Python Tutorial | Python Programming Language

Python is a general-purpose, dynamically typed, high-level, compiled and interpreted, garbage-collected, and purely object-oriented programming language that supports procedural, object-oriented, and functional programming.

Features of Python:

- **Easy to use and Read -** Python's syntax is clear and easy to read, making it an ideal language for both beginners and experienced programmers.
- **Dynamically Typed** - The data types of variables are determined during run-time. We do not need to specify the data type of a variable during writing codes.
- **High-level** - High-level language means human readable code.
- **Compiled and Interpreted** - Python code first gets compiled into bytecode, and then interpreted line by line.
- **Garbage Collected** - Memory allocation and de-allocation are automatically managed. Programmers do not specifically need to manage the memory.
- **Purely Object-Oriented** - It refers to everything as an object, including numbers and strings.
- **Cross-platform Compatibility** - Python can be easily installed on Windows, macOS, and various Linux distributions, allowing developers to create software that runs across different operating systems.
- **Rich Standard Library** - Python comes with several standard libraries that provide ready-to-use modules and functions for various tasks, ranging from **web development** and **data manipulation** to **machine learning** and **networking**.
- **Open Source** - Python is an open-source, cost-free programming language. It is utilized in several sectors and disciplines as a result.

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.

- **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.
- **Increased Productivity:** Python has a simple syntax and powerful libraries that can help developers write code faster and more efficiently.
- **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.
- **Console-based Applications:** Python is also commonly used to create command-line or console-based applications because of its ease of use and support for advanced features such as input/output redirection and piping.
- **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.
- **Software Development:** Python is considered one of the best software-making languages. Python is easily compatible with both from Small Scale to Large Scale software.

- **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 HTML and CSS.
- **Enterprise Applications:** Python can be used to develop large-scale enterprise applications with features such as distributed computing, networking, and parallel processing.
- **3D CAD Applications:** Python can be used for 3D computer-aided design (CAD) applications through libraries such as Blender.
- **Machine Learning:** Python is widely used for machine learning due to its simplicity, ease of use, and availability of powerful machine learning libraries.
- **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.
- **Education:** Python's easy-to-learn syntax and availability of many resources make it an ideal language for teaching programming to beginners.
- **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.
- **Finance:** Python has libraries like Pandas, Scikit-learn, and Statsmodels for financial modeling and analysis.
- **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.
- **Writing scripts:** Python is used for writing utility scripts to automate tasks like file operations, web scraping, and data processing.

Python History and Versions

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in December 1989 by **Guido Van Rossum** at Centrum Wiskunde & Informatica (CWI) in Netherland.
- In February 1991, **Guido Van Rossum** published the code (labelled version 0.9.0) to alt.sources.
- *ABC programming language* is said to be the predecessor of Python language, which was capable of Exception Handling and interfacing with the Amoeba Operating System.
- The following programming languages influence Python:
  - ABC language.
  - Modula-3

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**".

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.

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

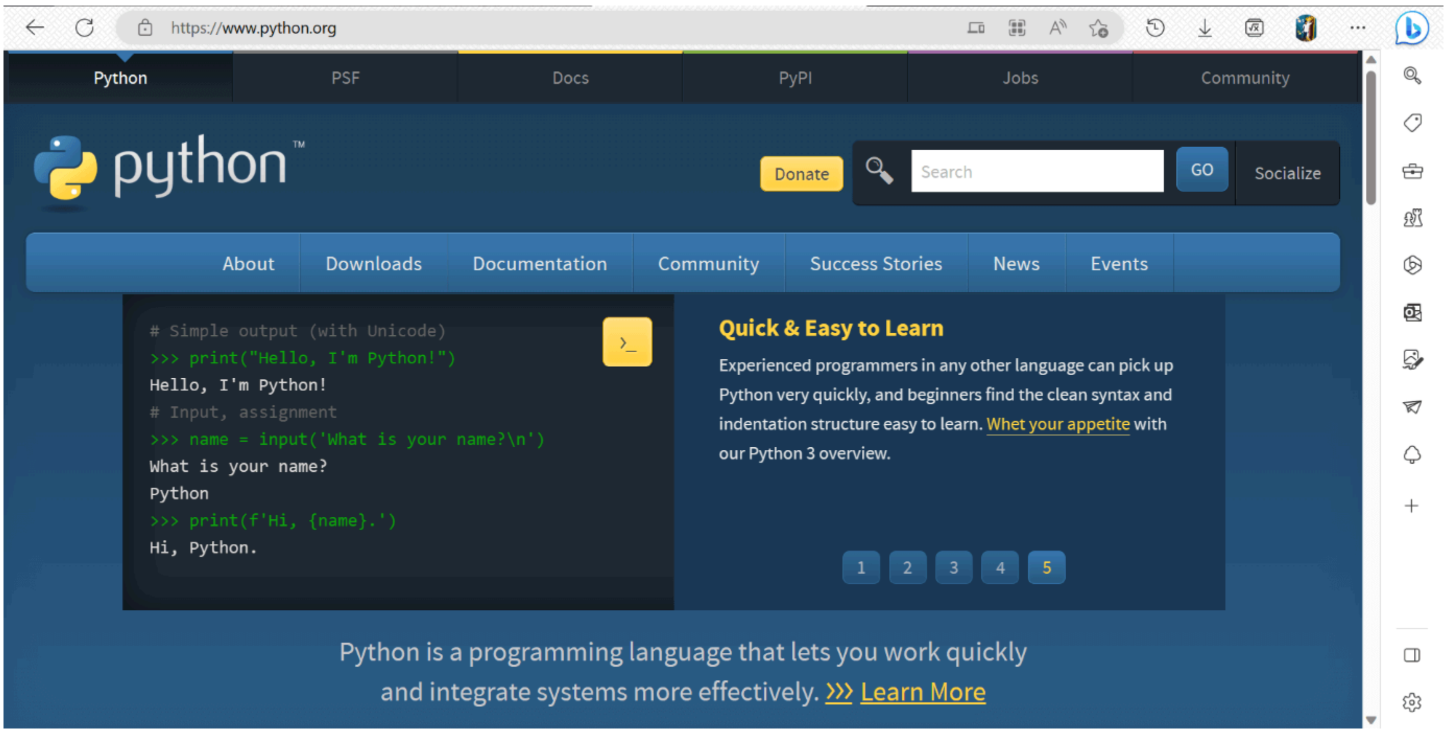
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	June 27, 2018
Python 3.8	October 14, 2019

How to Install Python

In order to become Python developer, the first step is to learn how to install or update

Installation on Windows

Visit the link ***https://www.python.org*** to download the latest release of Python. In this process, we will install Python **3.11.3** on our Windows operating system. When we click on the above link, it will bring us the following page.



```
# Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!
# Input, assignment
>>> name = input('What is your name?\n')
What is your name?
Python
>>> print(f'Hi, {name}.')
Hi, Python.
```

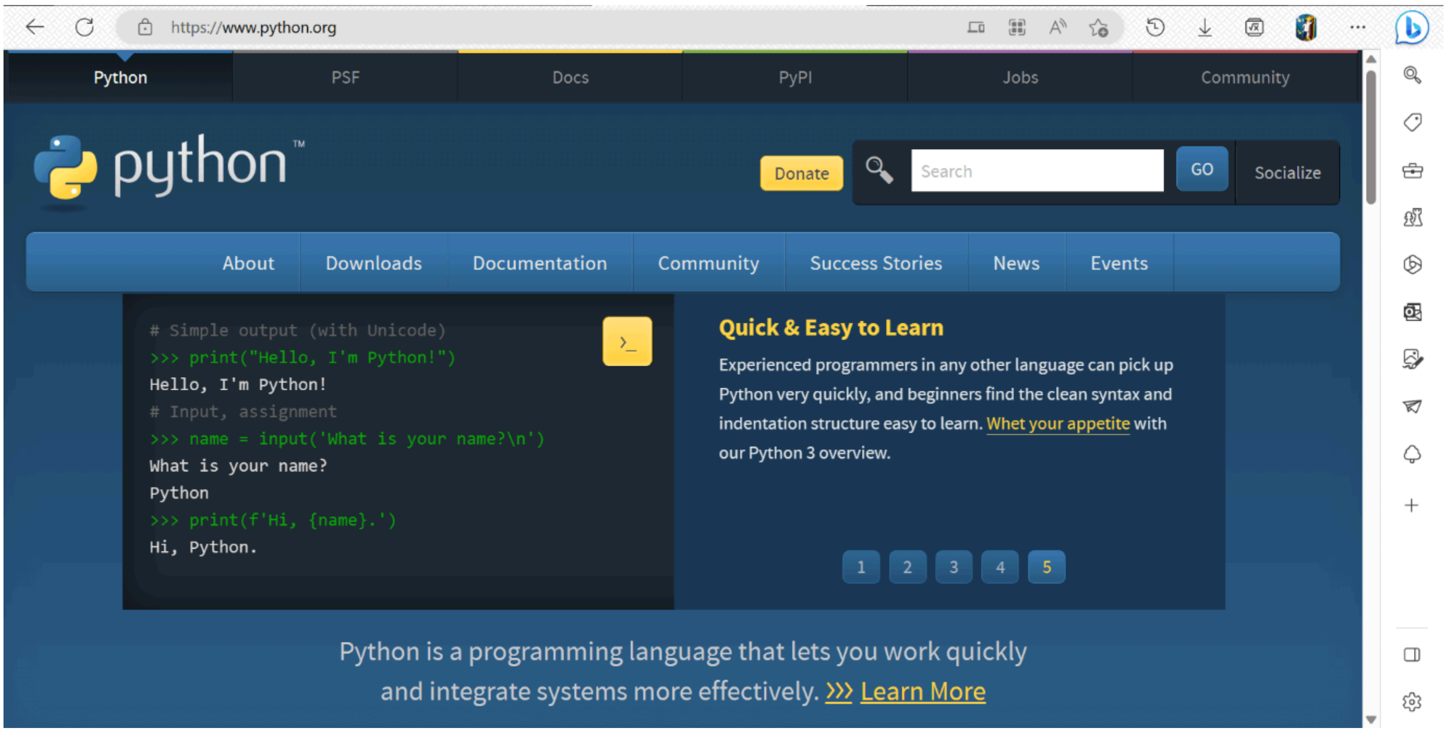
Quick & Easy to Learn

Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. Whet your appetite with our Python 3 overview.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Step - 1: Select the Python's version to download.

Click on the download button to download the exe file of Python.



```
# Simple output (with Unicode)
>>> print("Hello, I'm Python!")
Hello, I'm Python!
# Input, assignment
>>> name = input('What is your name?\n')
What is your name?
Python
>>> print(f'Hi, {name}.')
Hi, Python.
```

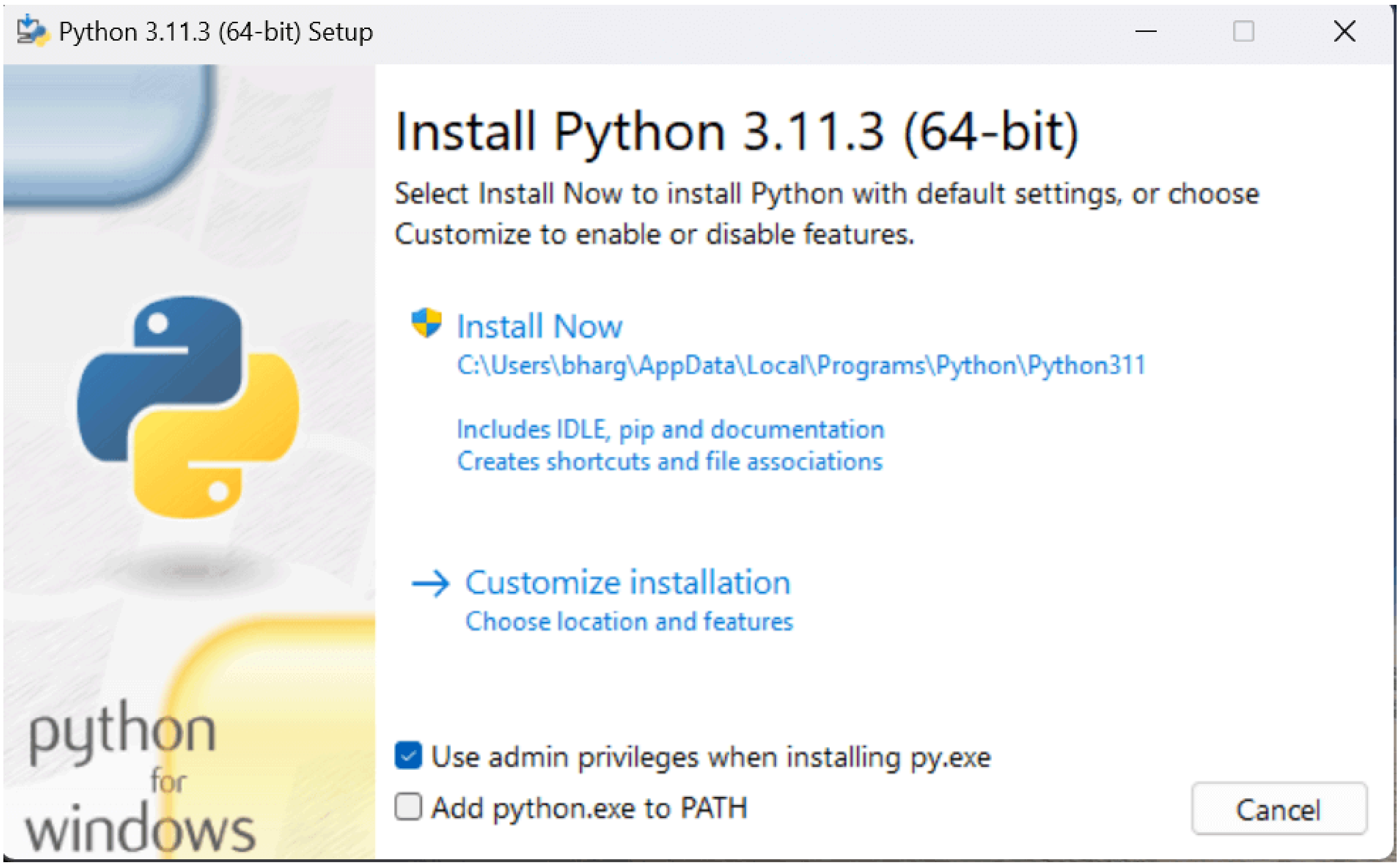
Quick & Easy to Learn

Experienced programmers in any other language can pick up Python very quickly, and beginners find the clean syntax and indentation structure easy to learn. Whet your appetite with our Python 3 overview.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Step - 2: Click on the Install Now

Double-click the executable file, which is downloaded.



Python 3.11.3 (64-bit) Setup

Install Python 3.11.3 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

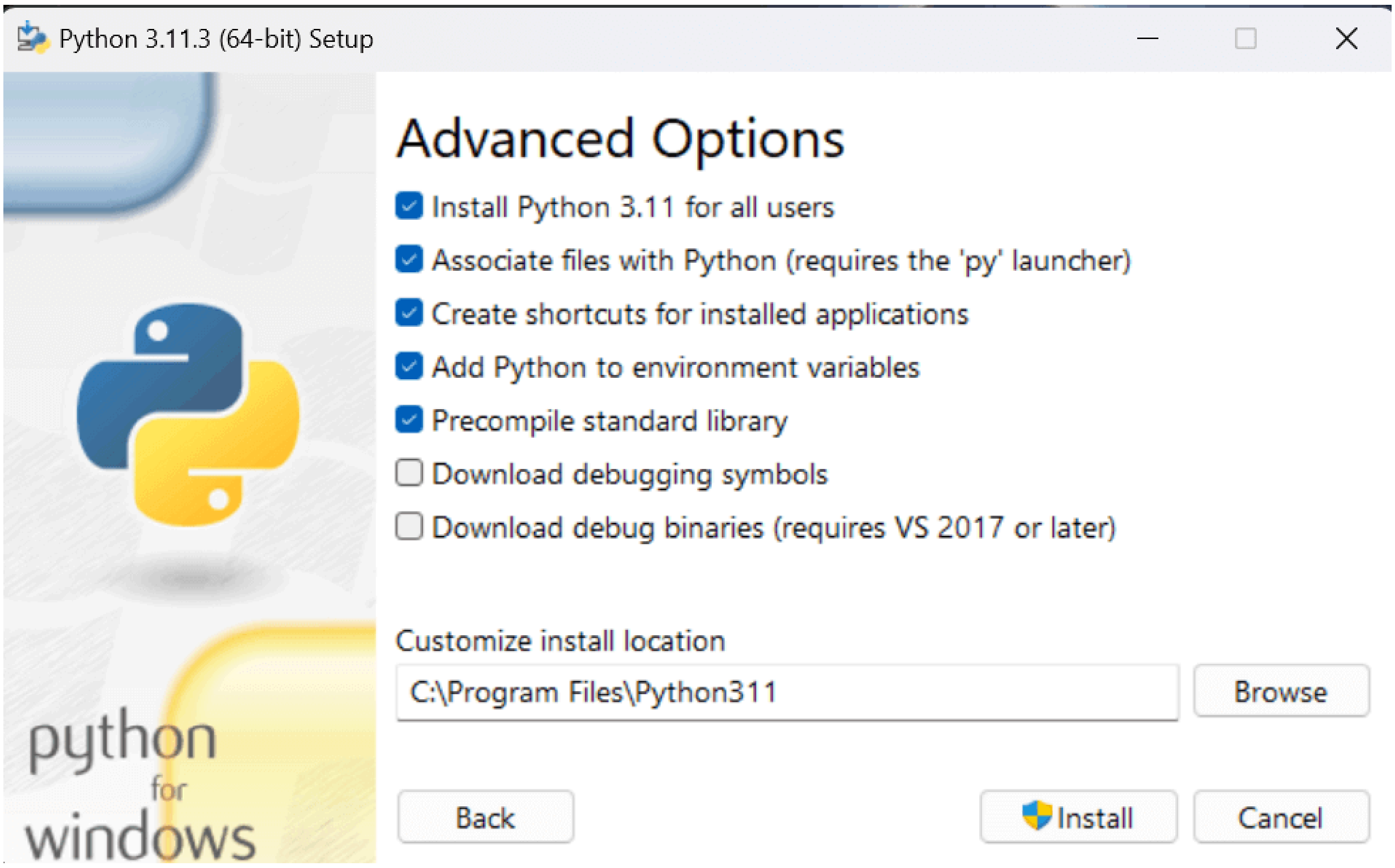
Install Now
C:\Users\bharg\AppData\Local\Programs\Python\Python311

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

[✓] Use admin privileges when installing py.exe
[ ] Add python.exe to PATH

The following window will open. Click on the Add Path check box, it will set the Python path automatically.


