> if the condition is true then execute while body else Terminate
-> if condition :
execute body_of_while
else:
Terminate while

WAP to print 1 to 10 numbers using while loop

```
i = 1

while i  <= 10 :
   print(i)
   i = i + 1 # i += 1
```

==========

3 : Transfer statement :

break :-
use :-> To terminate my loop.
if (condition):
break_loop
expl :-
according to some condition if i want to break my loop then use break statement.

continue :
if (loop condition == True) :
plz_skip_current_iteration and execute next iteration
expl :->
if i don't want to execute my loop each time then skip the current iteration and go for next Iteration.

pass :
if i want to reserve some block for future purpose then we can pass.

==========

 # Input Output functions:

input function :->
use:->
To take run input from keyboard.
sty:
var_name = input ("MSG to end user")

NOTE:

-> Input function is always works on string only.
-> When we use input function typecasting is must.

eval function :->
use :- To evaluate an expression we can use eval function
sty:
eval('string_of_expression')
NOTE :
-> eval function is always works on string only.
-> when we use eval function with the input function then there is no need of typecasting.

==========

 # Output functions:

use: - To print output(msg) on console / screen.
sty:
print ('msg', var_name , sep = ' ' , end = '\n')

IMP POINTS: -
1 : sep attribute is used to separate output's.
-> default value of sep attribute is empty space.
2 : end Attribute is used to seprate line's
-> default value of end is '\n' and meaning of '\n' is new line.

==========

 # String Formatting

There are Two ways to format my string

God Sai address is shirdi and marks are 100

1: eg: .formate() method

```
name  = "sai"
marks = 100
add   = "shirdi"
```

{} => replacement operator with .format method

```
print('God {} address is {} and marks are {}'.format(name,add,marks))
```

2 : f" method

```
name  = "sai"
marks = 100
add   = "shirdi"
```

{var_name} => replacement operator with variable

```
print(f'God {name} address is {add} and marks are {marks}')
```

==========

Types of prog :

1 : Sequential prog :

statement1
statement2
statement3
eg :

```
a = 10
b = 20
print(a+b)
```

2 : functional prog :

use:->

-> In My prog set/group of statements i want repeatedly then plz take that group of statement inside a function and whenever that group of statements you want then just call your function.

==========

Function :->

When we talk about function Two things are very Imp:

1 : function Definition
2 : function calling

What is use functions??

=> Functions are always acting on data (Variables)
=> Function is used to process a data(variables)
=> To process a data whatever logic is required that logic I'm going to write inside function block(body).

1 : function Definition :->

-> We can define function use def keyword.
sty:

```
def function_name(input arguments):
    ----------
    ----------
     processing of input args
        logic
        body_of_function
    ----------
    ----------
```

2 : function calling :->

-> By using function_name we can call function.
sty:

```
function_name(input arguments)
```

NOTE :->

-> Relation between function_defination(one) and function_call(many)
-> Relation between function_defination and function_call is one to many

```
================================================================================
# WAP To Calculate length of given string using user define function: =>
        # There are two types of function

                # 1: inbuilt functions:
                        print() , id() , type() , len()

                # 2: user define functions:
                        def my_length(s):
                                logic

                        my_length("Santosh") # 7
prog:
        def my_length(s):
                logic


        my_length("Santosh") # 7




# WAP To calculate power of given number using for loop and user define function:
        NOTE:
                Do not use inbuilt power operator (**)

def my_power(a , b):
        logic

my_power(10 , 2) # 100

================================================================================
# Return statement in function :=>
        -> Python functions can return one or more values.

eg:
        def function_name(input_args):
                processing_of_input_args
                function_body / function_block
                retrun value1 , value2 , value 3...,etc


        t = function_name(input_args)
        type(t) # "tuple"


eg:
        def cal(*t): # now here t variable is of type tuple
                sum = 0
                mul = 1
```

```
    for x in t:
        sum = sum + x
        mul = mul * x

    return sum , mul


t = cal(10,20,30,40,50)
print(t) # (150 , 425)

for x in t:
    print(x)

a , b = cal(10,20,30,40,50)
print(a) # 150
print(b) # 425
```

Define two functions and first function is having two variables and i want addition of these two variables in function two.

Local space and Global space :

-> anything at starting of program and outside of functions(body / block) that is Global scope
-> anything in block, body is nothing but local scope.

Global space :->

-> Global scope data is accessible through the program.

Local space :->

-> Local scope data is accessible only within the body or block.

==========

Employee Management System using functional programming and Dict.

