Languages :

Language is used for Communication.

Classification of Languages :

In terms of Language / Computer we have two types of Languages

Low Level Language → these are understandable by System

High Level Language → these are understandable by Users

To Convert HLL to LLL we use

Compilers [translates Entire Soure code in a Single Run]

Interpreters [translates Entire Source code line by line]

Based on type of translators we use High level languages into two types

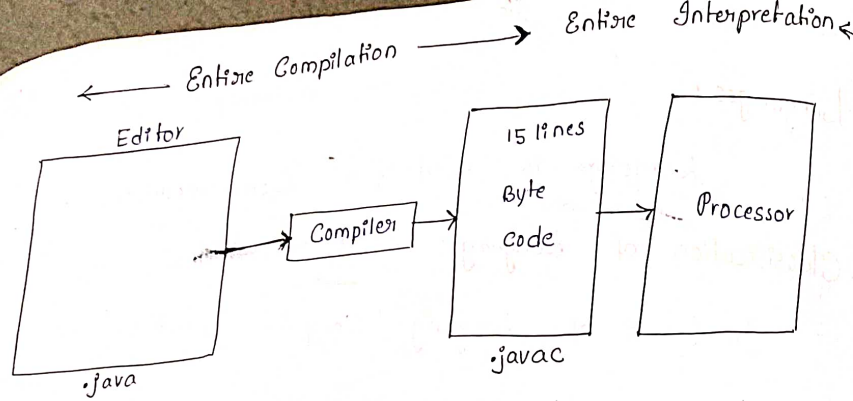
① Programming Language

② Scripting Language

Programming Languages

1. In Programming Language we will use Compiler
2. Entire set of Instructions will be Converted at a time into Binary format and entire Converted Code will be executed at one time

Examples : C C++ Java

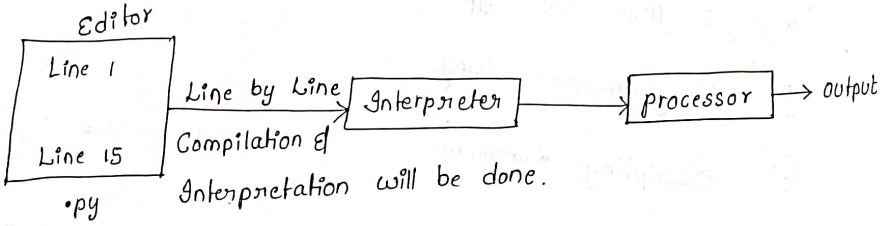


Scripting Language :

① In Case of Scripting Language we will be using Interpreter

② In Scripting Language Line by Line Compilation and Interpretation will take place.

Example : Python, JavaScript.



21Dec2022

Definition of Python / Features of python :

1. Python is High level Open Source Interpreted Scripting Language
2. Python is Dynamic typed language [there is no need of declaration of variable datatype]
3. Python Supports Functional Programming and Object Oriented Programming [↑ code reusability and ↓ code redundancy]
4. Python has more built in libraries.
5. Python is Case Sensitive Language { Code we write in Django }
6. Python is Easy to understand and Learn

Fields in which we use Python :

1. Web Development.
2. Artificial Intelligence
3. Data Science
4. Machine Learning
5. IOT (Internet of things)
6. Automation etc

During Runtime, the interpreter Converts these instructions into Readable machine code that a processor can Execute.

Scripting Language :

Is a computer programming Language that provides instructions called scripts for software in apps and websites. Scripts Contain series of commands that a software, application can scripting engine interpret one at a time within a Runtime Environment

→ How to Install Python Software :

- Search for Python.org in Browser
- Click on downloads - Click on download Python 3 one Python exe file will be downloaded
- Double click on the exe file
- Before clicking on Install Now please check the Add python path checkbox
- Now Click on Install Now

* Editors of Python :

1. Pycharm
2. Vscode
3. Jupiter Notebook
4. Sublime text

* Python Syllabus *

1.1 Introduction to Python.

- Installation and Environment Setting
- Introduction to shell

1.2 Variables, Keywords, Data types and Identifiers

- Variables
- Keywords
- Datatypes
- Identifiers

1.3 String, List, set, Tuple and Dictionary and Slicing

- String Data types
- List Data types
- Set Data types
- Tuple Data types
- Dictionary Data types
- Slicing

1.4 Operators

- Arithmetic Operators
- Logical Operators
- Relational Operators
- Bitwise Operators
- Assignment Operators
- Membership Operators
- Identity Operators

1.5 Control Statements :

- Decisional
- Looping
- Break Continue and Pass

1.6 Input and Print Statements :

- Input Statements
- Print Statements.

1.7 Functions (or) Methods :

- Types of function
- Recursion
- Arguments
- Packing and Unpacking (Varargs)

1.8 Oops :

Class, Objects

Inheritance

Method, Overriding

Access Specifies

1.9 File Handling and Json :

- Flat File handling
- Json
- Pickle

1.10 Exception on Handling :

Try.

Except and Finally

Custom Exceptions

Raising Exceptions

1 Assertions

1.11 Comprehension

List Comprehension

1.12 Decorators

Method

Class level

1.13 Map Filter and Lambda Expressions

1.14 Iterators and Generators.

Steps to Install pycharm : Search for Below Mentioned url

https://www.jetbrains.com/pycharm/

Click on download

Click on community Version

After Installation of exe.file double click on it and install with default Setting

IDLE (IDE):

IDLE stands for Integrated Development and Learning Environment

IDLE is Environment Equipped with all the default implementations of python Language.

We can Utilize the IDLE in two ways

1. Python Shell
   1. It is an interactive console
   2. we can executive only one statement at a time in python shell
   3. In python shell we can't Save [no need to enter print]

2. Python Module,:

It is File with .py extension where we can execute multiple python statements.

Steps to Create module from Shell :

Open IDLE shell

Click on new and click on new file (ctrl+N)

Save the file without giving extension

(Because we are creating it in python shell)

Command for Executing python module

press F5

Syntax for Executing python module Through command prompt

python modulename.py

(or)

py modulename.py

6/12/2022

Tokens : Tokens are Essential Elements for writing the Python Program

Tokens of Python :

1. Keywords
2. Variables
3. Identifiers
4. Datatypes

Keywords : Keywords are reserved / Built in Words which are defined for doing some task

2. True, None, False are title Case Remaining Keywords in lower case.
3. We can't change the functionality of Keywords
4. In python we have 35 Keywords

5. To see List of Keywords follow the below Command.

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', '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']
>>> len(keyword.kwlist)
    ↳function
35
```

Import Keyword :

1. Import Keyword is used for performing the process importing.
2. Importing is the process of accessing the Content of another module into Current module.

Variables :

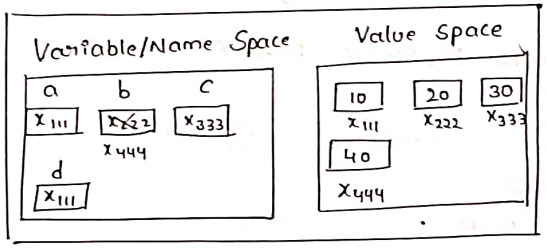
1. Variable is a name given to the Memory allocation (address)
2. The Value store in a variable might get Varied (change)

Syntax : To Declare a Variable (Single)

Variablename = Value.

(Right to Left)

```
a = 10 ←
b = 20 ←
c = 30 ←
d = 10 ←
b = 40 ←
```



X_{111} = address

Id = address of a variable [>>> id(a)]

Note = If we assign multiple variables with same integer values then they will point to same memory address

If we assign new value for a existing variable then it will put to new one memory address

Garbage Collection :

It is used for deleting memory of un-used values

Drawbacks of Garbage collection when it is done by developers :

1. Deleting the memory before completion of its usage
2. Not deleting the memory even after Completion of its usage.

PMM : (PYTHON MEMORY MANAGEMENT)

1. In python Garbage Collection will be done by PMM
1. PMM will delete the Value when it has Zero references (Reference Count)

Id Function :

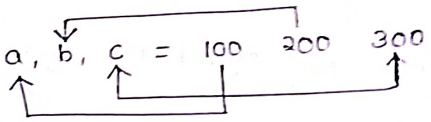
It is used for getting the Address of Variable

Syntax : id(variablename)

Ex : id(a)

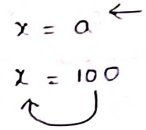
Defining Multiple Variables :

Syntax : Varname1, Varname2, ... VarnameN = Value1, Value2, ... ValueN



Defining New Variable with Value of Existing Variable.

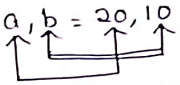
Syntax : NewVariableName = Existing Variable



Swapping the Values of a Variable.

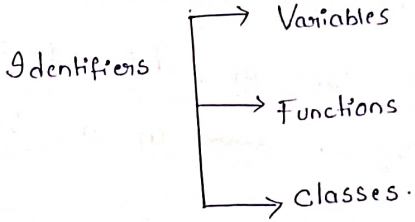
Syntax : Var1, Var2 = Var2, Var1

Ex : a, b = b, a



08IDec2022

Identifiers : Identifiers is a name with which we can identify Variables, functions (or) classes.



Rules to be followed while declaring the Variables

1. We should not use Keyword as Identifier
2. The first character should not be numeric Value
3. Other than underscore we should not use any other Special character (Because every other Special character has some functionality in python)
4. We can use combination of Alphabets, numbers and underscore
5. Standards of Identifiers :

   Variable → use only lower case

   Functions → one word → lowercase

   Two word → first word lower case and remaining words in title Case

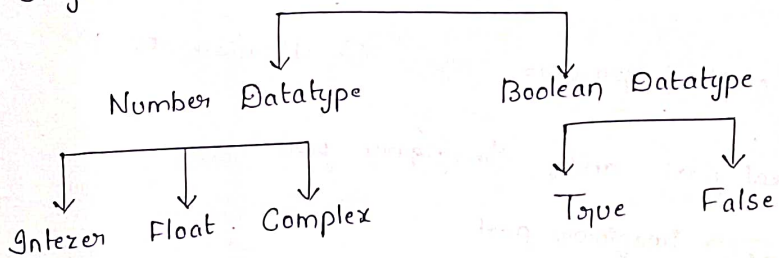
   Classes → Title Case.

Datatypes : Datatypes are used for Defining the type of data that we are going to store in a Variable.

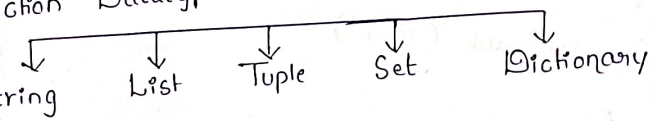
Classification of Datatypes.

Based on number of Values stored in a Variable.

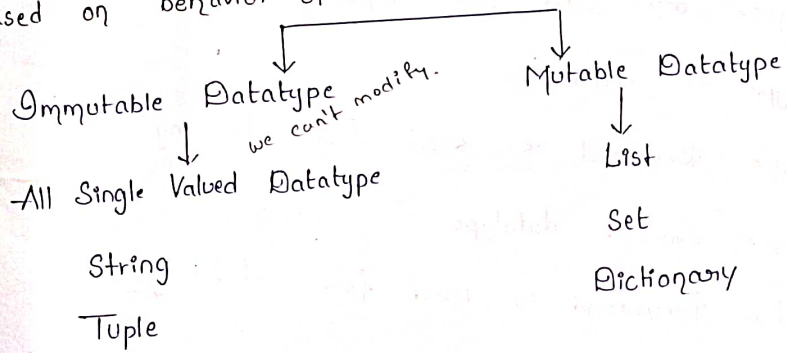
Single Valued Datatype



Collection Datatype



Based on behavior of Values stored in a Variable



1Dec

Single Valued Datatype :

We Can store only one single data in Variable.

Int : The whole numbers without decimal points

Float : The whole numbers with decimal points.

Complex : Complex numbers are combination of Real part and Imaginary part

$A + Bj$ → imaginary part
↳ real part

j - constant ($\sqrt{-1}$)

Boolean Datatype :

These are used for defining yes (or) no type of data

True Keyword is used for defining True Boolean datatype

False Keyword is used for defining False Boolean datatype.

type function :

It is used to define the type of data stored in a Variable.

Syntax : type (variable)

Note : 1. Every thing in Python is considered as an object [Real time entity]

2. We can store only one value (or) one memory address at a time in ~~such~~ sub memory address

```
>> a = 10
>> type (a)
<class 'int'>
>> b = 9.9
>> type (b)
<class 'float'>
c = a + bj
type (c)
<class 'complex'>
d = True
type (d)
<class 'bool'>
e = False
type (e)
<class = 'bool'>
```

Multi-Valued (or) Collection Datatype :

In this category we will store more than one Data item (or) value.

Inorder to bound elements into one collection we have use boundaries.

1. If we store multiple values inside the memory will be divided into sub memory address.
2. We can Identify sub memory address using Index positions.

In python we have Positive and Negative index position.

Positive Index Position :

Direction : Left to Right

Range : 0 to $n-1$

where n = len (Collection)

↓

(Length of Elements)

Negative Index Positions :

Direction : Right to Left

Range : -1 to $-n$

Classification of CDT (Collection Data type)

1. Ordered CDT
2. Unordered / Random CDT

1. Ordered CDT : In ordered CDT, in memory data will be stored in the same order as that we have defined in Variable

Ex : String (' ', " ", ''' ''')

List [ ]

Tuple ( )

Dictionary { }

2. Unordered CDT : In unordered CDT data will be stored in random order.

6. → 1, 12 & 3

Ex : Set

String :

1. String is a Collection of individual elements which are Enclosed in a pair of quotation marks.
2. String is Immutable Datatype
3. String is Ordered Data type
4. Indexing and Slicing is Possible

→ For representing single line strings we use single (or) double quotes

→ For representing Multi line strings we must use triple quotes

Syntax for declaring single line string Datatype

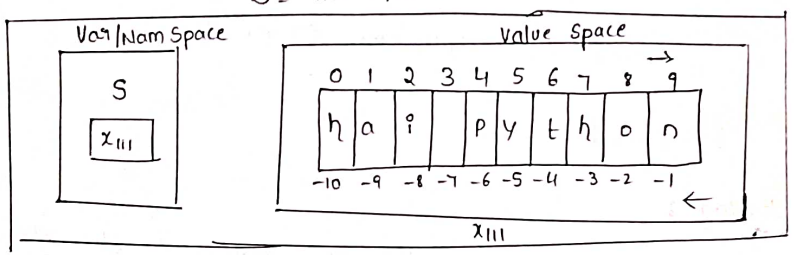
Variablename = 'element1 element2 ... elementn'

(or) "element1 element2 ... elementn"

Syntax for declaring multi line string Datatype

Variablename = '''element1 element2 ...
... elementn'''

S = 'hai Python'



12|Dec|2022

Indexing :

It is a process of Extracting single element from a given collection. by using index positions.

Syntax for performing Indexing

Variablename [index-position]

we can give both positive and negative index positions But internally Negative indexes are converting into positive indexe positions value only.

Ex : S[-5] = S[5] = 'y'

>> S = 'hai Python' >> S[3] >> S[-7]

' ' ' '

>> S[1] >> S[-5]

'a' 'y'

>> S[4] >> S[-3]

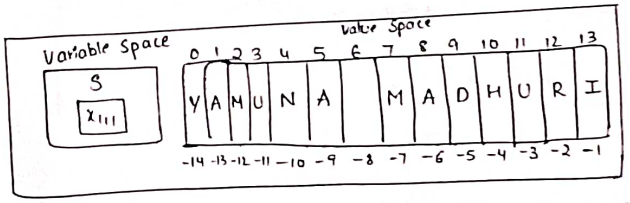
'p' 'h'

>> S[8] >> S[-1]

'o' 'n'

Performing Indexing for the string 'YAMUNA MADHURI' :

S = 'YAMUNA MADHURI'



S[5] = 'A' S[9] = 'D' S[11] = 'U' S[6] = ' ' S[13] = 'I'

S[-14] = 'Y' S[-8] = ' ' S[-5] = 'D' S[-2] = 'R' S[-1] = 'I'

Slicing :

It is a process of Extracting Multiple Elements from a given collection

We can perform Slicing in two ways

1. positive slicing
2. negative slicing

Positive Slicing : Process of Extracting multiple elements in the Left to Right direction

Syntax : variablename [startindex : endindex+1 : updation]
↓
difference

While performing positive slicing interpreter check for below condition

Ex S[4 : 9 : 1] S = 'hai Python'

SIP < EIP

d1 < 9 => True >> 'PYTHO' [output]

4+1 = 5 < 9 => True

5+1 = 6 < 9 => True

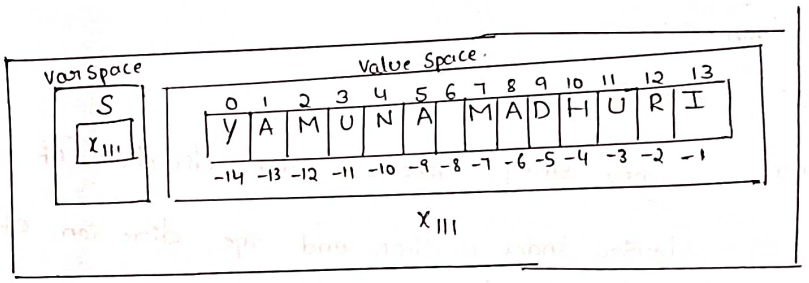
6+1 = 7 < 9 => True

7+1 = 8 < 9 => True

8+1 = 9 < 9 => False (stops)

```
>>> S[4:9:1]
'PYTHO'
>>> S[4:10:1]
'PYTHON'
>>> S[0:3:1]
'hai'
>>> S[1:8:1]
'ai pyth'
>>> S[0:9:2]
'hipto'
>>> S[1:9:7]
'ao'
>>> S[1:8:7]
'a'
>>> S[8:4:2]
'' [Empty string]
```

Positive slicing for the variable S = 'Yamuna Madhuri'



While performing positive slicing interpreter checks the following conditions.

S = 'yamuna Madhuri'

Ex S[4:9:1]

>>> 'na ma'

S[5:10:1]

'a mad'

Start Index Position < END Index position

$5^{+1} < 10 \Rightarrow T$

$6^{+1} < 10 \Rightarrow T$

$7 < 10 \Rightarrow T$

$8 < 10 \Rightarrow T$

$9 < 10 \Rightarrow T$

$10 < 10 \Rightarrow F$ (Stops)

output >>> 'a mad'

>>> S[0:10:2]

'ymn a'

>>> S[10:5:1]

'' Empty string because the direction of Starting Index position and the direction of updation are in opposite direction so the output will be Empty string with no space

>> S[6:12:3]

'd'

>> S[3:10:3]

'u d'

>> S[2:12:4]

'm h'

>> S[9:13:4]

'd'

In case of Positive slicing Default values for

Starting index = 0

Ending index = len(collection)

Updation = +1

>> S[4:10:] >> S[:] >> S[::]

'python' 'python' 'hai python'

Note: If the specified index position is more than the length then

In Indexing - Error

In slicing - it will Extract till the end of the String

Ex: S[10]

Index Error: String index out of range

>> S[2:19:1]

' i python'

Negative Slicing :

It is the process of Extracting multiple Elements in the right to Left direction

Syntax for negative slicing :

Variablename [startindex : endindex-1 : updation]

In Negative Slicing Default values for

Starting index = -1

Ending index = -(len(collection)+1)

For Updation there is no default value, we need to provide in Negative values

Note : It is mandatory to pass updation value in case of Negative slicing (in Negative Values only)

S = 'hai Python'

S[-1:-8:-1]

'typ'

S[-4:-8:] [It is mandatory to provide updation value]

' '

S[-11::-1] [-(n+1)] default value of EIP

'typ iah'

s[::-1] [reverse format]

'nohtyp iah'

List Datatype :

List is a Collection of both Homogenous (Elements of same data type) and Heterogenous Data in which each and every Element is separated by , operator and enclosed in the pair of [ ] (square brackets)

[may not be of small datatype]

List is a Mutable Data type

List is a ordered CDT

In List Indexing and slicing is possible.

Syntax of Declaring List data type

variablename = [element1, element 2, ..., element n]

Ex

>> L = [123, 11, 24, 66, 77]

>> L[2] → 24

>> L[-4] → 11

>> L[1:4:] → [11, 24, 66]

>> L[2::] → [24, 66, 77]

>> L[4] → 77

>> L[-2:-5:-2] → [66, 11]

>> L[::-2] → [77, 24, 123]

Behavior of CDT

Immutable Datatype :

In case of immutable Datatype we cannot modify its value space

Mutable Datatype :

In case of mutable Datatype we can modify its value space

Syntax for Modifying value space by Indexing

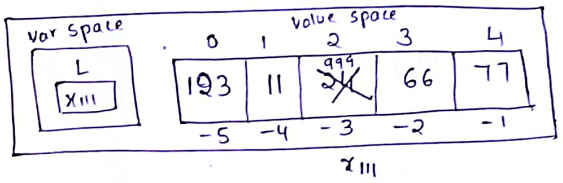
Variablename [index position] = New_value

Syntax for deleting value space by indexing

del variablename [index_position]

Note : String is immutable DT we cannot modify Value Space

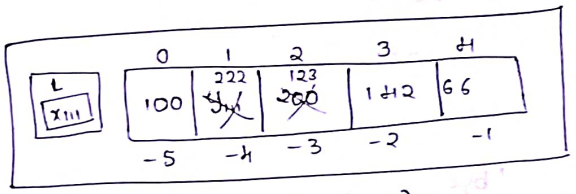
```
>> L[2] = 999
>> L
   [123, 11, 999, 66, 77]
>>  del S[2]
    'str' does not support item deletion
>>  del L[3]
>>  L
   [123, 11, 999, 77]
```

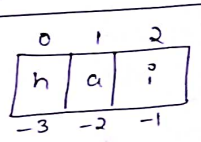


x111

3Dec2022

```
>> L = [100, 'hai', 200, 142, 66]   L = [100, 'hai', 200, 142, 66]
>> L[3]
   142
>> L[1]
   'hai'
>> L[1][1]
   'a'
>> L[1][::2]
   'hi'
>> L[1][::-2]
   'ih'
>> L[2] = 123
>> L
   [100, 'hai', 123, 142, 66]
>> L[1][1] = 'i'
   'str object does not support'
>> L[1] = 222
>> L
```





y111

```
[100, 222, 123, 1212, 66]

>> L = [89, 67, 'bye', 'Hello', 99]

>> L[2][1]

'y'

>> L[2][1] = 'e'

Error

>> L[2]

'bye'

>> L[2] = 999

>> L

[89, 67, 999, 'hello', 99]

>>> L[3][4]

'o'

>> L[3][4] = 'e'

Error

>> L[3] =

'hello'

>> L[3] = 888

>> L

[89, 67, 999, 888, 99]
```

```
L = [22, 'hai', [44, 66], ['Ashu', 'nikky']]

L[3][0]

'Ashu'

L[3][0][:3:]

'Ash'

L[3][1][:-5:-1]

'ykki'

L[3][0] = 'mahi'

L

[22, 'hai', [44, 66], ['mahi', 'nikky']]

> L[2][1] = 'Ashu'

> L

[22, 'hai', [44, 'Ashu'], ['mahi', 'nikky']]

> del L[2][0]

-> L
```

[22, 'hai', ['Ashu'], ['mahi', 'nikky']]

>> L[3][0][1:3:]

'ah'

>> L[2][0][0:3]

'Ash'

>> L

[22, 'hai', ['Ashu'], ['mahi', 'nikky']]

>> L[2]

['Ashu']

>> L[2][0]

'Ashu'

>> L[2][0][0:3:]

'Ash'

19|Dec|2022

Syntax for Modifying the value space by Slicing

Variablename[index_position] = CDT

[Collection data type]

[Ex: 'abc', ['abc']]

L = [11, 22, 33, 44, 55]

L[1:4:]

[22, 33, 44]

L[1:4:] = 66 * [we have to provide CDT]

Error only can assign iterable [Error occured because we have given Single Valued datatype]

> L[2] = 'hai'

>> L

[11, 22, 'hai', 44, 55]

> L[1:4:] = 'bye'

> L

[11, 'b', 'y', 'e', 55]

> L[3::] = ['ashu', 'nikky']

>> L

[11, 'b', 'y', 'ashu', 'nikky']

> L[:3:] = ['mahi']

> L

['mahi', 'nikky']

>> L[0:1:1] = ['a', 'b', 'c', 'd']

>> L[1:3:] = [66]

[11, 66, 'ashu', 'nikky']

```
>> L
['a', 'b', 'c', 'd', 'nikky']

>> L[1:4:] = [['ashu','nikky'], 29]

>> L
['a', ['ashu','nikky'], 29, 'nikky']

>> L[0:3:] = 'xyz'

L ['x', 'y', 'z', 'nikky']

>> L[0:3:] = 'pqrst'