Python 3.11.3 (64-bit) Setup

Advanced Options

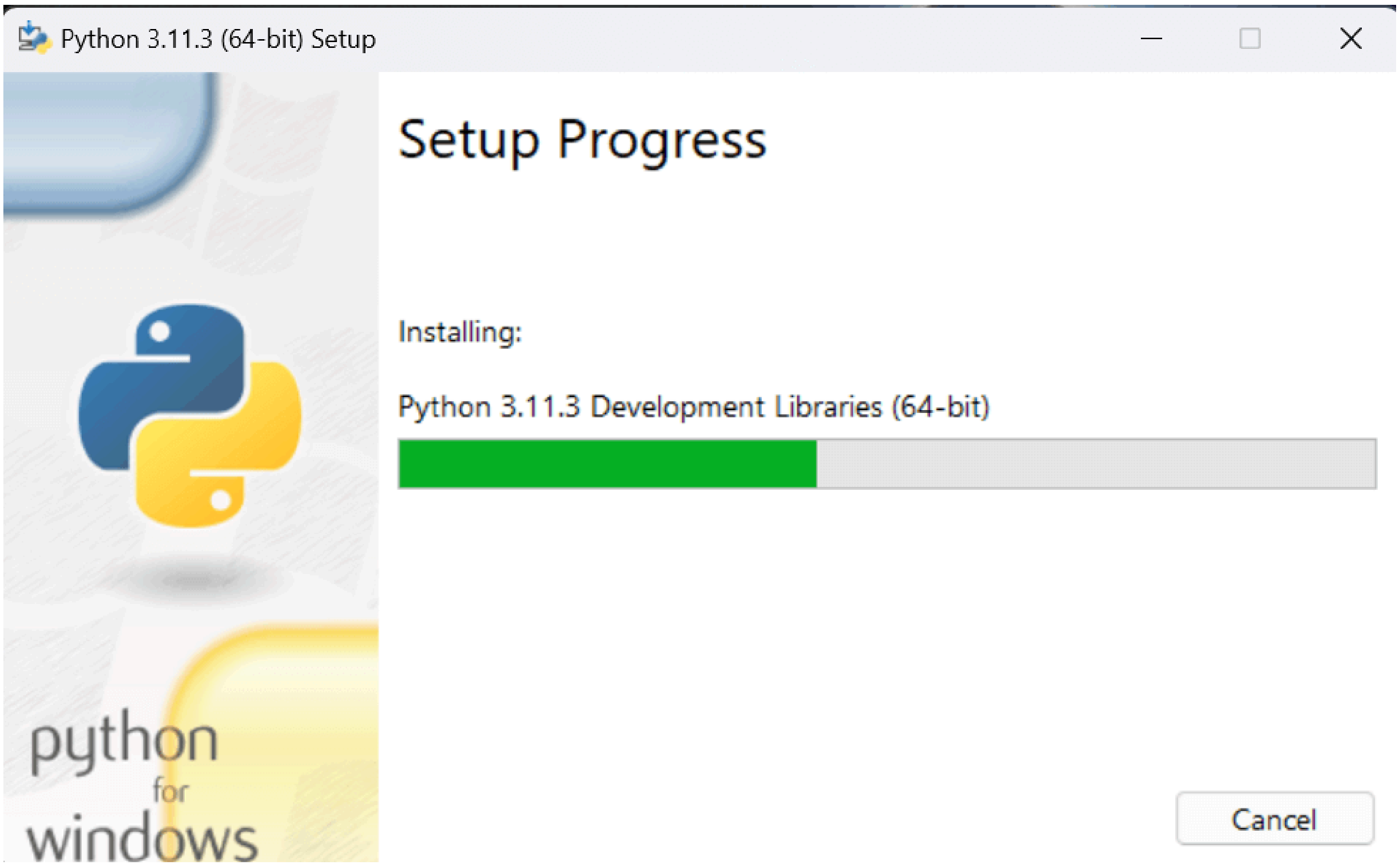
- [✓] Install Python 3.11 for all users
- [✓] Associate files with Python (requires the 'py' launcher)
- [✓] Create shortcuts for installed applications
- [✓] Add Python to environment variables
- [✓] Precompile standard library
- [ ] Download debugging symbols
- [ ] Download debug binaries (requires VS 2017 or later)

Customize install location

C:\Program Files\Python311

Step - 3 Installation in Process

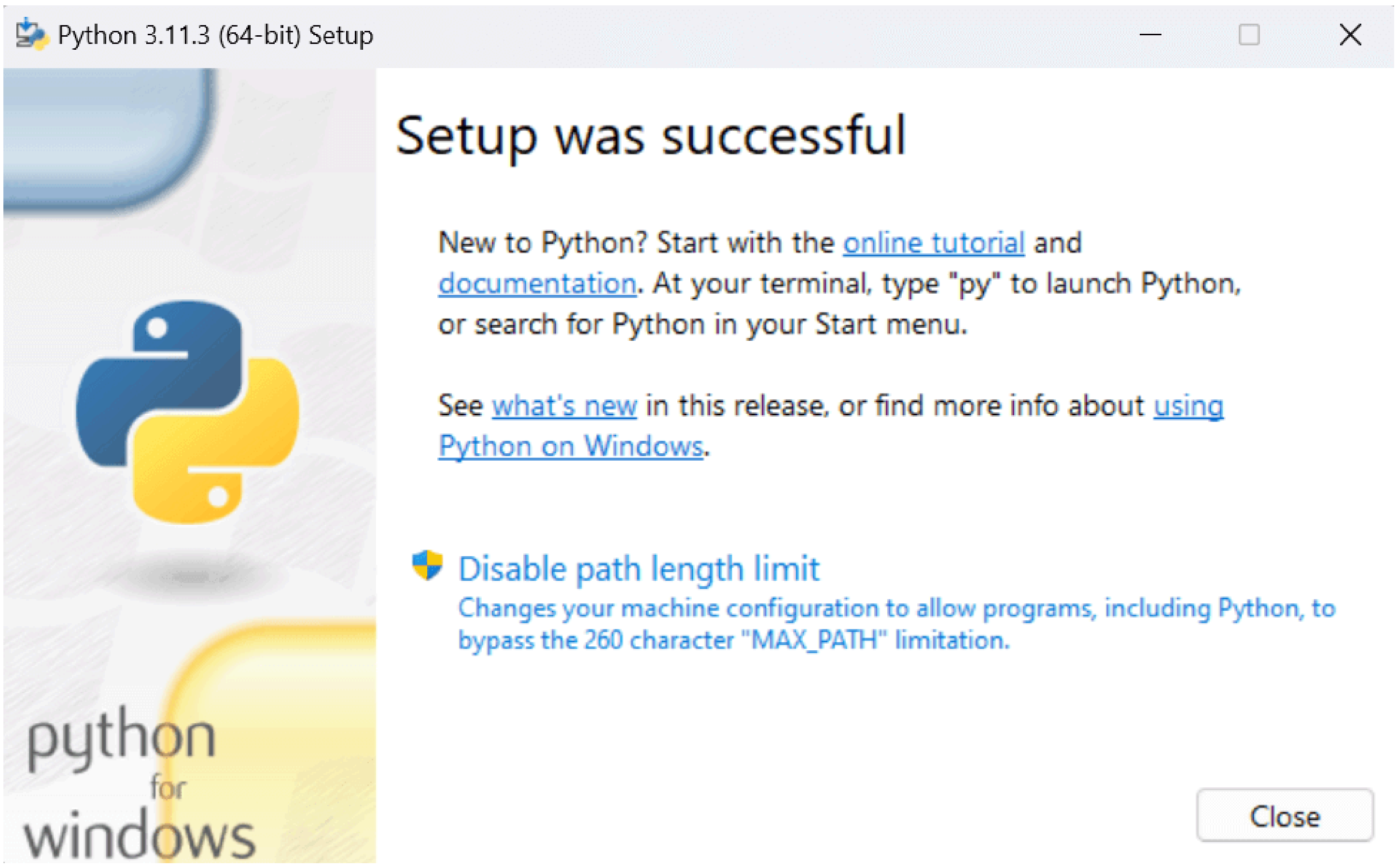
The set up is in progress. All the python libraries, packages, and other python default files will be installed in our system. Once the installation is successful, the following page will appear saying " Setup was successful ".


Python 3.11.3 (64-bit) Setup

Setup Progress

Installing:

Python 3.11.3 Development Libraries (64-bit)


Python 3.11.3 (64-bit) Setup

Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

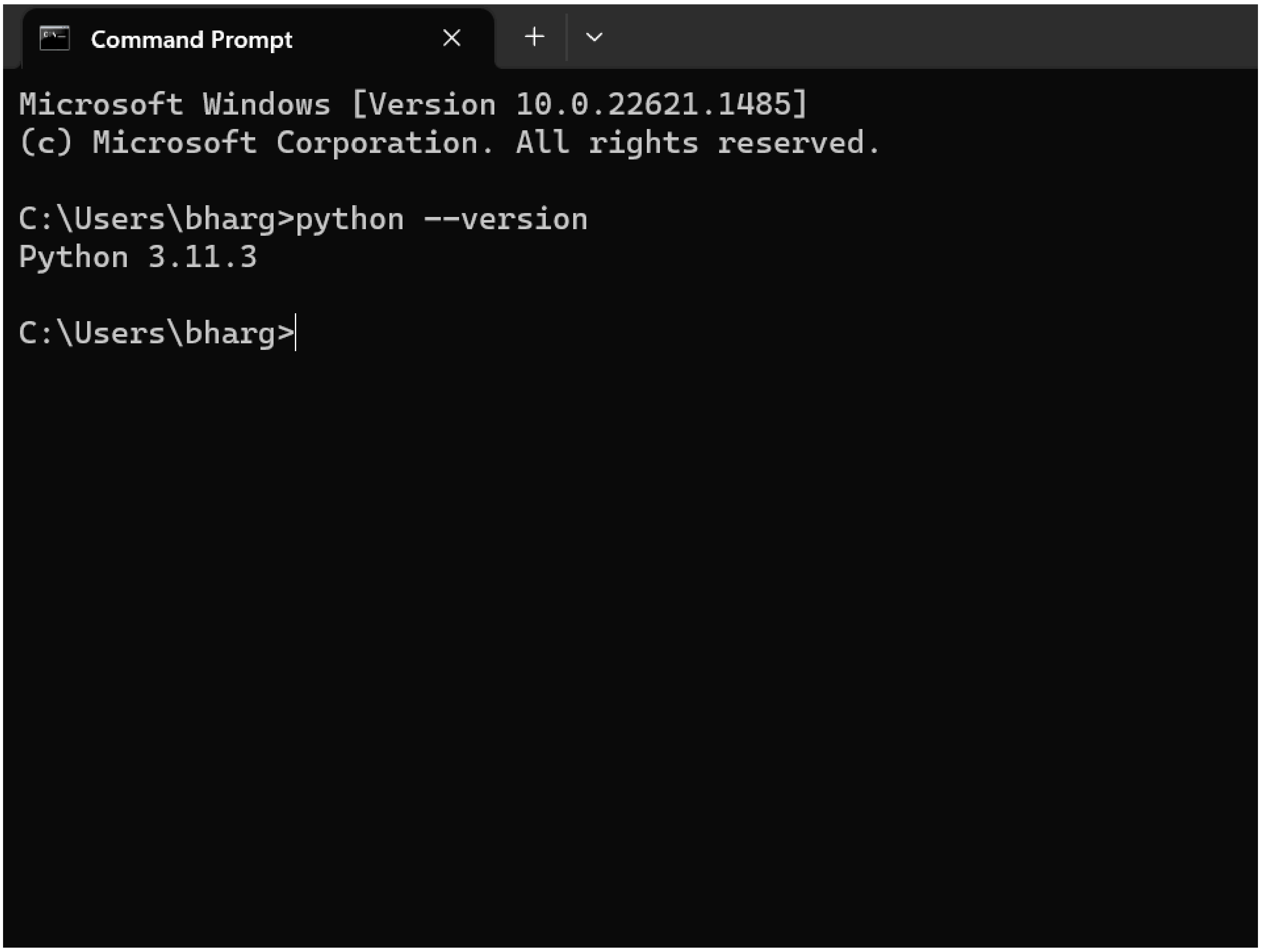
Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

Step - 4: Verifying the Python Installation

To verify whether the python is installed or not in our system, we have to do the following.

- Go to "Start" button, and search " cmd ".
- Then type, " **python - - version** ".
- If python is successfully installed, then we can see the version of the python installed.
- If not installed, then it will print the error as " **'python' is not recognized as an internal or external command, operable program or batch file.** ".

We are ready to work with the Python.

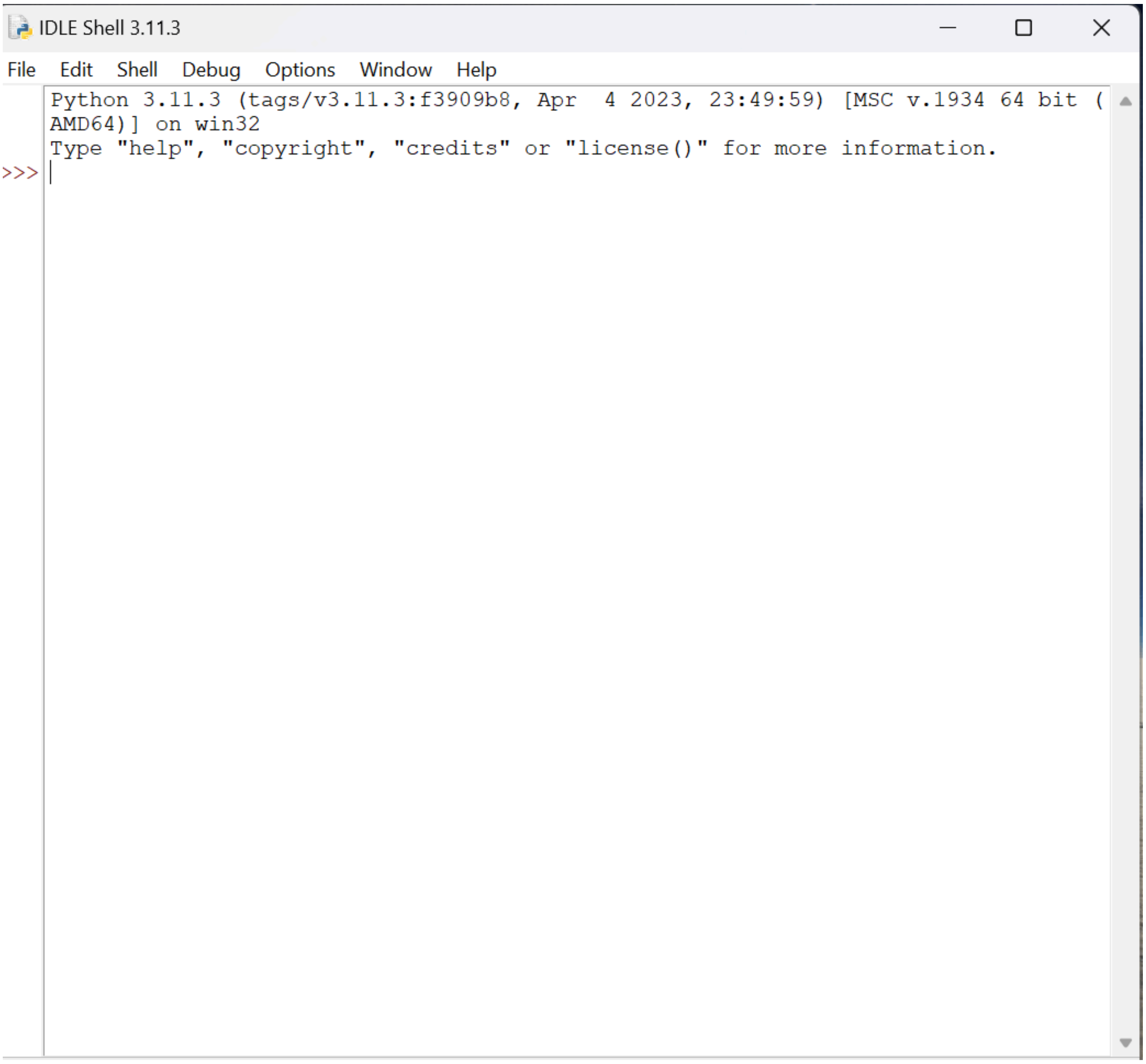


```
Microsoft Windows [Version 10.0.22621.1485]
(c) Microsoft Corporation. All rights reserved.

C:\Users\bharg>python --version
Python 3.11.3

C:\Users\bharg>
```

Step - 5: Opening idle

Now, to work on our first python program, we will go the interactive interpreter prompt(idle).

