PYTHON

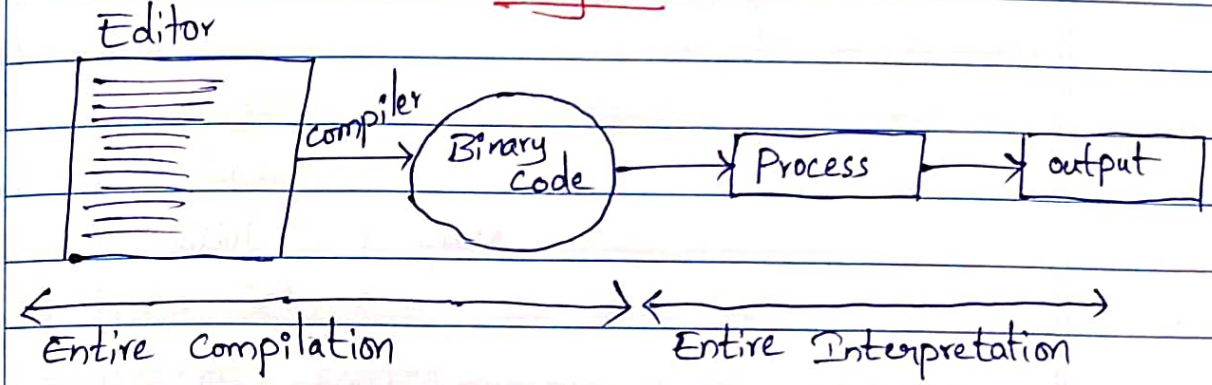
Date: / /

Programming Language :-

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

Ex :- C, C++, java .....

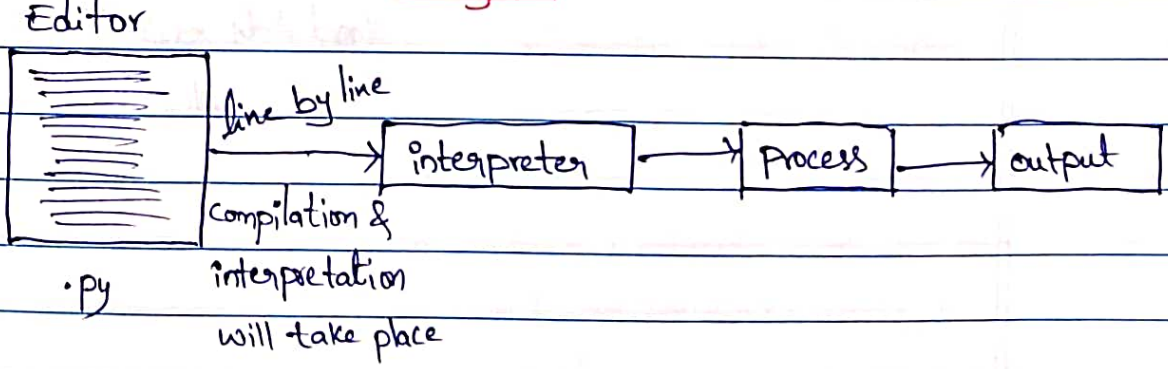
Diagram :-



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.

Diagram



Features of Python:-

Date: / /

1. Python is high-level open source interpreted scripting language.
2. Python is a Dynamic typed language.
3. Python is very easy to understand & learn.
4. Python has more built in libraries.
5. Python case-Sensitive language.
6. Python supports functional programming and oops concepts.
7. Python is a community language.

Fields to use python:-

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

Editors:-

* Pycharm
* VS code
* Sublime text
* atom
* Jupiter Note book
* edit plus

Date: / /

IDLE :-

1. IDLE stands for Integrated Development & Learning Environment.
2. IDLE is a environment which is integrated with all the python default implementations that are required for development and learning.

⇒ We can use idle in 2 ways :-

1. Python Shell :-

It is a interactive console where we can execute only one python statement at a time.

2. Python Module :-

1. It is a file with .py extension
2. In python module we can write and save multiple python statements.

Process to create python module from shell :-

open python shell
press ctrl + n
while saving just provide the name of module.

Command for executing python module :-

Function key + F5

Syntax for executing python module from command prompt :-

python modulename.py

(or)

py modulename.py

Date / /

Note:-

While creating and saving the python module from external editors it is mandatory to provide .py extension along with name.

Tokens:-

Tokens are essential elements for writing a python program. In python we have 4 tokens:-

1. keywords
2. Variables
3. Identifiers
4. Data types

=> keywords:-

keywords are built are reserved words which are used for performing specific task.

=> In python we have 35 keywords other than True, False, None remaining keywords are defined in lowercase.

syntax:-

```
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', 'rais' 'return', 'try', 'while', 'with', 'yeild']

```
len(keyword.kwlist)
35
```

Date: / /

Import :-

⇒ import keyword is used for performing importing process.

⇒ Importing is a process of accessing the content of another module in to current module.

Variables :-

⇒ Variable is a name given to a memory location.

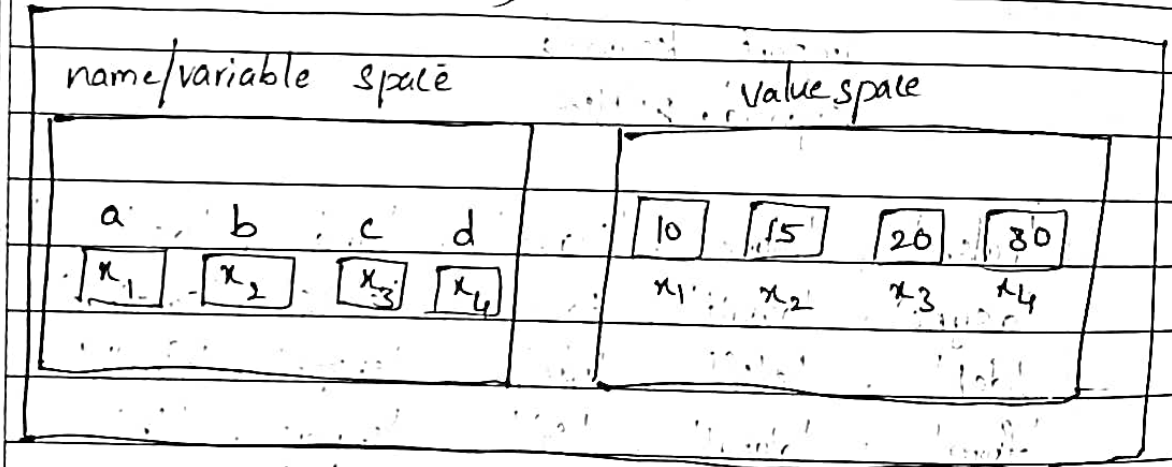
⇒ Value stored in variable might or might not changed.

Syntax for define single variable :-

Varname = value (execution done from right to left direction)

⇒ Once we create a variable a name variable & value spaces and created in memory.

Memory :-



⇒ a = 10, b = 15, c = 20, d = 30

Date / /

Note :-

⇒ If we assign same integer value to multiple variables then they will point to same memory Address.

⇒ If we assign a new value for existing variable, it will point to new memory.

id()

⇒ It is used for to know the address of the particular variable.

Ex :- id (a) → o/p → 1980456572432

Garbage Collection :-

⇒ It is the process of deleting unused memory.

⇒ Drawbacks which are caused when developers are performing Garbage collection.

⇒ Deleting memory before completion of it's usage.

⇒ Not deleting even after it's usage.

⇒ In python garbage collection is performed by PMM (Python Memory Management).

PMM (Python Memory Management)

⇒ PMM stands for python Memory Management.

⇒ PMM will delete the value when it has zero references.

Deleting multiple variables :-

Syntax: var1, var2, var3 ..... varN = value1, value2, value3 ..... valueN

Ex :- a, b, c = 10, 20, 30

Date: / /

Ex:- Assigning other variable value to new variable

Syntax:-

[new var = existing var]

```
z = j
```

error is occured because j is not defined

```
z = a
z
10
```

'a' is already assigned.

Swapping of a Variables:-

Syntax:-

```
var1, var2 = var2, var1
```

Ex:-

```
a = 100
b = 200
a, b = b, a
a, b = 200, 100
```

Datatypes

Date / /

Based on no. of values stored in variable

- Single value Datatype
  - Singlevalu Number Datatype
    - Integer
    - Float
    - Complex
  - Boolean Datatype
    - True
    - False
- Multivalued / collection Datatype
  - String
  - List
  - Tuple
  - set
  - Dictionary

Based on behaviour of value stored in variable

- Immutable Data type
  - All single value DT
  - String
  - Tuple
- Mutable Data type
  - List
  - Set
  - Dictionary

Note:- Data types are used for defining the type of data stored in a variable.

Single value Data type:-

We can store only one single data in variable.

Integer:-

The whole numbers without Decimal point Ex:- 81,22

Float:-

The whole number with Decimal point Ex:- 91.35

complex:-

The complex numbers are the combination of real part and imaginary part.

Ex:- $A + Bj \longrightarrow \sqrt{-1}$ (constant)

A: Real value, Bj: imaginary

Date: / /

Boolean Data Type:- (True, False)

These datatype are used for defining the yes or No type of data. True keyword is used for defining true boolean type. False keyword is used for defining false boolean type.

type() function:-

It is used for returning the type of data stored in a variable.

Syntax:-

