Unit I: Introduction

History of Python, Need of Python Programming, Applications, Basics of Python Programming, Using the IDLE, Running Python Scripts, Installation of Jupyter Notebook, Variables, Assignment, Keywords, Input-Output, Indentation, comments

Compiler	**Interpreter**
The compiler saves the Machine **Language in form of Machine Code on disks**.	The Interpreter **does not save the Machine Language**.
Compiled codes run **faster** than Interpreter.	Interpreted codes run **slower** than Compiler.
Any change in the source program after **the compilation requires recompiling the entire code.**	Any change in the source program during the translation **does not require retranslation of the entire code.**
Errors are displayed in Compiler after Compiling together at the current time.	Errors are **displayed in every single line.**
The compiler **can see code upfront which helps in running the code faster** because of performing Optimization.	The Interpreter **works by line working of Code, that's why Optimization is a little slower** compared to Compilers.
Execution of the program takes place **only after the whole program is compiled.**	Execution of the program happens after **every line is checked or evaluated.**
Compilers more often **take a large amount of time** for analyzing the source code.	In comparison, Interpreters **take less time for analyzing the source code.**
CPU utilization is more in the case of a Compiler.	**CPU utilization is less** in the case of a Interpreter.
The use of Compilers mostly happens **in Production Environment**.	The use of Interpreters is mostly **in Programming and Development Environments.**
C, **C++**, **C#**, etc are programming languages that are compiler-based.	**Python**, **Ruby**, **Perl**, **SNOBOL**, **MATLAB**, etc are programming languages that are interpreter-based.

What is Python

- Python **is an interpreted, general-purpose, high-level programming language with easy syntax and Dynamic semantics**.

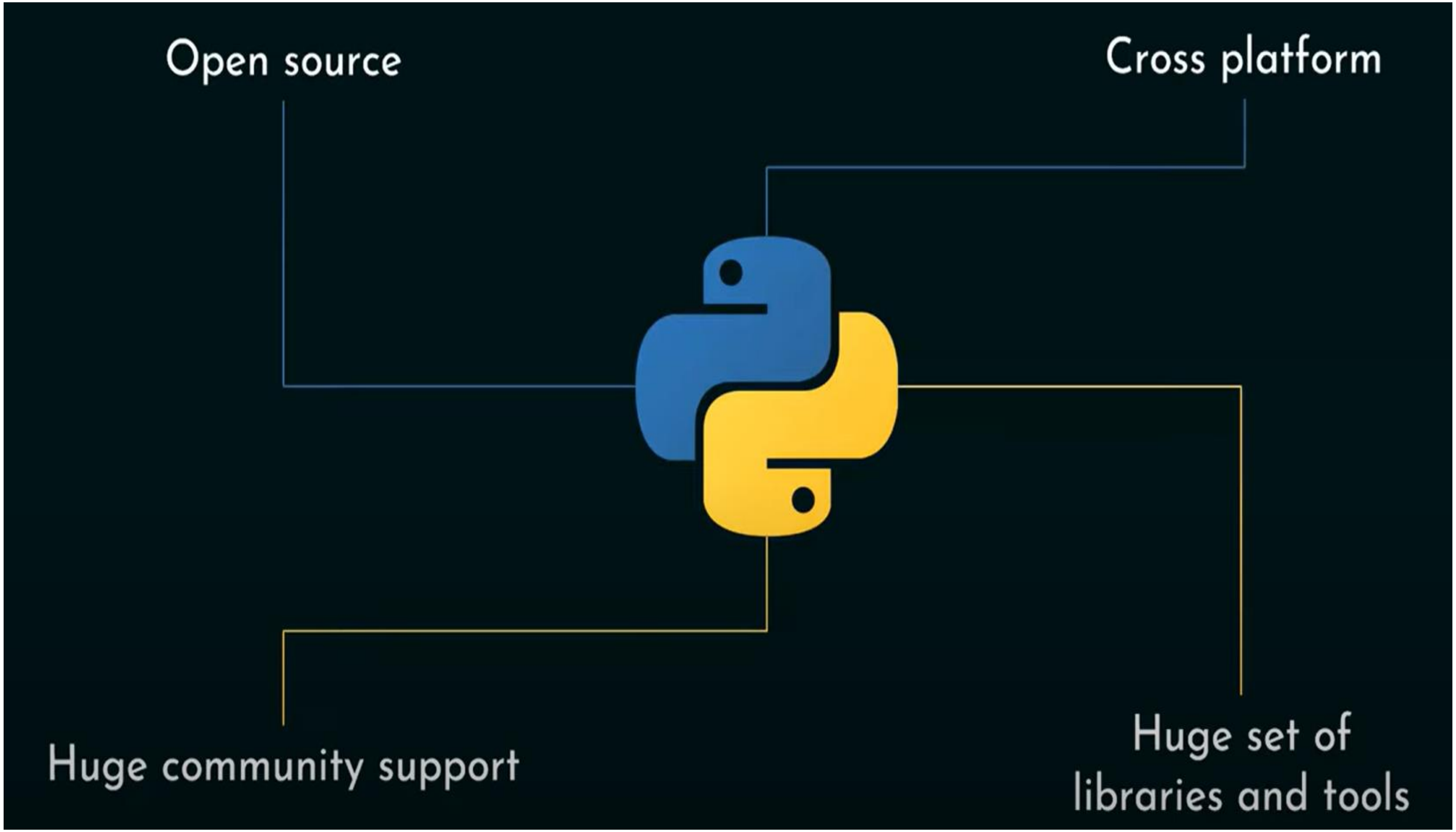
- Python is an easy-to-learn yet powerful and versatile scripting language, which makes it attractive for **Application Development**.

- With its interpreted nature, Python's syntax and dynamic typing make it an ideal language for **scripting and rapid application development**.

- Python supports multiple programming patterns, including **object-oriented, and functional or procedural programming styles.**

- Python is not intended to work in a particular area, such **as web programming**. It is a 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 has many **web-based assets, open-source projects, and a vibrant community**. Learning the language, working together on projects, and contributing to the Python ecosystem are all made very easy for developers.
- Python is an **open-source, cost-free programming language.** It is utilized in several sectors and disciplines as a result.


Open source
Cross platform
Huge community support
Huge set of libraries and tools

Python differences with other languages

Parameters	**Python**	**C++**
Code	Python has fewer lines of code	C++ tends to have long lines of code
Compilation	Python is interpreted	C++ is precompiled
Speed	It is slower since it uses an interpreter and also determines the data type at run time	It is faster once compiled as compared to python
Nature	It is dynamically typed.	is statically typed
Functions	Python Functions do not have restrictions on the type of the argument and the type of its return value	In C++, the function can accept and return the type of value which is already defined
Scope of Variable	Variables are accessible even outside the loop	The scope of variables is limited within the loops
Extension	Python programs are saved with the .py extension	C++ programs are saved with the .cpp extension
Popularity	It has huge community support. When it comes to popularity, beginner and novice programmers tend to turn to Python	It also has dedicated followings online. But only the people who have some experience in the field show a lot of interest in C++
Application Domain	Web development, data analysis, scientific computations, etc	Game development, embedded systems, etc
Variable Declaration	Variables are declared by simply writing their name with their datatype	While declaring a variable it is necessary to mention its datatype

TOPIC	C++	Java	Python
Compiled vs. Interpreted	Compiled Programming Language	Java is both Compiled and Interpreted.	Interpreted Programming Language
Platform Dependence	C++ is platform dependent	Java is platform-independent	Python is platform-independent
Operator Overloading	C++ supports operator overloading	Java does not support operator overloading	Python supports operator overloading
Inheritance	C++ provides both single and multiple inheritances	In Java, single inheritance is possible while multiple inheritances can be achieved using Interfaces	Python provides both single and multiple inheritances
Thread Support	C++ does not have built-in support for threads; It depends on Libraries	Java has built-in thread support	Python supports multithreading
Execution Time	C++ is very fast. It's, in fact, the first choice of competitive programmers	Java is much faster than Python in terms of speed of execution but slower than C++.	Due to the interpreter, Python is slow in terms of execution
Program Handling	Functions and variables are used outside the class	Every bit of code(variables and functions) has to be inside the class itself.	Functions and variables can be declared and used outside the class
Library Support	C++ has limited library support	Java provides library support for many concepts like UI	Python has a huge set of libraries and modules.
Code Length	Code length is lesser than Java, around 1.5 times less.	Java code length is bigger than Python and C++.	Python has a smaller code length

Python Basic Syntax

There is no use **of curly braces or semicolon** in Python programming language. It is English-like language. But Python **uses the indentation to define a block of code**. Indentation is nothing but **adding whitespace** before the statement when it is needed. **For example –**

```
def func():
      statement 1
      statement 2
       …………………
       …………………
       statement N
```

In the above example, the statements that are the same level to the right belong to the function. Generally, **we can use four whitespaces** to define indentation.