To open this, go to "Start" and type idle. Then, click on open to start working on idle.



```
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

First Python Program

In this Section, we will discuss the basic syntax of Python, we will run a simple program to print **Hello World** on the console.

Python provides us the two ways to run a program:

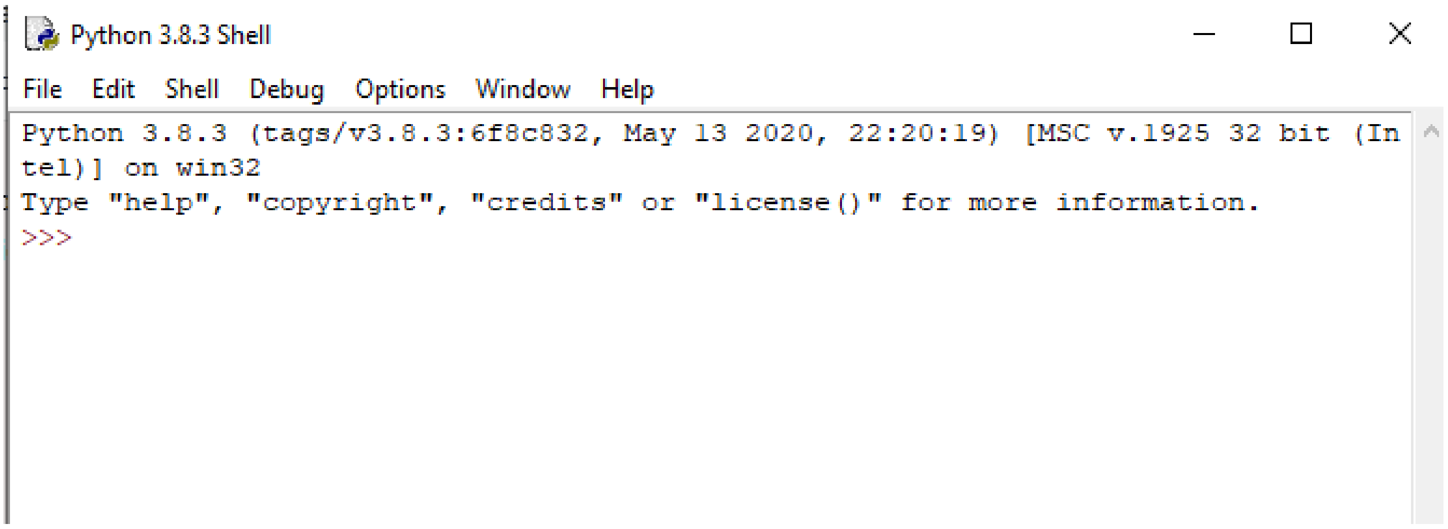
- Using Interactive interpreter prompt
- Using a script file

Let's discuss each one of them in detail.

Interactive interpreter prompt

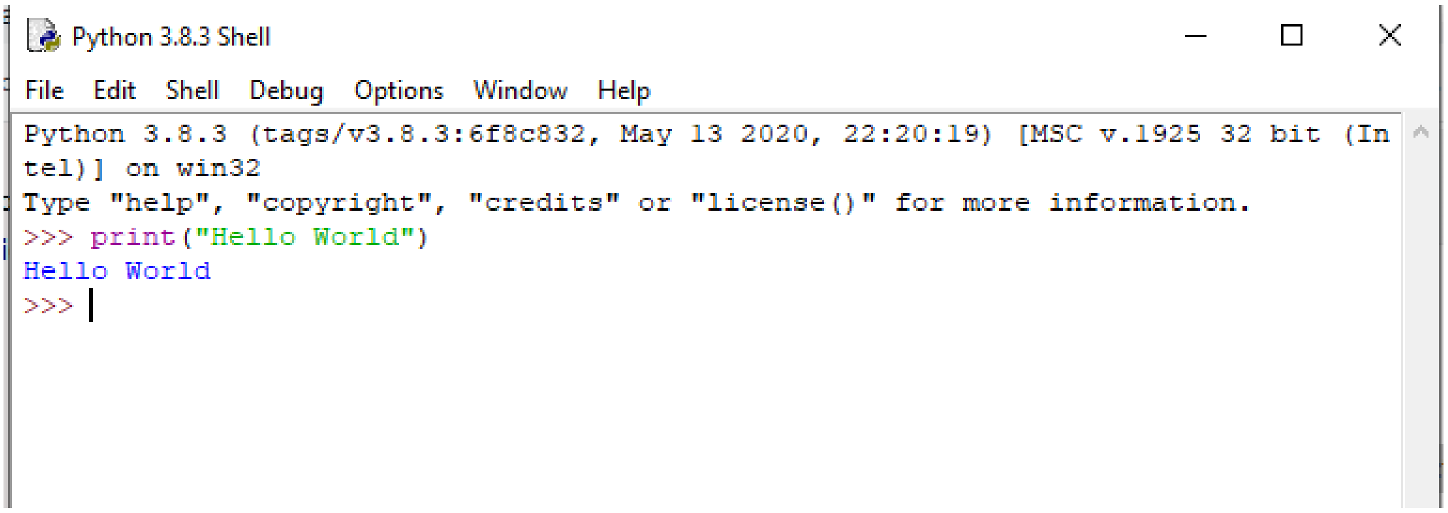
Python provides us the feature to execute the Python statement one by one at the interactive prompt. It is preferable in the case where we are concerned about the output of each line of our Python program.

It will open the following prompt where we can execute the Python statement and check their impact on the console.



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

After writing the print statement, press the **Enter** key.



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>>
```

Here, we get the message **"Hello World !"** printed on the console.

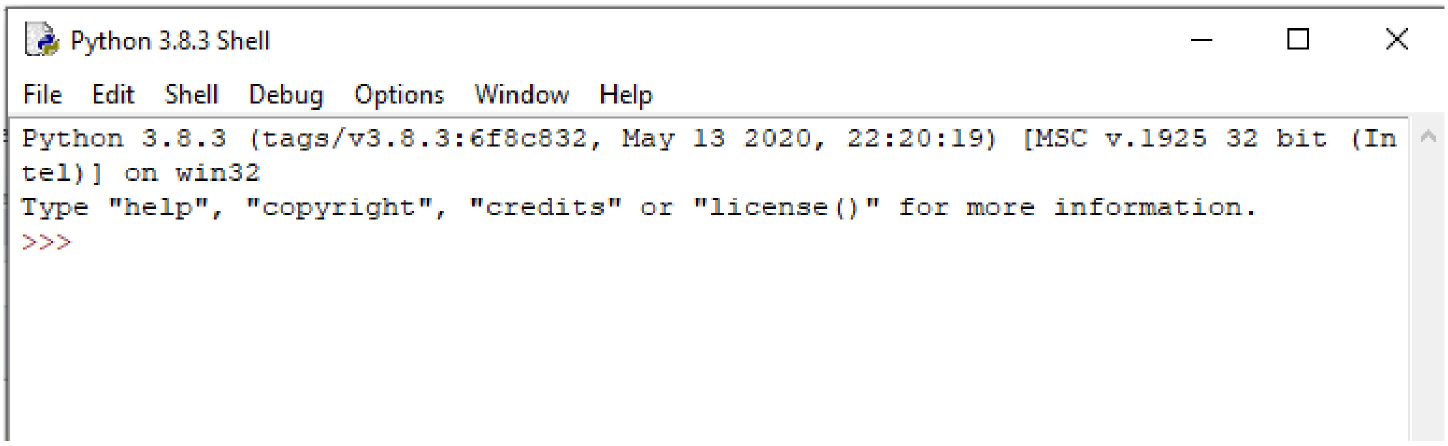
Using a script file (Script Mode Programming)

The interpreter prompt is best to run the single-line statements of the code. However, we cannot write the code every-time on the terminal. It is not suitable to write multiple lines of code.

Using the script mode, we can write multiple lines code into a file which can be executed later. For this purpose, we need to open an editor like notepad, create a file named and save it with **.py** extension, which stands for **"Python".** Now, we will implement the above example using the script mode.

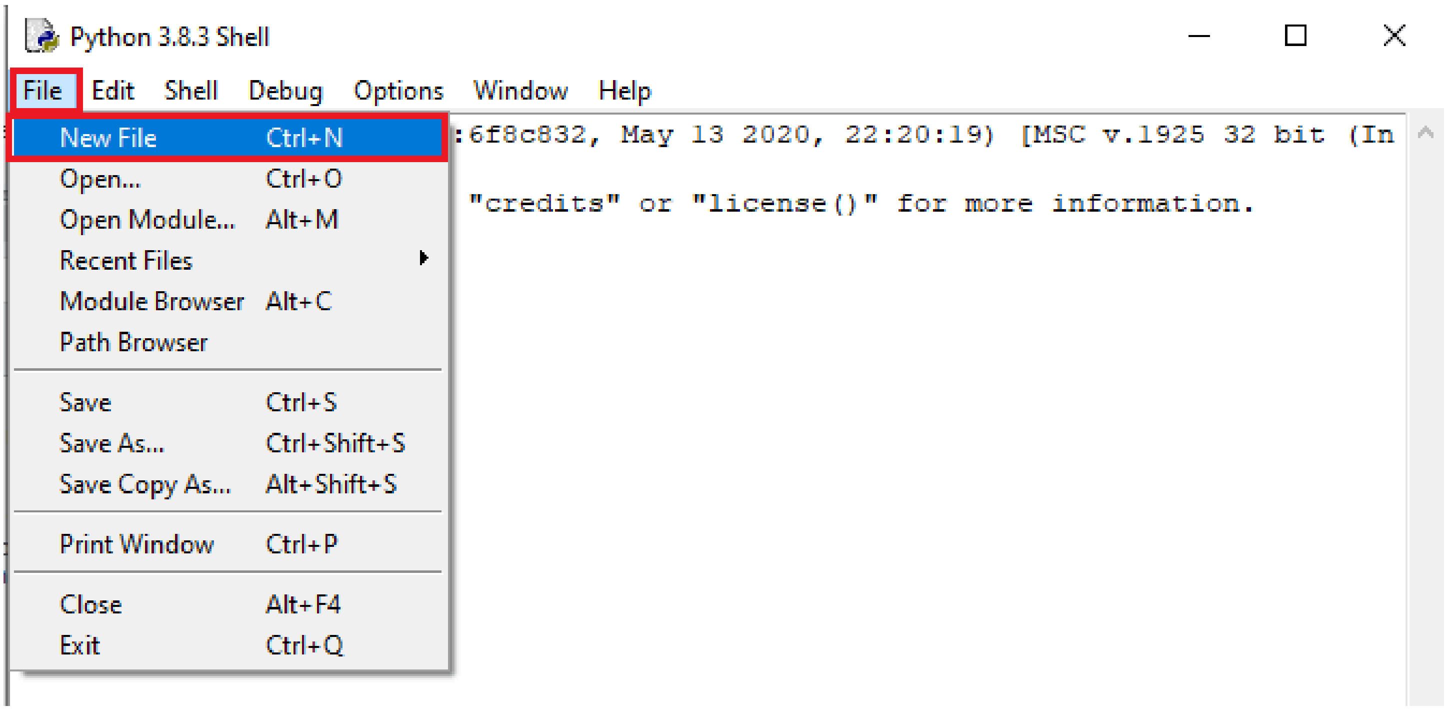
```
print ("hello world");
```

To run this file named as first.py, we need to run the following command on the terminal.



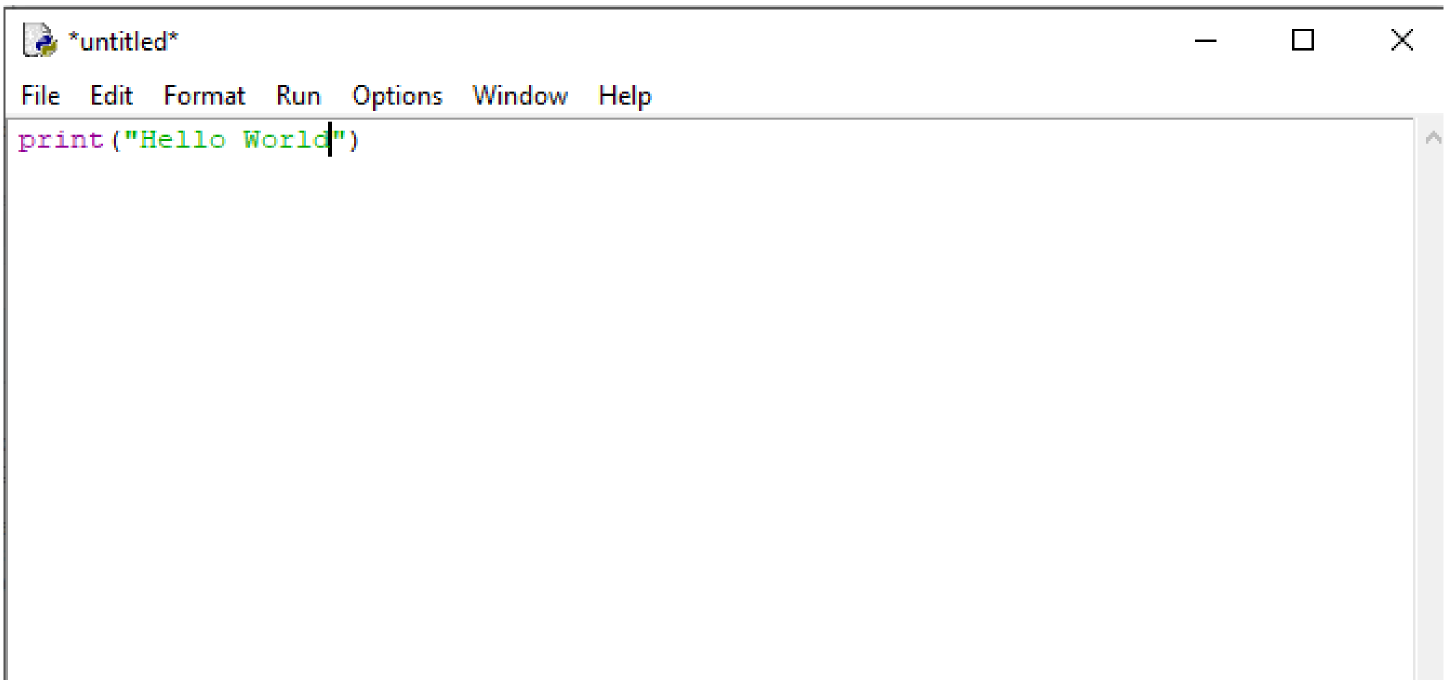
```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Step - 1: Open the Python interactive shell, and click **"File"** then choose **"New",** it will open a new blank script in which we can write our code.



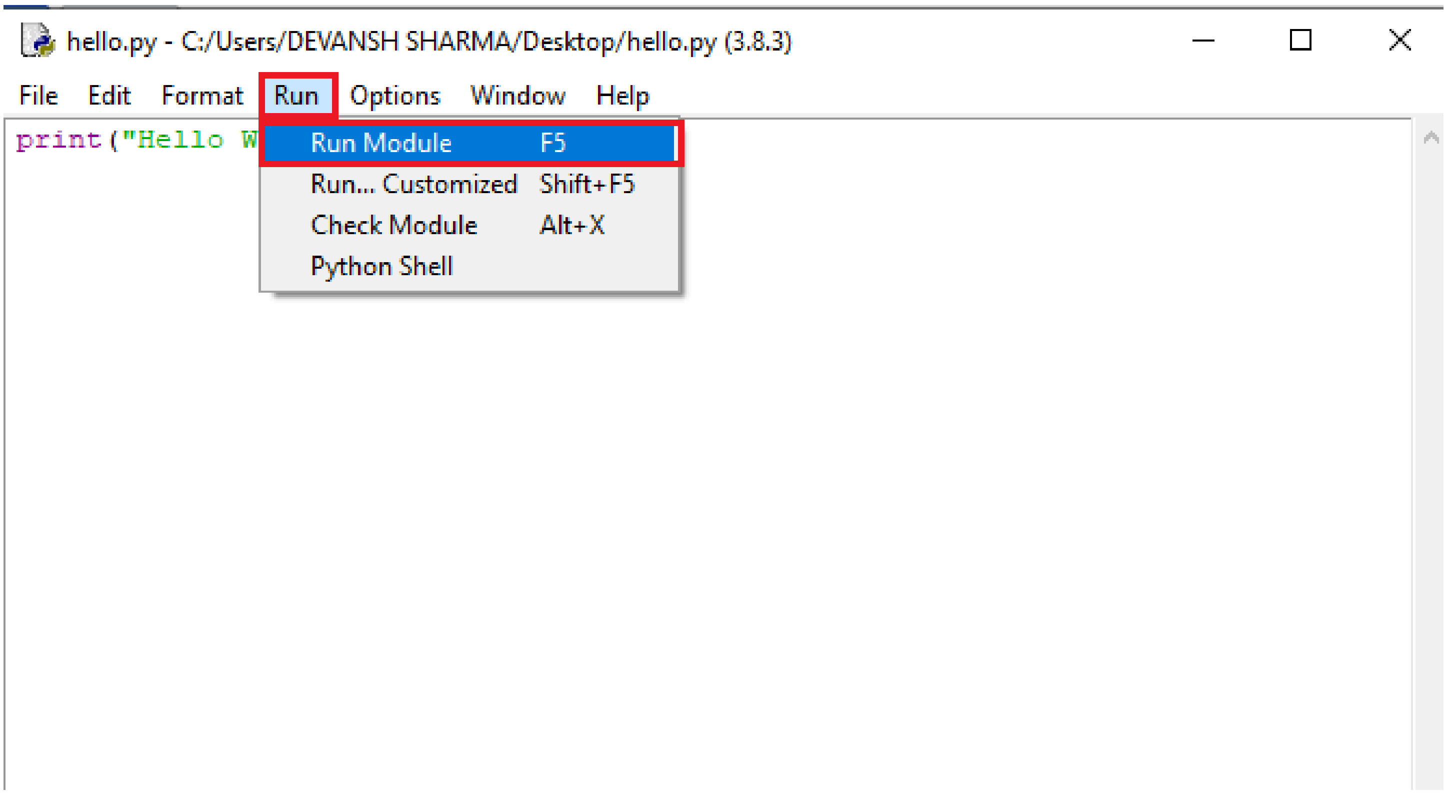
```
:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
 "credits" or "license()" for more information.
```

Step -2: Now, write the code and press **"Ctrl+S"** to save the file.



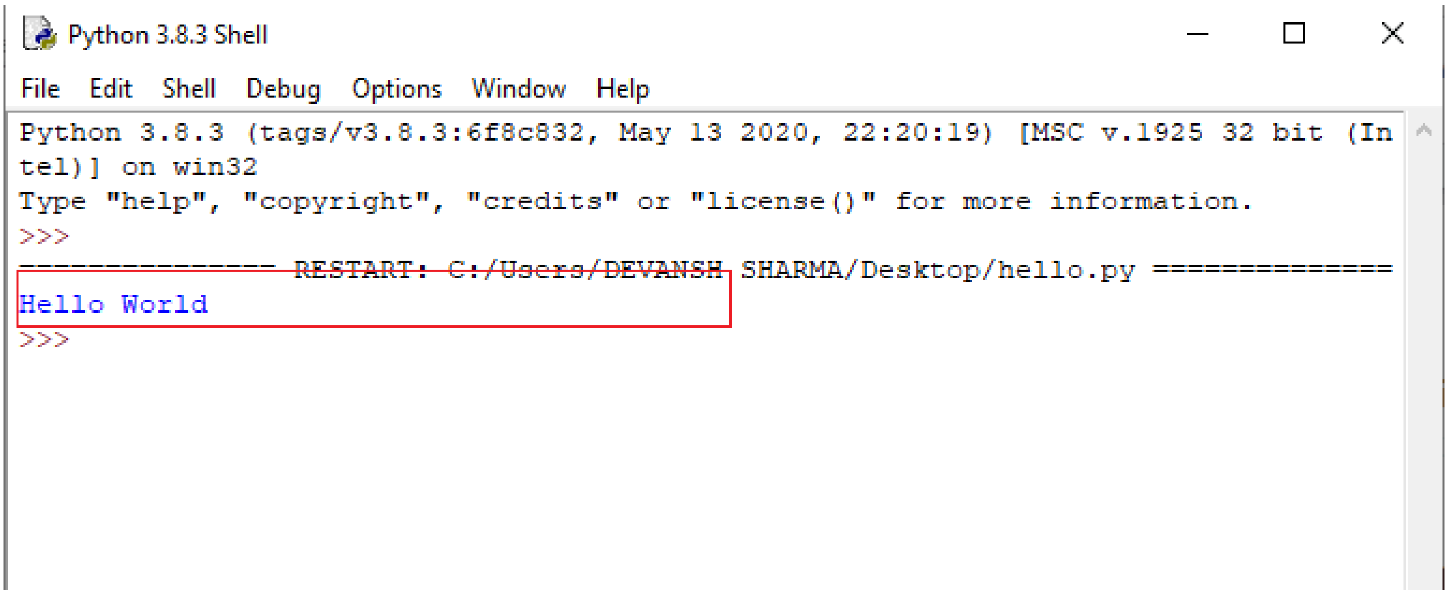
```
print("Hello World")
```

Step - 3: After saving the code, we can run it by clicking "Run" or "Run Module". It will display the output to the shell.


hello.py - C:/Users/DEVANSH SHARMA/Desktop/hello.py (3.8.3)
File Edit Format Run Options Window Help
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell

```
print("Hello W
```

The output will be shown as follows.



```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:/Users/DEVANSH SHARMA/Desktop/hello.py ==============
Hello World
>>>
```

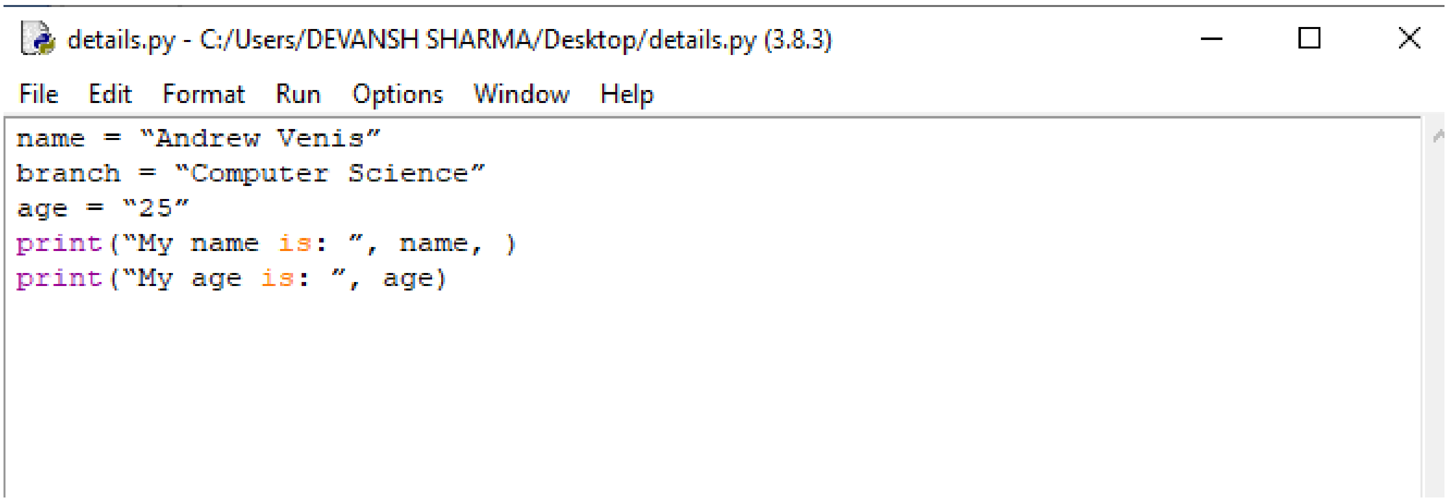
Multi-line Statements

Multi-line statements are written into the notepad like an editor and saved it with **.py** extension. In the following example, we have defined the execution of the multiple code lines using the Python script.

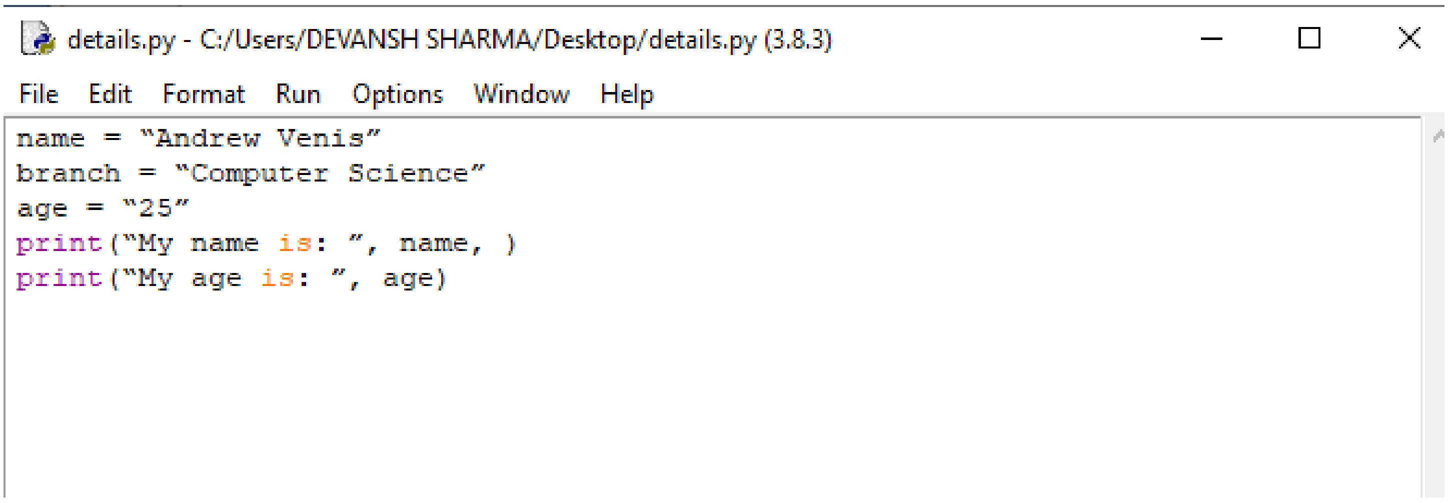
Code:

```
name = "Andrew Venis"
branch = "Computer Science"
age = "25"
print("My name is: ", name, )
print("My age is: ", age)
```

Script File:


details.py - C:/Users/DEVANSH SHARMA/Desktop/details.py (3.8.3)
File Edit Format Run Options Window Help

