

## Unit 1

### Python Programming

#### Introduction

- Python tutorial provides basic and advanced concepts of Python. Our Python tutorial is designed for beginners and professionals.
- Python is a simple, general purpose, high level, and object-oriented programming language.
- Python is an interpreted scripting language also. **Guido Van Rossum** is known as the **founder of Python programming in the year 1991**.

#### What is Python?

**Python** is a general purpose, dynamic, high-level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

It is used for:

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

#### What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

### **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.

### **Key Points to remember**

- Python is easy to learn yet **powerful** and **versatile scripting** language, which makes it **attractive** for **Application Development**.
- Python's **syntax and dynamic typing with its interpreted** nature make it an ideal language for scripting and rapid application development.
- Python supports **multiple programming pattern**, including **object-oriented**, imperative, and **functional or procedural** programming styles.
- Python is not intended to work in a particular area, such as web programming. That is why it is known as multipurpose programming language because it can be used with web, enterprise, 3D CAD, etc.
- We **don't** need **to use data types** to declare variable because it is **dynamically typed** so we can write **a=10** to assign **an integer value in an integer variable**.
- Python makes the **development and debugging fast** because there is **no compilation step included in Python development**, and edit-test-debug cycle is very fast.

## **Features of Python**

- **Easy to code:** Python is a very developer friendly language which means that anyone can learn to code it.
- **Open source and free:** It is an open-source programming language that is anyone can create and contribute to its development.
- **Object-oriented Approach:** It recognizes the concept of class and object encapsulation.
- **Support for other language:** Being coded in C, Python by default support the execution of code written in other programming language such as Java, C and C++.
- **High-level language:** This means no need to aware of the coding structure, architecture as well as memory management while coding Python.
- **Support for GUI (Graphical User Interface):** GUI has the ability to add flair to code and make the results more visual.
- Highly portable
- Extensive array of library

## **Applications of Python**



Python has wide range of libraries and frameworks widely used in various fields such as machine learning, artificial intelligence, web applications, etc. We define some popular frameworks and libraries of Python as follows.

**1. Web-Development:** Python is easily extensible one that provides good integration with databases and another web standard material. So, Python is a popular language for web development.

**Example :** World Best Search engine **Google** has been built using Python, World Best Video Sharing site **YouTube** developed using Python, The **front page** of internet **reddit** written in Python.

**2. Embedded Scripting Language:** Python is used as an embedded scripting language for various types of testing, building, deployment, monitoring frameworks, scientific apps, etc.

**3. Image Processing:** Python is used to make 2D imaging software such as GIMP, Paint Shop Pro, etc.

Python also used in 3D to make 3D animation software packages such as 3Ds Max, Maya, Blender, etc.

**4. GUI-Based Applications:** Python has a simple syntax, rich text processing, modular architecture and the ability to work on multiple operating systems makes Python a preferred choice for developers. Python is a GUI-based application so, it has various GUI tool-kits are available like **PyQt, wxPython, PyGtk**. This tool kit helps developers to create high function **Graphical User Interface** (GUI).

## **5. Game Development**

Python is also used in the development of interactive games. There are libraries such as **PySoy** which is a **3D game engine supporting Python 3, PyGame** which provides functionality and a library for game

development. Games such as **Civilization-IV, Disney's Toontown Online, Vega Strike** etc. have been **built using Python**.

## 6. Science and Numeric Applications

This is one of the widespread applications of Python programming. With its power, it comes as no surprise that Python finds its place in the scientific community. For this, we have:

- **SciPy** – A collection of packages for mathematics, science, and engineering.
- **Pandas**– A data-analysis and -modeling library
- **IPython** – A powerful shell for easy editing and recording of work sessions. It also supports visualizations and parallel computing.
- Also, **NumPy** enables us to deal with complex numerical calculations.

## 7. Software Development

Software developers make use of Python as a support language. They use it for build-control and management, testing, and for a lot of other things:

- **SCons** – for build-control
- **Buildbot, Apache Gump** – for automated and continuous compilation and testing
- **Roundup, Trac** – for project management and bug-tracking.
- The roster of Integrated Development Environments

## 8. Database Access

With Python, you have:

- Custom and ODBC interfaces to **MySQL, Oracle, PostgreSQL, MS SQL Server**, and others. These are freely available for download.
- Object databases like **Durus and ZODB**
- Standard Database API

## 9. Network Programming

With all those possibilities, how would Python slack in network programming? It does provide support for lower-level network programming:

- **Twisted Python** – A framework for asynchronous network programming. We mentioned it in section 2.
- An easy-to-use **socket interface**

## 10. Education

Thanks to its simplicity, brevity, and large community, Python makes for a great introductory programming language. Applications of Python programming in education has a huge scope as it is a great language to teach in schools or even learn on your own.

### Applications of Python Programming (2 Marks)

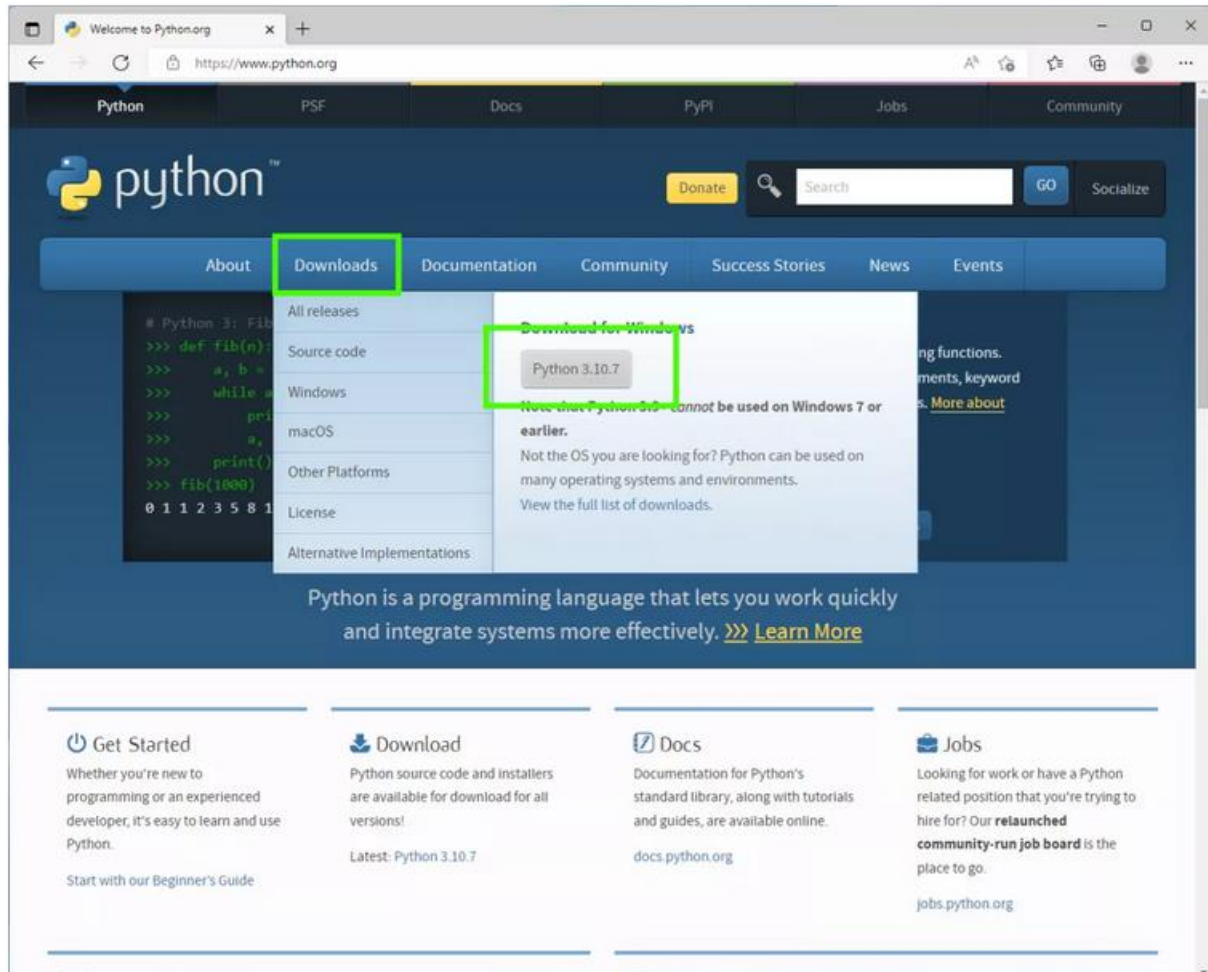
- Console-based Applications
- Audio or Video-based Applications
- Applications for Images
- Enterprise Applications
- 3D CAD Applications
- Computer Vision (Facilities like face-detection and color-detection)
- **Machine Learning**
- Robotics
- Web Scraping (Harvesting data from websites)
- Scripting
- **Artificial Intelligence**
- Data Analysis (The Hottest of Python Applications)