Instead of Semicolon as used in other languages, Python **ends its statements with a NewLine character**.

Python **is a case-sensitive language**, which means that uppercase and lowercase letters are treated differently. For example, 'name' and 'Name' are two different variables in Python.

In Python, comments can **be added using the '#' symbol**. Any text written after the '#' symbol is considered a comment and is ignored by the interpreter. This trick is useful for adding notes to the code or temporarily disabling a code block. It also helps in understanding the code better by some other developers.

'If', 'otherwise', 'for', 'while', 'try', 'except', and 'finally' are a few reserved keywords in Python that cannot be used as variable names. These terms are used in the language for particular reasons and have fixed meanings. If you use these keywords, your code may include errors, or the interpreter may reject them as potential new Variables.

Why learn Python?

Python provides many useful features to the programmer. These features make it the most popular and widely used language. We have listed below few-essential features of Python.

•**Easy to use and Learn:** Python has a simple and **easy-to-understand syntax**, unlike traditional languages like C, C++, Java, etc., making it easy for beginners to learn.

•**Expressive Language:** It allows programmers to express complex concepts in just **a few lines of code** or reduces Developer's Time.

•**Interpreted Language:** Python **does not require compilation**, allowing rapid development and testing. It uses Interpreter instead of Compiler.

•**Object-Oriented Language:** It supports **object-oriented programming**, making writing reusable and modular code easy.

•**Open Source Language:** Python is **open source and free to use**, distribute and modify.

•**Extensible:** Python can be **extended with modules** written in C, C++, or other languages.

•**Learn Standard Library:** Python's standard library contains **many modules and functions that can be used for various tasks,** such as string manipulation, web programming, and more.

•**GUI Programming Support:** Python provides several **GUI frameworks, such as Tkinter and PyQt, allowing developers to create desktop applications easily**.

•**Integrated:** Python can easily integrate with other languages and technologies, such as C/C++, Java, and . NET.

•**Embeddable:** Python code can be **embedded into other applications** as a scripting language.

•**Dynamic Memory Allocation:** Python **automatically manages memory allocation**, making it easier for developers to write complex programs without worrying about memory management.

•**Wide Range of Libraries and Frameworks:** Python has a **vast collection of libraries and frameworks**, such as NumPy, Pandas, Django, and Flask, that can be used to solve a wide range of problems.

•**Versatility:** Python is a universal language in various domains such as **web development, machine learning, data analysis, scientific computing, and more**.

•**Large Community:** Python has a vast and **active community of developers contributing to its development and offering support**. This makes it easy for beginners to get help and learn from experienced developers.

•**Career Opportunities:** Python is a highly popular language in the job market. Learning **Python can open up several career opportunities in data science, artificial intelligence, web development, and** more.

•**High Demand:** With the growing demand for automation and digital transformation, the need for Python developers is rising. Many industries seek skilled **Python developers to help build their digital infrastructure.**

•**Increased Productivity:** Python has a simple syntax and powerful libraries that can help developers write code **faster and more efficiently**. This can increase productivity and save time for developers and organizations.

•**Big Data and Machine Learning:** Python has become the go-to language for big data and machine learning. Python has become popular among data scientists and **machine learning engineers with libraries like NumPy, Pandas, Scikit-learn, TensorFlow, and more.**

Where is Python used?

Python is a general-purpose, popular programming language, and it is used in almost every technical field. The various areas of Python use are given below.

•**Data Science:** Data Science is a vast field, and Python is an important language for this field because of its simplicity, ease of use, and availability of powerful data analysis and **visualization libraries like NumPy, Pandas, and Matplotlib.**

•**Desktop Applications: PyQt and Tkinter** are useful libraries that can be used in GUI - Graphical User Interface-based Desktop Applications. There are better languages for this field, but it can be used with other languages for making Applications.

.

•**Mobile Applications:** While Python is not commonly used for creating mobile applications, it can still be combined **with frameworks like Kivy or BeeWare to create cross-platform mobile applications.**

.

•**Artificial Intelligence:** AI is an emerging Technology, and Python is a perfect language for artificial intelligence and machine learning because of the availability of powerful libraries such as TensorFlow, Keras, and PyTorch.

•**Web Applications:** Python is commonly used in **web development on the backend with frameworks like Django and Flask and on the front end with tools like JavaScript and HTML.**

•**Enterprise Applications:** Python can be used to develop **large-scale enterprise applications with features such as distributed computing, networking, and parallel processing**.

•**Finance:** Python has libraries **like Pandas, Scikit-learn, and Statsmodels** for financial modeling and analysis.

•**3D CAD Applications:** Python can be used for 3D computer-aided design (CAD) applications through libraries such as Blender.

•**Computer Vision or Image Processing Applications:** Python can be used for computer vision and image processing applications through powerful libraries such **as OpenCV and Scikit-image**.

•**Speech Recognition:** Python can be used for speech recognition applications through libraries **such as SpeechRecognition and PyAudio.**

•**Scientific computing:** Libraries **like NumPy, SciPy, and Pandas** provide advanced numerical computing capabilities for tasks like data analysis, machine learning, and more.

•**Testing:** Python is used for writing automated tests, providing frameworks **like unit tests and pytest that help write test cases and generate reports.**

•**Gaming:** Python has libraries **like Pygame,** which provide a platform for developing games using Python.

•**IoT:** Python is used in IoT for developing **scripts and applications for devices like Raspberry Pi, Arduino**, and others.

•**Networking:** Python is used in **networking for developing scripts and applications for network automation, monitoring, and management**.

•**DevOps:** Python is widely used in **DevOps for automation and scripting of infrastructure management,** configuration management, and deployment processes.
•**Audio and Music:** Python **has libraries like Pyaudio, which is used for audio processing**, synthesis, and analysis, and Music21, which is used for music analysis and generation.

Python Popular Frameworks and Libraries

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.

•**Web development (Server-side) -** Django Flask, Pyramid, CherryPy
•**GUIs based applications -** Tk, PyGTK, PyQt, PyJs, etc.

•**Machine Learning -** TensorFlow, PyTorch, Scikit-learn, Matplotlib, Scipy, etc.
•**Mathematics -** Numpy, Pandas, etc.

•**BeautifulSoup:** a library for web scraping and parsing HTML and XML
•**Requests:** a library for making HTTP requests

•**SQLAlchemy:** a library for working with SQL databases
•**Pygame:** a library for game development

•**Pytest:** a testing framework for Python
•**Django REST framework:** a toolkit for building RESTful APIs

•**FastAPI:** a modern, fast web framework for building APIs
•**Streamlit:** a library for building interactive web apps for machine learning and data science
•**NLTK:** a library for natural language processing

History of Python

•Python laid its foundation **in the late 1980s.**
•The implementation of Python was started in December 1989 by **Guido Van Rossum** at CWI in Netherland.
•In February 1991, **Guido Van Rossum** published the code (labeled version 0.9.0) to alt.sources.
•In 1994, Python 1.0 was released with new features like **lambda, map, filter, and reduce.**
•Python 2.0 added new features such as **list comprehensions, garbage collection systems.**
•On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flaw of the language.

Why the Name Python?
There is a fact behind choosing the name Python. **Guido van Rossum** was reading the script of a popular BBC comedy series "**Monty Python's Flying Circus**". It was late on-air 1970s.
Van Rossum wanted to select a name which unique, sort, and little-bit mysterious. So he decided to select naming Python after the "**Monty Python's Flying Circus**" for their newly created programming language.
The comedy series was creative and well random. It talks about everything. Thus it is slow and unpredictable, which made it very interesting.

Python is also versatile and widely used in every technical field, such as **Machine Learning, Artificial Intelligence, Web Development, Mobile Application, Desktop Application, Scientific Calculation, etc.**

Python Version List

Python programming language **is being updated regularly with new features and supports**. There are lots of update in Python versions, started from 1994 to current release.