1 : Add Employee
2 : Update Employee
3 : Remove Employee
4 : Show all Employee

==========

WAF That will convert given string in uppercase.

- eg:

i/p => "sanTosH"
o/p => SANTOSH

i/p => sAntoSh@33
o/p => SANTOSH@33

==========

 # OOP

sequential programming :->
-> statement by statement coding
sty:
```
statement 1
statement 2
statement 3
statement 4
```
eg:
```
a = input()
b = input()
print(a+b)
```

functional programming :->
-> In functional programming total code is divided into functions and functions use to process a variables.
sty:
```
def function_name(inp_args):
    body

function_name(input_args)
```

eg:
```
def add(a , b):
    print(a+b) # Here we are processing a and b variable


a = input() # a is variable
b = input() # b is variable

add(a, b)
```

==========

Object Oriented programming :->
-> Total code is divided into a object.
-> Three Things are very important in OOP
1 : class
2 : Objects
3 : Reference variable.

Q- What is class??
=> Class is a blueprint to construct an object
=> class is a user define datatype.
=> class is a logical entity
=> class does not required memory for execution.
=> sty :
```
class ClassName: # class definition
    body_of_class
```

Q- what is Object??
=> It is a Physical Entity
=> It is a Instance of class ----> Instance means object
=> objects required memory for execution.
=> sty:

ref_var = ClassName() # object creation

Q- What is reference variable??
=> It is a variable which is pointing to object and use to access the functionality of the object.

NOTE :

1 : relation between class and object ??
one(class) to many(objects)

2 : relation between object and reference variable ??
one(object) to one(reference variable)

3 : by using class we can group variable

4 : reference variable is used to access the functionality of the object.
Q- How to access any functionality of the object by using ref variable??
=> by using . operator which is also called as membership operator (accessor).
sty :

```
ref_var = ClassName() # object
ref_var.funality # accessing

# functionalities means we can access variables and functions of the object.
```

Q- What is class ??
=> sty :

```
class ClassName:
    variables (instance variable, static variable , local variable)
      +
    Methods   (instance Method , class Method , static Method)
```

NOTE :->
Q- What is method ??
=> It is a function only.
=> function within the class called as method.

eg:

```
class Employee:
    def __init__(self , id , name , salary ):
        self.name      = name
        self.id        = id
        self.salary    = salary
        self.email     = self.name + '.company' + '@gamil.com'

    def show_emp_data(self):
        print(f'Employee name is {self.name} and email is {self.email}')
        print()
```

```
jay = Employee(100 , 'jay' , 1200)
jay.show_emp_data()

kiran = Employee(200 , 'kiran' , 1300)
kiran.show_emp_data()

pavan = Employee(300 , 'pavan' , 1400)
pavan.show_emp_data()
```

==========

 # What is self and Constructor :

Q-> What is self ??
=> It is a reference variable which is pointing to current object .
=> Use of self is to access the functionality of the object within class.

eg:

```
class Student:

    def __init__(self):
        print(f'Id of self within the class : {id(self)}')

obj1 = Student()
print(f'Id of obj1 outside of the class {id(obj1)}')
```

Use of Self:
To access the functionality(variables & methods) of the object within the class we can use self reference variable

Use of obj_ref:
To access the functionality(variables & methods) of the object from outside of the class we can use object reference variable.

Q- What is constructor ??
=> Use of constructor is to create object in the memory.
=> To Initialize the object in the memory.
=> Constructor will execute automatically when object is created.
=> Per object constructor will execute only once.

```
sty:
        def __init__(self):
                body_of_construtor
```

eg:

```
class Student:

    def __init__(self , name , marks , roll_no):
```

```
        self.name      = name
        self.marks     = marks
        self.roll_no   = roll_no

    def show_student_data(self):
        print(f'Student Name is {self.name} and marks are {self.marks} and roll number is {self.roll_no}')
        print()


obj1 = Student('Kiran' , 88 , 11)
#obj1.show_student_data()
Student.show_student_data(obj1)

obj2 = Student('Pavan' , 77 , 22)
#obj2.show_student_data()
Student.show_student_data(obj2)
```

==========

```
class ClassName:
        variables(instance , static , local)
          +
        methods(instance , class , static)
```

==========

```
# Methods :
        1 : Instance method
        2 : Static method
        3 : class method


# 1 : Instance method :=>
        => Within the method if we access Instance variables and the first arg of the method is self then that is called as Instance method.
        => NOTE :
                We can access instance method by using object reference only.

        sty :
                def method_name(self):
                        body(Access Instance variables)

# 2 : class method :=>
        => Within the method if we access class/static variables and the first arg of the method is cls then that is called as class method.
        => NOTE :
                - It is mandetory to use @classmethod decorator
                - We can access class method by using object reference and classname but recommonded to use classname only.

        sty :
                @classmethod
                def method_name(cls):
                        body(access static variables)
```

