Python ①
Python ①
Python ①

- To check the python version

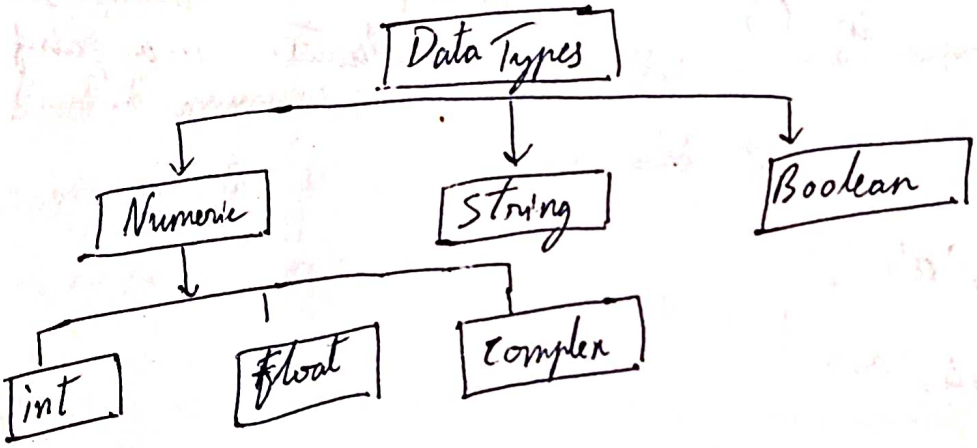
  In command prompt give command `python --version`

  python (space) --version

- Indentation refers to the spaces at the beginning of the code line. (generally there are 4 spaces left)
- We can always use # to create comments.

```
print ("Hello world")  # This is a print function
```

- Data Types In Python :-



Examples

int (integers) → 5, 10, 2, 1, 6, 100, 1246 etc

float (with decimals) → 10.5, 4.0, 1.0, 1246.2681 etc

Complex → 2+3j, 4+6j, 7+6j etc

String → "Vishal", 'tupperware', '2489', "4682" etc

Boolean → True (and) False

Built-in Data Types :-

Text type → str

numeric type → int, float, complex

Sequence type → list, tuple, range

Mapping type → dict

set type → set, frozenset

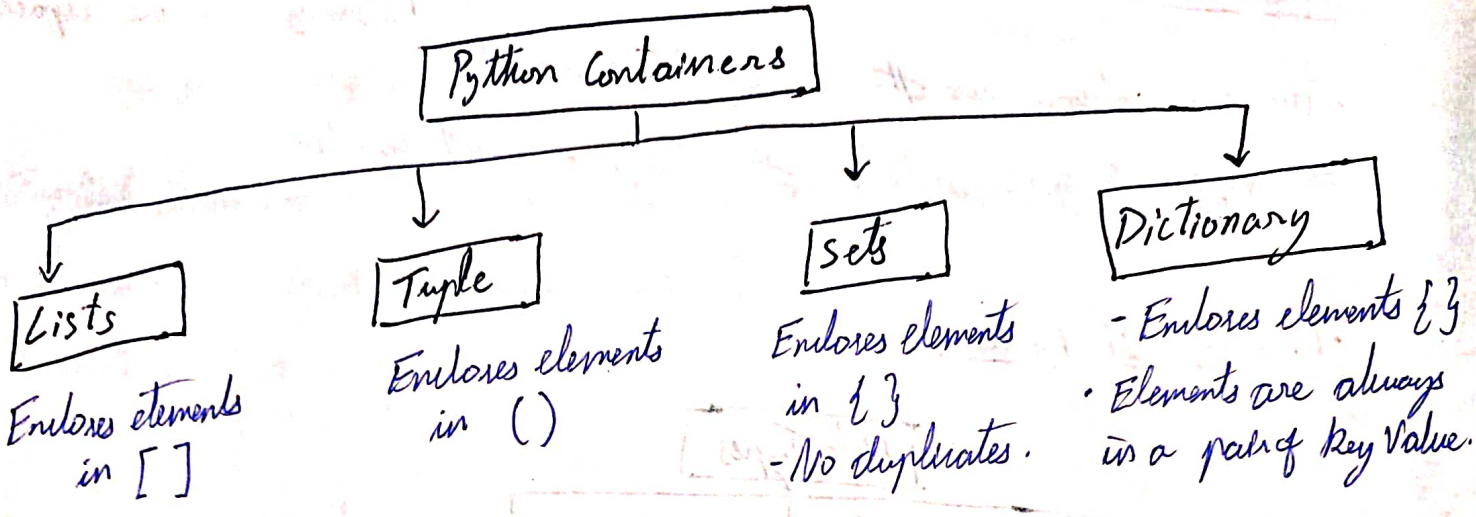
Boolean Type → bool

Binary Types → bytes, bytearray, memoryview

Python Containers :-

Containers are objects that holds an arbitrary number of objects.

