Unit
2

Python Introduction

2.1 Introduction to Python

Python is an open-source (free) programming language that is used in web programming, data science, artificial intelligence, and many scientific applications. Learning Python allows the programmer to focus on solving problems, rather than focusing on syntax.It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation.

Python Features

1. Easy Language

Python is an easy language. It is easy to read, write, learn and understand.

- Python has a **smooth learning curve**. It is easy to learn.
- Python has a **simple syntax** and Python code is easy to understand.
- Since it's **easy to understand**, you can easily read and understand someone else's code.
- Python is also **easy to write** because of its simple syntax.

2. Readable

The Python language is designed to make developers life easy. Reading a Python code is like reading an English sentence. This is one of the key reasons that makes Python best for beginners.

Python uses **indentation** instead of **curly braces**, unlike other programming languages. This makes the code look clean and easier to understand.

3. Interpreted Language

Python is an interpreted language. It comes with the **IDLE (Interactive Development Environment)**. This is an interpreter and follows the **REPL structure (Read-Evaluate-Print-Loop)**. It executes and displays the output of one line at a time.

So it displays errors while you're running a line and displays the entire stack trace for the error.

4. Dynamically-Typed Language

You don't need to declare data type while defining a variable. The interpreter determines this at runtime based on the types of the parts of the expression. This is easy for programmers but can create runtime errors.

5. Object-Oriented

Python is object-oriented but supports both functional and object-oriented programming. Everything in Python is an object.

6. Open-Source

Python is open-source and the community is always contributing to it to improve it. It is **free** and its source code is **freely available** to the public. You can download Python from the official Python Website.

7. Large Standard Library

The standard library is large and has many **packages** and **modules** with common and important functionality. If you need something that is available in this standard library, you don't need to write it from scratch. Because of this, you can focus on more important things.

8. Platform-Independent

Python is platform-independent. If you write a program, it will run on different platforms like **Windows**, **Mac** and **Linux**. You don't need to write them separately for each platform.

9. Extensible and Embeddable

Python is **extensible**. You can use code from other languages like C++ in your Python code.It is also **embeddable**. You can embed your Python code in other languages like C++.

10. GUI Support

You can use Python to create GUI (Graphical User Interfaces).

Applications of Python Programming

Python can be used for various domains :

Domain	Description
Desktop and Web Applications	A Desktop application is one that runs stand-alone in a desktop or laptop computer for example BitTorrent, Blender, Juice while a Web application requires a Web browser to run, for example Mailman, Plone, MoinMoin.
Data Science	It is a field that uses scientific methods such as data collection; algorithms and machine learning techniques to extract, analyze and process insights from raw data.
Machine Learning	It is an application of artificial intelligence (AI) that gives systems the ability to automatically learn and improve from experience and data without being explicitly programmed
Robotics	It is a branch of engineering that deals with the conception, design, manufacture, and operation of robots.
Artificial Intelligence	It is a broad field that deals with enabling machines to demonstrate intelligence similar to human's intelligence such as decision-making, facial recognition, etc.

	Artificial intelligence incorporates other fields like Machine Learning, Robotics, Natural Language Processing(NLP), etc.
Internet of Things (IoT)	It is a field that describes the network of things that are embedded with software, and other technologies for the purpose of connecting and exchanging data with other devices over the internet.
Gaming	It is the art of designing and programming games for entertainment, educational, or experimental purposes and that runs on computers and mobile devices.
Mobile Applications	It is a computer program or app designed to run on a mobile device such as a phone, table, or watch.
Natural Language processing 3D CAD	It is a field that analyses speech in both audible speech, as well as text of a language. Python can create a 3D CAD application by using the following functionalities. - Fandango (Popular) - CAMVOX - HeeksCNC - AnyCAD - RCAM
Data visualization	Data visualization is another popular and developing area of interest. Python provides a variety of graphing libraries with all kinds of features. You can find a library to create simple graphical representation or even a more interactive plot in Python. Examples include Pandas Visualization and Plotly.
Finance	Python is increasingly being utilized in the world of finance, often in areas such as quantitative and qualitative analysis. It can be a valuable tool in determining asset price trends and predictions.
Image Processing	Python contains many libraries that are used to work with the image. Some libraries of image processing are given below. - Ooen CV - Pillow - SimpleITK

Some Drawbacks of Python are:

- Slow speed
- Memory inefficient
- Ineffective in mobile computing.
- Undeveloped database layers.
- Run time error prompt due to its dynamism.

2.2 Python Installation

Before we start Python programming, we need to have an interpreter to interpret and run our programs. There are certain online interpreters like https://ide.geeksforgeeks.org/, http://ideone.com/ or http://codepad.org/ that can be used to run Python programs without installing an interpreter.

Windows: There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) that comes bundled with the Python software downloaded from http://python.org/.

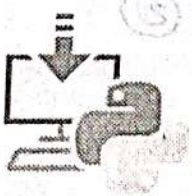
- **Python 3 Installation on Windows**

Step 1 : Download Python 3.9

To start, go to python.org/downloads and then click on the button to download the latest version of Python :

Step 2 : Run the .exe file

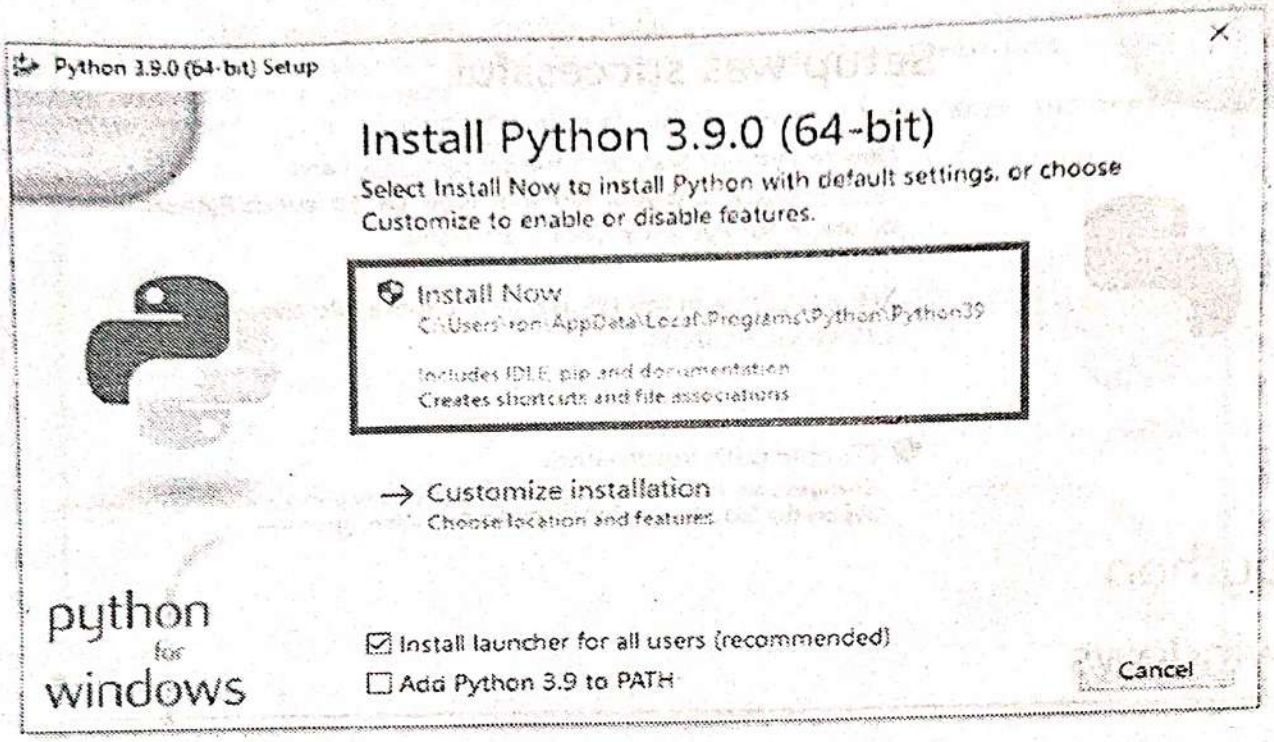
Next, run the .exe file that you just downloaded:



