1. Introduction

Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics, used for general-purpose programming. It was created by **Guido van Rossum**, and first released on February 20, 1991.

It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently.

While you may know the python as a large snake, the name of the Python programming language comes from an old BBC television comedy sketch series called **Monty Python's Flying Circus**.

One of the amazing features of Python is the fact that it is actually one person's work. Usually, new programming languages are developed and published by large companies employing lots of professionals, and due to copyright rules, it is very hard to name any of the people involved in the project. Python is an exception.

Of course, van Rossum did not develop and evolve all the Python components himself. The speed with which Python has spread around the world is a result of the continuous work of thousands (very often anonymous) programmers, testers, users (many of them aren't IT specialists) and enthusiasts, but it must be said that the very first idea (the seed from which Python sprouted) came to one head.

2. Why Python?

Python has become one of the most popular programming languages in the world in recent years. It's used in everything from machine learning to building websites and software testing. It can be used by developers and non-developers alike. Python is popular for a number of reasons as given below:

- Python has simple, conventional syntax. Python statements are very close to those of pseudocode algorithms, and Python expressions use the conventional notation found in algebra. Thus, students can spend less time learning the syntax of a programming language and more time learning to solve interesting problems.
- Python has safe semantics. Any expression or statement whose meaning violates the definition of the language produces an error message.
- Python scales well. It is very easy for beginners to write simple programs in Python. Python also includes all of the advanced features of a modern programming language, such as support for data structures and object-oriented software development, for use when they become necessary.
- Python is highly interactive. Expressions and statements can be entered at an interpreter's prompts to allow the programmer to try out experimental code and receive immediate feedback. Longer code segments can then be composed and saved in script files to be loaded and run as modules or standalone applications.
- Python is general purpose. In today's context, this means that the language includes resources for contemporary applications, including media computing and networks.
- Python is free and is in widespread use in industry. Students can download Python to run on a variety of devices. There is a large Python user community, and expertise.

- Python is versatile. Python can be used for many different tasks, such as, web development, machine learning, data analysis etc.
- Python has a number of libraries that enable coders to write programs for data analysis and machine learning more quickly and efficiently.

3. Installing and running Python

Download installer from http://www.python.org/download/ and install it. To launch an interactive session with Python's shell from a terminal command prompt, open a terminal window, and enter python at the prompt. To end the session on Unix machines (including macOS), press the Control+D key combination at the session prompt. To end a session on Windows, press Control+Z, and then press Enter.

To run a script, enter python, followed by a space, followed by the name of the script's file (including the .py extension), followed by any command-line arguments that the script expects.

You can also launch an interactive session with a Python shell by launching IDLE (Integrated Development and Learning Environment). IDLE is a default editor that comes with python. There are many advantages to using an IDLE shell rather than a terminal-based shell, such as color-coded program elements, menu options for editing code and consulting documentation, and the ability to repeat commands. IDLE also helps you manage program development with multiple editor windows. You can run code from these windows and easily move code among them.

The are several other free and commercial IDEs (Integrated Development Environments) with capabilities that extend those of IDLE. Some popular python IDEs are PyCharm, Spyder, Visual Studio Code, Atom, Jupyter etc.

4. Installing Third Party Libraries

In order to install third-party libraries, we need to use the Python Package Installer, known as pip. Chances are that it is already available to you within your virtual environment, but if not, you can learn all about it on its documentation page (https://pip.pypa.io). For example, on Windows you would enter **pip install packagename**, where packagename is the name of the package.

If you already have the package installed but would like to upgrade it to the latest version available on PyPI,(Python Package Index) run **pip install –U packagename**. To uninstall a package, you simply use **pip uninstall packagename**.

5. Working with Virtual Environments

When working with Python, it is very common to use virtual environments. Virtual environments are isolated Python environments, each of which is a folder that contains all the necessary executables to use the packages that a Python project would need.

A virtual environment is a tool that helps to keep dependencies required by different projects separate by creating isolated python virtual environments for them. This is one of the most important tools that most Python developers use.

It is generally good to have one new virtual environment for every Python-based project you work on. So, the dependencies of every project are isolated from the system and each other.