```
type(variable)
```

Note points:-

1. Everything in python is an object.
2. We can store only one value or one memory address at a time in a variable.

Multi valued or Collection Data Type:-

⇒ In this category we will be storing more than one data item or value.

⇒ In order to bound individual elements into one collection we have to use boundaries.

1. If we store multiple values inside variable memory will be divided into sub memory address.
2. We can identify sub memory addresses by using index positions.

Index Positions

Date: / /

In python we have both positive index positions and Negative index positions.

positive index positions:-

Direction ⟶ left to Right (⟶)

Range ⟶ 0 to n-1 [where n = len(collection)]

Negative index Positions:-

Direction ⟶ Right to left (⟵)

Range ⟶ -1 to -n [where n = len(collection)]

Classification of Collection Data Type:- (CDT)

CDT are classified in to 2 types:-

1. Ordered CDT
2. Unordered CDT / Random ordered CDT

1. Ordered CDT:-

In Ordered CDT the in memory data will be stored in the same order as that we have defined in variable.

Ex:- String
Tuple
List
Dictionary

2. UnOrdered CDT:-

In unordered CDT in memory data will be stored in Random order.

Ex:- Set

Date: / /

String.

1. String is a collection of individual elements which are enclosed in a pair of Quetation marks.
2. String is immutable data type.
3. String is ordered CDT.
4. In String indexing and Slicing is possible.

⇒ For representing single line strings we use single or Double Quotes. ('___', "___")

⇒ For representing multiple line string we use triple Quotes. ("""___""")

Syntax for declaring the single line String data type:-

variablename = 'element1element2 ---- elementn'

(or)

variablename = "element1 element2 ---- elementn"

Syntax for declaring the multiple lines string data type:-

variablename = """ element1 element2
element3 ----

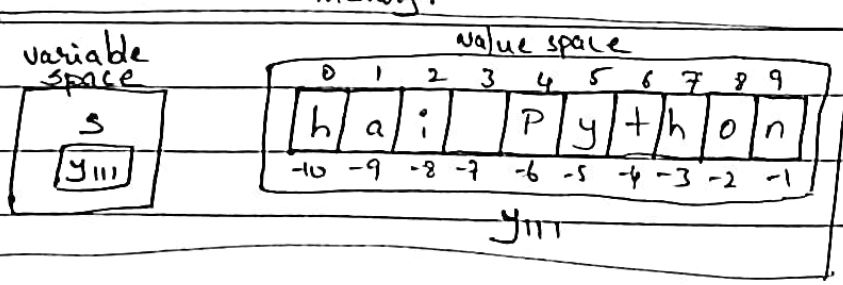
---elementn"""

Empty string declaration

str() or ' '

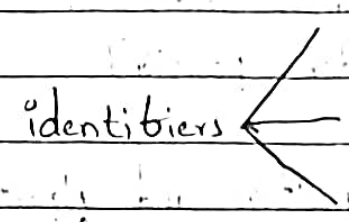
Ex: s = 'hai python'

memory.



Identifiers

Identifiers is a name with which we can identify the variables, functions & classes.



Rules of identifiers :-

1. We should not use keyword as identifiers.
2. As a first character of identifiers we should not use as a number.
3. Other than underscore (_) we cannot use any special character in identifiers because each and every other special characters has some functionality in python.
4. We can use combination of alphabets, numbers and underscore.

Standards of identifiers :-

Variables → use only lowercase

Functions → one word → lowercase

two words → Firstword in lowercase & remaining words in title case.

Classes → titlecase

Constants → Uppercase.

Date: / /

Indexing:-

⇒ It is a process of extracting individual elements from a given collection.

Syntax for performing indexing:-

Variablename [index - position]

We can give both positive and Negative index positions, But internally Negative indexes are converted into positive indexes only.

Ex:- s = 'hai python'

s[5] → 'y'

s[8] → 'o'

s[-4] → 't'

s[-7] → ' '

Slicing:-

⇒ It is a process of extracting multiple characters from a given collection.

⇒ We can perform slicing in 2 ways:-

1. Positive Slicing
2. Negative slicing

→ **Positive Slicing:-**

It is the process of extracting multiple elements in the left to Right direction.

Syntax for performing positive slicing

VariableName [startindex : Endindex+1 : updation

Date: / /

In positive slicing default values for:-

Starting index = 0

ending index = len(collection) + 1

updation = +1

⇒ Once i perform positive slicing interpreter will check for below mentioned condition.

start index position < end index position

Ex:- s = 'hai python'

s[4:10:1]

4 < 10 = true

4+1 = 5 < 10 = true

5+1 = 6 < 10 = true

6+1 = 7 < 10 = true

7+1 = 8 < 10 = true

8+1 = 9 < 10 = true

9+1 = 10 < 10 = false

output :- 'python'

Examples:-

	o/p
s[4:8:1]	→ 'pyth'
s[4:9:2]	→ 'pto'
s[0:9:2]	→ 'hipto'
s[0:7:3]	→ 'ht'
s[::4]	→ 'hpo'
s[4::]	→ 'python'
s[::]	→ 'hai python'

Date: / /

→ Negative slicing:-

It is the process of extracting multiple elements in the Right to left direction.

Syntax for performing Negative slicing is:-

VariableName [startindex : endindex-1 : updation]

In negative slicing default values are:-

Startindex = -1

Endinder = -(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 when we use negative index positions in slicing.

⇒ -

⇒ In case of Negative slicing interpreter will check for below condition

Startindexposition > Endinder position.

Ex:- S = 'hai python'

S[-1:-5:-1]

-1 > -5 = true

-1-1 = -2 > -5 = true

-2-1 = -3 > -5 = true

-3-1 = -4 > -5 = true

-4-1 = -5 > -5 = false

output:- 'nobt'

~~5-1 =~~ Reversing a string:-

S[::-1] → 'nohtyp iah'

Date: / /

Ex:-

s[-2:-7:-2] → 'otp'

s[-2-7:] → ' '

s[-3::] → 'hon'

s[6::-1] → 'typ iah'

Note:- 1. In case of indexing

⇒ If the specified index position is more than the length then it will be throws an error.

Ex:- s[-11] → Error

2. In case of slicing

⇒ If the specified end index position is more than the length then it will till the length of CDT.

Ex:- s[1:127:1] → 'ai python'

List:-

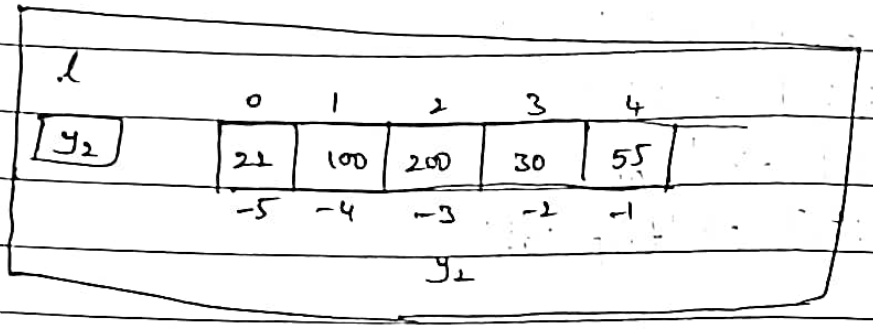
1. List is a collection of both Homogenous and Heterogenous data in which each and every element is seperated by (,) Comma operator and enclosed in the pair of [] square brackets.
2. List is a mutable data type.
3. List is a ordered CDT.
4. In list slicing & indexing is possible.

Syntax for declaring list data type:-

Variablename = [element1, element2, ... elementN]

Date: / /

l = [22, 100, 200, 30, 55]



Ex:-

L[-3] → 200
L[1] → 100
L[3] → 30
L[1:4:] → [100, 200, 30]
L[1:4:2] → [100, 30]
L[::-1] → [55, 30, 200, 100, 22]
L[-2:-5:-2] → [30, 100]

Behaviour of CDT:-

Immutable Data type:-

In case of immutable data type we cannot modify its value space.

Mutable Data type:-

In case of mutable we can modify its value space.

Syntax for modifying the value space by indexing:-

VariableName [index_position] = NewValue

Date / /

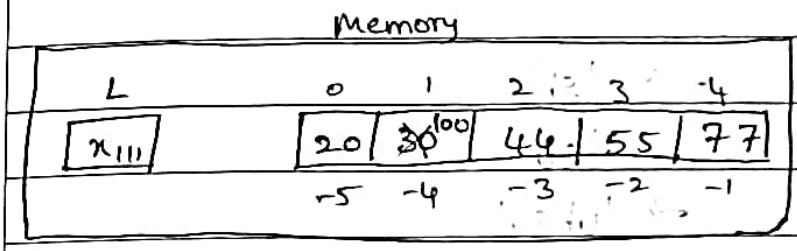
Syntax for deleting the value space by indexing:-

del variablename [index-position]

Note:-

String is immutable data type so we cannot modify the value space.

L = [20, 30, 44, 55, 77]



Ex:-

L[1] → 30

L[1] = 100

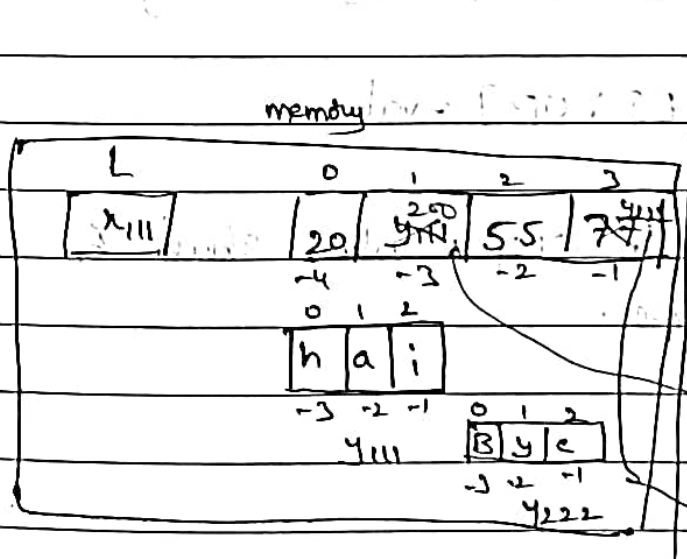
L → [20, 100, 44, 55, 77]

Ex:-

S = 'harshad'

S[2] = k → error (because it is immutable data type)

L = [20, 'hai', 55, 77]



L[2] → 55

L[1] → 'hai'

L[1][1] → 'a'

L[1][::2] → 'hi'

L[1][::-2] → 'ih'

L[1][1] = 'i' → error (strings are immutable)

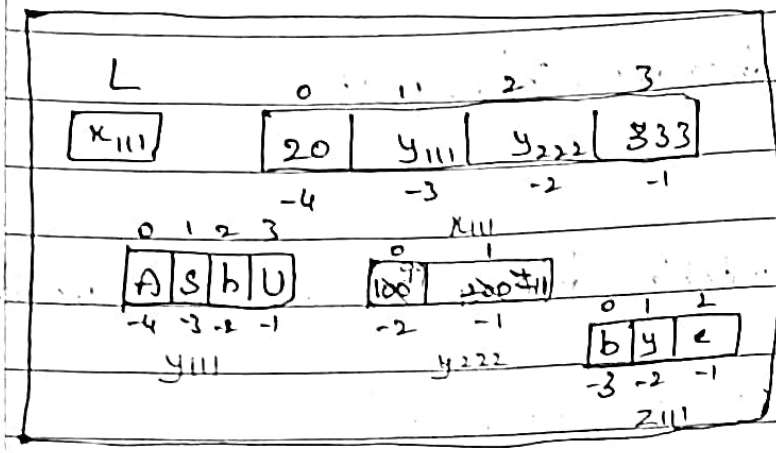
L[1] = 200

L[3] = 'bye'

L → [20, 200, 55, 'bye']

Date: /

L = [20, 'Ashu', [100, 200], 333]



L[1][1:3:] → 'Sh'

L[1][1::2] → 'SU'

L[1][::-1] → 'uhsa'

L[2][1] → 200

L[2][0] = 99

L → [20, 'Ashu', [99, 200], 333]

L[2][1] = 'bye'

L → [20, 'Ashu', [99, 'bye'], 333

L[2][1][0:2] → 'by'

⇒ Modification of list by using slicing:-

Syntax:-

Var[SI : EI : UP] = value

Note:- where value is in CDT, value should be in collection data type.

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

L[1:3:1] = 100 → error

L[1:3:1] = 'hai'

L → [11, 'h', 'a', 'i', 33, 44, 55]

L[1:4:1] = [100, 200]

L → [11, 100, 200, 44, 55]

L[1:4:1] = [333]

L → [11, 333, 55]

L[1::] = [22, 33, 44, 55]

L → [11, 22, 33, 44, 55]

L[1:4:] = ['hai', 'bye', 'hello']

L → [11, 'hai', 'bye', 'hello', 55]

L[2:4:] = [[111, 222], 'harshad']

L → [11, 'hai', [111, 222], 'harshad', 55]

L[2][0::] → [11, 22, [333, 444], 44, 55]

L[2][::] = → [333, 444] → Changing behaviour with same location of memory.

L[2:3:] = → [333, 444] → changing the location of memory.

Date: / /

Tuple :-

1. Tuple is a collection of both homogeneous and Heterogeneous data in which easy and every element is seperated by comma operator (,) and enclosed in the pair of paranthesis ()
2. Tuple is a immutable CDT.
3. Tuple is a ordered CDT.
4. Comma operator defines the tuple not the paranthesis.
5. Both indexing & Slicing can be performed in tuple.

→ Syntax for defining multiple values in tuple :-

variableName = (element1, element2, ... elementN)

(or)

VariableName = element1, element2, ... elementN

→ Syntax for defining single value in tuple :-

variableName = (element,)

(or)

VariableName = element,

→ Syntax for declaring the empty tuple data type :-

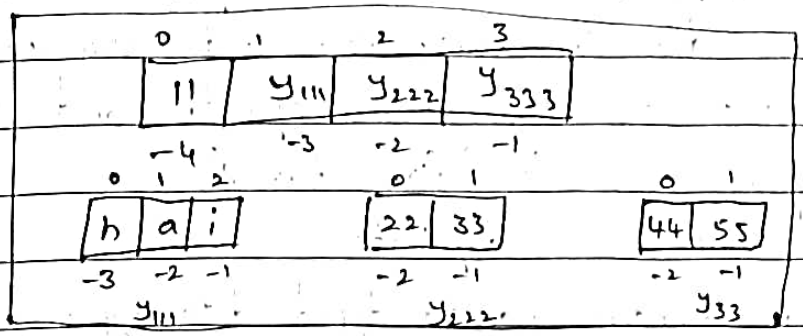
VariableName = ()

(or)

Variablename = tuple()

Date: / /

Ex:- T (11, 'hai', [22,33], (44,55))



T[2] → [22, 33]

T[2][1] → 33

T[0] → 11

T[0] = 22 → Error (It is immutable DT)

T[3][0] = 100 → Error (It is immutable DT)

T[2][0] = 44 → It is mutable

T[2][1] = 66

T → (11, 'hai', [44, 66], (44, 55))

T[2] = 7654 → error.

output formats

	Indexing	Slicing
String	String	String
List	cannot identified	List
tuple	Cannot identified	Tuple

String, List, Tuple } Ordered
indexing & slicing
Duplicates
Any type of data.

Date: / /

Set

1. Set is a collection of both homogeneous and heterogeneous data in which each and every element is seperated by (,) comma operator and enclosed in the pair of {} (flower braces).
2. Set does not allow duplicates.
3. Set is an random ordered typed data type.
4. An set is an random-ordered we cannot perform indexing & slicing.
5. Set is a mutable data type.

→ Syntax for declaring set :-

VariableName = {element1, element2, ... elementN}

→ Syntax for declaring the empty set data type :-

VariableName = set()

Note :- We cannot have mutable data types inside the set.

Ex :- S = {90, 78, 67, 56, 89, 67, 'hai', 77}

S → {67, 56, 89, 90, 'hai', 77, 78}

S = {90, 78, (90), 78, 56, 45}

S → {56, 90, 45, 78}

S = {56, (90,), 90, 45, 78}, 45, 78}

S → {50, (90,), 90, 45, 78}

S = {90, 78, (9990), 78, 56, 45}

S → {9990, 56, 90, 45, 78}

S = {90, 78, [9990], 78, 56, 45}

S → error (because list is mutable DT)

Date / /

Dictionary

1. Dictionary is a collection of keys and value pairs.
2. In dictionary keys & values are enclosed in pair of {}.
3. Each and every key, value pair are seperated by using (,) comma operator.
4. Each and every key and values are seperated by using (:) colon operator.
5. Dictionary is mutable data type.
6. As the data is in the form of pairs we cannot perform indexing & slicing.

→ Syntax for declaring the dictionary :-

VariableName = {'keyName' : 'value1', 'keyName' : 'value2', ...
..... , 'keyName' : 'valueN'};

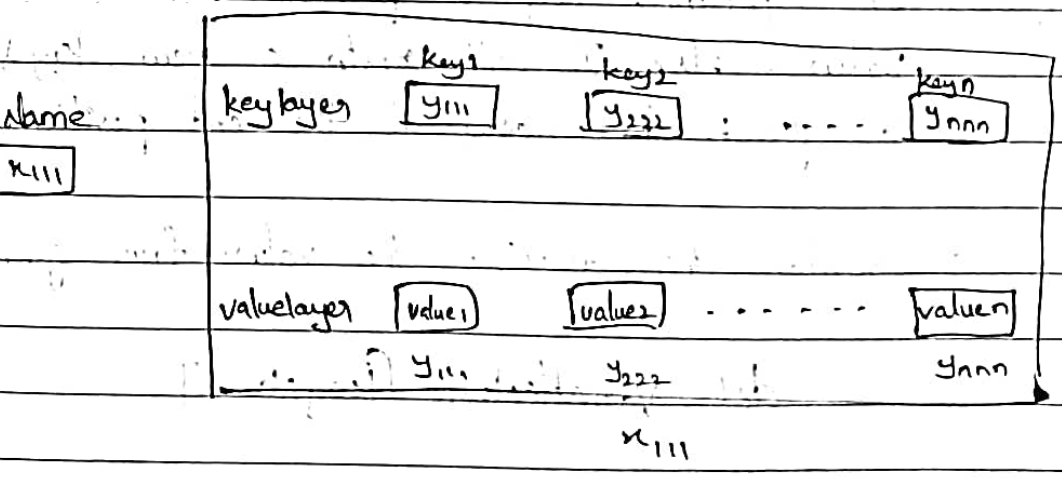
→ Syntax for declaring the empty dictionary data type :-

VariableName = dict()

→ Syntax for accessing the values from a give dictionary :-

VariableName ['keyName']

Memory



Date: / /

→ Properties of keys in dictionary:-

* 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.
* keys are case sensitive.
* If we declare duplicate keys then the recent value will be assigned to that key.

→ Properties of values in dictionary:-

* We can use duplicate values as values in dictionary.
* We can use any type of data in values.

Ex:- d = {'name' : 'Ashu', 'age' : 2}

d['name'] → 'Ashu'

d['age'] → 2

d['name'][0] → 'A'

d['name'][0:3] → 'Ash'

→ Syntax for modifying the values from a given dictionary:-

VariableName[keyName] = New value.

Note:- If specified key is present then it will update the value, else it will create a new key & value pair in given dictionary at the end position.

→ Syntax for deleting the values from given dictionary:-

del VariableName[keyName]

Date / /

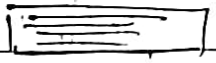
Ex:-

```
d['age'] = 3
d → {'name':'Ashu', 'age':3}
d['mobile'] = 8310692638
d → {'name':'Ashu', 'age':3, 'mobile':8310692638}
del d['mobile']
d → {'name':'Ashu', 'age':3
```

Ex:-

