UNIT 1
INTRODUCTION TO PYTHON

Introduction

- Python is a widely used general-purpose, Interpreted, object oriented, scripting and high-level programming language.
- It was initially designed by Guido van Rossum (Dutch programmer) in 1991
- It was mainly developed to emphasize code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Features of python

1. Simple Syntax

Python has a very simple and clean syntax. It is similar to English, which makes it easy to read and understand. This helps beginners to start coding without much difficulty.

2. Easy to Learn

Python is one of the easiest programming languages to learn. It does not have complicated rules like other languages (e.g., C or Java). Even students with no programming background can learn it quickly.

3. Free and Open Source

Python is completely free to download and use. It is open source, which means its source code is freely available for anyone to view, modify, and improve. This helps the Python community to grow rapidly.

4. High-Level Language

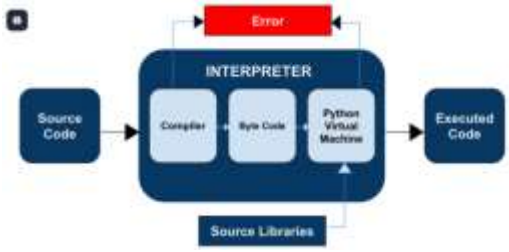
Python is a high-level programming language. This means programmers do not need to worry about low-level details like memory management or hardware operations. The language handles all these things in the background.

5. Interpreted Language

Python is an interpreted language. It runs the code line by line, using an interpreter. This makes debugging (finding and fixing errors) easier.

Example:

If there is an error in line 5, the interpreter stops at line 5 and shows the error, instead of checking all lines first.



6. Memory Allocation

Python handles memory management automatically. It uses a process called Garbage Collection, which removes unused variables from memory and frees up space. This makes the program more efficient.

7. Portable

Python is portable, meaning you can write a Python program on one operating system (like Windows) and run it on another (like Linux or Mac) without changing the code.

8. Object-Oriented Language

Python supports Object-Oriented Programming (OOP), which allows you to create and use objects and classes. OOP concepts like inheritance, encapsulation, and polymorphism are supported in Python.

9. Procedure-Oriented Language

Python also supports Procedure-Oriented Programming (POP), where you can write functions (procedures) to perform tasks. This makes the code modular and reusable.

10. Extensible

Python is extensible. This means you can include code written in other languages like C or C++ inside your Python programs to improve performance or use existing libraries.

11. Wide Range of Built-in Functions

Python has a large number of built-in functions. These are ready-to-use and save time for the programmer. Some examples include print(), len(), type(), input(), max(), sum(), etc.

12. Robust

Python is a robust language. It has strong error-handling features. You can use try-except blocks to handle errors and prevent your program from crashing unexpectedly.

13. GUI Programming Support

Python supports the development of Graphical User Interfaces (GUIs). Libraries like Tkinter, PyQt, and Kivy help you create windows, buttons, labels, etc., to build desktop applications with ease.

Variables

Python Variable is containers that store values. We do not need to declare variables before using them or declare their type. While creating a variable we must assign a value to it. A Python variable is a name given to a memory location.

- The value stored in a variable can be changed during program execution.
- A Variables in Python is only a name given to a memory location, all the operations done on the variable effects that memory location.

Ex: Var = "Python programming"
print(Var)
output: Python programming

Rules for Python variables

- ✓ A Python variable name must start with a letter or the underscore character.
- ✓ A Python variable name cannot start with a number.
- ✓ A Python variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- ✓ Variable in Python names are case-sensitive (name, Name, and NAME are three different variables).
- ✓ The reserved words(keywords) in Python cannot be used to name the variable in Python.

- Python variables do not need to be declared with any particular type, It automatically takes the type by the value we assign to the variable and can even change type after they have been set.