A list of Python versions with its released date is given below.

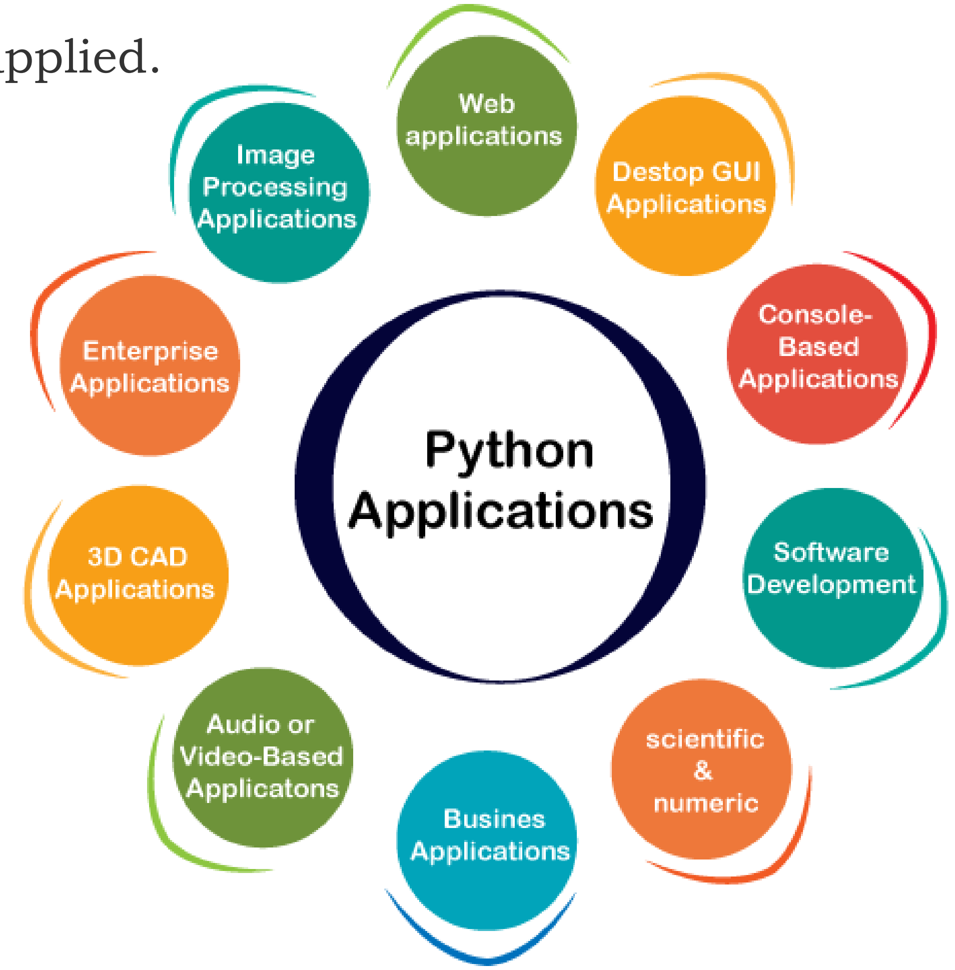
Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
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

Make it Clear Why We Want to Learn
The goal should be clear before learning the Python. Python is an easy, a vast language as well. It **includes numbers of libraries, modules, in-built functions and data structures**. If the goal is unclear then it will be a boring and monotonous journey of learning Python. Without any clear goal, you perhaps won't make it done.
So, first figure out the motivation behind learning, which can anything be such as knowing something new, develop projects using Python, switch to Python, etc. Below are the general areas where Python is widely used. Pick any of them.
•**Data Analysis and Processing**
•**Artificial Intelligence**
•**Games**
•**Hardware/Sensor/Robots**
•**Desktop Applications**

Explore Libraries and Frameworks
Python consists of **vast libraries and various frameworks**. After getting familiar with Python's basic concepts, the next step is to explore the Python libraries. Libraries are essential to work with the domain specific projects. In the following section, we describe the brief introduction of the main libraries.
•**TensorFlow -** It is an **artificial intelligence library** which allows us to **create large scale AI based projects**.
•**Django -** It is an **open source framework** that allows us to develop web applications.
•**Flask -** It is also an open source web framework. It is used to develop lightweight web applications.
•**Pandas -** It is a **Python library which is used to perform scientific computations**.
•**Keras -** It is an **open source library, which is used to work around the neural network**.

Python Applications

Python is known for its general-purpose nature that makes it applicable in almost every domain of software development. Python makes its presence in every emerging field. It is the fastest-growing programming language and can develop any application.
Here, we are specifying application areas where Python can be applied.



1) Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser, etc. **One of Python web-framework named Django is used on Instagram**. Python provides many useful frameworks, and these are given below:

•Django and Pyramid framework(Use for heavy applications)
•Flask and Bottle (Micro-framework)
•Plone and Django CMS (Advance Content management)

2) Desktop GUI Applications

The GUI stands for the Graphical User Interface, which provides a smooth interaction to any application. Python provides a **Tk GUI library to develop a user interface**. Some popular **GUI libraries** are given below.

- **Tkinter or Tk**
- **wxWidgetM**
- **Kivy (used for writing multitouch applications)**
- **PyQt or Pyside**

3) Console-based Application

Python can develop this kind of application very effectively. It is famous for having REPL, which means **the Read-Eval-Print Loop** that makes it the most suitable language for the command-line applications.
Python provides many free library or module which helps to build the command-line apps. The necessary **IO** libraries are used to read and write. It helps to parse argument and create console help text out-of-the-box. There are also advance libraries that can develop independent console apps.

4) Software Development

Python is useful for the software development process. It works as a support language and can be used to build control and management, testing, etc.

- **SCons** is used to build control.
- **Buildbot** and **Apache** Gumps are used for **automated continuous compilation and testing**.
- **Round** or **Trac** for **bug tracking and project management.**

5) Scientific and Numeric

Python language is the most suitable language for Artificial intelligence or machine learning. It consists of many scientific and mathematical libraries, which makes easy to solve complex calculations.
Implementing machine learning algorithms require complex mathematical calculation.

Python has many libraries for scientific and **numeric such as Numpy, Pandas, Scipy, Scikit-learn, etc**. If you have some basic knowledge of Python, you need to import libraries on the top of the code. Few popular frameworks of machine libraries are given below.

•SciPy
•Scikit-learn
•NumPy
•Pandas
•Matplotlib

6) Business Applications

Business Applications differ from standard applications. E-commerce and ERP are an example of a business application. This kind of application requires extensively, scalability and readability, and Python provides all these features.
Oddo is an example of the all-in-one Python-based application which offers a range of business applications. Python provides a **Tryton platform which is used to develop the business application**.

7) Audio or Video-based Applications

Python is flexible to perform multiple tasks and can be used to create **multimedia applications**. Some multimedia applications which are made by using Python are **TimPlayer, cplay,** etc. The few multimedia libraries are given below.

•Gstreamer
•Pyglet
•QT Phonon

8) 3D CAD Applications

The CAD (Computer-aided design) is used to design engineering related architecture. It is used to develop the 3D representation of a part of a system. **Python can create a 3D CAD application by using the following functionalities.**

•Fandango (Popular)
•CAMVOX
•HeeksCNC
•AnyCAD
•RCAM

9) Enterprise Applications

Python can be used to create applications that **can be used within an Enterprise or an Organization.** Some real-time applications are **OpenERP, Tryton, Picalo, etc**.