```
Details = {'name':'R.Jyothiprakash',
           'age' : 20,
           'mobile' : {8310692638, 9742826403},
           'achivement' : {'cricket' : 1,
                           'chess' : 2}
          }
```

→ Differences Between Functions & Methods

Functions	Methods
* Functions are individual block of code which used for performing a task.	Method is a function which is defined inside a class.
* Syntax:- def function(): 	Syntax:- class className: def methodname() 
* Functionname()	objectreference.methodName() (or) classreference.methodName()

Typecasting

Date: / /

It is the process of converting one type of data to another type of data.

Into which	From which	Syntax
Integer	float, boolean, string containing only integers.	int (value / variable name)
Float	integer, boolean, string containing int & float	float (value / variable name)
Complex	integer, float, boolean, string containing int, float & complex	complex (value / variable name)
Boolean	All type of data False → 0 and empty DT	bool (value / variable name)
String	all the type of data	str (value / variable name)
List	String, tuple, set, dictionary (consider only keys)	list (value / variable name)
Tuple	string, List, set, dictionary (consider only keys)	tuple (value / variable name)
Set	string, list, tuple, dictionary (only keys)	set (value / variable name)
Dictionary	list, tuple, set (in the form of pairs).	dict (value / variable)

isnumber } Boolean
isfunction }

Date / /

Built in methods of Strings:-

Note:- All the methods can perform some operations on strings but they will not modify Actual value of strings.

Case Conversion Methods of Strings

Method Name	Syntax of method	Functionality.
→ lower()	S.lower()	It is used for converting each and every character of given string in to lower case.
→ upper()	S.upper()	It is used for converting each and every character of given string in to upper case.
→ title()	S.title()	First character of each and every word of a given string will be converted in to uppercase and remaining characters in to lower case.
→ capitalize()	S.capitalize()	First character of entire string will be converted into uppercase and remaining characters in to lower case.
→ Swapcase()	S.Swapcase()	It is used for converting all lowercase characters to uppercase and uppercase characters to lowercase.

Ex:- S = 'Hai PYthon'

S.lower() → 'hai python'

S → 'Hai PYthon'

S.upper() → 'HAI PYTHON'

S.title() → 'Hai Python'

S.capitalize() → 'Hai python'

S.Swapcase() → 'hAI pyTHON'

Date: / /

S → 'hai harshad1234 vali@gmail.com'
S.title() → 'Hai Harshad1234 Vali@Gmail.Com'
S.capitalize() → 'Hai harshad1234 vali@.gmail.com'

ismethod
isfunction } → Boolean

MethodName	Syntax of method	Functionality
islower()	S.islower()	Returns true if string satisfies lower method else return false.
isupper()	S.isupper()	Returns true if string satisfies upper method else returns false.
istitle()	S.istitle()	Return true if string satisfies the title method else return false

Ex:- 'Hai Harshad'
~~S.idow:~~
S.islower() → False
S.isupper() → False
S.istitle() → True.

Ex- 'HAI PYTHON'
S.islower() → False
S.isupper() → True
S.istitle() → False

Ex:- 'hai space'
S.islower() → True
S.isupper() → False
S.istitle() → False.

Date: / /

Method Name	Syntax of method	Functionality
→ isalpha()	s.isalpha()	Returns true if string contains only alphabets else return false.
→ isdigit()	S.isdigit()	Returns true if string contains only digits else return false.
→ isalnum()	S.isalnum()	Returns True if string contains only alphabets and digits a combination of both else return False.
→ isspace()	S.isspace()	Returns True if string contains only space else return False.
→ strip(), rstrip(), lstrip()	S.strip() S.rstrip() S.lstrip()	These methods are used for removing the leading spaces from the given string

Ex:-

S = "R.Jyothi prakash"

S.isalpha() → False

S1 = "1234567"

S1.isdigit() → True

S2 = "Prakash629"

S2.isalnum() → True

S.isalnum() → True

S1.isalnum() → True

S3 = "Hai python"

S3.isspace() → False

Ex:- for strip(), rstrip(), lstrip()

S = 'hai python'

S.strip() → 'hai python'

S.lstrip() → 'hai python '

S.rstrip() → ' hai python'

Date / /

Object reference . Methodname (arguments) :-
Arguments are the essential parameters require for the function / methods for its execution.

Mandatory argument → directnames
Default argument → [names]

Split method

Split()	rsplit()
→ It is used for splitting the given string based on given Delimator.	It is used for splitting the given string based on given delimator.
→ Syntax split ([delimator], [count])	Syntax rsplit ([delimator], [count])
Default values for delimator = space for count = how many times ur delimator is repeated	Default values for delimator = space for count = how many times your delimator is repeated
→ Direction Left to Right	Direction Right to left
Ex = output format is list	output format of split is list

S = 'hai python'
S.split ('a') → ['h', 'i python']
S.split ('y') → ['hai p', 'thon']
S.split ('h', 1) → ['', 'ai python']
S1 = 'hai harshad'
S1. split('a', 2) → ['h', 'i h', 'rsh', 'd']
S. rsplit('h') → ['', 'ai pyt', 'on']
S. rsplit ('h', 1) → ['hai pyt', 'on']

split ([Delimator], [count])

↳ No. of times delimator is repeated.

Default values:-

Delimator → space

count → No. of times delimator is repeated

elements +1 → substring.

split() → It takes blank spaces.

Replace()

Method Name	Syntax of method	Functionality
replace()	replace (oldstring, newstring, [count]). Default value for count = no. of times your old string is repeated	It is used for replacing the existing characters with new characters.

Ex:-

S = 'hai python'

S.replace ('a', 'i') → 'hii python'

S.replace ('h', 'b') → 'bai python'

S.replace ('h', 'b', 1) → 'bai python'

S.replace ('h','b').replace ('p','k') → 'bai kython'

S.replace ('ha','bk') → 'bki python'

Count()

Method Name	Syntax of Method	Functionality
Count()	count (value, [start-index]; [end-index])	It is used for returning how many times given substring is represented in given string

Date: / /

Default values:-

Start index → 0

End index → len(CDT)

Ex:- S='hai python'

S.count('h') → 2

S.count('h',2) → 1

S.count('h',2,7) → 0

S.count('h',2,8) → 1

S.count('h',7) → 1

S.count('ha') → 1

S.count('ha python') → 0

Index() & Find()

index()	Find()
→ syntax	Syntax
index('value',[startindex],[endindex])	find('value',[startindex],[endindex])
→ It is used for returning the index position if value is present else it returns error as output	It is used for returning the index position if value is present else it returns -1 as output
→ Direction	Direction
Left to right	left to right

rindex() & rfind()

rindex()	rfind()
functionality is same as in index but the direction of execution is Right to left	Functionality is same as in find but the direction of execution is Right to left.

Date: / /

Ex:- S = 'hai python'

S.index('h') → 0
S.index('h',1) → 7
S.index('p') → 4
S.index('ho') → 7
S.index('w') → error
S.index('h',1,8) → 7
S.index('h',1,7) → error

} index()

S.rindex('h') → 7
S.rindex('ho') → error

} rindex()

S.find('h') → 0
S.find('h',1) → 1
S.find('w',1) → -1

} find()

S.rfind('w') → -1
S.rfind('h') → 7

} rfind()

Startswith()

It is 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.

Syntax:-

Startswith(substring, [startindex], [endindex])

Endswith()

It is used for checking whether the string is ending with specified substring or not, if it is ending, it will return True else it returns False.

Syntax:-

Endswith(substring, [startindex], [endindex])

Date: / /

Ex:- S = 'hai python'

S.Startswith ('h') → True
S.Starts with ('ha') → True
S. Starts with ('ha', 1) → False
S. Starts with ('ai', 1, 2) → False
S.Starts with ('ai', 1) → True
S. Starts with ('h', 6) → False

S. Ends with ('n') → True
S. ends with ('n', 4) → True
S.ends with ('n', 4, 9) → False
S ends with ('n', 8, 9) → False
S. ends with ('n', 9, 9) → False
S. ends with ('no') → False
S. ends with ('no', 9) → False
S. ends with ('on', 8) → True

format()

method Name	Syntax of method	Functionality
format()	'Content {} content {}'. format (value1, value2)	It is used for creating dynamic strings, we need to create placeholders are created by using {}

Ex = 'Hai, This is {} and my age is {}'.format ('Ashu', 2)
'Hai, This is Ashu and my age is 2'

Ex
a = 8 (8)
n = nikky
f 'Hai, this is {n} and my age is {a}'
'Hai, this is nikky and my age is 8'

Join()

Date / /

Methodsname	Syntax of method	Functionality
Join()	'glucharacter'.join (collectiondatatype)	It is used for creating new string by joining the elements of given collection DT with glucharacter.

Ex:-

'#'.join('hello') → 'h#e#l#l#o'

'@','@'.join('hello') → 'h@e@l@l@o'

'@'.join(['hello','hai',bye]) → 'hallo'@.hai.@bye'

' '.join(['hello','I','am','Ashu']) → 'hello I am Ashu'

' '.join(['hello','I','am','Ashu','age',';is',2]) → error

' '.join(['hello','I','am','Ashu','age',';is',2'])

'hello I am Ashu age is 2

'@'.join({'name':'Ashu','age':2})

'name @ age'

Date: / /

List Built-in Methods

MethodName	Syntax of method	Functionality
Count()	count(value)	It is used for counting how many times a given element is repeated.
index()	index(value[SI],[EI])	It is used for finding the index position of given element

Ex :- L = [11, 22, 33, 44, 55, 66, 11]

L.index(11) → 0
L.index(12) → error } index
L.index(11,1) → 6
L.count(11) → 2 } count
L.count(1192) → 0

Clear()

MethodName	Syntax of method	Functionality
Clear()	Clear()	It is used for deleting all the elements of given list but not the memory

Ex :- L = [22, 33, 55, 44, 66]

L.clear()

L → [ ] → empty list, it cleared all the elements from list.

Copy()

	Shallowcopy()	Deepcopy()
→	It perform shallow copy	It perform deep copy
→	It is a process of copying only the elements.	It is the process of copying both elements & memory
→	If we modify one variable other will not get affected/modified	If we modify one variable the other variable will be get modified/affected
→	Ex:- a = L.copy()	Ex:- b = L

Date / /

Ex: Shallow copy:

L = [11, 22, 33]

a = L.copy()

a → [11, 22, 33]

id(L) → 123456

id(a) → 562456

only elements will copy to another variable.

Ex: deep copy:

L = [11, 22, 33]

b = L

b → [11, 22, 33]

id(b) → 123456

id(L) → 123456

Here both elements & memory both will copy

Del()

It is used for deleting the elements which are present in the list.

Ex: @ L = [11, 22, 33, 44, 55, 66, 11]

a = L.copy() → Shallow copy

del a[1]

a → [11, 33, 44, 55, 66, 11]

L → [11, 22, 33, 44, 55, 66, 11]

b = L → Deep copy

del b[4]

b → [11, 22, 33, 44, 66, 11]

L → [11, 22, 33, 44, 66, 11]

Date: / /

Insertion methods of Lists

append()	extend()	insert()
→ append(value)	extend(value)	insert(indexposition, value)
→ values => SVDT/CDT	values => only CDT	It is used for adding the elements at specified index positions.
→ It is used for adding element into list at the end.	It is also used to add the elements into list at the end.	It will not replace the elements.
→ Entire CDT is considered as only one elements.	It will extract individual elements from CDT and add them at the end.	If specified IP > len then element will be add at the end.

Ex:-

L = [89, 67, 562]

L.append(12) ~~insert()~~ append()

L → [89, 67, 562, 12]

L.append(99)

L → [89, 67, 562, 12, 99]

L.append('hai')

L → [89, 67, 562, 12, 99, 'hai']

L.append([13, 55])

L → [89, 67, 562, 12, 99, 'hai', [13, 55]]

extend()

L.extend(4.5) → error (because it is Single value DT)

L.extend('bye') → [89, 67, 562, 12, 99, 'hai', [13, 55], 'b', 'y', 'e']

b = [99, 78, 67]

b.extend(['hai', 'hello'])

b → [99, 78, 67, 'hai', 'hello']

Date: / /

insert().

L.extend(45) → error (it is single value DT)

b. insert(1,1234)

b → [99, 1234, 78, 67, 'hai', 'hello']

b. insert (4, 900)

b → [99, 1234, 78, 67, 900, 'hai', 'hello']

b. insert (1111, 800)

b → [99, 1234, 78, 67, 900, 'hai', 'hello', 800]

b. insert (3, 'Ashu')

b → [99, 1234, 78, 'Ashu', 67, 900, 'hai', 'hello', 800]

[ ] → Default Arguments

( ) → Mandatory Arguments

Deletion Methods.

	pop()	remove()
→	pop([index position])	remove (value)
→	It is used to delete an element from list based on index position	It is used for deleting an element from list based on specified value.
→	By default pop method deletes last element from list.	If value is present for multiple times it will delete only first element.
→	If ip > len then it will throw an error.	If specified value is not present it will throws an error.
Ex	L = [11, 22, 'hai', 90, 78, 12]	L = [11, 22, 'hai', 90, 78, 12, 11, 99]
	L.pop(3) → 90	L.remove(11)
	L.pop(1) → 22	L → [22, 'hai', 90, 78, 12, 11, 99]
	L.pop() → 12	L.remove(90)
	L.pop(11) → error	L.remove(10) → error
	L.pop() → 78	L.remove() → error.

Date: / /

ASCII

ASCII → American Standard Codes Information Interchange

For finding the ASCII value we can use the function of ord() & char()

ex:- ord ('a') → 97
ord ('z') → 122
ord ('1') → 49
char (66) → 'B'
char (69) → 'E'

Sort()

→ It is used for arranging the elements of list either in ascending & descending order.

→ Sort () → ascending order

→ Sort (reverse = True) → Descending order

→ In case of CDT's if compares first element of CDT

→ In strings compares ASCII values to find out min & max

Ex:- L = [22, 67, 34, 12, 89, 64]

L.Sort()

L → [12, 22, 34, 64, 67, 89]

L.Sort(reverse = True)

L → [89, 67, 64, 34, 22, 12]