3 : static method :=>
=> Within the method if we access Local variables Then this method is called as static method.
=> NOTE :
- It is mandetory to use @staticmethod decorator
- We can access static method by using object reference and classname but recommonded to use classname only.

sty :
```
@staticmethod
def method_name(cls):
    body(access local variables)
```

==========

NOTES:
1: if We call any method by object reference then that method is called as Instance method.
2: If We call any method by using class name then that method is called as Static method. If we call static method by using class name then @staticmethod decorator is not required.
3: We can call class method by using object reference of class name but @classmethod decorator is mandatory.

==========

Inheritance concept in Python :->
use ?

eg:
```
class Parent:

    def my_property(self):
        print(['Land' , 'Gold' , 'Car' , 'Bike' , 'House' , 'Bank_balance'])


class Child(Parent): # relationship

    pass # (There is no property of the child)

abhi = Parent()

jay = Child()
jay.my_property()
```

 # Types of Inheritance :
1 : single / simple
2 : Multi Level
3 : Hierarchical
4 : Multiple
5 : Hybrid
6 : Cyclic (Python Does not support for Cyclic Inheritance)

1 : single / simple :->

-> Single parent and single child such type of Inheritance called as single Inheritance.

eg :->

```
class P :
    def m1(self):
        print("Parent Class m1 Method")

class C (P) :
    def m2(self):
        print('Child class m2 method')

c1 = C()
c1.m1()
c2.m2()
```

2 : Multi Level :->

eg:

```
class P:

   def m1(self):
      print("This is m1 method")


class C(P):

   def m2(self):
      print("This is m2 method")


class CC(C):

   def m3(self):
      print("This is m3 method")

c1 = CC()
c1.m1()
c1.m2()
c1.m3()
```

3 : Hirachical :->

-> Single parent can have Multiple childs called as Hirachical Inheritance.

eg:

```
class P:

   def m1(self):
      print("This is m1 method")
```

```
class C1(P):

    def m2(self):
        print("This is m2 method")


class C2(P):

    def m3(self):
        print("This is m3 method")

class C3(P):

    def m4(self):
        print("This is m4 method")

c1 = C1()
c1.m1()
c1.m2()

print()

c2 = C2()
c2.m1()
c2.m3()

print()

c3 = C3()
c3.m1()
c3.m4()
```

4 : Multiple :->

Multiple Parent and single child type of Inheritance called as Multiple.

eg :

```
class P1:

    def m1(self):
        print("This is m1 method OF PARENT 1")

class P2:

    def m1(self):
        print("This is m1 method OF PARENT 2")

class P3:

    def m1(self):
        print("This is m1 method OF PARENT 3")
```

```
class C(P2 , P3 , P1):

    def m1(self):
        print("This is m1 method OF CHILD CLASS")

c1 = C()

print(C.mro())
```

Hybrid Inheritance :
-> Combination of all types of Inheritance is nothing but Hybrid Inheritance.

==========

Next concept of Inheritance :
super() :

Q- What is super() ??
=> super is a method

Q- Use of super method ??
=> from the child class if i want to access a parent class members then use super method.

NOTE:
Super method is always talks about Parent class and Inheritance.

Purpose of super method ??
=> To access parent class construtor and parent class methods we can use super method.

sty:

```
super().__init__() # This way you can access parent class constrtor
super().method_name() # This way you can access parent class methods.
```

eg:

```
class Person:

    def __init__(self, name , age):
        self.name   = name
        self.age    = age
            # 100 variables

    def display(self):
        print(f'Name is: {self.name}')
        print(f'Age is : {self.age}')


class Student(Person):

    def __init__(self , name , age , marks , roll_num):
            super().__init__(name , age)
```

```
        self.marks      = marks
        self.roll_num   = roll_num


    def student_data(self):
        super().display()
        print(f'Roll number is : {self.roll_num}')
        print(f'Marks are : {self.marks}')


class Techer(Person):

    def __init__(self , name , age , subject , salary):
        super().__init__(name , age)
        self.subject        = subject
        self.salary         = salary


    def techer_data(self):
        super().display()
        print(f'Special subject is : {self.subject}')
        print(f'salary is  : {self.salary}')


jay = Student('Jay' , 23 , 88 , 11)
jay.student_data()
```

==========

Q- How to Access perticular parent class method ??