Built-in Containers → tuples, lists, sets, dictionaries.



Examples :-

List [1, 2, 3, 4, 'abc']

output [1, 2, 3, 4, 'abc']

tuple (1, 2, 3, 4)

output (1, 2, 3, 4)

Set {2, 's', 3, 5, 5}

output {2, 3, 5, 's'}

Dictionary {1: "Jane", 2: "George", 3: "Sam"}

output {1: 'Jane', 2: 'George', 3: 'Sam'}

Variables in Python :-

→ A python variable points to a reserved memory location

→ Data or objects are stored in these memory locations.

→ Variables can be declared by any names or even alphabets.

- input() always accepts value in "str" format

Data Type Conversion

Implicit Conversion :- Conversion done by the python interpreter without programmer's intervention.

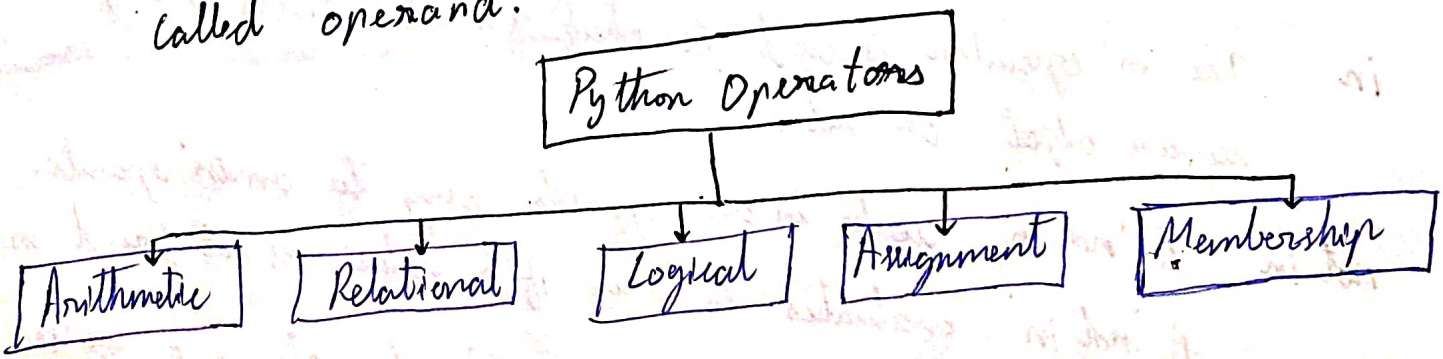
Example :-
```
a = 10  # int
b = 2.5 # float
a * b
```
output 25.0

Explicit Conversion :- Conversion that is user-defined that forces an expression to be of specific data type.

Example :-
```
a = "10"
b = 5
int(a) + b
```
output 15

Operators in Python

Operators are special symbols in python that carry out computations. The value that the operator operates on is called operand.



Arithmetic Operators :- Used to perform mathematical operations

List of Arithmetic operators :-

+	Addition	%	Modulus (Returns remainder)
-	Subtraction	//	Floor Division (Returns integer part)
*	Multiplication	**	Exponential
/	Division		

- Relational Operator :- (Comparison operator)

==	Equal To
!=	Not equal to
>	greater than
<	less than
>=	greater than (or) equal to
<=	less than (or) equal to

Logical Operators :-

and	Returns true if both the statements are true
or	Returns true if one of the statements is true
not	Reverse the result, returns false if the result is true.

Membership Operators :-

in	The 'in' operator is used to check if a value exists in any sequence object or not.
not in	A 'not in' works in an opposite way to an 'in' operator. A 'not in' evaluates to True if a value is not found in the specified sequence object. Else it returns a false.

Assignment Operators :-

The assignment operators are used to store data in a variable.

=	c = a + b	assigns value of a+b into c
+=	c += a	is equivalent to c = c + a
-=	c -= a	meaning c = c - a
*=	c *= a	→ c = c * a
/=	c /= a	→ c = c / a
%=	c %= a	→ c = c % a
**=	c **= a	→ c = c ** a
//=	c //= a	→ c = c // a

Identity Operators :-

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location.