python-3.9.0

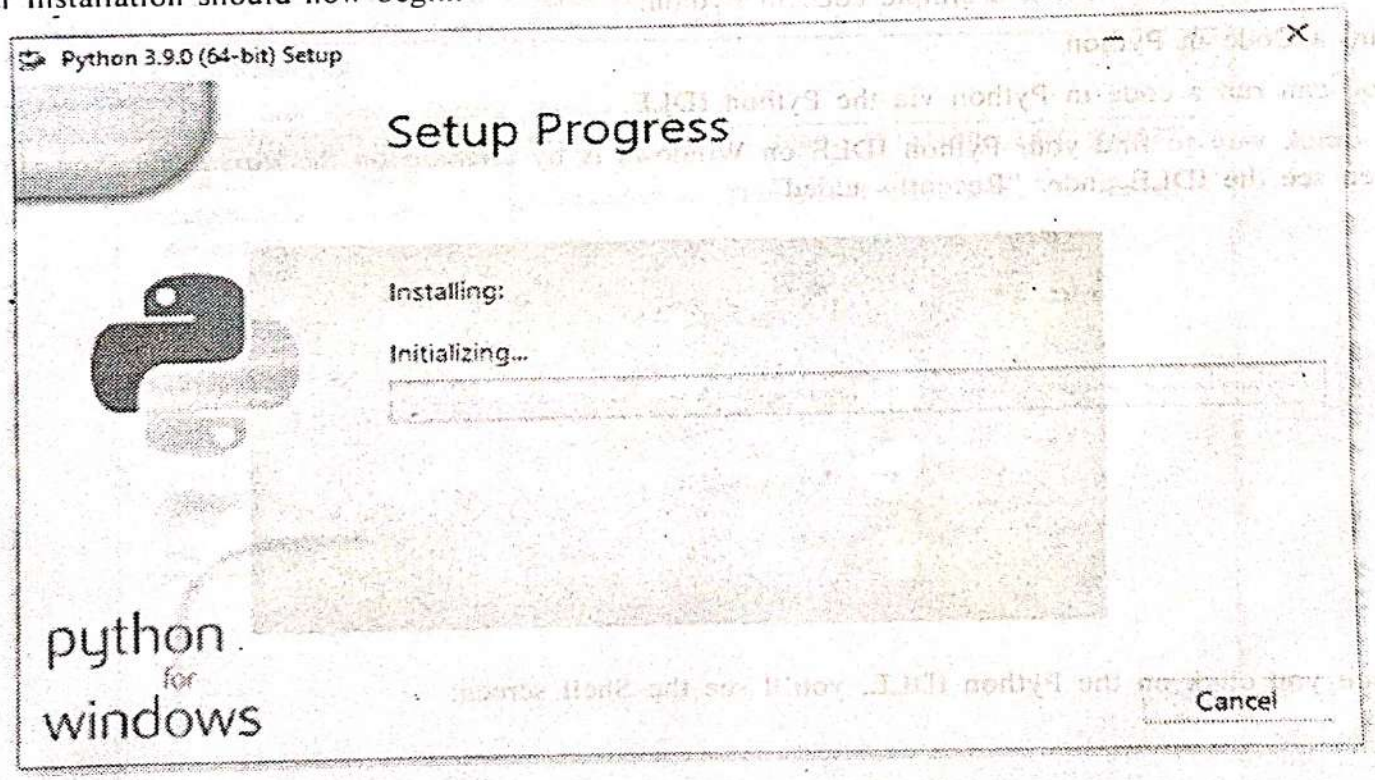
Step 3 : Install Python 3.9

You can now start the installation of Python by clicking on Install Now :

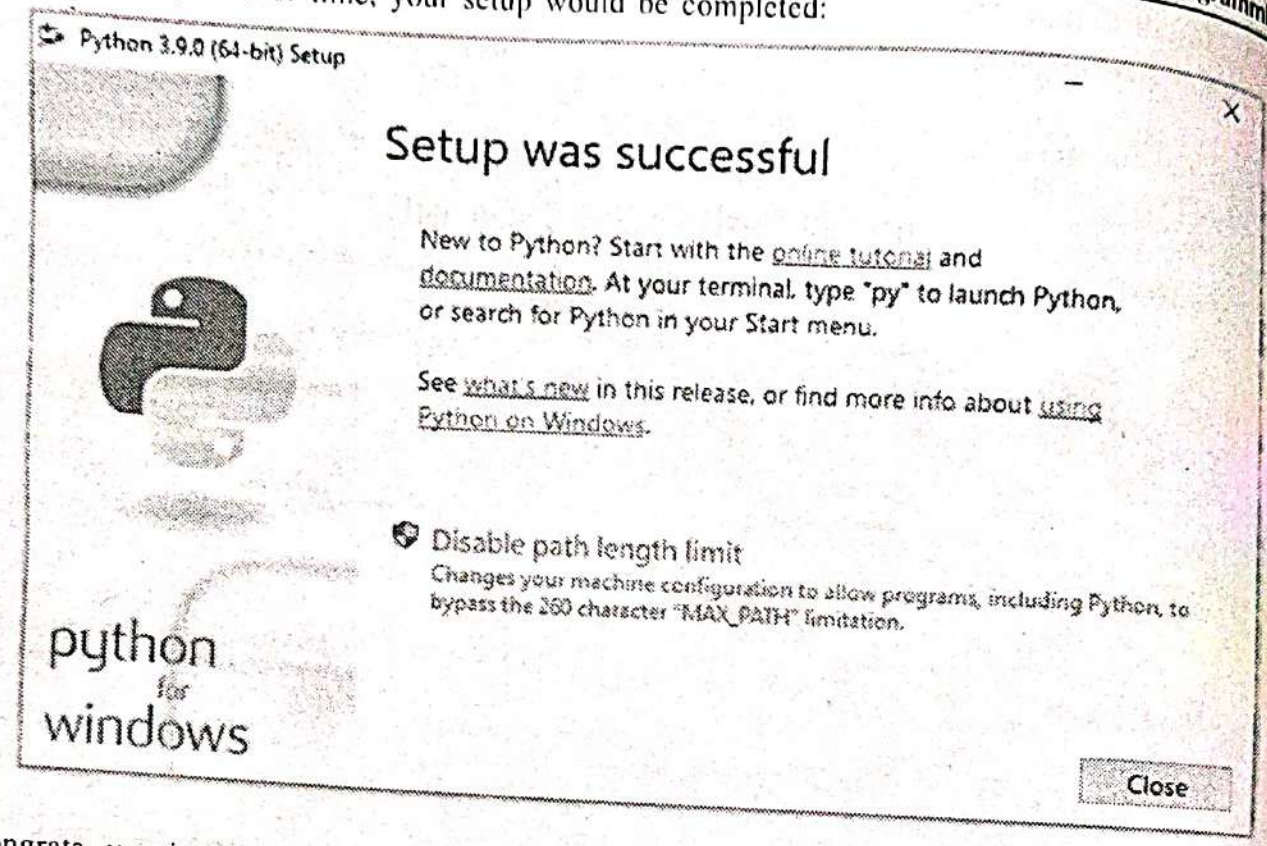


Note : that depending on your needs, you may also check the box to add Python to the Path.