=> sty:

```
        ParentClassName.method_name(self)
```

eg:

```
class A:

    def m1(self):
        print("m1 method in class A")


class B(A):

    def m1(self):
        print("m1 method in class B")


class C(B):

    def m1(self):
        print("m1 method in class C")


class D(C):
```

```
    def m1(self):
        print("m1 method in class D")


class E(D):

    def m1(self):
        #print("m1 method in class E")

        B.m1(self)

e = E()
e.m1()
```

==========

 # Polymorphism :->

Ploy means => many

Morphs means => forms

One name but Multiple Forms is the concept of Polymorphism.

eg:

10 + 20 => 30

"10" + '20' => '1020'

as a part of Polymorphism we will learn two things

1: Overloading

2: Overriding

1: Overloading :->

Two types of Overloading

1: Operator Overloading

2: Method Overloading

1: Operator Overloading

eg:

10 + 20 => 30

"10" + '20' => '1020'

prog:

```
class Book:

    def __init__(self , name , pages):
        self.name   = name
        self.pages  = pages

    def __add__(self , other):
        return self.pages + other.pages

    def __mul__(self , other):
        return self.pages * other.pages
```

```
    def __gt__(self , other):
        return self.pages > other.pages

    def __lt__(self , other):
        return self.pages < other.pages

    def __str__(self):
        return f'{self.name} book is having {self.pages} pages.'


b1 = Book('Basics of Python' , 500)
b2 = Book("Python Pro" , 700)


# print(b1 + b2)
# print(b1 * b2)
# print(b1 < b2)

print(b1)
print(b2)

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

# 2: Method Overloading :
        # Python is Dynamically typed prog lang that's why there is no need of Method Overloading in Python.
        # NOTE :
                - Method overloading is not there in python
                - Construtor overloading is not there in python.
                - if we Overload a method or construtor PVM will take last or most recent one method or constrtor.
eg:
class Student:

    def __init__(self , name):
        self.name   = name
        print("I'm in construtor 1")

    def __init__(self , name , marks):
        self.name   = name
        self.marks  = marks
        print("I'm in construtor 2")

    def __init__(self , name , marks , roll_num):
        self.name   = name
        self.marks  = marks
        self.roll_num  = roll_num

        print("I'm in construtor 3")

s1 = Student("Jay" , 55 , 11)

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

Overriding:
-> We will learn method and variable Overriding.

NOTE :
-> IT means redefining the parent class methods and variables in child class.
-> Overriding is always talks about parent class.

eg:

```
class Parent:

    def property(self):
        print("Cash + Gold + car + bike + Land")


    def marray(self):
        print("Black Girl")

class Child(Parent):

    def marray(self): # Method Overriding
        # super().marray()
        print("Red Girl")

c = Child()

c.marray()
c.property()
```

variable overriding:
eg:

```
class Parent:

    prog_lang = "Java"

class Child(Parent):

    prog_lang = "Python"


c = Child()
print(c.prog_lang)
```

==========

Inheritance , polymorphism , Encapsulation , Abstraction.

==========

 # Abstraction :=>

1: Abstract class
2: Interface

Q- How to create abstract class in Python ??
=> by using abc module
=> abc means Abstarct Base Class and abc is a module
=> abc module contains ABC class and by using this class we can create Abstarct class in Python.
=> sty to create Abstarct class in Python ??

```
from abc import ABC
class ClassName(ABC):

    one or more asbstract and non-abstract methods.
```

Q- How to define abstract methods in Python ??
=> we can define abstract methods by using @abstarctmethod decorator.
=> Abstract methods are thoes methods wich is not having any Implimentation.Abstarct methods don't have any body.
=> sty:

```
from abc import ABC , abstractmethod
class ClassName(ABC):

    @abstarctmethod
    def method_name(self):
        pass
```

Q- What is Interface ??
=> In python you can create Interface using Python abstarct class.
=> Interface is a abstract class wich contains only abstract methods called as Inetrface.

NOTE:

1: Abstarct Class can conatins abstract methods as well as non-abstract methods.
2: Interface class conatins only abstarct methods.
3: Can not create object of Inetrface or Abstract class

eg:

```
from abc import ABC , abstractmethod

class RBI(ABC):

