MODULE 1

Python was developed by **Guido van Rossum** at the National Research Institute for and Computer Science in Netherlands during 1985-1990. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell and other scripting languages.

1.1 FEATURES OF PYTHON

a) Simple and easy-to-learn- Python is a simple language with few keywords, simple structure and its syntax is also clearly defined. This makes Python a beginner's language.

b) Interpreted and Interactive - Python is processed at runtime by the interpreter. We need not compile the program before executing it. The Python prompt interact with the interpreter to interpret the programs that we have written. Python has an option namely interactive mode which allows interactive testing and debugging of code

c) Object-Oriented - Python supports Object Oriented Programming (OOP) concepts that encapsulate code within objects. All concepts in OOPs like data hiding, operator overloading, inheritance etc. can be well written in Python. It supports functional as well as structured programming.

d) Portable - Python can run on a wide variety of hardware and software platforme and has the same interface on all platforms. All variants of Windows, Unix, Linu and Macintosh are to name a few.

e) Scalable - Python provides a better structure and support for large programs than shell programming. It can be used as a scripting language or can be compiled to bytecode (intermediate code that is platform independent) for building large applications.

f) Extendable- You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient. It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

g) Dynamic- Python provides very high-level dynamic data types and supports dynamic type checking. It also supports automatic garbage collection.

h) GUI Programming and Databases- Python supports GUI applications that can be created and ported to many libraries and windows systems, such as Windows,
Microsoft Foundation Classes (MFC), Macintosh, and the X Window system of Unix.

Python also provides interfaces to all major commercial databases.

i) Broad Library - Python's library is portable and cross platform compatible on UNIX, Linux, Windows and Macintosh. This helps in the support and development of a wide range of applications from simple text processing to browsers and complex games.

1.2 HOW TO RUN PYTHON

There are three different ways to start Python.

a) Using Interactive Interpreter

You can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window. Get into the command line of Python. For Unix/Linux, you can get into interactive mode by typing $python or python%.

For Windows/Dos it is C:>python.

b) Script from the Command Line

This method invokes the interpreter with a script parameter which begins the execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active. A Python script can be executed at command line by invoking the interpreter on your application, as follows.

For Unix/Linux it is $python script.py or python% script.py. For Windows/ Dos it is C:> python script.py.

c) Integrated Development Environment

You can run Python from a Graphical User Interface (GUI) environment as well, if you have a GUI application on your system that supports Python. IDLE is the Integrated Development Environment (IDE) for UNIX and PythonWin is the first Windows Interface for Python.

1.3 IDENTIFIERS

A Python identifier is a name used to identify a variable, function, class, module or any other object. Python is case sensitive and hence uppercase and lowercase letters are considered. The following are the rules for naming an identifier in Python.

a) Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z)
or digits (0 to 9) or an underscore (_). For example Total and total is different.

b) Reserved keywords cannot be used as an identifier.

c) Identifiers cannot begin with a digit. For example 2more, 3times etc. are invalid identifiers.

d) Special symbols like @, ! ,#, $, % etc. cannot be used in an identifier. For example sum@, #total are invalid identifiers.
e) Identifier can be of any length.

1.4 RESERVED KEYWORDS

These are keywords reserved by the programming language and prevent the user or the programmer from using it as an identifier in a program. There **are 33 keywords** in Python 3.3.This number may vary with different versions. To retrieve the keywords in Python the following code can be given at the prompt. All keywords except True, False and None are in owercase. The following list in Table 1.1 shows the Python keywords.

```
>>> import keyword
>>> print (keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', '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]
```

Table 1.1 Python Keywords

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

1.5 VARIABLES

Variables are reserved memory locations to store values. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables. The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

Eg:
```
a=10
b=4.5
```

1.6 COMMENTS IN PYTHON

Comments are very important while writing a program. It describes what the source code has done. Comments are for programmers for better understanding of a program. In Python, we use the hash (#) symbol to start writing a comment. A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end the physical line are part of the comment. It extends up to the newline character. Python interpreter ignores comment.