```
name = "Andrew Venis"
branch = "Computer Science"
age = "25"
print("My name is: ", name, )
print("My age is: ", age)
```


details.py - C:/Users/DEVANSH SHARMA/Desktop/details.py (3.8.3)
File Edit Format Run Options Window Help

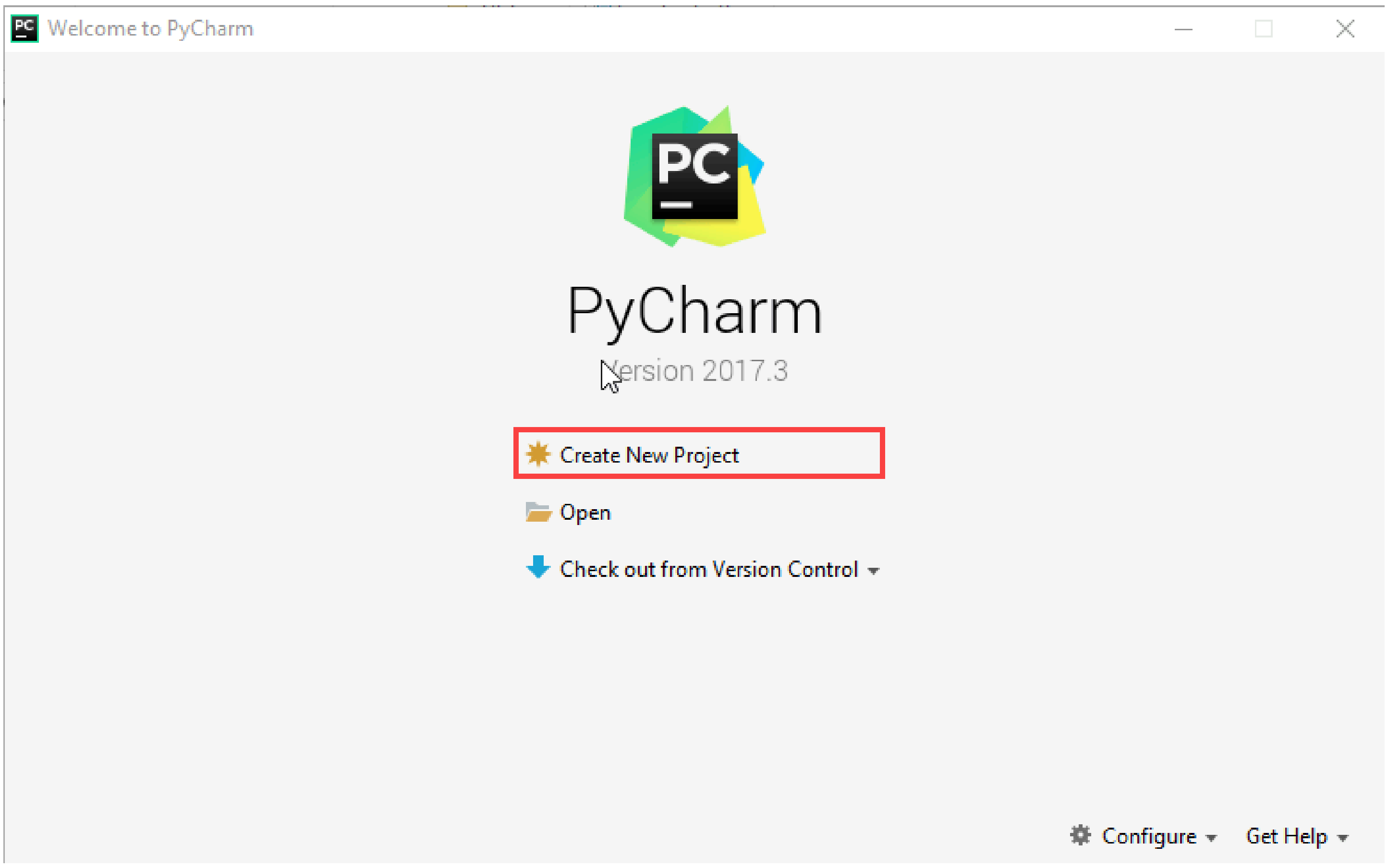
```
name = "Andrew Venis"
branch = "Computer Science"
age = "25"
print("My name is: ", name, )
print("My age is: ", age)
```

Get Started with PyCharm

Installing PyCharm on Windows is very simple. To install PyCharm on Windows operating system, visit the link https://www.jetbrains.com/pycharm/download/download-thanks.html?platform=windows to download the executable installer. **Double click** the installer (.exe) file and install PyCharm by clicking next at each step.

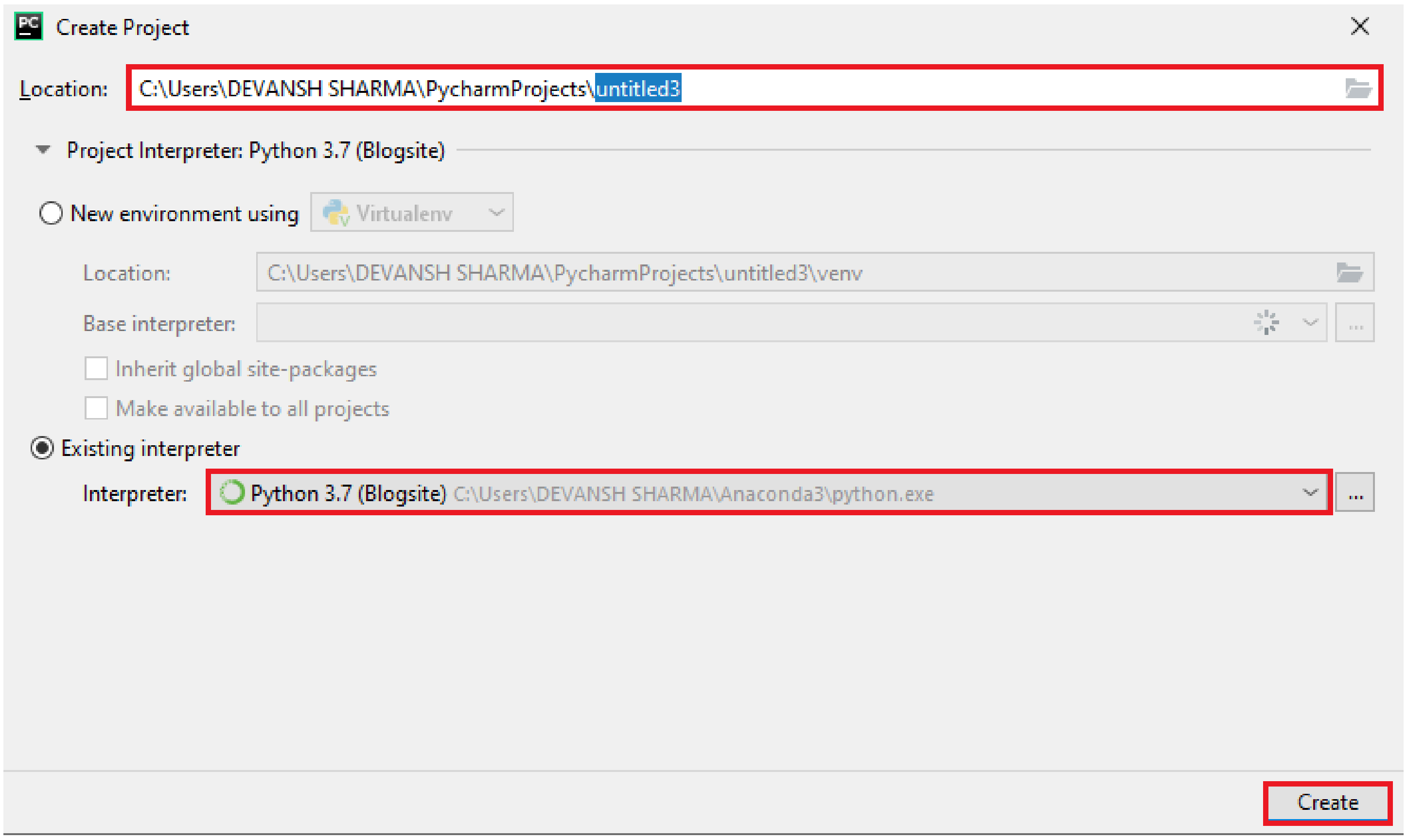
To create a first program to Pycharm follows the following step.

Step - 1. Open Pycharm editor. Click on "Create New Project" option to create new project.



Step - 2. Select a location to save the project.

a. We can save the newly created project at desired memory location or can keep file location as it is but atleast change the project default name **untitled** to **"FirstProject"** or something meaningful.
b. Pycharm automatically found the installed Python interpreter.
c. After change the name click on the "Create" Button.


Create Project

Location: C:\Users\DEVANSH SHARMA\PycharmProjects\untitled3

Project Interpreter: Python 3.7 (Blogsite)

New environment using Virtualenv

Location: C:\Users\DEVANSH SHARMA\PycharmProjects\untitled3\venv

Base interpreter:

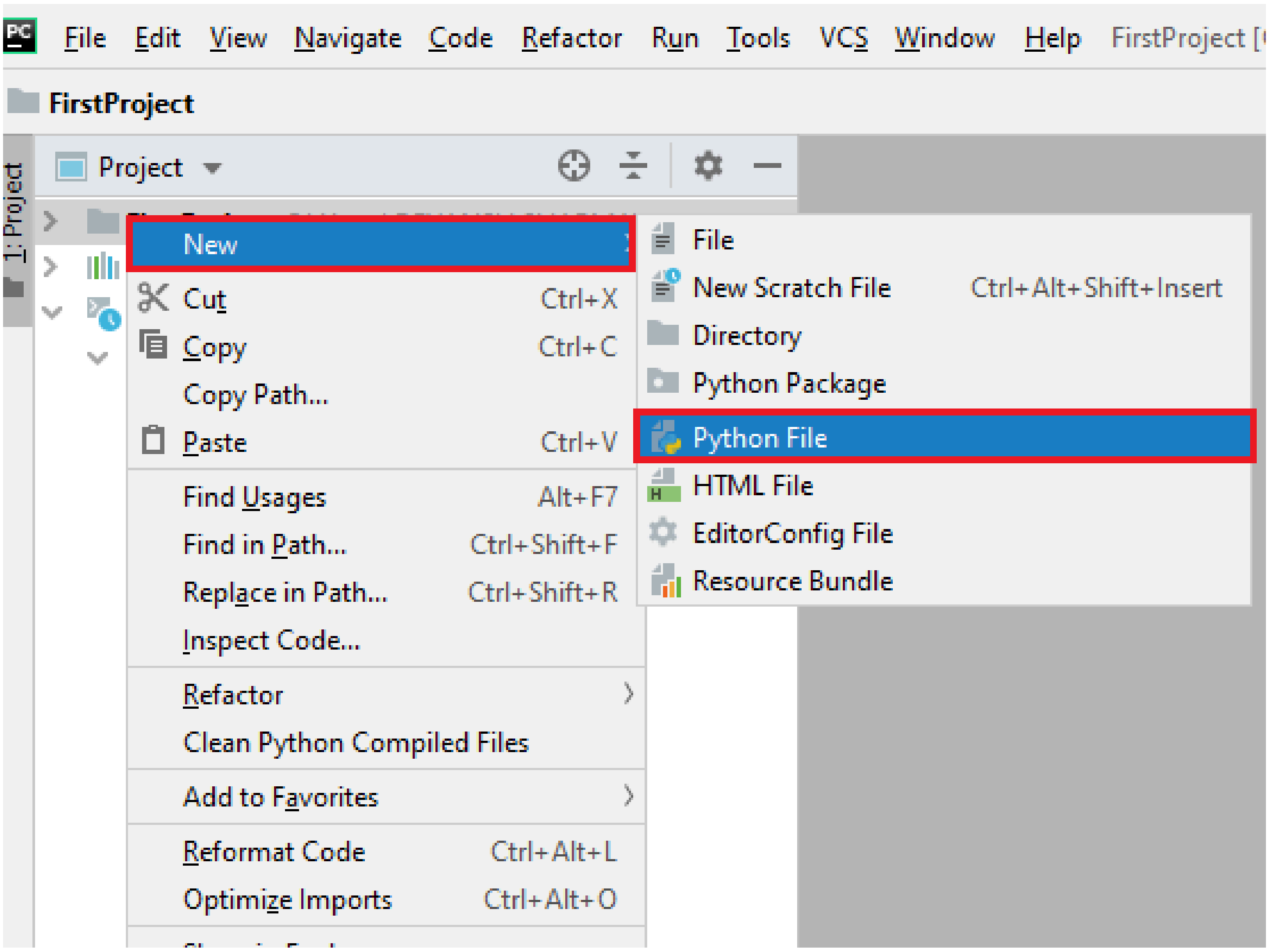
Inherit global site-packages

Make available to all projects

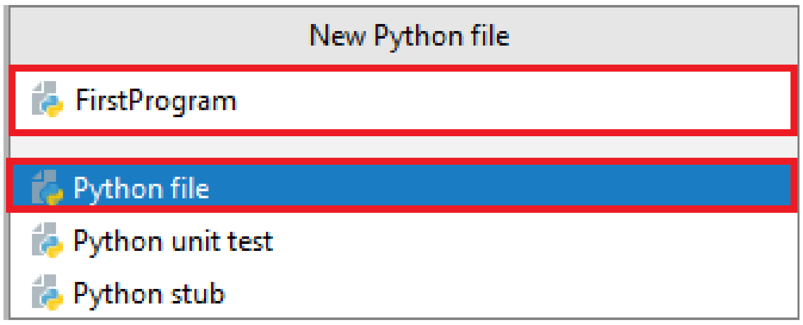
Existing interpreter

Interpreter: Python 3.7 (Blogsite) C:\Users\DEVANSH SHARMA\Anaconda3\python.exe

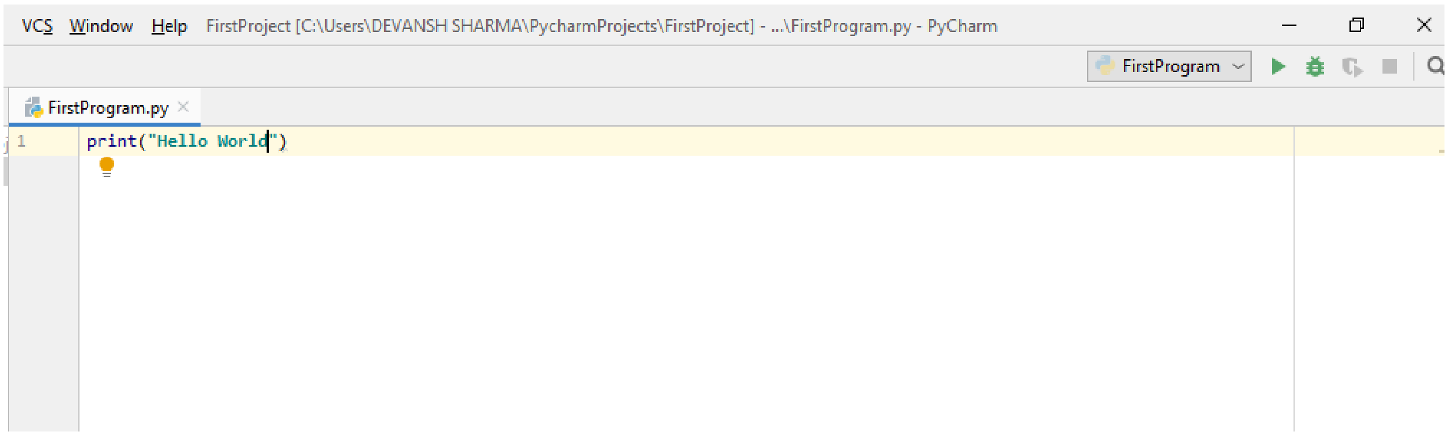
Step - 3. Click on "**File"** menu and select **"New"**. By clicking "New" option it will show various file formats. Select the "Python File".


File Edit View Navigate Code Refactor Run Tools VCS Window Help FirstProject
FirstProject
Project
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path...
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Inspect Code...
Refactor
Clean Python Compiled Files
Add to Favorites
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
File
New Scratch File Ctrl+Alt+Shift+Insert
Directory
Python Package
Python File
HTML File
EditorConfig File
Resource Bundle

Step - 4. Now type the name of the Python file and click on "OK". We have written the "FirstProgram".

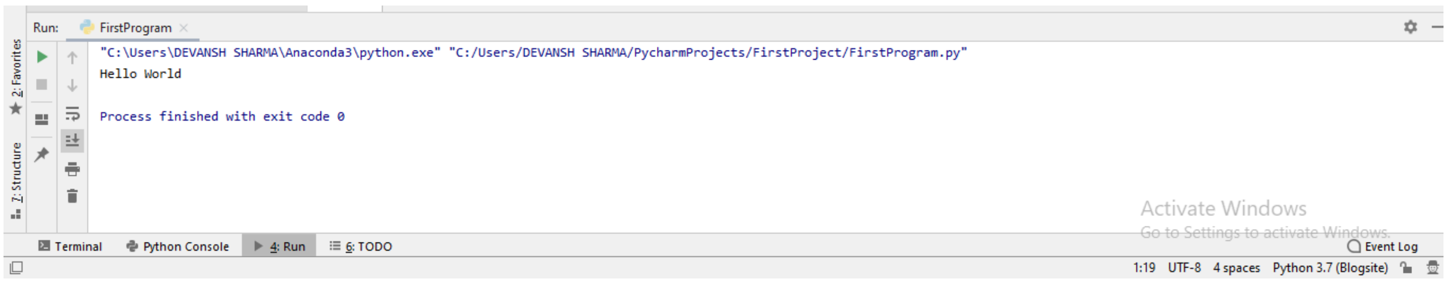


Step - 5. Now type the first program - print("Hello World") then click on the "Run" menu to run program.


VCS Window Help FirstProject [C:\Users\DEVANSH SHARMA\PycharmProjects\FirstProject] - ...\FirstProgram.py - PyCharm
FirstProgram
FirstProgram.py

```
print("Hello World")
```

Step - 6. The output will appear at the bottom of the screen.



```
"C:\Users\DEVANSH SHARMA\Anaconda3\python.exe" "C:/Users/DEVANSH SHARMA/PycharmProjects/FirstProject/FirstProgram.py"
Hello World

Process finished with exit code 0
```

Comments in Python

Comments are essential for defining the code and help us and other to understand the code. By looking the comment, we can easily understand the intention of every line that we have written in code.

Types of Comment

Python provides the facility to write comments in two ways- single line comment and multi-line comment.

Single-Line Comment - Single-Line comment starts with the hash # character followed by text for further explanation.

```
# defining the marks of a student
Marks = 90
```

We can also write a comment next to a code statement. Consider the following example.

```
Name = "James" # the name of a student is James
```

```
Marks = 90 # defining student's marks

Branch = "Computer Science" # defining student branch
```

Multi-Line Comments - Python doesn't have explicit support for multi-line comments but we can use hash # character to the multiple lines. **For example -**

```
# we are defining for loop

# To iterate the given list.

# run this code.
```

We can also use another way.

```
" " "
This is an example
Of multi-line comment
Using triple-quotes
" " "
```

Python Variables

A variable is the name given to a memory location. A value-holding Python variable is also known as an identifier.

Identifier Naming

Identifiers are things like variables. An Identifier is utilized to recognize the literals utilized in the program. The standards to name an identifier are given underneath.

- The variable's first character must be an underscore or alphabet (_).
- Every one of the characters with the exception of the main person might be a letter set of lower-case(a-z), capitalized (A-Z), highlight, or digit (0-9).
- White space and special characters (!, @, #, %, etc.) are not allowed in the identifier name. ^, &, *).
- Identifier name should not be like any watchword characterized in the language.
- Names of identifiers are case-sensitive; for instance, my name, and MyName isn't something very similar.
- Examples of valid identifiers: a123, _n, n_9, etc.
- Examples of invalid identifiers: 1a, n%4, n 9, etc.

Declaring Variable and Assigning Values

The equal (=) operator is utilized to assign worth to a variable.

Let's understand the following example

```
a = 50
```

Suppose we assign the integer value 50 to a new variable b.

```
a = 50
b = a
print(a,b)
```

Next,

```
name = "HARISH"
age = 30
marks = 80.50

print(name)
print(age)
print(marks)
```

The multi-word keywords can be created by the following method.

- **Camel Case -** In the camel case, each word or abbreviation in the middle of begins with a capital letter. There is no intervention of whitespace. For example - nameOfStudent, valueOfVaraible, etc.
- **Pascal Case -** It is the same as the Camel Case, but here the first word is also capital. For example - NameOfStudent, etc.
- **Snake Case -** In the snake case, Words are separated by the underscore. For example - name_of_student, etc.

Multiple Assignment

Multiple assignments, also known as assigning values to multiple variables in a single statement, is a feature of Python.

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
```

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/DEVANSH SHARMA/PycharmProjects/Hello/multiprocessing.py", line 389, in

    print(x)

NameError: name 'x' is not defined
```

Python Data Types

Every value has a datatype, and variables can hold values. The interpreter binds the value implicitly to its type.

Consider the following illustration when defining and verifying the values of various data types.

```
a=10
b="Hi Python"
c = 10.5
print(type(a))
print(type(b))
print(type(c))
```

Output:

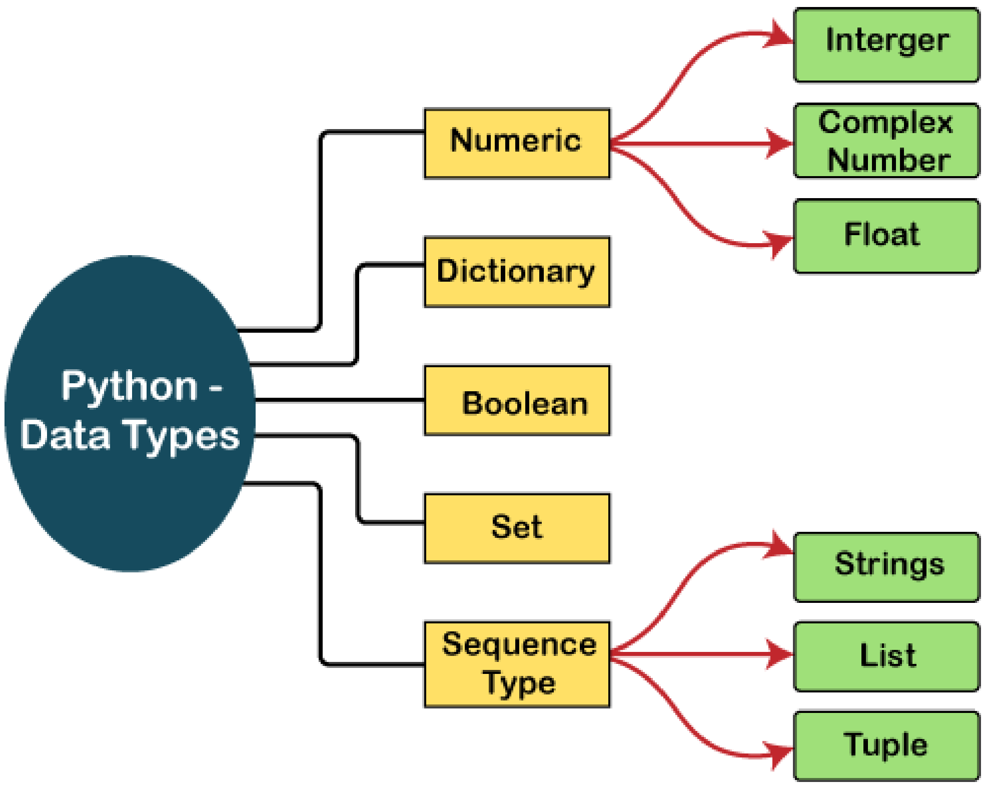
```
<type 'int'>
```

```
<type 'str'>
<type 'float'>
```

Standard data types

A variable can contain a variety of values. The following is a list of the Python-defined data types.

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



Numbers

Numeric values are stored in numbers. The whole number, float, and complex qualities have a place with a Python Numbers datatype.

When a number is assigned to a variable, Python generates Number objects. For instance,

```
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
```

Sequence Type

String

The sequence of characters in the quotation marks can be used to describe the string. A string can be defined in Python using single, double, or triple quotes.

When dealing with strings, the operation "hello"+" python" returns "hello python," and the operator + is used to combine two strings.

Because the operation "Python" *2 returns "Python," the operator * is referred to as a repetition operator.

The Python string is demonstrated in the following example.

Example - 1

```
str = "string using double quotes"
print(str)
s = '''A multiline
string'''
print(s)
```

Output:

```
string using double quotes
```

```
A multiline

string
```

Look at the following illustration of string handling.

Example - 2

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

Output:

```
he
o
hello studypointhello studypoint
hello studypoint how are you
```

List

Lists in Python are like arrays in C, but lists can contain data of different types.

Look at the following example.

Example:

```
list1 = [1, "hi", "Python", 2]
#Checking type of given list
print(type(list1))

#Printing the list1
print (list1)

# List slicing
print (list1[3:])

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

# List Concatenation using + operator
```

```
print (list1 + list1)

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

Output:

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

Tuple

In many ways, a tuple is like a list. Tuples, like lists, also contain a collection of items from various data types.