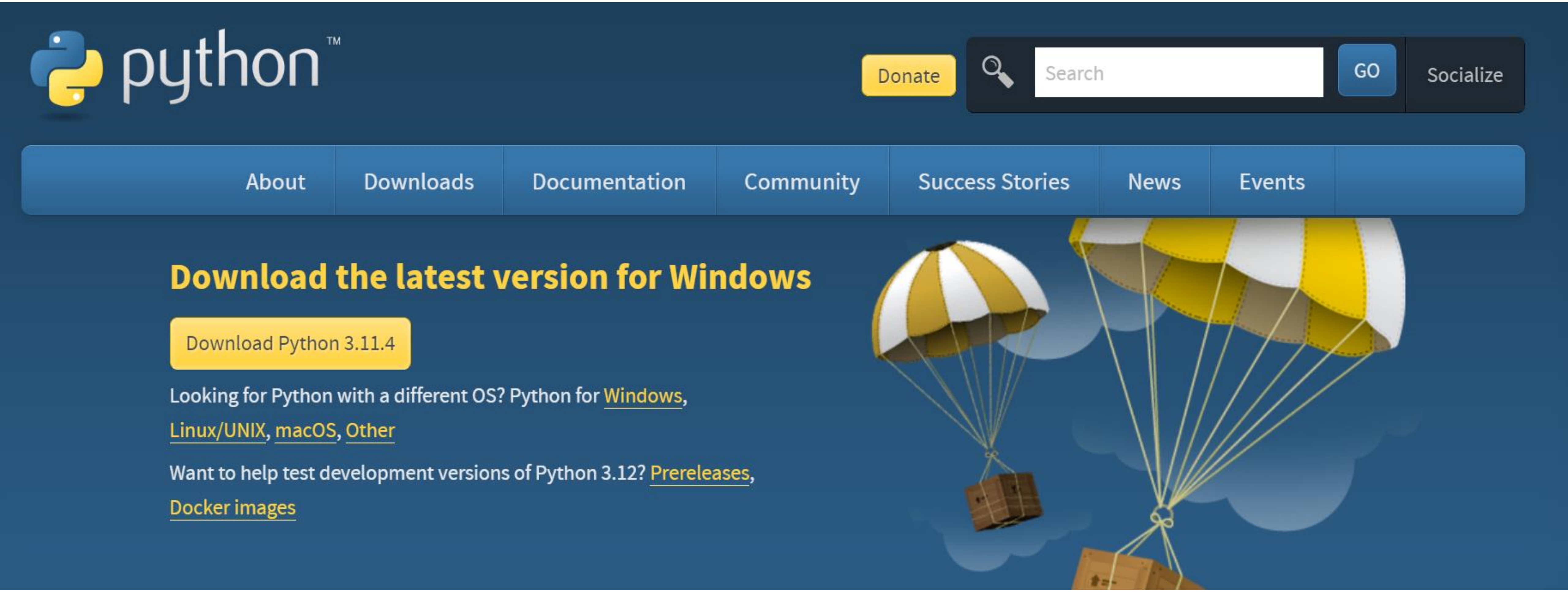
10) Image Processing Application

Python contains many libraries that are used to work with the image. The image can be **manipulated according to our requirements**. Some libraries of image processing are given below.

•OpenCV
•Pillow
•SimpleITK

Using the IDLE, Running Python Scripts

https://www.python.org/downloads/



Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python 3.12? Prereleases, Docker images

Comments

Types of Comments in Python
They are described below:

Single-Line Comments
Single-line remarks in Python have shown to be effective for providing quick descriptions for parameters, function definitions, and expressions. A single-line comment of Python is the one that has **a hashtag # at the beginning of it** and continues until the finish of the line. If the comment continues to the next line, add a hashtag to the subsequent line and resume the conversation. Consider the accompanying code snippet, which shows how to use a single line comment:
Code

```
# This code is to show an example of a single-line comment
print( 'This statement does not have a hashtag before it' )
```

Output:
```
This statement does not have a hashtag before it
```

The following is the comment:
```
# This code is to show an example of a single-line comment
```

The Python compiler ignores this line.
Everything following the # is omitted.

Multiline Comments
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!")
```

Or, not quite as intended, 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:**

Example

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

Installation of Jupyter Notebook(using pip)

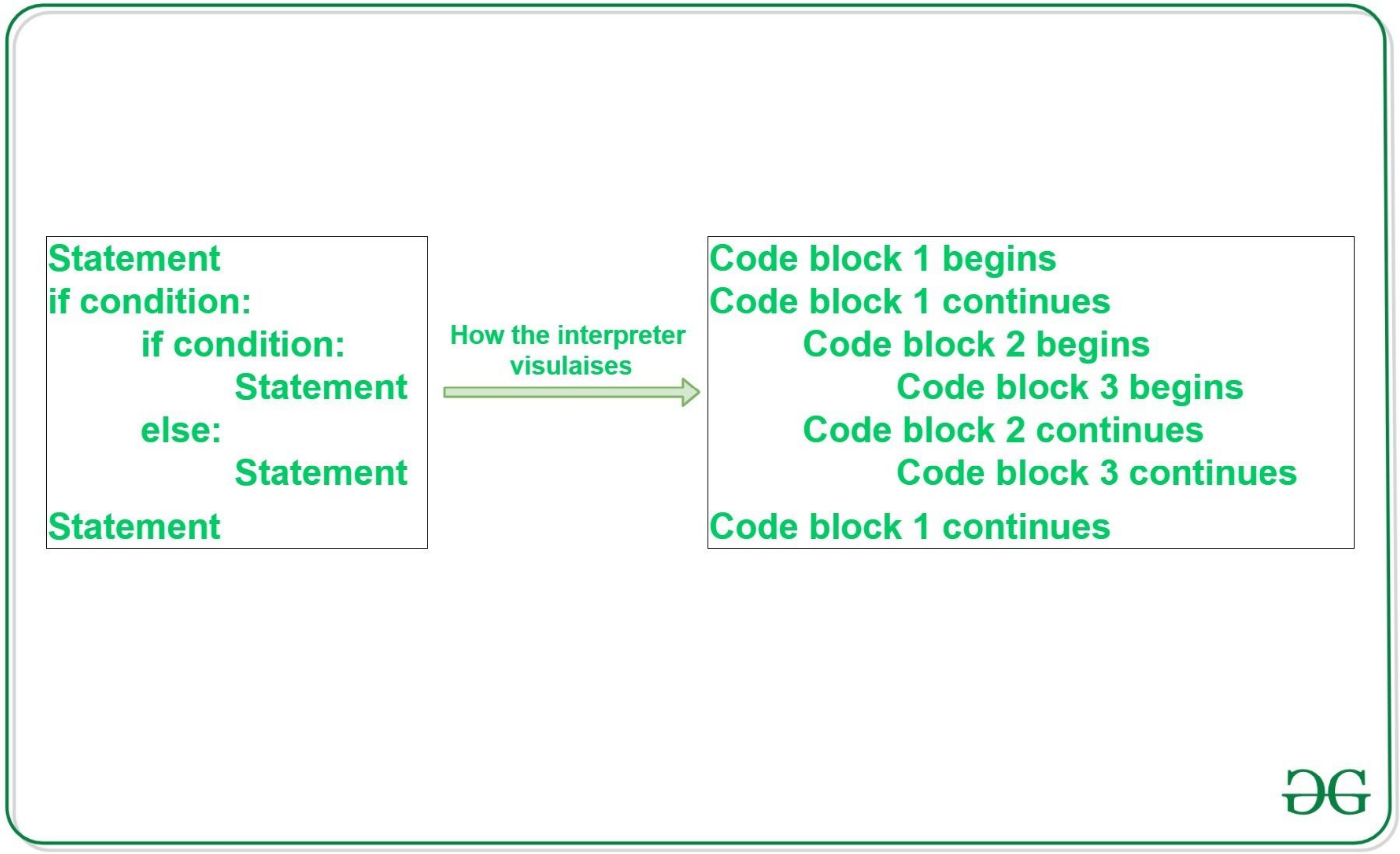
```
python -m pip install --upgrade pip
```

```
python -m pip install jupyter
```

```
jupyter notebook
```

Indentation

Python indentation refers to **adding white space before a statement to a particular block of code**. In another word, all the statements with the **same space to the right, belong to the same code block.**



```
Statement
if condition:
        if condition:
                Statement
        else:
                Statement
Statement
```

```
Code block 1 begins
Code block 1 continues
        Code block 2 begins
                Code block 3 begins
        Code block 2 continues
                Code block 3 continues
Code block 1 continues
```

Python indentation is a way of telling a Python interpreter that the group of statements **belongs to a particular block of code**. A block is a combination of all these statements. Block can be regarded as the grouping of statements for a specific purpose**. Most programming languages like C, C++, and Java use braces { } to define a block of code.** Python uses indentation to highlight the blocks of code. **Whitespace is used for indentation** in Python. All statements with the same distance to the right belong to the same block of code. **If a block has to be more deeply nested, it is simply indented further to the right**.

Variables

- Variable is a name that is used to refer **to memory location**. Python variable is also known as an identifier and **used to hold value**.
- In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to **get variable type**.
- Variable names can be a group of both the letters and digits, but they have to **begin with a letter or an underscore.**
- It is recommended to use lowercase letters for the variable name. **Rahul and rahul both are two different variables.**

Identifier Naming

Variables are the example of identifiers. An Identifier is used **to identify the literals** used in the program. The rules to name an identifier are given below.

- The first character of the variable must be an **alphabet or underscore (_)**.
- All the characters except the first **character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).**
- Identifier name must **not contain any white-space**, or **special character (!, @, #, %, ^, &, *).**
- Identifier name must **not be similar to any keyword** defined in the language.

- Identifier names are **case sensitive**; for example, my name, and MyName is not the same.
- Examples **of valid identifiers: a123, _n, n_9, etc**.
- Examples of **invalid identifiers: 1a, n%4, n 9, etc.**

Declaring Variable and Assigning Values

Python does not bind us to declare a variable before using it in the application. It allows us to create a variable at the required time.
We don't need to declare explicitly variable in Python. **When we assign any value to the variable, that variable is declared automatically.**
The equal (=) operator is used to assign value to a variable.

Object References

Python is **the highly object-oriented programming language**; that's why **every data item belongs to a specific type of class.** Consider the following example.