**Python Versions**

<b>Python Version</b>	<b>Released Date</b>
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	June 27, 2018
Python 3.8	October 14, 2019
Python 3.9	05 Oct 2020
Python 3.10	04 Oct 2021
Python 3.11	06 DEC 2022
Python 3.11	08 Feb 2023

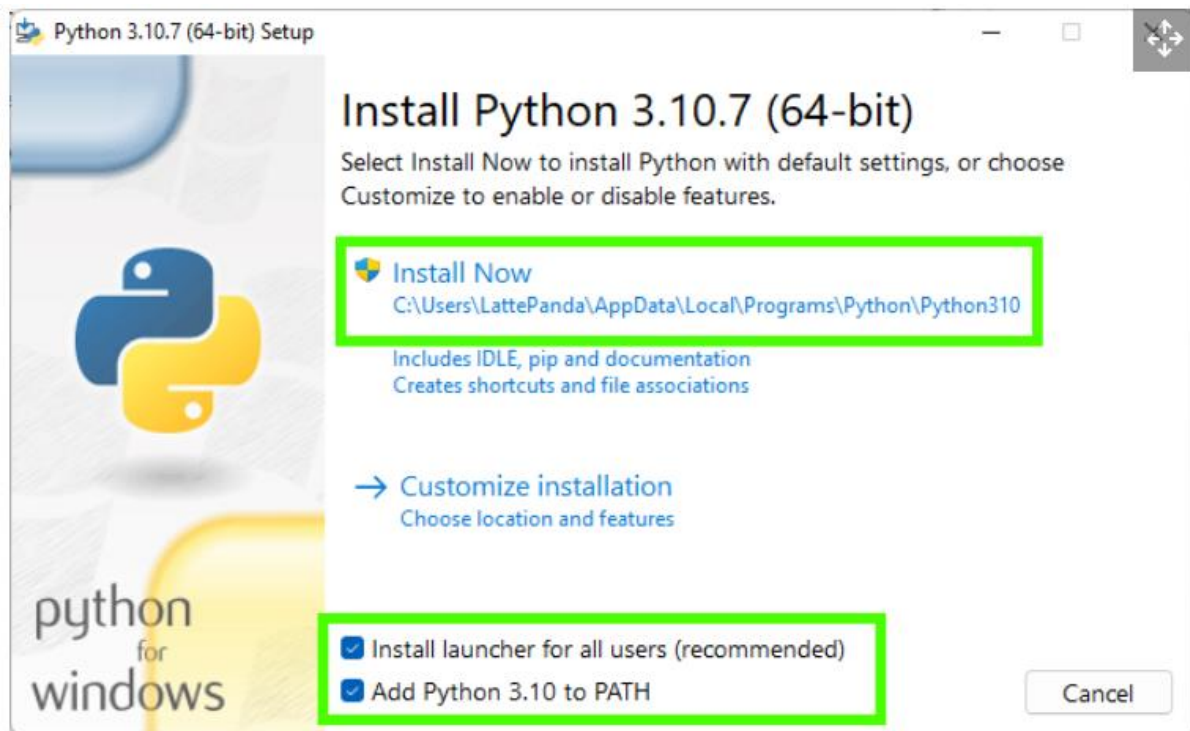
## **Installation of Python**

1. Open a browser to the Python website and download the Windows installer.

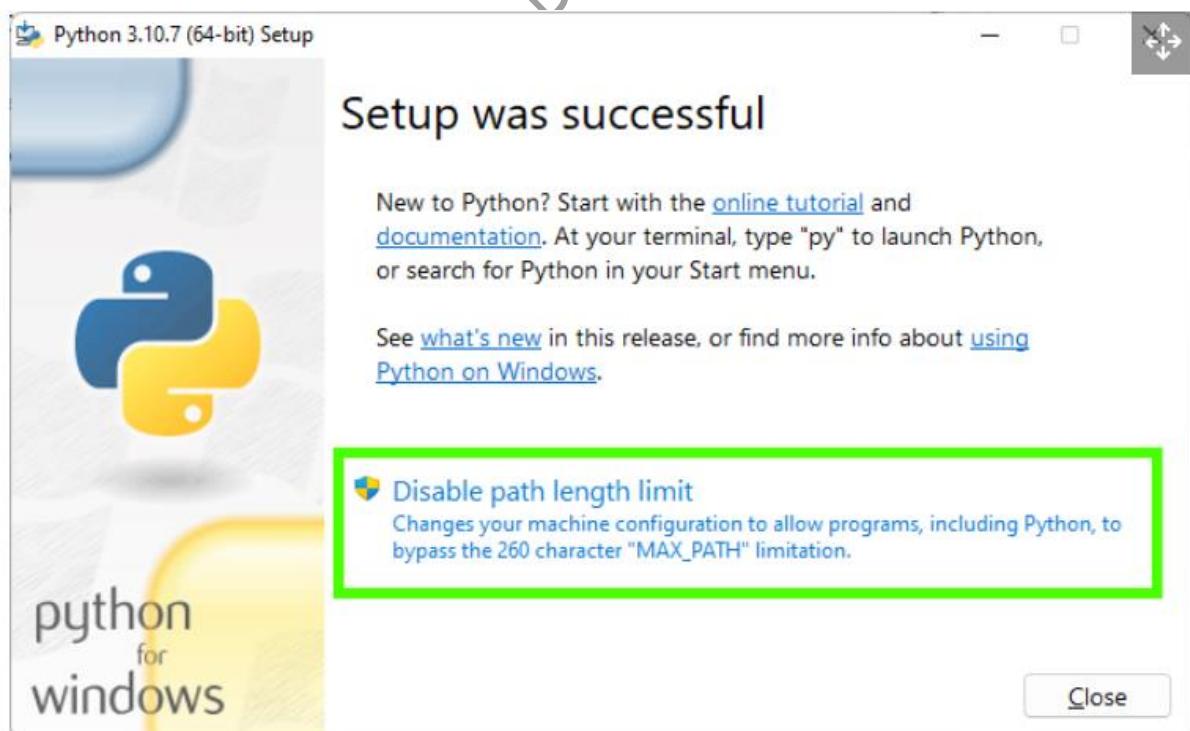


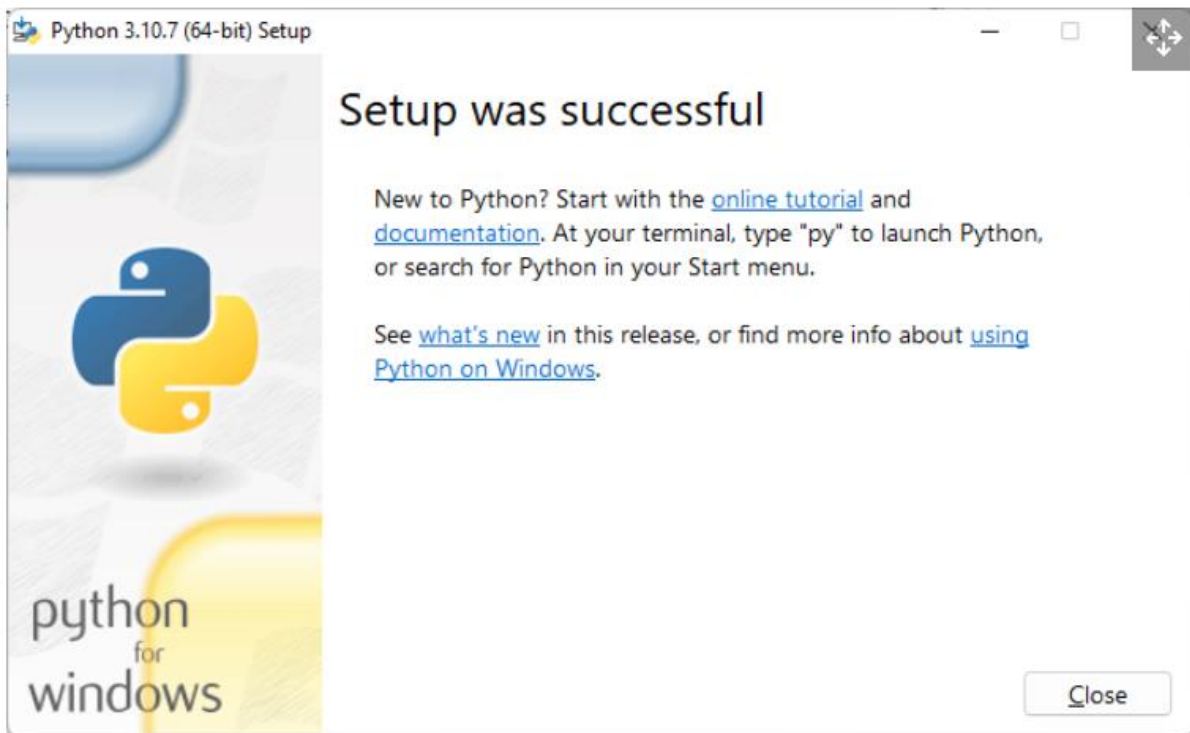
2. **Double click on the downloaded file and install Python for all users, and ensure that Python is added to your path. Click on Install now to begin.** Adding Python to the path will enable us to use the Python interpreter from any part of the file system.



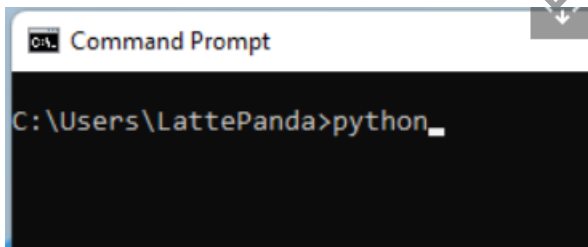


**3. After the installation is complete, click Disable path length limit and then Close.** Disabling the path length limit means we can use more than 260 characters in a file path.



**4. Click Close to end the installation.****Running Python in Windows**

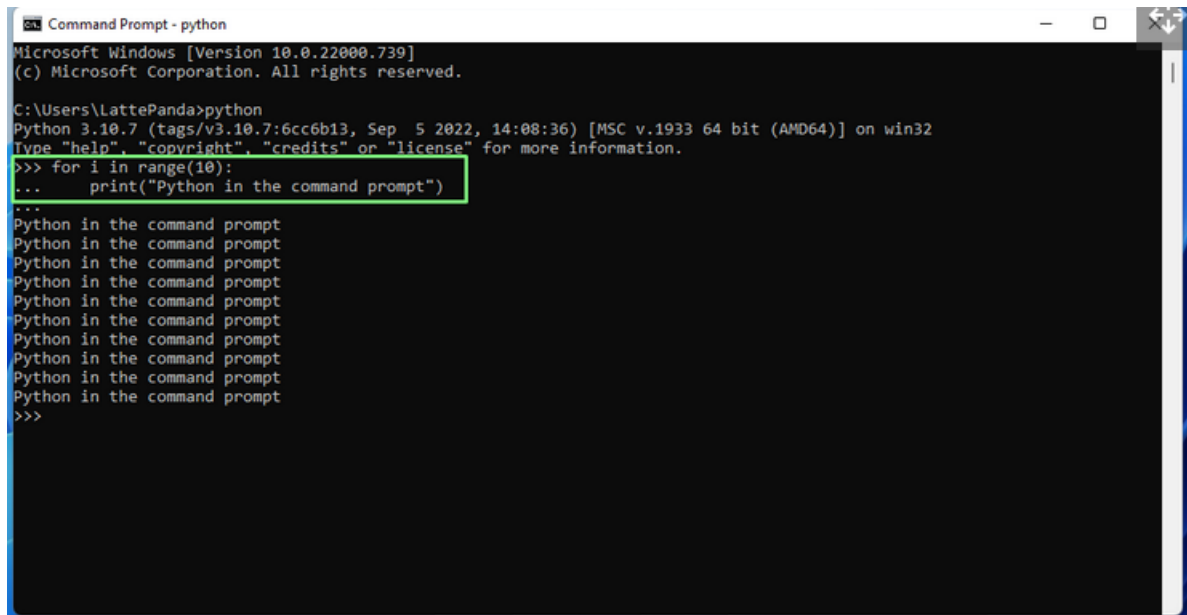
1. Open a Command Prompt and type "python" then press Enter.



2. Create a short Python script that uses a for loop to print a message to the Python shell ten times. Press space four times to indent the second line, otherwise Python will produce an error. Press Enter to run the code.

**Example**

```
for i in range(10):  
    print("Python in the command prompt")
```



```
Command Prompt - python
Microsoft Windows [Version 10.0.22000.739]
(c) Microsoft Corporation. All rights reserved.

C:\Users\LattePanda>python
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep 5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range(10):
...     print("Python in the command prompt")
...
Python in the command prompt
Python in the command prompt
Python in the command prompt
Python in the command prompt
Python in the command prompt
Python in the command prompt
Python in the command prompt
Python in the command prompt
Python in the command prompt
Python in the command prompt
>>>
```

## Python Command Line mode and Python IDEs

Python has two basic modes:

- Script mode
- Interactive mode.

1. **Script mode/normal mode** is the mode where the scripted and finished **.py** files are run in the Python interpreter.
2. **Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory.**

## **Python IDE(Integrated Development Environment)**

### **1. PyCharm**

PyCharm was developed by the Jet Brains, and it is a cross-platform Integrated Development Environment (IDE) specially designed for python. It is the most widely used IDE and available in both paid version and free open-source as well. It saves sample time by taking care of routine tasks.

It is a complete python IDE that is loaded with a rich set of features like auto code completion, quick project navigation, fast error checking and correction, remote development support, database accessibility, etc.

#### **Features**

- Smart code navigation
- Errors Highlighting
- Powerful debugger
- Supports Python web development frameworks, i.e., Angular JS, Javascript



### **2. Spyder**

Spyder is an open-source that has high recognition in the IDE market and most suitable for data science. The full name of Spyder is Scientific Python Development Environment. It supports all the significant platforms Linux, Windows, and MacOS X.

It provides a set of features like localized code editor, document viewer, variable explorer, integrated console, etc. and supports no. of scientific modules like NumPy, SciPy, etc.

#### **Features**

- Proper syntax highlighting and auto code completion
- Integrates strongly with IPython console
- Performs well in multi-language editor and auto code completion mode.



### **3. PyDev**

**PyDev** is defined as one of the commonly used Python IDE, which is an external plugin for Eclipse. It is a natural choice of the **Python developers** that are coming from the Java background and very popular in the market as Python interpreter.

Aleksandar Totic is famous for his contribution to Mosaic browser and worked on Pydev project during 2003-2004.

Pydev has a feature which includes Django integration, automatic code completion, smart indents and block indents, etc.