- **is** Returns true if both variables are the same object
- **is not** Returns True if both the variables are not the same object

Python Bitwise Operators

&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of the two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill Left shift	Shift left by pushing zeroes in from the right and let the leftmost bit fall off
>>	Signed right Shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off.

Concepts behind Bitwise Operators :-

8	4	2	1
2^3	2^2	2^1	2^0

Example :-

① Complement (or) tilde (~) operator

Lets take ~12

output −13

Complement :- It will simply do the reverse of the binary format.

12 → Binary format is 0 0 0 0 1 1 0 0

Reverse the format → 1 1 1 1 0 0 1 1 (~12)

We store only positive numbers in the system. In order to store negative numbers, we can do it with 2's complement.

Formula → 2's Complement = 1's Complement + 1

Binary format of +13 → 00001101

1's Complement of 13 → 11110010

+ 0 0 0 0 0 0 0 1 [+1 to find 2's Complement]

11110011 (Same like ~12)

~ is useful whenever we want to store negative number.

AND

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

OR

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

0 0 → 0
0 1 → 1
1 0 → 1
1 1 → 0

NOT

0 → 1
1 → 0

Examples:-

→ 12 & 13

output 12

12's format 00001100

13's format 00001101

Performing & when we do OR — 00001100 (we get 12)

00001101 (we get 13)

Performing XOR — 00000001 (we get 1)

Example 10 << 2 output is 40 (Lets see how?)

Assume 10 is 00001010 (move two places left) → 00101000 (40) Left shift

(output is 40).

---

10 >> 2 (we will be loosing two bits). Right shift

(we get the output 2)

1010 → we will be loosing 2 spaces

In left shift we will be gaining bits while in right shift we will be loosing bits

8	4	2	1
2^3	2^2	2^1	2^0

Slicing and Indexing

- List elements can be accessed by index
- Individual elements in a list can be accessed using an index in square brackets.
- List indexing is zero-based.

```
a = ['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
     0      1      2        3         4      5
    -6     -5     -4       -3        -2     -1
```

We can access the objects inside a list using :-

a[0] = 'spam' , a[3] = 'tomato' etc.

a[6] = error (list index is out of range)

a[-1] = 'lobster', a[-4] = 'bacon' etc.

Slicing is indexing syntax that extracts a portion from a list

If a is a list a[m:n] returns the portion of a

- starting with position m
- And up to but not including position n

Example :-

```
a = [3, 8, 9, 3, 4, 6, 0]
    0  1  2  3  4  5  6
   -7 -6 -5 -4 -3 -2 -1
```

a[2:5]

output :- [9, 3, 4]

a[-5:-2]

output :- [9, 3, 4]

- Omitting the first index a[:n] starts the slice at the beginning of the list.
- Omitting the last index a[m:] extends the slice from the first index m to the end of the list.
- Omitting both indexes a[:] returns a copy of the entire list

- Adding an additional : and a third index designates a stride (also called step index)
- Reversing a string a [: : -1]

[m : n : 0] (Slicing)

start stop step

0	1	2	3	4	5
P	Y	T	H	O	N
-6	-5	-4	-3	-2	-1

Length of string = 6

Conditional statements :-

. if - statement

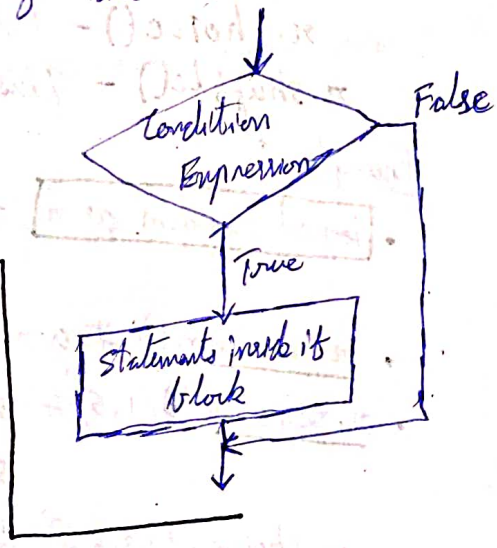
→ Used in Python for decision making

→ It is written by using the if keyword.

→ The if keyword is followed by condition later followed by indented block of code which will be executed only if the condition is true.