   @abstractmethod
   def account(self):
      pass

   @abstractmethod
   def loan(self):
```

```
        pass

    @abstractmethod
    def min_balance(self):
        pass

    @abstractmethod
    def repayment(self):
        pass


class SBI(RBI):

    def account(self):
        print("To wrok with sbi first open bank account")

    def loan(self):
        print("Will provide loan with minimum rate")

    def min_balance(self):
        print("in your account at least 500 rs must be there")

    def repayment(self):
        print("To get loan you have to submit 20 blank checks")



class BOI(RBI):

    def account(self):
        print("To wrok with sbi first open bank account")

    def loan(self):
        print("Will provide loan with minimum rate")

    def min_balance(self):
        print("in your account at least 500 rs must be there")

    def repayment(self):
        print("To get loan you have to submit 20 blank checks")


class CBI(RBI):

    def account(self):
        print("To wrok with sbi first open bank account")

    def loan(self):
        print("Will provide loan with minimum rate")

    def min_balance(self):
        print("in your account at least 500 rs must be there")
```

```
    def repayment(self):
        print("To get loan you have to submit 20 blank checks")

    def debit_card(self):
        print("we will provide you debit card")


cbi = CBI()
cbi.account()
cbi.loan()
cbi.debit_card()
```

#Encapsulation :->

- Wrapping up data and methods together in one entity(class) is called Encapsulation.
- You can hide the internal state of the object from outside. This is knwn as info hiding

Problem :

```
class Counter:

    def __init__(self):

        self.count = 0

    def increment(self):
        self.count += 1

    def disp_count(self):
        return self.count

    def reset_count(self):
        self.count = 0


c1 = Counter()
print('Count value is',c1.disp_count())
c1.increment()
c1.increment()
c1.increment()
c1.increment()
print('Count value is',c1.disp_count())
c1.count = 777
print('Count value is',c1.disp_count())
```

---

private variable :->

Q. What is private variable ??

=> A variable which can be accessed only whithin the class (class member).

=> we can not access private variables outside using object.

Q. How to create private variable ??
=> Just preceed variable with two underscores

eg: __count

print(c1._Counter__count)

There is no pure encapsulation in Python

Making Methods Private :
- Just preced method name with two underscores

eg:

```
class Counter:

    def __init__(self):

        self.__count = 0

    def increment(self):
        self.__count += 1

    def disp_count(self):
        self.__reset_count()
        return self.__count

    def __reset_count(self): # Private method
        self.__count = 0


c1 = Counter()
print('Count value is',c1.disp_count())
c1.increment()
c1.increment()
c1.increment()
c1.increment()
print('Count value is',c1.disp_count())
c1.__reset_count()
print('Count value is',c1.disp_count())
```

WAP for student management system using OOP
db = []
db.append(objects)
student objects
class Student()
attributes = Name, Roll Number, Marks

db = []

```
class Student:

    def __init__(self, nm, roll, mk):
        self.name   =nm
        self.roll   =roll
        self.marks  =mk

    def add_student(self):
        u_name = input("Enter the name of student: ")
        u_roll = input("Enter the roll number of student: ")
        u_marks = input("Enter the marks of student: ")

        s2 = Student(u_name,u_roll,u_marks)
        db.append(s2)
        print("Student added in database successfully")

    def get_student_info(self, ob):
        print(f"Student name is : {ob.name}")
        print(f"Student Roll number is : {ob.roll}")
        print(f"Student marks are : {ob.marks}")

    def search_student(self):
        roll = int(input("Enter the roll number of student: "))
        for i in range(len(db)):
            if db[i].roll == roll:
                print("Student fount in db")
                return i

    def delete_student(self):
        i = obj.search_student()
        del db[i]
        if len(db) == 0:
            print("database is empty")

    def update_student_info(self):
        i = obj.search_student()

        u_name = input("Enter the updated name of student: ")
        u_roll = input("Enter the updated roll number of student: ")
        u_marks = input("Enter the updated marks of student: ")

        db[i].name = u_name
        db[i].roll = u_roll
        db[i].marks = u_marks

        print('Student record updated successfully')


obj = Student(",0,0)
s1 = Student('Pavan', 101, 77)
db.append(s1)
```

```
while 1:
    print("Welcome to Student Management System")
    print("""
                1. Add
                2. Display
                3. Update
                4. delete
                5. Search student
                6. exit

            """)
    ch = int(input("Enter your choice: "))

    if ch == 1:

        obj.add_student()

    elif ch == 2:
        for i in range(len(db)):
            if len(db) == 0:
                print("No student in DB")
            else:
                obj.get_student_info(db[i])

    elif ch == 3:
        obj.update_student_info()
    elif ch == 4:

        obj.delete_student()

    elif ch == 5:
        if len(db) == 0:
            print("No student in DB")
        else:
            i = obj.search_student()

    elif ch == 6:
        break
    else:
        print("Invalid Input for choices.")