Your installation should now begin:



After a short period of time, your setup would be completed:



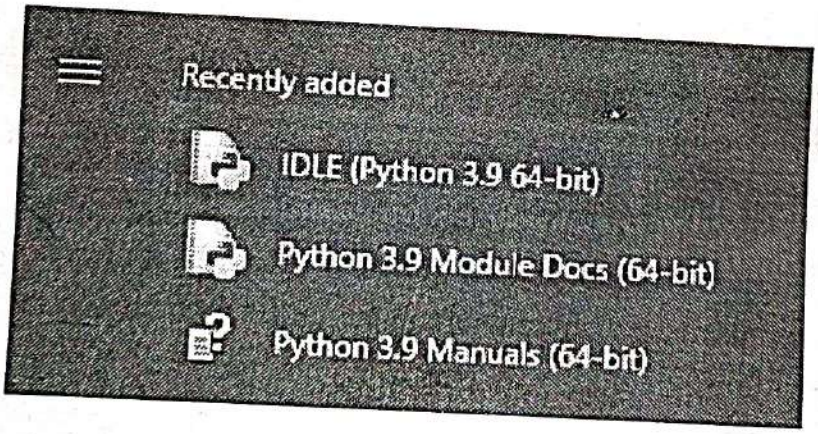
Congrats, you just installed Python on Windows!

Let's now see how to run a simple code in Python.

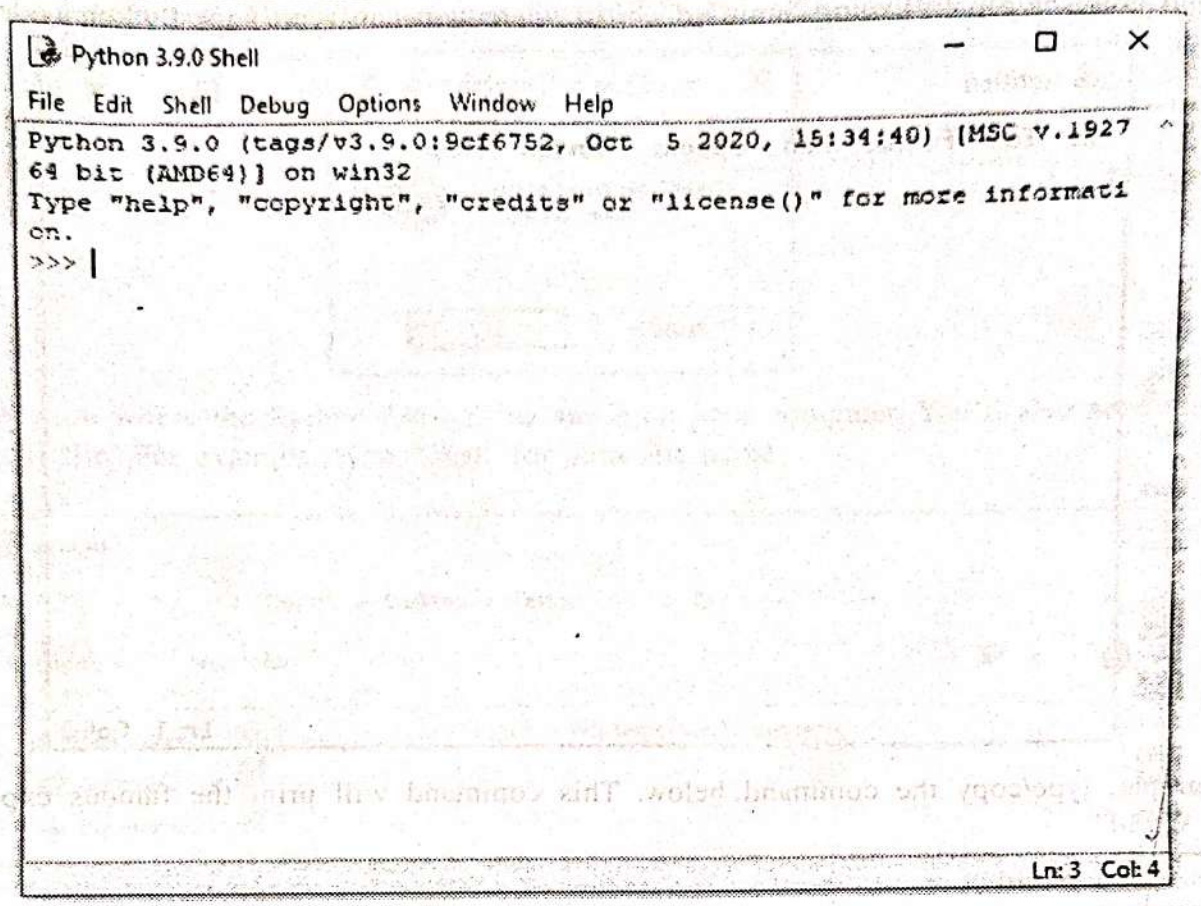
Run a Code in Python

You can run a code in Python via the Python IDLE.

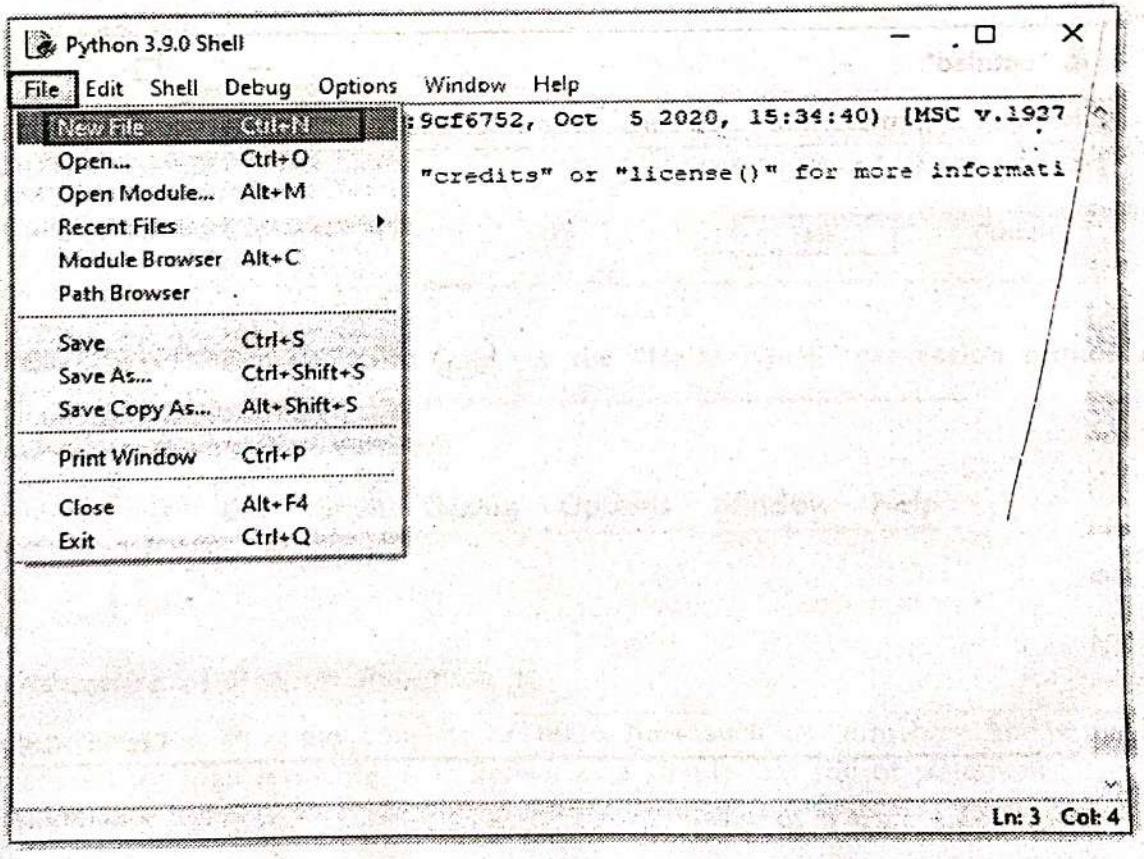
A quick way to find your Python IDLE on Windows is by clicking on the *Start* menu. You should then see the IDLE under "Recently added"