→ Syntax -

```
if <condition expression>:
    # code to be executed
```



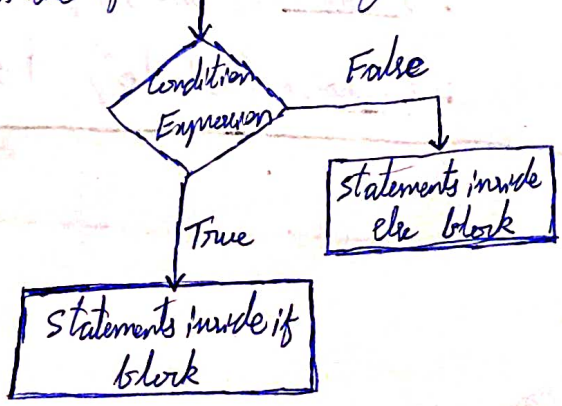
. if .... else - statement

→ The 'if ... else' statement evaluates a test expression and will execute :

- the indented if block of code if the condition is True.
- the indented else block of code if the condition in the if statement is false.

→ Syntax -

```
if <condition expression>:
    # code to be executed
else:
    # code to be executed
```



Nested if... else statement :-

- Nesting means using an if statement within another if or else statement.
- If the first 'if' condition is satisfied then the program executes the

One liner if.... else -statement :-

Syntax - <statement if true> if <Condition expression> else <statement if False>

---

random module in Python

→ Generates a random value

→ Importing random module -

```
import random as rn
```

Frequently used funtions :-

rn.random() - Generates a random float number between 0.0 to 1.0
rn.randint() - Returns a random integer between the specified integers
rn.randrange() - Returns a randomly selected element from the range created by the start, stop and step arguments.
rn.choice() - Returns a randomly selected element from a non-empty sequence
rn.shuffle() - This funtion randomly reorders the elements in a list

Examples :-

- `import random as rn` # as is an operator - used to assign a variable to a module while importing.
- rn.random() # generates a random value between 0 and 1. [0.34482] (random)
- rn.randint(1,5) # generates a random integer between mentioned start and stop values. [output = 2].
- rn.choice([1,2,3,4]) # generates a random value from the iterable or sequence output = 3
- rn.choice(["abc","pqr","xyz"]) | output → 'abc'
- rn.choice("abcde") | output → 'e'.

Nested if... else statement :-

- Nesting means using an if statement within another if or else statement.
- If the first 'if' condition is satisfied then the program executes the commands within that 'if'.

Syntax :-

```
if <condition expression>
    # code to be executed
    if <condition expression>:
        # code to be executed
    else:
        # code to be executed
else:
    # code to be executed
```

if... elif... else - statement :-

- elif statement is used to control multiple conditions only if the given if condition is false.
- It is also called else-if ladder.
- Syntax :-

```
if <condition expression>:
    #code to be executed
elif <condition expression>:
    #code to be executed
elif <condition expression>:
    #code to be executed
else:
    #code to be executed
```

- We can concatenate two strings using '+' operator.
- We can use '\n' for different lines.

Example :- Print
*
**
***

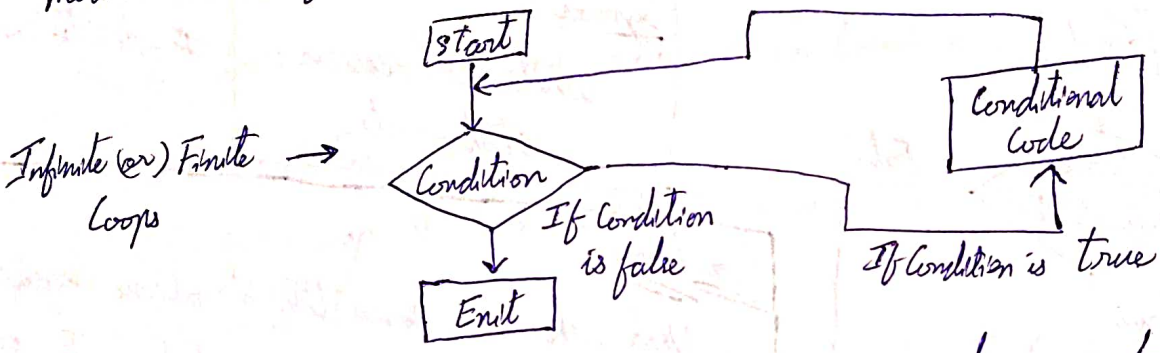
```
a = '  *  '
b = '  ** '
c = '  *** '
print(a, b, c, sep='\n')
```

- **Palindrome** :- A string is said to be a palindrome if the reverse of the string is same as the string.
  Example :- radar, civic, madam, level etc.

