Unit 2: Introduction to Python

- *Basics of Python programming, Python interpreter - interactive and script mode, the structure of a program, indentation, identifiers, keywords, constants, variables, types of operators, precedence of operators, data types, mutable and immutable data types, statements, expressions, evaluation and comments, input and output statements, data type conversion, debugging.*
- *Control Statements: if-else, for loop*
- *Lists: list operations - creating, initialising, traversing and manipulating lists, list methods and built-in functions.*
- *Dictionary: concept of key-value pair, creating, initialising, traversing, updating and deleting elements, dictionary methods and built-in functions.*

2.1 Python Fundamentals

1. What is Python:- Python is a general-purpose, high level programming language. It was created by Guido van Rossum, and released in 1991.
·In order to communicate with a computer system, we need a computer programming language. This language may be C, C++, Java, Python or any other computer language. Here we will discuss Python.
·Python is inspired by two languages – (i) ABC language which was an optional language of BASIC language. (ii)Modula-3

2. Why Python:-
- Platform Independent (Cross-Platform)
- Free and Open Source
- simple syntax similar to the English language.
- fewer lines than some other programming languages. interpreted Language.
- Python can be treated in a procedural way, an object-orientated way. (OOL and POL.)

3. What can Python do?:- It is possible to develop various Apps/ Software's with Python like–
- Web development
- Machine learning
- Data Analysis
- Scripting
- Game development.
- Embedded applications
- Desktop applications

4. How To Use Python?:-
4.1 Python can be downloaded from www.python.org. (Standard Installation)
4.2. It is available in two versions- • Python 2.x • Python 3.x (It is in Syllabus)
OR
Apart from above standard Python. We have various Python IDEs and Code Editors. Some of them are as under:-
(i) Anaconda Distribution:- free and open-source distribution of the Python. having various inbuilt libraries and tools like jupyter Notebook, Spyder etc

(ii) PyCharm	(iii) Canopy
(iv) Thonny	(v) Visual Studio Code
(vi) Eclipse + PyDev	(vii) Sublime Text
(viii)Atom	(ix) GNU Emacs
(x) Vim	(xi) Spyder and many more...

5. Python Interpreter - Interactive And Script Mode :-
We can work in Python in Two Ways:-
  (i) Interactive Mode
  (ii) Script Mode

(i) Interactive Mode:- it works like a command interpreter as shell prompt works in DOS Prompt or Linux. On each (>>>) symbol we can execute one by one command.
(ii) Script Mode:- it used to execute the multiple instruction (complete program) at once.

6. Python Character Set:-
Character Set is a group of letters or signs which are specific to a language. Character set includes letter, sign, number and symbol.

- Letters: A-Z, a-z
- Digits: 0-9
- Special Symbols: _, +, -, *, /, (, #,@, {, } etc.
- White Spaces: blank space, tab, carriage return, newline, formfeed etc.
- Other characters: Python can process all characters of ASCII and UNICODE.

7. Python Tokens:-
Token is the smallest unit of any programming language. It is also known as Lexical Unit. Types of token are
i. Keywords
ii. Identifiers (Names)
iii. Literals
iv. Operators
v. Punctuators

7.1 Keywords:- Keywords are reserved words. Each keyword has a specific meaning to the Python interpreter. As Python is case sensitive, these cannot be used as identifiers, variable name or any other purpose keywords must be written exactly as given below:-

7.2 Identifiers:- In programming languages, identifiers are names used to identify a variable, function, or other entities in a program. The rules for naming an identifier in Python are as follows:

- The name should begin with an uppercase or a lowercase alphabet or an underscore sign (_).
- This may be followed by any combination of characters a-z, A-Z, 0-9 or underscore (_). Thus, an identifier cannot start with a digit.
- It can be of any length. (However, it is preferred to keep it short and meaningful).
- It should not be a keyword or reserved word.
- We cannot use special symbols like !, @, #, $, %, etc. in identifiers.

Legal Identifier Names Example: myvar my_var _my_var myVar myvar2
Illegal Identifier Names Example:- 2myvar my-var my var=

7.3 Literals/ Values:- Literals are often called Constant Values. Python permits following types of literals -
  a. String literals - "Rishaan"
  b. Numeric literals – 10, 13.5, 3+5i
  c. Boolean literals – True or False
  d. Special Literal None
  e. Literal collections

7.4 Operators:- An operator is used to perform specific mathematical or logical operations on values. The values that the operator works on are called operands. Python supports following types of operators -

Unary Operator

- •Unary plus (+)
- •Unary Minus (-)
- •Bitwise complement (~)
- •Logical Negation (not)