→ L = [22, 67, 34, 12, 89, 64, 'hai']

L.Sort() → error

→ L = [22, 33, '4445', 66]

L.Sort() → error

Date: / /

String

A = ['ashu', 'nikky', 'prakash', 'Ammu', 'harshad']

A.sort()

A → ['Ammu', 'ashu', 'harshad', 'nikky', 'prakash']

List

X = [[89,78,56], [12,88,67,44,33,222], [199,187], [45,888]]

X.sort()

X → [[12,88,67,44,33,222], [45,888], [89,78,56], [199,27]]

Function → Sorted()

Differences between sort() and sorted():-

	sort	Sorted
→	It is a method of list	It is a normal built-in function
→	sort method is used only with list data type.	Sorted can be used on any collection data type.
→	syntax for ascending order objectreference.sort()	syntax for ascending order Sorted(collection)
→	syntax for descending order objectreference.sort(reverse=True)	syntax for descending order Sorted(collection, reverse=True)
→	output format is list	output format is list
Note:-	It will modify the given values	It will not modify given CDT

Ex:- Sorted('hai') → ['a', 'h', 'i']

Sorted('hai', reverse=True) → ['i', 'h', 'a']

→ L = [90, 89, 67, 56, 5678]

Sorted(L) → [56, 67, 89, 90, 5678]

→ T = (99, 78, 56)

Sorted(T) → [56, 78, 99]

Sorted(T), reverse=True) → [99, 78, 56]

Date: / /

Built-in Methods of Tuple

→ count()

→ index()

Note:- Functionality of Tuple index() and count() methods are same as of list index() and count() methods.

Ex:-

T = (90, 23, 14, 89, 90)

T.count(90) → 2

T.index(90) → 0

T.index(90, 2) → 4

Built in methods of SET

→ copy()

It is used for performing the shallow copy.

→ clear()

It is used for clearing the elements of given set

Ex:- S = {89, 76, 45, 23, 99}

S1 = S.copy()

id(S) → 28045

id(S1) → 18025

S1.clear()

S1 → set()

→ Add()

It is used for adding the elements in to set randomly

Syntax:-

add(value)

Note:- List, set, dictionary should not be used as value.

Date: / /

Ex:-

S {99, 23, 89, 76, 45}

S.add(123456)

S→{123456, 99, 23, 89, 76, 45}

S.add('hi')

S → {123456, 99, 'hi', 23, 89, 76, 45}

S.add ([90, 78, 56]) → error

S.add (89)

S → {123456, 89, 99, 'hi', 23, 76, 45}

Deletion methods in Set

	POP()	remove()	Discard()
Syntax →	pop()	remove (value)	Discard (value)
→	It is used for deleting the first value from the given set.	It is used for deleting the elements based on given value	It is used for deleting the element based on given value.
→	It will not take any arguments	If value is present then remove() will delete that element else it will throw an error	If value is present then discard() will delete that element else it will not perform any operation.

Ex:- S = {89, 76, 45, 23, 99}

S → {99, 23, 89, 76, 45}

S.pop() → 99

S → {23, 89, 76, 45}

S.pop() → 23

S → {89, 76, 45}

S.remove (45)

S → {89, 76}

S.remove (45) → error

S.discard (89)

S → {76}

S.discard (89) → No operation

Date: / /

operations on Sets

	union()	intersection()	Dibference()
Syntax →	baseset.union(set1,	baseset.intersection(set1,	baseset.dibference
	set2, ... setn)	set2, ... setn)	(set1, set2 .... setn)
→	It is used for getting	It is used for getting	It is used for getting
	all the elements from	the only common elements	the un-common
	the specified sets	from the specified sets	elements from the base

Note :- By above operations any of the specified sets will not be modified

Ex:- S = {11, 22, 33, 44, 55}

S1 = {44, 60, 4, 22}

S2 = {11, 100, 222}

S3 = {22, 99, 80}

S.union(S1) → {33, 4, 11, 44, 22, 55, 60}

S → {33, 4, 22, 55, 11}

S1 → {44, 4, 60, 22}

S.Union (S1, S2) → {33, 4, 100, 11, 44, 22, 55, 60, 222}

S.intersection(S1) → {4, 22}

S.intersection(S1, S2) → set()

S.intersection(S1, S3) → {22}

S1.difference(S) → {44, 60}

S.difference(S1) → {33, 11, 55}

Date: / /

update()	Intersection_update()
Syntax → Baseset.update(set1, set2 ..... setn)	Baseset.intersection_update(set1, set2, ..., setn)
→ It will perform union operation and update the output into the baseset	It will perform intersection operation and update the output in to the Baseset

difference_update()	Symmetric_difference_update()
tax → Baseset.difference_update(set1, set2, ... setn)	Baseset.Symmetric_difference_Update() (set1)
→ It will perform difference operation and update the output into base set.	It will perform symmetric difference operation and update the output in to the baseset

Ex: S.update(S1)

S → {33, 4, 11, 44, 22, 55, 60}.

S1 → {44, 4, 60, 22}

S.intersection_update(S2)

S → {11}

S1.difference_update(S3)

S1 → {44, 4, 60}

S.update(S1, S2)

S → {4, 100, 11, 44, 222, 60}

S = {990, 67, 23, 9999}

S1 = {89, 23, 9999}

S.Symmetric_difference_update(S1)

S → {67, 89, 990}

Date: / /

issuperset()	issubset()	isdisjoint()
Returns True if the Baseset is the superset of specified set else it returns false.	Returns True if the Baseset is the subset of specified set else if returns False	Returns True if the specified sets has only un-common elements else it returns False.

Ex:-

S = {11, 22, 33, 44, 55}

S1 = {22, 33, 11}

S2 = {33, 4, 60}

S.issuperset(S1) → True

S.issuperset(S2) → False

S.issuperset(S1, S2) → Error.

S.issubset(S1) → False.

S.issubset(S) → True.

S2.issubset(S) → False

S.isdisjoint(S1) → False

S.isdisjoint(S2) → False

S3 = {99, 89}

S.isdisjoint(S3) → True.

S = {11, 22, 33, 4, 55}

S1 = {11, 4, 33, 55, 22}

S.issubset(S1) → True

S.issuperset(S1) → True

→ Interview Question.

Dictionary Built-in Methods

Date: / /

	Keys	Values	Items
Syntax →	keys()	values()	items()
→	It is used for getting the keys of specified Dictionary	It is used for getting the values of specified Dictionary	It is used for getting the both keys and values of specified Dictionary

Ex:-

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

→ d.keys()

dict_keys (['name', 'age'])

→ d.values()

dict_values (['Ashu', 2])

→ d.items()

dict_values ([('names', 'Ashu'), ('age', 2)])

	get()	setdefault()
Syntax →	get (keyname, [default value])	setdefault (keyname, [default value])
→	If specified key is present then get will display the value of specified key else get will display Value	If specified key is present then setdefault will display the value of specified key Else dsetdefault will update the dictionary with specified key and default value.

Ex:-

d → {'name' : 'Ashu', 'age' : 2}

d.get ('name') → 'Ashu'

d.get ('name' : Nikky) → 'Ashu'

d.get ('Name' : 'Nikky') → 'Nikky'

d → {'name' : 'Ashu', 'Age' : 2}

d.setdefault ('name') → 'Ashu'

d.setdefault ('name', 'nikky') → 'Ashu'

d.setdefault ('Name', 'nikky') → 'nikky'

Date: / /

d → {'name':'Ashu', 'age':2, 'Name':'nikky'}
d.setdefault('Name', 'Harshad') → 'nikky'
d.setdefault('mobile')
d → {'name':'Ashu', 'age':2, 'Name':'nikky', 'mobile': None}

Insertion Method :-

→ update()
It is used for updating the actual dictionary with multiple key and value pairs at a time.
Note :- It is used for merging dictionaries.
Syntax :-
BaseDictionary.update({'key':'value', 'key2':'value2', ...})

→ If specified key is present then it will update the value.
→ If key is not present then it will create a new key value pair
Ex :-
@ d {'name':'Ashu', 'age':2, 'Name':'Nikky', 'mobile': None}
→ d.update({'mobile':23456, 'gender':'male'})
→ d
{'name':'Ashu', 'age':2, 'Name':'Nikky', 'mobile':23456, 'gender':'Male'}
→ d.update({'mobile':876546, 'gender':'m'})
→ d
{'name':'Ashu', 'age':2, 'Name':'Nikky', 'mobile':876546, 'gender':'m'}

Date: / /

Deletion Methods in Dictionary

	pop()	popitem()
Syntax →	pop(key-name)	popitem()
→	It is used for deleting the key value pair based on specified key	It is used for deleting last key and value pair by default.
→	If key is present then it will delete else it will throw me an error.	output format of position popitem is a tuple.
→		If dictionary is empty it throws an error.

Ex:-

→ d.pop('gender') → 'm'

→ d

{'name': 'Ashu', 'age': 2, 'Name': 'nikky', 'mobile': 876544}

→ d.pop('gender') → error.

→ d.popitem() → ('mobile', 876544)

→ d.popitem() → ('Name', 'nikky')

→ d.popitem() → ('age', 2)

→ d.popitem() → ('name', 'Ashu')

→ d.popitem() → Error

popitem()

Operators

Date: / /

→ Operators are used for performing some operations on given operands.

Types of operators :- (floordivision.)

1. Arithmetic operator → (+, -, *, /, %, //, ** → degree)
2. Logical operator → (and, or, not)
3. Relational operator → (==, !=, <, <=, >, >=)
4. Bitwise operator → (&, |, ^ XOR, ~ NOT, <<, >>)
5. Assignment operator → (=, +=, -=, *=, /=, %=, //=, **=)
6. Membership operator → (in, not in)
7. Identity operator → (is, is not)

Arithmetic operator :-

Symbol	N to N	N to CDT	CDT to CDT
+	addition	Can't be performed	Concatinate (same CDT we can't perform set & Dictionary.
-	Substraction	Can't be performed	only between set to set (bitterence operation
*	multiplication	Contatinate by specified no. of times (not in set & Dictionary)	Cannot be performed.

/ → gives quotient in float.
// → give quotient after deleting decimal value.
% → gives remainder.

Ex :-

'hai' * 2 → 'haihai'
'hai' * 3 → 'haihaihai'
[2,3,4] * 2 → [2,3,4,2,3,4]
[0] * 4 → [0,0,0,0] → interview Question.

Date / /

5/2 → 2.5

5//2 → 2

5%2 → 1

**:-

a^b → a**b

2**3 → 2^3 → 8

3**3 → 3^3 → 127

5**3 → 5^3 → 125

Relational Operators

→ 1. Relational operators are used for returning a boolean value as an output.

→ 2. These Relational operators are used in conditional statements

→ >, >=, <, <=, !=

⇒ Number to Number Comparision we can do it easily.

⇒ But when we compare strings then comparision is done based on the ASCII values of the characters.

⇒ In case of CDT to CDT it will compare first element.

Ex:-

9==9 → False

9==19 → False

'hi'=='hai' → False

[90,89]==[90,89,0] → False

'hello'>'bye' → True

[90,78]>[9876] → False

Logical operators

Date: / /

⇒ 1. Logical operators are used for returning a boolean value as an output.

2. These logical operators are used in conditional statements

Truth Table of AND

operand1	operator	operand2	output	Ex:-
True	AND	True	True	1 and 5 → 5
True	AND	False	False	1 and 0 → 0
False	AND	True	False	0 and 5 → 0
False	AND	False	False	'' and [] → []

Truth table of OR

True or True	True	Ex:- 9 or 7 → 9
True or False	True	'hi' or 0 → 'hi'
False or True	True	0 or [89,67] → [89,67]
False or False	False	

Truth table of NOT

	Ex:-
True → False	not 8 → False
False → True	not 0 → True

Assignment operator:-

These are used for assigning some values to the variables by using = (equal) operator we can assign the values for a variable.

Ex:- a=10

(a=a+10)	a+=10 → 20	a//=2 = 5	(a=a//5)
(a=a-10)	a-=10 → 10	a%=2 = 1	(a=a%2)
(a=a*5)	a*=5 → 50	a**=5 = 1	(a=a**5)
(a=a/5)	a/=5 = 10.0		

Date / /

Membership operator

Syntax:

SVDT/CDT in CDT

Ex:

1 in '123' → Error

1 in 123 → Error

'1' in '123' → True

'hi' in [1,2,3,4] → False

[1] in [1,2,3,4] → False

[1] in [[1],2,3,4] → True

2 in {'name':'Ashu','age':2} → False.

2 not in {'name':'Ashu','age':2} → True

'hai' not in 'haai hello' → True.

→ It is used for checking whether the element is part of given collection or not.

→ In operator:-

It returns True if value is part of given collection else return false.

Syntax:- SVDT/CDT in CDT

→ Not in operator:-

It returns True if value is not a part of given collection else returns false.

Syntax:- SVDT/CDT not in CDT

Rules:-

1. Rightside must have CDT
2. If rightside have a String CDT then leftside also must & should have string CDT only.
3. In dictionary CDT they check only in keys.

Date: / /

Identity operator

→ These are operators are used for comparing the memory addr
→ is operator :-
→ It returns True if variables are pointing to same memory address else returns False.

Syntax :-

operand1 is operand2

→ is not operator :-
→ It returns True if variables are not pointing to same memory address else return False.

Syntax :-

operand1 is not operand2

→ Ex :-
→ S = 'hai'
S1 = 'hai'
S == S1 → True
→ L = [11, 22, 33]
L1 = [11, 22, 33]
L == L1 → True
→ L is L1 → False
→ T = (11, 22, 33)
T1 = (11, 22, 33)
T == T1
T is T1 → False
→ S = {90, 72, 67}
S1 = {90, 72, 67}
S == S1 → True
S is not S1 → True

Date: / /

Bitwise operator (It performs only on Numbers)

→ These are operators are used for performing operations on the "binary values" of given numbers.

→ Bitwise & → Returns "1" when both operands are "1" | 1 & 1 → 1

Ex:-

1 & 2 → 0	0001 & 0010 → 0000
2 & 3 → 2	0010 & 0011 → 0010

→ Bitwise | → Returns "0" when both operands are "0" | 0 | 0 → 1

Ex	1 | 2 → 3	0001 | 0010 → 0011
	2 | 3 → 3	00010 | 0011 → 0011

→ Bitwise ^ → returns "0" if both operands are same. | 0 ^ 0 → 0 | 1 ^ 0

Ex	1 ^ 2 → 3	0001 ^ 0010 → 0011
	2 ^ 3 → 1	0010 ^ 0011 → 0001

→ Bitwise ~ → It performs 2's compliment | ~n = -(n+1)

Ex- ~5 → -6

~-5 → 4

~16 → -17

→ Right shift >> → Shift binary to right side specified number of position

Syntax → operand >> no. of position

Ex:- 4 >> 1 → 2	0100 >> 0001 → 0010
4 >> 2 → 1	0100 >> 0010 → 0001

→ left shift << → Shifts binary to left side by specified no. of positions

Syntax: operand << no. of position

Ex:-

2 << 1 → 4	0010 << 0001 → 0100
9 << 1 → 18	1001 << 0001 → 10010

Date: / /

Flow Control Statements

→ Collecting data from the user during runtime :-

→ 1. input() function is responsible for collecting the data from the 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 spetil special function which is responsible for evaluating converting the data into its equivalent submitted data.

Ex :-

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

Ex :-