Example Program

```
#This is demo of comment
#Display Hello
print ("Hello")
```

This produces the following result Hello

For multiline comments one way is to use (#) symbol at the beginning of each line. The following example shows a multiline comment. Another way of doing this is to use triple quotes, either" or """.

Example 1

```
#This is a very long sentence
#and it ends after
#Three lines
```

Example 2

```
"""This is a very long sentence
and it ends after
Three lines"""
```

1.7 INDENTATION IN PYTHON

Most of the programming languages like C, C++ and Java use braces { } to define a block of code. Python uses indentation. A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation can be decided by the programmer, but it must be consistent throughout the block. Generally four whitespaces are used for indentation and is preferred over tabspace. For example

```
if (a<b):
        print(a)
else:
        print(b)
```

1.8 MULTIPLE STATEMENT GROUP (SUITE)

A group of individual statements, which make a single code block is called a **suite** in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite. Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite. For example

```
if expression:
        suite
elif expression:
        suite
else:
        suite
```

1.9 QUOTES IN PYTHON

Python accepts single ('), double (") and triple ('''or """) quotes to denote string literals. The same type of quote should be used to start and end the string. The triple quotes are used to span the string across multiple lines. For example, all the following are legal.

```
word = 'single word'
sentence=  "This is a short sentence."
Paragraph=  """This is a long paragraph. It consists of
                several lines and sentences. """
```

1.10 INPUT, OUTPUT AND IMPORT FUNCTIONS

1.10.1 Displaying the Output

The function used to print output on a screen is the **print** statement where you can pass zero or more expressions separated by commas. The **print** function converts the expressions you pass into a string and writes the result to standard output.

Example 1

```
a=10
print ("Value of a is", a)
o/p:            Value of a is 10
```

1.10.2 Reading the Input

Python provides two built-in functions to read a line of text from standard input which by default comes from the keyboard. These functions are **raw_input()** and **input()**. The raw_input() function is not supported by Python 3.

The **raw_input( )** function reads one line from standard input and returns it as a string (removing the trailing newline). This prompts you to enter any string and it would display same string on the screen.

Example

```
str =  raw_input("Enter your name:");
print("Your name is ", str)
```

Output

```
Enter your name: Collin Mark
Your name is: Collin Mark
```

The **input( )** function is equivalent to raw_input(), except that it assumes the input is a valid Python expression and returns the evaluated result to you.

Example

```
n = input("Enter a number:");
print("The number is: ", n)
```

Output

```
Enter a number: 5
The number is 5
```

1.11 Import function

When the program grows bigger or when there are segments code that is frequently used, it can be stored in different modules. A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension .py. Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the **import** keyword to do this. For example, we can import the math module by typing in import math.

```
import math
print( math.pi)
O/p:    3.141592653589793
```

1.12 OPERATORS

Operators are the constructs which can manipulate the value of operands. Consider the expression a = b+c. Here a, b and c are called the operands and +, = are called the operators. There are different types of operators. The following are the types of operators supported by Python.

- Arithmetic Operators
- Comparison (Relational Operators)
- Assignment operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

1.12.1 Arithmetic Operators

Arithmetic operators are used for performing basic arithmetic operations. The following are the arithmetic operators supported by Python.

Operator	Name	Description	
+	Addition	Adds values on either side of the operator.	
-	Subtraction	Subtracts right hand operand from left hand operand.	

*	Multiplication	Multiplies values on either side of the operator.
/	Division	Divides left hand operand by right hand operand.
%	Modulus	Divides left hand operand by right hand operand and returns reminder.
**	Exponent	Performs exponential(power) calculation on operators.
//	Floor division	The division on operands where the result is the quotient in which the digits after the decimal point are removed.

```
a,b=10,3
print("Exponent", b**2)
print("Floor division",a//b)
```

O/P:
```
Exponent  9
Floor Division  3
```

1.12.2 Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called relational operators. Python supports the following relational operators.