>> L
['p', 'q', 'r', 's', 't', 'nikky']
```

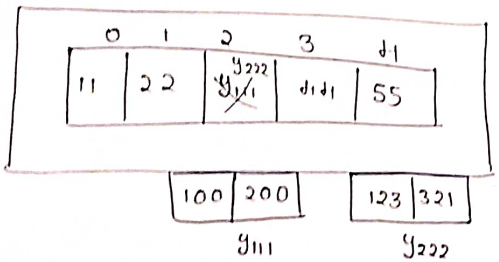
L = [11, 22, [100, 200], 44, 55]

to

L = [11, 22, [123, 321], 44, 55]

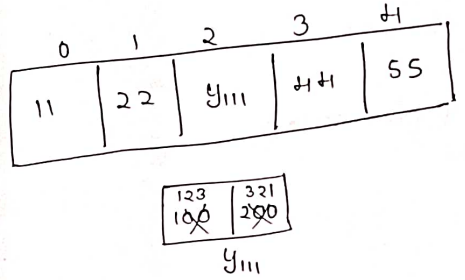
process : L[2] = [123, 321]

Indexing [Completely deletes the list and replace new list]



② L[2][::] = [123, 321]

slicing [changes the behavior instead of deleting the List]



2/2/2022:

Tuple Data Type :

1. Tuple is a Collection of Both Homogenous and Heterogenous Data in which each and every element is separated by, operator and enclosed in the pair of ( ) parenthesis
2. Tuple is a Immutable CDT
3. Tuple is a Ordered CDT
4. Comma operator defines the Tuple not parenthesis (optional)
5. Both Indexing and Slicing can be performed

Syntax for declaring multiple values in Tuple

Variablename = (element1, element2 .... elementn) or

Variablename = element1, element2, ..., elementn

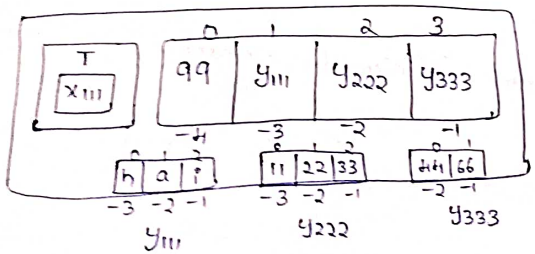
Syntax for Defining Single Value

Variablename = element,

Variablename = (element,)

```
>> T = 12,
>> T
(12, )
>> T = 11, 22, 33, 44, 55
>> type(T)
<class 'tuple'>
>> T[2]
33
>> T[4]
55
>> T[-4]
22
>> T[-2:-5:-2]
(44, 22)
>> T = (99, 'hai', [11,22,33], (44,66))
```

[Indexing and Slicing Syntax are given in [ ] brackets irrespective of Str, List and Tuple]




```
>> T[1][1]
'a'
>> T[2][1]
22
>> T[2][1] = 88
```

```
>> T
(99, 'hai', [11, 88, 33], (44, 66))
```

[We can modify the value space of List inside a Tuple (Immutable) but we can't handle modify the Value space of Tuple inside a List (Mutable)]

List	Set	Tuple
In List data is stored between [ ] brackets.	• In set data is Enclosed in flower braces { }	• In Tuple data is Enclosed in ( ) (parenthesis)
List is a Ordered Collection Data type	• Set is a Random Ordered Datatype	• Tuple is a Ordered CDT
List is a Mutable DT	• Set is a Mutable DT	• Tuple is a Immutable DT
Each Element is Separated by comma (,)	• Each Element is Separated by (,)	• Tuple is defined by comma (,) operator
Indexing and Slicing Possible	• Indexing and Slicing not possible	• Indexing and Slicing is Possible
Any type of data we can Store	• We can't Store Mutable data.	• Any type of data can be Stored.

Set Data Type :

1. Set is a Collection of Both homogenous and heteorogenous Data in which each and every element is separated by (,) operator and enclosed in the pair of { }
2. Set does not allow Duplicates
3. Set is a Random order typed Data type
4. Set is a Random ordered we cannot perform Indexing and Slicing.
5. Set is a Mutable Data type
6. In set we can store only Immutable datatype

Syntax for declaring set :

Variablename = { element 1 , element 2 , ... element n }

Syntax for declaring Empty set * (Later)

Variablename = set { }

```
>> S = { 22, 66, 90, 12, 89 }
>> S
{ 66, 22, 89, 90, 12 }
>> S = {'hai', 90, 67, 89.8 }
>> S
{ 89.8, 90, 67, 'hai' }
>> S = { 90, (90), 78, 56 }
>> S
>> { 56, 90, 78 }
>> S = { 90 (90,), 78, 56 }
>> S
{ (90), 56, 90, 78 }
```

Dictionary Data Type :

1. Dictionary is a Collection of Key and Value pairs
2. In dictionary Key and values are enclosed in pair of { }
3. Each and Every Key, Value pairs are Separated

by using (,) operator

4. Each and Every Keys and values are separated by using : operator

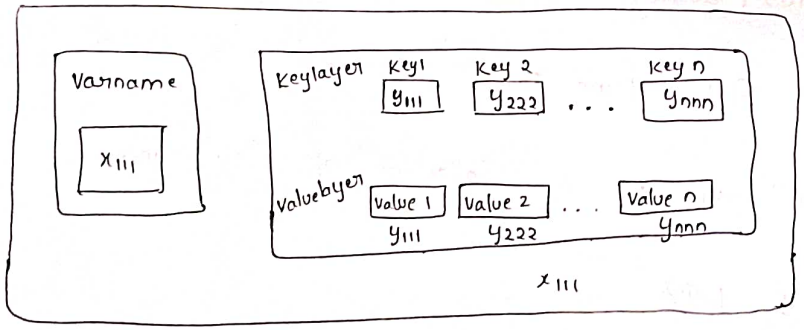
5. Dictionary is a Mutable Data Type

6 As the data is in the form of pairs we Cannot perform Indexing and Slicing

Syntax :

Variablename = {" Keyname1": Value1 , 'Keyname2' : Value2, 'Keyname n' : Value n }

Eg: d= {'name':'Ashu', 'age' : '2' }



Syntax for accessing values from a given dictionary

Variablename ["keyname"]

Syntax for modifying values from a given Dictionary

Variablename ["Keyname"] = Newvalue

Note :

If specified key is present it will update value Else it will create a New key Value pair in given dictionary.

Syntax for deleting element from a given Dictionary

del Variablename [keyname]

Properties for Keys in Dictionary :

1. We can use immutable data types as keys but it is advised to use only strings as keys Because keys are used for defining the values

2 Keys are Case Sensitive

3 If we declare Duplicate Keys then the recent values will be assigned to that key.

Properties of Values.

- Any type of Data

We can use Duplicate values as values in Dictionary.

```
>> d = { 'name' : 'Ashu' , 'age' : 2 }

>> d
{ 'name' : 'Ashu' , 'age' : 2 }

>> d['name']
'Ashu'

>> d['age']
2

>> d['mobile']
Error

>> d['Ashu']
Key Error

>> d['age'] = 3

>> d
{ 'name' : 'Ashu' , 'age' : 3 }
```

→ d['name'][0]

'A'

→ d['name'][::3]

'AU'

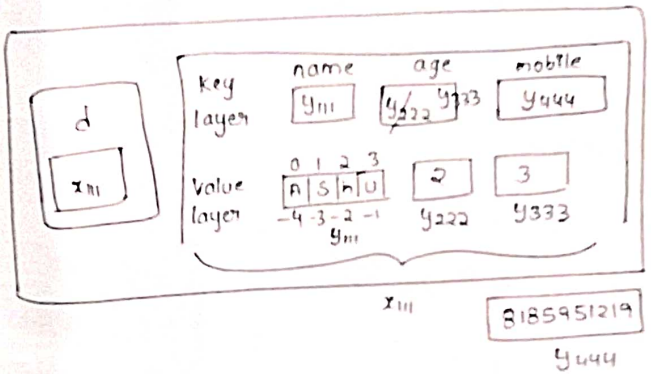
→ d['name'][::-1]

'uhsA'

→ d['0']

Errors

```
>> d['mobile'] = 8185951219

>> d
{ 'name' : 'Ashu' , 'age' : 3 , 'mobile' : 8185951219 }
```




```
>> d
{ 'name' : 'Ashu' , 'age' : 3 , 'mobile' : 8185951219 }
```

Indexing and Slicing are not performed on dict

Indexing and Slicing are performed on value Space of string

i.e., d['name'][::3] = 'AU'

d['0'] = Error.

```
d = { 'name' : 'Ashu' , 'age' : 27 , 'mobile' : { 8185951219,
      7676330226 } , achievements : {'cricket' : 1 , 'fb' : 2 } }

>> d ['achievments']
{ 'cric' : 1 , 'fb' : 2 }

>> d ['achievments'] ['cricket']
'

>> d['achievments'] ['cricket'] = 2

>>
>> d
{ 'name' : 'Ashu' , 'age' : 2 , 'mobile' : { 8185951219, 767633022
  'achievments' : { 'cric' : 2 , 'fb' : 2} }

>> d [ 'achievments'] ['cric'] = d ['achievments'] ['cric'] + 1

>> d
{'name' : 'Ashu' , 'age' : 2 , 'mobile' : { 8185, 7676 } ,
  'achievments' : {'cric' : 3 , 'fb' : 2} }.
```

Syntax for Defining Empty collection Data type

String → ' ' , " " , ''' ''' , str( )

List → [ ] , list( )

Tuple → tuple( )

Set → set( )

Dictionary → Dic( ) , { }

⇒
```
d = { }
type(d)
< class 'dict' >
```

26|Dec|2022

Type Casting : It is the process of Converting one type of data into another type of data

' First the given collection is defined and after that only it will be given for Respective DT for type Casting '.

' It will throw Errors only in Case of defining ' / and

Into which datatype	from which DT	Syntax
Integer	float, boolean, string Containing only int	int (value / Variable)
Float	integer, boolean, string Containing both int and float	float (Value / Variable)
Complex	integer, float, boolean, string Containing both int and float, Complex.	Complex (Value/ Variable)
Boolean	All data type's (It will only give false in case of '0 and Empty CDT's and remaining Scenarios it will give 'T')	bool (value/ Variable)
String	All Datatype's	Str (value / variable)
List	String, Tuple, Set, Dict (In dictionary it will consider only Keys)	list (value/ Variable)
Tuple	String, List, Set, Dictionary (only keys)	tuple (value/variable)
Set	String, List, Tuple, Dictionary (only Keys)	Set (value / variable)
Dict	List, Tuple, Set (Data Should be in form of pairs) (In set we should only use tuple for representing the pairs)	dict (value / variable)

>> List ((23, 35), 45, 56))

[(23, 35), 45, 56]

>> len (list ((23, 56), 45, 56)) / >> len (str ([11, 22, 33]))

=> 3 ↓ 12 ↓

List, set, Tuple calucates the Elements / In string it Calculates Each and Every Element

Difference between Functions and ~~Classes~~ Methods.

Functions

① It is independent block of code which is defined for doing some task

② def functionname( )

block of code

③ functionname( )

(Directly by their names)

Directly defined

Ex : len( )

↓

len is a function

Methods

① It is also a function which is defined inside a class

② Class className( )

def Methodname( )

Block of code

③ Methods can be accessed only by their class names (or) object names

* ClassReference . Methodname( )

(or)

* objectname . Methodname( )

↓

when we are dealing with objects.

Built-in Methods of Strings :

Note :

** All the methods can perform some Operations on strings but they will not Modify actual value of string (because it is Immutable)

Case Conversion Method of strings

Method Name	Syntax of Method	Functionality
lower( )	s.lower( )	It is used for Converting Each and Every character of the string to lowercase
upper( )	s.upper( )	It is used for Converting Each and Every character of the string to Upper
title( )	s.title( )	It is used for converting first character of the Each and Every word of a given string will be Converted into upper and remaining characters into lower
Capitalize( )	s.capitalize( )	Only first character of Entire String will be Converted into upper and remaining all lower
Swapcase( )	s.swapcase( )	It is used for Converting all Lower case to upper Case

and all upper case characters to lower case

Examples : S = 'HAi PYThon'

* Here S is the objectnames

```
>>> s.lower( )
'hai python'
>> S. upper ( )
'HAI PYTHON'
>> S. title( )
'Hai Python'
>> S. capitalize( )
'Hai python'
>> S. swapcase( )
'hAI pytHON'
```

islower( ) — S. islower( ) — Returns True if string satisfies lower method else returns false

isupper( ) — S. isupper( ) — Returns True if string satisfies upper method (or) else false

istitle( ) — S.istitle( ) — Returns True if string satisfies title method else returns false.

* [if we give is before the method it means it gives boolean datatype value]

Ex:

```
S = 'hai python'
>> S. islower( )
True
>> S. isupper( )
False
>> S1= 'HAI'
>> S. islower( )
False
>> S. isupper( )
True
>> S2 = 'Hai Python'
>> S2 istitle( )
True
>> S2. isupper( )
False
```

isalpha( )	S.isalpha( )	Returns True if string contains only alphabets else false
isdigit( )	S.isdigit( )	Returns True if string contains only digits else returns false
isalnum( )	S.isalnum( )	Returns True if string contains only alphabets, only digits, or combinations of alphabets and numbers Else false .
isspace( )	S.isspace( )	Returns True if string contains only space else returns false

Examples :

```
>> S = 'hai Python'
>> S.isalpha( )
False [ Because there is a space ]
>> S = 'hai'
>> S.isalpha( )
True
>> S = '123 567'
>> S.isdigit( )
False ( space )
>> S = '123567'
>> S.isdigit( )
True
>> S1 = 'hai123'
>> S1.isalnum( )
True
>> S2 = 'hai'
>> S2.isalnum( )
True
>> S1 = 'hai123'
>> S1 = 'hai 123'
>> S1.isalnum( )
False
>> S1 = ' '
>> S1.isspace( )
True
```

Arguments : Arguments are Essential parameters required for function / Method for its Execution.

Mandatory Argument directnames /positional

Default Argument [names] / Non mandatory

Split( ) rsplit( )

Used for splitting the given string based on given Delimator is Repeated same

Output format of split is a List — Same

Syntax:

split([delimator], [count]) — Same

Default Values

for delimator = Space

for count = how many times ur delimator is Repeated

Direction: Left to Right — Right to Left

Examples

```
>> S = 'hai Python'
>> S.split('y')
['hai p', 'thon']   [ delimator before value
                      delimator after value ]
```

⇒ S.split('a')

```
['h', 'i python']
```

⇒ S.split('h')

```
[' ', 'ai pyt', 'on']   [ There is no value before delimator so it is empty ]
```

```
S1 = 'harshad vali'
>> S1.split('a')
['h', 'rsh', 'd v', 'li']
>> S1.split('i')
['harshad val', '']   Nothing after the delimator so it is '' [Empty]
>> S1.split('a', 2)
['h', 'rsh', 'd vali']
>> S = 'hai Python'
>> S.split()   ⇐ [By default it takes spaces as a delimator]
['hai', 'python']
>> S.split('h')
['', 'ai pyt', 'on']
>> * S.rsplit('h')
['', 'ai pyt', 'on']
```

Method	Syntax	functionality
replace()	replace(oldstring, Newstring, [count]) Default values for count = no. of times ur old string is repeated	Used to replacing Existing Characters with new characters

24/Oct

Count() count(Substring, [start_index], [end_index]) It is used for counting how many times given substring is repeated in given string.

```
>> S = 'hai python'
>> S.count('h')
   2
>> S.count('i')
   1
>> S.count('w')
   0
>> S.count('h', 1)
   1
>> S.count('h', 1, 7)   (End index position before value)
   0
>> S.count('h', 1, 8)
>> S.count('a', 1, 8)   (It will Consider the starting index position as '1' only)
   1
```

index()

Syntax('value', [start index], [end index])

Used for returning the index position of a substring, if value is present (or) else it returns Error

Direction: Left to Right

rindex()

functionality is same as index but Execution direction will be Right to left

find()

Syntax:

('value', [start index], [end index])

It is used for returning index position if value is present else it returns -1 as output

Direction: Left to Right

rfind()

functionality is same as find but Execution direction will be Right to Left

```
s.index('h')
0
s.index('p')
4
s.index('h',1)
7
s.index('ha')   ('h' index position is 0)
0
s.index('th')
6
s.index('w')
Error: (Substring not found)
```

```
>> s.rindex('h')
7
>> s.rindex('w')
Substring not found
```

```
>> s.find('h')
0
>> s.find('h',1)
7
>> s.find('w')
-1
[w is not available in string so it displays -1]
```

```
>> s.rfind('h')
7
>> s.rfind('w')
-1
```

Startswith :

Used for checking whether the string is Starting with specified substring (or) not

If it is starting it will return true Else it returns False

[check start Index position]

Syntax :

startswith (substring, [start_index], [end_index])

Endswith : Used for checking whether the string is Ending with the specified substring (or) not, if it is ending it will return True Else it returns false

Syntax : endswith (substring, [start_index], [end_index])

[Check End Index position]

```
>> s = 'hai python'
>> s.startswith('h')
True
>> s.startswith('a')
False
>> s.startswith('h',7)
True
```

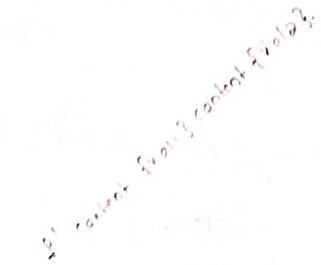
```
>> s.startswith('h',7,7)
>> False   [End Index position, start Index position are same, it's Empty sh
>> s.endswith('h')
>> True
>> s.endswith('no')
False
>> s.endswith('on')
True
>> s.endswith('on',5)
True
>> s.endswith('on',5,9)
False
>> s.endswith('o',5,9)
True
```

format()

'content{} content{}'.format(value1, value2)

f'content{val1}content{val2}'

- Used for Creating Dynamic Strings
- We need to create place holders
- placeholders are created by using {}

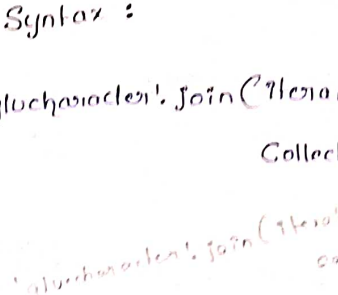


```
>> 'this is {} and his/her age is {}'.format('nikky',20)
'this is Nikky and his/her age is 20'
>> 'this is {1} and his/her age is {0}'.format(20,'nikky')
'this is Nikky and his/her age is 20'
>> 'this is {n} and his/her age is {a}'.format(a=10,
                                                n='nikky')
'this is Nikky and his/her age is 10'
>> a=3
>> n='Ashu'
>> f'this is {n} and his/her age is {a}'
'this is Ashu and his/her age is 3'
>> s=f'this is {n} and his/her age is {a}'
>> s
'this is Ashu and his/her age is 3'.
```

Join()

Syntax:

'glucharacter'.join(iterable/Collection)



Functionality

Used for creating new string by joining the elements by using glue characte

- By using join method we can form a sentence
- In collection DT we have give only strings as an Elements
- Glue character can be given anything but it should be given in the form of string i.e., '#', ' ', '*',

Examples:

```
>> '#'.join('hai')          '*'.join('hai python')
'h#a#i'                     'h*a*i* *p*y*t*h*o*
                                                n'
```

```
>> '#'.join('hai 123')
'h#a#i# #1#2#3'
>> '#'.join(['hai', 'bye', 'see', 'you', '89'])
'hai#bye#see#you#89'
>> '#'.join({'hai': 'bye', 'see': 99})
'hai#see'
>> ' '.join(['hai', 'bye', 'see', 'you', '89'])
'hai bye see you 89'
```

Input

```
S = 'want to be different'        (or)
                                   *
                                   => ' '.join(S.split()[::-1])
L = S.split()                                  'different be to want'
  = ['want', 'to', 'be', 'different']
K = L[::-1]
  ['different', 'be', 'to', 'want']
' '.join(K)
  ['different be to want']
```

Method

strip(), rstrip(), lstrip()

Syntax

s.strip(), s.rstrip(), s.lstrip()

-functionality

→ These methods are used for removing the Leading Spaces from given strip

→ lstrip removes Space from Left Side

→ rstrip removes Space from Right side

```
>> s = ' hai python '        >> s.lstrip()
>> s.strip()                    'hai python '
'hai python'                 >> s.rstrip()
                                ' hai python'
```

Examples

* List Built in Methods *

Method name	Syntax	-functionality
count()	count(value)	Used for counting how many times a given element is Repeated
index()	index(value, [st], [ed])	Used for finding the index position of given Element

* Insertion Methods of List *

append	extend	insert
③ ① append(value)	extend(value)	insert (index position, ② argument value)
Value → SVDT / CDT	Value → CDT	value → SVDT / CDT
It is used to add Element into the List at the end	It is also used to add the Elements at the end of the List	It is used to add Elements into the List at Specific IP (will not update)
If we give CDT it will be considered as one Element [If we give two arguments it will give Type Errors]	First Elements of CDT will be extracted and added at the end	* CDT is considered as one Element * If Specified I.P > len the Elem will be added at

```
>> L = [89, 67, 15]
>> L
[89, 67, 15]
>> L.append(90)
>> L
[89, 67, 15, 90]
>> L.append(77)
>> L
[89, 67, 15, 90, 77]
>> L.append('hai')
>> L
[89, 67, 15, 90, 77, 'hai']
>> L = [89, 67, 15]
>> L.insert(1, 100)
>> L
[89, 100, 67, 15]
```

```
>> L.append((55,44))
>> L
[89, 67, 15, 90, 77, 'hai', (55,44)]
>> L.extend('bye')
>> L
[89, 67, 15, 90, 77, 'hai', (55,44), 'b', 'y', 'e']
>> L = [1, 2, 3]
>> L.extend(['hello', 'bye'])
>> L
[1, 2, 3, 'hello', 'bye']
>> L.insert(2, 'ashu')
[89, 100, 'ashu', 67, 15]
>> L.insert(98760, 789)
[89, 100, 'ashu', 67, 15, 789]
```

[As the index position is beyond the range so it will be inserted at the end of the List]

→ Deletion Methods of List ←

pop( )

→ Syntax :

pop(index - position)

↓

non- mandatory

→ Default value is -1

→ Based on given Index position pop will delete the Element

→ If specified Index position is more than length it will throw Error(Index Error)

[l.pop returns removed value]

Examples :

```
          0     1     2
L = [ [26, 23, 22], 5, 'hello' ]

>> L.pop(0)
>> [26, 23, 22]
>> L
   [5, 'hello']
>> L.pop(1)
   'hello'
>> L
```

remove( )

→ Syntax :

remove(value)

↓

mandatory

→ Based on given value it will delete an Element

→ If specified value is not present it will throw Error(value Error)

[l.remove doesn't returns removed value]

```
>> [5]
>> L.pop(2)
Error  [ Specified Index position is not available ]
>> L = [89, 99, 100]
>>   L.remove(89)
>>   L
     [99, 100]
>> L.append(100)
>> L
   [100, 99, 100]
>> L.remove(100)  [ only the first occurence will be
>> L                deleted ]
   [99, 100]      [ In order to delete more than one
                    we have to perform for loop ]
```

pop(0p) remove(value)

Clear( )	Clear( )	Used for deleting all the Elements of given list but not memory
Copy( )	Copy( )	we can achieve shallow copy. shallow copy is the process of copying only the Elements. If you modify one variable then other will not get affected
= (operator)	var1 = var2	we can achieve deep copy deep copy is the process of copying the memory address. If you modify one variable then other will also be affected.
Sort( )	Sort[reverse=False]	It will arrange the Elements of given list in Ascending Order
Sort(reverse=True)		It will arrange the Elements of given list in Descending Order

Difference between sort and Sorted

Sort	Sorted
Method of List	It is normal Built in Function
Sort method is used only with List Data type	Sorted can be used on any Collection DT
Syntax: Ascending order object reference.sort( )	Syntax: Ascending order Sorted (collection)
Syntax: Descending order object.reference.sort (reverse=True)	Syntax: Descending order Sorted (Collection, reverse=True)
Output format is a List	Output format is a List
	Note: It will not modify CDT

Ex:

```
L = [11, 22, 33, 99]
L1 = [98, 78, 67, 56]
del L
>> L1.clear( )
L1[ ]
A = L.copy( )
B = L
L [11, 22, 33]
A [11, 22, 33]
B [11, 22, 33]
```

Built in methods of Tuple Data type

Index( )

Count( )

Functionality of Tuple index( ) and count( ) methods is same as of in list index( ) and count( ) methods

Built in methods of Set

Copy( )

Clear( )

→ Copy( ) : Used for performing shallow copy

→ Clear( ) : Used for deleting all the elements of a given set

Examples :

```
S = {11, 22, 33, 44}
>> S1 = S.copy( )
S {33, 11, 44, 22}
>> S1 {33, 11, 44, 22}
id(S)
2159211572448
id(S1)
2159211562144
```