Because we cannot alter the size or value of the items in a tuple, it is a read-only data structure.

Let's look at a straightforward tuple in action.

Example:

```
tup = ("hi", "Python", 2)
# Checking type of tup
print (type(tup))

#Printing the tuple
print (tup)

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

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

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

# Adding value to tup. It will throw an error.
```

```
t[2] = "hi"
```

Output:

```
<class 'tuple'>
('hi', 'Python', 2)
('Python', 2)
('hi',)
('hi', 'Python', 2, 'hi', 'Python', 2)
('hi', 'Python', 2, 'hi', 'Python', 2, 'hi', 'Python', 2)

Traceback (most recent call last):
  File "main.py", line 14, in <module>
    t[2] = "hi";
TypeError: 'tuple' object does not support item assignment
```

Dictionary

A dictionary is a key-value pair set arranged in any order. It stores a specific value for each key, like an associative array

he comma (,) and the curly braces are used to separate the items in the dictionary.

Look at the following example.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}

# Printing dictionary
print (d)

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

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

Output:

```
1st name is Jimmy
2nd name is mike
```

```
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
dict_keys([1, 2, 3, 4])
dict_values(['Jimmy', 'Alex', 'john', 'mike'])
```

Boolean

True and False are the two default values for the Boolean type.

Look at the following example.

```
# Python program to check the boolean type
print(type(True))
print(type(False))
print(false)
```

Output:

```
<class 'bool'>

<class 'bool'>

NameError: name 'false' is not defined
```

Set

The data type's unordered collection is Python Set. It is iterable, mutable(can change after creation), and has remarkable components. The elements of a set have no set order; It might return the element's altered sequence.

Look at the following example.

```
# Creating Empty set
set1 = set()

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

#Printing Set value
print(set2)

# Adding element to the set

set2.add(10)
print(set2)
```

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

Output:

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

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

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

Python Keywords

Every scripting language has designated words or keywords, with particular definitions and usage guidelines.

Introducing Python Keywords

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

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

Code

```
# importing keyword library which has lists
import keyword
```

```
# displaying the complete list using "kwlist()."
print("The set of keywords in this version is: ")
print( keyword.kwlist )
```

Output:

```
The set of keywords in this version is :

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

By calling help(), you can retrieve a list of currently offered keywords:

Code

```
help("keywords")
```

The Keywords True and False

True and False are those keywords that can be allocated to variables or parameters and are compared directly.

Code

```
print( 4 == 4 )
print( 6 > 9 )
print( True or False )
print( 9 <= 28 )
print( 6 > 9 )
print( True and False )
```

Output:

```
True
False
True
True
False
False
```

Code

```
1. print( True == 3 )
```

```
2. print( False == 0 )
3. print( True + True + True)
```

Output:

```
False

True

3
```

The None Keyword

None is a Python keyword that means "nothing." None is known as nil, null, or undefined in different computer languages.

If a function does not have a return clause, it will give None as the default output:

Code

```
print( None == 0 )
print( None == " " )
print( None == False )
A = None
B = None
print( A == B )
```

Output:

```
False
False
False
True
```

Operator Keywords: and, or, not, in, is

Several Python keywords are employed as operators to perform mathematical operations.

All of these are keyword operations in Python:

Mathematical Operations	Operations in Other Languages	Python Keyword
AND, ∧	&&	and
OR, ∨	||	or
NOT, ¬	!	not
CONTAINS, ∈		in
IDENTITY	===	is

The and Keyword

The outcome will be True if both components are true. If one is false, the outcome will also be False:

Truth table for and		
X	**Y**	**X and Y**
True	**True**	**True**
False	**True**	**False**
True	**False**	**False**
False	**False**	**False**

<component1> **and** <component2>

The or Keyword

If the first argument is true, the or operation yields it; otherwise, the second argument is returned:

```
<component1> or <component2>
```

Truth table for or		
X	**Y**	**X or Y**
True	**True**	**True**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

The not Keyword

Not, unlike and, and or, determines the specific Boolean state, True or False, afterward returns the inverse.

Truth Table for not	
X	not X
True	False
False	True

Code

```
False and True
False or True
not True
```

Output:

```
False
True
False
```

The in Keyword

The in keyword of Python is a robust confinement checker, also known as a membership operator. If you provide it an element to seek and a container or series to seek into, it will give True or False, depending on if that given element was located in the given container:

```
<an_element> in <a_container>
```

Testing for a certain character in a string is a nice illustration of how to use the in keyword:

Code

```
container = "studypoint"
print( "p" in container )
print( "P" in container )
```

Output:

```
True
False
```

The is Keyword

In Python, it's used to check the identification of objects. The == operation is used to determine whether two arguments are identical. It also determines whether two arguments relate to the unique object.

When the objects are the same, it gives True; otherwise, it gives False.

Code

```
print( True is True )

print( False is True )

print( None is not None )

print( (9 + 5) is (7 * 2) )
```

Output:

```
True

False

False

True
```

True, False, and None are all the same in Python since there is just one version.

Code

```
print( [] == [] )

print( [] is [] )

print( {} == {} )

print( {} is {} )
```

Output:

```
True

False

True

False
```

A blank dictionary or list is the same as another blank one. However, they aren't identical entities because they are stored independently in memory. This is because both the list and the dictionary are changeable.

Code

```
print( '' == '' )

print( '' is '' )
```

Output:

```
True

True
```

Python Literals

Python Literals can be defined as data that is given in a variable or constant.

Python supports the following literals:

1. String literals:

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes to create a string.

Example:

```
"Aman" , '12345'
```

Types of Strings:

There are two types of Strings supported in Python:

a) Single-line String- Strings that are terminated within a single-line are known as Single line Strings.

Example:

```
text1='hello'
```

b) Multi-line String - A piece of text that is written in multiple lines is known as multiple lines string.

There are two ways to create multiline strings:

1) Adding black slash at the end of each line.

Example:

```
text1='hello\
user'
print(text1)
'hellouser'
```

2) Using triple quotation marks:-

Example:

```
str2='''welcome
to
SSSIT'''
print str2
```

Output:

```
welcome
to
SSSIT
```

II. Numeric literals:

Numeric Literals are immutable. Numeric literals can belong to following four different numerical types.

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)
```

Output:

```
20 100 141 301

100.5 150.0

(5+3.14j) 3.14 5.0
```

III. Boolean literals:

A Boolean literal can have any of the two values: True or False.

Example - 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)
```

Output:

```
x is True

y is False

z is False

a: 11

b: 10
```

IV. Special literals.

Python contains one special literal i.e., **None.**

None is used to specify to that field that is not created. It is also used for the end of lists in Python.

Example - Special Literals

```
val1=10
val2=None
print(val1)
print(val2)
```

Output:

```
10

None
```

V. Literal Collections.

Python provides the four types of literal collection such as List literals, Tuple literals, Dict literals, and Set literals.

List:

- List contains items of different data types. Lists are mutable i.e., modifiable.
- The values stored in List are separated by comma(,) and enclosed within square brackets([]). We can store different types of data in a List.

Example - List literals

```
list=['John',678,20.4,'Peter']
list1=[456,'Andrew']
print(list)
print(list + list1)
```

Output:

```
['John', 678, 20.4, 'Peter']

['John', 678, 20.4, 'Peter', 456, 'Andrew']
```

Dictionary:

- Python dictionary stores the data in the key-value pair.
- It is enclosed by curly-braces {} and each pair is separated by the commas(,).

Example

```
dict = {'name': 'Pater', 'Age':18,'Roll_nu':101}
print(dict)
```

Output:

```
{'name': 'Pater', 'Age': 18, 'Roll_nu': 101}
```

Tuple:

- Python tuple is a collection of different data-type. It is immutable which means it cannot be modified after creation.
- It is enclosed by the parentheses () and each element is separated by the comma(,).

Example

```
tup = (10,20,"Dev",[2,3,4])
print(tup)
```

Output:

```
(10, 20, 'Dev', [2, 3, 4])
```

Set:

- Python set is the collection of the unordered dataset.
- It is enclosed by the {} and each element is separated by the comma(,).

Example: - Set Literals

```
set = {'apple','grapes','guava','papaya'}
print(set)
```

Output:

```
{'guava', 'apple', 'papaya', 'grapes'}
```

Python Operators

In every programming language, some operators perform several tasks. Same as other languages, 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

Arithmetic operators used between two operands for a particular operation. There are many arithmetic operators. It includes the exponent (**) operator as well as the + (addition), - (subtraction), * (multiplication), / (divide), % (reminder), and // (floor division) operators.

Program Code:

Now we give code examples of arithmetic operators in Python. The code is given below -

```
a = 32 # Initialize the value of a
b = 6 # Initialize the value of b
print('Addition of two numbers:',a+b)
print('Subtraction of two numbers:',a-b)
print('Multiplication of two numbers:',a*b)
print('Division of two numbers:',a/b)
print('Reminder of two numbers:',a%b)
print('Exponent of two numbers:',a**b)
print('Floor division of two numbers:',a//b)
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

```
Addition of two numbers: 38

Subtraction of two numbers: 26

Multiplication of two numbers: 192

Division of two numbers: 5.333333333333333

Reminder of two numbers: 2

Exponent of two numbers: 1073741824

Floor division of two numbers: 5
```

Comparison operator

Comparison operators mainly use for comparison purposes. Comparison operators compare the values of the two operands and return a true or false Boolean value in accordance. The example of comparison operators are ==, !=, <=, >=, >, <. In the below table, we explain the works of the operators.

Program Code:

Now we give code examples of Comparison operators in Python. The code is given below -

```python
a = 32 # Initialize the value of a
b = 6 # Initialize the value of b
print('Two numbers are equal or not:',a==b)
print('Two numbers are not equal or not:',a!=b)
print('a is less than or equal to b:',a<=b)
print('a is greater than or equal to b:',a>=b)
print('a is greater b:',a>b)
print('a is less than b:',a<b)
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

```
Two numbers are equal or not: False

Two numbers are not equal or not: True
```

```
a is less than or equal to b: False

a is greater than or equal to b: True

a is greater b: True

a is less than b: False
```

Assignment Operators

Using the assignment operators, the right expression's value is assigned to the left operand. There are some examples of assignment operators like =, +=, -=, *=, %=, **=, //=. In the below table, we explain the works of the operators.

Program Code:

Now we give code examples of Assignment operators in Python. The code is given below -

```
a = 32 # Initialize the value of a
b = 6 # Initialize the value of b
print('a=b:', a==b)
print('a+=b:', a+b)
print('a-=b:', a-b)
print('a*=b:', a*b)
print('a%=b:', a%b)
print('a**=b:', a**b)
print('a//=b:', a//b)
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

```
a=b: False
a+=b: 38
a-=b: 26
a*=b: 192
a%=b: 2
a**=b: 1073741824
a//=b: 5
```

Bitwise Operators

The two operands' values are processed bit by bit by the bitwise operators. The examples of Bitwise operators are bitwise OR (|), bitwise AND (&), bitwise XOR (^), negation (~), Left shift (<<), and Right shift (>>). Consider the case below.

Program Code:

Now we give code examples of Bitwise operators in Python. The code is given below -

```
a = 5 # initialize the value of a
b = 6 # initialize the value of b
print('a&b:', a&b)
print('a|b:', a|b)
print('a^b:', a^b)
print('~a:', ~a)
print('a<<b:', a<<b)
print('a>>b:', a>>b)
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

```
a&b: 4

a|b: 7

a^b: 3

~a: -6

a<>b: 0
```

Logical Operators

The examples of logical operators are and, or, and not.

Program Code:

Now we give code examples of arithmetic operators in Python. The code is given below -

```
a = 5 # initialize the value of a
print('Is this statement true?:',a > 3 and a < 5)
print('Any one statement is true?:',a > 3 or a < 5)
print('Each statement is true then return False and vice-versa:',(not(a > 3 and a < 5)))
```

Output:

Now we give code examples of Bitwise operators in Python. The code is given below -

```
Is this statement true?: False
Any one statement is true?: True
Each statement is true then return False and vice-versa: True
```

Membership Operators

The membership of a value inside a Python data structure can be verified using Python membership operators. The result is true if the value is in the data structure; otherwise, it returns false.

rogram Code:

Now we give code examples of Membership operators in Python. The code is given below -

```
x = ["Rose", "Lotus"]
print(' Is value Present?', "Rose" in x)
print(' Is value not Present?', "Riya" not in x)
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

```
Is value Present? True

Is value not Present? True
```

Identity Operators

Program Code:

Now we give code examples of Identity operators in Python. The code is given below -

```
a = ["Rose", "Lotus"]
b = ["Rose", "Lotus"]
c = a
print(a is c)
print(a is not c)
print(a is b)
print(a is not b)
print(a == b)
print(a != b)
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
True
False
False
```

```
True
True
False
```

Operator Precedence

The order in which the operators are examined is crucial to understand since it tells us which operator needs to be considered first. Below is a list of the Python operators' precedence tables.

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 Comments

We'll study how to write comments in our program in this article. We'll also learn about single-line comments, multi-line comments, documentation strings, and other Python comments.

Single-Line Comments

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. As a result, we may put the program mentioned above in one line as follows:

Code

```
print( 'This is not a comment' ) # this code is to show an example of a single-line comment
```

Output:

```
This is not a comment
```

This program's output will be identical to the example above. The computer overlooks all content following #.

Multi-Line Comments

Python does not provide the facility for multi-line comments. However, there are indeed many ways to create multi-line comments.

With Multiple Hashtags (#)

In Python, we may use hashtags (#) multiple times to construct multiple lines of comments.

Every line with a (#) before it will be regarded as a single-line comment.

Code

```
# it is a
# comment
# extending to multiple lines
```

In this case, each line is considered a comment, and they are all omitted.

Using String Literals

Because Python overlooks string expressions that aren't allocated to a variable, we can utilize them as comments.

Code

```
'it is a comment extending to multiple lines'
```

Python Docstring

The strings enclosed in triple quotes that come immediately after the defined function are called Python docstring. It's designed to link documentation developed for Python modules, methods, classes, and functions together. It's placed just beneath the function, module, or class to explain what they perform. The docstring is then readily accessible in Python using the __doc__ attribute.

Code

```
# Code to show how we use docstrings in Python

def add(x, y):
 """This function adds the values of x and y"""
 return x + y

# Displaying the docstring of the add function
print( add.__doc__ )
```

Output:

```
This function adds the values of x and y
```

Python If-else statements

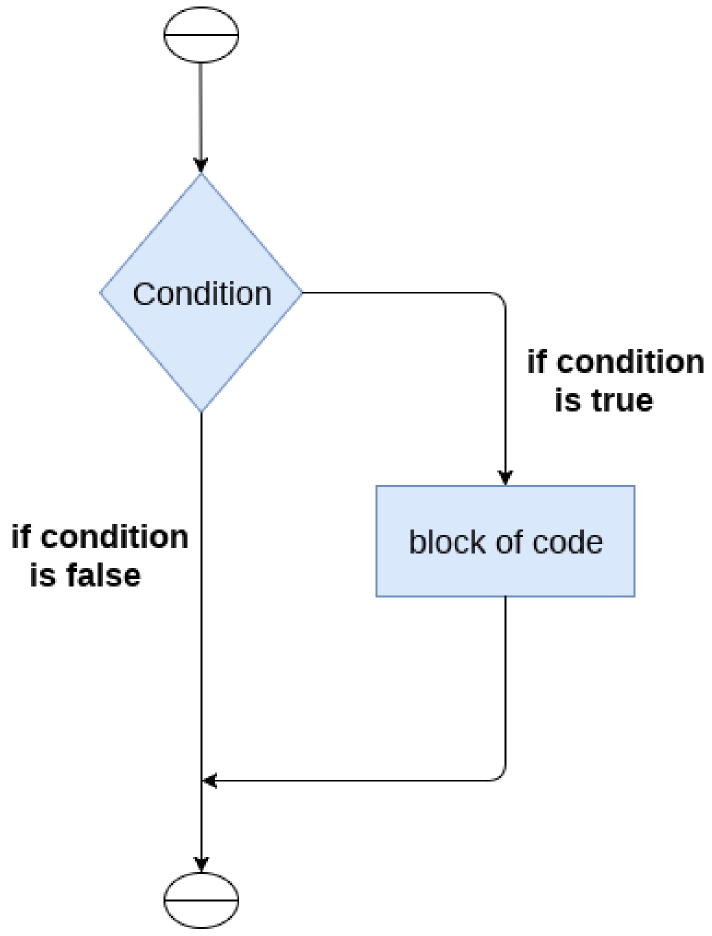
Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

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

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

The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block.