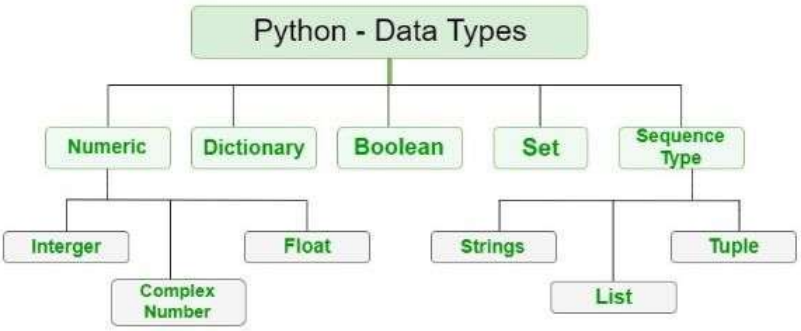
```
a= 4            # a is now type int
a= "python"     #a is now of type str
print(a)        # output as python
```

- If we use the same name for different types, the variable starts referring to a new value and type.
- Python allows assigning a single value to several variables simultaneously with "="operators.
  Ex: a = b = c = 10
- Python allows assigning different values in a single line with "," operators.
  Ex: a, b, c = 10, 20.2, "Python"
- The Python plus operator + provides a convenient way to add a value if it is a number and concatenate if it is a string and for different types would produce an error.

```
Ex: a = 10                  a = "python "
    b = 20                  b = "programming"
    print(a+b) #30          print(a+b) #python programming
```

Datatypes

- Python Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, Python data types are classes and variables are instances (objects) of these classes.
- To define the values of various data types of Python and check their data types we use the type() function.
  - Numeric
  - Sequence Type
  - Boolean
  - Set
  - Dictionary



Numeric Data Types

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number.

- Integers – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.
- Float – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point.
- Complex Numbers – A complex number is represented by a complex class. It is specified as (real part) + (imaginary part)j. For example – 2+3j

```
a = 5
print("Type of a: ", type(a))
b = 5.0
print("Type of b: ", type(b))
c = 2 + 4j
print("Type of c: ", type(c))
```

```
Type of a:  <class 'int'>
Type of b:  <class 'float'>
Type of c:  <class 'complex'>
```

Sequence Data Types

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion.

- Python String - A string is a collection of one or more characters put in a single quote, double-quote, or triple-quote.

```
a="Python"
print(a[0])
print(a[-1])
print(a)
```

```
P
n
Python
```

- Python List –
  - ✓ Lists are just like arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.
  - ✓ Lists are mutable, ordered and allows duplicates.
  - ✓ Lists in Python can be created by just placing the sequence inside the square brackets[].
  - ✓ We can create a multi-dimensional list (a list of lists)

```
b=["hi", 1989 ,4.0]
print(b)
print(b[2])
#multi-dimentional
c = [['python', 'programming'], ['language' , 1991]]
print(c)
```

```
['hi', 1989, 4.0]
4.0
[['python', 'programming'], ['language', 1991]]
```

List functions

```
# len() - returns the number of items in a list
lst = [10, 20, 30, 40, 50]
len(lst) # 5

# append() - adds an item to the end of the list
lst = [10, 20, 30, 40, 50]
lst.append(60)
lst   # [10, 20, 30, 40, 50, 60]

# insert() - inserts an item at a specified index
lst = [1, 2, 4, 5]
lst.insert(2, 3) # insert 3 at index 2
lst   # [1, 2, 3, 4, 5]
```

```
# count() - returns the number of occurrences of an item
lst = [1, 2, 2, 3, 3, 3, 4]
lst.count(3)    # 3

# index() - returns the index of the first occurrence of an item
lst = [10, 20, 30, 40, 50]
lst.index(30)   # 2

# sort() - sorts the list in ascending order by default
lst = [5, 2, 8, 1, 9]
lst.sort()
lst     # [1, 2, 5, 8, 9]
```

```
# reverse() - reverses the order of the list
lst = [1, 2, 3, 4, 5]
lst.reverse()
lst   # [5, 4, 3, 2, 1]

# pop() - removes and returns the item at a given index (or the last item if no index is specified)
lst = [100, 200, 300, 400]
lst.pop(1) # remove item at index 1
lst   # [100, 300, 400]

# pop() - removes and returns the item at a given index (or the last item if no index is specified)
lst = [10, 20, 30]
lst.pop() # remove last item
lst   #  [10, 20]
```