Operators	Description
==	If the values of two operands are equal, then the condition becomes true.
!=	If the values of two operands are not equal, then condition becomes true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.

1.12.3 Assignment Operators

Python provides various assignment operators. Various short hand operators for addition, subtraction, multiplication, division, modulus, exponent and floor division are also supported by Python.

Operator	Description
=	Assigns values from right side operands to left side operand.
+=	It adds right operand to the left operand and assign the result to left operand.
-=	It subtracts right operand from the left operand and assign the result to left operand.
*=	It multiplies right operand with the left operand and assign the result to left operand
/=	It divides left operand with the right operand and assign the result to left operand.
%=	It takes modulus using two operands and assign the result to left operand.
**=	Performs exponential (power) calculation on operators and assign value to the left operand
//=	It Performs floor division on operators and assign value to the left operand

Eg 1:
```
a,b=10,5
a+=b
print(a)
```
O/P: 15

Eg 2:
```
a,b=10,5
a+=b
print(a)
```
O/P: 50

1.12.4 Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. The following are bitwise operators supported by Python:

Operator	Description
&	**Binary AND** Operator copies a bit to the result if it exists in both operands.
|	**Binary OR** Operator copies a bit if it exists in either operand.
^	**Binary XOR** Operator copies the bit if it is set in one operand but not both.
~	**Binary Ones Complement** Operator is unary and has the effect of 'flipping' bits.
<<	**Binary Left Shift Operator**. The left operands value is moved left by the number of bits specified by the right operand.
>>	**Binary Right Shift Operator**. The left operands value is moved right by the number of bits specified by the right operand.

1.12.5 Logical Operators

Operator	Description
and	**Logical AND** operator. If both the operands are true then then condition becomes true.
or	**Logical OR** Operator. If any of the operands are true then condition becomes true.
not	**Logical NOT** Operator. Used to reverses the logical state of its operand.

1.12.6 Membership Operators

Python’s membership operators test for membership in a sequence, such as strings, lists or tuples. There are two membership operators as shown in below:

Operator	Description
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.

not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

Example Program

```
S= ‘morning’
print('m' in S)
print ("f" in S)
print('f' not in S)
```

Output

```
True
False
True
```

1.12.7 Identity Operators

Identity operators compare the memory locations of two objects. There are two identity operators;

Operator	**Description**
is	Evaluates to true if the variable on either side of the operator point to the same object and false otherwise.
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.

Example Program

```
a, b, c = 10,10,5
print (a is b)
print (a is c)
print (a is not b)
```

Output

```
True
False
False
```

1.12.8 Operator Precedence

Operator associativity determines the order of evaluation, when they are of the same precedence, and are not grouped by paranthesis. An operator may be left-associative or right-associative. In left-associative, the operator falling on the left side will be evaluated first, while in right-associative, falling on the right will be evaluated first. In Python, = and '**' right-associative while all other operators are left-associative.

Operator	Description
**	Exponentiation (raise to the power)
~ ,+, -	Complement, unary plus and minus
*, /, %, //	Multiply, divide, modulo and floor division
+, -	Addition and subtraction
>>, <<	Right and left bitwise shift
&	Bitwise 'AND'
^, |	Bitwise exclusive `OR' and regular `OR'
<=, < >, >=	Comparison operators
<>, ==, !=	Equality operators
=, %=, /= ,//=, -=, +=, *=, **=	Assignment operators
is ,is not	Identity operators
in, not in	Membership operators
not, or, and	Logical operators

REVIEW QUESTIONS

1. Explain the Features of Python

2. Operators in Python.
3. Is Python a case sensitive language?
4. How can you run Python?
5. Give the rules for naming an identifier in Python.
6. How can you give comments in Python?
7. Write notes on input() and raw_input() function in Python.
8. List out the operator precedence in Python.
9. What is the purpose of // operator?
10. Briefly explain input and output functions used in Python.