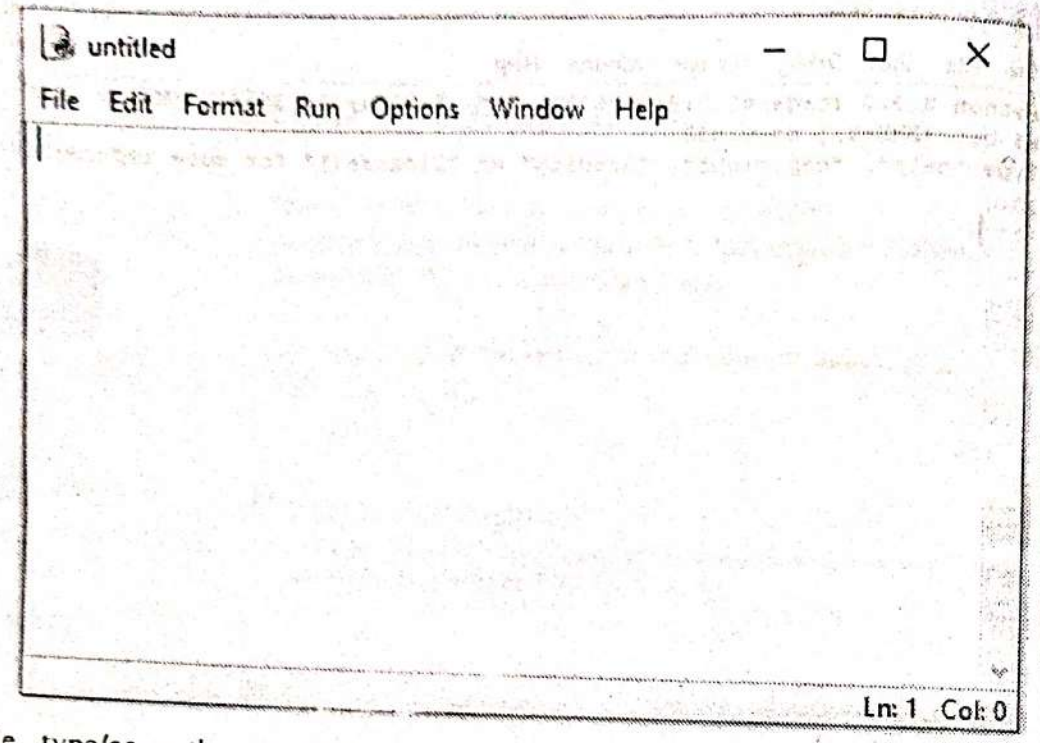
Once your click on the Python IDLE, you'll see the Shell screen:



Click on File and then select New File (alternatively, you may use the keyboard shortcut of Ctrl+N):



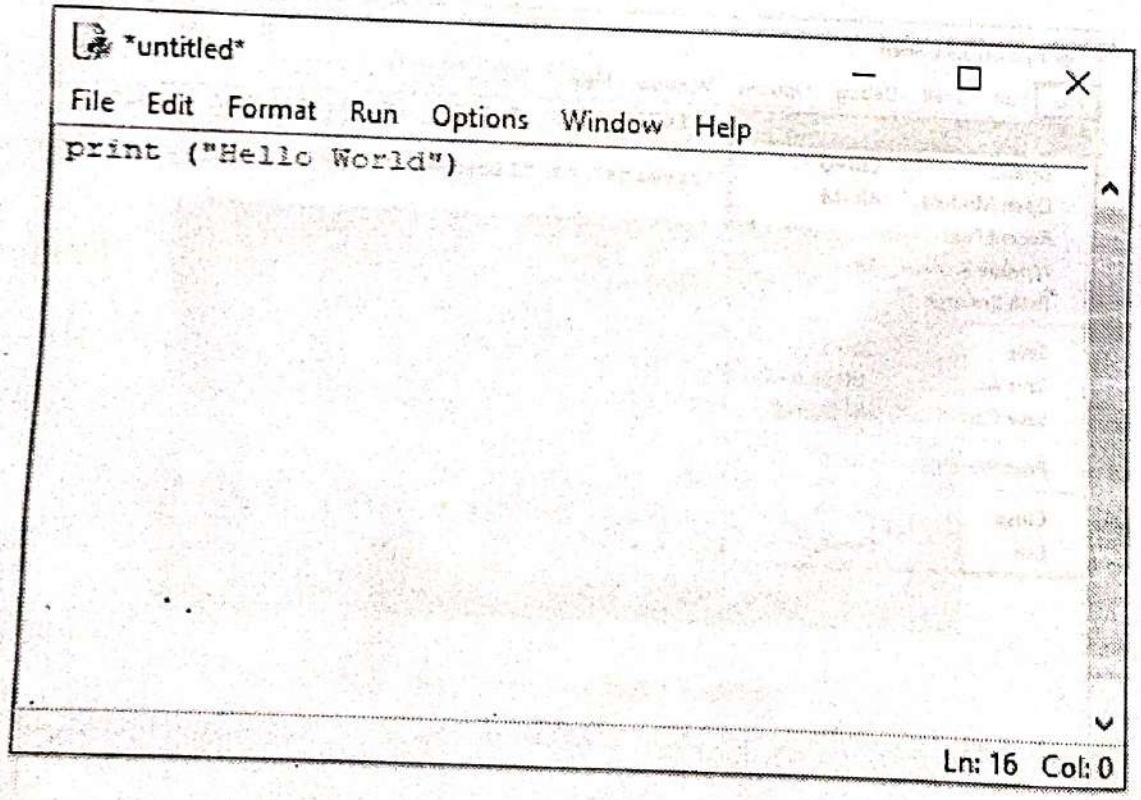
You would now see the following "untitled" box, where you can type your Python code :



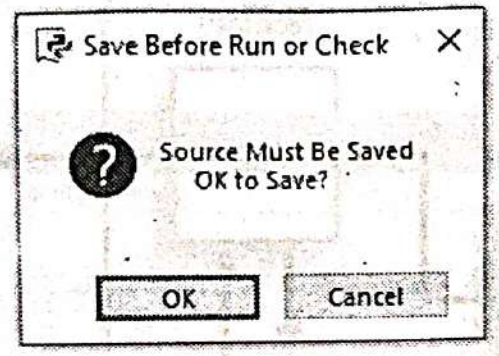
For example, type/copy the command below. This command will print the famous expression o "Hello World"