```
a = eval(input('Enter a value :')) # evaluating the submitted data
print(a)
print(type(a))
```

Commenting

→ 1. Comments are the non-executable statements.

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

→ we have 2 types of comments

1. Single line Comments (#)

2. multi line comments (''' ''' (triple Quotes)

Date / /

print()

print function is used for printing some content.

Syntax:-

print(val1, val2, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Note:- We can change the values of sep and end.

Ex:-

```
print(20)                  → 20
print(40, 60)              → 40,60
print(100,200,300)         → 100,200,300
```

Ex:-
```
print(20, end='@')
print(40, 60)
print(100, 200, 300)
```
→ 20@40 60
100 200 300

Ex:-
```
print(20, end='@')
print(40,60)
print(100,200,300, sep='$')
```
→ 20@40 60
100$200$300

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 or Decisional statements
2. Looping statements

Date: / /

Conditional or Decision Statements:-

In this type based on the one condition we will decide whether to execute set of instructions or not to execute set of instructions.

→ In Python we have 4 types of conditional statements

1. If condition
2. If-else condition
3. If-elif condition.
4. Nested if condition.

If-condition.

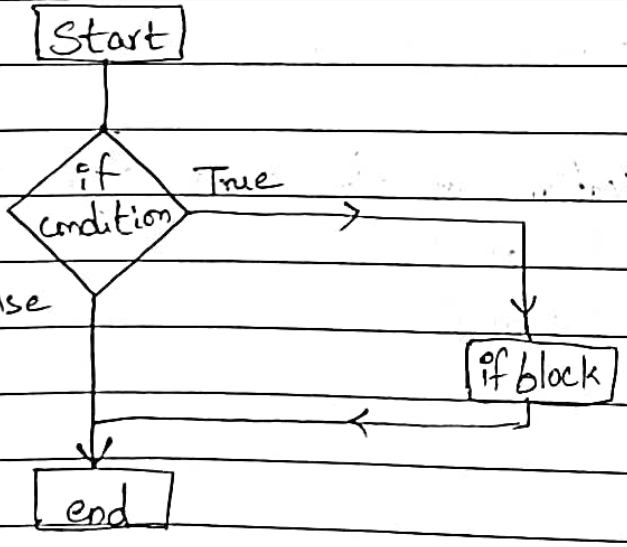
→ When we have only one condition to check then we will use if condition.

Syntax:-

```
if condition:
    statements
    of
    if block
```

(1 Tab → indentation)

Flow Diagram



Date / /

Ex:-

```
print ('start')
a = int (input ('enter a value'))
if  a > 10 :
    print ('i am in if block')
    print (a)
print ('end')
```

if - else condition

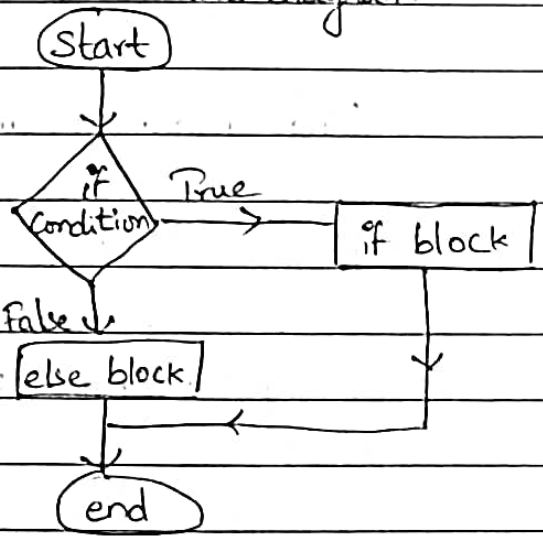
→ 1. When we have 2 conditions to check then we will use if - else condition.

2. Among 2 conditionets we will have
   a) Mandatory condition
   b) Default condition.

Syntax:-

```
if condition:
    statements
    of
    if-block
else:
    else
    block
```

Flow Diagram



Date: / /

Ex:-

```
print ('woke up get ready for proposing')
decision = input ('tell your decision')
if decision == 'yes':
    print ('Life is beautiful')
else:
    print('oi am alone, no one loves me i'll die')
print('go to meghana wine shop')
```

→ WAP to find given number is even or odd.

```
n = int (input('Enter Number:'))
if n %. 2 != 0:
    print('n is odd')
else:
    print('n is even')
```

→ Find maximum among given two number.

```
num1 = int(input ('Enter num1 value:'))
num2 = int(input ('Enter num2 value:')).
if num1 > num2:
    print('num1 is greater than num2'):
else:
    print ('num2 is greater than num1')
```

Date: / /

→ Find out given two strings are anagram strings or not.

```
str1 = input('Enter str1 value')
str2 = input('Enter str2 value')
if (sorted(str1) == sorted(str2)):
    print('Given str is anagram string')
else:
    print('Given str is not anagram string')
```

→ Find out given string is palindrome strings or not.

```
num = int(input('Enter a string:'))
l = str(num)
if num == num[::-1]:
    print('Given num is palindrome')
else:
    print('Given num is not palindrome')
```

if-elif Statements

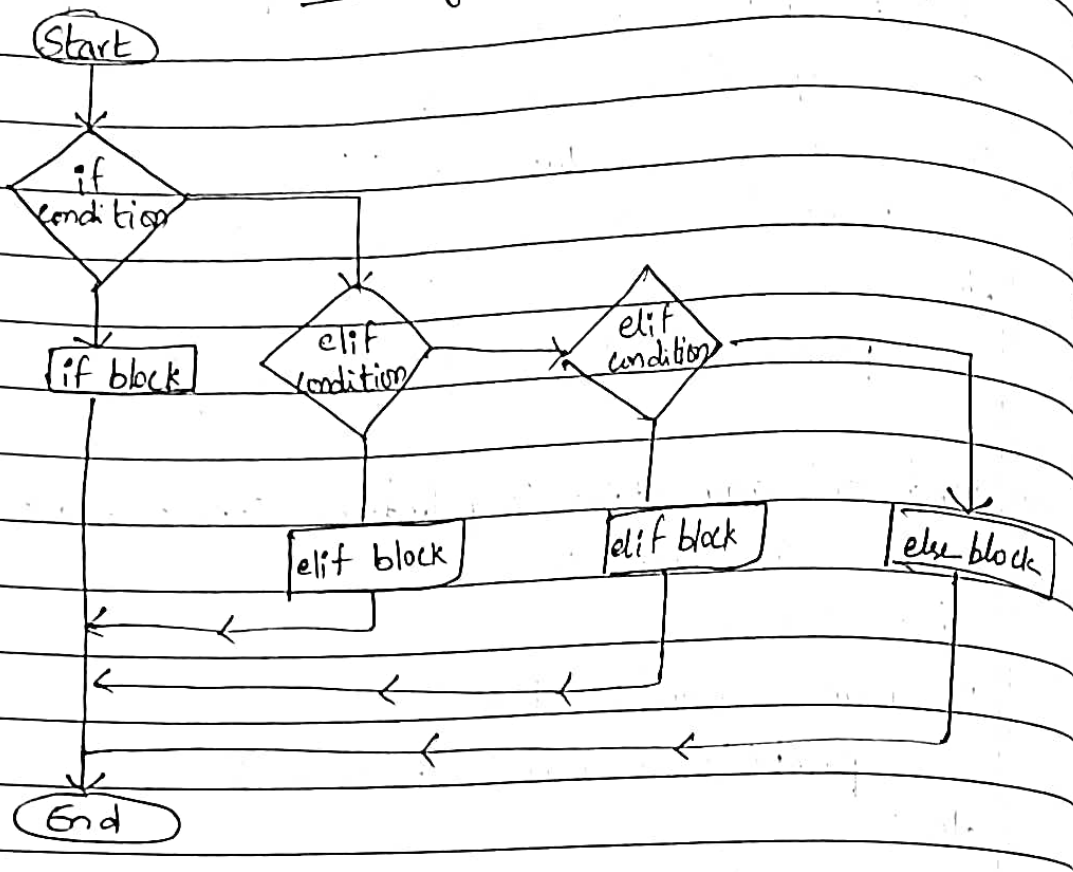
→ When we have more than 2 conditions to check then we will use if-elif condition.

Syntax:-

```
if condition:
    statements
elif condition:
    statements
:
elif condition:
    nth block
else:
    else statements
```

Date: / /

Flow Diagram :-



Ex:-

```
Choice = int(input('mam/sir please select an option'))
if choice ==1 :
    print('serve indian food & collect bill')
elif choice==2 :
    print('serve italian food & collect bill')
elif choice ==3 :
    print('serve chinese food & collect bill')
else :
    print('Dabaang')
```

Date: / /

→ Write a program to find maximum given three Numbers.

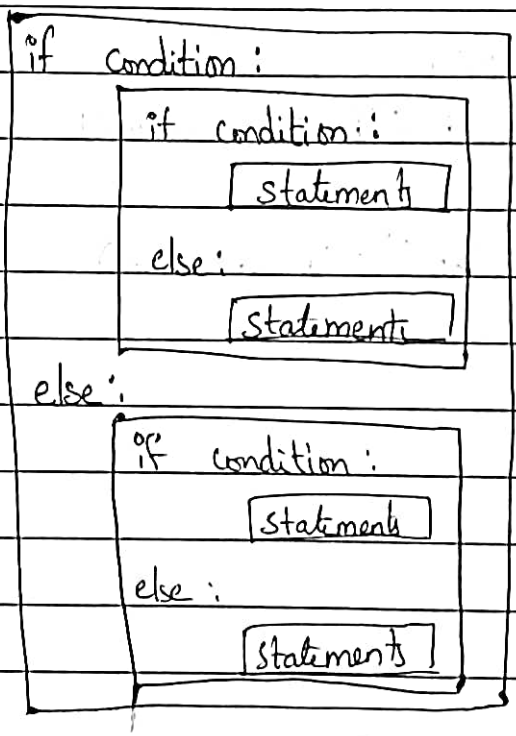
```
n1 = int (input ('Enter n1 value:'))
n2 = int (input ('Enter n2 value:'))
n3 = int (input ('Enter n3 value:'))
if  n1 > n2  and  n1 > n3 :
    print ('n1 is maximum')
elif  n2 > n3 :
    print ('n2 is maximum')
else :
    print ('n3 is maximum')
```

Nested if Statement

→ 1. It is the process of creating the condition inside another condition.

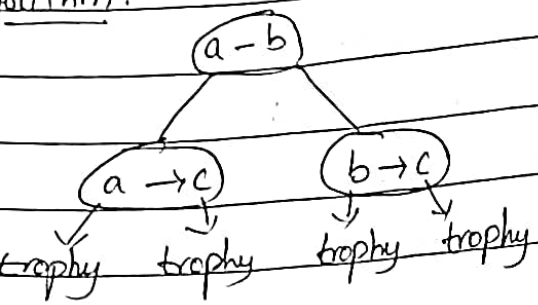
2. When we have more than 2 conditions to check then we will use nested if condition.

Syntax :-



→ Biggest of 3 numbers by using nested if

algorithm:



→ Source code:

```
a = int (input ('Enter a value'))
b = int (input ('Enter b value'))
c = int (input ('Enter c value'))
if a>b:
    if a>c:
        print('a is maximum')
    else:
        print('c is maximum')
else:
    if b>c:
        print ('b is maximum')
    else:
        print('c is maximum')
```

Looping Statements

Date: / /

→ 1. when we have to execute some set of instructions repeatedly then we will use looping statements

2. In python we have two types of looping statements.

1. For loop (when you know how many times to iterate)

2. while loop (when you are not sure how many times we will iterate)

Note:- If given CDT is empty the control will not enter in to for loop.

For Loop.

→ we use for loop when we know how many times to we have to iterate.

→ We can use for loop in 2 ways.

1. With collection Data types (CDT)
2. With range function.

Syntax:-

```
for variable in CDT:
    statements
    of for loop
```

→ For loop performs two operations.

1. Initialization
2. Traversing

Ex:- String

```
s = 'hello'
for i in s:
    print(i)
```

output:-

```
h
e
l
l
o
```

Date: /

Ex:- List

```
L = [11, 22, 33, 'hai', [90,78]]
for k in L:
    print(k)
```

output:-

```
11
22
33
'hai'
[90,78]
```

Ex:- Tuple

```
T = (11, 22, 33, 'hai', [90,78])
for k in T:
    print(k)
```

output:-

```
11
22
33
'hai'
[90,78]
```

Ex:- Set

```
S = {11, 22, 33, 'hai', (90,78)}
for k in S:
    print(k)
```

output:-

```
33
(90,78)
11
22
'hai'
```

Ex:- Dictionary

```
d = {'name':'Ashu', 'age':2}
for i in d:
    print(i, d[i])
```

output:-

```
name Ashu
age 2
```



```
d = {'name':'Ashu', 'age':2}
for k,v in d.items():
    print(k,v)
```

output:-

```
name Ashu
age 2
```

Date: / /

→ WAP to print how many elements present in given CDT

```
l = eval(input('Enter the elements :'))
elements = 0
for i in l:
    elements += 1
print('No. of elements are', elements)
```

output:-

Enter the elements : [11, 22, 33, 44, 55]
No. of elements are 5

→ WAP to print how many times a given substring is present is specified string.

```
string = input('Enter a String :')
Sub = input('Enter a substring :')
count = 0
for i in string:
    if i == sub:
        count += 1
print(count)
```

output :-

Enter a String : harshad
Enter a substring : W
0

Enter a String : prakash
Enter a substring : a
2

Date: / /

→ WAP to print how many vowels are present in string

```
S = input('Enter string:')
c = 0
vowels = 'aeiouAEIOU'
for i in S:
    if i in vowels:
        c+=1
print(c)
```

→ WAP to print how many consonants are present in string

```
S = input('Enter string:')
c = 0
vowels = 'aeiouAEIOU'
for i in S:
    if i not in vowels:
        c+=1
print(c)
```

(or)

```
S = input('Enter a string:')
c = 0
vowels = 'aeiouAEIOU'
for i in S:
    if i.isalpha():
        if i not in vowels:
            c+=1
print(c)
```

Date: / /

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

```
s = input ('Enter string :')
digits = '0 1 2 3 4 5 6 7 8 9'
c = 0
for i in s:
    if i.isdigit():
        if i in digits:
            c += 1
print (c)
```

→ WAP to print sum of digits present in given string.