#### **Features**

- Strong Parameters like refactoring, debugging, code analysis, and code coverage function.
- It supports virtual environments, Mypy, and black formatter.
- Also supports PyLint integration, remote debugger, Unit test integration, etc.



### **4. Atom**

Atom is developed by GitHub, which is initially started as an open-source, cross-platform. It is based on a framework, i.e., Electron which enables cross-platform desktop application using Chromium and Node.js and generally known as "Hackable Text Editor for the 21<sup>st</sup> century".

#### **Features**

- Visualize the results on Atom without open any other window.
- A plugin named "Markdown Preview Plus" provides built-in support for editing and visualizing Markdown files.



### **5. Wing**

It is defined as a cross-platform IDE that is packed with necessary features and with decent development support. Its personal edition is free of cost. The pro version comes with a 30 days trial for the developers to try it out.

It has several features that include auto-completion, syntax highlighting, indents, and debugging.

**Features**

- Customizable and can have extensions as well.
- Supports remote development, test-driven development along with the unit test.

**6. Jupyter Notebook**

Jupyter is one of the most used IPython notebook editors that is used across the Data Science industry. It is a web application that is based on the server-client structure and allows you to create and manipulate notebook documents. It makes the best use of the fact that python is an interpreted language.

**Features**

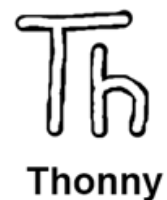
- Supports markdowns
- Easy creation and editing of codes
- Ideal for beginners in data science

**7. Thonny**

Thonny is another IDE which is best suited for learning and teaching programming. It is a software developed at the University of Tartu and supports code completion and highlight syntax errors.

**Features**

- Simple debugger
- Supports highlighting errors and auto code completion

**8. Rodeo**

Rodeo is defined as one of the best IDE for python that is most widely used for data science projects like taking data and information from different resources.

It supports cross-platform functionality and provides auto-completion of code.



**Features**

- Allows the functions for comparing data, interact, plot, and inspect data.
- Auto code completion, syntax highlighter, visual file navigator, etc.

**9. Microsoft Visual Studio**

Microsoft Visual Studio is an open-source code editor which was best suited for development and debugging of latest web and cloud projects. It has its own marketplace for extensions.

**Features**

- Supports Python Coding in Visual studio
- Available in both paid and free version

**10. Eric Python**

The Eric Python is an editor which is developed in Python itself and can be used for both professional and non-professional work.

**Features**

- Offers configurable window layout, editors, source code folding
- Advanced project management capability, version control
- In-built debugger and task management support

**Simple Python Program.****Example****1. Add Two Numbers**

**# This program adds two numbers**

num1 = 1.5

num2 = 6.3

**# Add two numbers**

sum = num1 + num2

**# Display the sum**

print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))

**Output**

The sum of 1.5 and 6.3 is 7.8

**2. # Python Program to find the area of triangle**

```
a = 5
b = 6
c = 7
```

```
# Uncomment below to take inputs from the user
# a = float(input('Enter first side: '))
# b = float(input('Enter second side: '))
# c = float(input('Enter third side: '))
```

```
# calculate the semi-perimeter
s = (a + b + c) / 2
```

```
# calculate the area
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print('The area of the triangle is %0.2f' %area)
```

**Output**

The area of the triangle is 14.70

**Python Basics:****Identifiers**

A Python identifier is a name used to identify a **variable, function, class, module or other object**. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Python does **not allow punctuation characters such as @, \$, and %** within identifiers. Python is a **case sensitive programming language**.

**Example**

```
language = 'Python'
```

Here, **language** is a **variable** (an **identifier**) which holds the **value 'Python'**.



**Rules**

- Identifiers cannot be a keyword.
- Identifiers are case-sensitive.
- It can have a sequence of letters and digits. However, it must begin with a letter or `_`. The first letter of an identifier cannot be a digit.
- It's a convention to start an identifier with a letter rather `_`.
- Whitespaces are not allowed.
- We cannot use special symbols like `!`, `@`, `#`, `$`, and so on.

**Keywords**

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers:

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.

is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

### **Difference between Statements and Expressions in Python**

We have earlier discussed statement expression in Python, let us learn the differences between them.

<b>Statement in Python</b>	<b>Expression in Python</b>
A statement in Python is used for creating variables or for displaying values.	The expression in Python produces some value or result after being interpreted by the Python interpreter.
A statement in Python is not evaluated for some results.	An expression in Python is evaluated for some results.
The execution of a statement changes the state of the variable.	The expression evaluation does not result in any state change.
A statement can be an expression.	An expression is not a statement.
<b>Example :</b> $x=3$ <b>Output :</b> 333	<b>Example:</b> $x=3+6$ <b>Output :</b> 999

## **Expressions in Python**

**1. Constant Expressions:** These are the expressions that have constant values only.

**Example:**

**# Constant Expressions**

```
x = 15 + 1.3
```

```
print(x)
```

**Output:** 16.3

**2. Arithmetic Expressions:** An arithmetic expression is a combination of numeric values, operators, and sometimes parenthesis. The result of this type of expression is also a numeric value. The operators used in these expressions are arithmetic operators like addition, subtraction, etc. Here are some arithmetic operators in Python:

<b>Operators</b>	<b>Syntax</b>	<b>Functioning</b>
+	x + y	Addition
-	x - y	Subtraction
*	x * y	Multiplication
/	x / y	Division
//	x // y	Quotient
%	x % y	Remainder
**	x ** y	Exponentiation

**Example:**

Let's see an exemplar code of arithmetic expressions in Python :

**# Arithmetic Expressions**

```
x = 40
y = 12
add = x + y
sub = x - y
pro = x * y
div = x / y
print(add)
print(sub)
print(pro)
print(div)
```

**Output**

```
52
28
480
3.3333333333333335
```

**3. Integral Expressions:** These are the kind of expressions that produce only integer results after all computations and type conversions.

**Example:****# Integral Expressions**

```
a = 13
b = 12.0
c = a + int(b)
print(c)
```

**Output**

```
25
```

**4.. Floating Expressions:** These are the kind of expressions which produce floating point numbers as result after all computations and type conversions.

**Example:**

**# Floating Expressions**

```
a = 13
b = 5
c = a / b
print(c)
```

**Output**

2.6

**5. Relational Expressions:** In these types of expressions, arithmetic expressions are written on both sides of relational operator (> , < , >= , <=). Those arithmetic expressions are evaluated first, and then compared as per relational operator and produce a boolean output in the end. These expressions are also called Boolean expressions.

**Example:**

**# Relational Expressions**

```
a = 21
b = 13
c = 40
d = 37
p = (a + b) >= (c - d)
print(p)
```

**Output**

True

**6. Logical Expressions:** These are kinds of expressions that result in either *True* or *False*. It basically specifies one or more conditions. For example,  $(10 == 9)$  is a condition if 10 is equal to 9. As we know it is not correct, so it will return *False*. Studying logical expressions, we also come across some logical operators which can be seen in logical expressions most often. Here are some logical operators in Python:

Operator	Syntax	Functioning
and	P and Q	It returns true if both P and Q are true otherwise returns false
or	P or Q	It returns true if at least one of P and Q is true
not	not P	It returns true if condition P is false

**Example:**

**#Let's have a look at an exemplar code :**

P =  $(10 == 9)$

Q =  $(7 > 5)$

**# Logical Expressions**

R = P and Q

S = P or Q

T = not P

print(R)

print(S)

print(T)

**Output**

False

True

True

**7. Bitwise Expressions:** These are the kind of expressions in which computations are performed at bit level.

**Example:**

```
# Bitwise Expressions
a = 12
x = a >> 2
y = a << 1
print(x, y)
```

**Output**

3 24

**8. Combinational Expressions:** We can also use different types of expressions in a single expression, and that will be termed as combinational expressions.

**Example:**

```
# Combinational Expressions
a = 16
b = 12
c = a + (b >> 1)
print(c)
```

**Output**

22

But when we combine different types of expressions or use multiple operators in a single expression, operator precedence comes into play.