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

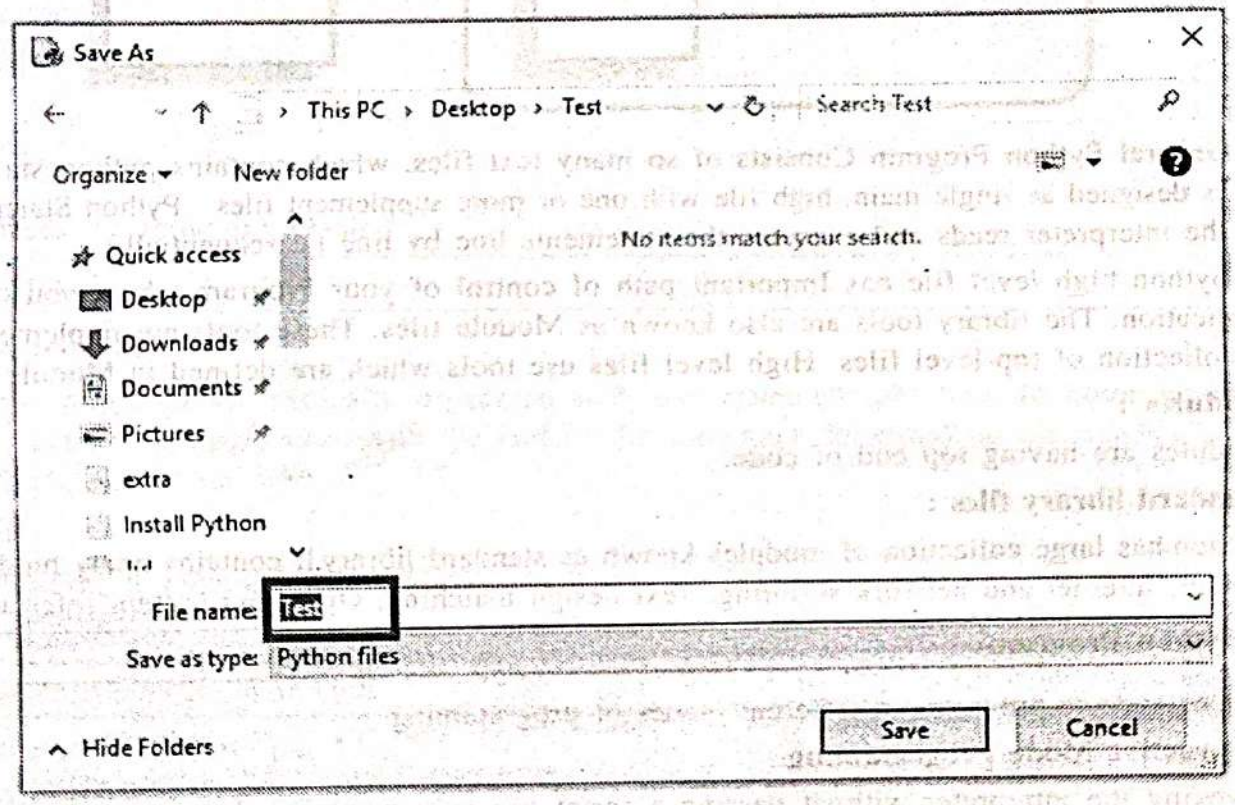
This is how the syntax would look like in the "untitled" box :



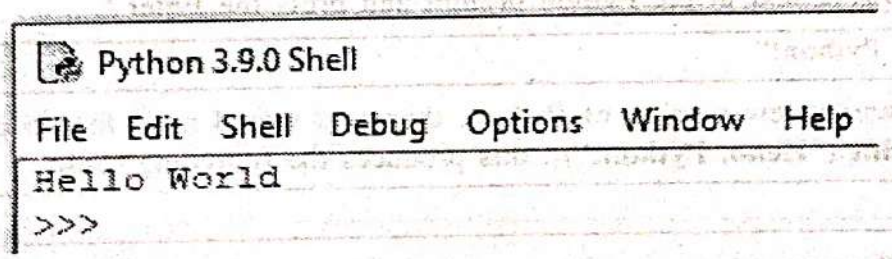
Press F5 on your keyboard. You will then get the following message to save your code :



Choose a location where the Python file will be saved on your computer. You'll also need to type a name for your file. For example, type "Test" for your file name:

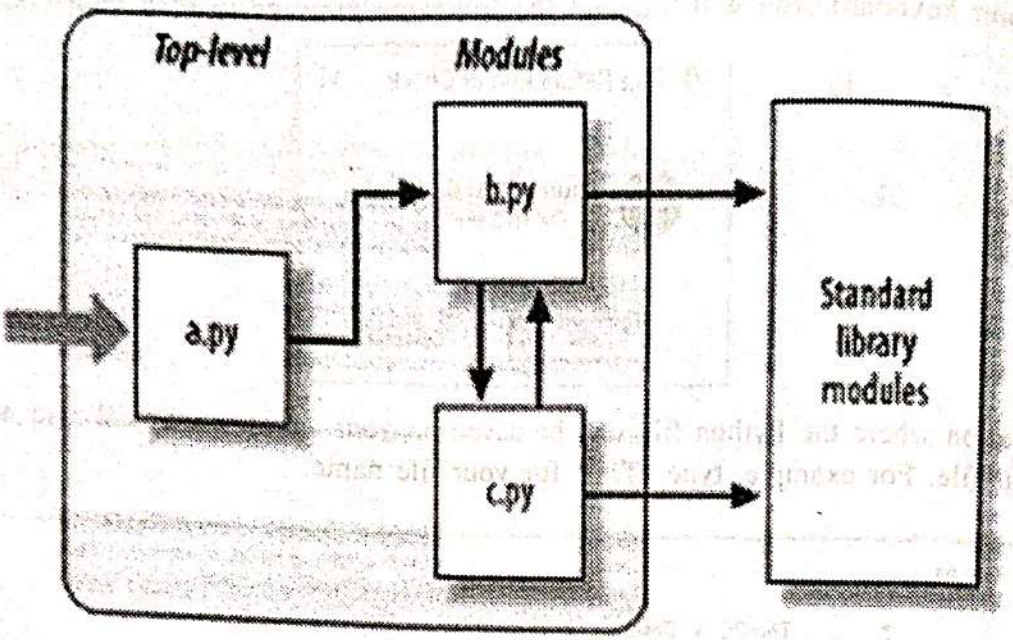


Once you're done, press Save, and you'll then see the "Hello World" expression printed on your Python Shell:



2.3 Basic Structure of Python Program :

The structure Python Program consists of three files such as :a.py,b.py and c.py. The file model a.py is chosen for high level file. it is known as a simple text file of statements. ... Files b.py and c.py are modules



In General Python Program Consists of so many text files, which contains python statements. Program is designed as single main, high file with one or more supplement files Python Statements In general, the interpreter reads and executes the statements line by line i.e sequentially.

In python high level file has Important path of control of your Program where you can start your application. The library tools are also known as Module files. These tools are implemented for making collection of top-level files. High level files use tools which are defined in Module files.

- **Modules :**

Modules are having top end of code.

- **Standard library files :**

Python has large collection of modules known as standard library.it contains many modules for GUI Design, Internet and network scripting, Text design matching, Operating system Interfaces.

First Python Program

Let us execute programs in different modes of programming.

- **Interactive Mode Programming**

Invoking the interpreter without passing a script file as a parameter brings up the following prompt "

Type the following text at the Python prompt and press the Enter "

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in **print ("Hello, Python!");**. this produces the following result "

```
Hello, Python !
```