```
num = input ('Enter a value')
Sum = 0
for i in num:
    if (i.isdigit())
    Sum = Sum + int(i)
print(sum)
```

→ WAP to print sum of even numbers presents in given string.

```
s = input('Enter a string:')
sum = 0
for i in s:
    if i.isdigit():
        n = int(i)
        if (n % 2 == 0)
            sum += n
print (sum)
```

Date: / /

For loop in range()

range() function:-
It is used for providing lower and upper limits to for loop to iterate.

Syntax:-
range (lowerlimit, upperlimit+1, updation)

Default values:-
lower limit → 0
updation → 1
only give 1 value it takes as upper limit.

→ Syntax of for Loop with range.

```
for variable in range (LL,UL):
    Statements of
    Loops
```

Ex:-

```
for i in range (1,5):
    print(i)
```

→ WAP to find sum of first Natural numbers.

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

Date: / /

→ WAP to find factorial of a given Number.

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

→ WAP to sum of even numbers in a given st range.

```
n = int(input('Enter a number:'))
n1 = int(input('Enter a number:'))
even =0
for i in range(n, n+1):
    if i%.2 ==0:
        even += i
print(even)
```

→ WAP to sum of odd numbers in a given range.

```
n =int(input('Enter a number:'))
n1 = int(input('Enter a number:'))
odd =0
for i in range(n,n+1):
    if i%.2!= 0:
        print(i)
```

Date: / /

For loop with combination of range & CDT

→ For loop functionality when we use with combinations of range and collection.

→ It is used when we are dealing with index positions of range & CDT.

→ Syntax:

```
for i in range (len (cdt)):
    Statements of for loop
    with both range & CDT
```

→ Ex:

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

output:

```
0 h
1 e
2 l
3 l
4 o
```

→ WAP to print index positions of vowels.

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

Date: / /

→ WAP to find sum of even index positions of vowels in a given string.

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

→ WAP to find sum of odd index positions of vowels in a given string.

```
v = input('Enter a string:')
Vowels = 'aeiouAEIOU'
Summ = 0
for i in range(len(v)):
    if v[i] in vowels:
        if i%.2 != 0:
            Summ += i
print(Summ)
```

(or)

```
s = input('Enter a string:')
Summ = 0
Vowels = 'aeiouAEIOU'
for i in range(len(s)):
    if s[i].isalpha():
        if s[i] not in vowels:
            if i%.2 == 1:
                Summ += i
print(Summ)
```

Date: / /

→ WAP to create to a new string after replacing vowels with the output. which is obtained after multiplying their index position with 3.

```
S = input ('Enter a string:')
dummy = ''
Vowels = 'AEIOUaeiou'
for i in range(len(s)):
    if s[i] in vowels:
        k = i * 3
        dummy += str(k)
    else:
        dummy += s[i]
print (dummy)
```

Test case 1:

Q. Input = L = [11, 22, 33, 44, 55] → odd

output - Invalid input

Test case 2:

Input L = [100, 22, 33, 55, 66, 700, 80, 99]

output [22, 100, 55, 33, 700, 66, 99, 80]

```
L = eval (input ('Enter list'))
n = len(L)
if n % 2 == 1:
        print ('Invalid input')
else:
    for i in range (0, len(L), 2):
        ~~L[i], 0~~
        L[i], L[i+1] = L[i+1], L[i]
    print(L)
```

While Loop.

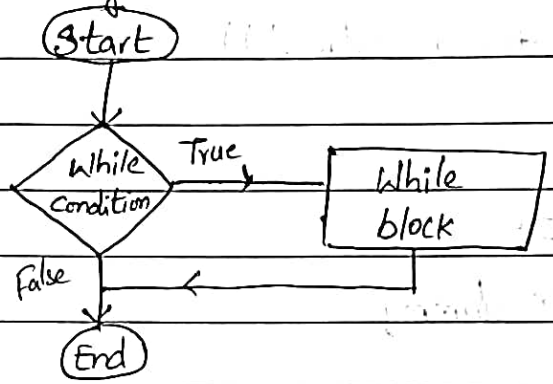
Date: / /

1. We use while loop when we are not sure how many times we have to iterate.
2. While loop will be executed based on one condition
3. There is a chance to execute your while loop for infinite times, in order to avoid that make sure at one point of time the condition become false.

Syntax:-

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

Flow Diagram:-



Ex:-

```
a = 10
while a > 5 :
    print(a)
    a -= 1
```

output:-

6
7
8
9
10

Ex:-

```
b = 10
while b < 15 :
    print(b)
    b += 1
```

output:-

10
11
12
13
14

Date: / /

→ Wap to find sumn of first n natural numbers by using while loop.

```
n = int (input('Enter a value :'))
Sumn = 0
dummy = 1
while dummy <= n:
    Sumn += dummy
    dummy += 1
print (sumn)
```

→ WAP to find the factorial of given number by using while loop.

```
F = int (input ('Enter a value :'))
fact = 1
dummy = 1
while dummy <= F:
    fact = fact * dummy
    dummy += 1
print (fact)
```

Special Statements

Date: / /

→ Special statements of python

1. break } Note :- we can use break & continue only in loops.
2. Continue }
3. Pass

→ 1. break :-

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

Ex :-

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

output :-

1
2
3
4

Ex :-

```
a = 10
while a>1:
    print(a)
    if a==5:
        break
    a-=1
```

output :-

10
9
8
7
6

→ 2. Continue :-

Continue is used for skipping the single iteration based on given condition.

Ex :-

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

Date: / /

→ 3. Pass:-

1. pair d is used for creating empty blocks.
2. pass acts as the placeholder 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.

Ex:-

```
def function():
    pass
```

Ex:-

```
if 10>1:
    pass
```

Ex:-

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

→ WAP to print sum of all digits in a given number.

```
x = int(input('Enter your number:'))
T = x
res = 0
while x>0:
    rem = x%10
    res = res + rem
    x = x//10
print(f'sum of {T} is {res}')
```

output:-

Sum of 213 is 6

Date / /

→ WAP to find product of all digits in a given number.

```
x = int(input('Enter a number:'))
T = x
res = 1
while x > 0:
    rem = x % 10
    res = res * rem
    x = x // 10
print(f'Sum of {T} is {res}')
```

output:-

Sum of 213 is 12

→ WAP to print sum of even digits and product of odd digits in a given number.

```
x = int(input('Enter a number:'))
T = x
res1 = 0
res2 = 1
while x > 0:
    rem = x % 10
    x = x // 10
    if rem % 2 == 0:
        res1 = res1 + rem
    else:
        res2 = res2 * rem
print(f'Sum of {T} is {res1}')
print(f'product of {T} is {res2}')
```

Date: / /

→ WAP to check the given number is perfect number or not

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

output:-

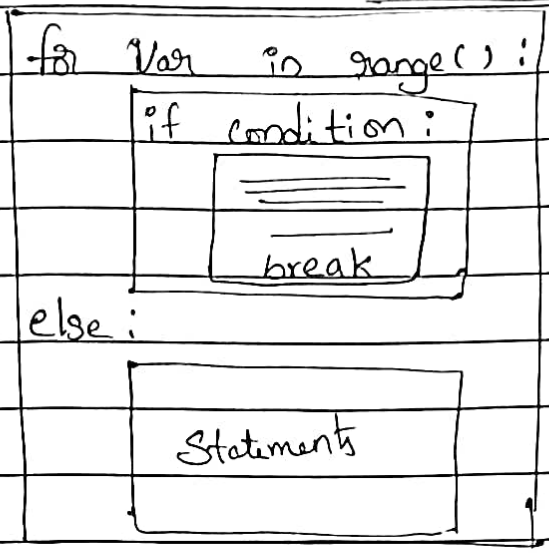
Enter a number is : 6
n is perfect.
Enter a number is : 10
n is not perfect.

(or) By using while loop.

```
n = int(input('Enter a number is :'))
Sum = 0
dummy = 1
while dummy = n//2 :
    if n % dummy == 0:
        Sum += dummy
        dummy += 1
if Sum == n:
    print('N is perfect')
else:
    print('N is not perfect')
```

For-Else

Date: / /



Note:-

→ In case of for-else block, else block is executed when for loop is not terminated.

→ If it terminate happened else block is not executed.

Ex:-

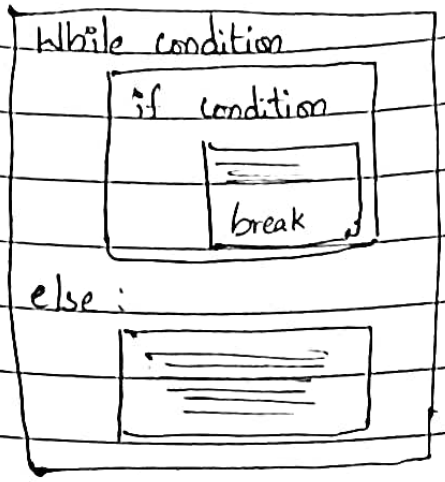
```
for i in range(1,10):
    print(i)
    if i==15:
        break
else:
    print('I am else block.')
```

Ex:- WAP to print & find given number is prime or not by using for-else.

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

Date: / /

While-Else



Ex:- 1

```
a = 1
while a < 10 :
    print(a)
    if a == 5:
        break
    a += 1
else:
    print('I am else block')
```

→ WAP to find given number is prime or not by using while

```
n = int(input('Enter a number is:'))
i = 2
while i <= n//2:
    if (n%i == 0):
        print('n is not prime')
        break
        i += 1
else:
    print('n is prime')
```

Date / /

→ WAP to find given number is pali-prime or not

```
n = int(input('Enter a number:'))
for i in range(2, n//2+1):
    if (n%i==0):
        print('n is not pali-prime')
        break
else:
    if (str(n)==str(n)[::-1]):
        print('n is prime-pali')
    else:
        print('n is not pali-prime')
```

output:-

11 is pali prime

13 is not pali-prime

→ WAP to find the given number is Emirp number or not.

```
n = int(input('Enter a number is:'))
for i in range(2, n//2+1):
    if (n%i==0):
        print('n is not emirp')
        break
else:
    rev = int(str(n)[::-1])
    for i in range(2, rev//2+1):
        if (rev%i==0):
            print('n is not Emirp')
            break
    else:
        print('n is Emirp Number')
```

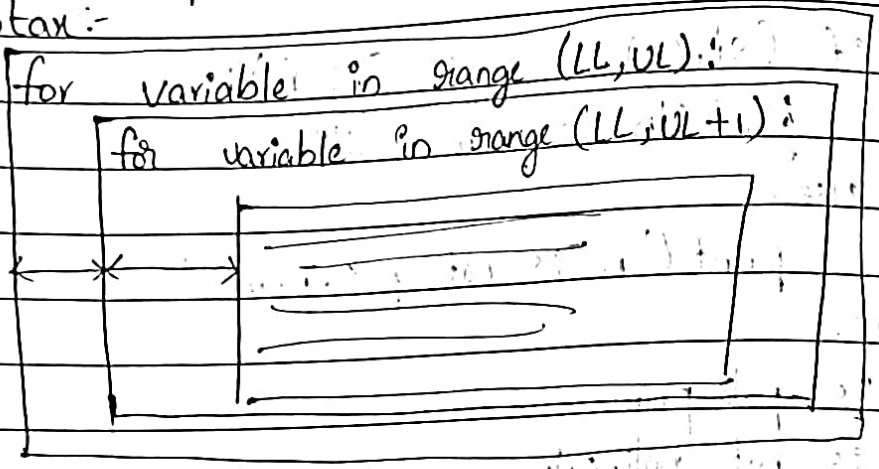
Date: / /

Nested loops

For → for
For → while
while → for
while → while

Nested loops :-
It is the process of debining a loop inside another loop

Syntax :-



Ex :-

```
for i in range (1,4):
    for j in range (1,6):
        print(i,j)
```

output :-

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

Date: / /

For in For

→ Terminating Innerloop with inner loop value.

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

→ Terminating Inner loop with outer loop value.

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

→ Terminating outerloop with outer loop value.

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

→ Terminating outerloop with inner loop value.

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

(or)

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

Date: / /

→ Print all the prime numbers in a given range.

```
LL = int (input ('Enter LL'))
UL = int (input ('Enter UL'))
for i in range (LL, UL+1):
    if i>1:
        for j in range (2, i//2 +1):
            if i%j==0:
                break
        else:
            print(i)
```

→ WAP to print every second prime in a given range

```
LL = int(input ('Enter a lower limit'))
UL = int (input ('Enter a Upper limit'))
dummy =0
for n in range (LL, UL+1):
    if n>1:
        for i in range (2, n//2+1):
            if n%i==0:
                break
else:
    dummy +=1
    if dummy %2 ==0:
        print(n)
```

interview Question.

Date / /

→ WAP to print every third number in a given range.

```
LL = int(input("Enter a LL value"))
UL = int(input("Enter a UL value"))
dum = 0
for n in range(LL, UL+1):
    if n>1:
        for i in range(2, n//2+1):
            if n % i == 0:
                break
        else:
            dum += 1
            if dum % 3 == 0:
                print(n)
```

Interview question

→ Find sum of digits of a given number.

```
n = int(input('Enter a value :'))
S = str(n)
Sum = 0
for i in S:
    Sum += int(i)
print(sum)
```

type casting Technique.

(or)

```
n = int(input('Enter a value :'))
Sum = 0
while n>0:
    rem = n % 10
    Sum += rem
    n = n // 10
print(sum)
```

Date: / /

→ WAP to check Armstrong Number or not.

```
n = int(input('Enter a value:'))
L = len(str(n))
Sum = 0
num = n
while n > 0:
    rem = n % 10
    Sum += rem**L
    n = n//10
if Sum == num:
    print('Armstrong')
else:
    print('Not Armstrong')
```

interview Question

→ WAP to check the given number is Disarium number or not.

```
n = int(input('Enter a value:'))
L = len(str(n))
Sum = 0
num = n
while n>0:
    rem = n % 10
    Sum += rem**L
    n = n//10
    L -= 1
if Sum == num:
    print('n is Disarium Number')
else:
    print('n is not Disarium Number')
```

interview Question.

Date: / /

Harshad / Niven Number:-

=> A harshad number (or niven number) in a given number base is an integer that is divisible the sum of its digits.

(or)

=> A number is said to be harshad / Niven number if it is perfectly divided with sum of its digits.

→ Check the given number is Harshad number or not.