Fibonacci Series :- A Fibonacci sequence is the integer sequence of 0, 1, 1, 2, 3, 5, 8.....
The first two terms are 0 and 1. All other terms are obtained by adding the preceding two terms. This means to say that the nth term is the sum of (n-1)th term and (n-2)th term.

Loops in Python:-

- Loops allow us to execute a statement or a group of statements multiple times.
- In order to enter the loop there are certain conditions defined in the beginning.
- Once the condition becomes false the loop stops and the control moves out of the loop.



In an infinite loop the condition never becomes false and it keeps on executing whereas in an infinite loop, once the condition is false the loop stops executing.

Post Test Loops and Pre Test Loops:-

In post test loop, the control will first enter the loop and then it will check the condition. but in pre test, the control will enter the loop only when the condition is true.

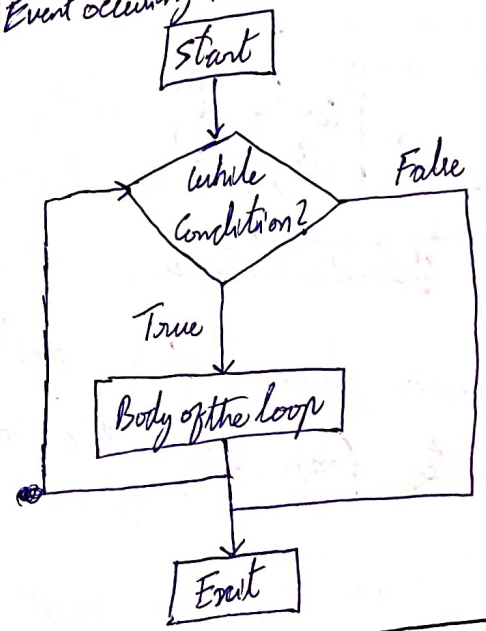
Condition is checked in the beginning of the loop in pre test whereas condition is checked at the end of the loop in post test.

There are only pre-test loops in python.

While Loop and For Loop :-

While Loop :-

Used when we don't know how many iterations are required. While loops are known as indefinite or conditional loops. They will keep on iterating until certain conditions are met. There is no guarantee ahead of time regarding how many times the loops will iterate. Event occuring inside the loop determines the number of iterations.



Syntax :-

```
while expression :
    statements
```

```
while < condition statement >:
    # code to be executed
```

- Counter driven loop

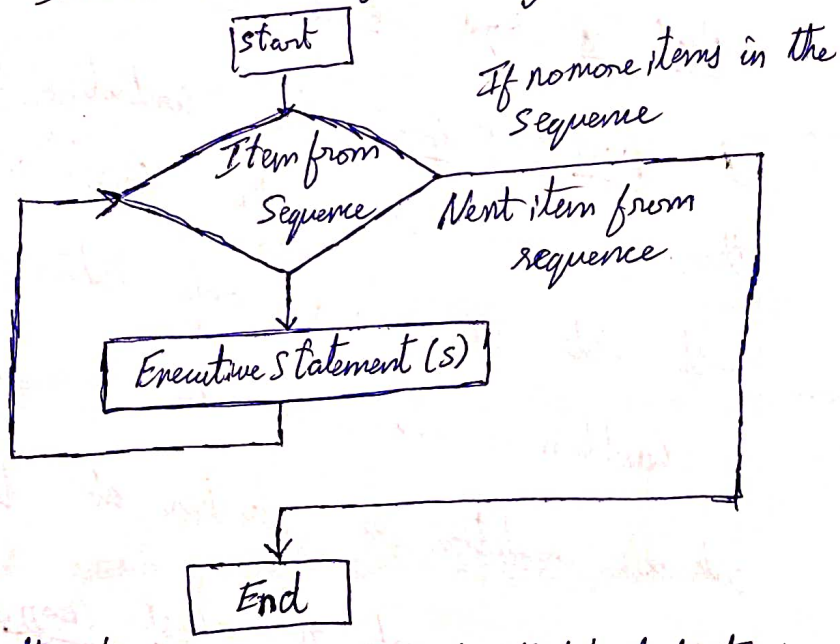
For Loops :-

Used when we know the iterations required. For loops repeats a group of statements a specified number of times. Inside syntax :-

→ Boolean Condition
→ The initial value of Counting variable
→ Incrementation of Counting variable

Syntax :-

```
for <variable> in <range>:
    stmt 1
    stmt 2
    ...
    stmt n
```




```
for <var> in <sequence>:
    # code to be executed
```

- The 'for loop' in Python is used to iterate over the items of a sequence object like list, tuple, string and other iterable objects. The iteration continues until we reach the last item in the sequence object.