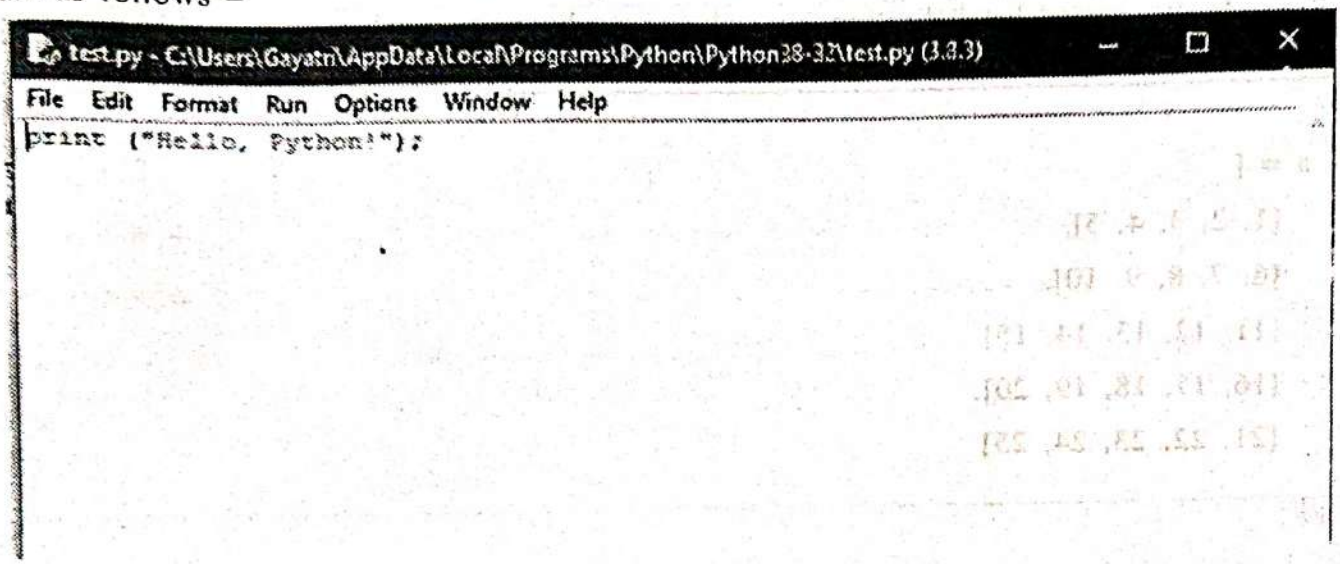
- **Script Mode Programming**

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension **.py**. Type the following source code in a test.py file "

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this rogram as follows –



This produces the following result "

```
Hello, Python!
```

Python Statements

Python programs are typically organized with one statement per line. In other words, each tatement occupies a single line, with the end of the statement delimited by the newline character hat marks the end of the line.

Example :

```
print('Welcome to python programming')
```

Output :

```
Welcome to python programming
```

Multiple Statements per Line We can also write multiple statements per line, but it is not a ood practice as it reduces the readability of the code. For Example, consider the following code.

Example :

```
a = 10; b = 20; c = b + a

print(a); print(b); print(c)
```

Output :

```
10
20
30
```

Implicit Line Continuation :

Any statement containing opening parentheses ('('), brackets ('['), or curly braces ('{') is presume to be incomplete until all matching parentheses, brackets, and braces have been encountered.

For example, the nested list definition from above can be made much more readable using implic line continuation because of the open brackets:

```
>>>
>>> a = [
...     [1, 2, 3, 4, 5],
...     [6, 7, 8, 9, 10],
...     [11, 12, 13, 14, 15],
...     [16, 17, 18, 19, 20],
...     [21, 22, 23, 24, 25]
... ]


>>> a
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15],
[16, 17, 18, 19, 20], [21, 22, 23, 24, 25]]
```

A long expression can also be continued across multiple lines by wrapping it in grouping parenthes PEP 8 explicitly advocates using parentheses in this manner when appropriate :

```
>>>
>>>someone_is_of_working_age = (
...     (person1_age >= 18 and person1_age <= 65)
...     or (person2_age >= 18 and person2_age <= 65)
...     or (person3_age >= 18 and person3_age <= 65)
... )


>>>someone_is_of_working_age
True
```

If you need to continue a statement across multiple lines, it is usually possible to use implic line continuation to do so. This is because parentheses, brackets, and curly braces appear so frequent in Python syntax :

Parentheses

- Expression grouping

```
>>>
>>> x = (
...     1 + 2
...     + 3 + 4
...     + 5 + 6
... )
>>> x
21
```

- Function call

```
>>>
>>>print(
...     'foo',
...     'bar',
...     'baz'
... )
foo bar baz
```

- Method call

```
>>>
>>> 'abc'.center(
...     9,
...     '-'
... )
'---abc---'
```

• Tuple definition

```
>>>
>>> t = (
...     'a', 'b',
...     'c', 'd'
... )
```

Curly Braces

• Dictionary definition

```
>>>
>>> d = {
...     'a': 1,
...     'b': 2
... }
```

• Set definition

```
>>>
>>> x1 = {
...     'foo',
...     'bar',
...     'baz'
... }
```

Square Brackets

• List definition

```
>>>
>>> a = [
...     'foo', 'bar',
...     'baz', 'qux'
... ]
```

• Indexing

```
>>>
>>>a[
...  1
...  ]
'bar'
```

• Slicing

```
>>>
>>>a[
...  1:2
...  ]
['bar']
```

• Dictionary key reference

```
>>>
>>>d[
...  'b'
...  ]
2
```

- **Comments in Python**

Symbols used for writing comments include Hash (#) or Triple Double Quotation marks (""""). Hash is used in writing single line comments that do not span multiple lines. Triple Quotation Marks are used to write multiple line comments. Three triple quotation marks to start the comment and again three quotation marks to end the comment.

Example :

```
####### This example will not print Hello World ####### print('Hello World')  # This is a comment
```

Multiline Comments

```
""" This example will demonstrate
    multiple comments """

""" The following
    a variable contains the
    string 'Your age'
"""
a = 'Your age?'

""" The following statement prints
    what's inside the variable a
"""
print(a)
```

Keywords

The following list shows the Python keywords. These are reserved words and you cannot use the as constant or variable or any other identifier names. All the Python keywords contain lowerca letters only.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Identifiers and Variables

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

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, **name** and **Name** are two different identifiers in Python.

Python is not "statically typed". We do not need to declare variables before using them or declare their type. A variable is created the moment we first assign a value to it. A variable is a name given to a memory location. It is the basic unit of storage in a program.

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

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables :

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Valid Variable	Invalid Variable
myCountry = "India" my_Country = "India" _my_Country = "India" myCountry = "India" MYCOUNTRY = "India" myCountry2 = "India"	2myCountry = "India" my-Country = "India" my Country = "India"

Example variable creation :

```
# An integer assignment
age = 41
# A floating point
salary = 1234.5
# A string
name = "India"
print(age)
print(salary)
print(name)
```

Output :

```
41
1234.5
India
```

- **Declare the Variable**

Let's see how to declare the variable and print the variable.

```
# declaring the var
Number = 100

# display
print( Number)
```

- **Output :**

```
100
```

- **Variable redeclaration:**

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

```
# declaring the var
Number = 100

# display
print("Before declare: ", Number)

# re-declare the var
Number = 120.3

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

- **Output :**

```
Before declare:  100
After re-declare: 120.3
```

Assigning a single value to multiple variables:

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

For example :

```
a = b = c = 10

print(a)
```

```
print(b)
print(c)
```

Output :

```
10
10
10
```

Assigning different values to multiple variables:

Python allows adding different values in a single line with "," operators.

```
a, b, c = 1, 20.2, "India"
print(a)
print(b)
print(c)
```

Output :

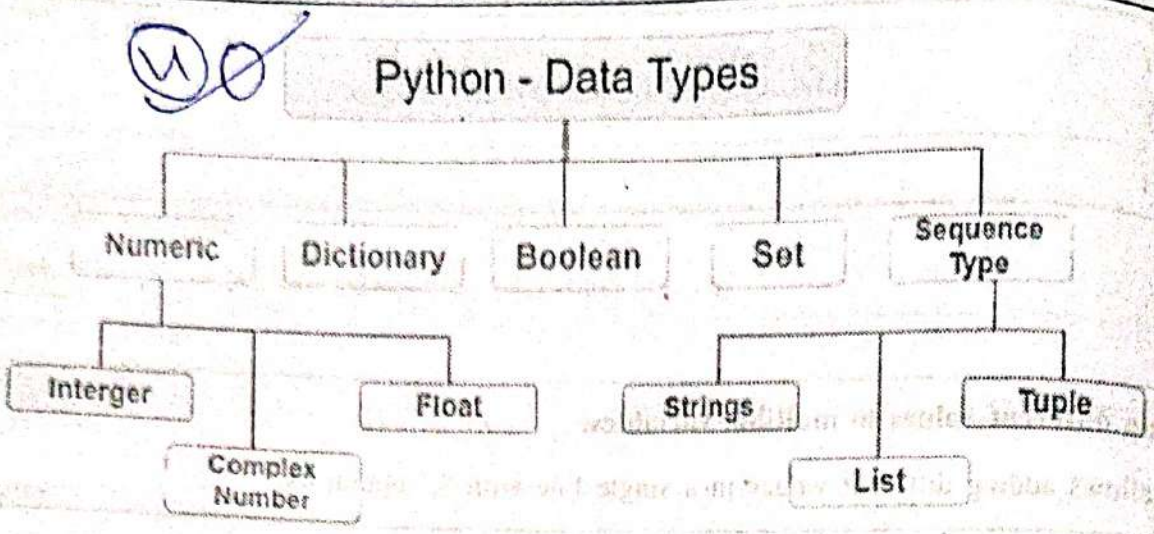
```
1
20.2
India
```

DATA TYPES

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

Following are the standard or built-in data type of Python :

- Numbers
- String
- List
- Tuple
- Dictionary



- **Numbers**

Python supports four distinct **numeric types** : integers, long, float and complex numbers. In addition, **Booleans** are a subtype of plain integers. Integers or int are positive or negative whole numbers with no **decimal point** . Long integers have unlimited precision and floats represent real numbers and are written with a decimal point dividing the integer and fractional parts. **Complex numbers** have a real and imaginary part, a + bc, where a is the real part and b is the imaginary part.

Example :