```
print("John")
```
Output:
```
John
```

The Python object creates an integer object and displays it to the console. In the above print statement, we have created a string object. **Let's check the type of it using the Python built-in type() function.**

```
type("John")
```

Output:

```
<class 'str'>
```

In Python, variables are a **symbolic name that is a reference or pointer to an object**. The variables are used **to denote objects by that name.**

Object Identity

In Python, every created object identifies uniquely in Python. **The built-in id() function, is used to identify the object identifier.** Consider the following **example.**

```
a = 50
b = a
print(id(a))
print(id(b))
# Reassigned variable a
a = 500
print(id(a))
```

Output:

```
140734982691168
140734982691168
2822056960944
```

We assigned the b = a, a and b both point to the same object. When we checked by the id() function it returned the same number. We reassign a to 500; then it referred to the new object identifier.

Variable Names

Variable names can be any length can have uppercase, lowercase (A to Z, a to z), the digit (0-9), and underscore character(_). Consider the following example of valid variables names.

Consider the following valid variables name.

```
name = "A"
Name = "B"
naMe = "C"
NAME = "D"
n_a_m_e = "E"
_name = "F"
name_ = "G"
_name_ = "H"
na56me = "I"

print(name,Name,naMe,NAME,n_a_m_e, NAME, n_a_m_e, _name, name_,_name, na56me)
```

Output:

```
A B C D E D E F G F I
```

Multiple Assignment
Python allows us to assign a value to multiple variables in a single statement, which is also known as multiple assignments.

We can apply multiple assignments in two ways, **either by assigning a single value to multiple variables or assigning multiple values to multiple variables.** Consider the following example.

1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
print(x)
print(y)
print(z)
```

Output:

```
50
50
50
```

2. Assigning multiple values to multiple variables:

Eg:

```
a,b,c=5,10,15
print a
print b
print c
```

Output:

```
5
10
15
```

The **values will be assigned in the order** in which variables appear.

Python Variable Types

There are two types of variables in Python - Local variable and Global variable. Let's understand the following variables.

Local Variables

Local variables are the variables that **declared inside the function and have scope within the function**. Let's understand the following example.

```
def greet():

    # local variable
    message = 'Hello'

    print('Local', message)

greet()

# try to access message variable
# outside greet() function
print(message)
```

Output:

```
Local Hello
NameError: name 'message' is not defined
```

Global Variables

Global variables **can be used throughout the program**, and its **scope is in the entire program**. We can use global variables inside or outside the function.

A variable declared outside the function is the global variable by default. **Python provides the global keyword to use global variable inside the function.** If **we don't use the global keyword, the function treats it as a local variable.** Let's understand the following example.

Example –

```
# declare global variable
global message = 'Hello'

def greet():
    # declare local variable
    print('Local', message)

greet()
print('Global', message)
```

Output:

```
Local Hello
Global Hello
```

Delete a variable

We can delete the variable **using the del keyword**. The syntax is given below.

Syntax -

```
del <variable_name>
```

In the following example, we create a variable x and assign value to it. We deleted variable x, and print it, we get the error "variable x is not defined". The variable x will no longer use in future.

Example -

```
# Assigning a value to x
x = 6
print(x)
# deleting a variable.
del x
print(x)
```

Output:

```
6
Traceback (most recent call last):
  File "C:/Users/arshp/Python/Hello/delete.py", 
line 389, in
    print(x)
NameError: name 'x' is not defined
```

Maximum Possible Value of an Integer in Python

Unlike the other programming languages, Python doesn't have long int or float data types. It treats **all integer values as an int data type**. Here, the question arises. **What is the maximum possible value can hold by the variable in Python?**

Example -

```
# A Python program to display that we can store
# large numbers in Python

a = 100000000000000000000000000000000000000000000
a = a + 1
print(type(a))
print (a)
```

Output:

```
<class 'int'>
100000000000000000000000000000000000000000001
```

As we can see in the above example, we assigned a large integer value to variable x and checked its type. It printed class <int> not long int. Hence, there is no limitation number by bits and we can expand to the limit of our memory.

Python doesn't have any special data type to store larger numbers.

Print Single and Multiple Variables in Python

We can print multiple variables within the single print statement. Below are the example of single and multiple printing values.

Example - 1 (Printing Single Variable)

```
# printing single value
a = 5
print(a)
print((a))
```

Output:

```
5
5
```

Example - 2 (Printing Multiple Variables)

```
a = 5
b = 6
# printing multiple variables
print(a,b)
# separate the variables by the comma
Print(1, 2, 3, 4, 5, 6, 7, 8)
```

Output:

```
5 6
1 2 3 4 5 6 7 8
```

Basic Fundamentals:
This section contains the fundamentals of Python, such as:

Tokens and their types.

a)Tokens:

The tokens can be defined as a **punctuator mark, reserved words, and each word in a statement.**
The token is the smallest unit inside the given program.
There are following tokens in Python:

Identifiers.
Literals.
Operators.
Keywords.

Python Operators

Python also has some operators, and these are given below -

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators
- Arithmetic Operators

Arithmetic Operators

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 10, b = 10 => a+b = 20
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 5 => a - b = 15
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 4 => a * b = 80
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
**** (Exponent)**	As it calculates the first operand's power to the second operand, it is an exponent operator.
// (Floor division)	It provides the quotient's floor value, which is obtained by dividing the two operands.

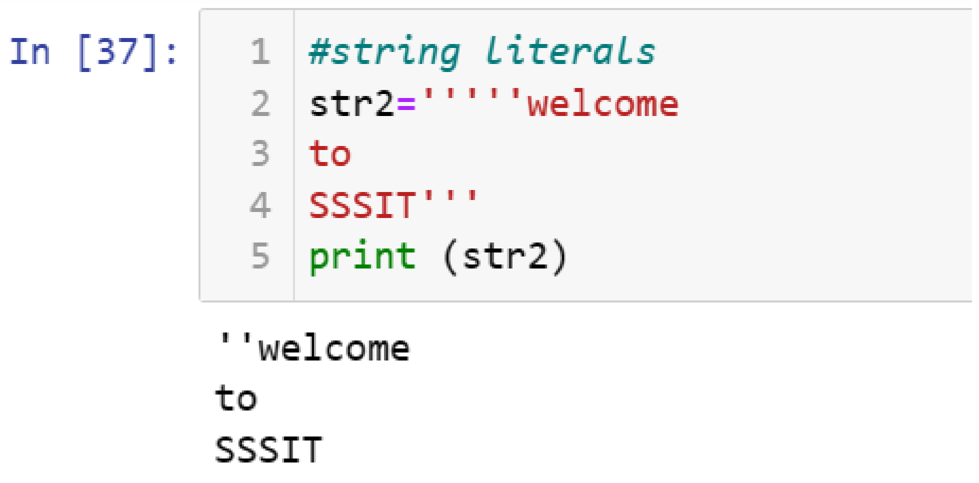
Comparison operator

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal, then the condition becomes true.
<=	The condition is met if the first operand is smaller than or equal to the second operand.
>=	The condition is met if the first operand is greater than or equal to the second operand.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Operator Precedence

Operator	Description
**	Overall other operators employed in the expression, the exponent operator is given precedence.
~ + -	the minus, unary plus, and negation.
* / % //	the division of the floor, the modules, the division, and the multiplication.
+ -	Binary plus, and minus
>> <<	Left shift. and right shift
&	Binary and.
^ |	Binary xor, and or
<= < > >=	Comparison operators (less than, less than equal to, greater than, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