Nested Loops:- Loop inside another loop.

Syntax:
```
for iterating_var in sequence:
    for iterating_var in sequence:
        statements
    statements
```

Syntax:
```
while expression:
    while expression:
        statements
    statements
```

for... else loop:-

- The 'for loop' in python comes with a combination of else block.
- The rule states that:-
  - If for loop is terminated abruptly (using break) else block is not executed.
  - If for loop is successfully executed (till the last element in the sequence) else block is executed.

Syntax:-

```
for <var> in <sequence>:
    # code to be executed
else:
    # Code in this block executes on
    successful Completion of for statement
```

Continue statement:- ignores all the remaining statements in the iteration of the current loop and moves the control back to the beginning of the loop. Can be used in both 'while' loop and 'for' loop. It is always used with conditional statements.

Break statement:- ends the loop and resumes execution at the next statement. Can be used in both 'while' loop and 'for' loop.
It is always used with conditional statements.

Sequence:- Collection of arbitrary number of elements

Types of Sequence:-

Ordered Sequence:-
- Stores elements in indexed fashion eg:- strings, lists, tuples
- Operations like indexing and slicing are applicable to ordered sequences.

Unordered Sequence:-
Stores elements randomly eg:- Sets, Dictionary

Sequences generated by Functions:-
- range(), reversed(), zip() etc

Operations On Sequences:-
Iteration
Membership
Concatenation - applicable to ordered sequences
Repetition - applicable to ordered sequences.

Generic Python Built-in functions on Sequences:-
len() - returns the number of elements in a sequence
min() - returns the smallest element from the sequence.
max() - " " largest " " " " "
sum() - sum of all elements in the sequence (must be int type)
sorted() - returns a list object of all elements from the sequence in sorted order.
reversed() - returns a reversed object by reversing the elements of sequence.

- Strings is immutable
- Tuple is immutable
- Lists are mutable (we can change the values inside list)
- Sets and Dictionaries are mutable.
- Sets never contain duplicate elements.

Dictionaries :- Python dictionary is an unordered collection of elements. It maps keys to values and these key-value pairs provide a useful way to store data in python.

Syntax :-

```
dictionary_name = {key-1 : value-1, key-2 : value-2, key-3 : value-3, }
```

4th session by Shambavi Shukla :-

Shambavi Shukla
- Sr Data Scientist
9 years in predictive modelling
9588432296
shambhavi@greatlearning.in

- import math
  math.factorial() (To find the factorial)
  ~~access~~ accession operator (gives access to data or library)

- 5+7j
  abc.conjugate()
  5-7j abc = 5+7j
  id(abc) → To find the memory location

abc.real
5.0
abc.imag
7.0

Print formating Manual

- fstrings example :-
  data = {'fruits' : 'apples'}
  print(f"The most popular fruit in winter is {data['fruits']}")
- Hopping function will happen only in for loop.
- Important Basic Programs :-

Fibonacci Series
Factorial Program
LCM and HCF
GCD
Prime numbers
swapping numbers
Palindrome (numerical and alphabets)
Anagrams

- Data scientist → lead data scientist → Sr. Data Scientist, Chief Data Scientist.

Normally, they ask features of Python in the Interview:-

Dedicated Libraries, Packages, Tools and Toolkits

- In f string {} ← These are place holders.

List Comprehensions in Python :-

Example :-

- numbers = [0, 1, 2, 3, 4]

```
doubled_numbers = [ ]

for number in numbers:
    doubled_numbers.append(number * 2)
print(doubled_numbers)
    # Apart from the above code we can use list comprehensions
doubled_numbers = [number * 2 for number in numbers]
print(doubled_numbers)
```

- friend_ages = [22, 31, 35, 37]

```
age_strings = [f"My friend is {age} years old." for age in friend_ages]
print(age_strings)
```

- (Remaining Programs in Jupyter Lab Notebook)

Functions :-

- print, len, zip are functions
- def (define)
- Lambda functions (examples in jupyter notebook)