```
#integer example
x=9999
print("type of x is ", type(x))
#float example
y=3.141
print("The type of y is ", type(y))
#complex example
z=99+5j
print("The type of z is ", type(z))
```

Output :

```
Type of x is  < class 'int' >
The type of y is  < class 'float' >
The type of z is  < class 'complex' >
```

- **String**

A **String** is an array of characters. They are formed by a list of characters, which is really an "array of characters". They are less useful when storing information for the computer to use. An important characteristic of each string is its length, which is the number of characters in it. There are numerous **algorithms** for processing strings, including for searching, sorting, comparing and transforming.

In Python, string is a sequence of **Unicode character**. Unicode was introduced to include every character in all languages and bring uniformity in encoding. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.

```
str = "Hello World"   //double quotes
str1 = 'Hello World!'//using single quotes
```

Python strings are "immutable" which means they cannot be changed after they are created. Characters in a string can be accessed using the standard [ ] syntax and zero-based indexing.

Example :

```
str = "Hello World"
print (str[0])
print (str[6:11])
print (str + " !!")
print (len(str))
```

Output :

```
H
World
Hello World !!
11
```

- **List**

Python List is one of the most frequently used and very versatile datatype. Lists work similarly to strings: use the **len()** function and square brackets [ ] to access data, with the first element at index 0.

Example :

```
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print (weekdays[0])
print (weekdays[4])
```

Output :

```
Monday
Friday
```

- **Tuple**

A **tuple** is a container which holds a series of comma-separated values between parentheses. A tuple is similar to a list. Since, tuples are quite similar to **lists**, both of them are used in similar situations

as well. The only the difference is that list is enclosed between square bracket, tuple between parenthesis and List have **mutable objects** whereas Tuple have immutable objects.

Example :

```
my_Tuple_1 = (1,2,"Hello",3.14,"world")
print(my_Tuple_1)
print(my_Tuple_1[3])
my_Tuple_2 = (5,"six")
print(my_Tuple_1 + my_Tuple_2)
```

Output :

```
(1, 2, 'Hello', 3.14, 'world')
3.14
(1, 2, 'Hello', 3.14, 'world', 5, 'six')
```

- **Dictionary**

Pyhton **Dictionaries** allow you store and retrieve related information in a way that means something both to humans and computers. Dictionaries are non-ordered and contain **"keys"** and **"values"** . Each key is unique and the values can be just about anything, but usually they are string, int, or float, or a list of these things. Like lists dictionaries can easily be changed, can be shrunk and grown ad libitum at run time. Dictionaries don't support the sequence operation of the sequence data types like strings, tuples and lists. Dictionaries belong to the built-in mapping type.

Example :

```
# Creating a Dictionarywith Integer Keys
Dict = {1: I, 2: 'For', 3: India}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)
```

Output :

```
Dictionary with the use of Integer Keys :
{1: 'I', 2: 'For', 3: 'India'}
```

To tell what the value type is, you can use the built-in type() function. In the following examples we use the type() function to display the value type :

```
>>> x = 42
>>> print(type(x))
<class 'int'>
>>> x = 'hi'
```

```
>>> print(type(x))
<class 'str'>
>>> x = 3.14
>>> print(type(x))
<class 'float'>
>>> x = False
>>> print(type(x))
<class 'bool'>
>>>
```

Checking for Type Equality

While the type() function lets us check which type a variable contains. The is operator used for identity checking. Numeric values may compare as equal to each other using the equality test == yet not match on their type. Consider this example:

```
>>> x = 1
>>> y = 1.0
>>> x == y
True
>>> x is y
False
>>>
```

In the above example, x is assigned the *integer* value 1 and y is assigned the *float* value 1.0. When tested using the equality match ==, the result is True. Yet when tested using the object identity operator is, the result is False since *float* and *int* are different types.

Operators

517

Operators are used for carrying out operations on values and variables.

Python has 7 types of Operators as stated below:

- Arithmetic Operator
- Comparison operators
- Logical operators
- Bitwise operators
- Assignment Operator

- Identity operators
- Membership operators

1) Arithmetic Operators

Python programming language supports different kinds of arithmetic operators for both integer a floating point like addition, subtraction, multiplication, division and so on.

Operator Type	Definition
Addition (+)	Addition operator
Subtraction (-)	Subtraction operator
Multiplication (*)	Multiplication operator
Division (/)	Division operator
Modulus (%)	Reminder operator
Floor division (//)	Divides and returns the value of the remainder.
Exponentiation (**)	Raises the left operand to the power of right.

- **Example :**

```
x = 15
y = 10
print('x + y =', x+y)
```

Output: x + y = 25

```
print('x - y =', x-y)
```

Output: x – y = 5

```
print('x * y =', x*y)
```

Output: x * y = 150

```
print('x / y =', x/y)
```

Output: x / y = 1.5

```
print('x % y =', x%y)
```

Output: x % y = 5

```
print('x // y =', x//y)
```

Output: x // y = 1

```
print('x ** y =', x**y)
```

Output: x ** y = 576650390625

Python 3.6.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (I
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> x = 15
>>> y = 10
>>> print('x + y =', x+y)
x + y = 25
>>> print('x - y =', x-y)
x - y = 5
>>> print('x * y =', x*y)
x * y = 150
>>> print('x / y =', x/y)
x / y = 1.5
>>> print('x % y =', x%y)
x % y = 5
>>> print('x // y = ', x//y)
x // y =  1
>>> print('x ** y = ', x**y)
x ** y =  576650390625
```

2) Comparison Operators

Comparison operators are used for comparing values. It either returns True or False according to the condition.

Operators	Definition
Greater than (>)	True if left operand is greater than the right
Less than (<)	True if left operand is less than the right
Equal to (==)	True if both operands are equal
Not equal to (!=)	True if operands are not equal
Greater than or equal to (>=)	True if left operand is greater than or equal to the right
Less than or equal to (<=)	True if left operand is less than or equal to the right

Example :

x = 8

y = 15

('x>y is',x>y)

Output : x > y is False

print('x< y is', x<y)

Output : x < y is True

print('x == y is', x==y)

Output : x == y is False

print('x != y is', x!=y)

Output : x != y is True

print('x >= y is', x>=y)

Output : x >= y is False

print('x<= y is', x<=y)

Output : x <= y is True

Python 3.6.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> x = 8
>>> y = 15
>>> print('x > y is', x>y)
x > y is False
>>> print('x < y is', x<y)
x < y is True
>>> print('x == y is', x==y)
x == y is False
>>> print('x != y is', x!=y)
x != y is True
>>> print('x >= y is', x>=y)
x >= y is False
>>> print('x <= y is', x<=y)
x <= y is True
```

3) Logical Operators

Logical operators are used for performing AND, OR and NOT operations. It either returns True or False according to the condition.

Operators	Definitions
and	True if both the operands are true
or	True if either of the operands is true
not	True if operand is false

Example :

a = True

B = False

print('a and b is', a and b)

Output : a and b is False

print('a and b is', a or b)

Output : a or b is True

print('a and b is', a not b)

Output : not a is False

Python 3.6.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> a = True
>>> b = False
>>> print('a and b is', a and b)
a and b is False
>>> print('a or b is', a or b)
a or b is True
>>> print('not a is', not a)
not a is False
```

4) Bitwise Operators

Bitwise operators operate on bits and perform bit by bit operation.

