Python Introduction

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.
- **Python is Interpreted −** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive −** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented −** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- **Python is a Beginner's Language −** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

Following are important characteristics of **Python Programming** −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python Features

Python provides many useful features which make it popular and valuable from the other programming languages. It supports object-oriented programming, procedural programming approaches and provides dynamic memory allocation. We have listed below a few essential features.

1) Easy to Learn and Use

Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language for beginners.

2) Expressive Language

Python can perform complex tasks using a few lines of code. A simple example, the hello world program you simply type `print("Hello World")`. It will take only one line to execute, while Java or C takes multiple lines.

3) Interpreted Language

Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being interpreted language, it makes debugging easy and portable.

) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop the software for several competing platforms by writing a program only once.

5) Free and Open Source

Python is freely available for everyone. It is freely available on its official website www.python.org

. It has a large community across the world that is dedicatedly working towards make new python modules and functions. Anyone can contribute to the Python community. The open-source means, "Anyone can download its source code without paying any penny."

6) Object-Oriented Language

Python supports object-oriented language and concepts of classes and objects come into existence. It supports inheritance, polymorphism, and encapsulation, etc. The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into byte code, and any platform can use that byte code.

8) Large Standard Library

It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting. There are various machine learning libraries, such as Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

9) GUI Programming Support

Graphical User Interface is used for the developing Desktop application.

It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C,C++ Java. It makes easy to debug the code.

11. Embeddable

The code of the other programming language can use in the Python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

12. Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to x, then we don't need to write int x = 15. Just write x = 15.

Magic commands or magic functions are one of the important enhancements that IPython offers compared to the standard Python shell. These magic commands are intended to solve common problems in data analysis using Python. In fact, they control the behaviour of IPython itself.

Python - Magic Commands

Types of Magic Commands

There are two types of magic commands −

- Line magics
- Cell magics

Line Magics

They are similar to command line calls. They start with % character. Rest of the line is its argument passed without parentheses or quotes. Line magics can be used as expression and their return value can be assigned to variable.

Cell Magics

They have %% character prefix. Unlike line magic functions, they can operate on multiple lines below their call. They can in fact make arbitrary modifications to the input they receive, which need not even be a valid Python code at all. They receive the whole block as a single string.

Built-in line magics

```
%autocall [mode]
```

This magic function makes a function automatically callable without having to use parentheses. It takes three possible mode parameters: **0 (off), 1 (smart) is default** or **2 (always on)**.

%automagic

Magic functions are callable without having to type the initial % if set to 1. Without arguments it toggles on/off. To deactivate, set to 0.

%cd

This line magic changes the current directory. This command automatically maintains an internal list of directories you visit during your IPython session, in the variable _dh. You can also do ‘cd -<tab>’ to see directory history conveniently.

The %cd command can be used in the following ways −

- **%cd <dir>** − Changes current working directory to <dir>
- **%cd..** − Changes current directory to parent directory
- **%cd** − changes to last visited directory.

%dhist

This magic command prints all directories you have visited in current session. Every time %cd command is used, this list is updated in **_dh** variable.

%edit

This magic command calls upon the default text editor of current operating system (Notepad for Windows) for editing a Python script. The script is executed as the editor is closed.

%env

This magic command will list all environment variables. It also reads value of particular variable or set the value of environment variable.

Usage

The %cd command can be used in the following ways −

- **%env** − Lists all environment variables
- **%env var** − Gets value for var
- **%env var val** − Sets value for var

%lsmagic

Displays all magic functions currently available

%pwd

This magic function returns the present working directory.

%pylab

This function populates current IPython session with matplotlib, and numpy libraries.

%recall

When executed without any parameter, this function executes previous command.

%run

This command runs a Python script from within IPython shell.

%time

This command displays time required by IPython environment to execute a Python expression

%who

This line magic prints all interactive variables, with some minimal formatting. If any arguments are given, only variables whose type matches one of these are printed.

Python Advantages and Disadvantages

Let’s first dive into the advantages of Python.

Advantages of Python

1. Easy to Read, Learn and Write

Python is a high-level programming language that has English-like syntax. This makes it easier to read and understand the code.
Python is really easy to pick up and learn, that is why a lot of people recommend Python to beginners. You need less lines of code to perform the same task as compared to other major languages like C/C++ and Java.