The syntax of the if-statement is given below.

```
1. if expression:
2.  statement
```

Example 1

```
# Simple Python program to understand the if statement
num = int(input("enter the number:"))
if num%2 == 0:
    print("The Given number is an even number")
```

Output:

```
enter the number: 10

The Given number is an even number
```

Example 2 : Program to print the largest of the three numbers.

```
# Simple Python Program to print the largest of the three numbers.
a = int (input("Enter a: "));
b = int (input("Enter b: "));
c = int (input("Enter c: "));
```

```
if a>b and a>c:
    print ("From the above three numbers given a is largest");
if b>a and b>c:
    print ("From the above three numbers given b is largest");
if c>a and c>b:
    print ("From the above three numbers given c is largest");
```

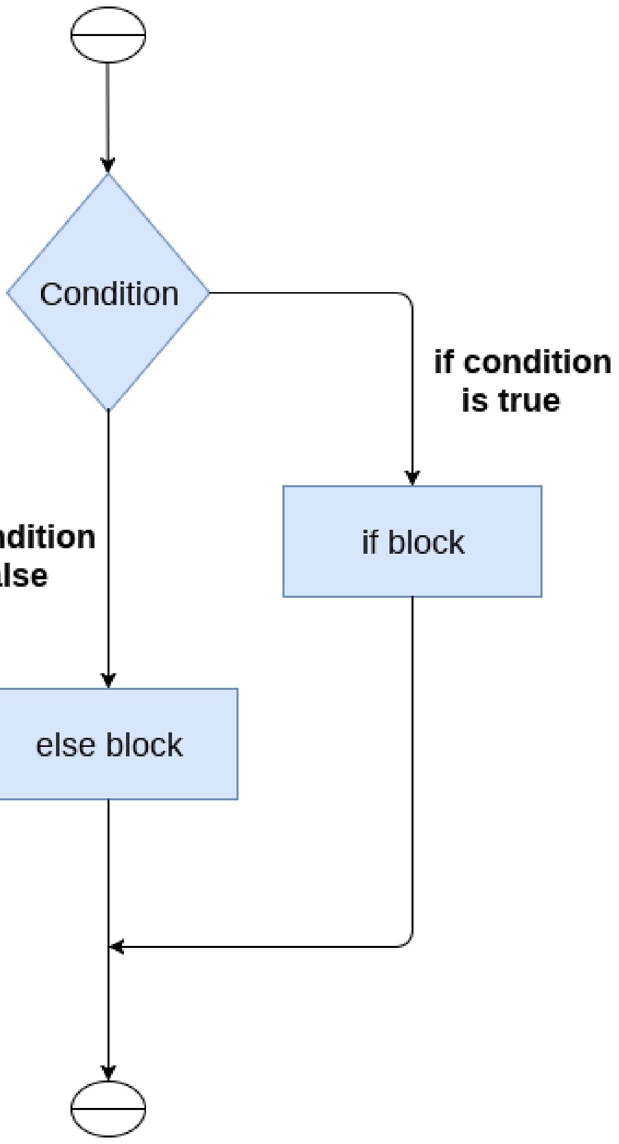
Output:

```
Enter a: 100

Enter b: 120

Enter c: 130

From the above three numbers given c is largest
```

The if-else statement

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

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



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

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

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

```
# Simple Python Program to check whether a person is eligible to vote or not.
age = int (input("Enter your age: "))
if age>=18:
    print("You are eligible to vote !!");
else:
    print("Sorry! you have to wait !!");
```

Output:

```
Enter your age: 90

You are eligible to vote !!
```

Example 2: Program to check whether a number is even or not.

```
# Simple Python Program to check whether a number is even or not.
num = int(input("enter the number:"))
if num%2 == 0:
    print("The Given number is an even number")
else:
    print("The Given Number is an odd number")
```

Output:

```
enter the number: 10

The Given number is even number
```

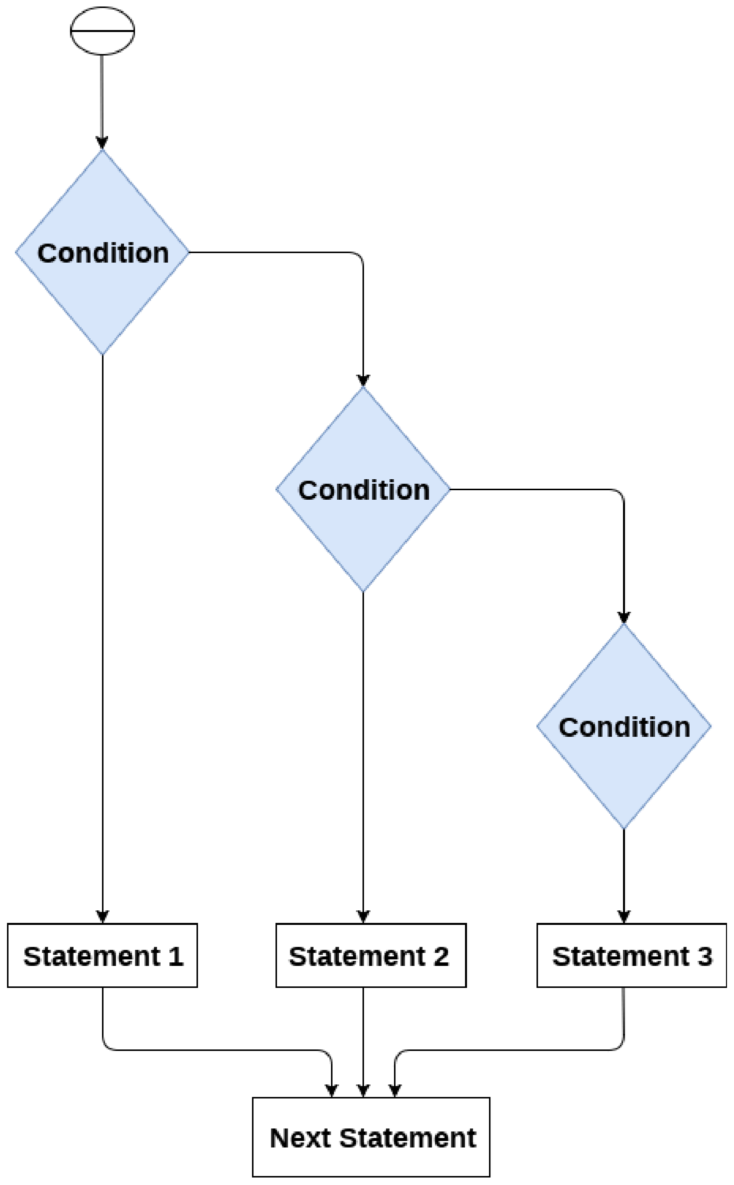
The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.

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

The syntax of the elif statement is given below.

```
if expression 1:
 # block of statements

elif expression 2:
 # block of statements

elif expression 3:
 # block of statements

else:
 # block of statements
```



Example 1

```
# Simple Python program to understand elif statement
number = int(input("Enter the number?"))
if number==10:
    print("The given number is equals to 10")
elif number==50:
    print("The given number is equal to 50");
elif number==100:
    print("The given number is equal to 100");
else:
    print("The given number is not equal to 10, 50 or 100");
```

Output:

```
Enter the number?15

The given number is not equal to 10, 50 or 100
```

Example 2

```
# Simple Python program to understand elif statement
marks = int(input("Enter the marks? "))
if marks > 85 and marks <= 100:
    print("Congrats ! you scored grade A ...")
elif marks > 60 and marks <= 85:
    print("You scored grade B + ...")
elif marks > 40 and marks <= 60:
    print("You scored grade B ...")
elif (marks > 30 and marks <= 40):
    print("You scored grade C ...")
else:
    print("Sorry you are fail ?")
```

Output:

```
Enter the marks? 89

Congrats ! you scored grade A ...
```

Python Loops

We can run a single statement or set of statements repeatedly using a loop command.

The following sorts of loops are available in the Python programming language.

Sr.No.	Name of the loop	Loop Type & Description
1	**While loop**	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	**For loop**	This type of loop executes a code block multiple times and abbreviates the code that manages the loop variable.
3	**Nested loops**	We can iterate a loop inside another loop.

Loop Control Statements

Statements used to control loops and change the course of iteration are called control statements.

Let us quickly go over the definitions of these loop control statements.

Sr. No.	Name of the control statement	Description
1	**Break statement**	This command terminates the loop's execution and transfers the program's control to the statement next to the loop.
2	**Continue statement**	This command skips the current iteration of the loop. The statements following the continue statement are not executed once the Python interpreter reaches the continue statement.
3	**Pass statement**	The pass statement is used when a statement is syntactically necessary, but no code is to be executed.

Code

```
numbers = [4, 2, 6, 7, 3, 5, 8, 10, 6, 1, 9, 2]
square = 0
squares = []

for value in numbers:
    square = value ** 2
    squares.append(square)

print("The list of squares is", squares)
```

Output:

```
The list of squares is [16, 4, 36, 49, 9, 25, 64, 100, 36, 1, 81, 4]
```

Using else Statement with for Loop

As already said, a for loop executes the code block until the sequence element is reached.

Only if the execution is complete does the else statement comes into play. It won't be executed if we exit the loop or if an error is thrown.

Here is a code to better understand if-else statements.

Code

```
string = "Python Loop"

for s in a string:
    if s == "o":
        print("If block")
    else:
        print(s)
```

Output:

```
P
y
t
h
If block
n

L
If block
If block
p
```

Now similarly, using else with for loop.

Syntax:

```
for value in sequence:
    # executes the statements until sequences are exhausted
```

```
else:
    # executes these statements when for loop is completed
```

Code

```
tuple_ = (3, 4, 6, 8, 9, 2, 3, 8, 9, 7)

for value in tuple_:

    if value % 2 != 0:

    print(value)

else:

    print("These are the odd numbers present in the tuple")
```

Output:

```
3
9
3
9
7
These are the odd numbers present in the tuple
```

The range() Function

With the help of the range() function, we may produce a series of numbers. range(10) will produce values between 0 and 9. (10 numbers).

We can give specific start, stop, and step size values in the manner range(start, stop, step size). If the step size is not specified, it defaults to 1.

Code

```
print(range(15))

print(list(range(15)))

print(list(range(4, 9)))

print(list(range(5, 25, 4)))
```

Output:

```
range(0, 15)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

[4, 5, 6, 7, 8]

[5, 9, 13, 17, 21]
```

Code

```
tuple_ = ("Python", "Loops", "Sequence", "Condition", "Range")

for iterator in range(len(tuple_)):

    print(tuple_[iterator].upper())
```

Output:

```
PYTHON

LOOPS

SEQUENCE

CONDITION

RANGE
```

While Loop

While loops are used in Python to iterate until a specified condition is met. However, the statement in the program that follows the while loop is executed once the condition changes to false.

Syntax of the while loop is:

```
while <condition>:

    { code block }
```

Code

```
counter = 0

while counter < 10:

    counter = counter + 3
```

```
        print("Python Loops")
```

Output:

```
Python Loops

Python Loops

Python Loops

Python Loops
```

Using else Statement with while Loops

As discussed earlier in the for loop section, we can use the else statement with the while loop also. It has the same syntax.

Code

```
counter = 0
while (counter < 10):
    counter = counter + 3
    print("Python Loops")
else:
    print("Code block inside the else statement")
```

Output:

```
Python Loops

Python Loops

Python Loops

Python Loops

Code block inside the else statement
```

Single statement while Block

The loop can be declared in a single statement, as seen below. This is similar to the if-else block, where we can write the code block in a single line.

Code

```
# Python program to show how to write a single statement while loop

counter = 0

while (count < 3): print("Python Loops")
```

Loop Control Statements

Now we will discuss the loop control statements in detail. We will see an example of each control statement.

Continue Statement

It returns the control to the beginning of the loop.

Code

```
for string in "Python Loops":
    if string == "o" or string == "p" or string == "t":
        continue
    print('Current Letter:', string)
```

Output:

```
Current Letter: P
Current Letter: y
Current Letter: h
Current Letter: n
Current Letter:
Current Letter: L
Current Letter: s
```

Break Statement

It stops the execution of the loop when the break statement is reached.

Code

```
for string in "Python Loops":
    if string == 'L':
        break
    print('Current Letter: ', string)
```

Output:

```
Current Letter:  P
Current Letter:  y
Current Letter:  t
Current Letter:  h
Current Letter:  o
Current Letter:  n
Current Letter:
```

Pass Statement

Pass statements are used to create empty loops. Pass statement is also employed for classes, functions, and empty control statements.

Code

```
# Python program to show how the pass statement works
for string in "Python Loops":
    pass
print( 'Last Letter:', string)
```

Output:

```
Last Letter: s
```

Python String

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes.

Each character is encoded in the ASCII or Unicode character. So we can say that Python strings are also called the collection of Unicode characters.

Consider the following example in Python to create a string.

Syntax:

```
str = "Hi Python !"
```

Creating String in Python

We can create a string by enclosing the characters in single-quotes or double- quotes. Python also provides triple-quotes to represent the string, but it is generally used for multiline string or **docstrings**.

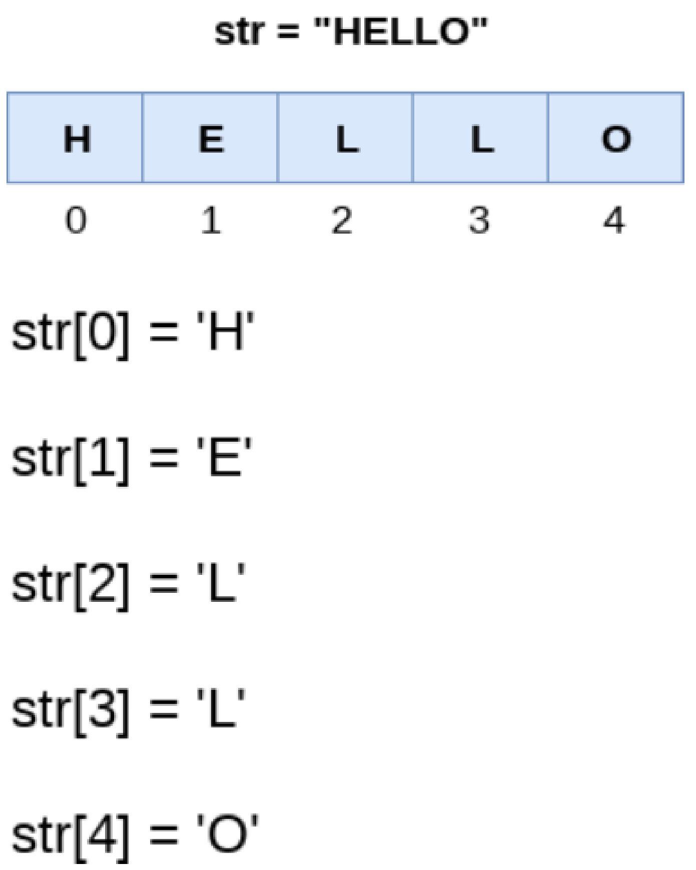
```
str1 = 'Hello Python'
print(str1)

str2 = "Hello Python"
print(str2)

str3 = '''Triple quotes are generally used for
 represent the multiline or
 docstring'''
print(str3)
```

Output:

```
Hello Python
Hello Python
Triple quotes are generally used for
    represent the multiline or
    docstring
```

Strings indexing and splitting

Like other languages, the indexing of the Python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.



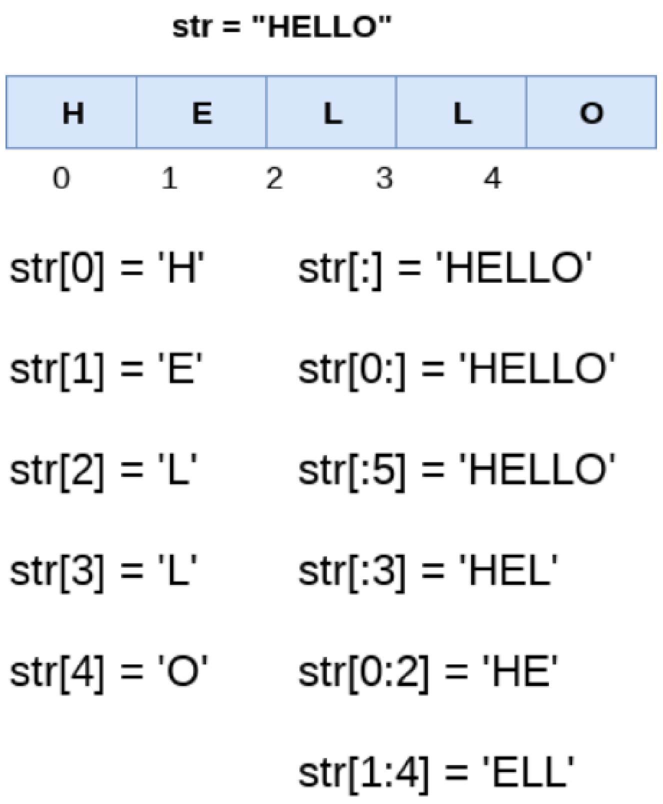
Consider the following example:

```
str = "HELLO"
print(str[0])
print(str[1])
print(str[2])
print(str[3])
print(str[4])
# It returns the IndexError because 6th index doesn't exist
print(str[6])
```

Output:

```
H
E
L
L
O
IndexError: string index out of range
```

As shown in Python, the slice operator [] is used to access the individual characters of the string. However, we can use the : (colon) operator in Python to access the substring from the given string. Consider the following example.