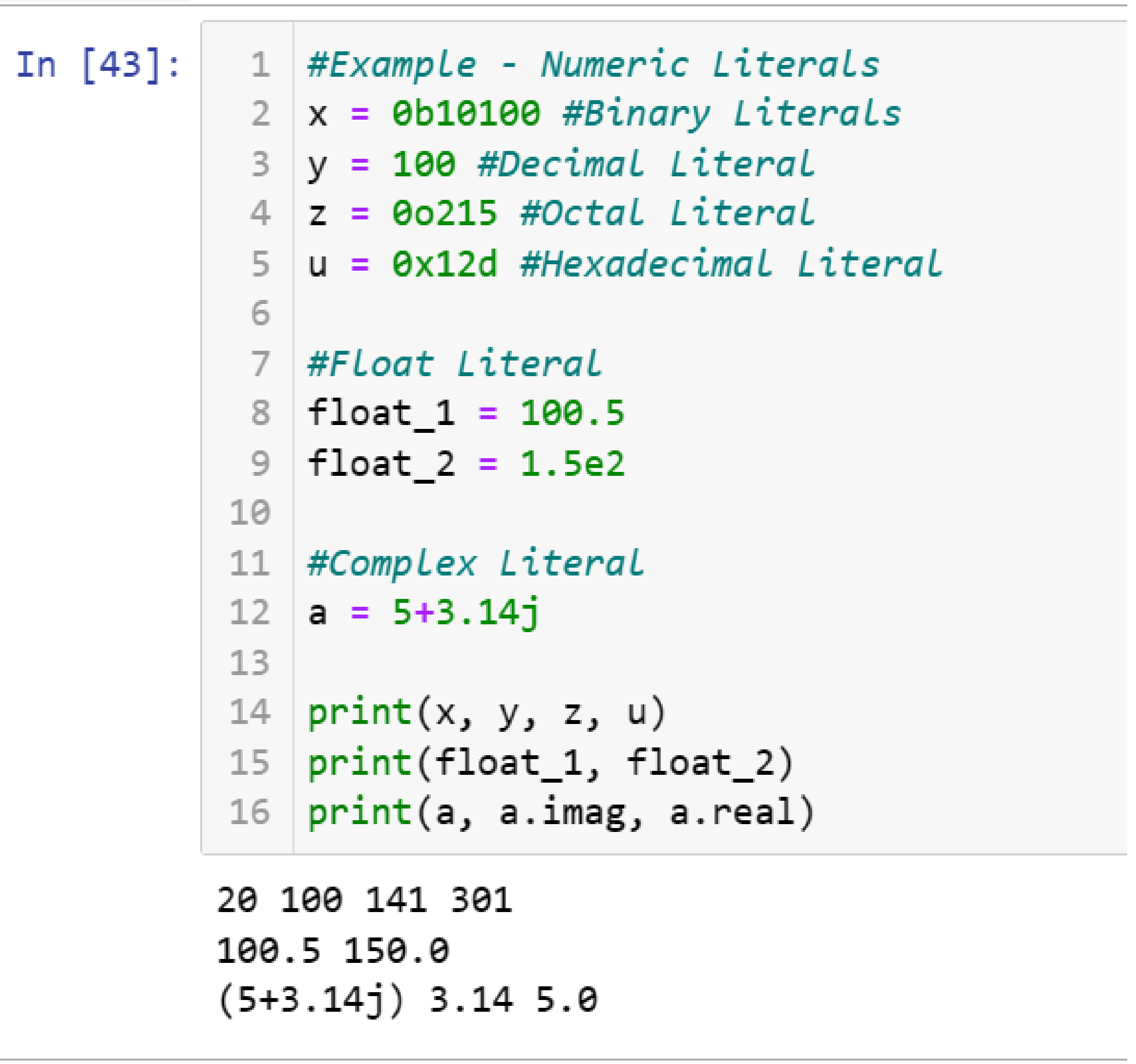
Python Literals



In [37]:

```
#string literals
str2='''''welcome
to
SSSIT'''
print (str2)
```

```
''welcome
to
SSSIT
```



In [43]:

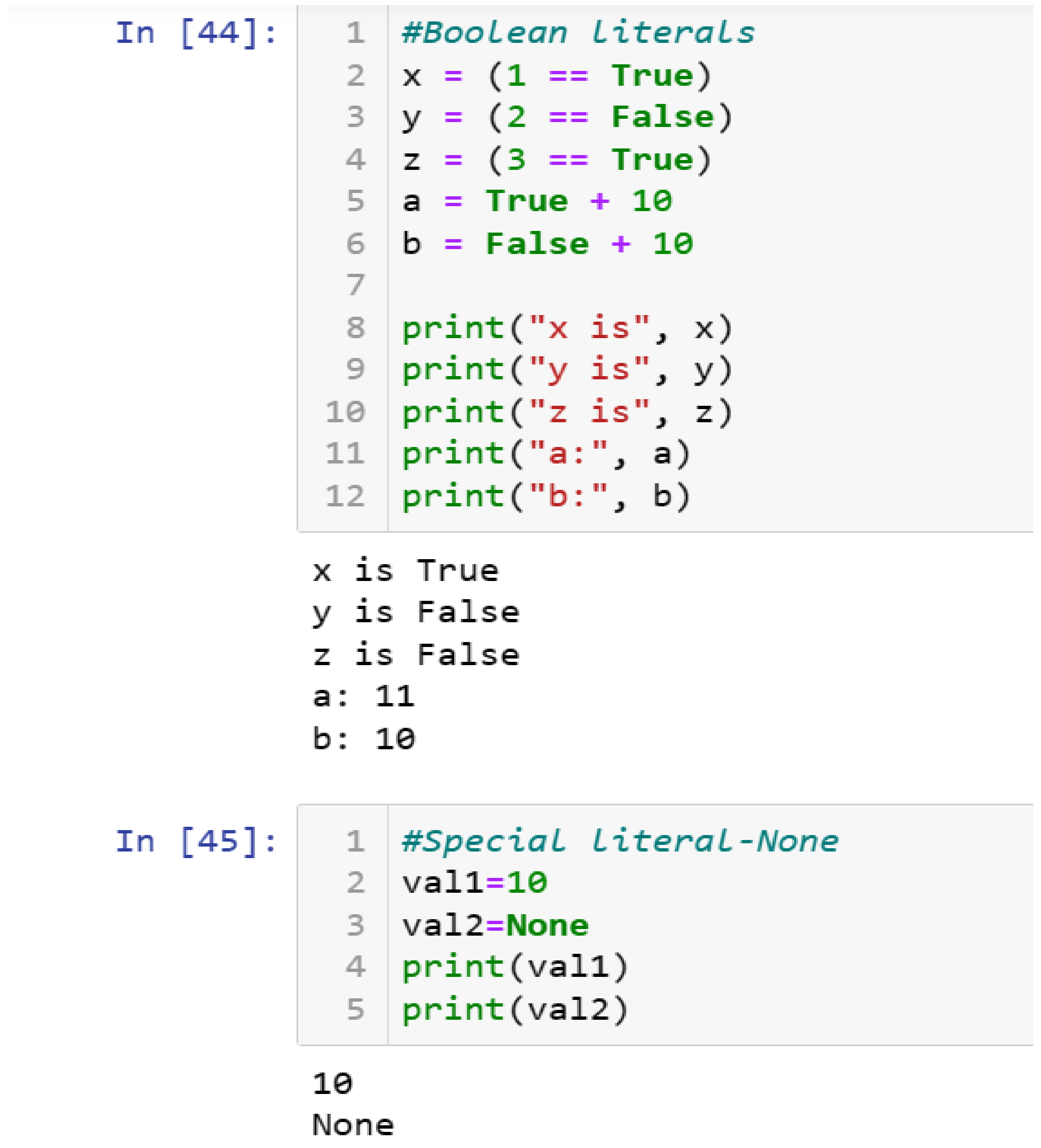
```
#Example - Numeric Literals
x = 0b10100 #Binary Literals
y = 100 #Decimal Literal
z = 0o215 #Octal Literal
u = 0x12d #Hexadecimal Literal

#Float Literal
float_1 = 100.5
float_2 = 1.5e2

#Complex Literal
a = 5+3.14j

print(x, y, z, u)
print(float_1, float_2)
print(a, a.imag, a.real)
```

```
20 100 141 301
100.5 150.0
(5+3.14j) 3.14 5.0
```



In [44]:

```python
#Boolean literals
x = (1 == True)
y = (2 == False)
z = (3 == True)
a = True + 10
b = False + 10

print("x is", x)
print("y is", y)
print("z is", z)
print("a:", a)
print("b:", b)
```

```
x is True
y is False
z is False
a: 11
b: 10
```

In [45]:

```python
#Special literal-None
val1=10
val2=None
print(val1)
print(val2)
```

```
10
None
```

Keywords

Python Keywords

Every scripting language has designated words or keywords, with particular definitions and usage guidelines. Python is no exception. The fundamental constituent elements of any Python program are Python keywords.