```
>> S1.clear( )
>> S1
set( )   Empty Set
>> S
{33, 22, 44, 11}
```

Insertion Methods of ~~List~~ Set

add( ) : Used for adding Elements into set randomly

Syntax : add(value)

Note : list, set, dictionary should not be used as Value

Deletion Methods of set → [Elements are called as Keys]

pop( )	remove( )	discard( )
pop( )	remove(value)	discard(value)
→ It is used for deleting the first value from the defined set	Used for deleting the Elements based on given value	Used for deleting the Elements based on given Value
→ If will not take any argument	→ If Value is present then remove( ) will delete that element	If Value is present then discard( ) will delete that element
	Else it will throw Error ↓ Key Error	Else it will not perform any Operation.

31 Jan 2024

union( )

Syntax :

baseset.union(set1, set2..)

It is used for getting all Elements from Specified sets.

intersection( )

Syntax :

baseset.intersection (set1, set2 ...)

It is used for getting only common Elements from specified sets.

difference( )

Syntax :

baseset.difference (set1, set2 ...)

It is used for getting uncommon Elements from baseset

Note : By Above Operations any of the specified sets will not get modified.

Update( )

Syntax :

baseset.update(set1, set2...)

It will perform ~~uniform~~ union Operation and update the Output into the Baseset

Intersection-update( )

Syntax :

baseset.intersection-update (set1, set2 ...)

It will perform Intersection Operation and update the output into Baseset

difference_update()

Syntax :

Baseset.difference_update (set1, set2 ...)

It will perform difference operation and update the output into baseset

S = {11, 22, 33, HH, 55}

S1 = {22, 90, 67}

S2 = {33, 19, 12, 89}

S.union(S1, S2)

{33, 67, 11, 99, HH, 12, 22, 55, 89, 90}

S.intersection(S1, S2) (No common Elements)

set()

S.intersection(S1)

{22}

S.difference(S1)

{33, 11, HH, 55}

S.difference(S)

{90, 67}

Symmetric_difference_update()

Syntax :

Baseset.symmetric_difference_update(set1)

It will perform symmetric difference operation and update the output into the Baseset

S.update(S1)

S

{33, 67, 22, 55, 90, 11, HH}

>> S1.intersection_update(S2)

>> S1

set()

>> S.intersection_update(S2)

>> S2

>> {33}

>> S = {11, 22, 33, HH, 55}

>> S1 = {22, 90, 67}

>> S.difference_update(S1, S2)

>> S

{55, 11, HH}

>> S.Symmetric_difference_update(S1,

* Error.

↓

Symmetric difference updates takes Exactly one argument

>> S.Symmetric_difference_update(S1)

>> S

{67, 22, 55, 90, 11, HH}

issuperset()

Syntax :

Baseset.issuperset(set1)

Returns true if the Base set is superset of Specified set else False

[which means it should contain all the items and or more elements]

issubset()

Syntax:

Baseset.issubset(set1)

Returns true if the Baseset is subset of Specified set else False.

isdisjoint()

Syntax :

Baseset.isdisjoint(set

Returns true if Specified set has uncommon Elements else it returns false

```
>> S = {11, 22, 33, 44, 55}
>> S1 = {11, 33, 44}
>> S2 = {22, 55, 66}
>> S.issuperset(S1)
   True
>> S.issuperset(S2)
   False
>> S.issuperset(S1, S2)
   Error (issuperset takes only one argument)
>> S1.issubset(S)
   True
>> S2.issubset(S)
   False  [ Because there is a common Element 22]
   A = {89, 78, 67}
   B = {89, 78, 67}
>> A.issuperset(B)
   True
>> A.issubset(B)
   True
```

```
>> X = {11, 22, 33}
   Y = {22, 89, 67}
   Z = {44, 55, 66}
>> X.isdisjoint(Z)
   True
>> Y.isdisjoint(Z)
   True
>> X.isdisjoint(Y)
   False  (22 is common)
```

Built-in Methods of Dictionary

copy( ) → Used for performing shallow copy

clear( ) → Clear all the Elements from given Dictionary

Examples :
>> ① d = {'name': 'Ashu', 'age': 2}
② d1 = d.copy( )
>> ③ d = {'name': 'Ashu', 'age': 2}
>> ④ d1 = {'name': 'Ashu', 'age': 2}
>> d1.clear( )
>> d1
{ }

keys( )	values( )	items( )
Syntax: keys( )	Syntax: values( )	Syntax: items( )
Used for getting the keys of Specified Dictionary	Used for getting the values of a Specified Dictionary	Used for getting both keys and values of specified dictionary.

Examples : d = {'name': 'madhu', 'age': 28}

>> d.keys( )
dict_keys(['name', 'age']) → dictionary objects.

d.values( )

dict_values (['madhu', 28])

d. items ( )

dict_~~values~~ items ([('name', 'Ashu'), ('age', 3)])

~~Jan~~

get ( )

Syntax :

get (keyname, [Default value])

If Specified key is present then Get will display the value of specified key

Else

Get will display the default value

setdefault ( )

Syntax :

setdefault (keyname, [Default value])

If specified key is present then setdefault will display the value of specified key

Else

setdefault will update the dictionary with specified key and default value

» d = {'name' : 'Ashu', 'age' : 3}

» d. get ('name')

'Ashu'

» d. get ('name', 'Nikky')

'Ashu'

» d. get ('NAME', 'Nikky')

'Nikky'

» d

{'name' : 'Ashu', 'age' : 3}

» d. setdefault ('name')

'Ashu'

d. setdefault ('name', 'Nikky')

'Ashu'

d. setdefault ('Name', 'Nikky')

'Nikky'

d

{'name' : 'Ashu', 'age' : 3, 'Name' : 'Nikky'}

d. setdefault ('Name', 'Mahi')

'Nikky'

d. setdefault ('mobile')

d

{'name' : 'Ashu', 'age' : 3, 'Name' : 'Nikky', 'mobile' : None}

↓

[Keyword]

Update method ( ) :

It is used for updating the actual dictionary with multiple keys and value pairs at a time

Note :

It is also used for merging of two dictionaries

Syntax :

Base dictionary . update ({'key1' : 'value1', 'key2' : 'value2'...}

- If specified key is present then it will update the value
- If key is not present then it will Create a New Value pair.

```
» d = {'name' : 'chaitu', 'age' : 7 }
» d
  {'name' : 'chaitu', 'age' : 7 }
  d. update ( {'age' : 8 , 'mobile' : 8191716151 } )
  d
  {'name' : 'chaitu', 'age' : 8 , 'mobile' : 8191716151 }
  d. update ( {'mobile' : 1234156789 10 , 'gender' : 'female'} ).
  d
{'name' : 'chaitu', 'age' : 8, 'mobile' : 1234156789 10 ,
        'gender' : 'female'}
```

Deletion Methods :

pop( )	popitem( )
Syntax :	Syntax :
• pop (keyname)	• popitem( )
• It is used for deleting the keyvalue pairs based on specified key	• Used for deleting the last key and value pair by default
• If key is present then it will delete else it will throw me an Error	• Output format Tuple
	• If dictionary is empty it throws an Error.

```
» d
  {'name' : 'chaitu', 'age' : 8 , 'mobile' : 1234156789 10, 'gender' : female}
» d. pop ('gender')
  ('gender', 'female') 'female'
» d
  {'name' : 'chaitu', 'age' : 8 , 'mobile' : 1234156789 10}
» d. pop ('age')
  8
» d
  {'name' : 'chaitu', 'mobile' : 1234156789 10}
» d. pop ('age')
  Error
```

```
>> d.popitem()
('mobile', 1234156789 10)

>> d
{'name': 'chaitu'}

>> d.popitem()
('name', 'chaitu')

>> d
{}

>> d.popitem()
Error (Dictionary is Empty)
```

Fromkeys :

Syntax : fromkeys(CDT, [defaultvalue])

It is used for extracting each and every element of given CDT and represent them as Keys and default value as a value of that Keys.

Examples :
```
>> {}.fromkeys('hai', 3)
{'h': 3, 'a': 3, 'i': 3}

>> {}.fromkeys(['hai', 'bye', 'hello'], 'wish')
{'hai': 'wish', 'bye': 'wish', 'hello': 'wish'}

>> {}.fromkeys(['hai', 'bye', 'hello'])
{'hai': None, 'bye': None, 'hello': None}

>> {}.fromkeys('hai', [])
{'h': [], 'a': [], 'i': []}

>> d = {}.fromkeys('hai', [])   # stored in variable
>> d
{'h': [], 'a': [], 'i': []}
```

5/Jan/2022

* Operators. *

Operators are Elements which are used for performing operations b/w Operands

Arithmetic operators (+, -, *, /, %, //, **)

Relational / Comparison (>, <, <=, >=, ==, !=)

Logical (and, or, not)

Assignment (=, +=, -=, *=, ... **=)

Bitwise (&, |, ^, ~, >>, <<)

Membership (in, not in)

Identity (is, is not)

Arithmetic Operators :

Operator	Number to Number	Number to CDT	CDT to CDT
+	Addition	Cannot be Performed	Concatenation Sets
-	Substraction	Cannot be Performed	
*	Multiplication	Concatenation [String, List, Tuple]	Cannot be performed

» [0] * 3 , 'ha' * 3 , [12, 34, 78] * 2

[0, 0, 0] 'hahaha' [12, 34, 78, 12, 34, 78]

* [Concatenation is the process of joining two strings]

/ (Normal (or) True Division) :

Perform division and gives quotient as output in the form of float

5/2 = 2.5

// (floor division)

perform division and eliminates the decimal values from output

5//2 = 2

% (Modular division):

Used for returning remainder

30 % 2 = 1

(Should be of same dT) (String, Lists, Tuples)

(Difference Operation)

** (power operator) : A ** B => A^B (A to the power B)

↓

Exponent

3 ** 3 = 27

Relational Operators :

- Relational Operators are used for returning a Boolean value as an output
- These Relational operators are used in conditional Statements. while loop

  >, >=, ==, <, <=, !=
- Number to Number Comparison we can do it easily
- But when we Compare strings then Comparison is done based on ASCII values of characters
- In case of CDT to CDT it will Compare first Element [Based on ASCII values]

» 2 > 1

True

» 90 > 90

False

» 90 >= 90

True

» 'hello' > 'hi' [Based on ASCII values]

True

» [11, 22, 334] < [10, 89, 67, 5677]

False

» [11, 22, 334] > [10, 89, 67, 5677]

True

>> 22 == 22

True

>> 'hai' == 'hai'

True

Assignment Operators : These are used for assigning some values to a variable.

* By using = operator we can assign value for a variable

A += B → A = A + B

A -= B → A = A - B

A *= B → A = A * B

A **= B → A = A ** B

A /= B → A = A / B

A //= B → A = A // B

A %= B → A = A % B

6 JAN

Logical Operators :

Logical Operators are used for returning a Boolean Value as an output

These Logical Operators are used in Conditional Statements.

Operand1	Operator	Operator2	Result
True	and	True	True
True	and	False	False
False	and	True	False
False	and	False	False

Examples:

>> 2 and 5. [Bool value of ① 2 → T, ② 5 → T]

5

>> 6 and 0 [Bool value of 0 → F]

0

>> '' and 99

''

>> [ ] and { }

[ ]

Or Operator :

Operand1	Operator	Operand2	Result
True	or	True	True
True	or	False	True
False	or	True	True
False	or	False	False

>> 3 or 'hai' [If the first value is True so it will give you directly as T without considering second one]

3

>> [12,34] or ' '

[12, 34]

>> [ ] or 989

989

>> '' or { }

{ }

NOT Operator : Not Operator is known as negation operator.

Syntax : not Operand

Examples : not 7

False

not [ ]

True

Membership Operator : Used for giving checking whether the element is part of given collection or not.

In operator :

It returns True if value is present in given collection else returns false.

Syntax : SVDT / CDT in Collection

Note :

1. Right hand side value must be a CDT
2. If we use string in righthand side then the left hand side value must and should be a string.

Examples :

Not in Operator : It returns True if Value is not present in given collection else returns False

Syntax : SVDT/CDT not in Collection.

Examples :

>> 2 in 123

Error [Righthand side there must be a CollectionDT]

>> 2 in '123'

Error [when the Righthand side is a string the left hand side must and should be a string]

>> '8' in '123'

False

>> '3' in '123'

True

```
» 1 in ['hai', 'hello']
False
» [2] in [1, 7, 89, 2]
False.
» [2] in [1, 7, 89, [2]]
True
» (2) in [1, 7, 89, 2]
True
» (2,) in [1, 7, 89, 2]
False.
» 3 in {'name': 'ASHU', 'age': 3}
False [ In Dictionary it will consider only Keys ]
» 'age' in {'name': 'ASHU', 'age': 3}
True
» 3 not in {'name': 'ASHU', 'age': 3}
True
» 'name' not in {'name': 'ASHU', 'age': 3}
False
```

Identity Operators :

Note points : Integers till 256 will have same memory address

→ If same value of immutable datatypes are stored in variable then they will point to same Memory Address

→ But in case of tuple, even though tuple is a immutable they will not point to same address Bcoz there is chance of storing List inside tuple (mutable)

== operator Compares Value directly

Is, is not operators compares memory address

Identity Operators :

These are the operators used for Comparing Memory address

is operator :

It returns True if Variables are pointing to Same Memory address else returns False

Syntax :

operand1 is operand2

is not : It returns True if Variables are not pointing to same memory address else returns False

Syntax : operand1 is not Operand2

```
>> a = 256
>> b = 256
>> a == b
True
>> a is b
True
>> c = 257
>> d = 257
>> c is d
False
>> e = 2.3
>> f = 2.3
>> e is f
False
>> x = 7+9j
>> y = 7+9j
x is y
False
S = 'hai'
S1 = 'hai'
S is S1
True
L = [11, 22, 33]
l = [11, 22, 33]
L is L1
False
```

```
>> T = (90, 89)
>> T1 = (90, 89)
>> T is T1 [In tuple there is a chance of List so it is False]
False
>> S = {11, 22, 33}
>> S1 = {11, 22, 33}
>> S is not S1
True.
>> d = {'name': 'ashu', 'age': 3}
>> d1 = {'name': 'ashu', 'age': 3}
>> d is not d1
True.
>> d is d1
False
>> a = 90
>> b = 90
>> a is not b
False
```

Bitwise Operators :

These are operators are used for performing Operations on Binary values of given numbers

Bitwise & (and) Returns 1 when both operands are 1 Else 0

Bitwise | (or) Returns 0 when both operands are 0 Else 1

Bitwise ^ (xor) Returns 0 if both operands are same else 1

Bitwise ~ It performs 2's Compliments .....> $\sim n = -(n+1)$

Right shift Shifts binary to right side by specified number of positions

Syntax :

Operand >> No. of positions to shift

left shift : Shifts binary to left side by specified number of positions

Syntax :

Operand << No. of positions to shift

```
>> 2 & 3
2
>> 1 & 3
1
>> 1 & 2
0
>> 1 | 3
3
```

0 → 0 0 0 0
1 → 0 0 0 1
2 → 0 0 1 0
3 → 0 0 1 1
4 → 0 1 0 0

```
>> 1 | 2
   3
>> 2 ^ 3
   1
>> 1 ^ 3
   2
>> 1 ^ 2
   3
>> ~2
   -3
>> ~16
   -17
>> ~-6
   5
>> 2 >> 1
   1
>> 2 << 1
   4
>> 3 >> 1
   1
>> 41 >> 2
   1
```

```
2 >> 1

0 0 1 0 >> 1

0 0 1 0 |

0 0 0 1 → 1
```

```
2 << 1

0 0 1 0 << 1

0 0 0 1 0 |

0 1 0 0 → 4
```

aljas

→ Use of Print function :

Print function is used for printing some Content

Syntax for print function :

print (value1, value2 .... , sep=' ' , end = '\n' file = sys.stdout, flush = False)

[By default]

We can change the values of sep and end

Examples :

```
print (20, 22, 25, sep=' , ' , end = ' ')
print (30, 33, 34)
print (44)
```

O/P :

```
>> 20 , 22 , 25   30  33  34
>> 20 , 22 , 25   30  33  34
   44
```

Commenting : / Comments.

Comments are non- Executable statements.

Comments are used by Developers for providing the hints about their code.

We have two types of comments.

Single line comments (#)

Multi line comments ("'' Triple quotes"')

If we provide Multi line ~~Comments~~ without any Variable (or) functions are referred as multiline Comments with """ Triple quotes """

Examples :

```
print ("Hello world") #To print hello world.
```

Collecting data from the user during runtime

1. input() function is responsible for collecting data from user
2. Output format of collected data will be in String
3. so we can type cast based on our requirement

Eval()

It is a special function which is responsible for Evaluating data into its equivalent Submitted DT

Example 1 :

```
a = int (input ('enter a value : '))
print(a)
print (type(a))
```

Example 2 :

```
a = eval (input ('enter a value :'))
# Evaluating Submitted data
print(a)
print (type(a))
```

Flow Control Statements :

1. In any language the flow of Execution of statements follows water fall Approach (Top to Bottom)
2. Flow Control Statements are the Special Statements which are used for controlling (or) changing the actual flow of Execution

Classification of Flow Control Statements

1. Conditional / Decisional
2. Looping Statements.

Conditional (or) Decisional Statements

In this type Based on one condition we will decide whether to Execute set of instructions (or) not to Execute set of instructions

In python we haves 4 types of Conditional Statements

If Condition

If - Else Condition

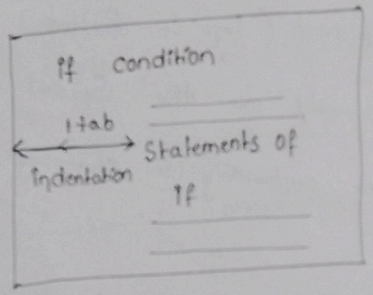
If - Else if (Elif) Condition

Nested if Condition

If Condition :

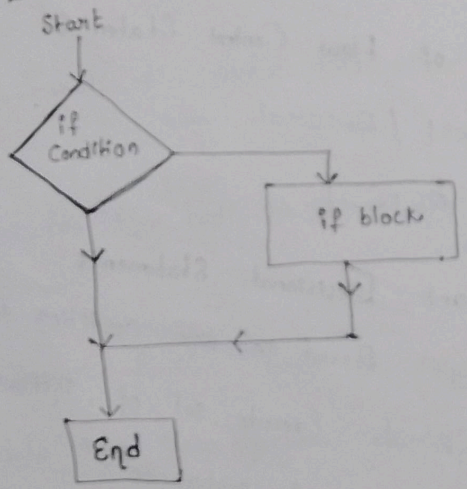
1. When we have only one condition to check then we will use if condition.

Syntax :



→ Indentation is a common space which represents block of code

D →



→ When we want to get out from the block we use back-space and again if we want to Enter we have to provide a 1 tab space

```
print ('start')
A = int (input ('is there any possibility to meet'))
if A==1
    print ('go to her place meet')
print ('end')
```

→ write a program print a number if the given number is more than 100.

```
a = int (input ('enter a number'))
if a>100:
    print (a)
    print (f'given number {a} is greater than {100}')
```

output :

enter a number1234

1234

given number 1234 is greater than 100

→ print the string if the given string is starting with h

```
a= input ('enter a string:')
#1
if a[0].lower( )=='h':
    print (a)
#2
if a.lower( ).startswith('h'):
    print (a)
```

Q: Print a string if the given string is having more than 3 words?

→ #①

```
S = input('Enter a string')
L = S.split()
if len(L) > 3
    print(S)
```

#②

```
S = input('Enter a string')
if s.count(' ') > 2
    print(S)
```

10th Jan

If Else Condition: When we have two conditions to check then we will use if-Else Condition.

Among 2 Conditions we will have

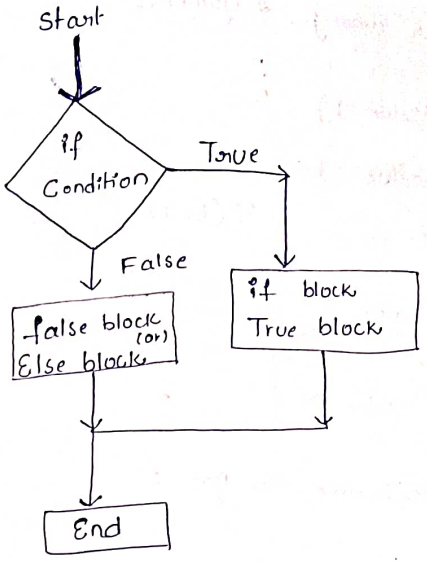
→ Mandatory

→ Default

Note: we can choose any condition as mandatory the other will be the default.

Syntax:

```
if condition:
    1tab indentation  Statements of if-block
Else:
    1tab indentation  Statements of Else-block
```



Ex:

```
print('start')
a=input('asking GF can we meet or not')
if a=='yes':
    print('meet GF')
Else:
    print('meet Ex GF')
print('end')
```

Ex: WAP to check given no is Even (or) odd

#1

```
S = int(input('Enter a number'))
if s%2==0:
    print('given number Even')
Else:
    print('given number is odd') //
```

#2

```
S = int(input('Enter a number:'))
if s%2==1:
    print('given number is odd')
Else:
    print('given number is Even') //
```

WAP to find maximum number among 2 numbers :

```
a = int (input (" Enter first value"))
b = int (input (" Enter Second value"))
if a>b:
    print ('a is maximum value')
Else :
    print ('b is maximum value')
```

```
if (b>a):
```

WAP to given string palindrome (or) not :

```
S= input (" Enter a string")
If s == S[::-1]
    print ('given string is palindrome')
Else :
    print ('given string is not palindrome')
```

#

```
S1 = input(' Enter first String')
S2 = input (' Enter second String')
If sorted (S1) == sorted (S1):
    print (' given strings are Anagrams')
Else :
    print (' given strings are not Anagrams')
```

#2 Leap (or) Not

```
S = int (input (' Enter a year'))
if (S%4 == 0 and S%100 != 0) or S%400 == 0 :
    print ('S is a Leap Year')
Else :
    print (' S is not a Leap year')
```

If-elif Condition :

- When we have more than two Conditions
- to check then we will use if-elif

Syntax :

if condition :

Statements of if block

elif condition :

elif 1st block

elif Condition :

elif 2nd block

.....

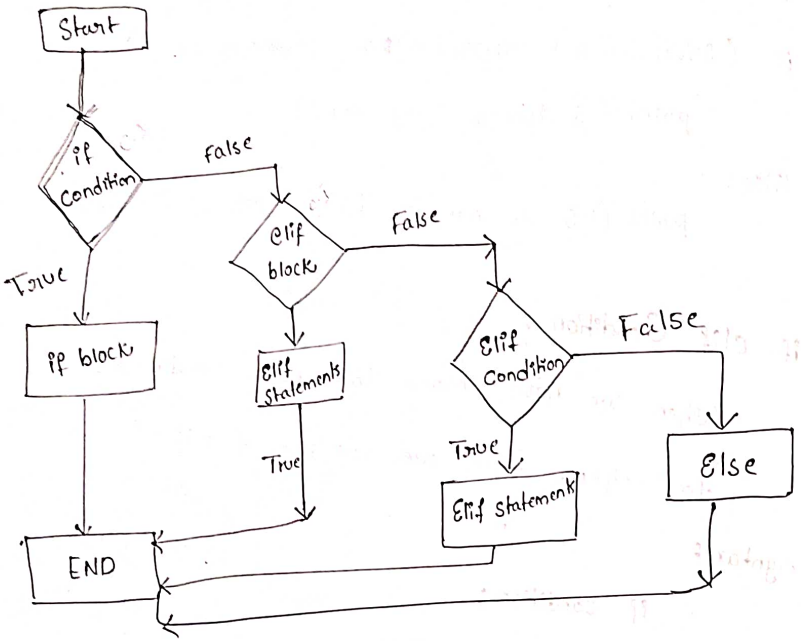
elif Condition :

elif nth block

else :

else blocks

flow diagram of if elif :




```
Print ("""Menu
        1. Indian food
        2. Russian food
        3. Italian food """)
Choice = int (input (' enter your choice Sir:'))
if  choice == 1 :
        print (' Serve Indian food')
        print (' Collect Bill and Tip')
elif  Choice == 2 :
        print (' Serve Russian')
        print (' collect Bill and pay')
elif  Choice == 3 :
        print (' Serve Italian')
        print (' Collect Bill and pay')

elif
Else :
        print (' please Leave Sir')
```

Maximum among 3 numbers

```
a = int(input (' Enter 1st number:'))
b = int(input (' Enter and number:'))
c = int (input (' Enter 3rd number:'))
if  (a>b and a>c):
        print (' a is maximum')
Elif  (b>c):
        print ('b is maximum')
Else :
        print (' c is maximum')
```

* Maximum of four Numbers :

```
a= int (input (' Enter 1st number'))
b= int (input (' Enter and number'))
c= int (input (' Enter 3rd number'))
d= int (input (' Enter 4th number'))
if  (a>b and a>c and a>d ):
        print (' a is maximum')
Elif  (b>c and b>d ):
        print (' b is maximum')
Elif  c>d :
```

```
s = int(input('Enter a leap year'))
if s%4==0 and s%100!=0:
    print('Enter leap year')
elif s%400==0:
    print('Enter leap year')
else:
    print('Enter not a leap year')
```

Nested Conditional Statements

It is the process of defining a conditional statement inside another conditional statement

```
if Condition:
    if condition:
        A
    else:
        B
else:
    if condition:
        C
    else:
        d
```

⇒
```
d = {'ashu': {'password': 'nikky'}, 'harshad': {'password': 'mahi'}}
username = input('enter username')
if username in d:
    pw = input('enter password')
    if pw==d[username]['password']:
        print('login is successful')
    else:
        print('password is not matching')
else:
    print('username is not available')
```

* Looping Statements *

1. When we have to Execute some set of instructions repeatedly then we will use looping statements

2. On python we have two types of Looping Statements.

   1. for loop (when we know how many times to iterate)
   2. while loop (when u r not sure how many times we will Execute)

For Loop : for loop will perform below mentioned Operations

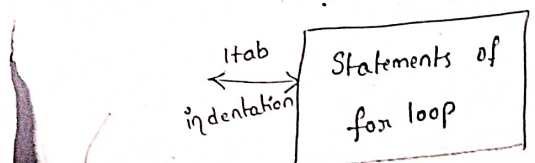
initialisation
Traversing

We can use for loop in two ways :

1. With Collection Data Type
2. With Range functions (integers) * * *

Syntax of for loop with COT :

```
for variable in COT :
    Statements of for loop
```



Extracting Each Element from string

```
S = 'ASHU'
for k in S:
    #print('i am in for')
    print(k)
```

output:
A
S
H
U

Extracting each element from empty string

```
Sl = ''
for k in Sl:
    print(k)
```

output:

#Note: Given COT is Empty then control will not enter into for block.

Extracting each element from List

```
L = ['ASHU','NIKKY','MAHI','ASIKA']
for i in L:
    print(i)
```

output:
ASHU
NIKKY
MAHI
ASIKA

Extracting Each Element from Tuple

```
T = ('ASHU','NIKKY',['MAHI','ASIKA'])
for i in T:
    print(i)
```

output:
ASHU
NIKKY
['MAHI','ASIKA']

Extracting Each Element from Set

```
S = {'ASHU','NIKKY','MAHI','ASIKA'}
for i in S:
    print(i)
```

output:
ASHU
NIKKY
MAHI
ASIKA

Extracting each element from dictionary

```
d = {'name':'ASHU','age':3}
```

Extracting keys

```
for k in d:
    print(k)
for k in d.keys():
    print(k)
```

o/p:
name
age

Extracting values

```
for k in d:
    print(d[k])
for v in d:
    print(v)
```

o/p:

Extracting both keys and values

```
for k in d:
    print(k,d[k])
for k,v in d:
    print(k,v)
```

WAP to print how many Elements are present in given CDT without using len function.