2. Improved Productivity

Python is a very productive language. Due to the simplicity of Python, developers can focus on solving the problem. They don’t need to spend too much time in understanding the syntax or behavior of the programming language. You write less code and get more things done.

3. Interpreted Language

Python is an interpreted language which means that Python directly executes the code line by line. In case of any error, it stops further execution and reports back the error which has occurred.

4. Dynamically Typed

Python doesn’t know the type of variable until we run the code. It automatically assigns the data type during execution. The programmer doesn’t need to worry about declaring variables and their data types.

5. Free and Open-Source

Python comes under the OSI approved open-source license. This makes it free to use and distribute. You can download the source code, modify it and even distribute your version of Python. This is useful for organizations that want to modify some specific behavior and use their version for development.

6. Vast Libraries Support

The standard library of Python is huge, you can find almost all the functions needed for your task. So, you don’t have to depend on external libraries.

But even if you do, a Python package manager (pip) makes things easier to import other great packages from the Python package index (PyPi). It consists of over 200,000 packages.

7. Portability

In many languages like C/C++, you need to change your code to run the program on different platforms. That is not the same with Python. You only write once and run it anywhere.
However, you should be careful not to include any system-dependent features.

Disadvantages of Python

1. Slow Speed

We discussed above that Python is an interpreted language and dynamically-typed language. The line by line execution of code often leads to slow execution.
The dynamic nature of Python is also responsible for the slow speed of Python because it has to do the extra work while executing code. So, Python is not used for purposes where speed is an important aspect of the project.

2. Not Memory Efficient

To provide simplicity to the developer, Python has to do a little tradeoff. The Python programming language uses a large amount of memory. This can be a disadvantage while building applications when we prefer memory optimization.

3. Weak in Mobile Computing

Python is generally used in server-side programming. We don’t get to see Python on the client-side or mobile applications because of the following reasons. Python is not memory efficient and it has slow processing power as compared to other languages.

4. Database Access

Programming in Python is easy and stress-free. But when we are interacting with the database, it lacks behind.

The Python’s database access layer is primitive and underdeveloped in comparison to the popular technologies like JDBC and ODBC.
Huge enterprises need smooth interaction of complex legacy data and Python is thus rarely used in enterprises.

5. Runtime Errors

As we know Python is a dynamically typed language so the data type of a variable can change anytime. A variable containing integer number may hold a string in the future, which can lead to Runtime Errors.
Therefore Python programmers need to perform thorough testing of the applications.

Data Types in Python:

Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

```
a = 53
```

The variable a holds integer value and we did not define its type. Python interpreter will automatically interpret variables a as an integer type. Python enables us to check the type of the variable used in the program. Python provides us the type() function, which returns the type of the variable passed.

```
a=50
b="Hello Python"
c = 50.5
print(type(a))
print(type(b))
print(type(c))
```

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. Numbers
2. Sequence Type
3. Boolean
4. Set
5. Dictionary

Numeric data type: Integer, Float, Complex

Sequence Type: Strings, List, Tuple

Numbers or Numeric Data Type:

Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type. Python provides the type() function to know the data-type of the variable. Similarly, the isinstance() function is used to check an object belongs to a particular class.

Python creates Number objects when a number is assigned to a variable. For example;

```
a = 52
print("The type of a", type(a))
b = 4.5
print("The type of b", type(b))
c = 10+31j
print("The type of c", type(c))
print(" c is a complex number", isinstance(10+31j,complex))
```

Numeric data types:

1. Int - Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to int
2. Float - Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.
3. complex - A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Sequence Type

String

The string can be defined as the sequence or collection of characters represented in the quotation marks. In Python, we can use single, double, or triple quotes to define a string.

Types of strings in Python:

1. Normal: “hello world”

2.Raw: A string prefixed with r or R is called a raw string. A 'raw string' is slightly different from a string where backslash \ , is just taken as a backslash, and not escape sequences to represent newlines, tabs, backspace, and so on.

Unicode: Unicode is a standard encoding system that is used to represent characters from almost all languages. Every Unicode character is encoded using a unique integer code point between 0 and 0x10FFFF . A Unicode string is a sequence of zero or more code points.

```
u = unicode(text, 'utf-8')
```

```
str1 = 'hello javatpoint' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

List

Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

Consider the following example.