    choice = input("Do you want to contiinue y/n : ")
    choice = choice.lower()

    if choice != 'y':
        break
```

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

 # File Handling In Python

Q- Writing data to the file ?
=> There are two methods to write data to the file .

1: write():
-> write method will take string as a input.
-> sty:
fd.write(str)

2: writelines() :
-> writelines method will take input as a list of lines as string.

Q- How to read data from the file ??
=> There are 4 methods.

1: read() :
-> To read total data from the file.
-> sty:
var_name = fd.read()

2: read(n) :
-> To read first n characters from the file we can use read n method.
-> sty:
fd.read(n)

3: readline():
-> to read only one line we can use readline() method
-> sty:
fd.readline()

4: readlines():
-> To read all lines into a list.
-> sty:
fd.readlines()

```
================================================================================
====================================
# WAP to write data to plian text file .
fd  = open("student.txt" , 'a')


name    = input("Enter Student Name: ")
marks   = input("Enter Student Marks: ")
roll_num = input("Enter Student ROll Number: ")


fd.write(f'Student name is {name} Marks are {marks} and roll number is {roll_num} \n')

print(f'\n Student {name} is added successfully to the database')

fd.close()
```

```
-------------------------------------------------------------------------------
fd  = open("student.txt" , 'a')


name    = input("Enter Student Name: ")
marks   = input("Enter Student Marks: ")
roll_num = input("Enter Student ROll Number: ")

l = [name , "~" , marks , "~" , roll_num , "~" , "\n"]

fd.writelines(l)

fd.close()
```

==========

```
# WAP to read data from a plain text file.


fd  = open("student.txt")

data  = fd.read()

print(data)
---------------------------------------
fd  = open("student.txt")

s = " "

while s:
   s = fd.readline()

   print(s)


fd.close()
---------------------------------------
fd  = open("student.txt")

data  = fd.readlines()

print(data)
print(data[0])

fd.close()
```

==========

 # With function :=>

```
        # File Handling in Python using with function.

        # When we use with function then there is no need of closing the file.
```

sty:
    with open(filename , mode) as fd :
        operations

WAP that will read one text file and write that data to the other text file.

==========
Project
Project on Employee management system using file handling and functional programming

Employee data save and read Employee data.

==========
 # Execption Handling :=>

    # Exception :=>
        -> unwanted , unexpetced event wich distured the normal flow of prog is called execption.
        -> Exeption is nothing but Error wich occures at run time.
        -> At the time of execption prog terminates Abnormally. (AT)
        -> In program there is no Error and my total prog excuted without Error called normal termination. (NT)

Execption Handling :->
    defning the alternative way to continue my rest of the prog so that at the end will get NT.

4 imporatant Keywords in Exception Handling.
    1: try
    2: except
    3: else
    4: finally

    5: raise => To define our own error.

There are two types of execptions:
    1: predefine / inbuilt exceptions :
        ZeroDivisionError , TypeError , ValueError , KeyError , IoError , EOFError , ... , etc

    2: coustomize / user define exception

1: try :

-> To write risky code.
-> The code which may raise an error that code is risky code.
-> The code which may raise an error that code we have to write inside a try block.

2: except :
-> Error handling code.
-> If there is Error in try then and then only except block will execute.
-> when we use try block it is mandetory to use except block.

3: else :
-> If there is no error in try then execute else.

4: finally :
-> This will execute always.
-> Irrespective or error it will exectute.
-> clean-up activity.

==========
sty1 :=>

```
try:
    riscky code

except ExecptionName:
    error handling code
    NOTE :-> ExceptioName is optional.
```

---
sty2 :=>

```
try:
    riscky code

except ExecptionName:
    error handling code
    NOTE :-> ExceptioName is optional.

else:
    group of statement that executed if there is no error in try.
```

---
sty3 :=>

```
try:
    riscky code

except ExecptionName:
    error handling code
    NOTE :-> ExceptioName is optional.

else:
```

group of statement that executed if there is no error in try.

finally:
clean-up code

==========

eg:

```
print("Good Evening")


print("Hey")

try:
   print(10 / 0)

except ZeroDivisionError as e:
   print(e) # division by zero

print("Hii")


print("Bye")
```

==========

eg:

```
try :
   print("This is try")
   print(10 /0)


finally:
   print('This is finally')
```

==========

```
try :
   print("This is try")
   print(10 / 0)


except:
   print("This is except")


else:
   print("No Error in try ---> i'm else block ")


finally:
   print('This is finally')
```

==========

```
try :
   print("This is outer try")
```

```
    print(10 / 0)

    try :
        print("This is inner try")
        int("10.5")

    except (ZeroDivisionError , ValueError , TypeError):
        print("This is inner except")


except Exception:
    print("This is outer except")
```

==========

eg:

```
try:

    age = int(input("Enter your age : "))


except ValueError as e:
    print(f"{type(e).__name__ }: {e} ")


else:

    if 18 < age < 50:
        print("You can play game !")

    else:
        print("You can not play game !")
```

==========

eg:

```
try:
    fd = open('test.txt' , 'r')

except FileNotFoundError:
    print("Specify the correct file path")


try:
    # file io operations
    print("This is try")
    data = fd.read(3)
    print(fd.tell())
    print(data)
    fd.seek(0)
    data = fd.read()
    print(data)
```

```
finally:
   # closing of file
   fd.close()
   print("This is finally block")
================================================================================
# customized Execption / user define execptions:=>

# One python ExceptionClass is equals to one python exception(Error)
NOTE :
        We can raise Error by using raise keyword.

# sty to define execption in python:

        class ClassName(Exception):
                def __init__(self , msg):
                        self.msg = msg


# sty to raise an execption in python :
        raise classname("msg")


NOTE :=>
        1-> python class must be inherite by parent Exception class.
        2-> self.msg is a predefine variable wich is used to print msg on console.
        3-> When python Exception class will execute ...??
                => If we create object of python exception and and i will raise object with some msg.

================================================================================
class TooBigException(Exception):

   def __init__(self , msg):
      self.msg = msg

class TooSmallException(Exception):

   def __init__(self , msg):
      self.msg = msg


age = int(input("Enter your age: "))

if 0 < age < 18 :
   raise TooSmallException("Your Age Is To small To Play Game !")


elif age > 50:
   raise TooBigException("your Age Is To Big To play Game !")


else:
   print("Welcome To Game World ... Please wait Game is Loading ... !")
```

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

class IncorrectUsernamePasswordException(Exception):
    def __init__(self , msg):
        self.msg = msg


print("Login To Python Project World.com")

username = input("Enter User Name: ")

password = input("Enter password: ")


if username == "sai@gmail.com" and password == "sai3333":
    print("Welcome To Python Project World...")

else:
    raise IncorrectUsernamePasswordException("Please Enter correct username and password")
================================================================================
############################## Python Test ####################################

Q- WAP for swapping values of two variable
        NOTE :
                -> Do not use Temporaroy variable


Q- Count digit of the given number.
        eg:
                # i/p :-> a = 123
                # o/p :-> total number of digit's are 3


Q- reverse the given number
        eg:
                # i/p :->  123
                # o/p :->  321


Q- Find factorial of given number using for loop.


Q- WAP to calculate BMI weight(kg) and height(m).

================================================================================
# Menu Driven Prog for Student Management System using Dict and class.


db = {}

class Student(object):
```

```
def __init__(self , name , marks , roll_num):
    self.name       = name
    self.marks      = marks
    self.roll_num   = roll_num

def save(self):
    db[self.name] = self

def display(self):
    print(f'Student Name is {self.name} Student marks are {self.marks} and roll number is {self.roll_num}')


@classmethod
def show_all_students(cls):
    if db:
        for x in db:
            obj = db.get(x)
            obj.display()
            print('--'*50)
    else:
        print("Database is Empty")


@classmethod
def get_student_data(cls , name):
    if name is db:
        obj = db.get(name)
        print(obj)
        obj.display()
    else:
        print(f'Student {name} object is not found')


@classmethod
def delete_student(cls , name):
    if name in db:
        del db[name]
    else:
        print(f"student {name} is not fount to delete")

@classmethod
def update_student_data(cls , name):
    if name in db:
        obj  = db.get(name)
        obj.name    = input("Enter Updated Student Name: ")
        obj.marks    = input("Enter Updated Student marks: ")
        obj.roll_num    = input("Enter Updated Student roll number: ")

    else:
        print(f"student {name} is not fount to update")
```

```
while True:

    print("""

            1) List of Student
            2) Add Student
            3) Update Student
            4) Delete Student

    """)

    choice = input("Enter your choice: ")

    if choice == '1':
        Student.show_all_students()

    elif choice == '2':
        name        = input("Enter  Student Name: ")
        marks       = eval(input("Enter  Student marks: "))
        roll_num    = eval(input("Enter  Student roll number: "))
        obj = Student(name , marks , roll_num)
        obj.save()

    elif choice == '3':
        name        = input("Enter Student Name To Update: ")
        Student.update_student_data(name)

    elif choice == '4':
        name        = input("Enter Student Name To Delete: ")
        Student.delete_student(name)

    ch = input("Do you want to continue the prog [yes / no]: ")

    if ch.lower() == 'no':
        break
```

==========

What is HTML?