```
S = eval(input('enter the CDT'))
count = 0
for i in S:
    count+=1
print(count)
```

[when we give print outside the for loop then we get the count value].

WAP to print how many times 11 is repeated in the given string.

```
S = eval(input('enter a list: '))
L = [11]
count = 0
for i in S:
    if i in L:
        count+=1
        print(count)
```

10/Jan/2023

```
L = eval(input('Enter the List: '))
count = 0          (assuming)
for i in L:        Extract Each Element from given CDT
    if i == 11:
        count+=1
print(count)
```

* WAP to print how many vowels are present in given string ?

```
S = input('Enter the string:')
count = 0
Sl = 'aeiouAEIOU'
for i in S:
    ←----->if i in Sl:
        ←--->C+=1
print(c)
```

WAP to print how many ~~vowels~~ Consonants are present in given String ?

```
S = input('Enter a String:')
Sl = 'aeiouAEIOU'
count = 0
for i in S:
    if i.isalpha():        // if i.isalpha() and i not in Sl:
        if i not in Sl:
            count+=1
Print(count)
```

WAP to print all vowels

```
S = input ('Enter a string')
S1 = 'aeiouAEIOU'
for i in S:
    if i in S1:
        print (i)
```

WAP to print all consonants.

```
S = input ('Enter a string')
S1 = 'aeiouAEIOU'
for i in S:
    if i.isalpha() and i not in S1:
        print(i)
```

WAP to print how many digits are present in given string ?

```
S = input ('Enter a string:')
Count = 0
for i in S:
    if i.isdigit():
        Count+=1
print (count)
```

Print how many even digits are present in given string

```
S = input ('Enter a string:')
count = 0
for i in S:
    if int(i)%2==0:
        count+=1
print (count)
```

~~S = eval(input(~~

```
S = input ('Enter a string:')
count = 0
for i in S:
    if (int(i)%2==0:
        count+=1
print (count)
```

print how many odd digits are present in given string.

```
S = input ('Enter a string')
count = 0
for i in S:
    if i%2!=0:
        count+=1
print (count)
```

→

```
if i.isdigit():
    if int(i)%2!=0:
        count+=1
print (count)
```

WAP to print sum of digits present in given string.

```
S = input ('Enter a string')
sum = 0
for i in S:
    if i.isdigit():
        k = int(i)
        sum+=k
print(sum)
```

(variable (k) in which we are storing type casting value of i)

WAP to print sum of Even digits

```
S= input('Enter a string:')
sum= 0
for i in S:
    if i.isdigit():
        K= int(i)   (type conversion)
        if K%2== 0:
        ←→sum+= K
print(sum)
```

Sum of odd digits

```
S= input('Enter a string:')
sum= 0
for i in S:
    if i.isdigit():
        K= int(i)
        if K%2== 1:
        ←→ sum+=K
print(sum)
```

wap to print absolute difference of Even and odd in a given string.

```
S= input('Enter a string:')
Sumeven= 0
Sumodd= 0
for i in S:
    if i.isdigit():
        K= int(i)
        if K%2== 0:
            sumeven+= 1
        Else:
            Sumodd+= 1
→ (outside the for loop)
if (sumeven>sumodd):
    print(sumeven-sumodd)
Else:
    print(sumodd - sumeven)
```

Absolute difference of Even and odd in a given List.

```
S= Eval(input('Enter a string'))
Sumeven= 0
sumodd= 0
for i in S:
    if type(i)==int
    if i%2== 0:
        sumeven+= i
    Else
        sumodd+= i
if (sumeven > sumodd):
    print(sumeven- sumodd)
Else:
    print(sumodd - sumeven)
```

when?
['hai',11, 22, 33]

sum of digits in a given List

```
S= eval(input('Enter a list'))
sum= 0
for i in s:
    sum+= i
print(sum)
```

Sum of Even digits of a given List

```
S= Eval(input('Enter a list:'))
Sum= 0
for i in S:
    if i%2== 0:
        sum+= i
print(sum)
```

Sum of odd digits in a given List:

```
S= Eval(input('Enter a List:'))
Sum= 0
for i in S:
    if i%2!= 0:
        sum+= i
print(sum)
```

WAP to find factorial of a number.

```
n = int(input('Enter a number:'))
fact = 1 [ 0!=1, so we have taken 1]
for i in range(1,n+1):
    fact = fact * i
print(fact)
```

WAP to find Sum of given first n numbers

```
n = int(input('Enter a number:'))
summ=0 [ sum of '0' is 0]
for i in range(1, n+1): [range never takes UL in result so (n+1) (n) which includes 'n']
    summ+=i
print(summ)
```

WAP to reverse a string (without using slicing)

```
S = input('Enter a string:')
empty='' [If the given string is empty the reverse str is also'']
n = len(s) [when we deal with → [ range( ) with COT ] use len( )]
for i in range(-1,-(n+1),-1):
    empty+=S[i]
print(empty)
```

i → we get index positions

S[i] → we get elements

negative index positions to get reverse

2nd Jan

Range function : Range function is used for providing Limits inorder to Execute a for loop

range (Lowerlimit , upperlimit ± 1, updation)
start , stop , step

Examples :

list(range(1,10,1)) [Range never prints upperlimit value]

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

list(range(1,10,2))

[1, 3, 5, 7, 9]

list(range(1,10,3))

[1, 4, 7]

list(range(1,10)) [Default value of updation is 1]

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

list(range(10)) → [Default value of Lowerlimit will be 0, updation is 1]

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

list(range(0,10,2))

[0, 2, 4, 6, 8]

list(range(10,2)) [updation value is +1 ~~and~~ by default.

[ ]

list(range(10,2,-1))

[10, 9, 8, 7, 6, 5, 4, 3]

→ list(range(-1,-11,-1))

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

Syntax for loop with range :

```
for variable in range(LL, UL, UP):
    Statements of for loop with range
```

1tab

Example :

```
for i in range(1,10):
    print(i)
```

o/p:
1
2
3
4
5
6
7
8
9

WAP to find sum of given first n numbers

```
n = int(input('Enter a number'))
summ = 0
for i in range(1, n+1):
    summ = i
print(summ)
```

For loop with range and CDT :

Syntax :

```
for variable in range(len(CDT)):
    Statements of for loop with range and CDT
```

1tab indentation

Note : for → CDT → only Elements

for → range and CDT → Index position → (i)
perform Indexing(s[i]) to get Elements

Examples :

```
s = 'Python'
for i in range(len(s)):
    print(i, s[i])
```

WAP to print Even index position Elements.

```
s = input('Enter a string:') // s = input('Enter a String:')
n = len(s)                   for i in range(len(s)):
for i in range(n, 2)             if i%2 == 0:
    print(s[i])                      print(s[i])
```

LL UL

WAP to find sum of index positions of vowels

```
S = input('Enter a String:')
Summ = 0
v = 'aeiouAEIOU'
for i in range(len(S)):
    if S[i] in v:
        Summ += i
print(Summ)
```

→ Extracted index positions

[while we compare, Both sides should be a string]

→ string

S[i] / v

sum of indexpositions of consonants:

```
S = input('Enter a string:')
v = 'aeiouAEIOU'
Summ = 0
for i in range(len(S)):
    if S[i].isalpha():
        if S[i] not in v:
            Summ += i
print(Summ)
```

Sum of even index positions of vowels

```
S = input('Enter a String:')
Summ = 0
v = 'aeiouAEIOU'
for i in range(len(S)):
    if S[i] in v:
        if i%2 == 0:
            Summ += i
print(Summ)
```

WAP to find how many times given substring is present in given string (without using count)

```
S = input('Enter a String:')
S1 = input('Enter substring:')
C = 0
for i in range(len(S)):
    if S[i] == S1:
        C += 1
print(C)
```

↑ optimised solution

```
if S[i:i+len(sub)] == sub:
    C += 1
print(C)
```

Ex: S = 'PYTHON'

S[0:0+1:] => P

↑ gp to len(sub)

from gp

S[0:1] = from index position to

Extract one element

WAP to replace vowels in a given string to their index positions

```
S = input('Enter a string:')
dummy = ''
vowels = 'aeiouAEIOU'
for i in range(len(S)):
    if S[i] in vowels:
```

```
        dummy+ = str(i)
    Else:
        dummy+= s[i]
print(dummy)
```

WAP Remove all vowels and print the string.

```
S= input('Enter a string:')
dummy=''
vowels = 'aeiouAEIOU'
for i in range(len(s)):
    if s[i] not in vowels:
        dummy+ = s[i]
print(dummy)
```

// → for i in s:
    if i not in vowels:
        dummy+=i
print(dummy)

Print how many words are present in given string without using split methods.

```
S= input('Enter a String:')
if len(s) == 0:
    print('No words are present in S')
Else:
    count = 1          / count = 0
    for i in s:   [for with CDT only Elements]
        if i==' ':
            count+=1
```

print(c, 'words are present in S') / [print(c+1,'words are present in S')
If there are 2 spaces there will be 3 words so c+1]

WAP to print all the words without using split methods.

```
S= input('Enter a String:')
L=[]          II   L=[]      [string].
new=''              new=''
for i in range(len(s)):
    if s[i]==' ':
        L.append(new)  / L+=[new]  [list is one DT and RHS is str DT so we should not compare]
        new=''         new=''
    Else:
        new+=s[i]
```

[By using '+' operator we can add Elements
Ex: L=[11,22,33]+[66]
-> L=[]
-> L+=[new]]

#this Condition is used for adding only last word into the List

```
    if i==len(S)-1:
        L.append(new) / L+=new
print(L)
```

Without using append method.

```
S = input('Enter string:')
L = [ ]
new = ''
for i in range(len(s)):
    if s[i] == ' ':
        L+ = [new]
        new = ' '
    Else:
        new+ = s[i]
    if i == len(s) - 1
        L+ = [new]
print(L)
```

WAP to check given number is perfect number or not ?

for Example (1, n)

10 → 1, 2, 5 (half of it and below)

```
n = int(input('Enter a number:'))
summ = 0
for i in range(1, n//2+1):        (divisors)
    if n%i == 0:
        sum+ = i
if n == summ:
    print('n is perfect')
else:
    print('n is not perfect')
```

n//2 -> Any number u take it will divisible by half and of it only it will not consider above numbers so for that we have taken n//2

To avoid unnecessary iteration

Special Statements of python

1. break
2. Continue
3. Pass

break : break is used for terminating the execution of loops based on given condition

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

Continue : Continue is used for skipping the single iteration of loop based on given condition

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

Break and Continue we can use only in loops and along loops with Conditional statements

Pass : • Pass is used for creating Empty blocks

• pass acts as the place holder for statements which are to be written in future.

3. pass can be utilized in loops, conditions, in functions and as well as in classes

Examples :

```
def function ():
    pass
if 10 > 1:
    pass
for i in range (1, 10):
    pass
Class A :
    pass
```

25/Jan/2023:

for Else block :

```
for variable in range ():
    If condition :
        break
Else :
    block
```

Note : in case of for-Else block

1. If for loop is terminated then Else block will not be Executed
2. if for loop is not terminated then only Else block will get Executed

Examples of for-Else with break :

```
for i in range (1, 10):
    print (i)
    if i == 5:
        break
Else :
    print ('i am in Else block')
```

O/P :
1
2
3
4
5

Examples of for-Else without break :

```
for i in range (1, 10):
    print (i)
Else :
    print ('i am in Else block')
```

O/P :
1
2
3
4
5
6
7
8
9
i am in Else block.

wap to check prime number (or) not

→ Take a number (user input)

→ Set up a for loop range starts from 2, to half of its value i.e., (2, n//2+1)

→ checking the divisors

→ If any number is dividing given number then number is not prime break → terminate Execution

→ If loop is not terminated then that number is prime.

```
n = int(input("Enter a number:"))
if n > 1:
    for i in range(2, n//2+1):
        if n%i == 0:
            print("n is not a prime")
            break
    else:
        print("n is prime")
else:
    print("n is not prime")
```

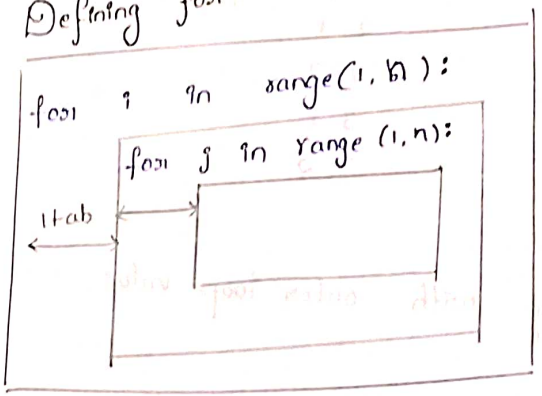
Nested Looping Statements

It is the process of defining a loop inside another loop

We can define nested loops in 4 ways:

- for loop inside another for loop
- for loop inside another while loop
- while loop inside another while loop
- while loop inside another for loop

Defining for inside another for loop:



Ex:

```
for i in range(1,6):
    for j in range(1,6):
        print(i, j)
```

o/p:

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

Breaking inner loop with inner loop value

Example 1 :

```
for i in range(1, 6):
    for j in range(1,6):
        print(i,j)
        if j==3:
            break
```

o/p :
```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
4 1
4 2
4 3
5 1
5 2
5 3
```

Example 2 :

```
for i in range(1,6):
    for j in range(1,6):
        if j==3:
            break
        print(i,j)
```

o/p :
```
1 1
1 2
2 1
2 2
3 1
3 2
4 1
4 2
```

Breaking inner loop with outer loop value

Example 1

```
for i in range(1,6):
    for j in range(1,6):
        print(i,j)
        if i==3:
            break
```

②

```
for i in range(1,6):
    for j in range(1,6):
        if i==3:
            break
        print(i,j)
```

Breaking outer loop with outer loop value

Example 1 :

```
for i in range(1,6):
    if i==3:
        break
    for j in range(1,6):
        print(i,j)
```

Example 2 :

```
for i in range(1,6):
    for j in range(1,6):
        print(i,j)
    if i==3:
        break.
```

Breaking outer Loop with inner loop value.

```
for i in range(1, 6):
    if j==3:
        break
    for j in range(1, 6):
        print(i, j)
```

o/p: j is not defined i.e., we are comparing before initialization //

Example(2):

```
for i in range(1, 6):
    for j in range(1, 6):
        print(i, j)
    if j==5:
        break.
```

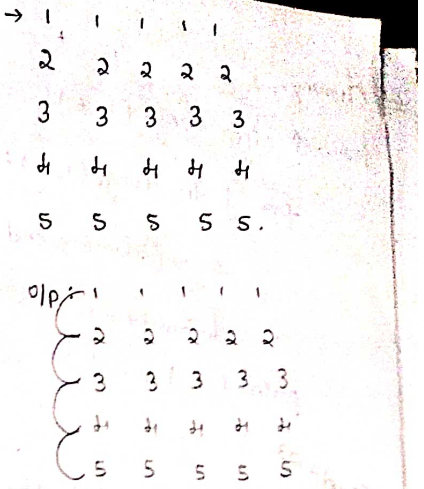
[if j==3: the control is coming out with j value as 5 so it will not satisfy the condition]

WAP to print the pattern

```
for i in range(1, 6):
    for j in range(1, 6):
        print('*', end=' ')
    print()
```

→ to make the control to move to next line after completion of the 1 iteration)

WAP to print the pattern →



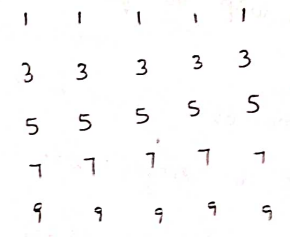
```
dummy = 1
for i in range(1, 6):
    for j in range(1, 6):
        print(dummy, end=' ')
    print()
    dummy+=1
```

O.Loop

increamented in O.Loop

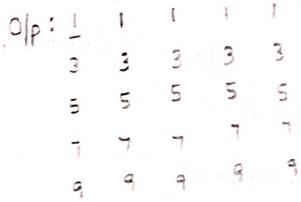
o/p:

WAP to print the pattern

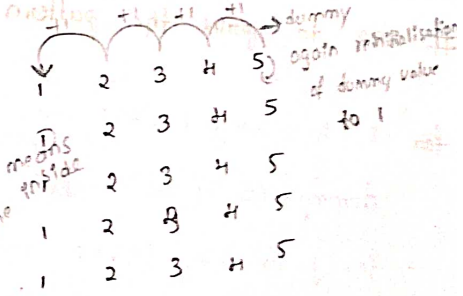


```
dummy = 1
for i in range(1, 6):
    for j in range(1, 6):
        print(dummy, end=' ')
    print()
    dummy+=2
```

o/p:



WAP to print the pattern




```
for i in range(1,6):
    dummy=1
    for j in range(1, 6):
        print(dummy, end=' ')
        dummy+=1
    print()
```

```
dummy = 1
for i in range(1, 6):
    for j in range(1, 6):
        print(dummy, end=' ')
        dummy += 1   [Increeemented inside]
    print( )
```

1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

After printing

28/Jan/2023

WAP to print the pattern

A A A A A
B B B B B
C C C C C
D D D D E
E E E E E

[Loop d value remains same for complete iteration]

```
dummy = 65   (based on ASCII value)
for i in range(1, 6):
    for j in range(1, 6):
        print(ch(dummy), end=' ')
    print( )
    dummy += 1
```

WAP to print the pattern A B C D E

```
for i in range(1, 6):
    dummy = 65   [ord('A')]   dummy value reset Holding
    for j in range(1, 6):
        print(chr(dummy), end=' ')
        dummy += 1
    print( )
```

A B C D E
A B C D E
A B C D E
A B C D E

WAP to print the pattern

A B C D E
F G H I J
K L M N O
P Q R S T
U V W X Y

```
dummy = 65
for i in range(1, 6):
    for j in range(1, 6):
        print(chr(dummy), end=' ')
        dummy += 1
    print( )   [New Statement]
```

① # WAP to print

[i and j are Equal print '*']

```
for i in range(1, 6):
    for j in range(1, 6):
        if i == j:
            print('*', end=' ')
        Else:
            print('_', end=' ')
    print( )
```

WAP to print

*
*
*
*

① Outer Loop :
Iterating on Row wise.

② Two inner Loops :
1st inner Loop printing spaces
and inner Loop printing stars

```
n = 5
sp = 0 [
stars = 1 [ Everytime we have to print one star ]
for i in range (n):
    for s in range (sp):
        print (' ', end=' ')
    for m in range (stars):
        print ('*', end=' ')
    sp+=1
    print( )
```

① (0,0) → UI = LW [Not satisfying].

So next Loop (i.e., stars Loop)

(0,1) → Loops runs for 1 time so print '*'

* After completion of 1st iteration entire spaces are incrementated by 1 i.e. sp = 0+1 = 1.

② (0,1) runs for 1 time so print - Space

(2)

WAP to print pattern :

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

```
n = int (input('Enter a range:'))
sp = n-1
stars = 1
for i in range (n):
    for s in range (sp):
        print(' ', end=' ')
    for m in range (stars):
        print('*', end=' ')
    sp-=1
    stars+=1
    print( )
```

(0, 6)
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5

(3)

WAP to print the pattern

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

```
n = int (input('Enter a range:'))
sp = 0
stars = 1
for i in range(n):
    (for s in range(sp):
        print(' ', end=' ')) we can remove this loop as spaces are '0'
    for m in range (stars):
        print('*', end=' ')
    sp-=1 [Not necessary Space value is zero always].
    stars+=1
    print( )
```

(4)

WAP to print the pattern

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

```
n = int (input ('Enter a range:'))
sp = 0 (some).
stars = n
for i in range(n):
    for s in range (sp):
        print (' ', end=' ')
    for m in range (stars):
        print ('*', end=' ')
    sp+=1
    stars-=1
    print( )
```

(5)

= WAP to print pattern =>

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

```
n = 5
sp = 0
stars = 1
dummy = 1
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ')
    print()
    stars+=1
    dummy+=1
```

(6)

WAP to print the pattern =>

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

```
n = 5
# sp = 0
stars = n
dummy = 1
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ')
    print()
    sp+=1
    stars-=1
    dummy+=1
```

WAP to print the pattern

(7)

```
1  -  -  -  -
2  3  -  -  -
4  5  6  -  -
7  8  9  10 -
11 12 13 14 15
```

```
n = int(input('Enter a range:'))
stars = 1
dummy = 1
sp = 0
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print('dummy', end=' ')
        dummy+=1
    stars+=1
    print()
```

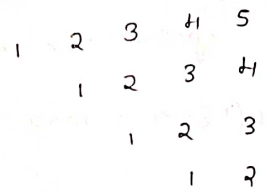
WAP to print the pattern →

(8)

```
-  -  -  -  1
-  -  -  2  3
-  -  4  5  6
-  7  8  9  10
11 12 13 14 15
```

```
n = 5
sp = n-1
stars = 1
dummy = 1
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ')
        dummy+=1
    sp-=1
    stars+=1
    print()
```

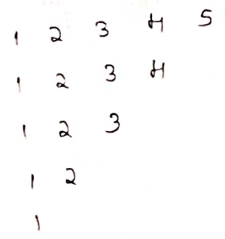
WAP to print pattern



(9)

```
n = 5
sp = 0
stars = n
for i in range(n):        # outer loop for running how many times [Don't manipulate range]
    dummy = 1
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ')
        dummy += 1
    sp += 1
    stars -= 1
    print()
```

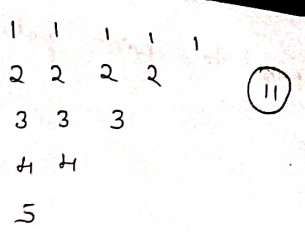
WAP to print pattern



(10)

```
n = 5
sp = 0
stars = n
for i in range(n):
    dummy = 1
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ')
        dummy += 1
    stars -= 1
    print()
```

WAP to print pattern



(11)

```
n = 5
stars = 5
sp = 0
dummy = 1
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ')
    print()
    stars -= 1
    dummy += 1
```

WAP to print the pattern

(12)

```
n = 5
stars = 5
sp = 0
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print('*', end=' ')
    sp += 1
    stars -= 1
    print()
```

WAP to print pattern (13)

```
1  2  3  4  5
6  7  8  9
10 11 12
13 14
15
```

```
n = 5
sp = 0
stars = n
dummy = 1
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ')
        dummy += 1
    stars -= 1
    sp += 1
    print()
```

OIP :

```
1  2  3  4  5
6  7  8  9
10 11 12
13 14 15
```

WAP to print (14)

```
1  2  3  4  5
6  7  8  9
10 11 12
13 14
15
```

```
n = 5
sp = 0
stars = n
dummy = 1
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ')
        dummy += 1
    stars -= 1
    print()
```

WAP to print (15)

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

```
n = 5
sp = n - 1
stars = 1
dummy = 1
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ')
    stars += 1
    sp -= 1
    print()
    dummy += 1
```

WAP to print pattern (16)

```
A B C D E
A B C D
A B C
A B
A
```

```
n = 5
stars = n
for i in range(n):
    dummy = 65
    for d in range(stars):
        print(chr(dummy), end=' ')
        dummy += 1
    stars -= 1
    print()
```

WAP to print pattern (17)

```
A A A A A
B B B B
C C C
D D
E
```

```
n = 5
Stars = n
dummy = 65
sp = 0
for i in range(n):
    print
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(chr(dummy), end=' ')
    stars -= 1
    print()
    dummy += 1
```

WAP to print pattern (18)

```
A A A A A
  B B B B
    C C C
      D D
        E
```

```
n = 5
stars = n
dummy = 65
sp = 0
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(chr(dummy), end=' ')
    stars -= 1
    sp += 1
    print()
    dummy += 1
```

WAP to print pattern (19)

```
A B C D E
A B C D
  A B C
    A B
      A
```

```
n = 5
stars = n
sp = 0
for i in range(n):
    dummy = 65
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(chr(dummy), end=' ')
        dummy += 1
    stars -= 1
    sp += 1
    print()
```

WAP to print pattern (20)

```
A
A B
A B C
A B C D
A B C D E
```

```
n = 5
stars = 1
for i in range(n):
    dummy = 65
    for d in range(stars):
        print(chr(dummy), end=' ')
        dummy += 1
    stars += 1
    print()
```

WAP to print pattern (21)

```
A
B B
C C C
D D D D
E E E E E
```

```
n = 5
stars = 1
dummy = 65
for i in range (n):
    for d in range (stars):
        print (chr(dummy), end=' ')
    stars += 1
    print( )
    dummy += 1
```

WAP to print pattern (22)

```
        A
      A B
    A B C
  A B C D
A B C D E
```

```
n = 5
sp = n-1
stars = 1
for i in range(n):
    dummy = 65
    for s in range (sp):
        print(' ', end=' ')
    for d in range (stars):
        print(chr(dummy), end=' ')
        dummy += 1
    sp -= 1
    stars += 1
    print( )
```

WAP to print (23)

```
        A
      B B
    C C C
  D D D D
E E E E E
```

```
n = 5
stars = 1
sp = n-1
dummy = 65
for i in range (n):
    #print(' ', end=' ')
    for s in range(sp):
        print(' ', end=' ')
    for d in range (stars):
        print ( chr(dummy), end=' ')
    sp -= 1
    stars += 1
    print( )
    dummy += 1
```

WAP to print pattern (24)

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

```
n = 5
stars = 1
sp = n-1
for i in range (n):
    for s in range (sp):
        print(' ',end=' ')
    for d in range (stars):
        print ('*', end=' ')
    sp -= 1
    print( )
```

WAP to print pattern (25)

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

```
n = 5
stars = 5
sp = 0
dummy = 1
for i in range(n):
    for s in range(sp):
        print(' ', end=' ')
    for d in range(stars):
        print(dummy, end=' ') | print('*', end=' ')
        dummy += 1
    sp += 1
    stars -= 2
    print()
```

o/p:

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

WAP to print

```
0 1 2 3 4 5 6 7 8
i=0 - - - - 1
i=1 - - - 2 1 2
i=2 - - 3 2 1 2 3
i=3 - 4 3 2 1 2 3 4
i=4 5 4 3 2 1 2 3 4 5
```

n = 5
sp = n-1
stars = 1

1st half decrementing

1st half incrementing.