Operators	Definitions
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
^	Bitwise XOR
>>	Bitwise right shift
<<	Bitwise left shift

5) Assignment Operator

An assignment operator is used to assign a value to a variable.

Operators	Definitions	Output
=	x = 15	x = 15
+=	x += 15	x = x + 15
-=	x -= 15	x = x - 15
*=	x *= 15	x = x * 15
/=	x /= 15	x = x / 15
%=	x %= 15	x = x % 15
//=	x //= 15	x = x // 15
**=	x **= 15	x = x ** 15
&=	x &= 15	x = x & 15
|=	x |= 15	x = x | 15
^=	x ^= 15	x = x ^ 15
>>=	x >>= 15	x = x >> 15
<<=	x <<= 15	x = x << 15

6) Identity Operators

Python offers 2 types of identity operators i.e is and is not.

Both are used to compare if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operators	Definitions
is	True if the operands are identical
is not	True if the operands are not identical

Example :

```
a1 = 3
b1 = 3
a2 = "Python"
b2 = "Python"

a3 = [4,5,6]
b3 = [4,5,6]

print(a1 is not b1)
```

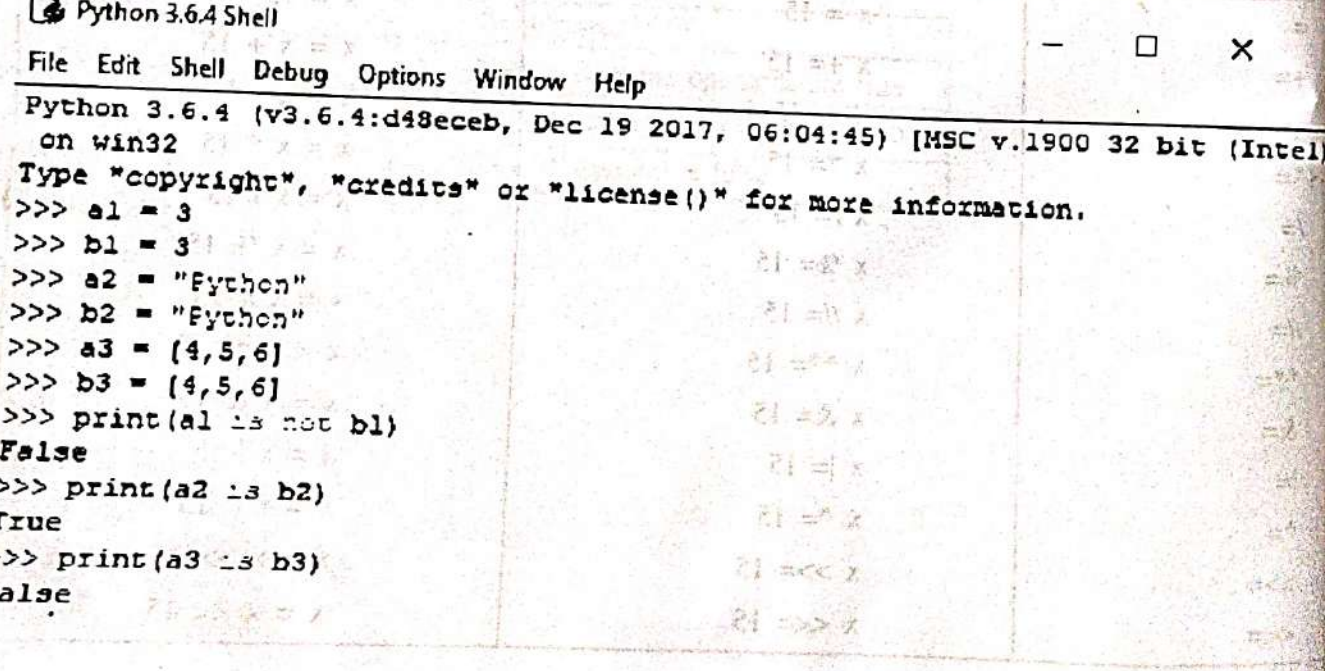
Output : False

```
print(a2 is b2)
```

Output : True

```
print(a3 is b3)
```

Output : False



Here a3 and b3 are listed, interpreter allocates memory separately and even though they are equal, it returns False.

Membership Operators

Python offers 2 types of membership operators i.e in and not in.

Both are used to test whether a value or variable is in a sequence.

Operators	Definitions
in	True if value is found in the sequence
not in	True if value is not found in the sequence

Example :

```
a = "Python operators"
b = {1:'x', 2:'y'}

print ("P" in a)
```

Output : True

```
print ("python" not in a)
```

Output : False

```
print (1 in b)
```

Output : True

```
print ("y" in b)
```

Output : False

```
Python 2.7.14 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:19:30) [MSC v.1500 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a = "python operators"
>>> b = {1:'x', 2:'y'}
>>> print("p" in a)
True
>>> print("python" not in a)
False
>>> print(1 in b)
True
>>> print('y' in b)
False
```

1 is key and 'x' is the value in dictionary b. Hence, 'y' in b returns False.

Operators Precedence

The following table lists all operators from highest precedence to lowest.

Sr. No.	Operator & Description
1	**Exponentiation (raise to the power)
2	~ + -Complement, unary plus and minus (method names for the last two are +@ and -@)
3	* / % //Multiply, divide, modulo and floor division
4	+ -Addition and subtraction
5	>><<Right and left bitwise shift
6	&Bitwise 'AND'
7	^ |Bitwise exclusive 'OR' and regular 'OR'
8	<= <>>=Comparison operators
9	<> == !=Equality operators
10	= %= /= //= -= += *= **=Assignment operators
11	is is not Identity operators
12	in not in Membership operators
13	not or and Logical operators

2.4 Type Conversion

Type Conversion

The process of converting the value of one data type (integer, string, float, etc.) to another da type is called type conversion.

1. Type Conversion is the conversion of object from one data type to another data type.
2. Implicit Type Conversion is automatically performed by the Python interpreter.
3. Python avoids the loss of data in Implicit Type Conversion.
4. Explicit Type Conversion is also called Type Casting, the data types of objects are converte using predefined functions by the user.
5. In Type Casting, loss of data may occur as we enforce the object to a specific data typ

Python has two types of type conversion.

1. Implicit Type Conversion
2. Explicit Type Conversion

Implicit Type Conversion

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Example 1 Converting integer to float

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

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

Output :

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

As we can see the type od 'x' got automatically changed to the "float" type from the "integer" type. this is a simple case of Implicit type conversion in python.

Explicit Type Conversion

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.

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

Syntax :

```
<required_datatype>(expression)
```

Typecasting can be done by assigning the required data type function to the expression.

In Explicit Type Conversion in Python, the data type is manually changed by the user as per their requirement. Various form of explicit type conversion are explained below:

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

Example: Explicit type conversion

```
# Python code to demonstrate Type conversion
# using int(), float(), str()
```

Integers :

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Floats :

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
```