```
list1 = [1, "hi", "Python", 2]

#Checking type of given list
print(type(list1))

#Printing the list1
print (list1)
```

```
# List slicing
print (list1[3:])

# List slicing
print (list1[0:2])

# List Concatenation using + operator
print (list1 + list1)

# List repetition using * operator
print (list1 * 3)
```

Output:

```
[1, 'hi', 'Python', 2]

[2]

[1, 'hi']

[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]

[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]
```

Tuple

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Let's see a simple example of the tuple.

```
tup  = ("hi", "Python", 2)

#Printing the tuple
print (tup)
```

```
# Tuple slicing
print (tup[1:])
print (tup[0:1])

# Tuple concatenation using + operator
print (tup + tup)

# Tuple repetition using * operator
print (tup * 3)

# Adding value to tup. It will throw an error.
t[2] = "hi"
```

Dictionary

Dictionary is an unordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Values in a dictionary can be of any data type and can be duplicated whereas keys cannot be repeated and must be immutable. Keys are case sensitive if two keys have same name but different cases, it will be treated distinctly.

The items in the dictionary are separated with the comma (,) and enclosed in the curly braces {}.

Consider the following example.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}
# Printing dictionary
print (d)

# Accesing value using keys
print("1st name is "+d[1])
print("2nd name is "+ d[4])
print(d.get(3))

print (d.keys())
print (d.values())
```

Output:

```
1st name is Jimmy

2nd name is mike

john

dict_keys([1, 2, 3, 4])

dict_values(['Jimmy', 'Alex', 'john', 'mike'])
```

Boolean

Boolean type provides two built-in values, True and False. These values are used to determine the given statement true or false. It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'. Consider the following example.

```
# Python program to check the boolean type

str=”Tony Mark”
print(“Mark” in str)
```

output

```
True
```

Set

Python Set is the unordered collection of the data type. It is iterable, mutable(can modify after creation), and has unique elements. In set, the order of the elements is undefined; it may return the changed sequence of the element. The set is created by using a built-in function set(), or a sequence of elements is passed in the curly braces and separated by the comma. It can contain various types of values. Consider the following example.

```
set2 = {'James', 2, 3,'Python'}

#Printing Set value
print(set2)
```

```python
# Adding element to the set

set2.add(10)
print(set2)

#Removing element from the set
set2.remove(2)
print(set2)
```

Output:

```
{3, 'Python', 'James', 2}

{'Python', 'James', 3, 2, 10}

{'Python', 'James', 3, 10}
```

Python If-else statements

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

In python, decision making is performed by the following statements.

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

Indentation in Python

For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.

The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

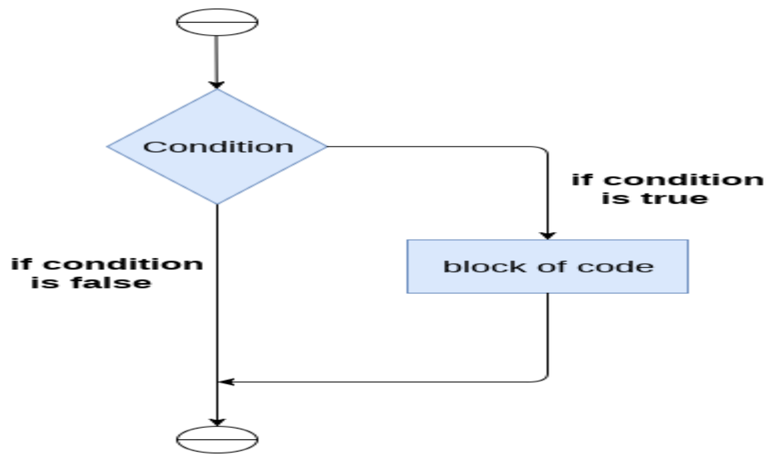


Figure - If else condition Flowchart:

The syntax of the if-statement is given bel

```
if expression:
    statement
```

Example 1

```
num = int(input("enter the number?"))

if num%2 == 0:
    print("Number is even")
```

Output:

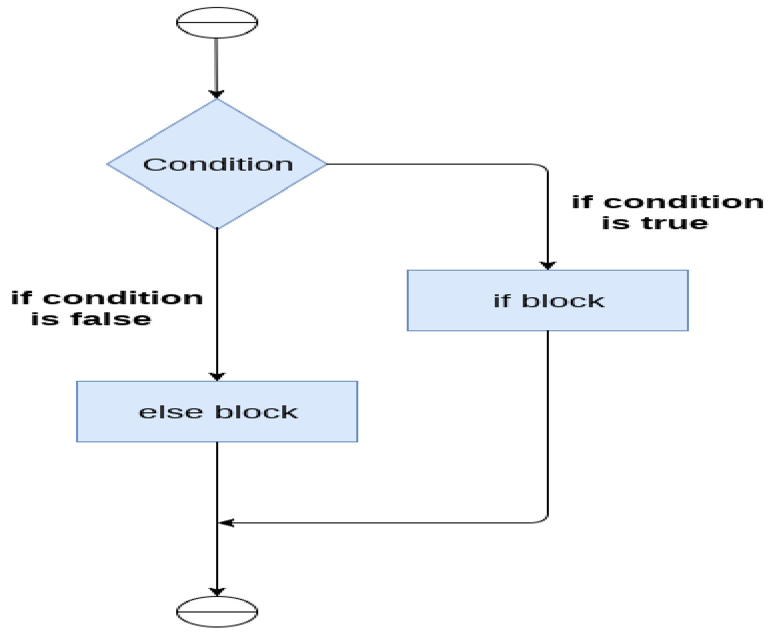
```
enter the number?10

Number is even
```

The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



The syntax of the if-else statement is given below.

```
if condition:
    #block of statements
else:
    #another block of statements (else-block)
```

Example 1 : Program to check whether a person is eligible to vote or not.

```
age = int (input("Enter your age? "))
if age>=18:
    print("You are eligible to vote !!");
else:
   print("Sorry! you have to wait !!");
```

Output:

```
Enter your age? 90

You are eligible to vote !!
```

The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

The syntax of the elif statement is given below.

```
if expression 1:
    # block of statements

elif expression 2:
    # block of statements

elif expression 3:
    # block of statements

else:
    # block of statements
```

Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list:

```
city = ["Delhi", "Mumbai", "Agra"]
for x in city:
  print(x)
```

The break Statement

With the `break` statement we can stop the loop before it has looped through all the items:

Example

Exit the loop when `x` is "banana":

```
city = ["Delhi", "Mumbai", "Agra"]
for x in city:
  print(x)

    if x== "Mumbai":
    break
```

The continue Statement

With the `continue` statement we can stop the current iteration of the loop, and continue with the next:

Example

Do not print banana:

```
city = ["Delhi", "Mumbai", "Agra"]
for x in city:

    if x == "Mumbai":
    continue
    print(x)
```

The range() Function

To loop through a set of code a specified number of times, we can use the `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.

Example

Using the range() function:

```
for x in range(6):
  print(x)
```

Else in For Loop

The `else` keyword in a `for` loop specifies a block of code to be executed when the loop is finished:

Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

Python While Loops

In coding, loops are designed to execute a specified code block repeatedly.

Introduction of Python While Loop

The Python while loop iteration of a code block is executed as long as the given condition, i.e., conditional_expression, is true.

If we don't know how many times we'll execute the iteration ahead of time, we can write an indefinite loop.

```
while conditional_expression:
    Code block of while
```

The given condition, i.e., conditional_expression, is evaluated initially in the Python while loop. Then, if the conditional expression gives a boolean value True, the while loop statements are executed. The conditional expression is verified again when the complete code block is executed. This procedure repeatedly occurs until the conditional expression returns the boolean value False.

- The statements of the Python while loop are dictated by indentation.
- The code block begins when a statement is indented & ends with the very first unindented statement.
- Any non-zero number in Python is interpreted as boolean True. False is interpreted as None and 0.

Python While Loop Example

```
i=1

while i<=5:

        print(i * ‘*’)

         i=i+1
```

Python Try Except

The `try` block lets you test a block of code for errors.

The `except` block lets you handle the error.
The `else` block lets you execute code when there is no error.
The `finally` block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling:

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.These exceptions can be handled using the `try` statement:

Example

The `try` block will generate an exception, because `x` is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Since the try block raises an error, the except block will be executed.
Without the try block, the program will crash and raise an error:

Example

This statement will raise an error, because `x` is not defined:

```
print(x)
```

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

Print one message if the try block raises a `NameError` and another for other errors:

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Python try...finally

The `try` statement in Python can have an optional `finally` clause. This clause is executed no matter what, and is generally used to release external resources.

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
finally:
  print("Program did not run")
```