```
for i in range(n): (0,5)
    dummy = i+1 or i  (i -> 0 -> 4)
    for s in range(sp):
        print(' ', end=' ')
    for c in range(stars): (0,1) columns range.
        print(dummy, end=' ')
        if c < stars//2:   (0 < 0 ; 1 < 1)
            dummy -= 1
        Else:
            dummy += 1
    sp -= 1
    stars += 2
    print()
```

WAP to pattern

```
        A
    B   A   B
C   B   A   B   C
D   C   B   A   B   C   D
E   D   C   B   A   B   C   D   E
```

i = 0, 1, 2, 3, 4

n = 5
sp = n-1
stars = 1
dummy = 65

```
for i in range(n): (outer loop for number of lines)
    dummy = i + 65
    for s in range(sp): (inner loop for spaces)
        print(' ', end=' ')
    for c in range(stars): (Nested loops for columns)
        print(chr(dummy), end=' ')
        if c < stars//2:
            dummy -= 1
        Else:
            dummy += 1
    sp -= 1
    stars += 2
    print()
```

WAP to print pattern

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

dummy = stars//2 + 1
i.e., 1//2+1 => 1

initially
Stars → 1 → 1 (dummy value)

Stars → 3 → 2 (starts with - 3 2 1 2 3)

Stars → 5 → 3 (" - - - 2 1 2)

Stars → 7 → 4 (" - - - - 1.)

Stars → 9 → 5

Stars → 7 → 4

Stars → 5 → 3

Stars → 3 → 2

Stars → 1 → 1

```
n = 9
sp = n - 1
stars = 1
for q in range(n):
    dummy = stars//2 + 1
    for s in range(sp):
        print(' ', end=' ')
    for c in range(stars):
        print(dummy, end=' ')
        if c < stars//2:
            dummy -= 1
        Else:
            dummy += 1
    if i < n//2:
        sp -= 1
        stars += 2
    Else:
        sp += 1
        stars -= 2
    print( )
```

WAP to print pattern

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

```
n = 9
o, e = [ ], [ ]
for i in range (1, n+1):
    if i%2 == 1:
        o.append (i)
    else:
        e.append (i)
L = o + e[::-1]
Sp = 0
Stars = 1
for i in range (n):
    for s in range (Sp):
        print (' ', end=' ')
    for c in range (Stars):
        print (L(r), end=' ')
    if r < n//2:
        Sp+=1
    else:
        Sp-=1
    print ( )
```

WAP to print pattern

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

```
n = 5
Sp = n-1
Stars = 1
for i in range (n):
    for j in range (Sp):
        print (' ', end=' ')
    for k in range (Stars):
        print ('*', end=' ')
    if k < Stars//2:
        Stars += 2
        Sp-=1
    Stars += 2
    Sp-=1
```

WAP to print pattern

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

```
n = 5
Sp = 0
Stars = 2 * n - 1   [2 * 5-1] → 10-1 = 9
for i in range (Sp):
    for j in range (Sp):
        print (' ', end=' ')
    for s in range (St):
        print ('*', end=' ')
    print ( )
    St-=2
    Sp+=1
```

WAP to print

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

```
n=5
for i in range(n):
    for j in range(n):
        if (i==0 or j==0):
            print('*', end=' ')
        elif(i==2):
            print('*', end=' ')
        elif(i==4):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print()
```

WAP to print pattern

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

```
n = 5
for i in range(n):
    for j in range(n):
        if (i==0 or j==0):
            print('*',end=' ')
        elif(i==2):
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
```

WAP to print pattern

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

```
n= 5
for i in range(n):
    for j in range(n):
        if (i==0 or j==0):
            print('*', end=' ')
        elif (i==2 and j==2) or (i==2 and j==3):
            print('*', end=' ')
        elif (i==2 and j==4):
            print('*', end=' ')
        elif (i==3 and j==2) or (i==3 and j==4):
            print('*', end=' ')
        elif (i==4 and j==1) or (i==4 and j==4) or (i==4 and j==2):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print()
```

WAP to print pattern

```
  j=0 1 2 3 4
i=0 * * * * *
1     *
2     *
3 *   *
4 * * *
```

```
n= 5
for i in range(n):
    for j in range(n):
        if (i==0) or (j==2):
            print('*', end=' ')
        elif (i==4 and j==0) or (i==4 and j==1):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print()
```

```
        elif (i==4 and j==2) or (i==3 and j==0):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print()
```

WAP to print pattern * * * * *

```
n=5
for i in range(n):
    for j in range(n):
        if (i==0 or i==n-1):
            print('*', end=' ')
        elif (j==2):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print()
```

WAP to print pattern

```
n=5
for i in range(n):
    for j in range(n):
        if (j==0 or j==n-1):
            print('*', end=' ')
        elif (i==2):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print()
```

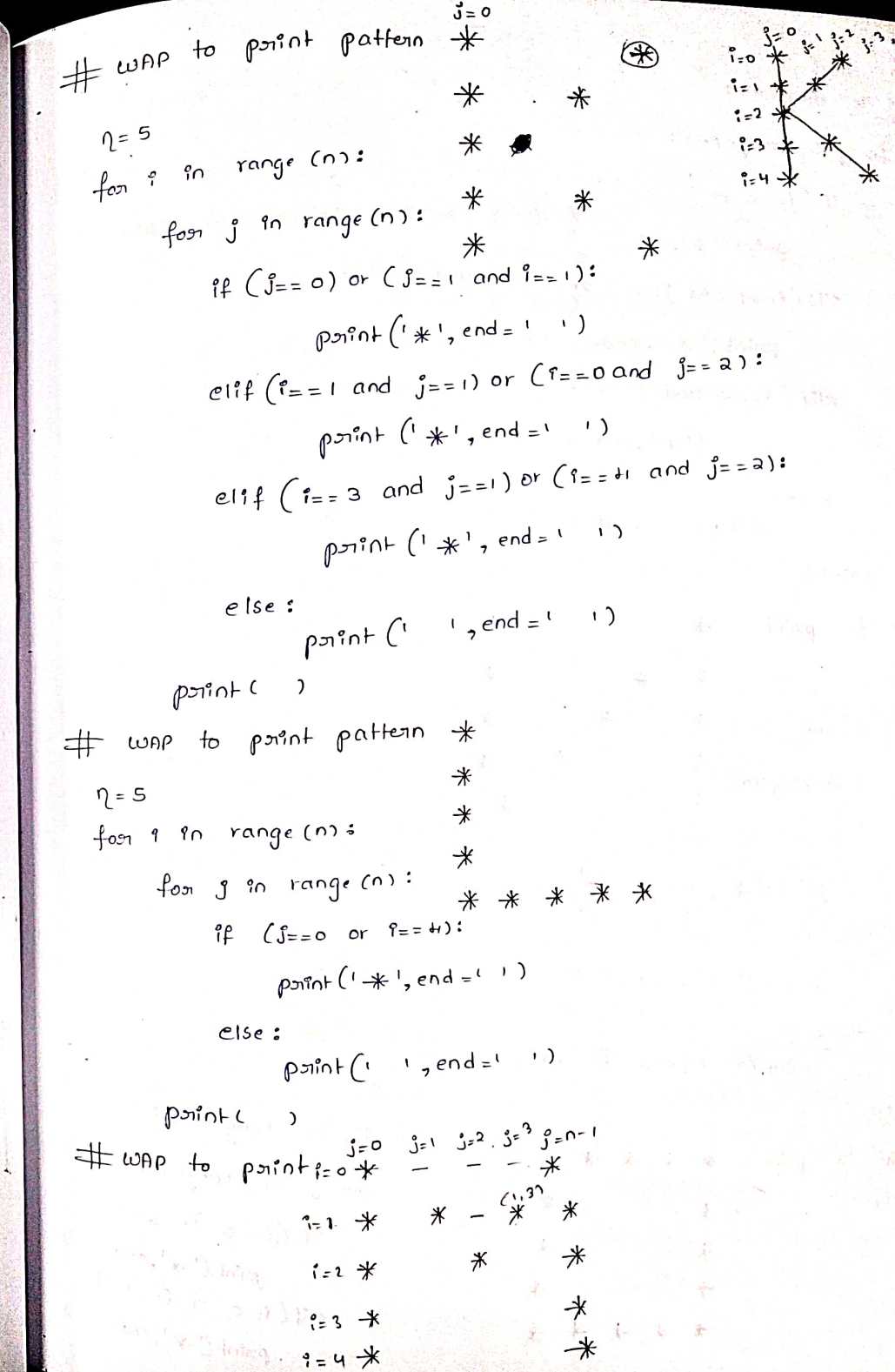
WAP to print pattern

```
n=5
for i in range(n):
    for j in range(n):
        if (j==0) or (j==1 and i==1):
            print('*', end=' ')
        elif (i==1 and j==1) or (i==0 and j==2):
            print('*', end=' ')
        elif (i==3 and j==1) or (i==4 and j==2):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print()
```

WAP to print pattern

```
n=5
for i in range(n):
    for j in range(n):
        if (j==0 or i==4):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print()
```

WAP to print



```
n=5
for i in range(n):
    for j in range(n):
        if (j==0 or j==n-1):
            print('*',end=' ')
        elif(i==1 and j==1) or (i==1 and j==3):
            print('*',end=' ')
        elif (i==2 and j==2):
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
```

WAP to print

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

```
n=5
for i in range(n):
    for j in range(n):
        if (j==0 or j==n-1):
            print('*',end=' ')
        elif (i==j):
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
```

WAP to print

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

```
n=5
for i in range(n):
    for j in range(n):
        if (j==0 or j==n-1):
            print('*',end=' ')
        elif (i==0 or i==n-1):
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
```

WAP to print

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

```
n=5
for i in range(n):
    for j in range(n):
        if (j==0 or i==n-1):
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
```

WAP to print

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

1st dummy = 0 [dummy variable]
for n in range (1, 100): (getting numbers)
    if n > 1:
        for i in range (2, n//2 + 1): (get p n)
            if n%i == 0
                break
    else:
        print(n)
        dummy += 1
        if dummy%2 == 0:
            print(n)

[Number Entered Else which means if is a prime]

dummy value
1 → 2
(2) → 3
3 → 5
(4) → 7
5 → 11
(6) → 13

Even positions

2nd approach:

```
L = [ ]
for n in range (1, 100):
    if n > 1:
        for i in range (2, n//2 + 1):
            if n%i == 0:
                break
        else:
            L.append(n)
print(L[1::2])
```

```
dummy = 0
for n in range (1, 100):
    if n > 1:
        for i in range (2, n//2 + 1):
            if n%i == 0:
                break.
        Else:
            dummy += 1
            if dummy%3 == 0:
                print(n)
```

o/p:
5 17 [3rd position]
13 61
23 73
37 89.

WAP to print 1st 10 prime numbers in a range.

```
dummy = 0
for n in range (1, 100):
    if n > 1:
        for i in range (2, n//2 + 1):
            if n%i == 0:
                break
        else:
            dummy += 1
            #if dummy <= 10: [Unnecessary Iterations]
            print (n)
    if dummy == 10:
        break.
```

o/p: 2 3 5 7 11 13 17 19 23 29

```
rows=7
dummy=0
e=[ ]
o=[ ]
for n in range (1,100):
    if n>1:
        for i in range (2, n//2+1):
            if n%i==0:
                break
        else: ~~else:~~
            dummy+=1
            if dummy%2==1
                o.append(n)
            else
                e.append(n)
    if dummy==rows
        break
L=o+e[::-1].
sp=0
st=1
for r in range (rows):
    for s in range (sp):
        print (' ', end=' ')
    for c in range (st):
        print (L(r), end=' ')
    if r<rows//2:
        sp+=1
    else
        sp-=1
    print ( )
```

print all perfect numbers in a given range

```
for n in range (1, 10000):
    summ=0                                  o/p:  6
    for i in range (1, n//2+1):                   28
        if n%i==0:                                496
            summ+=i                               8128
    if n==summ:
        print (n)
```

print every second perfect numbers in given range.

```
dummy=0                                     o/p: 28
for n in range (1, 500):
    summ=0
    for i in range (1, n//2+1):
        if n%i==0
            summ+=i
            dummy+=1
    If (n==summ) and (dummy%2==1):
        print (n)
```

print Every third Even number in a given range:

```
dummy = 0
for i in range (1,50):
    if i%2 == 0:
        dt = 1
        if dummy%3 == 0:
            print(i)
```

*While Loop : We use while loop when we are not sure how many times we have to iterate.

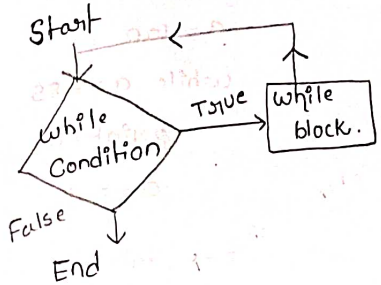
2. While Loop will be Executed on one condition.

3. There is a chance to Execute ur while loop for Infinite times, inorder to avoid that make sure at one point of time.

The condition becomes false (i.e.,) we must provide an increement the value (or) decreement the value

Syntax :

```
while condition:
    Statements of
    while loop
```



Examples :

```
a = 1
while a<10:
    print(a)
    print('inside while')
    a+=1
```

O/p:
1
inside while
2 "
3 "
4 "
5 "
6 "
7 "
8 "
9

```
a=15
while a>11:
    print(a):
    print('inside while')
    a-=1
```

O/p:
15
inside while
14
inside while
13
inside while
12
inside while

→ Differences in for and while loop ←

For	while
It is used when we know the number of iterations to be done.	It is used when we doesn't know the iteration.
→ for loop will not use condition to iterate	Based on condition only while loop is iterated
→ Range function is used in for loop	we cannot use range function in while loop
→ No chances of Infinite loop Execution	→ Infinite loops might be possible in while loop

12 to 25

```
a=12
while a<=25:
    print(a)
    a+=1
```

-76 to -89

```
a=-76
while a>=-89
    print(a)
    a-=1
```

100 to 85

```
a=100
while a>=85
    print(a)
    a-=1
```

#WAP to print even numbers b/w 50 to 120

```
a=50
while a<=120:
    if n%2==0:
        print(a)
    a+=1
```

Even numbers from 89 to 29

```
a=89
while a>=29:
    if a%2==0:
        print(a)
    a-=1
```

Inner loop with inner loop value

```
i=1
while i<6:
    j=1
    while j<6:
        print(i,j)
        if j==3:
            break
        j+=1
    i+=1
    print()
```

O/p:

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

Inner loop with outer loop value.

```
i=1
while i<6:
    j=1
    while j<6:
        print(i,j)
        if i==3:
            break
        j+=1
    i+=1
    print()
```

O/p:

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

Outer loop with outer loop value:

```
i=1
while i<6:
    j=1
    while j<6:
        print(i,j)
```

```
    j+= 1
if  j== 3:
    break
i+= 1
print( )
```

o/p:

1	1	2	1	3	1
1	2	2	2	3	2
1	3	2	3	3	3
1	4	2	4	3	4
1	5	2	5	3	5.

Outer Loop with Inner Loop Value

```
i= 1
while i< 6:
    j= 1
    while j< 6:
        print (i, j)
        j+= 1
    if  j== 3:
        break
    i+= 1
    print( )
```

o/p:

1	1	3	1	5	1
1	2	3	2	5	2
1	3	3	3	5	3
1	4	3	4	5	4
1	5	3	5	5	5.
2	1	4	1		
2	2	4	2		
2	3	4	3		
2	4	4	4		
2	5	4	5		

While Loop inside for Loop:

```
i= 1
while i< 6:
    for j in range(1, 6):
        print(i, j)
        if j==3:
            break
    i+= 1
    print( )
```

Inner Loop with Inner loop value

o/p:

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

Terminate Inner Loop with o.l value.

```
i= 1
while i< 6:
    for j in range(1, 6):
        print(i, j)
        if i== 3:
            break
    i+= 1
    print( )
```

o/p:

Terminate outer loop with outer Loop value

```
i= 1
while i< 6:
    for j in range(1, 6):
        print(i, j)
    if i== 3:
        break
    i+=1
    print( )
```

o/p:

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

Terminate outerloop with Inner Loop value.

```
i= 1
while i< 6:
    for j in range(1, 6):
        print(i, j)
        if j==5:
            break
    i+= 1
    print( )
```

o/p:

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

for inside while loop [J.L with, J I value]

```
for i in range(1, 6):
    j = 1
    while j < 6:
        print(i, j)
        if j == 3:
            break
        j += 1
    print( )
```

o/p:

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

[Inner Loop with outer loop value]:

```
for i in range(1, 6):
    j = 1
    while j < 6:
        print(i, j)
        if i == 3:
            break
        j += 1
    print( )
```

we are terminating J. Loop it doesn't affect my outer loop

o/p:

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

Outer Loop with outer Loop value:

```
for i in range(1, 6):
    j = 1
    while j < 6:
        print(i, j)
        j += 1
    if i == 3:
        break
    print( )
```

o/p:

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

WAP to print 1 to 10 by using Infinite loop

```
n = 1
while True:
    print(n)
    if n == 10:
        break
    n += 1
```

WAP to ~~print~~ take integer input and add it into List untill the given number is negative.

```
L = [ ]
while True:
    n = int(input('enter n value:'))
    if n < 0:
        break
    #L.append(n)
    L += [n]
print(L)
```

WAP to print first 10^{th} prime numbers ?

```
n = 1    10
t = int(input('Enter how many prime you need'))
c = 0    [count variable]
while True:
    if n > 1:
        for i in range(2, n//2+1):
            if n%i == 0:
                break
        Else:
            print(n)
```

for inside while loop [G.L with . G Lvalue]

```
for i in range(1, 6):
    j = 1
    while j<6:
        print(i, j)
        if j==3:
            break
        j+=1
    print( )
```

o/p:

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

[Inner Loop with outer loop value]:

```
for i in range(1,6):
    j=1
    while j<6:
        print(i, j)
        if i==3:
            break
        j+=1
    print( )
```

We are terminating G. Loop it doesn't affect my outer loop

o/p:

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

outer Loop with outer Loop value:

```
for i in range(1, 6):
    j=1
    while j<6:
        print(i,j)
        j+=1
    if i==3:
        break
    print( )
```

o/p:

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

WAP to print 1 to 10 by using Infinite Loop

```
n=1
while True:
    print(n)
    if n==10:
        break
    n+=1.
```

WAP to ~~print~~ take integer input and add it into List unhill the given number is negative.

```
L=[ ]
while True:
    n=int(input('enter n value:'))
    if n<0:
        break
    #L.append(n)
    L+=[n]
print(L)
```

WAP to print first 10 prime numbers ?

```
n=1   10
t=int(input('Enter how many prime you need'))
c=0   [count variable]
while True:
    if n>1:
        for i in range(2, n//2+1):
            if n%i==0:        if
                break
        Else:
            print(n)
```

```
        c+=1

    if c==t:
        break

    n+=1
```

WAP to print nth prime number.

```
n=1
t= int(input('Enter the range:'))
c=0
while True:
    if n>1:
        for i in range(2,n//2+1):
            if n%i==0:
                break

        else:
            c+=1

    if c==t:
        print(n) [printing n
        break

    n+=1
```

#

10th prime number to 20th prime number

```
n=1
LL= int(input('Enter lower limit'))
UL= int(input('Enter upper limit'))
c=0
while True:
    if n>1:
        for i in range(2,n//2+1):
            if n%i==0:
                break

    else:
        c+=1
        if c>=LL
            print(n)

    if c==UL:
        break

    n+=1
```

WAP to print summ of individual digits of a Numbers.

```
n= int(input('Enter digits'))
summ=0
while n>0:
    reminder=n%10   (To Extract last digit)
    summ+=reminder  (To reduce the last digit)
    n=n//10
print(summ)
print(n)
```

remainder

1234%10 → 4

1234//10 → 123

1234%10 = 4.(reminder)

0+4 = 4

1234//10 → 123

123%10 → 3

4+3=7

123//10 → 12

12%10 → 2

7+2=9

1

WAP to print reverse of a given number ?

```
n = int(input('Enter digits'))
rev = 0
while n>0:
    reminder = n%10
    rev = rev * 10 + reminder
    n = n//10
print(rev)
print(n)
```

WAP to check given number is palindrome (or) not

```
n = int(input('Enter a digit'))
temp = n
rev = 0
while n>0:
    reminder = n%10
    rev = rev * 10 + reminder
    n = n//10
if rev == temp:
    print('Given number is palindrome')
Else:
    print('Not a palindrome')
```

(After Complete iteration only we will get rev so we have to write if cond outside the loop)

[∵ while Comparing we should not compare it with n because it is reduced to 0]

rev == n ✗

WAP to check given number is Armstrong (or) not

```
n = int(input('Enter a number'))
L = len(str(n))
temp = n
summ = 0
while n>0:
    reminder = n%10
    summ += reminder**L
    n = n//10
if summ == temp:
    print('n is Armstrong')
Else:
    print('n is not a Armstrong')
```

371 → 3**3 + 7**3 + 1**3

1634 → 1⁴ + 6⁴ + 3⁴ + 4⁴

Armstrong number of 3 digits is an integer such that sum of the cubes of its digits is Equal to number itself

WAP to check given number is Disarium number or not ?

Disarium number → when sum of its digits raised to the power of their respective positions is Equal to number itself

89 → 8¹ + 9² → 89

135 → 1¹ + 3² + 5³ → 135

```
n = int(input('Enter a value'))
num = n
summ = 0
L = len(str(n))
while n>0:
```

```
reminder = n%10
summ = summ + reminder ** L
n = n//10
L -= 1
if summ == num:
    print('n is disarium number')
Else:
    print('not disarium number')
```

* WAP to check given number is Harshad/ Niven number or not ?

81 → 8+1 = 9

$\frac{81}{9} = 3$

```
n = int(input('Enter a number'))
temp = n
Summ = 0
while n>0:
    reminder = n%10
    summ += reminder
    n = n//10
if summ%temp == 0:
    print('Harshad number')
Else:
    print('not harshad number')
```

WAP to print armstrong number b/w 1 and to 1000 ?

```
for n in range(1, 1001):
    num = n
    summ = 0
    L = len(str(n))
    while n>0:
        rem = n%10
        summ = summ + rem**L
        n = n//10
    if num == summ:
        print(num)
```

WAP to print first 10 armstrong numbers ?