Binary Operator

- •Arithmetic operator (+, -, *, /, %, **, //)
- •Relational Operator(<, >, <=, >=, ==,!)
- •Logical Operator (and, or)
- •Assignment Operator (=, /=, +=, -=, *=, %=, **=, //=)
- •Bitwise Operator (& bitwise and, ^ bitwise xor,|bitwise or)
- •Shift operator (<< shift left, >> shift right)
- •Identity Operator (is, is not)
- •Membership Operator (in, not in)

7.5 Punctuators:-

Punctuators are symbols that are used in programming languages to organize sentence structure, and indicate the rhythm and emphasis of expressions, statements, and program structure. Common punctuators are: „ " # $ @ []{}=:;(),

8. Variable:-

- Variables are containers for storing data values.
- Unlike other programming languages, Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

Example:-

```
x 5
y "John"
print(x)
print(y)
```

· Variables do not need to be declared with any particular type and can even change type after they have been set.

```
x 4 # x is of type int
x "Sally" # x is now of type str
print(x)
```

· String variables can be declared either by using single or double quotes:

```
x "John"
# is the same as
x 'John'
```

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the
- underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Output Variable:-

The Python print statement is often used to output variables. To combine both text and a variable, Python uses the , character:

```
x "awesome"
print("Python is " , x)
```

Display Multiple variable:-

```
x "awesome“
y=56
print(“Value of x and y is=" ,x,y)
```

9. Data Type:-
Every value belongs to a specific data type in Python. Data type identifies the type of data which a variable can hold and the operations that can be performed on those data.

Data Types in Python:-

· Numbers :- int, float and complex.
· List :- [5, 3.4, "New Delhi", "20C", 45]
· Tuple :- (10, 20, "Apple", 3.4, 'a')
· Strings :- str 'Hello Friend'
· Set:- {2,6,9}
· Dictionary:- {'Fruit':'Apple', 'Climate':'Cold', 'Price(kg)':120}

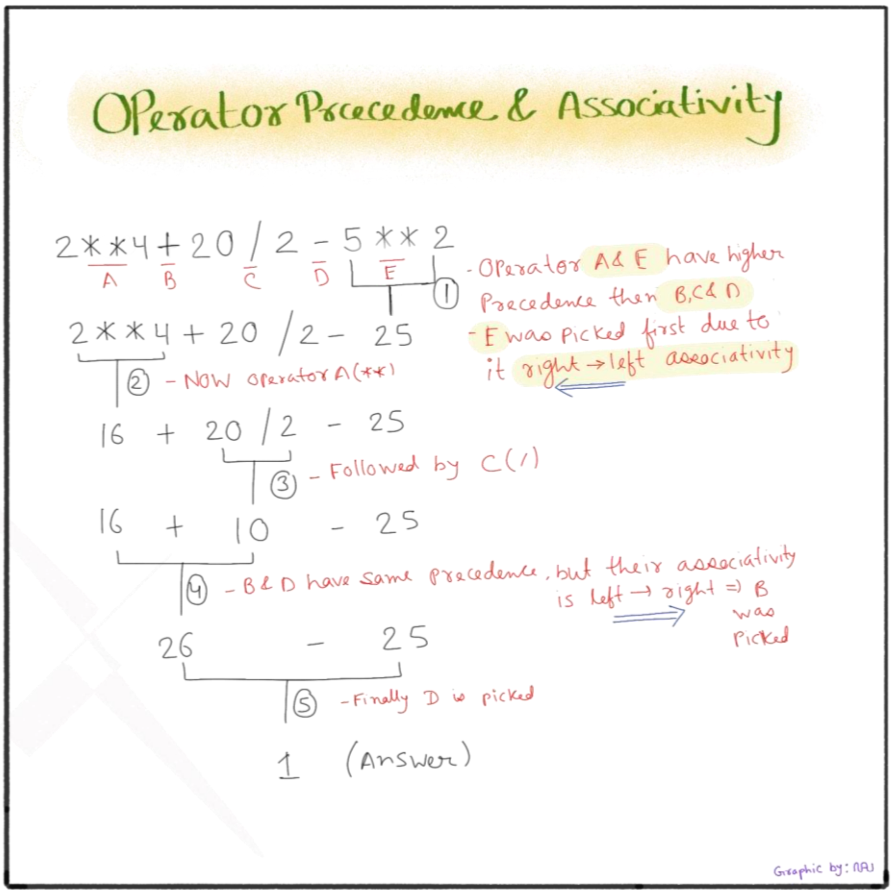
Mutable and Immutable data type
Mutable:- The values stored can be changes, Size of object can be changed, e.g. List, Dictionary
Immutable:- The value stored can not be changed, Size can not be changed, e.g. Tuple, String

10. Expression in python:- An expression is a combination of values, i.e., constant , variable and operator.
e.g. Expression 6-3*2+7-1 evaluated as 6

11. Precedence and associativity of Arithmetic Operators:- Precedence and associativity helps to evaluate an expression. Let’s take a look with following example:

Operator	**Description**	**Associativity**
( )	Parentheses	left-to-right
**	Exponent	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right

== !=	Relational is equal to/is not equal to	left-to-right
is, is not in, not in	Identity Membership operators	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
not	Logical NOT	right-to-left
and	Logical AND	left-to-right
or	Logical OR	left-to-right
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left


Operator Precedence & Associativity
2**4+20/2-5**2
A B C D E
① - Operator A & E have higher Precedence then B,C & D
- E was Picked first due to it right→left associativity
2**4+20/2- 25
② - NOW operator A(**)
16 + 20/2 - 25
③ - Followed by C(/)
16 + 10 - 25
④ - B & D have same precedence, but their associativity is left → right ⇒ B was Picked
26 - 25
⑤ - Finally D is picked
1 (Answer)
Graphic by: NAJ

12. Comments in python:- Comments are non-executable statements of python. It increases the readability and understandability of code.
Types of comment –

i. Single line comment (#) – comments only on a single line.
e.g. *a=7 # 7 is assigned to variable 'a'*
print(a) # displaying the value stored in 'a'
ii. Multi-line comment ('''..........''') – Comments multiple line.
e.g. *'''Program -1*
A program in python to store a value in
variable 'a' and display the value stored in it.'''
a=7
print(a)

13. Input and output in python:-

input ( ) method - It is used to take input from the user.

print ( ) method – It is used to display messages or result in output.

Example:-
Name input ("Enter your name : ")
Marks int(input ("Enter your marks "))
print("Your name is ", Name)
print("You got ", Marks, " marks")

14. Debugging:- Debugging is a process of locating and removing errors from a program.
Types of errors:
a. Compile Time Error: (i) Syntax Error (ii) Semantics Error
b. Run Time Error
c. Logical Error

a. Compile Time Error: Compile time errors are those errors that occur at the time of compilation of the program. There are two types of compile time errors:
i. Syntax Error: These errors occur due to violation of grammatical rules of a programming language.
Example:
```
x=10
y=20
if(x<y):
  Print("a is greater than b")     # Syntax error, P is capital in Print statement
```

b. Semantics Error: Semantic errors occur when the statements written in the program are not meaningful.
Example:
```
a=10
b=20
a+b = c  # expression cannot come on the left side of the assignment operator
```

b. Run Time Error: Errors that occur during the execution or running the program are known as run time errors. When a program "crashes" or "abnormally terminated" during the execution, then these type errors are run time errors.

Example: When a loop executes infinite time.

```
a=1
while a<5:      # value of a is always less than 5, infinite loop
   print(a)
   a=a-1
```

c. Logical Error: These types of errors occur due to wrong logic. When a program successfully executes but giving wrong output, then it is known as logical error.

Example:

```
a=10
b=20
c=30
average=(a+b+c)/2          # wrong logic to find the average
print(average)
```

OUTPUT:

```
30.0
```

Steps to debug a program:

a. Carefully follow the syntax of a language
b. Spot the origin of error
c. Read the type of error and understand it well
d. Use intermediate print statements to know the value and type of a variable
e. Trace the code carefully

Note: There are debuggers available in almost all sophisticated python IDEs. They are implemented with pdb. To know more, read the official documentation here: https://docs.python.org/3/library/pdb.html

15. Supplementary Reading:

Exception handling in python: (To deal with run time errors/exceptions)

Exception: The unusual and unexpected condition other than syntax or logical errors, encountered by a program at the time of execution, is called exception.

The purpose of the exception handling mechanism is to provide means to detect and report an exceptional circumstance, so that appropriate action can be taken.
Exception handling in python can be done using try and except blocks.

Syntax:

```
try:
   # Code that may generate an exception
except:
   # Code for handling the exception
```

Example:

```
num1=int(input("Enter first number :"))
num2=int(input("Enter second number: "))
try:
    r=num1/num2
    print("Result is :", r)
except:
    print("Divided by zero")
```

OUTPUT:

```
Enter first number : 30
Enter second number: 0
Divided by zero
```

2.2 Python Control Statements

In any programming language a program may execute sequentially, selectively or iteratively. Every programming language provides constructs to support Sequence, Selection and Iteration. In Python all these constructs can broadly be categorised in 2 categories.