- Python Tuple –
  - ✓ Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by a tuple class.
  - ✓ It creates a tuple from a string using the tuple() function.

```
d=("honey", 1980, 25.34)
print(d)
print(d[0])
e=tuple("PYTHON")
print(e)
#nested tuple
f=((23, 78,4,'abc'),('xyz',9))
print(f)
```

```
('honey', 1980, 25.34)
honey
('P', 'Y', 'T', 'H', 'O', 'N')
((23, 78, 4, 'abc'), ('xyz', 9))
```

Boolean Data Type

- ✓ Python Data type with one of the two built-in values, True or False.

```
a=True
print(a)
print(type(a))
```

```
True
<class 'bool'>
```

Set Data Type

- ✓ In Python Data Types, a Set is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.
- ✓ Sets can be created by using the built-in set() function.

```
#set with string
s=set("pythonprogramming")
print(s)
#set with list
t=set([20,3,21,9,4,7,6,1])
print(t)
#set with different values
z=set(['abc',84,90.78,3.14,'xyz'])
print(z)
```

```
{'p', 'g', 'i', 'y', 'm', 'n', 'a', 'r', 'h', 'o', 't'}
{1, 3, 4, 6, 7, 9, 20, 21}
{'abc', 'xyz', 3.14, 84, 90.78}
```

Dictionary Data Type

- ✓ A dictionary in Python is an unordered collection of data values, used to store data values like a map, unlike other Python Data Types that hold only a single value as an element, a Dictionary holds a key: value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon : , whereas each key is separated by a 'comma'.
- ✓ Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable.
- ✓ The dictionary can also be created by the built-in function dict().

```
Dict = {2: 'pqr', 3: 'xyz',1: 'abc'}
print(Dict)
print(Dict[1])
dic={'a':10,'b':30,'c':20}
print(dic)
print(dic['b'])
```

```
{2: 'pqr', 3: 'xyz', 1: 'abc'}
abc
{'a': 10, 'b': 30, 'c': 20}
30
```

Python Interpreter and Interactive Mode

Python Interpreter:

The Python interpreter is a special program that reads and executes Python code. It works by translating the Python code line by line into a form that the computer understands, and then it runs that code. This is different from compiled languages like C or Java, where the whole program is converted into machine code first before running.

For example, if you write a Python program like this in a file named hello.py:

```
print("Welcome to Python")
a = 10
b = 5
print("Sum is", a + b)
```

You can run this file using the command:
python hello.py
When you run this, the Python interpreter reads each line one by one — it prints the welcome message, adds the numbers, and shows the result. If there's any error in your code, the interpreter will stop and show you the error immediately. So, it helps in both executing and debugging your program.

Python Interactive Mode:

Python also offers something called interactive mode, which is like a live environment where you can type Python commands one by one and see the results instantly. You don't have to save your code in a file.
To open the interactive mode, you just open your command prompt or terminal and type:
python
You will see this prompt:

```
>>>
```

This >>> is called the Python prompt. It means the interpreter is ready to accept commands. You can now type anything like:

```
>>> 2 + 3
5
>>> print("Hello")
Hello
>>> x = 4
>>> x * 5
20
```

This is very useful for beginners to learn Python quickly because they can test each line and understand how Python works. It is also used by programmers to check logic or try small parts of a program before writing a full script.

- The Python interpreter is what actually runs your code and helps you see results. It can be used in two ways: by running a complete program (called script mode) or by using interactive mode, which lets you type and run code one line at a time. Interactive mode is very useful for quick tests and learning, while script mode is used for writing real-world programs.

Operator precedence

- In Python, when an expression contains more than one operator, it is important to know which operator will be executed first. This is called the precedence (or priority) of operators. Python follows specific rules to decide the order in which operations are performed.