**9. Multiple operators in expression (Operator Precedence)**

It's a quite simple process to get the result of an expression if there is only one operator in an expression. But if there is more than one operator in an expression, it may give different results on basis of the order of operators executed. According to priority wise.

Precedence	Name	Operator
1	Parenthesis	( ) [ ] { }
2	Exponentiation	**
3	Unary plus or minus, complement	-a , +a , ~a
4	Multiply, Divide, Modulo	/ * // %
5	Addition & Subtraction	+ -
6	Shift Operators	>> <<
7	Bitwise AND	&
8	Bitwise XOR	^
9	Bitwise OR	
10	Comparison Operators	>= <= > <
11	Equality Operators	== !=
12	Assignment Operators	= += -= /= *=
13	Identity and membership operators	is, is not, in, not in
14	Logical Operators	and, or, not

**Example****# Multi-operator expression**

```

a = 10 + 3 * 4
print(a)
b = (10 + 3) * 4
print(b)
c = 10 + (3 * 4)

```



```
print(c)
```

**Output**

```
22
```

```
52
```

```
22
```

**What is a Variable in Python?**

Python variables are the reserved memory locations used to store values with in a Python Program. This means that when you create a variable you reserve some space in the memory.

**Creating Variables**

Python has no command for declaring a variable.

A variable is created the moment you

**Example**

```
counter = 100      # Creates an integer variable
miles  = 1000.0    # Creates a floating point variable
name   = "Zara Ali" # Creates a string variable
```

Here “**counter, miles and name**” are variables

**Python Variable Types**

Every value in Python has a **data type**. Different **data types** in Python are **Numbers, List, Tuple, Strings, Dictionary**, etc. **Variables in Python can be declared by any name or even alphabets like a, aa, abc**, etc.

**Printing Python Variables**

Once we create a Python variable and assign a value to it, we can print it using **print()** function. Following is the extension of previous example and shows how to print different variables in Python:

```
counter = 100          # Creates an integer variable
miles   = 1000.0       # Creates a floating point variable
name    = "Divya Setty" # Creates a string variable
print (counter)
print (miles)
print (name)
```

**Output:**

```
100
1000.0
Divya Setty
```

DIVYA

**Rules for creating variables in Python**

- 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 (name, Name and NAME are three different variables).
- The reserved words (keywords) cannot be used naming the variable.

**Re-declare the Variable**

We can re-declare the python variable once we have declared the variable already.

**Example****# declaring the variable Number**

```
Number = 100
```

**# display**

```
print("Before declare: ", Number)
```

**# re-declare the variable**

```
Number = 120.3
```

```
print("After re-declare:", Number)
```

**Output:**

Before declare: 100

After re-declare: 120.3

**Assigning a single value to multiple variables**

Also, Python allows assigning a single value to several variables simultaneously with "=" operators.

**For example:**

```
a = b = c = 10
```

```
print(a)
```

```
print(b)
```

```
print(c)
```

**Output:**

10

10

10

**Can we use the same name for different types?**

If we use the same name, the variable starts referring to a new value and type.

**Example**

```
a = 10
a = "Python Programming"
print(a)
```

**Output:**

Python Programming

**Multiple Assignment**

Python allows you to assign a single value to several variables simultaneously which means you can create multiple variables at a time.

**For example**

```
a = b = c = 100
print (a)
print (b)
print (c)
```

**This produces the following result:**

```
100
100
100
```

**Types of a variable**

- a. Local Variable
- b. Global Variable

**Python Local Variable**

Python Local Variables are defined inside a function. We cannot access variable outside the function.

**Note:** Python **def** keyword is used **to define a function**

**Example**

```
def sum(x,y):  
    sum = x + y  
    return sum  
print(sum(5, 10))
```

**Output:**

15

**Python Global Variable**

Any variable created outside a function can be accessed within any function and so they have global scope. Following is an example of global variables:

**Example**

```
x = 5  
y = 10  
def sum():  
    sum = x + y  
    return sum  
print(sum())
```

**Output**

15

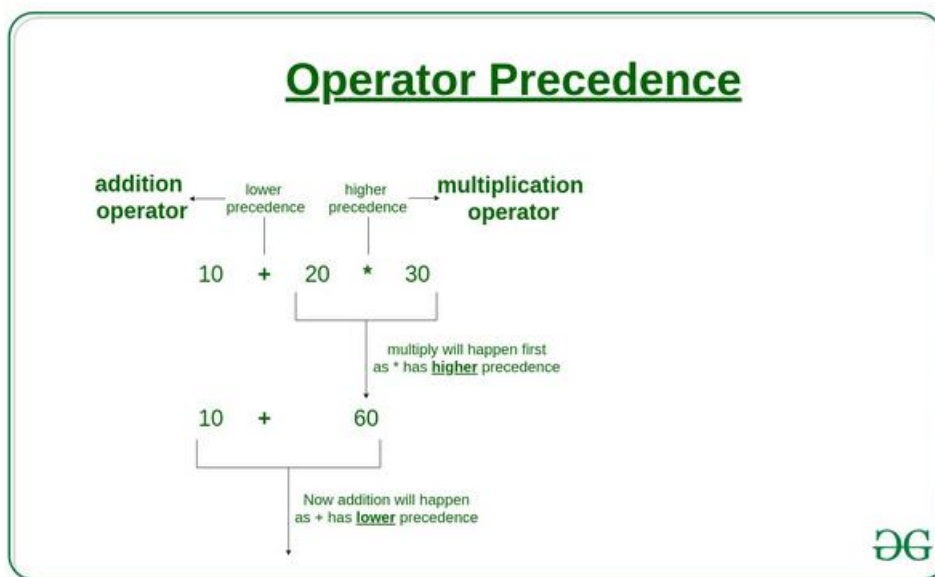
## **Precedence and Associativity of Operators in Python**

When dealing with operators in Python we have to know about the concept of Python operator precedence and associativity as these determine the priorities of the operator otherwise, we'll see unexpected outputs.

**Operator Precedence:** This is used in an expression with more than one operator with different precedence to determine which operation to perform first.

**Example:** Solve

$10 + 20 * 30$



### **# Precedence of '+' & '\*'**

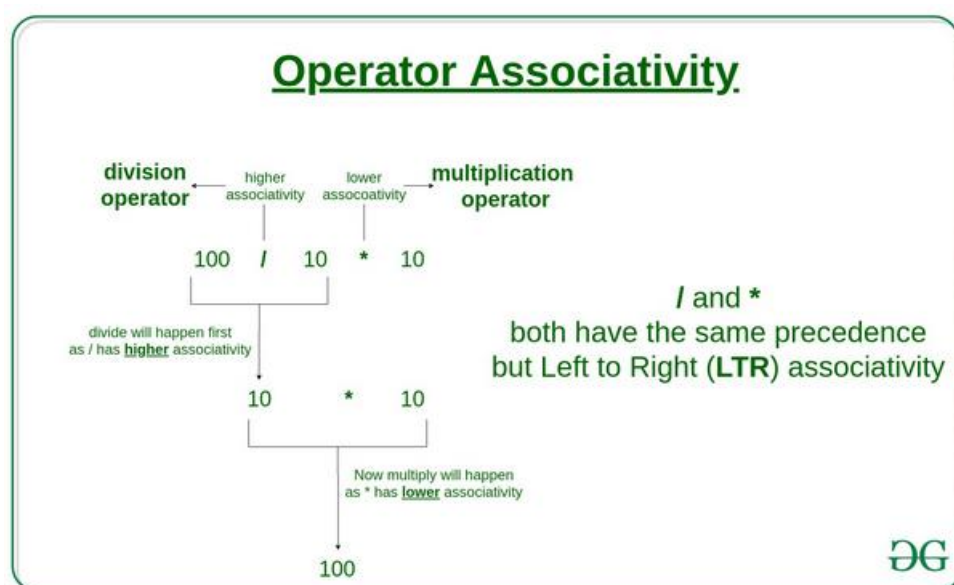
```
expr = 10 + 20 * 30
print(expr)
```

### **Output:**

610

**Operator Associativity:** If an expression contains two or more operators with the same precedence then Operator Associativity is used to determine. It can either be **Left to Right** or from **Right to Left**.