Conditional Control Construct
(Selection, iteration)

Unconditional Control Construct
(pass, break,continue, exit(), quit())

Python have following types of control statements
1. Selection (branching) Statement
2. Iteration (looping) Statement
3. Jumping (break / continue)Statement

Python Selection Statements

Python have following types of selection statements
1. if statement
2. if else statement
3. Ladder if else statement (if-elif-else)
4. Nested if statement

If statements

This construct of a python program consists of one if condition with one block of statements. When the condition becomes true then executes the block given below it.
Syntax:

```
if ( condition):
        Statement(s)
```

Example:

```
age=int(input("Enter Age: "))
if ( age>=18):
  print("You are eligible for vote")
if(age<0):
        print("You entered Negative Number")
```

if - else statements

This construct of a python program consists of one if condition with two blocks. When the condition becomes true then executes the block given below it. If the condition evaluates the result as false, it will execute the block given below else.
Syntax:

```
if ( condition):
        Statement(s)
else:
        Statement(s)
```

Example

```
age=int(input("Enter Age: "))
if ( age>=18):
        print("You are eligible for vote")
 else:
        print("You are not eligible for vote")
```

Ladder if else statements (if-elif-else)

This construct of a python program consists of more than one if condition. When the first condition evaluates the result as true then executes the block given below it. If the condition evaluates the result as false, it transfers the control at the else part to test another condition. So, it is a multi-decision making construct.

Syntax:

```
if ( condition-1):
        Statement(s)
elif (condition-2):
        Statement(s)
 elif (condition-3):
        Statement(s)
else:
        Statement(s)
```

Example:

```
num=int(input("Enter Number: "))
 If ( num>=0):
        print("You entered positive number")
elif ( num<0):
        print("You entered Negative number")
else:
        print("You entered Zero ")
```

Nested if statements

It is the construct where one if condition takes part inside of another if condition. This construct consists of more than one if condition. Block executes when the condition becomes false and the next condition evaluates when the first condition becomes true.
So, it is also a multi-decision making construct.

Syntax:

```
 if ( condition-1):
        if (condition-2):
            Statement(s)
        else:
        Statement(s)
```

Example:

```
num=int(input("Enter Number: "))
if ( num<=0):
        if ( num<0):
        print("You entered Negative number")
        else:
        print("You entered Zero ")
else:
        print("You entered Positive number")
```

Python Iteration Statements

The iteration (Looping) constructs mean to execute the block of statements again and again depending upon the result of condition. This repetition of statements continues till condition meets True result. As soon as the condition meets a false result, the iteration stops.

Python supports following types of iteration statements

1. while
2. for

Four Essential parts of Looping:

i. Initialization of control variable
ii. Condition testing with control variable
iii. Body of loop Construct
iv. Increment / decrement in control variable

Python while loop

The while loop is a conditional construct that executes a block of statements again and again till the given condition remains true. Whenever the condition meets result false then loop will terminate.

Syntax:

```
Initialization of control variable
while (condition):
…………………..
Updation in control variable
…………………
```

Example: Sum of 1 to 10 numbers.

```
num=1
sum=0
while(num<=10):
        sum + num
        num + 1
print("The Sum of 1- 10 numbers: ",sum)
```

Python range( ) Function

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Syntax:

range(**start** value, **stop** value, **step** value)

Where all 3 parameters are of integer type

- Start value is Lower Limit
- Stop value is Upper Limit
- Step value is Increment / Decrement

Note: The Lower Limit is included but Upper Limit is not included in result.

Example

```
range(5)        => sequence of 0,1,2,3,4
range(2,5)      => sequence of 2,3,4
range(1,10,2)   => sequence of 1,3,5,7,9
range(5,0,-1)   => sequence of 5,4,3,2,1
range(0,-5)     => sequence of [ ] blank list
                   (default Step is +1)
range(0,-5,-1)  => sequence of 0, -1, -2, -3, -4
range(-5,0,1)   => sequence of -5, -4, -3, -2, -1
range(-5,1,1)   => sequence of -5, -4, -3, -2, -1, 0
```

Python for loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a string etc.) With for loop we can execute a set of statements, and for loop can also execute once for each element in a list, tuple, set etc.

Example: print 1 to 10 numbers

```
for num in range(1,11,1):
        print(num, end=" ")
```

Output: 1 2 3 4 5 6 7 8 9 10

Un- Conditional Control Construct

(pass, break, continue, exit(), quit())

pass Statement (Empty Statement)

The pass statement does nothing, but it is used to complete the syntax of programming concepts. Pass is useful in the situation where the user does not require any action but syntax requires a statement. The Python compiler encounters a pass statement then it does nothing but transfer the control in flow of execution.

Example

```
a=int(input("Enter first Number: "))
b=int(input("Enter Second Number: "))
 if(b==0):
        pass
else:
  print("a/b=",a/b)
```

Jumping Statements

- break
- continue

break Statement

The jump- break statement enables you to skip over a part of code that is used in a loop even if the loop condition remains true. It terminates to that loop in which it lies. The execution continues from the statement which finds the loop terminated by break.

Continue Statement

Continue statement is also a jump statement. With the help of a continue statement, some of the statements in the loop skipped over and started the next iteration. It forcefully stop the current iteration and transfers the flow of control at the loop controlling condition.