```
n = 1
xt = int(input('Enter how many Armstrong number u need'))
c = 0
while True:
    num = n
    summ = 0
    L = len(str(n))
    while num>0: [we shouldn't compare it with n as the n value is incremented]
        reminder = num%10
        summ += reminder ** L
        num = num//10
    if summ == n:
        print(n)
        c += 1
```

```
    reminder = n%10
    Summ = Summ + reminder ** L
    n = n//10
    L -= 1
if summ == num:
    print('n is disarium number')

Else:
    print('not disarium number')
```

* WAP to check given number is Harshad/ Niven number or not ?

```
n = int(input('Enter a number'))
temp = n
Summ = 0
while n>0:
    reminder = n%10
    Summ += reminder
    n = n//10
if Summ%temp == 0:
    print('Harshad number')
Else:
    print('not harshad number')
```

21 → 2+1 = 3

21/3 = 3

WAP to print Armstrong number b/w 1 and to 1000 ?

```
for n in range(1, 1001):
    num = n
    Summ = 0
    L = len(str(n))
    while n>0:
        rem = n%10
        Summ = Summ + rem**L
        n = n//10
    if num == Summ:
        print(num)
```

WAP to print first 10 Armstrong numbers ?

```
n = 1
Xt = int(input('Enter how many Armstrong number u need'))
C = 0
while True:
    num = n
    Summ = 0
    L = len(str(n))
    while num>0: [we shouldn't compare it with n as the n value is incremented]
        reminder = num%10
        Summ += reminder ** L
        num = num//10
    if Summ == n:
        print(n)
        C += 1
```

```
if c == 10:
    break

n+=1
```

WAP to print nth Armstrong number ?

```
t = int(input('Enter number'))
n = 1
c = 0
while True:
    num = n
    summ = 0
    L = len(str(n))
    while num > 0:
        rem = num%10
        summ += rem**L
        num = num//10
    if summ == n:
        c+=1
    if c==t:
        print(n)
        break
    n+=1
```

WAP to print 5th A.N to 10th.

```
n = 1
LL = int(input('Enter Lower limit'))
UL = int(input('Enter upper Limit'))
c = 0
while True:
    num = n
    summ = 0
    L = len(str(n))
    while num > 0:
        rem = num%10
        summ = summ + rem**L
        num = num//10
        L = 1
    if summ == num:
        print('nums) disarium number')
    else:
```

```
if c >= LL:
    print(n)
if c == UL:
    break
```

WAP Disarium numbers from 1 to 1000 ?

```
for n in range(1,1001):
    num = n
    summ = 0
    L = len(str(n))
    while n > 0:
        rem = n%10
        summ = summ + rem**L
        n = n//10
        L = 1
    if summ == num:
        print('nums) disarium number')
```

o/p: 5
10
5
6
7
8
9
153

WAP to print first 10 Disarium number

WAP to print 10th Disarium

WAP to print 10th to 13th Disarium number

WAP to harshad number b/w 1 to 1000

WAP first 10 harshad numbers

WAP 10th harshad number

WAP 10th to 13th harshad number

WAP to print a number is a Special number ? * [145 → 1! + 4! + 5!, 145 = 145]

```
n = int(input('Enter a number'))
num=n
Summ=0
while num>0:
    reminder=num%10
    fact=1
    for i in range(1,reminder+1):
        fact*=i
    summ+=fact
    num=num//10
if summ==n:
    print('n is Special')
Else:
    print('n is not Special')
```

```
if sum==n:
    print(n)
    c+=1
if c==10:
    break
n+=1
```

WAP to Sort the Elements in a given List ?

Bubble Sort

L = [100, 334, 124, 555, 245, 75, 25]
0 1 2 3 4 5 6

```
n= len(L)
for passes in range(n-1):   [for 7 iterations 6 Elements are Sorted so we have taken (n-1)]
    for i in range(n-1-passes):   [for Every 1st iteration one Element is Sorted → 2nd iteration two Elements]
        if L[i]>L[i+1]:
            L[i],L[i+1]=L[i+1],L[i]   (Swapping)
print(L)
```

WAP to sort the Elements in descending order in a given List.

```
L = int(input('Enter a List'))
n= len(L)
for passes in range(n-1):
    for i in range(n-1-passes):
        if L[i]<L[i+1]:
            L[i],L[i+1]=L[i+1],L[i]
print(L)
```

Print the maximum number in a List ?

```
L = int(input('Enter a List'))
n = len(L)
for passes in range(n-1):
    for i in range(n-1-passes):
        if L[i]>L[i+1]:
            L[i], L[i+1] = L[i+1], L[i]

print(L[-1])
```

i/p [11, 22, 33, HH, 55, 66]
o/p [22, 11, 44, 33, 66, 55]

L = [11, 22, 33, HH, 55, 66]

0 +1 → 1
2 +1 → 3
4 +1 → 5
6 +1 → 7

```
for i in range(0, len(l), 2):
    l[i], l[i+1] = l[i+1], l[i]

print(L)
```

first 10 Disarium numbers ?

```
n = 1
c = 0
while True:
    num = n
    summ = 0
    L = len(str(n))
    while num > 0:
        rem = num % 10
        summ += rem ** L
        num //= 10
        L -= 1
    if summ == n:
        print(n)
        c += 1
    if c == 10:
        break
    n += 1
```

o/p: 1 2 3 4 5 6 7 8 9 89

WAP to print 10th Disarium

```
t = int(input('Enter nth disarium number:'))
n = 1
c = 0
while True:
    num = n
    summ = 0
    L = len(str(n))
    while num > 0:
        rem = num % 10
        summ += rem ** L
```

```
    num//=10
    L-=1
if summ==n:
    print(n)
    ct=1
if c==10:
    break
nt=1
```

WAP to print 10th to 13th Disarium number ?

```
t=int(input('Enter first nth value:'))
t1=int(input('Enter last nth value:'))
n=1
c=0
while True:
    num=n
    Summ=0
    L=len(str(n))
    while num>0:
        rem=num%10
        Summ+=rem**L
        num//=10
        L-=1
    if summ==n:
        ct=1
        if c>=t:
            print(n)
    if c==t1
        break
    nt=1
```

o/p: Enter first nth value : 10th
Enter second nth value : 13

89
135
175
518

WAP to print given number is EMIRP or not ?

```
n=int(input('Enter number:'))
num=n
rev=0
while n>0:
    rem=n%10
    rev=rev*10+rem
    n=n//10
if num!=rev:
    if num>1:
        for i in range(2,n//2+1):
            if num%i==0:
                print('NOT EMIRP')
                break.
        Else:
            for j in range(2,rev//2+1):
                if rev%j==0:
                    print(NOT EMIRP)
                    break.
            Else:
                print('EMIRP')
    Else:
        print('NOT EMIRP')
Else:
    print('NOT EMIRP')
```

To Check prime number

EMIRP:

↳ Not palindrome
↳ Not a prime
↳ reverse number should also a prime

o/p: 13
→ EMIRP ←

wap to print given number is pallyprime (or) not ?

```
n= int(input('Enter a number:'))
num= n
rev = 0
count = 0
while n>0:
    rem = n%10
    rev = rev * 10 + rem
    n = n//10
if num == rev:
    if num>1:
        for i in range (2, n//2+1):
            if i%num==0:
                print('Not PallyPrime')
                break
        Else:
            print ('PallyPrime')
    Else:
        print ('not pally prime')
Else:
    print ('not pallyprime')
```

pallyprime :

→ It should be a palindrome

→ It should be a prime number.

Count += 1
# Count%2 == 0:
print(num)

wap to print highest value in a given list without using in-built methods ?

```
L = [5, 1, 4, 9, 6]
max = L[0]          Assuming my first value is highest
for i in range(1, len(L)):   [ Range starts from 1 because I need to compare 1st Element with Second Element ]
    if L[i] > max:
        max = L[i]
print(max)
```

wap to find minimum value inside the list ?

```
L = [5, 1, 4, 9, 6]
min = L[0]
for i in range(1, len(L)):
    if L[i] < min:
print(min)
```

Selection Sort : Based on index positions we will sort the least Elements in first index positions.

wap to print ~~the sorted~~ to sort the list using selection sort ?

```
l = [9, 8, 6, 4, 1]
for i in range(len(l)-1):
    min = i
    for j in range(i+1, len(l)):
```

```
        if l[j]< l[min]:
            min=j
        l[i],l[min]= l[min],l[i]
    print(l)
```

o/p:

```
1 8 6 4 9
1 4 6 8 9
1 4 6 8 9
1 4 6 8 9
```

WAP to print how many times each and every character is present in given string ?

```
s = 'hai harshad'
d = { }
for i in range (len(s)):
```

— x —

```
s = input ('Enter a string:')
d = { }
for i in s:
    d[i] = s.count(i)
print (d)
```

** # WAP to print Each and Every character is present in given string without using count

```
s = input ('Enter a String:')
d = { }
for i in s:
    if i not in d:
        d[i] = 1
    else:
        d[i] += 1
print(d)
```

o/p: 'hai madhu'

{'h':2,'a':2,'i':1,' ':1, 'm':1,'d':1,'u':1}

WAP to most repeated character ?

```
s = input ('Enter a String:')
d = { }
for i in s:
    if i not in d:
        d[i] = 1
    else:
        d[i] += 1
mx = max(d.values())
for k in d:
    if d[k]== mx:
        print (k)
```

o/p: 'hai ashu vali'

a

(k, d[k]) → o/p: a 3

wap to print most repeated word

```
s = input('Enter a string:')
d = { }
L = s.split( )
for i in range L:
    if i not in d:
        d[i] = 1
    Else:
        d[i] += 1
print(d)
mx = max(d.values( ))
for k in d:
    if d[k] == mx:
        print(k)
```

wap to print lengthest word in the given string ?

```
s = input('Enter a string:')
d = { }
L = s.Split( )
for i in L:
    d[i] = len(i)
print(d)
mx = max(d.values( ))
for k in d:
    if d[k] == mx:
        print(k)
```

El → k
L → v

wap to print unique characters in a given string ?

```
s = input('Enter a string:')
d = { }
for i in s:
    if i not in d:
        d[i] = 1
    Else:
        d[i] += 1
print(d)
for k in d:
    if d[k] == 1:
        print(k)
```

o/p: 'harshad'

h
a
r
s
d

wap to print unique words in a given string ?

```
s = input('Enter a string:')
d = { }
L = s.Split( )
for i in L:
    if i not in d:
        d[i] = 1
    Else:
        d[i] += 1
print(d)
for k in d:
    if d[k] == 1:
        print(k)
```

o/p:

i/p: Santhosh is proposing a girl Tomorrow.

o/p: GirlTomorrow girl a proposing is Santhosh.

```
L = input('Enter a Sentence:')
S = L.split()
R = L[::-1]
L = S.split()
R = L[::-1]
print(' '.join(R))
```

```
L = input('Enter a Sentence:')
L.split()
n = len(s)
for i in range(-1,
```

i/p: Santhosh is proposing a girl tomorrow

o/p: hsothnas si gnisoporp a lrig worromot

```
S = 'hsothnas si gnisoporp a lrig worromot'
R = S[::-1]
print(R)
L = S.split()
R = [ ]
for i in L:
    R.append(i[::-1])
print(' '.join(R))
```

without append

```
R = R + [i[::-1]]
```

o/p: 'worrommor lrig a gnisoporp si hsohtnas'

```
S = input('Enter Sentence:')
r = S[::-1] - [Just reversing a string]
print(r)
```

→ WAP to print all palindrome words in a given string?

```
S = input('Enter a string:')
r = [ ]
L = s.split()
for i in L:
    if i==i[::-1]:
        r.append(i)
print(r)
```

```
S = input()
L = s.split()
for i in L:
    if i==i[::-1]:
        print(i)
```

→ WAP to print sub palindrome's in a given string? which are having length of more than one character.

S = 'Santhosh loves MOMOS and TATOO Girl'

→ WAP to print fibonacci series.

14/1/Feb

Functions :

→ Functions are independent block of code which are used for performing some task.

→ Functions are used for increasing code reusability.

→ Functions are used for reducing code redundancy (Repetition)

Classifications of Functions :

We have two types of functions :

Built in functions

User Defined functions

Built in functions :

These are functions which r created by Developers of python.

Example :

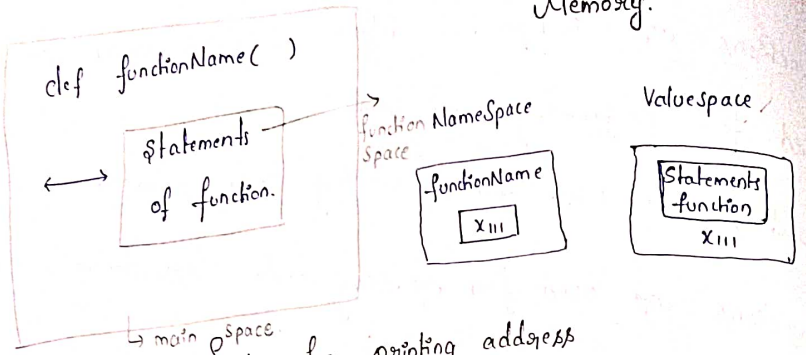
len , print( ), input( ) ....

User defined functions :

① These are the functions which r developed (or) defined by the user based on his requirements.

② def Keyword is used for defining user defined functions.

Memory.



functionName → Syntax for printing address

functionName( ) → Syntax for calling function.

Ex: def wishing( ):
print ('Happy Valentine's day')
print ('I Love u my dear')

print (wishing) # printing function address

wishing( ) # calling a function.

Classification of User defined function :

- functions without arguments and without return Type (static output)
- functions with arguments and without return Type
- functions without arguments and with return Type
- functions with arguments and with return Type.

Arguments : Arguments are Essential Elements (parameters) required for Execution of the statements of functions.

Return :

Functions without Arguments and without return Type :

This type of functions will not take any parameters for its Execution and they will not return any output as well.

Syntax :

```
def functionName( ):
    statements of functions
```

Ex :

```
def add(a,b):
    print(2+10)

add( )
add( )
add( )
add( )
```

O/P : 12
12
12
12

Functions with Arguments and without return Type :

In this function we will define arguments and we will not have return statement inside function.

Syntax :

```
def functionName (arguments):
    Statements
```

Classification of Arguments :

We have 4 types of Arguments :

1. Mandatory (or) Positional Arguments.
2. Default (or) Non mandatory Arguments.
3. Variable length arguments (*args)
4. Variable length Keyword Arguments (**kwargs)

Mandatory (or) Positional Arguments :

If you are defining a function with Mandatory (or) positional arguments compulsory we have to pass that many values while you call that functions.

[∵ n = n]

```
def add(a, b):
    print('performing addition operation')
    print('a value is', a)
    print('b value is', b)
    print(a + b)

# add()  # Error (2 missing arguments)
# add(89) Error. (1 missing argument)
# add(89, 78, 67) Error. (you have given more values)

add(23, 15) # output 68.
```

Default Arguments :

In this type arguments will have it's own default values.

Note :

① If you provide values for arguments then it will consider them as value.

② If you are not providing the values for arguments then it will consider Default values for arguments

① n<=n

```
def add(a=89, b=67):
    print('a value is', a)
    print('b value is', b)
    print('c value is', a+b)

# add()  a=89, b=67  a+b =
# add(100)  a=100, b=67  a+b =
# add(123, 231)  a=123  b=231  a+b =
# add(123, 456, 78) Error.
```

* Note : When you want to define a function with both Mandatory and default arguments then first define mandatory then only we have to define default argumen

```
def add(a, b=89):        # mandatory, default
    print('a value is', a)
    print('b value is', b)
    print(a+b)

# add() missing 1 required positional argument: 'a'
add(100)        a=100, b=89
add(123, 234)   a=123, b=234
add(123, 456, 78)  Error.
```

Variable length Arguments :-

1. In python * defines the variable length Arguments
2. If we define a function with variable length arguments then we can provide 0 to n values while calling it
3. Output format of variable length arguments Tuple

```
def add(*args):          # will takes all values
    print(args)
    print(type(args))
    summ = 0
    for i in args:
        summ += i
    print(summ)

add()  → Empty. o/p

add(100)
add(123, 234)
add(123, 456, 78)
```

Variable length Keyword Arguments :

→ In python ** defines Variable length Keyword Arguments.

→ Output format of variable length Keyword Argument is a Dictionary.

→ Once you define a function with variable length Keyword Arguments then we have to provide values parameters in the form of pairs as shown below

Argument1 = value, Argument2 = value .... Argumentn = value

```
def add(**kwargs):
    print(kwargs)
    print(type(kwargs))
    summ = 0
    for i in kwargs:
        summ += kwargs[i]
    print(summ)
```

```
add( )
add(a=10)
add(a=10, b=90, c=100)
```

Order of Defining the Arguments of a function:

position ---> Default ---> Variable length ---> Variable length Keyword Arguments

```
def function(a, b, c = 90, *args, **kwargs):
    print(a)
    print(b)
    print(c)
    print(args)
    print(kwargs)

function(78, 56, 999, 12,22,67,9,87, name='NIKKY', age=2)
         a   b   c    args           kwargs
```

Function without Arguments with return Type:

In this type of function we will define a function which is Not accepting arguments but it will have return Type

```
def function():
    statements
    return something
```

Return Statement:

Return Statement used for returning some values (or) output from function space to main space

Return output

Properties of return:

1. After return we cannot write a single statement inside that block (return is a final statement of Particular the block.)
2. By Using return we can return any type of data
3. By Using return we can return any Number of data.
4. If we return multiple data then Output format of return is Tuple.

⇒ If a function has return statement in it then it should be called in a the below mentioned ways.

Inside print function.
Inside a variable (assigning function call to a variable)
Inside Conditional Statements

5. Return will return the output to the place from where it is called.
6. By default w.if we don't provide return it will give None
7. If you give only ^ Return also it will give None.

⇒ Calling function inside print function.

It is used when we have to use the output of the function only for one time.

Example:
```
def add( )
    return 11+22

print(add( ))
```

⇒ Assigning function call to the variable:

It is used when we have to use the output function for multiple times.

Example:
```
def add( )
    return 11+22

result = add( )
print(result)
```

Function with Arguments and with return Type:

We will provide Arguments and when we call the function we provide values.

Ex:
```
def add(a,b):
    return a+b
result = add(10, 20)
print(result)
```

Packing:

wap to program to check given number is Even(or)not

If it is Even → True otherwise False.

```
def isEven(n):
    if n%2==0:
        return True → final statement of if Block
    Else:
        return False

print(isEven(10))
```

```
def Even(a):
    if a%2==0:
```

Prime (or) Not

```
def isPrime(n):
    for i in range(2, n//2+1):
        if n%i==0:
            return False
    Else:
        return True

print(isPrime(11))
```

WAP to print Even numbers in a given range.

```
def isEven(n):
    if n%2==0:
        return True
    Else:
        return False

def evenNumbers(LL,UL):
    for n in range(LL,UL+1):
        if isEven(n): [Calling inside a conditional statement]
            print(n)

evenNumbers(10,20)
```

WAP to print prime no in a given range.

```
def isPrime(n):
    for i in range(
```

```
def isPrime(n):
    if n<=1:
        return False
    for i in range(2,n//2+1):
        if n%i==0:
            return False
    Else:
        return True
def primeNumbers(LL,UL):
    for n in range(LL,UL+1):
        if isPrime(n):
            print(n)

primeNumbers(10,100)
```

Print Armstrong numbers in a given range.

```
def isArmstrong(n):
    L=len(str(n))
    temp=n
    summ=0
    while n>0:
        rem=n%10 (To Extract Last Element)
        summ+=rem**L
        n=n//10
    if summ==temp:
        return True
    Else:
        return False
def armstrongNumbers(LL,UL):
    for n in range(LL,UL+1):
        if isArmstrong(n):
            print(n)

armstrongNumbers(10,100)
```

Special numbers

```
def isSpecial(n):
    num=n
    summ=0
    while n>0:
        rem=n%10
        fact=1
        for i in range(1,rem+1)
            fact*=i
        summ+=fact
        n//=10
    if summ==num:
```

```
        return True

    Else:
        return False

def specialNumbers(LL, UL):
    for i in range(LL, UL+1):
        if isSpecial(n):
            print(n)

specialNumbers(10, 100)
```

Disarium Numbers:

```
def isDisarium(n):
    num=n
    summ=0
    L = len(str(n))
    while n>0:
        rem = n%10
        summ+ = rem** L
        n = n//10
        L- = 1
    if summ == num:
        return True

    Else:
        return False

def disariumNumbers(LL, UL):
    for i in range(LL, UL+1):
        if isDisarium(n):
            print(n)

disariumNumbers(10, 200)
```

Harshad Number

```
def isHarshad(n):
    num = n
    sum = 0
    while n>0:
        rem = n%10
        summ+ = rem
        n = n//10
    if num % summ == 0:
        return True

    Else:
        return False

def harshadNumber(LL, UL):
    for i in range(LL, UL+1):
        if isHarshad(n):
            print(n)

harshadNumber(10, 100)
```

EMIRP Numbers

```
def isEmirp(n)
    num = n
    rev = 0
    while n > 0:
        rem = n%10
        rev = rev *10 + rem
        n = n//10
    if num != rev:
    <----if num>1:
        <----for i in range(2, n//2+1):
            <----if num%i == 0:
                <----return False.
```

```
        Else :
            for i in range (2, rev//2+1) :
                if rev%i == 0 :
                    return False.

            Else :
                return True

        Else :
            return False

    Else:
        return False
def emirpNumbers(LL, UL) :
    for n in range (LL, UL+1) :
        if isEmirp(n) :
            print(n)

EmirpNumbers (1, 1000)
```

wap to print Emirp numbers in a given range.

```
def isPalindrome(n) :
    s = str(n)
    if s == s[::-1]
        return True
    else :
        return False.

def isPrime (n) :
    if n>1 :
        for i in range (2, n//2+1) :
            if n%i == 0 :
                return False

        Else :
            return True

    Else:
        return False.

def isEmirp(n) :
    rev = int ( str(n)[::-1] )
    if not isPalindrome(n) :
        if isPrime(n) and isPrime(rev) :
            return True

        Else :
            return false

    Else :
        return false

def EmirpNumbers (LL, UL) :
    for n in range (LL, UL+1) :
        if isEmirp(n) :
            print(n)

EmirpNumbers (1, 50)
```

```
def reverse(n):
    num = n
    rev = 0
    while num > 0:
        rem = num % 10
        rev = rev * 10 + rem
        num //= 10
    return rev.  = output
```

131

```
def isPalindrome(n):
    rev = reverse(n)
    if rev == n:
        return True
    Else:
        return False.
```

rev · we are storing reversed value of n

```
def isPrime(n):
    if n>1:
        for i in range(2, n//2+1):
            if n%i == 0:
                return False
        Else:
            return True
    Else:
        return False
```

```
def isPaliPrime(n):
    if isPalindrome(n) and isPrime(n):
        return True.
    Else:
        return False.
```

— x —

[Emirp]

for range:

```
def paliPrime(LL, UL):
    for n in range(LL, UL+1):
        if isEmirp(n):
            print(n)

EmirpNumbers
paliPrime(1, 50)
```

(5)

wap to print pattern-

```
n = int(input('Enter rows'))
def pattern(n):
    sp = n//2
    st = 1
    for i in range(n):
        for j in range(sp):
            print(' ', end=' ')
        for k in range(st):
            print('*', end=' ')
        if i < n//2:
            sp -= 1
            st += 2
        Else:
            sp += 1
            st -= 2
        print()
```

i = 0, 1, 2, 3, 4

sp: n//2
9//2 = 4

(6) pattern(9)

wap to print pattern

```
q = 5
def pyramid(n):
    st = 1
    sp = n-1
    for i in range(n):
        for j in range(sp):
            print(' ', end=' ')
        for k in range(st):
            print('*', end=' ')
        sp -= 1
        st += 2
        print()
pyramid(5)
```

Types of Variables in Functions :

In functions we have 3 types of Variables.

1. Local variable
2. Global variable
3. Non local variable.

} Scope of -Accessibility.

Local variable :

① These are variables which are created inside function space.

② Local variables can be accessed and Modified only in that function space.

③ We cannot Access Local variable in Main space.

Global variable :

① These are variables which are created outside the function and accessed and modified inside the function.

② Inorder to make a variable into global we have to use global keyword.

Syntax :

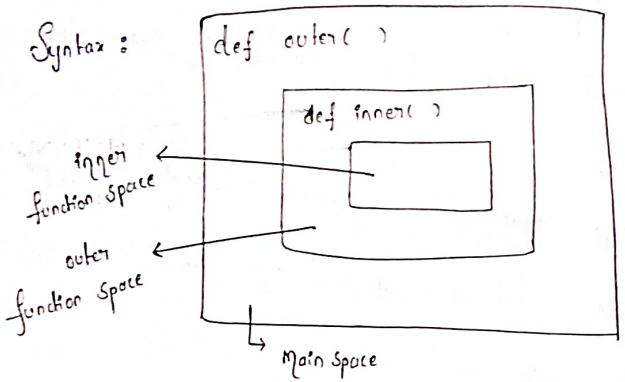
global variable1, variable2 .... variable n.