**Example:** '\*' and '/' have the same precedence and their associativity is **Left to Right**, so the expression "100 / 10 \* 10" is treated as "(100 / 10) \* 10".



### **Example**

**# Left-right associativity**

**# 100 / 10 \* 10 is calculated as**

**# (100 / 10) \* 10 and not**

**# as 100 / (10 \* 10)**

`print(100 / 10 * 10)`

**# Left-right associativity**

**# 5 - 2 + 3 is calculated as**

**# (5 - 2) + 3 and not**

**# as 5 - (2 + 3)**

`print(5 - 2 + 3)`

**# left-right associativity**

```
print(5 - (2 + 3))
```

**# right-left associativity**

**# 2 \*\* 3 \*\* 2 is calculated as**

**# 2 \*\* (3 \*\* 2) and not**

**# as (2 \*\* 3) \*\* 2**

```
print(2 ** 3 ** 2)
```

**Output:**

```
100
```

```
6
```

```
0
```

```
512
```

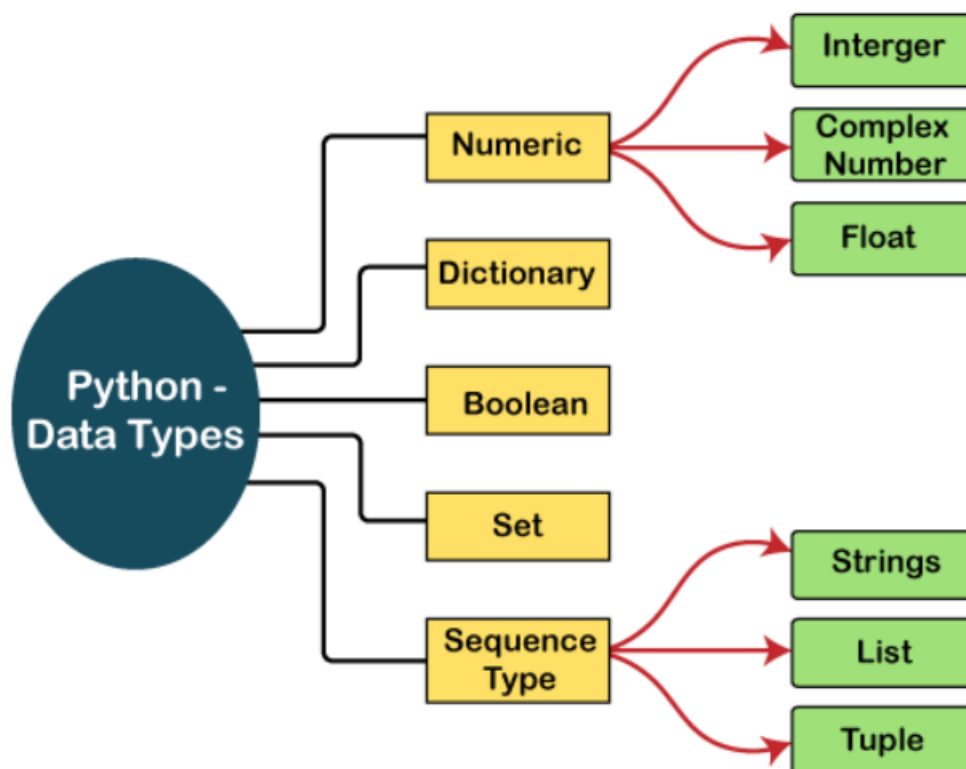
Operator	Description	Associativity
( )	Parentheses	left-to-right
**	Exponent	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
is, is not	Identity	left-to-right
in, not in	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



Operator	Description	Associativity
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^=  =	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	

## Data Types

There are different data types in python



## Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

<b>Text Type:</b>	str
<b>Numeric Types:</b>	int, float, complex
<b>Sequence Types:</b>	list, tuple, range

<b>Mapping Type:</b>	dict
<b>Set Types:</b>	set, frozenset
<b>Boolean Type:</b>	bool
<b>Binary Types:</b>	bytes, bytearray, memoryview
<b>None Type:</b>	NoneType

## 1. Numbers

- 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.

### Example

```

a = 5
print("The type of a", type(a))
b = 40.5
print("The type of b", type(b))
c = 1+3j
print("The type of c", type(c))
print(" c is a complex number", isinstance(1+3j,complex))

```

### Output:

```

The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
c is complex number: True

```

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.

## **2. String**

- A string is an ordered sequence of characters.
- We can use **single quotes** or **double quotes** to represent strings. Multi-line strings can be represented using **triple quotes, ''' or ''''**.
- Strings are immutable which means once we declare a string we can't update the already declared string

### **Example:**

```
String1 = "Welcome"  
String2 = "To Python"  
print(String1+String2)
```

**Output:** Welcome To Python

## **3. List**

- A list can contain a series of values.
- List variables are declared by using **brackets [ ]**.
- A list is **mutable**, which means we can **modify** the list.

### **Example:**

```
List = [2,4,5.5,"Hi"]  
print("List[2] = ", List[2])
```

**Output:** List[2] = 5.5

#### **4. Tuple**

- A **tuple** is a sequence of Python objects separated by **commas**.
- Tuples are **immutable**, which means tuples once created **cannot be modified**.
- Tuples are defined using **parentheses()**.

#### **Example:**

```
Tuple = (50,15,25.6,"Python")  
print("Tuple[1] = ", Tuple[1])
```

**Output:** Tuple[1] = 15

#### **5. Set**

- A set is an unordered collection of items.
- Set is defined by values separated by a comma inside braces { }.

#### **Example:**

```
Set = {5,1,2.6,"python"}  
print(Set)
```

**Output:** {'python', 1, 5, 2.6}

#### **6. Dictionary**

- Dictionaries are the most flexible built-in data type in python.
- Dictionaries items are stored and fetched by using the key.
- Dictionaries are used to store a huge amount of data.
- To retrieve the value we must know the key.
- In Python, dictionaries are defined within braces {}.

We use the key to retrieve the respective value. But not the other way around.

**Syntax:****Key:value****Example:**

```
Dict = {1:'Hi',2:7.5, 3:'Class'}
print(Dict)
```

**Output:** {1: 'Hi', 2: 7.5, 3: 'Class'}**Keywords in python**

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

**Note****>>> help ("keywords")**

Here is a list of the Python keywords. Enter any keyword to get more help.

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

## **Indentation**

- Indentation **refers to the spaces at the beginning of a code line.**
- Where in other programming languages the **indentation in code** is for **readability only**, the indentation in Python is very important.
- Python uses indentation to indicate a **block of code**.

### **Example**

```
if 5 > 2:
```

```
    print("Five is greater than two!")
```

**Output:** Five is greater than two!

**Or**

### **Example**

```
if 5 > 2:
```

```
    print("Five is greater than two!")
```

```
if 5 > 2:
```

```
    print("Five is greater than two!")
```

**Output:**

Five is greater than two!

Five is greater than two!

**Or**

### **Example**

```
if 5 > 2:
```

```
    print("Five is greater than two!")
```

```
        print("Five is greater than two!")
```

**Output:**

```
File "demo_indentation2_error.py", line 3
    print("Five is greater than two!")
    ^
IndentationError: unexpected indent
```

## **Python 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:

### **Example**

#### **a) Single Line Comment**

**#This is a comment**

```
print("Hello, World!")
```

#### **b) 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**

```
print("Hello, World!")
```

## **Built in Function - console input and console output in python**

Console Input/Output are the built in function in python, **output** is most important as it acts as the basic text based interface for the python programs

### **console input**

To read a number from console in Python input by user, you can use **input()** function.

**input()** functions enables your Console Python application to **read input from user**.

In this example, we shall read an input from user using **input()** function.