Introducing Python Keywords

Python keywords are **unique words reserved with defined meanings** and functions that we can only apply for those functions.

Assigning a particular meaning to Python keywords means **you can't use them for other purposes in our code**. You'll **get a message of SyntaxError if you attempt to do the same.**

Python contains **thirty-five keywords** in the most recent version, i.e., Python 3.11. Here we have shown a complete list of Python keywords for the reader's reference.

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

Input-Output

Sometimes a developer might want to take user input at some point in the program. To do this Python provides an **input() function**.

Syntax:

```
input('prompt')
```

where **prompt is an optional string** that is displayed on the string at the time of taking input.

Example 1: Python get user input with a message

```
# Taking input from the user
name = input("Enter your name: ")
print("Hello" + name)
print("Hello" , name)
print(type(name))
```

Output:

```
Enter your name: GFG
HelloGFG
Hello GFG
<class 'str'>
```

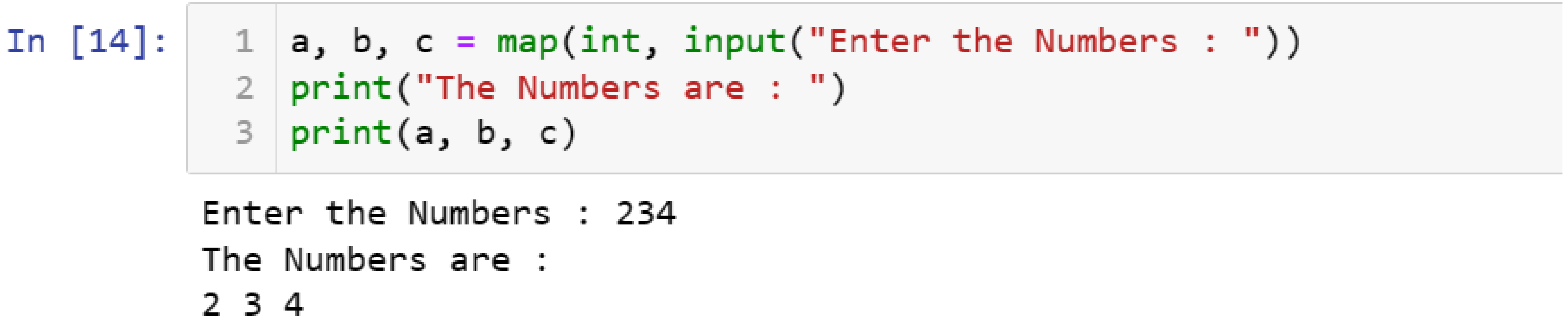
Example 2: Integer input in Python

```
# Taking input from the user as integer
num = int(input("Enter a number: "))
add = num + 1
print(add)
```

Output:

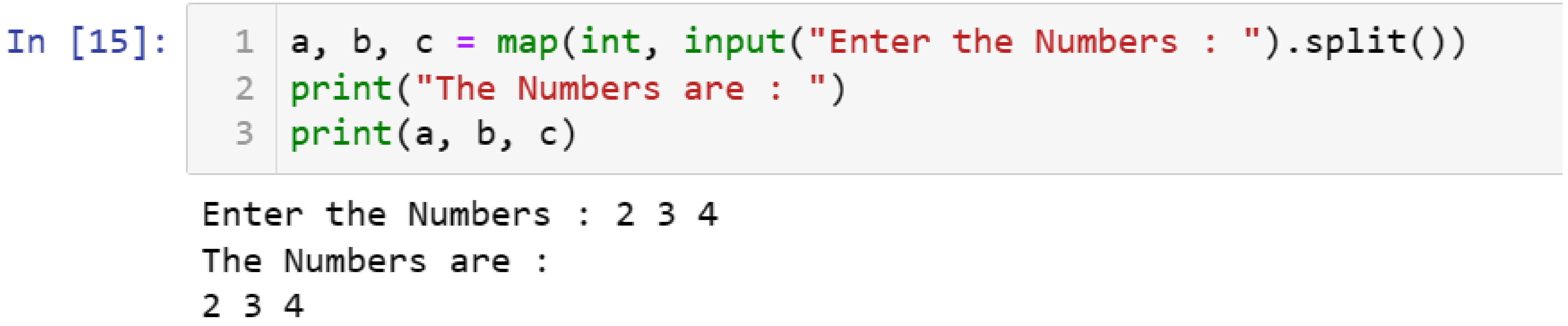
```
Enter a number: 25
26
```

How to take Multiple Inputs in Python :
we can take multiple inputs of the same data type at a time in python, **using map() method** in python.



```
In [14]:
a, b, c = map(int, input("Enter the Numbers : "))
print("The Numbers are : ")
print(a, b, c)
```

```
Enter the Numbers : 234
The Numbers are :
2 3 4
```



```
In [15]:
a, b, c = map(int, input("Enter the Numbers : ").split())
print("The Numbers are : ")
print(a, b, c)
```

```
Enter the Numbers : 2 3 4
The Numbers are :
2 3 4
```

How to Display Output in Python
Python provides the print() function to display output to the standard output devices.

Syntax: print(value(s), sep= ‘ ‘, end = ‘\n’)

Parameters:
value(s) : Any value, and as many as you like. Will be converted to string before printed
sep=’separator’ : (Optional) Specify how to separate the objects, if there is more than one.Default :’ ‘
end=’end’: (Optional) Specify what to print at the end.Default : ‘\n’

Example: Python Print Output
```
# Python program to demonstrate
# print() method
print("GFG")
print('G', 'F', 'G')
```

Output
```
GFG
G F G
```
In the above example, we can see that in the case of the 2nd print statement there is a space between every letter and the print statement always add a new line character at the end of the string. This is because after every character the sep parameter is printed and at the end of the string the end parameter is printed. Let’s try to change this sep and end parameter.

Example: Python Print output with custom sep and end parameter

```
# Python program to demonstrate
# print() method
print("GFG", end = "@")

# code for disabling the softspace feature
print('G', 'F', 'G', sep="#")
```

Output

```
GFG@G#F#G
```

Formatting Output

Formatting output in Python can be done in many ways.