```
a, b = 10, 20
def display():
    global a, b
    print(a, b) # 10, 20.
    a-=5
    b+=5
    print(a,b) # 5, 25.
display()
print(a, b) # 5, 25.
```

Nested function : Process of defining a function inside another function.

Syntax :

```
def outer():
    def inner():
```



Ex :

```
def outer():
    print('outer is started')
    def inner():
        print('inner is started')
        print('inner is Ended')
    inner()
    print('outer is ended')
outer()
```

Non Local variable :

1. non local variable are used in nested functions, to make variables defined in parent (outer) function to be accessed and modified in child (inner) function.
2. To make normal variables into non local we have to use non local keyword.

Syntax :

```
nonlocal variable1, variable2, ... variableN.
```

```
def outer():
    a, b = 10, 20
    print(a, b) # 10 20
    def inner():
        nonlocal a, b (If we not provide nonlocal keyword it will throw Error)
        print(a, b) # 10, 20
        a-=5
        b+=5
        print(a,b) # 5, 25.
    inner()
    print(a,b) # 5, 25
outer()
```

write diff b/w local and global variable.

Local	Global
• Local variables are defined inside functions and can be accessed, modified only in function Space.	• Global variables are created outside the function and accessed, modified inside the function.
• Scope of Accessibility only inside function Space	• Inorder to make it global we have to use Global Keyword.
• There is no need of Keyword.	• Scope of Accessibility Main to function Space

write diff b/w global and non local.

Global	Non-local
• Global variables are created outside the function and accessed modified inside the function.	• Non-local variables are created in outer function Space and can be accessed modified inside inner function Space
• Inorder to make it global we have to use global keyword.	• Inorder to make it non local we have to use non local keyword.
• Scope of Accessibility is main Space to function	• Scope of Accessibility is outer function Space to inner function Space.

fibo program :

```
def fibo(n):
    f = 0
    s = 1
    if n==1 or n==2:
        return n-1
    for i in range(n-2):
        T = f + s
        f, s = s, t
    return t
print(fibo(5))
```

fibo program Continuous :

```
def fibo(n):
    f = 0
    s = 1
    if n==1 or n==2:
        return n-1
    for i in range(n-2):
        T = f + s
        f, s = s, T
    return T
for j in range(1,10):
    print(fibo(j))
```

Given string s = 'aaabbcda'

```
s = 'aaabbcda'
x = ''
c = 1
for i in range(len(s)-1):
    if s[i] == s[i+1]:
        c += 1
    else:
        x = x + str(c) + s[i]
        c = 1
x = x + str(c) + s[i+1]
print(x)
```

o/p : 3a2b1c1d1a

```
# s = 'aaabbcda'
x = ''
c = 1
for i in range(len(s)-1):
    if s[i] == s[i+1]:
        c += 1
    else:
        x = x + s[i] + str(c)
        c = 1
x = x + s[i+1] + str(c)
print x
```

Recursion function: It is the process of calling function within itself until the given terminating condition satisfied

Advantages of Recursion:

→ Reduces Code Complexity

→ When you want to repeat the same process recursion is the best way

→ Recursion will be a very efficient and mathematically elegant approach to programming.

Disadvantages of Recursion:

Excess amount of memory consumption.

Syntax:

```
def recursion( ):
    If Condition :
        terminating block
    recursion
```

Print write sum of first n natural numbers by using Recursion &

```
def summ(n):
    if n==0:
        return 0
    return n+summ(n-1)
print(summ(11))
```

flow of Execution :

```
summ(11)
  n=11
  If 11==0:
  return 11+sum(3)

Summ(3):
  n=3
  if 3==0:
  return 3+sum(2)

def summ(n):
If (n==0):
  return 0
return n+sum(n-1)
print(summ(5))
sum(2):
  n=2
  If 2==0:
  return 2+summ(1)

Summ(1)
  n=1
  If 1==0:
  1+summ(0)

summ(0)
  n=0
  if 0==0:
```

Factorial of given number using Recursion

```
n=int(input( ))
def fact(n):
    if n==1:
        return 1
    return n*fact(n-1)
print(fact(5))
```

fact=5 n=5 If 5==1: return 5*fact(4)

fact=4 n=4 If 4==1: return 4*fact(3)

fact=3 n=3 If 3==1: return 3*fact(2)

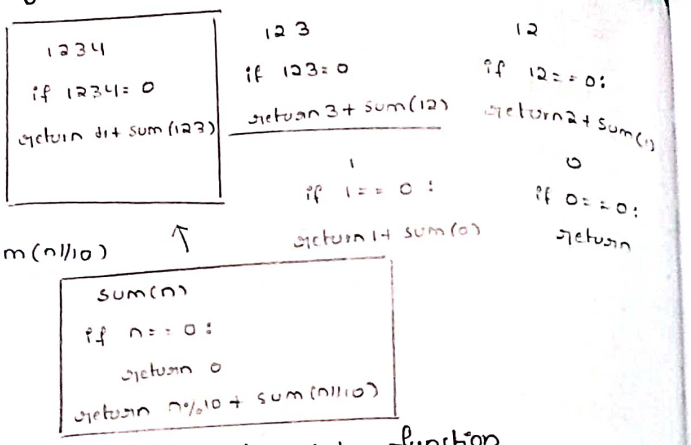
```
def fact
If n==1:
return 1
return n*f(n-1)
print fact(5)
```

fact=2 n=2 If 2==1: return 2*fact(1)

fact=1 n=1 If 1==1: return 1

WAP sum of individual digits of given number ?

```
def summ(n):
    if n == 0:
        return 0
    return n%10 + sum(n//10)
print (summ(1234))
```



Diff between function and Lambda function

Functions	Lambda
• def Keyword used to define function	lambda Keyword used to define.
• Name will be available for a function	Anonymous function without a name
• n arguments we can define	n arguments we can define
• n statements / Expressions can be written	we can write only one Expression.
• If you have written return some content we have to use return Keyword.	• By default lambda functions will return output of the Expression.

Lambda function :

1. Lambda functions are Anonymous (unknown) function
2. Lambda functions are Single line function

Syntax :

lambda arguments : Expression
(Keyword, multiple, only one condition.)

Note :

1. Lambda function will take multiple Arguments
2. Lambda function will take only one Expression, this Expression is Evaluated and returned.

lambda with one argument

```
add = lambda a : a+10        o/p : 232
print (add (222))
```

lambda with one argument returning True or False

```
iseven = lambda a: a%2 == 0      o/p : True
print (iseven(222))              222 % 2 == 0
                                 T
```

lambda with two arguments

```
mul = lambda a,b : a * b         o/p : 200
print (mul (10,20))
```

lambda with three arguments

```
mul = lambda a,b,c : a * b * c
print (mul (10, 20, 10))
```

Map function is responsible for Creating object with Stores the o/p of process of Extract Element from CDT with the help of function address./ boolean Values will not give in map.

Map function:

Map function in python takes in a function and a iterable one type user input only

* map( ) function return a map object (which is an iterator) of the result after applying given function to each item of a given iterable (List, Tuple Etc)

Syntax:

map (function address, sequence/CDT)

Parameters:

function address: It is a function to which map passes Each Element of given iterable.

Iterable: It is a iterable which is to be mapped.

map with Normal function

```
def add(n):
    return n+10
print (list(map(addition, [11, 22, 33, 44])))
```

map with lambda function

```
print (list(map(lambda n: n+10, [11, 22, 33, 44, 55])))
```

Note:

No. of inputs = No. of outputs.

```
L = [11, 22, 33]
def multiply(n):
    return n * 10
map (multiply, L)
print (tuple(map(lambda n: n* 10, [11, 22, 33])))
```

//

Space Separated Value as input of List

input format = '100 220 300 400 500'

output format = [100, 220, 300, 400, 500]

```
input( )
L = list(map(int, input( ).split( )))
print(L)
```

Each and Every Element Converted into int and stored in List

//

input 100 11 222 125 71 66 87

output:

```
L = list(map(int, input( ).split( )))
def isPrime(n):
    if n>1:
        for i in range(2, n//2 + 1):
            if n%i == 0:
                return False
        Else: Else:
            return True
    Else:
        return False
summ = 0
for n in L:
    if isPrime(n):
        sum+ = n
print(summ)
```

Filter Method : filter method filter the given Sequence with the help of a function that tests Each Element in Sequence is to be true (or) not

o/p <= i/p

Syntax :

filter (function, Sequence)

Parameters :

functions : function test each element of Sequence true (or) not.

Sequence : Sequence which needs to be filtered, it can be set, lists, tuple or containers of any iteratorys.

filter with normal function

```
def isEven(n):
    if n%2==0:
        return True
    Else:
        return False
print (list (filter (isEven, [11, 22, 33, 44])))
```

o/p : [22, 44]

Note : No. of outputs will be less than (or) Equal to no. of inputs of iterable.

filter with lambda function :

```
L = [11, 22, 33, 44]
print (list (filter (lambda n: n%2==0, L)))
```

o/p : [22, 44]

wap to print filter prime numbers in a given List.

```
def is_Prime(n):
    if n>1:
        for i in range (2, n//2+1):
            if n%i==0:
                return False
        Else:
            return True
    Else:
        return False
L = [11, 22, 33, 44, 17]
print (list (filter (is_prime, L)))
```

o/p :

[11, 17, 13]

WAP to filter Armstrong numbers from given List

```
def isArmstrong(n):
    num=n
    Summ=0
    s=len(str(n))
    while n>0:
        r=n%10
        Summ+=r**s
        n=n//10
    if Summ==num:
        return True
    Else:
        return False
l=[1,54,67,371]
print(list(filter(isArmstrong,l)))
```

WAP to filter Special numbers from given List.

```
def isSpecialNumber(n):
    num=n
    Summ=0
    while n>0:
        rem=n%10
        fact=1
        for i in range(1,rem+1):
            fact*=i
        Summ+=fact
        n=n//10
    if Summ==num:
        return True
    Else:
        return False
l=[1,21,145,859]
print(list(filter(isSpecialNumber,l)))
```

WAP to filter Disarium Numbers (or) Not

```
n=Eval(input(Enter List))
def isDisariumNumber(r):
    Summ=0
    num=r
    b=len(str(r))
    while r>0:
        rem=r%10
        Summ+=rem**b
        r//=10
        b-=1
    if Summ==num:
        return True
    Else:
        return False
print(list(filter(isDisarium,n)))
```

WAP to filter Harshad Numbers from list ?

```
def isHarshad(n):
    num = n
    summ = 0
    L = len(str(n))
    while n > 0:
        rem = n % 10
        summ = summ + rem ** L
        n = n // 10
    if summ % num == 0:
        return True
    Else:
        return False
L = [1, 2, 3, 11, 5, 221, 89, 76]
print(list(filter(isHarshad, L)))
```

//.

WAP to filter palindrome

```
def isPalindrome(n):
    num = n
    rev = 0
    while n > 0:
        rem = n % 10
        rev = rev * 10 + rem
        n = n // 10
    if rev == num:
        return True
    Else:
        return False
L = [1, 11, 9, 21, 121, 31, 89]
print(list(filter(isPalindrome, L)))
```

27/Feb

Reduce : The reduce(fun, seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in sequence passed along

This function is defined in "functools" module

```
import functools
```

[reduce functions takes two arguments

[only one output from reduce function] irrespective of number of inputs]

```
# initialising List
L = [100, 200, 1100, 600, 111, 222, 555]
# using reduce to compute sum of List
print ("The sum of the list Elements is :", end=" ")
print (functools.reduce(lambda x,y: x+y, L))
```

L = [100, 200, 1100, 600, 111, 222, 555]

reduce (lambda x,y: x+y, L) [flow of Execution]

① reduce (will Extract Elements from given List)

x = 100, 200 (as it takes two arguments)

↓

lambda → (performs summation) x + y = 300

↓ (300)

reduce [has 300 so it extracts 2nd Element x = 300, y = 400 [from List]

↓ gives to reduce.

lambda → 300 + 400 = 700

↓

reduce x = 700, y = 600

↓

lambda 700 + 600 = 1300

↗ reduce x = 1300, y = 111

↓

lambda 1300 + 111 = 1411

↓

reduce x = 1411 y = 222

→ lambda 1411 + 222 = 1633

↓

reduce x = 1633 y = 555

↓

lambda → x + y → 1633 + 555 → 2188

↓

reduce

→ Reduce function performs three Operations

↳ Summation
Maximum
Minimum

Ternary Operator : Ternary Operator is responsible for representing if Else condition in single line.

Syntax : TrueValue if condition Else FalseValue.

if condition is True consider TrueValue
if condition is False consider FalseValue.

using reduce to Compute maximum Elements from List

```
L = [100, 200, 400, 600, 111, 222, 555]

print ("The maximum Element of the List is :", end=" ")
print (functools.reduce (lambda a,b: a if a>b else b, L))
```

```
import functools
L = [100, 200, 400, 600, 111, 222, 555]
reduce
print ("The max of the List Elements is :", end=" ")
print (functools.reduce (lambda a, b : a if a>b else b, L))
```

reduce extracts a = 100 b = 200
↓
lambda → checks condition if a>b (a=100, b=200) (F)
else b → 200

reduce → a = 200, b = 400
lambda → a > b C- False → Else b → 400
reduce (400) → a: 400, b: 600
lambda → a > b C- False → Else b = 600
↓
reduce (600) → a = 600, b = 111
lambda → a > b → Cond-T → 600 a
reduce (600) → a = 600 b = 222
↓
lambda → a > b → cond-T → 600 = a
reduce (600) → a = 600 b = 555
↓
lambda → a > b → cond-T → a = 600.

reduce will return output as 600 which is maximum Value.

using reduce to compute minimum value

```
import functools
L = [100, 200, 400, 600, 111, 222, 555]
print (functools.reduce (lambda x, y : x if x<y else y, L))
```

flow :

Importing : Importing is a process of Extracting the properties / functionalities of one module into another module

- Importing can be achieved by using Import Keyword

We can import properties in 2 following ways

1. Importing Entire Module (all content)
2. Importing Specific functions / classes

Importing Entire Module (all content) :

1. When we want to access all the content from the module then we import Entire module
2. Syntax for importing Entire module :

```
import modulename
```

3. Syntax for accessing functions / classes if we import Entire module.

For Functions :

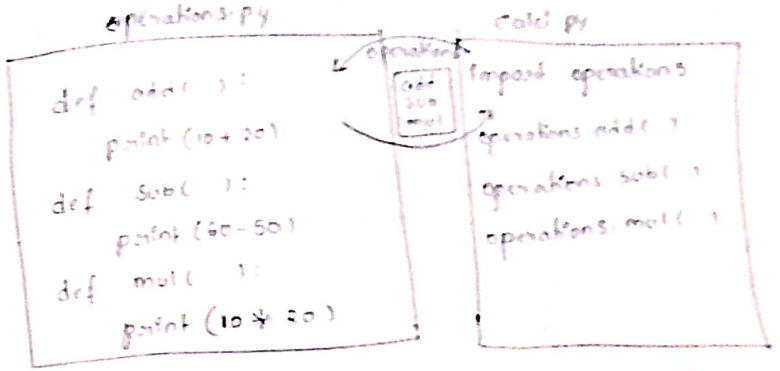
```
ModuleName.functionName(arguments)
```

note : arguments are not mandatory.

For classes :

```
Modulename.className(arguments)
```

note : arguments are not mandatory



Aliasing : Aliasing is a process of providing an alternate name for imported module or functions or classes

2. We can achieve aliasing by using as Keyword
3. Aliasing changes will be applicable only inside the respective module where you have applied

Syntax for providing an alternate name for Entire module :

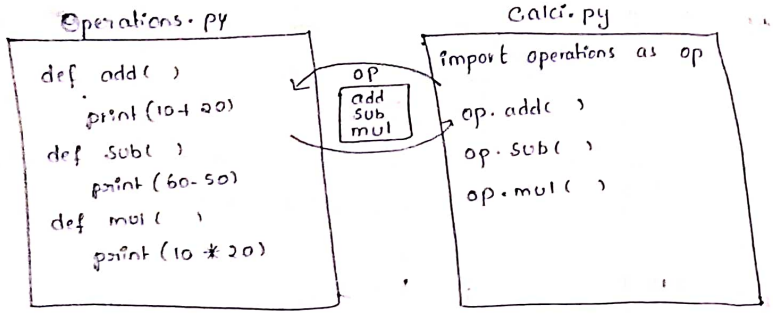
```
import modulename as AliasingName
```

Syntax for accessing functions once after aliasing

```
Aliasingname.functionname()
```

Note :

After aliasing we cannot use actual name of module for accessing the functions



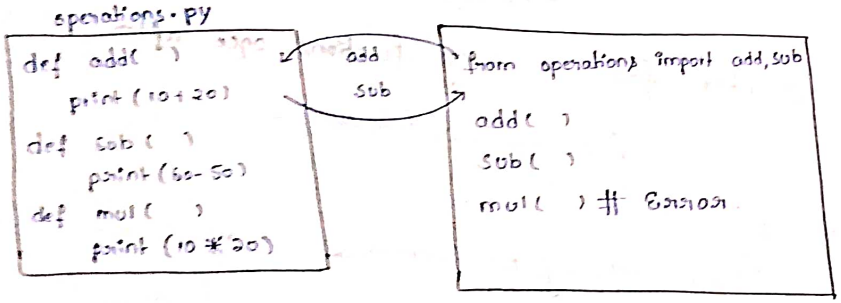
Importing Specific functions from a module :

1. We can import specific functions from module by using from, import keyword.
2. Syntax for importing specific functions

from modulename import functionName1, ---- functionNameN

3. Syntax for accessing the function

functionName ( )



from operations import add as a, sub as s

s( )

print (a( ))

Providing Alternate name for imported functions

Syntax :

from ModuleName import functionName1 as AliasingName,
functionName2 as AliasingName

Syntax for accessing the function

AliasingName ( )

Importing All functions Directly from a module:

Syntax :

from ModuleName import * → All functions

Syntax for accessing the function :

functionName ( )

from operations import *

print (add( ))
sub( )
mul( )

Note : If you are Executing a module directly the function call should be done

If you are Executing module from another module function call should not be done.

Restricting the function calls immediately after Importing (or) Prioritization of Executing the Imported functions.

1. We can set prioritization by using __name__ double underscore.

⇒ __name__ returns __main__ as output for current module Execution.

⇒ __name__ returns actualname as output for the imported module Execution.

⇒ __name__ gives → Name of the Module.
- as __main__ → when we Execute a Module Directly
- as Actualname → when we Execute it from Another Module

Operations.py

```
def add()
    return 10+20
def sub()
    print(60-50)
def mul()
    print(10*20)
print('name of operations is', __name__)
if __name__=='__main__':
    print(add())
    sub()
    mul()
```

Import operations

operations.sub()

Package:

- Package is Collection of python modules along with __init__.py
- __init__.py is a Blank python module which makes a folder of python modules as a python package.

Importing Entire Module (all content) of another package

1. When we want to access all the content from the module then we import Entire module
2. Syntax for importing Entire module

   from packageName import moduleName1, ...moduleName2
3. Syntax for accessing functions / classes if we import Entire module.

For functions:

ModuleName.functionName(arguments)

note. arguments are not mandatory.

For classes:

ModuleName.className(arguments)

note. arguments are not mandatory

* Aliasing a imported module of another package :

* from packageName import moduleName as AliasingName

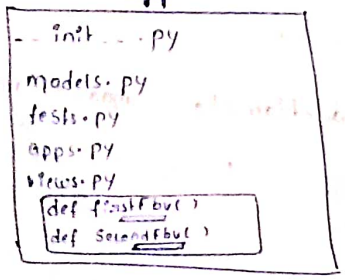
  → AliasingName.functionName( )

  L, Syntax for accessing functions after aliasing

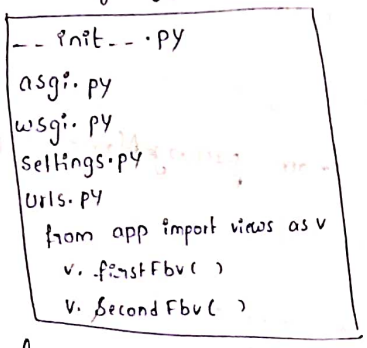
  L, Syntax for accessing providing an Alternate name for Entire module

Note : After aliasing we can't use actual name of module for accessing the functions.

app



project



Importing Specific functions from module of another package.

1. We can import specific function from module by using both from, import keywords

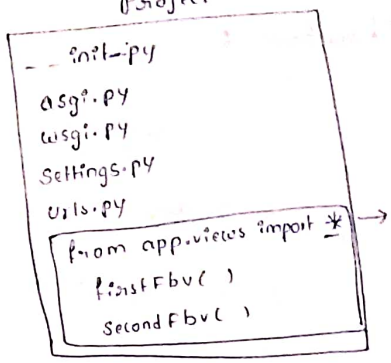
2. Syntax for importing specific functions.

from packageName.moduleName import functionName1, ... functionNameN

Syntax for accessing function

functionName( )

Project



→ all functions

Providing alternate name for imported functions of another package module

Syntax :

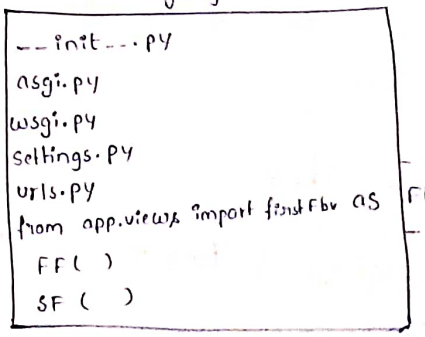
from packagename.ModuleName import functionName as Aliasing,

functionName as AliasingName

Accessing the function :

AliasingName( )

Project



SecondFbv as SF

3/Mar/23

Oops

Object Oriented Programming Structure :

1. Python is a multi paradigm programming Language.
   It Supports different programming approaches.
   A paradigm is a standard, perspective or set of ideas
2. It Supports both functional and oops
3. One of the popular approaches to solve a programming problem is by Creating objects
4. In oops every real time entity is treated as an object.
   An object has two characteristics
   - States → Attributes (Properties)
   - Behavior (functionalities)
5. Logically a object is Combination of (states ↑ Attributes) and behaviors.
6. To Create object we need design
7. Class is a design / prototype / Template of an object
8. From one ^ Class we can Create multiple Objects
   Once we define a class we need to create an object to allocate memory

Let's Example :

A parrot can be an object, following properties it has.

1. name, age, color as attributes
2. Singing, dancing as behavior

The Concept of oop in python focuses on Creating reusable code.

This Concept is also Known as DRY (Don't Repeat Yourself)

Import terms in Oops :

Class — Blue Print

Object — Entity which has states and Behavior

Instance — Current ^ class object which u are dealing with

Variables — defining states of an objects

Methods — defining behavior of an object

Important Concepts in Oops :

Encapsulation : Binding properties and functionalities of an Object under one roof.

Aggregation : Process of Creating / use of an object of one class in another class

Inheritance : Acquiring all properties of one class into another class (derived/child class)
Create a class → Parent / Base / class

Polymorphism : One method behaving in different ways (task can be performed)

Abstraction : Hiding Internal functionalities. details

Process of Oops :

Create a class (Blue print)

Create object from class

Use that Created objects

Creating a class :

Syntax :

```
class ClassName :
    States
    Behaviors
```

tab / indentation

Process that happens in memory once we Create a class :

- A dictionary will get created under className
- State and behavior names are considered as Keys and their values as values in dictionary

2. Create Object :

Syntax :

objectName = ClassName( ) → mandatory

Process that happens in memory once we create a object :

1. A dictionary will get Created under objectName
2. Derives all the class dictionary properties into Object Dictionary
3. Checks for Constructor if it is available it will be called.

```
class A :
    x = 10
    y = 20
    def hai(self):
        Code.
```

A → class dictionary

Keys	Values
x	10
y	20
hai	—

OA → object dictionary

Keys	Values
x	y_{111}
y	y_{222}
hai	y_{333}

x_{111}

* Every Class is a DataType of their object

Class → objects → states (Attributes) / (Data Members)
- C.A — class Attributes
- O.A — object Attributes
- C.M — class Member
- O.M — object Member

Behavior → (Methods)
- C.M (class Method)
- O.M (object Method)
- S.M (static Methods)

States :

1 States are properties (information) of an object

2 By using Variables we can define the states of an objects

Classification of states of an objects

States of an object are classified into 2 types

1. Generic properties (information) of an object
2. Specific properties (information) of an object

Generic Properties (information) :

1. These are common properties of all objects
2. Values for this generic properties will not change from object to objects
3. Generic properties are defined by using class Variable

Class Variable :

1 Class Variable are those Variables which defined inside the class and outside the method.