**input()** function returns a string

### **Python Program**

**#read integer from user**

```
n1 = int(input('Enter a number: ')) #Input from the user
```

```
n2 = int(input('Enter another number: ')) #Input from the user
```

```
print('The sum of two numbers is:', n1+n2) #Output function
```

### **Output**

Enter a number: 52

Enter another number: 14

The sum of two numbers is: 66

### **console output**

The built-in function **print()** of Python, prints any string passed to it to the screen.

### **Example:**

```
print('Hello python world')
```

### **Type Conversion**

Type conversion is the process of converting data of one type to another.

For example: converting **int data** to **str**.

There are two types of type conversion in Python.

- Implicit Conversion - automatic type conversion
- Explicit Conversion - manual type conversion



## **1) Implicit Type Conversion**

In certain situations, Python automatically converts one data type to another. This is known as implicit type conversion.

### **Converting integer to float**

Let's see an example where Python promotes the conversion of the lower data type (integer) to the higher data type (float) to avoid data loss.

```
x = 10
print("x is of type:",type(x))

y = 10.6
print("y is of type:",type(y))

z = x + y

print(z)
print("z is of type:",type(z))
```

#### **Output:**

```
x is of type: <class 'int'>
y is of type: <class 'float'>
20.6
z is of type: <class 'float'>
```

## **2) Explicit Type Conversion**

In Explicit Type Conversion, **users convert** the **data type** of an **object to required data type**.

We use the built-in functions like **int()**, **float()**, **str()**, etc to perform explicit type conversion.

This **type of conversion** is also called **typecasting** because the user casts (changes) the data type of the objects.

**Example**

```
integer = 2
decimal = 5.0
x = decimal + integer
print("Type after adding both values :")
print(type(x))
```

**Output**

```
Type after adding both values :
< class 'float' >
```

**Note:**

Function	Description
int(y [base])	It converts y to an integer, and Base specifies the number base. For example, if you want to convert the string in decimal numbers then you'll use 10 as base.
float(y)	It converts y to a floating-point number.
complex(real [imag])	It creates a complex number.
str(y)	It converts y to a string.
tuple(y)	It converts y to a tuple.
list(y)	It converts y to a list.
set(y)	It converts y to a set.
dict(y)	It creates a dictionary and y should be a sequence of (key, value) tuples.
ord(y)	It converts a character into an integer.
hex(y)	It converts an integer to a hexadecimal string.
oct(y)	It converts an integer to an octal string

## **Python Libraries; Importing Libraries**

In Python, libraries are used to refer to a collection of modules that are used repeatedly in various programs without the need of writing them from scratch.

Modules on the other hand refer to any Python file saved with the **.py** extension. Modules often contain code such as functions, classes and statements that can be imported and used within other programs.

## **Create a Module**

To create a module just save the code you want in a file with the file extension **".py"**

### **Example**

Save this code in a file named **mymodule.py**

```
def greeting(name):  
    print("Hello, " + name)
```

## **Use a Module**

Now we can use the module we just created, by using the import statement:

### **Example**

Import the module named **mymodule**, and call the greeting function:

```
import mymodule  
mymodule.greeting("Divya Setty")
```

## **Built-in Modules**

There are several built-in modules in Python, which you can import whenever you like.

### **a) platform**

#### **Example**

**#Import and use the platform module:**

```
import platform  
x = platform.system()  
print(x)
```

**Output:** Windows

### **b) Using the dir() Function**

There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

#### **Example**

**#List all the defined names belonging to the platform module:**

```
import platform  
x = dir(platform)  
print(x)
```

#### **Output:**

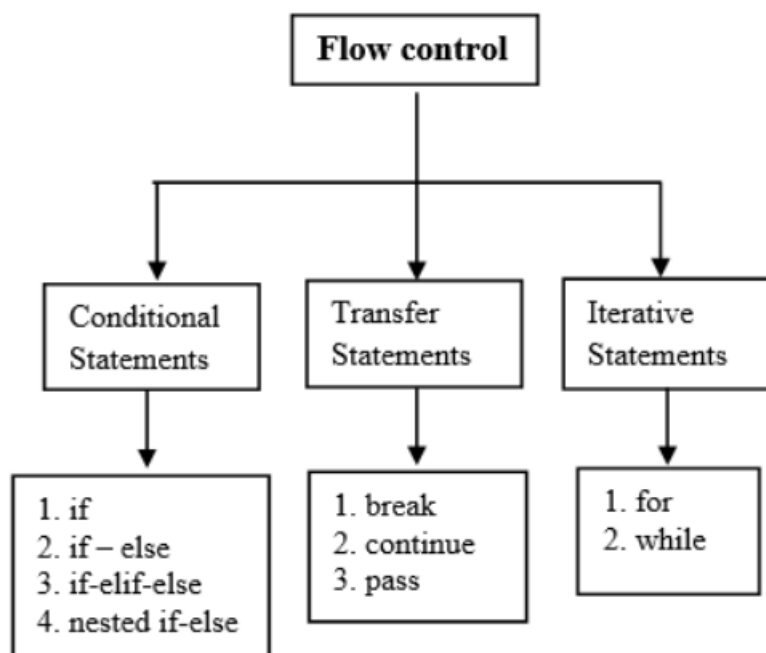
```
['DEV_NULL', '_UNIXCONFDIR', 'WIN32_CLIENT_RELEASES',  
'WIN32_SERVER_RELEASES', '__builtins__', '__cached__', '__copyright__', '__doc__',  
'__file__', '__loader__', '__name__', '__package__', '__spec__', '__version__',  
'_default_architecture', '_dist_try_harder', '_follow_symlinks',  
'_ironpython26_sys_version_parser', '_ironpython_sys_version_']
```

## **Python Control Flow**

The control flow of a Python program is regulated by conditional statements, loops, and function calls.

Python has *three* types of control structures:

1. Conditional statements
2. Iterative statements.
3. Transfer statements



### **Conditional statements**

In Python, condition statements act depending on whether a given condition is **true or false**. You can execute different blocks of codes depending on the outcome of a condition. Condition statements always evaluate to either **True or False**.

There are three types of conditional statements.

1. if statement
2. if-else
3. if-elif-else
4. nested if-else

### **Iterative statements**

In Python, iterative statements allow us to execute a block of code repeatedly as long as the condition is True. We also call it a loop statements.

Python provides us the following two loop statement to perform some actions repeatedly

1. for loop
2. while loop

### **Transfer statements**

In Python, transfer statements are used to alter the program's way of execution in a certain manner. For this purpose, we use three types of transfer statements.

1. break statement
2. continue statement
3. pass statements

## **Conditional statements**

### **a) If statement in Python**

In control statements, The if statement is the simplest form. It takes a condition and evaluates to either True or False.

If the condition is True, then the True block of code will be executed, and if the condition is False, then the block of code is skipped, and The controller moves to the next line

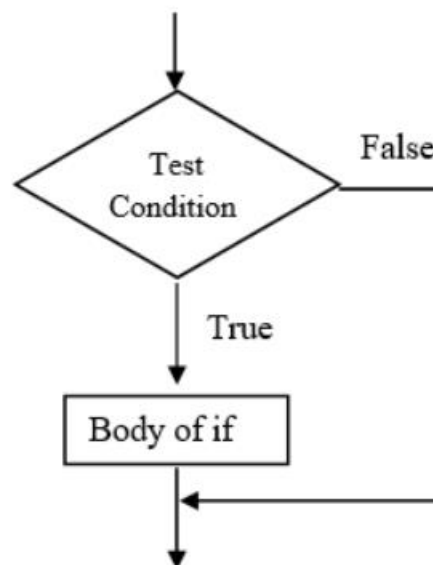
### **Syntax of the if statement**

if condition:

statement **1**

statement **2**

statement **n**



**Fig. Flowchart of if statement**

**Example**

```
number = 6
if number > 5:
    # Calculate square
    print(number * number)
print('Next lines of code')
```

**Output**

```
36
Next lines of code
```

**b) If – else statement**

The if-else statement checks the condition and executes the if block of code when the condition is True, and if the condition is False, it will execute the else block of code.