One most commonly used tool to create virtual environment is **virtualenv**. The virtualenv creates a folder that contains all the necessary executables to use the packages that a Python project would need.

- Installing virtual environment
  $ pip install virtualenv
- Creating virtual environment
  $ virtualenv my_name
- If you want to specify the Python interpreter of your choice, for example, Python 3, it can be done using the following command.
  $ virtualenv -p /usr/bin/python3 virtualenv_name
- Now after creating virtual environment, you need to activate it. Remember to activate the relevant virtual environment every time you work on the project. To activate virtual environment using windows command prompt change directory to your virtual env and activate.
  $ cd <envname>
  $ Scripts\activate
- Once the virtual environment is activated, the name of your virtual environment will appear on the left side of the terminal. This will let you know that the virtual environment is currently active. Now you can install dependencies related to the project in this virtual environment.
- Once you are done with the work, you can deactivate the virtual environment by the following command.
  (virtualenv_name)$ deactivate

6. Writing Comments

A comment is a piece of program text that the computer ignores but that provides useful documentation to programmers. Comments are used to make the code more readable. Python provides two ways of writing comments: single line and multiline comments.

- Single line comments start with a #. For example,
  # This is my first program
  print("Hello world") # This is output statement
- Since Python ignores string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it. For example,
  """ This is my
  first program """

7. Indentation

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

```
a = int(input("a = "))
b = int(input("b = "))
if a > b:
   print("a is larger")
elif b > a:
```

```
    print("b is larger")
else:
    print("both are equal")
```

Remember: You have to use the same number of spaces in the same block of code.

8. Tokens

Python breaks each logical line into a sequence of elementary lexical components known as tokens. Each token corresponds to a substring of the logical line. The normal token types are **identifiers**, **keywords**, **operators**, **delimiters**, and **literals**. You may freely use whitespace between tokens to separate them.

9. Identifiers

An *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). Case is significant in Python: lowercase and uppercase letters are distinct.
Normal Python style is to start class names with an uppercase letter (pascal case) and all other identifiers with a lowercase letter.

10. Keywords

Keywords are the tokens that Python reserves for special syntactic uses. We cannot use keywords as regular identifiers. The table below show Python keywords.

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

11. Literals

Literals are the fixed values or data items used in a source code. Python supports different types of literals such as:

- **String Literals:** The text written in single, double, or triple quotes represents the string literals in Python. Triple quotes are used to write multi-line strings. For example,
  'Kathmandu Nepal'
  "Kathmandu Nepal"
  """Kathmandu
  Nepal"""
- **Character Literals:** Character literal is also a string literal type in which the character is enclosed in single or double-quotes.
- **Numeric Literals:** These are the literals written in form of numbers. Python supports the following numerical literals:
  - **Integer Literal:** It includes both positive and negative numbers along with 0. It doesn't include fractional parts. It can also include binary, decimal, octal, hexadecimal literal.

  - **Float Literal:** It includes both positive and negative real numbers. It also includes fractional parts.
  - **Complex Literal:** It includes a+bi numeral, here a represents the real part and b represents the complex part.
- **Boolean Literals:** Boolean literals have only two values in Python. These are True and False.
- **Special Literals:** Python has a special literal 'None'. It is used to denote nothing, no values, or the absence of value.

12. Variables

Python Variables are containers that store values. 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 Python variable is a name given to a memory location. It is the basic unit of storage in a program.
Python allows you to assign values to multiple variables in one line. For example,

x, y, z = "Orange", "Banana", "Cherry"

You can assign the same value to multiple variables in one line. For example,

x = y = z = "Orange"

Python allows you to extract the values into variables from collection of values. This is called unpacking. For example,

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
```

The **print()** function is often used to output variables. For example,

```
x = "Python is awesome"
print(x)
```

With print() function, you output multiple variables, separated by a commas. For example,

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

You can also use + operator to output multiple variables. For numbers, the + character works as a mathematical operator. For example,

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

12.1. Global Variables

Variables declared outside a function is called global variable. Such variable can be used both inside and outside the function. For example,