```
Class Bank :
    bank_name = SBI
    bank_roi = 7
    bank_branch = 'Bangalore'
Meghana = Bank( )
Srujana = Bank( )
```

→ { objects we created inside our class }

Bank

bank Variables

	Keys	Values
B-1	bank_name	SBI
B-2	bank_roi	7
B-3	bank_branch	'Bangalore'

X_{111}

Meghana

	Keys	Values
M-1	bank_name	B-1
M-2	bank_roi	B-2
M-3	bank_branch	B-3

Y_{111}

Srujana

	Keys	Values
S1	bank_name	B-1
S2	bank_roi	B-2
S3	bank_branch	B-3

y_{222}

Syntax for Accessing class Variable by using object class :

ClassName.ClassMemberName (Variable)

print(Bank.bank_name) # "SBI"

Syntax for Accessing Class Variable by using object

ObjectName.ClassMemberName (Variable)

print(meghana.bank_name) # "SBI"

Modifying class member with class.

Syntax : ClassName.ClassMemberName = NewValue

Note : If we modify class member with Class Then all objects also get affected

```
Bank.bank_roi = 6
print (" After modifying class member with class")
print (Bank.bank_roi) # 6
print (meghana.bank_roi) # 6
print (srujana.bank_roi) # 6
```

Creating New Class Member with ClassName

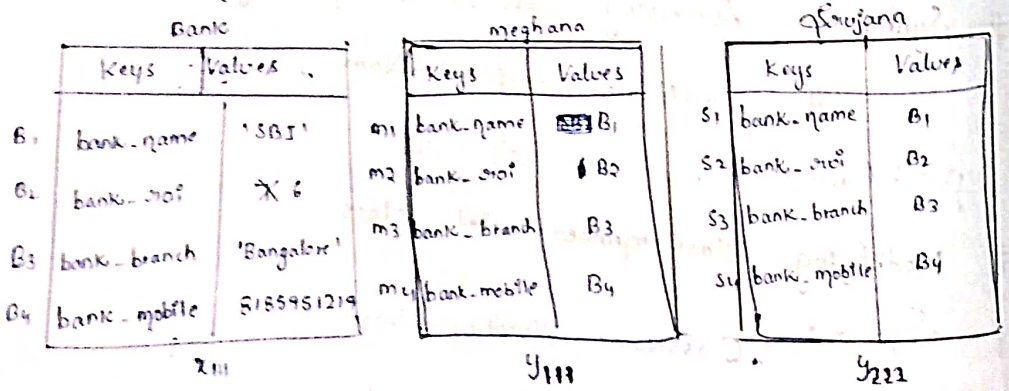
Syntax :

```
ClassName.NewClassVariable = Value
```

Note :

It will create a NewClassVariable in class and its objects

```
Bank.bank_mobile = 8185951219
print (Bank.bank_mobile) # 8185951219
print (meghana.bank_mobile) # 8185951219
print (srujana.bank_mobile) 8185951219
```



Modifying Class Member with Object

Syntax :

```
ObjectName.ClassVariable = NewValue
```

```
meghana.bank_roi = 5
print (Bank.bank_roi) # 7
print (meghana.bank_roi) # 5
print (srujana.bank_roi) # 7
```

Note : If we modify class Variable by using object then it will modify only in that object

Note :

→ Creating classVariable with object will be considered as a specific property but not a generic property

→ that Variable will be considered as a object Variable / Instance Variable

Specific Properties :

1. Specific Properties is used for defining the uncommon data of the object
2. Specific property will Vary from one object to other
3. Specific property can be defined by using object (or) instance Variables (inside constructor)
4. Specific properties cannot be accessed by using the class.

Syntax for creating object members :

```
ObjectName.ObjectVariableName = Value.
```

Syntax for accessing object members :

```
ObjectName.ObjectVariableName
```

Constructor : Constructor is a special method which is called implicitly when ever an object is created. [by default]

Example :

```
Class student( ):
    def __init__(self):
        print('executing__init__')

s = student( ) # calls constructor
```

↓ object is created it checks for Constructor and it calls it

- It is a special method which is called automatically by the interpreter
- we call __init__ as constructor because it is used for initializing the object members
- Self is mandatory and / first argument we have to pass for __init__
- Self is used for considering instance address

Types of Constructors :

- Constructor with no arguments
- parameterized ~~argu~~
- Constructor with default arguments.
  ↳ (obj, n, a, ac, bal, a=18)

```
Class Bank :
    bank-name = 'SBI'
    bank-roi = 7
    bank-branch = 'Kizikistan'
    def __init__(self, n, ac, bal, a=18):
        print(self)
        print('this is called')
        self.name = n
        self.age = a
        self.account = ac
        self.balance = bal
meghana = Bank('Meghana Shetty', 20, 1234567, 987654)
srujana = Bank('Srujana reddy', 21, 765432, 345678)

print(srujana.name)
```

(n → mandatory; * a=18)

Bank class

	Keys	Values
B1	bank - name	'SBI'
B2	bank - ac°	7
B3	bank - branch	'KIZIKISTAN'
B4	-- init --	=

x111

Meghana object (1)

	Keys	Values
M-1	bank - name	B - 1
M-2	bank - ac°	B - 2
M-3	bank - branch	B - 3
M-4	-- init --	B - d1
M-5	name	'meghana shetty'
M-6	age	20
M-7	account	1234567
M-8	balance	987654

y111

Srujana (object 2)

	Keys	Values
S-1	bank - name	B - 1
S-2	bank - ac°	B - 2
S-3	bank - branch	B - 3
S-4	-- init --	B - 4
S-5	name	'Srujana reddy'
S-6	age	21
S-7	account	765432
S-8	balance	345678

y222

School Jspiders

	Keys	Values
S-1	school - name	'Jspiders'
S-2	School - location	'Bangalore'
S-3	School - principal	'Harshad'
S-4	-- init --	=

x111

Student madhu

	Keys	Values
st-1	School - name	S - 1
st-2	School - location	S - 2
st-3	School - principal	S - 3
st-d1	-- init --	S - 4
st-5	~~student~~ name	'madhu'
st-6	age	13
st-7	mobile	8185951219
st-8	Class	10 ~~python~~

y111

Class ~~Jspiders~~ School :

    school_name = 'Jspiders'
    school_location = 'Bangalore'
    school_principal = 'harshad'

Generic properties

    def __init__(self, n, a, m, c):
        print(self) (not needed)
        print('this is called')
        student self.name = n
        student self.age = a
        student self.mobile = m
        student self.class = c

Specific properties

madhu = School('Madhu', 13, 8185951219, 10)

~~check~~

Methods / Behavior :

- Behaviors / Methods are functions that are used to perform some operations on states (properties) of the objects.
- The Methods are Classified into 3 Types. Based on type of Values they are going to Access, Behaviors are classified into 3 types methods.

1. Object Method
2. Class Method
3. Static Method.

Object Method :

- Objects Methods are used for Accessing and Modifying the Specific properties of an object
- They are purely belongs to objects only
- Objects Methods can be accessed directly by Objects
- If we want to access object Methods using Class we have to use object reference

Syntax : ClassName. ObjectMethod Name (obj reference, Arguments)

- Constructor is a Specialy Object Method.
- Object Method will take one Mandatory Argument that is self
- self is used for referring to Instance address of object

Syntax for defining Object Methods

```
Class A :                                  → ClassName
    def object_method_name (self, arguments):   → Instance / object Address
            statements of object-method

oa = A( )
↳ object
```

Syntax for Accessing object Methods by Using Objects

⇒ objectName. object_method-Name (Arguments)

Syntax for Accessing object Methods by using Class :

⇒ ClassName. Object_method-Name (obj reference, Arguments)

↳ mandatorily we have to provide which is object is calling the object Method

```
Class Bank :

    bank_name = 'SBI'
    bank_no = 7
    bank_branch = 'KIZIKISTAN'

    def __init__(self, n, a, ac, bal):
        self.name = n
        self.age = a
        self.account = ac
        self.balance = bal

    def display_specific(self):
        print(f'name of customer is {self.name}')
        print(f'age of customer is {self.age}')
        print(f'account no. of customer is {self.account}')
        print(f'balance of customer is {self.balance}')

    def withdraw(self, amount):
        if self.balance > amount:
            self.balance -= amount
            print('withdrawal is Successful')
        Else:
            print('Insufficient Balance')

    def deposite(self):
        amount = int(input('Enter amount'))
        self.balance += amount
        print('Deposit is Successful')
```

```
meghana = Bank('Meghana Shetty', 20, 1234567, 987654)
Srujana = Bank('Srujana Reddy', 21, 765432, 345678)

# meghana.display_specific()
# srujana.display_specific()
```
→ Calling object method name using objects as we can call directly we have no arguments

```
Bank.display_specific(meghana)
```
→ obj reference

→ Calling object method name using class Name

```
srujana.withdraw(200000)
print(srujana.balance)
srujana.deposit()
```

Bank

	Keys	Values
B-1	bank_name	SBI
B-2	bank_no	7
B-3	bank_branch	'KIZIKISTAN'
B-4	__init__	=
B-5	display_specific	=
B-6	withdraw	=
B-7	deposite	=

Meghana

	Keys	Values
M-1	bank_name	B-1
M-2	bank_no	B-2
M-3	bank_branch	B-3
M-4	__init__	B-4
M-5	display_specific	B-5
M-6	withdraw	B-6
M-7	deposite	B-7
M-8	name	'Meghana Shetty'
M-9	age	20
M-10	account	1234567
M-11	balance	987654 (it will modified)

Srujana

	Keys	Values
S-1	bank_name	B-1
S-2	bank_loc	B-2
S-3	bank_branch	B-3
S-4	__init__	B-4
S-5	display_specific	B-5
S-6	withdraw	B-6
S-7	deposite	B-7
S-8	name	'Srujana Reddy'
S-9	age	20
S-10	account	1234567
S-11	Balance	987654

y_{111} → It will be modified //

→

```
Class School :

    School_name = 'Jspiders'
    School_loc  = 'Bangalore'
    School_principal = 'Harshad'

    def __init__ (student, n, a, mo, c, m):
        Student.name = n
        Student.age = a
        Student.mobile = mo
        Student.class = c
        Student.marks = m

    def display_Specific (self):
        print (f'name of Student is {student.name}')
        print (f'age of Student is {student.age}')
        print (f'mobile of Student is {student.mobile}')
        print (f'class of Student is {student.class}')
        print (f'marks of Student is {student.marks}')

    def pass (student self, marks m):
        Student.marks
        passed = int (input (Enter marks))
        if passed > marks:
            student.standard += 1
            print (f'{student.name} is Successfully Promoted')
        Else:
            print ('fail')

Chaitra = School ('Chaitra Poornima', 7, 8185951219, 95)
haneesh = School ('haneesh Aditya', 6, 7676330226, 1, 95)
Chaitra.display_specific ( )
Chaitra.passed ( )
print (Chaitra.standard)
#
```

Nested - function returning integer as output

```
def outer (arg) :   -> defining Outer function (1)      [-> argument]
    print ('first line of outer')  <- (3) outer functions Starts Execution after Calling
    print (arg) (4) # 23
    def inner ( ) : -> defining inner - function (5)
 (7)    print ('first line of inner')
 Inner loop starts Execution
    (8) print (arg)
    (9) print ('last line of inner')
    inner ( ) -> calling inner - function (6)
    print ('last line of outer') (9)
    return 100 (return statement returns Value from function Space to main Space)
(2) result = outer (23) (Calling of outer function (2).
    > result will take return value Variable
print (result)
    -> print -> result (which contains return Value 100)
```

O/p :
first line of outer
23
first line of inner
23
last line of inner
last line of outer
100

returning Inner function address from Outer function Space to main Space.

```
def outer (arg) : -> defining outer function   [-> arguments]
    print ('first line of outer')
    print (arg)
    def inner ( ) : -> defining inner function
        print ('first line of inner')
        print (arg)
        print ('last line of inner')
    print ('last line of outer')
    return inner
result = outer(23) # result = Inner function Address
print ('*' * 30) to Separate
print (result) prints Inner function Address
result ( ) Inner function gets a call
```

=> O/p :
first line of outer
23
Last line of outer
* * *
<function. outer . <locals>.inner at 0x
00000D1A
first line of inner
23
Last line of inner

Passing another Function address as a value to the argument

```
def outer(arg) : # arg = hai function address
    print ('first line of outer')
    print (arg) hai function address.
    def inner() :
        print ('first line of inner')
        print (arg) hai function address
        arg() hai function gets a call
        print ('last line of inner')
    print ('last line of outer')
    return inner → inner function address.

def hai() :
    print ('hai is started')
    print ('end of hai')

Variable
↑
result = outer(hai) # result = Inner function address
print (result) print inner function address.
result() Inner function gets a call.
```

O/P :
```
first line of outer.
hai function address
last line of outer
Inner function
first line of inner
hai fonction add
hai started
End of hai
last line of inner
```

Decorators :-

- Decorators are used for adding additional functionalities to Decorated function by using Decorator function without modifying implementation of Decorated function.
- Decorators are Very powerful and useful feature in python
- Used for adding additional features and modifying behavior of functions (or) Classes.
- Decorators allows us to wrap another function in Order to Extend behavior of wrapped function without permanently modifying it

Normally functions can be created inside another function and we can pass functions as an Argument as well.

In order to perform Decoration we need.

- Decorator function
- Decorated function

Decorator Function : By which we perform Decoration Operation.

Properties of Decorator functions :

Rules to be followed.

① It should be of function with one argument

② Nested function (It should be)

③ Function with return Type.

DecoratedFunction :

It is function on which we perform Decoration Operation.

Properties of decorated function :

It can be of any type (no specific properties for decorated function)

```
def decoratorF(arg):
    ---
    ---
    return
```

```
@ decoratorF # decoratedF = decoratorF(address of decorated Function)
def decoratedF( ):
    ---
    ---
```

```
def outer(arg):
    print('first line of outer')
    print(arg) → # hai function address
    def inner( ):
        print('first line of inner')
        arg( ) (arg holds hai function address when we call arg( ) hai function gets started
        print('Last line of inner')
    print('Last line of outer')
    return inner (→ inner function address)
@ outer # hai = outer(Address of hai) [outer function gets a call by using address of hai as an arg]
    # hai = Innerfunction Address
def hai( )
    print('Starting of hai')
    print('Last statement of hai')
hai( ) → [As hai holds inner function address
          when we call hai( ) inner function gets started.
```

O/p :

```
first line of outer
<function hai at
Last line of outer
first line of inner
Starting of hai
Last statement of hai
Last line of Inner.
```

13/1/20

Class Methods :

→ Class Methods are used for accessing and Modifying class Members/Variables.

→ cls is a mandatory Argument for class Method.

→ cls used for referring class Address

→ Each and Every class Method be decorated with classmethod decorator i.e., @ classmethod

→ we can access class methods with both object as well as class but it modifies class member directly.

Syntax for Creating Class Method :

```
Class ClassName :
    @ classmethod
    def methodname (cls, arguments ):
        pass
```

Static Methods : (additional functionalities

→ Static Methods are neither belongs to Class nor objects

→ No need of any mandatory Arguments

→ by using static methods we cannot access (or) Modify class (or) object state.

→ These Methods are defined just to get some data while Execution of object (or) class Methods

→ Static Methods are decorated by using staticMethod decorator. i.e., @ staticMethod

Syntax :

```
Class ClassName :
    @staticmethod
    def method( ):
        Content
```

Example :

Bank

	Keys	Values
B-1	bank_name	'SBI'
B-2	bank_roi	7 5
B-3	bank_branch	'KIZIKSTHAN'
B-4	__init__	=
B-5	withdraw	=
B-6	modify_roi	=
B-7	display_generic	=
B-8	get_intezer_input	=

x_{111}

Meghana

	Keys	Values
m-1	bank_name	B-1
m-2	bank_roi	B-2
m-3	bank_branch	B-3
m-4	__init__	B-4
m-5	withdraw	B-5
m-6	modify_roi	B-6
m-7	display_generic	B-7
m-8	get_intezer_input	B-8
m-9	name	'MS' 'Meghana Shetty'
m-10	age	20
m-11	account	123456
m-12	Balance	987654

y_{111}

Srujana

	Keys	Values
S-1	bank_name	B-1
S-2	bank_roi	B-2
S-3	bank_branch	B-3
S-4	__init__	B-4
S-5	withdraw	B-5
S-6	modify_roi	B-6
S-7	display_generic	B-7
S-8	get_intezer_input	B-8
S-9	name	'SR' 'Srujana Reddy'
S-10	age	21
S-11	account	765432
S-12	Balance	345678

y_{222}

Program :

```
Class Bank :
    bank_name = 'SBI'
    bank_roi = 7
    bank_branch = 'KIZIKSTHIAN'
    def __init__ ( self , n , a , ac , bal ):
        self.name = n
        self.age = a
        self.account = ac
        self.balance = bal
    def withdraw (self) :
        amount = self . get_intezer_input ( )
        print (self)
        if self.balance > amount:
            self.balance -= amount
            print ('Withdraw is successful')
        Else :
            print ('Insufficient Balance')
    @classmethod
    def modify_roi (cls) :
        newroi = cls. get_intezer_input ( )
        print (cls)
        cls. bank_roi = newroi
        print ('ROI is Modified')
    @classmethod
    def display_generic (cls) :
        print(cls)
        print ('bank_name is ', cls. bank_name )
        print ('bank_roi is ', cls. bank_roi )
        print ('bank_branch is ', cls. bank_branch )
    @ staticmethod
    def get_intezer_input( ) :
        intezer = int (input ('Enter number'))
        return integer
meghana = Bank ('MS', 20, 12345, 987654)
Srujana = Bank ('SR', 21, 765432, 345678)
meghana . modify_roi ( )
print ( meghana . bank_roi) # 5
print (Srujana . bank_roi) # 5
print ( Bank . bank_roi ) # 5
meghana . withdraw ( )
# meghana . display_generic ( )
```

o/p :

Class Methods	Static Methods
• Class Methods belongs to class itself. • It have access to class Variables. • cls is a mandatory Argument for class method • Each and Every class Method be decorated with classmethod decorator i.e., @classmethod • We can access class methods with both object and class but it modifies only in class Syntax : Class ClassName @classmethod def methodname(cls, arg): pass • Access them using ClassName	Static Methods neither belongs to class nor objects • It doesn't have access to class and object Variables • There is no mandatory argument for static methods • Each and Every Static method be decorated with static method decorator i.e, @staticmethod. • We can't access (or) modify any of the class (or) object state. Syntax : Class ClassName : @staticMethod def method( ): content • Can be called (or) accessed by using className.

* Code Reusable Technique :

Ways of Accessing the properties and Behaviors of one class into Another class

Based on their relationship b/w Entities we can access properties / behaviors in 2 ways

1. Has a relationship (Aggregation)
2. is a relationship (Inheritance)

Has a Relationship (Aggregation) :

① Aggregation Represents has-a relationship of Real time Entity.

② Aggregation can be achieved by pointing to reference of object (or) creating the object of class inside the other class

③ Examples of has-a relationship are

→ Student has an Address
Object of Address class should be created inside Student class

→ Driver has a Car
Object of Car should be created inside Driver class

Team has a player

Object of player class should be Created in Team class

Car has an Engine

Object of Engine Class should be Created in Car class.

Demonstration of Has-a Relation by using Student Address. / Created object outside and passing as a Value

```
Class Student( ):
    def __init__(self, n, a, cl, addr):
        self.sname = n
        self.sage = a
        self.sclass = cl
        self.addressobject = addr
    def student_details(self):
        print(f'student name is {self.sname}')
        print(f'age of student is {self.sage}')
        print(f'sclass of student is {self.sclass}')
        print(f'address of student is {self.addressobject}')
#       self.addressObject = KIZIKISTAN
        self.addressobject.display_address( )
```

```
Class Address( ):
    def __init__(self, c, d, s, co):
        self.city = c
        self.district = d
        self.state = s
        self.country = co
    def display_address(self):
        print(f'city name {self.city}')
        print(f'district {self.district}')
        print(f'state name {self.state}')
        print(f'country name {self.country}')

KIZIKISTAN = Address('KIZIKISTAN', 'izikistan', 'zakistan', 'premistan')
dmm = Address('Bharmavaram', 'ATP', 'AP', 'India')
koti = Student('KOTI', 22, 'python', KIZIKISTAN)
koti.student_details( )
faroog = Student('faroog', 22, 'python', dmm)
faroog.student_details( )
```

Student

	Keys	Values
S-1	__init__	=
S-2	Student_details	=

X111

Address

	Keys	Values
A-1	__init__	=
A-2	display_address	=

X222

KIZIKISTAN

	KEYS	VALUES
K-1	__init__	A-1
K-2	display_address	A-2
K-3	City	KIZIKISTHIAN
K-4	dishict	IRIKISTAN
K-5	State	ZAKISTAN
K-6	Country	PREMISTAN

Y111

Koti

Keys	Values
__init__	S-I
Student_details	S-2
Sname	Koti
Sage	22
Sclass	python
address object	Y111

Y222

dmm

Keys	Values
__init__	A-1
display_address	A-2
city	'Bharmavaram'
Dishict	'ATP'
State	'AP'
Country	'INDIA'

Y333

farooq

Keys	Values
__init__	S-1
Student_details	S-2
Sname	farooq
Sage	22
Sclass	python
address object	Y333

Y444

→ Case II : Creating object inside the constructor of Current class

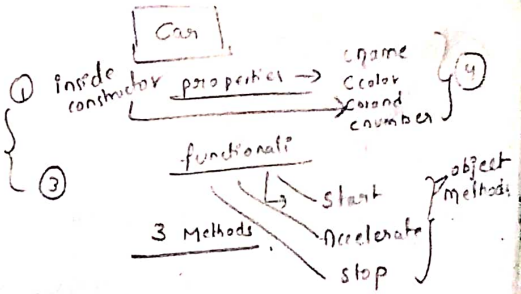
Address class (defining)

```
Class Student( ) :

    def __init__(self, n, a, cl) :
        Self.sname = n
        self.sage = a
        self.sclass = cl
        c = input('Enter city')
        d = input('Enter district')
        s = input('Enter state')
        co = input('Enter country')
        Address class object Aco = Address(c, d, s, co)
        Self.addressObject = Aco

    def student_details(self) :
        print(f'name of student is {self.sname}')
        print(f'age of student is {self.sage}')
        print(f'class of student is {self.sclass}')
        self.addressObject.display_address( )

koti = Student('koti', 22, 'PYTHON')
koti.student_details( )
farooq = student('farooq', 22, 'PYTHON')
farooq.student_details( )
```

Defining Address

```
Class Trainer( ):
    def __init__(self, n, a, cO):
        self.Tname = n
        self.Tage = a
        self.Tcourse = 60
        c = input('Enter city')
        d = input('Enter district')
        s = input('Enter state')
        coun = input('Enter country')
        ACO = Address(c, d, s, coun)
        # self.addressobject = ACO

    def Trainer_details(self):
        print(f'name of Trainer is {self.Tname}')
        print(f'age of Trainer is {self.Tage}')
        print(f'course of Trainer is {self.Tcourse}')
        self.addressObject.display_address( )

Harshad = Trainer('Harshad', 27, 'PYTHON')
Harshad.Trainer_details( )
Gireesma = Trainer('Gireeshma', 26, 'sql')
Gireesma.Trainer_details( )
```

```
Class Car :
    def __init__(self, n, c, b, num):
        self.cname = n
        self.Ccolor = c
        self.Cbrand = b
        self.Cnumber = num

    def Starte(self):
        print(self.cname, 'is started')

    def accelerate(self):
        print(self.cname, 'is accelerated')

    def stop(self):
        print(self.cname, 'is stopped by using brakes')
```



Driver
- Dname
- Dage
- Dexper

Dcar = object of carclass (use of car object)

functions :

Driving (use the functionalities of car)

```
Class Driver
```

```
Class Car :
    def __init__ (self, na, c, nu, b) :
        Self.cname = na
        Self.ccolor = c
        Self.cnumber = nu
        Self.cbrand = b
    def start (self) :
        print ( self.cname, ' is Started' )
    def accelerate (self) :
        print (self.cname, ' is accelerate')
    def stop (self ) :
        print (self.cname, 'is stopped by using brakes')
Class Driver ( ) :
    def __init__ (self, n, a, e) :
        self.Dname = n
        self.Dage = a
        self.Dexperience = e
    # Taking Inputs to create car new object
    na = input (' Enter Car name')
    c = input (' Enter Car color')
    nu = input (' Enter Car number')
    b = input (' Enter Car brand')
    # Creating Car Object
    Carobject = Car (na, c, nu, b)
    # Car object is assigned as a value to Dcar
      ( Specific Property of Driver)
    def driving (self) :
        Print (self.Dname, ' has Entered the Car')
        self.Dcar.start( )
        self.Dcar.accelerate( )
        self.Dcar.stop ( )
        Print (self.Dname, ' has come out of the Car')
Santhosh = Driver ('Santhosh', 22, 3)
Santhosh.driving( )
```

o/p :

Car

	Keys	Values
C1	--init--	
C2	start	
C3	accelerate	
C4	stop	

x_{111}

Driver

	Keys	Values
D-1	--init--	
D-2	driving	

y_{111}

Car Object

	Keys	Values
A-1	--init--	C-1
A-2	start	C-2
A-3	accelerate	C-3
A-4	stop	C-4
A-5	Cname	'Range Rover'
A-6	Ccolor	'Black'
A-7	Cnumber	9999
A-8	brand	'Velar'

y_{222}

Santhosh

Keys	Values
--init--	D-1
driving	D-2
Dname	'Santhosh'
Dage	22
DExperience	3
Dcar	y_{222}