```
n = int(input('Enter a value:'))
Sum = 0
num = n
while n > 0:
    rem = n % 10
    Sum += rem
    n = n // 10
if num % Sum == 0:
    print('Harshad / Niven number')
else:
    print('Not a Harshad / Niven number')
```

→ WAP to print all the niven numbers from given range.

```
LL = int(input('Enter a value:'))
UL = int(input('Enter a value:'))
for i in range(LL, UL+1):
    Sum = 0
    num = i
    while i > 0:
        rem = i % 10
        Sum += rem
        i = i // 10
    if num % Sum == 0:
        print(num)
```

Date: / /

→ To print all the disarium numbers from given range.

```
num = int(input("Enter the numbers:"))
length = len(str(num))
temp = num
sum = 0
rem = 0
while temp > 0:
    rem = temp % 10
    sum = sum + int(rem ** length)
    temp = temp // 10
    length = length - 1
print("The sum of digits:", sum)
if sum == num:
    print(num)
```

→ To print all the armstrong numbers from given numbers

```
start = int(input('Enter start value:'))
end = int(input('Enter end value:'))
for n in range(start, end+1):
    l = len(str(n))
    sum = 0
    num = n
    while n > 0:
        rem = n % 10
        sum += rem ** l
        n = n // 10
    if (num == sum):
        print(num)
```

Date: / /

→ WAP to print every second Harshad Number in a given Number

```
LL = int(input('Enter a LL number'))
UL = int(input('Enter a UL number'))
dummy = 0
for n in range(LL, UL+1):
    Summ = 0
    num = n
    while n>0:
        rem = n %. 10
        Summ += rem
        n = n //10
    if num %. Sum == 0:
        dummy += 1
        if dummy %. 2 == 0:
            print(num)
```

→ Check a given number is Special Number or Not.

```
n = int(input("Enter a number:"))
num = n
Sum = 0
while n>0:
    rem = n %. 10
    fact = 1
    for i in range(1, rem+1):
        fact *= i
    Sum += fact
    n = n //10
if num == Sum:
    print('Special number')
else:
    print('Not special number')
```

Date: / /

→ To print all the special numbers in a given range.

```
LL = int(input('Enter Lower Limit'))
UL = int(input('Enter Upper Limit'))
for n in range(LL, UL+1):
    num = n
    sum = 0
    while n>0:
        rem = n%10
        fact = 1
        for i in range(1, rem+1):
            fact *= i
        sum += fact
        n = n//10
    if num == sum:
        print(num)
```

Nested while loops

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

Date: / /

→ Breaking 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
```

→ Breaking innerloop 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
```

→ Breaking outerloop with outerloop value.

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

Date: / /

→ Breaking outer loop with inner loop value.

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

While inside for loop.

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

For inside while loop

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

For loop in while loop.

Date / /

Breaking innerloop with innerloop value.

```
for i in range(1,6):
    j=1
    while j<6:
        print(i,j)
        if j==4:
            break
        j+=1
```

Breaking innerloop with outerloop value

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

Breaking outerloop with outerloop value

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

Breaking outerbop with innerloop value.

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

Date: / /

While loop inside For loop

Breaking innerloop with innerloop value.

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

Breaking innerloop with outer loop value

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

Breaking outerloop with outer loop value

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

Breaking outerloop with innerloop value

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

Pattern Programs.

Date: / /

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	6,5
*	*	*	*	*

→ Square

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

output.

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

→ Diagonal

```
n = int(input('Enter a value:'))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i==j:
            print('*',end=' ')
        else:
            print(':',end=' ')
    print()
```

output.

```
*
: *
: : *
: : : *
```

→ Reverse Diagonal

```
n= int (input('Enter a value:'))
for i in range (1,n+1):
    for j in range (1,n+1):
        if i+j == n+1:
            print('*',end=' ')
        else:
            print (' ',end=' ')
    print()
```

output:

Enter a value 5

```
    *
  *
*
  *
    *
```

→ WAP to print X pattern

```
n = int (input ('Enter a value'))
for i in range (1,n+1):
    for j in range (1,j+1):
        if i+j == n+1 or i==j:
            print('*',end=' ')
        else:
            print (' ',end=' '):
    print()
```

output:-

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

→ Right angle Triangle ◺

```
n = int (input ('Enter a value:'))
for i in range (1,n+1):
    for j in range (1,n+1):
        if i >= j:
            print('*',end=' ')
        else:
            print (' ',end=' ')
    print()
```

output:-

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

→ left angle triagle ◺

```
n = int(input('Enter a value:'))
for i in range(1,n+1):
    for j in range(1,n+1):
        if (i+j)>=n+1):
            print('*',end=' ')
        else:
            print (' ',end=' ')
    print()
```

output:

```
Enter a value: 5
        *
      * *
    * * *
  * * * *
* * * * *
```

→ upper right angle ◹

```
n = int (input ('Enter a value :'))
for i in range (1,n+1):
    for j in range (1,n+1):
        if  i+j <= n+1 :
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
```

output:

```
Enter a value : 5
* * * * *
* * * *
* * *
* *
*
```

→ upper left angle triangle ◹

```
n = int (input ('Enter a value:'))
for i in range (1,n+1):
    for j in range (1,n+1):
        if i<=j:
            print('*',end=' ')
        else:
            print(' ',end=' ')
    print()
```

output:

```
Enter a value : 5
* * * * *
  * * * *
    * * *
      * *
        *
```

Date: / /

→ Down triangle.

```
n = int(input("Enter a value:"))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i>=j and i+j>=n+1:
            print('*', end=' ')
        else:
            print('@', end=' ')
```

output:-

@ @ @ @ @
@ @ @ @ @
@ @ * @ @
@ * * * @
* * * * *

→ up triangle

```
n = int(input("Enter a value:"))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i<=j and i+j<=n+1:
            print('*', end=' ')
        else:
            print('@', end=' ')
    print()
```

output:-

* * * * *
@ * * * @
@ @ * @ @
@ @ @ @ @
@ @ @ @ @

→ Right triangle.

```
n = int(input("Enter a value:"))
for i in range(1,n+1):
    for j in range(1,n+1):
        if i<=j and i+j>=n+1:
            print('*', end=' ')
        else:
            print('$', end=' ')
    print()
```

output:-

$ $ $ $ *
$ $ $ * *
$ $ * * *
$ * * * *
* * * * *

Date / /

→ Left Triangle.

```
n= int (input ('Enter a value :'))
for i in range (1, n+1):
    for j in range (1, n+1):
        if i>=j and i+j <= n+1:
            print ('*', end = ' ')
        else:
            print ('$', end = ' ')
    print()
```

output :-

```
* $ $ $ $
* * $ $ $
* * * $ $
* * $ $ $
* $ $ $ $
```

→ Horizontal Same number.

```
n = int (input ('Enter a value:'))
dummy = 1
for i in range (1, n+1):
    for j in range (1, n+1):
        print (dummy, end = ' ')
    dummy += 1
    print()
```

output :-

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

→ Vertical Same number

```
n = int (input ('Enter a value :'))
for i in range (1, n+1):
    dummy = 1
    for j in range (1, n+1):
        print (dummy, end = ' ')
        dummy += 1
    print()
```

output :-

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

Date: / /

→ Serial Numbers.

```
n = int(input('Enter a value:'))
dummy = 1
for i in range(1,n+1):
    for j in range(1,n+1):
        print(dummy,end=' ')
        dummy += 1
    print()
```

output:-

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

→ Horizontal Same character.

```
n = int(input('Enter a value:'))
dummy = 65
for i in range(1,n+1):
    for j in range(1,n+1):
        print(chr(dummy),end=' ')
    dummy += 1
    print()
```

output:-

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

→ Vertical same characters

```
n = int(input('Enter a value:'))
for i in range(1,n+1):
    dummy = 65
    for j in range(1,n+1):
        print(chr(dummy),end=' ')
        dummy += 1
    print()
```

output:-

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

Date: / /

→ Serial alphabets

```
n=int(input('Enter a value:'))
dummy=65
for i in range(1,n+1):
    for j in range(1,n+1):
        print(chr(dummy),end=' ')
        dummy+=1
    print()
```

output

```
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
```

→

```
n = int(input('Enter a value :'))
dummy = 65
for i in range(1,n+1):
    for j in range(1,n+1):
        if i>=j:
            print(chr(dummy),end=' ')
        else:
            print(' ',end=' ')
    dummy+=1
    print()
```

output

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

→

```
n=int(input('Enter a value :'))
dummy=65
for i in range(1,n+1):
    for j in range(1,n+1):
        if i<=j:
            print(chr(dummy),end=' ')
        else:
            print(' ',end=' ')
    dummy+=1
    print()
```

output:-

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

Date: / /

→ Right angle triangle with serial alphabets.

```
n = int(input("Enter a value:"))
dummy = 65
for i in range(1, n+1):
    for j in range(1, n+1):
        if i >= j:
            print(chr(dummy), end=' ')
            dummy += 1
        else:
            print(' ', end=' ')
    print()
```

output:

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

Functions

Date / /

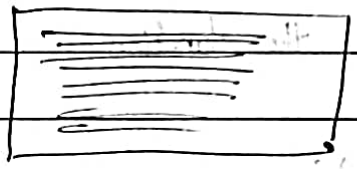
→ 1. Functions are independent block of instructions which are used for performing some task.
2. Functions are used for increasing the code re-usability.
3. Functions are used for reducing the code Redundancy (Repetation)

→ Classification of Functions:-
we have 2 types of Functions.
1. Built-in Functions.
2. User-Defined Functions.

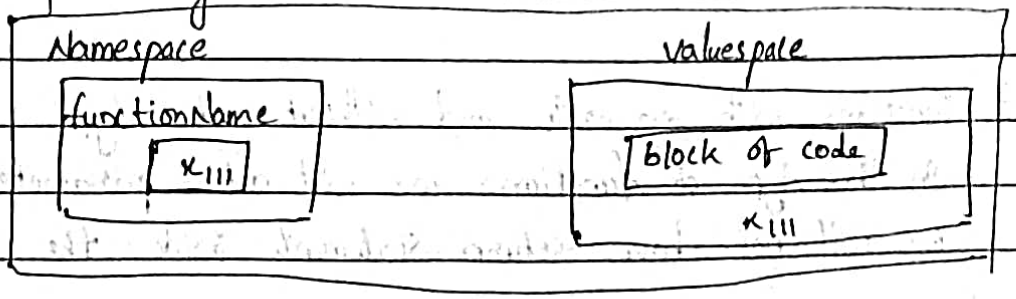
→ Built-in Functions:-
These are the functions which are created by the developers by python.
Ex:- len(), print(), input() etc....

→ User-Defined Functions:-
1. These are the functions which are developed or defined by the user on his requirements.
2. "def" keyword is used for defining user defined functions.

→ Syntax for user-defined Function:-
def functionName():



→ Memory Diagram



Date: / /

Ex:-
```
def fw:
def wish():
    print('welcome')
    print('Good Morning')
    print('How are you')
def tiffen():
    print('chocolate')
print(wish) # printing address of function
wish()      # calling the function
tiffen()    # calling the function
```

→ Classification of User-Defined Functions :-

→ User-defined functions are classified in to 4 types.

1. Functions without arguments and without return type
2. Functions with arguments and without return type
3. Functions without arguments and with return type
4. Functions with arguments and with return type.

→ Functions without arguments and without return type

In this type of functions we will not define parameters and return statements inside the function.

Syntax:-

```
def functionName():
    statements
```

→ Functions with arguments and without return type

In this type of functions we will define parameters and we will not have return statements inside the function.

Date / /

Ex:-

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

Arguments:- Arguments are the essential parameters which are required by the function for their execution.

→ Classification of Arguments:

we have 4 types of Arguments

1. Mandatory or positional Arguments
2. Non-mandatory or Default Arguments.
3. Variable length keyword Arguments (**kwargs)
4. Variable length Arguments (*args)

→ 1. Mandatory or positional Arguments:-

If you are defining a function with mandatory or positional Arguments are compulsory we have to pass that many parameters while you call that function.

Ex:-

```
def add(a,b):
    print('performing addition operation')
    print('a value is', a)
    print('b value is', b)
    print(a+b)
#add()
#add(89)
#add(89,78,67)
add(23,45) # output 68 → correct.
```

} errors

Date: / /

→ 2. Default or Non-mandatory arguments :-
In this type arguments will have its 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.

Ex :-

```
def add(a=89, b=67):
    print('a value is', a)
    print('b value is', b)
    print(a+b)
add()
add(100)
add(100,200)
# add(100,21,23,300) → error
```

} correct

Note :- If we want to define mandatory and non-mandatory in a function then first mandatory arguments should be defined after words give non-mandatory arguments.

Ex :-

```
def add(a, b=111):
    print(a)
    print(b)
    print(a+b)
add(444)
```

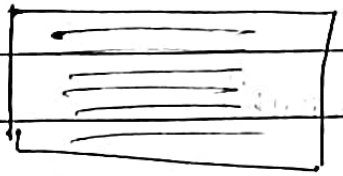
Date: / /

→ 3. 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 as parameters while calling it.
3. output format of variable length arguments is a tuple.

→ Syntax:-

```
def function (*name):
```



Ex:-

```
def add (*args):
    print (args)
    print (type(args))
    Summ = 0
    for i in args:
        Summ += i
    print (summ)
add()
add(11,22,33,44)
```

→ 4. Functions with variable length keyword Argument

1. In python ** defines variable length keyword Argument.
2. Output format of variable length keyword argument is a dictionary.
3. once you define a function with variable length keyword arguments then we have to provide parameters in the form of pairs as shown below:-

argument1 = value1, argument2 = value2, ... argumentn = valuen

Date: / /

→ Syntax:-

```
def add (**kwargs):
```

Ex:-

```
def add (**kwargs):
    print (kwargs)
    print (type (kwargs))
add()
add (a=19, b=20, name='prakash')
```

→ Defining all the arguments in one function.