```
x = "awesome"
def myfunc():
```

```
  print("Python is " + x)
myfunc()
print("Python is " + x)
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value. For example,

```
x = "awesome"
def myfunc():
   x = "fantastic"
   print("Python is " + x)
myfunc()
print("Python is " + x)
```

To create a global variable inside a function, you can use the **global** keyword. For example,

```
def myfunc():
   global x
   x = "fantastic"
myfunc()
print("Python is " + x)
```

Also, use the **global** keyword if you want to change a global variable inside a function. For example,

```
x = "awesome"
def myfunc():
   global x
   x = "fantastic"
myfunc()
print("Python is " + x)
```

13. Constants

Sometimes, you may want to store values in variables. But you don't want to change these values throughout the execution of the program. To do it in other programming languages, you can use constants. The constants are like variables but their values don't change during the program execution.

The bad news is that Python doesn't support constants. To work around this, you use all capital letters to name a variable to indicate that the variable should be treated as a constant. For example:

PI = 3.1415

When encountering variables like these, you should not change their values. These variables are constant by convention, not by rules.

Note: One way to implement constants is to declare and assign constants in a module (.py file) and use these constants in another python file.

14. Operators

Operators are special symbols in Python that perform operations on variables or values. The value that the operator operates on is called the operand. Python divides the operators in the following groups.

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

14.1. Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations like addition, subtraction, multiplication, etc.

Operator	Meaning	Example
+	Add two operands or unary plus	x + y
-	Subtract right operand from the left or unary minus	x – y
*	Multiply two operands	x * y
/	Divide left operand by the right one (results into float)	x / y
%	Modulus – remainder of the division of left operand by the right	x % y
//	Floor division – division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x ** y

14.2. Comparison Operators

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

Operator	Meaning	Example
>	Greater than – True if left operand is greater than the right	x > y
<	Less than – True if left operand is less than the right	x < y
==	Equal to – True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to – True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to – True if left operand is less than or equal to the right	x <= y

14.3. Logical Operators

Logical operators are used to combine conditional statements. They perform logical AND, logical OR, and logical NOT operations.

Operator	Meaning	Example
And	Logical AND – True if both the operands are true	x and y
Or	Logical OR – True if either of the operands is true	x or y
Not	Logical NOT – True if operand is false	not x

14.4. Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location.

Operator	Meaning	Example

is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

For example,

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is z) #True
print(x is y) #False
print(x == y) #True
```

14.5. Membership Operators

Membership operators are used to test a value or variable is found in a sequence (string, list, tuple, set and dictionary). In a dictionary, we can only test for presence of key, not the value.

Operator	Meaning	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

For example:

```
x = ["apple", "banana"]
print("banana" in x) #True
```

14.6. Bitwise Operators

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

Operator	Meaning	Example
&	Bitwise AND: Sets each bit to 1 if both bits are 1	x & y
|	Bitwise OR: Sets each bit to 1 if one of two bits is 1	x | y
~	Bitwise NOT: Inverts all the bits	~x
^	Bitwise XOR: Sets each bit to 1 if only one of two bits is 1	x ^ y
<<	Bitwise left shift: Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Bitwise right shift: Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

For example,

```
x = 10
y = 4
print(x & y) #0
print(x | y) #14
print(~x) #-11
print(x ^ y) #14
print(x << 2) #40
print(x >> 2) #2
```

14.7. Assignment Operators

Assignment operators are used to assign values to variables. Python has many assignment operators as given below.

Operator	Meaning	Example
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add and Assign: Add right side operand with left side operand and then assign to left operand	x += 2 (same as x = x + 2)

Like +=, we have many other assignment operators: -=, *=, /=, %=, //=, **=, &=, |=, ^=, >>=, and <<=

14.8. Precedence and Associativity

Please see the following precedence and associativity table for reference. This table lists all operators from the highest precedence to the lowest precedence.

Operator	Description	Associativity
( )	Parentheses	left-to-right
**	Exponent	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
is, is not in, not in	Identity Membership operators	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
Not	Logical NOT	right-to-left
And	Logical AND	left-to-right
Or	Logical OR	left-to-right
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= |=	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	