Using formatted string literals

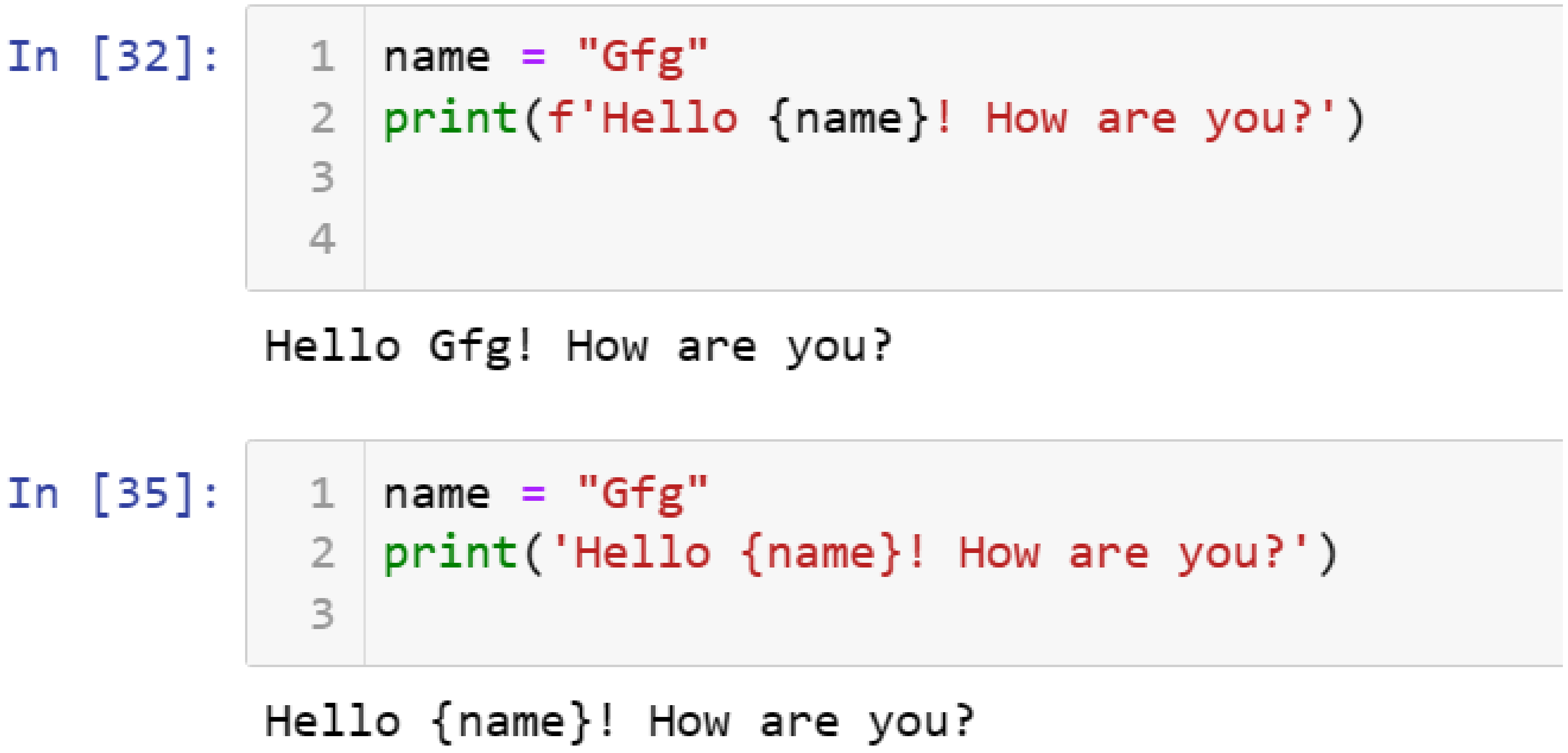
We can use formatted string literals, by starting a string with f or F before opening quotation marks or triple quotation marks. In this string, we can write Python expressions between { and } that can refer to a variable or any literal value.

Example: Python String formatting using F string

```
name = "Gfg"
print(f'Hello {name}! How are you?')
```

Output:

```
Hello Gfg! How are you?
```



```
In [32]:
name = "Gfg"
print(f'Hello {name}! How are you?')
```

```
Hello Gfg! How are you?
```

```
In [35]:
name = "Gfg"
print('Hello {name}! How are you?')
```

```
Hello {name}! How are you?
```

Using format()

We can also use **format() function** to format our output to make it look presentable. The curly braces { } **work as placeholders**. We can specify the order in which variables occur in the output.

Example: Python string formatting using format() function

```
# Initializing variables
a = 20
b = 10
sum = a + b
sub = a- b
print('The value of a is {} and b is {}'.format(a,b))
print('{2} is the sum of {0} and {1}'.format(a,b,sum))
print('{sub_value} is the subtraction of {value_a} and {value_b}'.format(value_a = a ,value_b =
b, sub_value = sub))
```

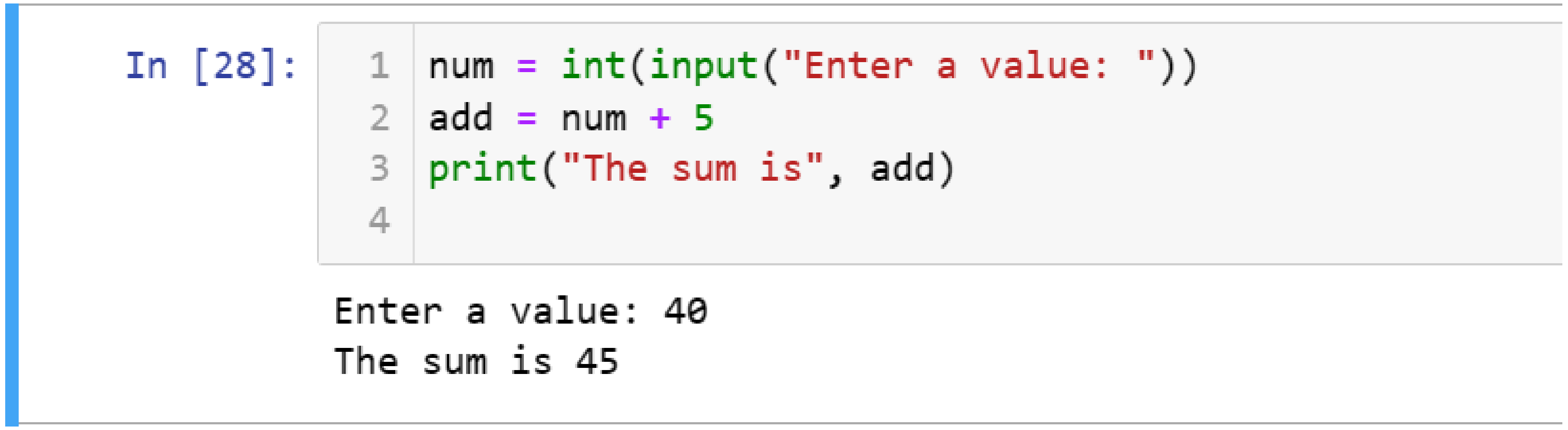
Output:

```
The value of a is 20 and b is 10
30 is the sum of 20 and 10
10 is the subtraction of 20 and 10
```

Using % Operator
We can use '%' operator. **% values are replaced with zero or more value of elements**. The formatting using % is similar to that of 'printf' in the C programming language.

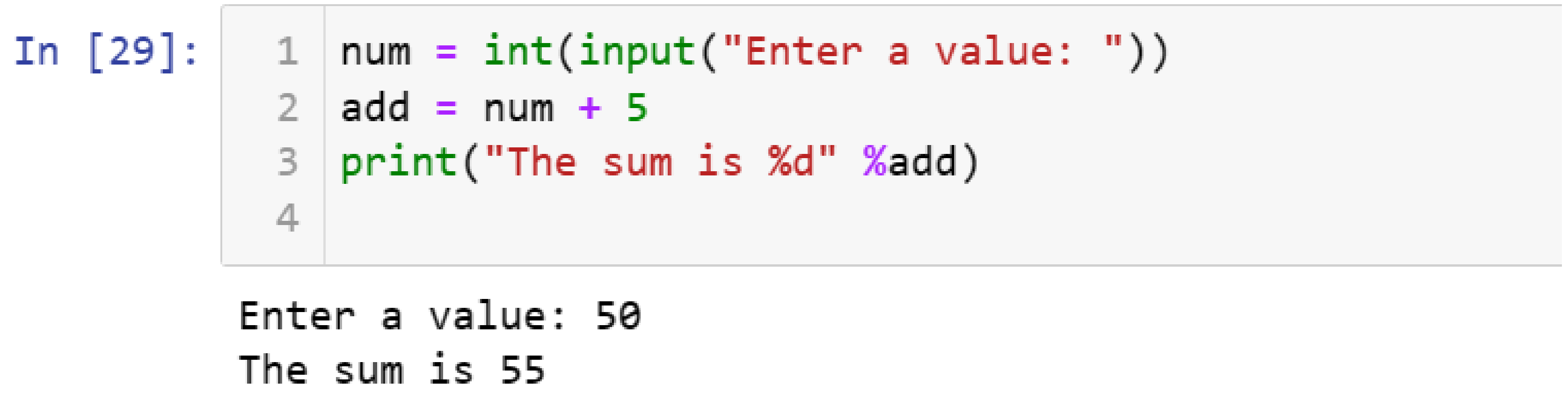
%d – integer
%f – float
%s – string
%x – hexadecimal
%o – octal

Example:



```
In [28]:
num = int(input("Enter a value: "))
add = num + 5
print("The sum is", add)

Enter a value: 40
The sum is 45
```



```
In [29]:
num = int(input("Enter a value: "))
add = num + 5
print("The sum is %d" %add)

Enter a value: 50
The sum is 55
```