```
def add (a, b=90, *args, **kwargs):
    print(a)
    print (b)
    print (args)
    print (kwargs)
add (12, 33, 45, 67, 55, x=90, y=78)
```

Output:-

```
12
33
(45, 67, 55)
{'x': 90, 'y': 78}
```

Tricky Question.

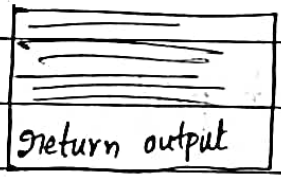
→
```
def add(a,b):
    print(a)
    print(b)
a = 20
b = 30
add(b,a)
```

Output -
30, 20

→
```
def add(a,b):
    print(a)
    print(b)
    if b>a:
        print(b)
    else:
        print(a)
a=20
b=30
add(b,a)
```

→ 3. Functions without arguments and with return type.
In this type of function we will define a function which is not accepting arguments but it will have no return type.
Syntax:-

```
def add():
```



Date: / /

Return Statement

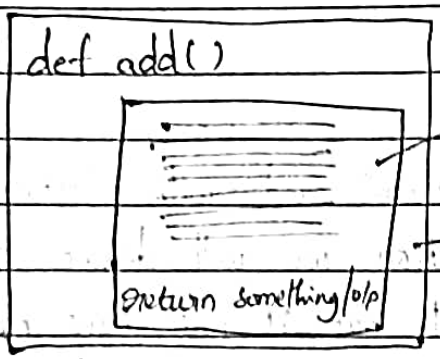
→ Return statement is used for returning some values or output from the function space to main space.

→ Properties of return:-

1. After return we cannot write a single statement in that 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 statements in it then it should be called in any of the mentioned ways.

1. Call a function inside print
2. Assign a function call in to a variable.
3. Call a function in conditional statements



Ex:- def add()
print ('am inside add')
return 5+7
print ('first') → Inside print
# print (add())
return = add() → Assigning function to variable.
print (result)
print (result/2)

Date: / /

→ output without return → output with return.

```
First                          First
i am inside add                i am inside add
None                           13
```

→ WAP to print Even numbers by using functions.

```
def evennumbers(LL,UL):                    ①
    for i in range(LL,UL+1):
        if iseven(i):
            print(i)
evennumbers(10,20)
def isEven(n):                             ②
    if n%2==0:
        return True
    else:
        return False
```

output :-
10
12
14
16
18
20

→ WAP to print prime numbers in the given range by using function

```
def isPrime(n):
    if n>1:                                ①
        for i in range(2,n//2):
            if n%i==0:
                return False
        else:
            return True
    else:
        return False
def primeNumber(LL,UL):                    ②
    for i in range(LL,UL+1):
        if isPrime(i):
            Print(i)
```

Date / /

→ WAP to print emirp number in the given range by using functions.

```
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):
    if isprime(n):
        rev = int(str(n)[::-1])
        if isprime(rev):
            return True
        else:
            return False
    else:
        return False.
def emirpnumbers(LL,UL):
    for i in range(LL,UL+1):
        if isEmirp(i):
            print(i)
#Calling Function
emirpnumbers(10,1000)
```

Date: / /

WAP to check armstrong or not

```
def isarmstrong(a):
    summ=0
    l=len(str(a))
    num=a
    while a>0:
        rem=n%10
        sum+=rem**l
        n=n//10
    if(num==sum):
        return True
    else:
        return False
def armstrong(LL,UL):
    for i in range(LL,UL+1):
        if isarmstrong(i):
            print(i)
armstrong(10,50)
```

Check disarium number or not

```
def isdisarium(d):
    sum=0
    l=len(str(d))
    num=d
    while d>0:
        rem=n%10
        sum+=rem**l
        d=d//10
        l-=1
    if(num==sum):
        return True
    else:
        return False
def disarium(LL,UL):
    for i in range(LL,UL+1):
        if isdisarium(i)
            print(i)
disarium(10,200)
```

Date: / /

Check number is harshad or not

```
def isharshad(h):
    sum=0
    num=h
    while h>0:
        rem=h%10
        sum+=rem
        h=h//10
    if(num%sum==0):
        return True
    else:
        return False
def harshad(LL,UL):
    for i in range(LL,UL+1):
        if isharshad(i):
            print(i)
harshad(10,1000)
```

check number is special or not

```
def issp1(s):
    sum=0
    num=s
    while s>0:
        rem=s%10
        fact=1
    for i in range(1,rem+1):
        fact*=i
        sum+=fact
        n=n//10
    if num==sum:
        return True
    else:
        return False
def sp1(LL,UL):
    for i in range(LL,UL+1):
        if issp1(i):
            print(i)
sp1(100,1000)
```

Date: / /

Q. s = 'harshad'

Replace the vowels with the output which has been obtained after performing below operation.

① multiply IP of vowels 20(k)

② you need to find out sum of prime numbers from 1 to k.

③ After that replace vowels with sum.

→
```
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 prime(LL,UL):
    Summ=0
    for n in range(LL,UL+1):
        if isprime(n):
            Summ+=n
    return summ
def manipulatestring(s):
    dummy=''
    vowels='aeiouAEIOU'
    for i in range(len(s)):
        if s[i] in vowels:
            k=i*20
            Sum=prime(1,k)
            dummy+=str(sum)
        else:
            dummy+=str[i]
```

Date: / /

Types of variables in Functions

In functions we have 3 types of variables.

1. Local variables
2. Global variables
3. Non local variables

→ Local Variables :-

1. These are the variables which are created inside function space.
2. Local variables can be accessed and modify only is with in that function.
3. We cannot Access Local variables in main space.

→ Global variables :-

1. These are the variables which are created outside the function and accessed and modified inside the function.
2. In order to make a normal variable into global, we have to use global keyword.

Syntax :-

```
global var1, var2, var3, .... varn
```

Ex :-

```
b, a = 10, 20
print (a,b)  # 20, 10
def bai():
    global a, b
    print (a, b)
    a -= 1
    b -= 2
    print (a,b) # 9, 18
bai()
print (a, b) # 9, 18
```

Date / /

→ Non-local variables:-

1. Nonlocal variables are used in ~~methods~~ nested functions, to make variable defined in parents function to be accessed and modified in the child function.
2. To make normal variables in to ~~normal~~ nonlocal we have to use nonlocal keyword.

Syntax:-

nonlocal var1, var2, var3, ... varn

Ex:-

```
def parent():
    a,b = 10,20
    print(a,b)
    def child():
        nonlocal a,b
        print(a,b)
        a -= 2
        b += 12
        print(a,b)
    child()
    print(a,b)
parent()
```

Recursive Functions

Date: / /

It is the process of calling the function with in itself until and unless the terminating condition satisfied.

→ Benifits of Recursion:-

1. It reduces the code complexity.
2. When you want to repeat the same process recursion is the best way.

→ Demerits of Recursion:-

1. memory consumption is more.

→ Syntax:-

```
def recursion_function():
    if condition:
        return
    recursion_function
```

→ Ex:-

```
def disphy(n):
    print(n)
    if n==8:
        return
    disphy(n-1)
disphy(10)
```

Date / /

→ WAP: to findout a factorial number by using recursion.

```
def fact(n):
    if n==1 or n==0:
        return 1
    return n * fact(n-1)
print(fact(4))
```

→ WAP to findout a sum of n natural numbers from 1 to 10.

```
def sum(n):
    if n==0:
        return 0
    return n + sum(n-1)
print(sum(10))
```

→ WAP to findout the sum of individual digits of given number by using recursion.

```
def sum_digits(n):
    if n==0:
        return 0
    return n%10 + sum_digits(n//10)
sum(145)
print(sum)
```

Importing

Date: / /

→ 1. Importing is the process of accessing the properties of one module into another module.
2. Import keyword is used for performing importing.

→ We can import the module in 2 ways.
1. Entire module
2. Specific functions from the module.

→ Syntax to import entire module:-
import modulename

→ Syntax for accessing functions from imported module:-
modulename.functionname()

Ex:-

Calci.py

```
def add(x):
    print(10+20)
def sub():
    print(20-10)
def mul()
    print(30*2)
```

calci
add()
sub()
mul()

operations.py

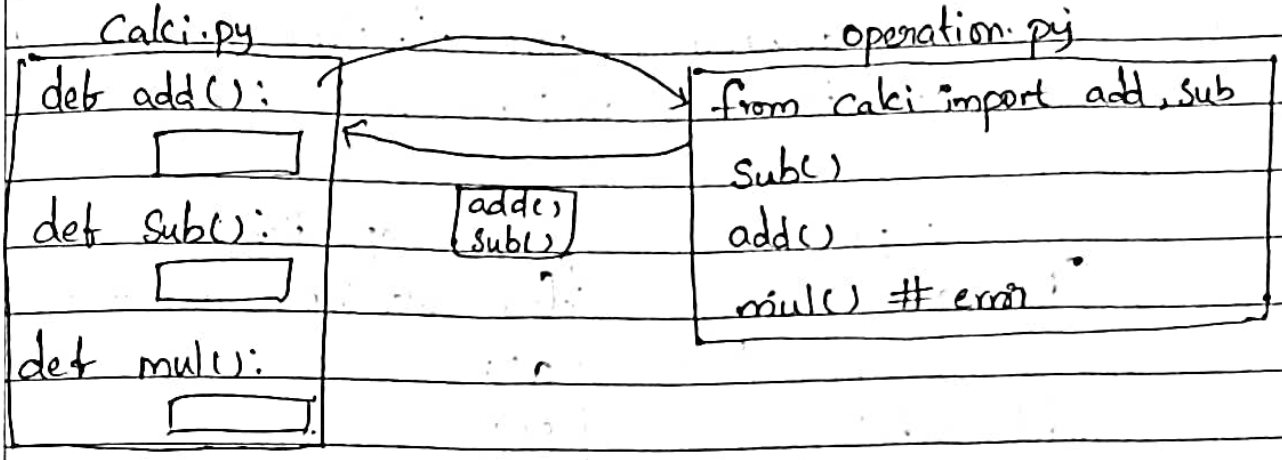
```
import calci
calci.add()
calci.sub()
calci.mul()
```

→ procedure & Syntax to import specific functions we use From import keyword.

→ Syntax to importing specific functions from module.
Ex: From modulename import add, sub, mul
From modulename import function1, function2, ... functionn

→ Syntax for accessing the functions.
As we have imported functions directly we can access functions with their names.
Functionname()

Ex:-



Aliasing

→ It is the process of renaming the imported module or functions or classes.

→ We can perform aliasing by using "as" keyword.

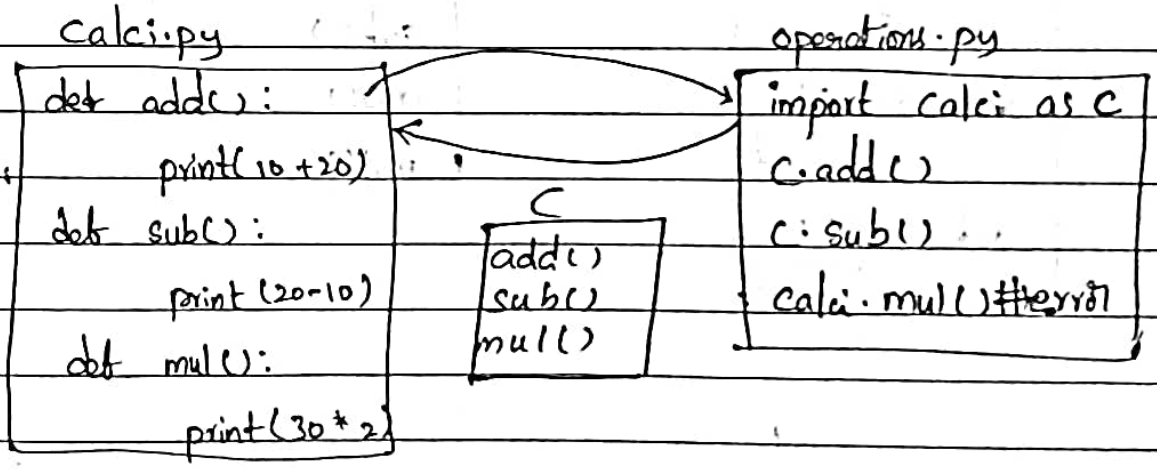
→ Syntax for aliasing the imported module.

import modulename as aliasingname

→ Syntax for accessing the functions of imported module.

aliasingname.functionname()

Ex:-



Date: / /

→ providing alternate names for imported functions.

Syntax:- From modulename import function1 as aliasingname, function2 as aliasingname,.....

Ex:-

calci.py

```
def add():
    [ ]
def sub():
    [ ]
def mul():
    [ ]
```

operation.py

```
from calci import add as a, sub as s
a()
s()
```

→ Syntax for importing all the functions directly

from modulename import *

Ex:-

calci.py

```
def add():
    [ ]
def sub():
    [ ]
def mul():
    [ ]
```

operations.py

```
from calci import *
sub()
add()
mul()
```

Date / /

→ 1. prioritization / restricting of execution of function which you are importing.
2. We can set the prioritization / restricting by using __name__ special variable.

→ Functionalities of __name__
→ 1. This is used for giving the names of the module.
2. If you are executing the module directly __name__ returns __main__ as the name of current module.
3. If you are executing the module from another module, then __name__ returns "Actual name" as the name of imported module.

Ex:-

Calci.py

```
def add():
    print(20+10)
def sub():
    print(20-10)
def mul():
    print(20*2)
if __name__=='__main__':
add()
sub()
mul()
print(__name__)
```

output

```
__main__
```

operations.py

```
import calci
calci.add
```

output

```
30
```

Date: / /

Packages

→ collection of modules along with __init__.py is known as package.

Importing module from a package:-

Syntax:-

from packagename import modulename

Syntax for accessing the functions.

modulename.functionname()

Ex:-

app

__init__.py
modules.py
test.py
admin.py
Views.py

```
def first():
    
def seond():
```

project

__init__.py
setting.py
assgi.py
NSgi.py
Urls.py

```
from app import views
Views.first()
Views.seond
```

→ Importing all functions from a package.

→ Syntax for importing:-

from packagename.modulename import *

→ Syntax for accessing the functions:-

functionname()

Date / /

Ex:-

app

```
__init__.py
module.py
test.py
admin.py
View.py
    def first():
        ...
    def second():
        ...
```

project

```
__init__.py
settings.py
asgi.py
Nsgi.py
urls.py
    from apps.views import *
    first()
    second()
```

→ Syntax for aliasing name for function.

from packagename import modulename as aliasingname

→ ~~accessing function syntax :-~~

Ex:-

app

```
__init__.py
models.py
tests.py
urls.py
    def first():
        ...
```

project

```
__init__.py
settings.py
test.py
views.py
    from app.urls import first as f
    f()
```

→ Syntax :-

from packagename import modulename as aliasingname

calling :-

modulename.function()

→ Ex :-

app

```
--init--.py
model.py
sets.py
views.py
    def first():
        ...
    def second():
        ...
```

project

```
--init--.py
seti.py
asgi.py
Nsgi.py
urls.py
from app.views as
```