Here, we must notice that the upper range given in the slice operator is always exclusive i.e., if str = 'HELLO' is given, then str[1:3] will always include str[1] = 'E', str[2] = 'L' and nothing else.

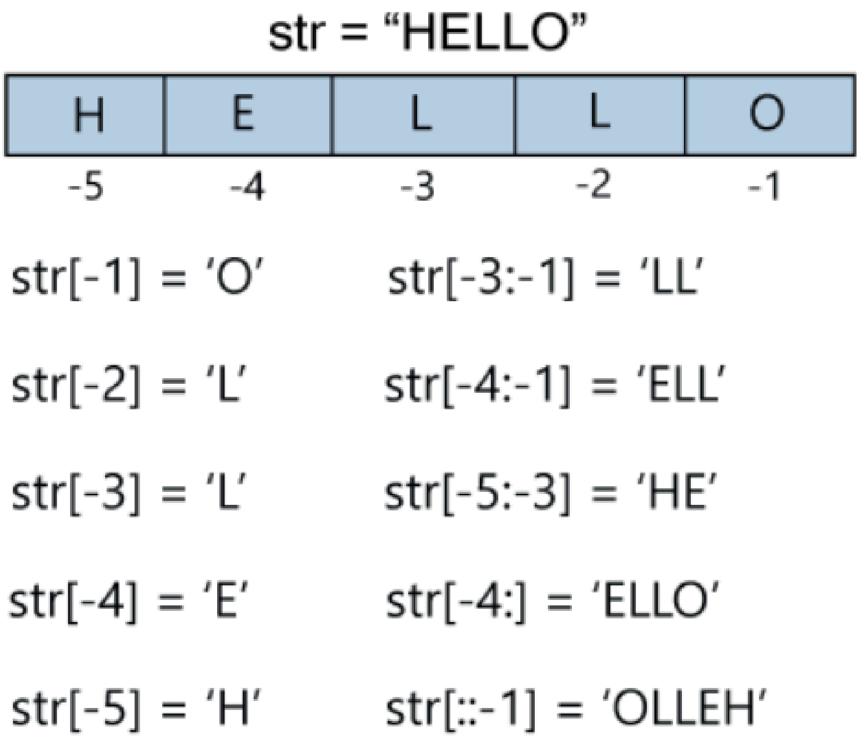
Consider the following example:

```
# Given String
str = "JAVATPOINT"
    # Start Oth index to end
print(str[0:])
    # Starts 1th index to 4th index
print(str[1:5])
    # Starts 2nd index to 3rd index
print(str[2:4])
    # Starts 0th to 2nd index
print(str[:3])
    #Starts 4th to 6th index
print(str[4:7])
```

Output:

```
JAVATPOINT
AVAT
VA
JAV
TPO
```

We can do the negative slicing in the string; it starts from the rightmost character, which is indicated as -1. The second rightmost index indicates -2, and so on. Consider the following image.



Consider the following example

```python
str = 'JAVATPOINT'
print(str[-1])
print(str[-3])
print(str[-2:])
print(str[-4:-1])
print(str[-7:-2])
# Reversing the given string
print(str[::-1])
print(str[-12])
```

Output:

```
T
I
NT
OIN
ATPOI
TNIOPTAVAJ
IndexError: string index out of range
```

Reassigning Strings

Updating the content of the strings is as easy as assigning it to a new string. The string object doesn't support item assignment i.e., A string can only be replaced with new string since its content cannot be partially replaced. Strings are immutable in Python.

Consider the following example.

Example 1

```
str = "HELLO"
str[0] = "h"
print(str)
```

Output:

```
Traceback (most recent call last):
  File "12.py", line 2, in <module>
    str[0] = "h";
TypeError: 'str' object does not support item assignment
```

However, in example 1, the string **str** can be assigned completely to a new content as specified in the following example.

Example 2

```
str = "HELLO"
print(str)
```

```
str = "hello"

print(str)
```

Output:

```
HELLO

hello
```

Deleting the String

As we know that strings are immutable. We cannot delete or remove the characters from the string. But we can delete the entire string using the **del** keyword.

```
str = "JAVATPOINT"
del str[1]
```

Output:

```
TypeError: 'str' object doesn't support item deletion
```

Now we are deleting entire string.

```
str1 = "JAVATPOINT"
del str1
print(str1)
```

Output:

```
NameError: name 'str1' is not defined
```

String Operators

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

Example

Consider the following example to understand the real use of Python operators.

```
str = "Hello"
str1 = " world"
print(str*3) # prints HelloHelloHello
print(str+str1)# prints Hello world
print(str[4]) # prints o
print(str[2:4]); # prints ll
print('w' in str) # prints false as w is not present in str
print('wo' not in str1) # prints false as wo is present in str1.
print(r'C://python37') # prints C://python37 as it is written
print("The string str : %s"%(str)) # prints The string str : Hello
```

Output:

```
HelloHelloHello
Hello world
o
ll
False
False
C://python37
The string str : Hello
```

Python String Formatting

Escape Sequence

Let's suppose we need to write the text as - They said, "Hello what's going on?"- the given statement can be written in single quotes or double quotes but it will raise the **SyntaxError** as it contains both single and double-quotes.

Example

Consider the following example to understand the real use of Python operators.

```
str = "They said, "Hello what's going on?""

print(str)
```

Output:

```
SyntaxError: invalid syntax
```

We can use the triple quotes to accomplish this problem but Python provides the escape sequence.

The backslash(/) symbol denotes the escape sequence. The backslash can be followed by a special character and it interpreted differently. The single quotes inside the string must be escaped. We can apply the same as in the double quotes.

Example -

```python
# using triple quotes
print('''They said, "What's there?"''')


# escaping single quotes
print('They said, "What\'s going on?"')


# escaping double quotes
print("They said, \"What's going on?\"")
```

Output:

```
They said, "What's there?"
They said, "What's going on?"
They said, "What's going on?"
```

The list of an escape sequence is given below:

Sr.	Escape Sequence	Description	Example
1.	\newline	It ignores the new line.	print("Python1 \ Python2 \ Python3") **Output:** Python1 Python2 Python3
2.	\\	Backslash	print("\\") **Output:** \
3.	\'	Single Quotes	print('\'') **Output:** '
4.	\\"	Double Quotes	print("\"") **Output:** "
5.	\a	ASCII Bell	print("\a")
6.	\b	ASCII Backspace(BS)	print("Hello \b World") **Output:** Hello World
7.	\f	ASCII Formfeed	print("Hello \f World!") Hello World!
8.	\n	ASCII Linefeed	print("Hello \n World!") **Output:**

			Hello World!
9.	\r	ASCII Carriege Return(CR)	print("Hello \r World!") **Output:** World!
10.	\t	ASCII Horizontal Tab	print("Hello \t World!") **Output:** Hello World!

11.	\v	ASCII Vertical Tab	print("Hello \v World!")
			
			Output:
			
			Hello
			
			  World!

12.	\ooo	Character with octal value	print("\110\145\154\154\157") **Output:** Hello
13	\xHH	Character with hex value.	print("\x48\x65\x6c\x6c\x6f") **Output:** Hello

Here is the simple example of escape sequence.

```
print("C:\\Users\\DEVANSH SHARMA\\Python32\\Lib")

print("This is the \n multiline quotes")

print("This is \x48\x45\x58 representation")
```

Output:

```
C:\Users\DEVANSH SHARMA\Python32\Lib

This is the

 multiline quotes

This is HEX representation
```

We can ignore the escape sequence from the given string by using the raw string. We can do this by writing **r** or **R** in front of the string. Consider the following example.

```
print(r"C:\\Users\\DEVANSH SHARMA\\Python32")
```

Output:

```
C:\\Users\\DEVANSH SHARMA\\Python32
```

The format() method

The **format()** method is the most flexible and useful method in formatting strings. The curly braces {} are used as the placeholder in the string and replaced by the **format()** method argument. Let's have a look at the given an example:

```
# Using Curly braces

print("{} and {} both are the best friend".format("Devansh","Abhishek"))


#Positional Argument

print("{1} and {0} best players ".format("Virat","Rohit"))


#Keyword Argument

print("{a},{b},{c}".format(a = "James", b = "Peter", c = "Ricky"))
```

Output:

```
Devansh and Abhishek both are the best friend

Rohit and Virat best players

James,Peter,Ricky
```

Python String Formatting Using % Operator

Python allows us to use the format specifiers used in C's printf statement. However, Python provides an additional operator %, which is used as an interface between the format specifiers and their values. In other words, we can say that it binds the format specifiers to the values.

Consider the following example.

```
Integer = 10;

Float = 1.290

String = "Devansh"

print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am string ... My value is %s"%(Integer,Float,String))
```

Output:

```
Hi I am Integer ... My value is 10

Hi I am float ... My value is 1.290000

Hi I am string ... My value is Devansh
```

Python String functions

Python provides various in-built functions that are used for string handling. Many String fun

Python String capitalize() Method

Python **capitalize()** method converts first character of the string into uppercase without altering the whole string. It changes the first character only and skips rest of the string unchanged.

Signature

1. capitalize()

Parameters

No parameter is required.

Return Type

It returns a modified string.

Python String Capitalize() Method Example 1

```
str = "studypoint"

str2 = str.capitalize()

print("Old value:", str)

print("New value:", str2)
```

Output:

```
Old value: studypoint

New value: studypoint
```

Python String Casefold() Method

Python **Casefold()** method returns a lowercase copy of the string. It is more simillar to lowercase method except it revomes all case distinctions present in the string.

For example in German, 'β' is equivelent to "ss". Since it is already in lowercase, lowercase do nothing and prints 'β' whereas casefold converts it to "ss".

Signature

1. casefold()

Parameters

No parameter is required.

Return Type

It returns lowercase string.

Python Version

This function was introduced in **Python 3.3**.

Python String Casefold() Method Example 1

```
# Python casefold() function example
# Variable declaration
str = "STUDYPOINT "
# Calling function
str2 = str.casefold()
# Displaying result
print("Old value:", str)
print("New value:", str2)
```

Output:

```
Old value: STUDYPOINT

New value: studypoint
```

Python String Casefold() Method Example 2

The strength of casefold, it not only converts into lowercase but also converts strictly. See an example below **'β' is converted into "ss"**.

```
# Python casefold() function example

# Variable declaration

str = "STUDYPOINT - β"

# Calling function

str2 = str.casefold()

# Displaying result

print("Old value:", str)

print("New value:", str2)
```

Output:

```
Old value: STUDYPOINT  - β

New value: studypoint ? ss
```

Python String Center() Method

Python **center()** method alligns string to the center by filling paddings left and right of the string. This method takes two parameters, first is a width and second is a fillchar which is optional. The fillchar is a character which is used to fill left and right padding of the string.

Signature

1. center(width[,fillchar])

Parameters

- **width** (required)
- **fillchar** (optional)

Return Type

It returns modified string.

Python String Center() Method Example 1:default fill char

Here, we did not pass second parameter. By default it takes spaces.

```
# Python center() function example
# Variable declaration
str = "Hello Studypoint"
# Calling function
str2 = str.center(20)
# Displaying result
print("Old value:", str)
print("New value:", str2)
```

Output:

```
Old value: Hello Studypoint
New value:   Hello Studypoint
```

Python String Center() Method Example 2

Here, we are providing padding char (optional) parameter as #. See the example.

```
# Python center() function example
# Variable declaration
str = "Hello Studypoint"
# Calling function
str2 = str.center(20,'#')
```

```
# Displaying result
print("Old value:", str)
print("New value:", str2)
```

Output:

```
Old value: Hello Studypoint
New value: ##Hello Studypoint##
```

Python String Count() Method

It returns the number of occurences of substring in the specified range. It takes three parameters, first is a substring, second a start index and third is last index of the range. Start and end both are optional whereas substring is required.

Signature

1. count(sub[, start[, end]])

Parameters

- **sub** (required)
- **start** (optional)
- **end** (optional)

Return Type

It returns number of occurrences of substring in the range.

Let's see some examples to understand the count() method.

Python String Count() Method Example 1

```
# Python count() function example
# Variable declaration
str = "Hello Studypoint"
str2 = str.count('t')
# Displaying result
print("occurences:", str2)
```

Output:

```
occurences: 2
```

The below example is using all three parameters and returning result from the specified range.

Python String Count() Method Example 4

```
# Python count() function example
# Variable declaration
str = "ab bc ca de ed ad da ab bc ca"
oc = str.count('a', 3, 8)
# Displaying result
print("occurences:", oc)
```

Output:

```
occurences: 1
```

Python String startswith() Method

Python startswith() method returns either True or False. It returns True if the string starts with the prefix, otherwise False. It takes two parameters start and end. Start is a starting index from where searching starts and end index is where searching stops.

Signature

1. startswith(prefix[, start[, end]])

Parameters

prefix : A string which is to be checked.

start : Start index from where searching starts.

end : End index till there searching performs.

Both start and end are optional parameters.

Return

It returns boolean value either True or False.

Let's see some examples of startswith() method to understand it's functionality.

Python String startswith() Method Example 1

Let's first create a simple example which prints True if the string starts with the prefix.

```
# Python String startswith() method
# Declaring variable
str = "Hello StudyPoint"
# Calling function
str2 = str.startswith("Hello")
# Displaying result
print (str2)
```

Output:

```
True
```

Python String endswith() Method

Python **endswith()** method returns true of the string ends with the specified substring, otherwise returns false.

Signature

1. endswith(suffix[, start[, end]])

Parameters

- **suffix** : a substring
- **start** : start index of a range
- **end** : last index of the range

Start and end both parameters are optional.

Return Type

It returns a boolean value either True or False.

Let's see some examples to understand the endswith() method.

Python String endswith() Method Example 1

A simple example which returns true because it ends with dot (.).

```
# Python endswith() function example
# Variable declaration
str = "Hello this is Studypoint."
isends = str.endswith(".")
# Displaying result
print(isends)
```

Output:

```
True
```

---

find() Function in String

Python *finds ()* method finds a substring in the whole String and returns the index of the first match. It returns -1 if the substring does not match.

Syntax:

The syntax of the *find()* method is as follows:

```
1. string.find(substring, start, end)
```

Parameters:

Where:

- **String** is the String in which you want to search for the substring
- **Substring** is the String you want to search for
- **Start** is the index from which the search should begin (optional, defaults to 0)
- **End** is the index at which the search should end (optional, defaults to the end of the String)

Return Type:

The find() method returns the index of the first occurrence of that substring in the given range. In case the substring is not found, it returns -1.

Some Examples:

Here's an example of using the find() method to search for a substring within a string:

Example 1:

Program to find a substring in a string:

```
string = "Hello, world!"
substring = "world"
index = string.find(substring)
print(index)
```

Output:

```
7
```

Example 2:

Finding a substring in a string with start and end indices:

```
string = "Hello, world!"
substring = "l"
start = 3
end = 8
index = string.find(substring, start, end)
print(index)
```

Output:

```
3
```

The format() Method:

A string with one or more placeholders is needed to begin using the format() method. Placeholders are set apart by wavy supports, with discretionary positional or named contentions inside. An elementary illustration:

```
name = "John"
age = 30
formatted_string = "My name is {} and I'm {} years old.".format(name, age)
print(formatted_string)
```

Output:

```
My name is John and I'm 30 years old.
```

Positional and Named Arguments:

The configuration() strategy upholds both positional and named contentions, giving you adaptability by the way you give values. Take for instance the following:

```
product = "Phone"

price = 999.99

formatted_string = "The {0} costs ${1:.2f}".format(product, price)

print(formatted_string)
```

Output:

```
The Phone costs $999.99
```

The positional placeholders "0" and "1" in this example correspond to the order of the arguments passed to the format() method. The : .2f inside the subsequent placeholder guarantees that the cost is shown with two decimal spots.

Python String index() Method

Python **index()** method is same as the find() method except it returns error on failure. This method returns index of first occurred substring and an error if there is no match found.

Signature

1. index(sub[, start[, end]])

Parameters

- **sub** : substring
- **start** : start index a range
- **end** : last index of the range

Return Type

If found it returns an index of the substring, otherwise an error ValueError.

Let's see some examples to understand the index() method.

Python String index() Method Example 1

```
# Python index() function example

# Variable declaration

str = "Welcome to the Studypoint.."
```

```
# Calling function
str2 = str.index("dy")
# Displaying result
print(str2)
```

Output:

```
18
```

Python String index() Method Example 2

An error is thrown if the substring is not found.

```
# Python index() function example
# Variable declaration
str = "Welcome to the Studypoint.."
# Calling function
str2 = str.index("ate")
# Displaying result
print(str2)
```

Output:

```
ValueError: substring not found
```

Python String index() Method Example 3

We can also pass start and end index as parameters to make process more customized.

```
# Python index() function example
# Variable declaration
str = "Welcome to the Studypoint."
# Calling function
str2 = str.index("p",19,21)
# Displaying result
print("p is present at :",str2,"index")
```

Output:

```
p is present at : 20 index
```