HTML stands for Hyper Text Markup Language

HTML is the standard markup language for creating Web pages

HTML describes the structure of a Web page

HTML consists of a series of elements

Eg:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

- The <!DOCTYPE html> declaration defines that this document is an HTML5 document
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the HTML page
- The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The <h1> element defines a large heading
- The element defines a paragraph

What is an HTML Element?

An HTML element is defined by a start tag, some content, and an end tag:

<tagname> Content goes here... </tagname>

The HTML **element** is everything from the start tag to the end tag:

<h1>My First Heading</h1>

My first paragraph.

HTML Headings

HTML headings are defined with the <h1> to <h6> tags.

<h1> defines the most important heading. <h6> defines the least important heading:

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

HTML Paragraphs

HTML paragraphs are defined with the <p> tag:

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

HTML Links

HTML links are defined with the <a> tag:

```
<a href="https://www.google.com">This is a link</a>
```

HTML is Not Case Sensitive

HTML tags are not case sensitive: <P> means the same as <p>.

HTML Styles

The HTML style attribute is used to add styles to an element, such as color, font, size, and more.

The HTML Style Attribute

Setting the style of an HTML element, can be done with the style attribute.

Sty: <*tagname* style="*property:value;*">

Background Color

The CSS background-color property defines the background color for an HTML element.

Sty:
```
<body style="background-color:powderblue;">

  <h1>This is a heading</h1>
  <p>This is a paragraph.</p>

  </body>
```

Eg2:

```
<body>

<h1 style="background-color:powderblue;">This is a heading</h1>
<p style="background-color:tomato;">This is a paragraph.</p>

</body>
```

==========

What is CSS?

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

Why to Use CSS?

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

CSS Syntax

A CSS rule consists of a selector and a declaration block.

Selector {property: value;}

Key Points

- Selector is HTML element to which CSS rule is applied.
- Property specifies the attribute that you want to change corresponding to the selector.
- Property can take specified value.
- Property and Value are separated by a colon (:).
- Each declaration is separated by semi colon (;).

Following are examples of CSS rules:

```
P { color : red;}

h1 (color : green; font-style : italic }

body { color : cyan; font-family : Arial; font- style : 16pt}
```

Types of CSS :

CSS can be added to HTML documents in 3 ways:

- **Inline** - by using the style attribute inside HTML elements
- **Internal** - by using a <style> element in the <head> section
- **External** - by using a <link> element to link to an external CSS file

Inline Style Sheets

Inline Style Sheets are included with HTML element i.e. they are placed inline with the element. To add inline CSS, we have to declare style attribute which can contain any CSS property.

Syntax:

```
<Tagname STYLE = " Declaration1 ; Declaration2 ">  …. </Tagname>
```

Internal CSS

An internal CSS is used to define a style for a single HTML page.

An internal CSS is defined in the <head> section of an HTML page, within a <style> element.

The following example sets the text color of ALL the <h1> elements (on that page) to blue, and the text color of ALL the <p> elements to red. In addition, the page will be displayed with a "powderblue" background color:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: powderblue;}
h1   {color: blue;}
p    {color: red;}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

External CSS

An external style sheet is used to define the style for many HTML pages.

To use an external style sheet, add a link to it in the <head> section of each HTML page:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
```

```
<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

==========

Bootstrap 4

What is Bootstrap?

- Bootstrap is a free front-end framework for faster and easier web development
- Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins
- Bootstrap also gives you the ability to easily create responsive designs

Why Use Bootstrap?

Advantages of Bootstrap:

- **Easy to use:** Anybody with just basic knowledge of HTML and CSS can start using Bootstrap
- **Responsive features:** Bootstrap's responsive CSS adjusts to phones, tablets, and desktops
- **Mobile-first approach:** In Bootstrap, mobile-first styles are part of the core framework
- **Browser compatibility:** Bootstrap 4 is compatible with all modern browsers (Chrome, Firefox, Internet Explorer 10+, Edge, Safari, and Opera)

==========

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database

- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Some of The Most Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Example

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

The SQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

SELECT Syntax

SELECT *column1*, *column2*, ...FROM *table_name*;

SELECT * FROM *table_name*;

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

UPDATE *table_name*
SET *column1 = value1*, *column2 = value2*, ...
WHERE *condition*;

The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

DELETE FROM *table_name* WHERE *condition*;

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table