**Syntax of the if-else statement**

```
if condition:
    statement 1
else:
    statement 2
```

If the condition is True, then statement 1 will be executed. If the condition is False, statement 2 will be executed. See the following flowchart for more detail.



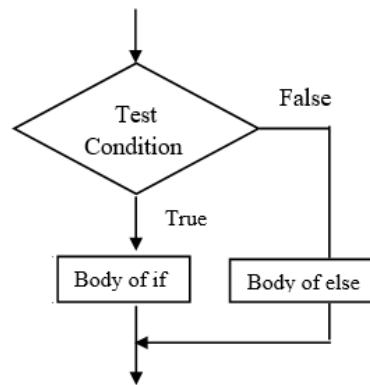


Fig. Flowchart of if-else

### Example

```
password = input('Enter password ')
```

```
if password == "Divya":
```

```
    print("Correct password")
```

```
else:
```

```
    print("Incorrect Password")
```

### Output 1:

Enter password **Divya**

Correct password

### Output 2:

Enter password **divya**

Incorrect Password

### c) if-elif-else

- In Python, the **if-elif-else** condition statement has an **elif** blocks to chain multiple conditions one after another. This is useful when you need to check multiple conditions.
- With the help of **if-elif-else** we can make a tricky decision. The **elif** statement checks multiple conditions one by one and if the condition fulfills, then executes that code.

**Syntax of the if-elif-else statement:**

if condition-1:

    statement 1

elif condition-2:

    statement 2

elif condition-3:

    statement 3

...

else:

    statement

**Example**

```
def user_check(choice):
```

```
    if choice == 1:
```

```
        print("Admin")
```

```
    elif choice == 2:
```

```
        print("Editor")
```

```
    elif choice == 3:
```

```
        print("Guest")
```

```
    else:
```

```
        print("Wrong entry")
```

```
user_check(1)
```

```
user_check(2)
```

```
user_check(3)
```

```
user_check(4)
```

**Output:**

Admin

Editor

Guest

Wrong entry

**d) Nested if-else statement**

- In Python, the nested **if-else** statement is an if statement inside another **if-else** statement. It is allowed in Python to put any number of if statements in another if statement.
- Indentation is the only way to differentiate the level of nesting. The nested **if-else** is useful when we want to make a series of decisions.

**Syntax of the nested-if-else:**

```
if conditon_outer:
```

```
    if condition_inner:
```

```
        statement of inner if
```

```
    else:
```

```
        statement of inner else:
```

```
    statement ot outer if
```

```
else:
```

```
    Outer else
```

```
statement outside if block
```

DIVYA

**Example****#Find a greater number between two numbers**

```
num1 = int(input('Enter first number '))
```

```
num2 = int(input('Enter second number '))
```

```
if num1 >= num2:
```

```
    if num1 == num2:
```

```
        print(num1, 'and', num2, 'are equal')
```

```
    else:
```

```
        print(num1, 'is greater than', num2)
```

```
else:
```

```
    print(num1, 'is smaller than', num2)
```

**Output 1:**

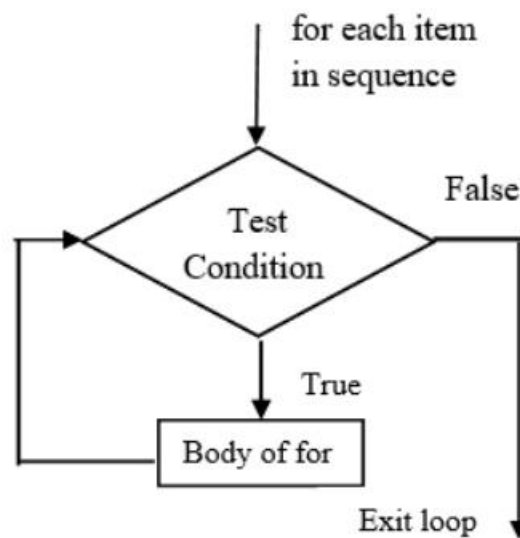
Enter first number 56  
Enter second number 15  
56 is greater than 15

**Output 2:**

Enter first number 29  
Enter second number 78  
29 is smaller than 78

**Looping Statements****a) for loop**

Using for loop, we can iterate any sequence or iterable variable. The sequence can be string, **list, dictionary, set, or tuple.**



**Fig. Flowchart of for loop**

**Syntax of for loop:**

for **element** in **sequence**:  
    body of for loop

**#Example to display first ten numbers using for loop**

```
for i in range(1, 11):  
    print(i)
```

**Output**

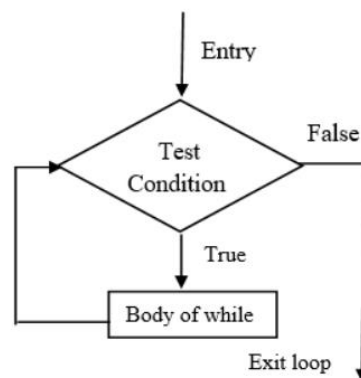
1  
2  
3  
4  
5  
6  
7  
8  
9  
10

**b) While loop**

In Python, The while loop statement repeatedly executes a code block while a particular condition is true.

In a while-loop, every time the condition is checked at the beginning of the loop, and if it is true, then the loop's body gets executed.

When the condition became False, the controller comes out of the block.



**Fig. Flowchart of while loop**

**Syntax of while-loop**

while **condition**:

    body of while loop

**#Example to calculate the sum of first ten numbers**

```
num = 10
```

```
sum = 0
```

```
i = 1
```

```
while i <= num:
```

```
    sum = sum + i
```

```
    i = i + 1
```

```
print("Sum of first 10 number is:", sum)
```

**output**

Sum of first 10 number is: 55

**Transfer statements****a) Break Statement**

- **The break statement is used inside the loop to exit out of the loop.** It is useful when we want to terminate the loop as soon as the condition is fulfilled instead of doing the remaining iterations.
- It reduces execution time. Whenever the controller encountered a break statement, it comes out of that loop immediately.

**#Example of using a break statement**

```
for num in range(10):
```

```
    if num > 5:
```

```
        print("stop processing.")
```

```
        break
```

```
    print(num)
```

**Output**

0  
1  
2  
3  
4  
5  
stop processing.

**b) Continue statement**

The **continue statement** is used to skip the current iteration and continue with the next iteration.

**#Example of a continue statement**

```
for num in range(3, 8):
```

```
    if num == 5:
```

```
        continue
```

```
    else:
```

```
        print(num)
```

**Output**

3  
4  
6  
7

**c) Pass statement**

- The **pass** is the keyword In Python, **which won't do anything.**
- A **pass statement** is a Python **null** statement. When the interpreter finds a pass statement in the program, it returns **no operation.**
- Nothing happens when the pass statement is executed.
- It is useful in a situation where we are implementing new methods or also in exception handling. It plays a role like a placeholder.

**Example**

```
months = ['January', 'June', 'March', 'April']  
for mon in months:  
    pass  
print(months)
```

**Output**

```
['January', 'June', 'March', 'April']
```

**range () and exit () functions.****range ()**

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

**The syntax is**

```
range(start, stop, step)
```



**Parameter Values**

Parameter	Description
<i>start</i>	Optional. An integer number specifying at which position to start. Default is 0
<i>stop</i>	Required. An integer number specifying at which position to stop (not included).
<i>step</i>	Optional. An integer number specifying the incrementation. Default is 1

**Example**

```
x = range(3, 6)
for n in x:
    print(n)
```

**Output**

```
3
4
5
```

**exit()**

exit() is a built-in function in the Python sys module that **allows us to end the execution of the program**. We can use the sys. exit() function whenever you want without worrying about code corruption.

The general syntax is

**exit()****Example**

```
for x in range(3, 10):
    print(x + 20)
    exit()
```

**Output:**

23

DIVYA