For example, consider this expression:

```
result = 10 + 5 * 2
```

- If we go left to right, we may wrongly do 10 + 5 first. But in Python, multiplication (*) has higher precedence than addition (+), so 5 * 2 is done first, giving 10. Then 10 + 10 = 20. So the final result is 20.
- This concept helps avoid confusion when multiple operations are combined in a single line.

```
value = 100 - 5 ** 2
```

- You might think 100 - 5 = 95 and then square it, but Python knows that exponentiation (**) has higher precedence, so it first calculates 5 ** 2 = 25, then 100 - 25 = 75.
- To control the order manually, we can use parentheses (). Anything inside brackets will be done first, regardless of the default precedence.

```
x = (10 + 5) * 2
```

- Here, 10 + 5 = 15 is done first, and then multiplied by 2, giving the result 30.

This is very useful in solving math expressions, logic conditions, and when writing complex formulas in code. Without knowing operator precedence, we may get wrong results even if the syntax is correct.

Operators	Associativity
() Highest precedence	Left - Right
**	Right - Left
+x , -x, ~x	Left - Right
*, /, //, %	Left - Right
+, -	Left - Right
<<, >>	Left - Right
&	Left - Right
^	Left - Right
|	Left - Right
Is, is not, in, not in, <, <=, >, >=, ==, !=	Left - Right
Not x	Left - Right
And	Left - Right
Or	Left - Right
If else	Left - Right
Lambda	Left - Right
=, +=, -=, *=, /= Lowest Precedence	Right - Left

Python Functions

Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Benefits of Using Functions

- ✓ Increase Code Readability
  - Functions help to organize code into manageable sections with meaningful names, making it easier to understand what each part of the code does.
- ✓ Increase Code Reusability
  - Functions allow you to write a piece of code once and reuse it multiple times. This avoids redundancy and reduces the amount of code you need to write and maintain.
- ✓ Modularity
  - By breaking your code into functions, you can divide your program into smaller, manageable chunks. Each function can be developed, tested, and debugged independently, which makes your code more modular and easier to manage.
- ✓ Easy code maintainance
  - Code maintenance with functions is simplified because you can update or fix a specific function without affecting the rest of the code. When a function is responsible for a particular task, changes to its implementation are localized, so you don't need to search through the entire codebase. This modular approach also makes it easier to test and debug individual parts of the code.
- ✓ Reduces the length of code
  - Functions enable you to break down complex operations into smaller, reusable pieces, which helps keep the overall codebase shorter and more organized.

There are 3 Types of Functions in python

- ➢ Built-in functions
- ➢ User defined functions
- ➢ Anonymous functions

Built-in functions

Python provides a rich set of built-in functions whose functionality is pre-defined in python.

Ex:print(), len(), type(), sum(), min(), max()

(for more read text book pg no:68)

User defined functions

We can create our own functions based on our requirements. Simple rules to define functions in python

- o Function blocks begin with the keyword def followed by the function name and

parentheses ().

- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function. A return statement with no arguments is the same as return None.

Syntax:

```
def function_name(parameters list):
    function suite or statements
    return [expression]
```

Example:

```
def simple():
    print("Hello ")
simple()
```

```
Hello
```

```
def simple():
    return "Hello"
print(simple())
```

```
Hello
```

```
def simple():
    print("Hello ")
print(simple())
```

```
Hello
None
```

Calling a python function

- The function gets executed when called.
- If the function has parameters, you need to pass arguments while calling it.
- The flow of control moves to the function definition during the call.
- After execution, the control returns to the caller.

Syntax:

```
function_name(arguments)
```

```
def add(a,b):
    return a+b
add(10,20)
```

```
30
```

Pass by Value vs Pass by Reference

In programming, when arguments are passed to a function, they can be passed either by value or by reference.

Pass by Value:

- The function receives a copy of the actual data.
- Changes made inside the function do not affect the original value.
- The original data remains unchanged.
- Used when variables are of immutable types in Python like:
  Ex: int, float, str, tuple

Key Points:

- ✓ Safe from unwanted modifications.
- ✓ Consumes more memory due to copying.

```
#call by value
def fun(a):
    a+=1
    print("Inside function ",a)
a=10
fun(a)
print("Outside function ",a)
```

```
Inside function  11
Outside function  10
```

Pass by Reference:

- The function receives the memory address of the actual variable.
- Changes made inside the function directly affect the original data.
- Used with mutable types in Python like:
  Ex: list, dict, set

Key Points:

- ✓ Efficient in terms of memory.
- ✓ Risk of modifying original data accidentally.

```
#call by reference
def fun(a):
    a[2]=100
    print("Inside ",a)
a=[10,20,30,40,50]
fun(a)
print("Outside ",a)
```

```
Inside  [10, 20, 100, 40, 50]
Outside  [10, 20, 100, 40, 50]
```

Types of Python Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following function argument types in Python:

- Default argument
- Keyword arguments (named arguments)
- Positional arguments
- Arbitrary arguments (variable-length arguments *args and **kwargs)

Default Arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments to write functions in Python.

```
def fun(x,y=10):
    print("X = ",x)
    print("Y = ",y)
fun(5)
```

```
X =  5
Y =  10
```

Keyword Arguments

The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.

```
def fun(fname,lname):
    print("first name = ",fname)
    print("last name = ",lname)
fun(lname="Van Rossum",fname="guido")
```

```
first name =  guido
last name =  Van Rossum
```

Positional Arguments

We used the Position argument during the function call so that the first argument (or value) is assigned to fist variable and the second argument is assigned to second variable, and so on.

```
def fun(name,age):
    print("I'm = ",name)
    print("My age is = ",age)
fun(20,"syam")
```

```
I'm =  20
My age is =  syam
```

Arbitrary Keyword Arguments

In Python Arbitrary Keyword Arguments, *args, and **kwargs can pass a variable number of arguments to a function using special symbols. There are two special symbols:

- ✓ *args in Python (Non-Keyword Arguments)
- ✓ **kwargs in Python (Keyword Arguments)

```
def fun(*nums):
    print(*nums)
fun(20,30,40,50)
```

```
20 30 40 50
```

```
def fun(**nums):
    print(nums)
fun(a=20,b=30,c=40,d=50)
```

```
{'a': 20, 'b': 30, 'c': 40, 'd': 50}
```

Flow of Execution

Flow of execution means the order in which Python runs the code. When a program includes functions, the flow depends on when and where the functions are defined and called.

Step-by-Step Explanation:

- Python starts from the first line of the program and moves downward, line by line.
- When Python sees a function definition using the def keyword, it does not run the function at that point.
  It simply remembers that a function exists with a specific name.
- The function will be executed only when it is called.
- When a function is called:
  Python jumps into the function definition.
  It executes all the lines inside the function, in order.
- After the function finishes:
  If there is a return statement, it sends a value back.
  Then Python comes back to the place where the function was called and continues from there.

```
def greet():
    print("Hello!")
    print("Welcome to Python.")

print("Start of program")
greet()
print("End of program")
```

```
Start of program
Hello!
Welcome to Python.
End of program
```

Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.
- Comments starts with a #, and Python will ignore them

```
print("Hello, World!") #This is a comment
```

```
Hello, World!
```

Multiline Comment:

- Python does not really have a syntax for multiline comments.
- To add a multiline comment you could insert a # for each line:
- Example:

  #This is a comment
  #written in
  #more than just one line

- Or, you can use a multiline string. Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it.

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

```
Hello, World!
```

Type Conversion

Python defines type conversion functions to directly convert one data type to another which is useful in day-to-day and competitive programming.

There are two types of Type Conversion in Python:

Python Implicit Type Conversion

Python Explicit Type Conversion

Implicit Type Conversion

In Implicit type conversion of data types in Python, the Python interpreter automatically converts one data type to another without any user involvement. Python prevents Implicit Type Conversion from losing data.

```
a=10
b=20.45
c=a+b
print(type(a),type(b))
print(c)
print(type(c))
```

```
<class 'int'> <class 'float'>
30.45
<class 'float'>
```

Explicit Type Conversion

In this conversion the data type is manually changed by the user as per their requirement. With explicit type conversion, there is a risk of data loss since we are forcing an expression to be changed in some specific data type.

- ✓ int(a, base) and int(): This function converts any data type to an integer. 'Base' specifies the base in which the string is.
- ✓ float(): This function is used to convert any data type to a floating-point number

```
binr='1100'
a=23.49
print(int(binr,2)) #12
print(int(a))      #23
b=20
print(float(b))    #20.0
```