

PYTHON PROGRAMMING LANGUAGE

Unit-1

Python is an elegant and robust programming language that delivers both the power and general applicability of traditional compiled languages with the ease of use (and then some) of simpler scripting and interpreted languages. It allows you to get the job done, and then read what you wrote later. You will be amazed at how quickly you will pick up the language as well as what kind of things you can do with Python, not to mention the things that have already been done. Your imagination will be the only limit.

- Work on Python began in late 1989 by **Guido van Rossum**, then at CWI (Centrum voor Wiskunde en Informatica, the National Research Institute for Mathematics and Computer Science) in the Netherlands.
- It was eventually released for public distribution in early 1991. How did it all begin? Like C, C++, Lisp, Java, and Perl, Python came from a research background where the programmer was having a hard time getting the job done with the existing tools at hand, and envisioned and developed a better way.
- At the time, van Rossum was a researcher with considerable language design experience with the interpreted language ABC, also developed at CWI, but he was unsatisfied with its ability to be developed into something more. Having used and partially developed a higher-level language like ABC, falling back to C was not an attractive possibility.
- Some of the tools he envisioned were for performing general system administration tasks, so he also wanted access to the power of system calls that were available through the Amoeba distributed operating system. Although van Rossum gave some thought to an Amoeba-specific language, a generalized language made more sense, and late in 1989, the seeds of Python were sown.

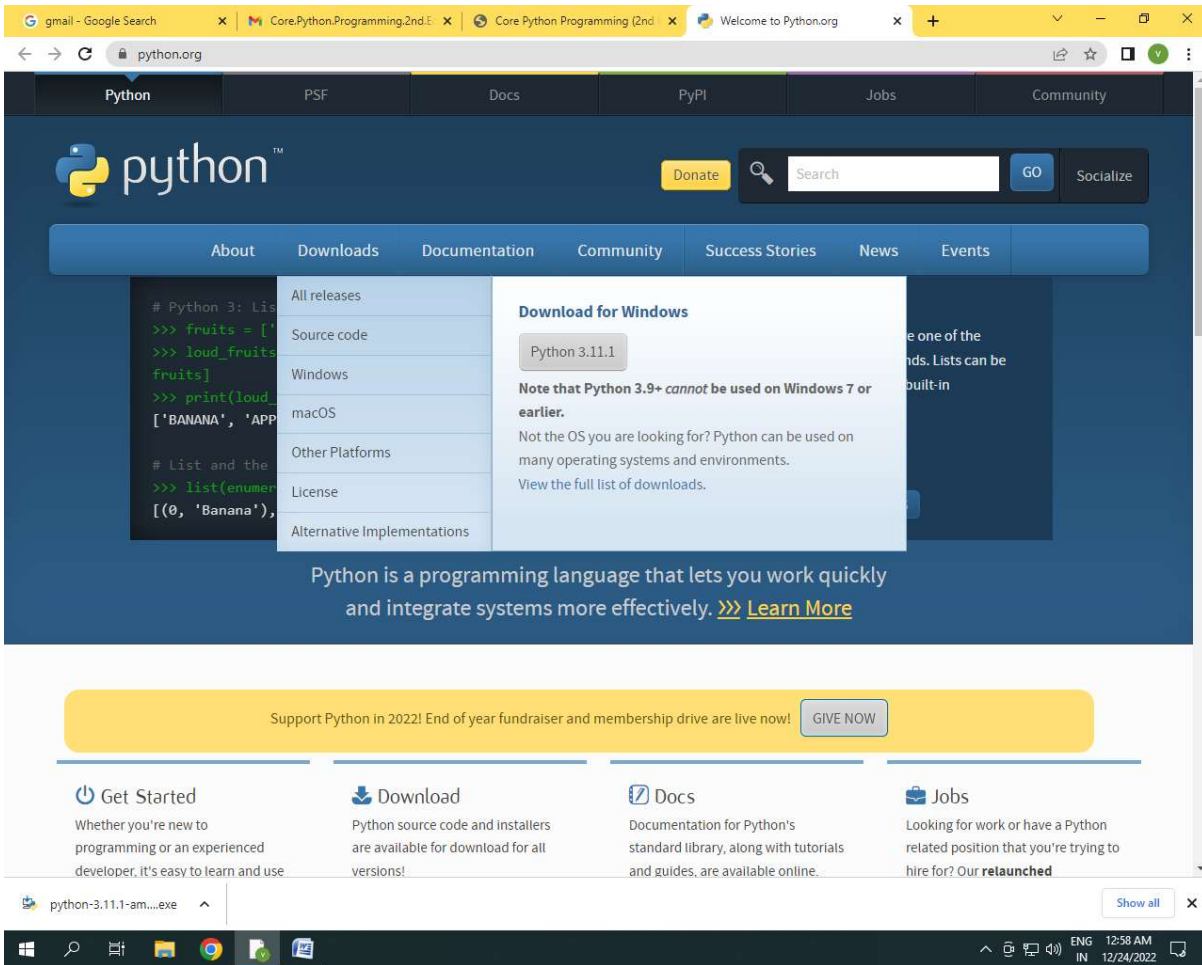
Features

- **High Level**
- Object Oriented
- Scalable:

Python is often compared to batch or Unix shell scripting languages. Simple shell scripts handle simple tasks. They may grow (indefinitely) in length, but not truly in depth. There is little code-reusability and you are confined to small projects with shell scripts.

- Extensible
- Portable
- Easy to Learn
- Easy to Read
- Easy to Maintain
- Robust
- Effective as a Rapid Prototyping Tool
- A Memory Manager
- Interpreted and (Byte-) Compiled

Downloading and Installing Python



On Windows, the default installation area is C:\Python2x. TRy to avoid installing Python in C:\Program Files.

Running Python

There are three different ways to start Python.

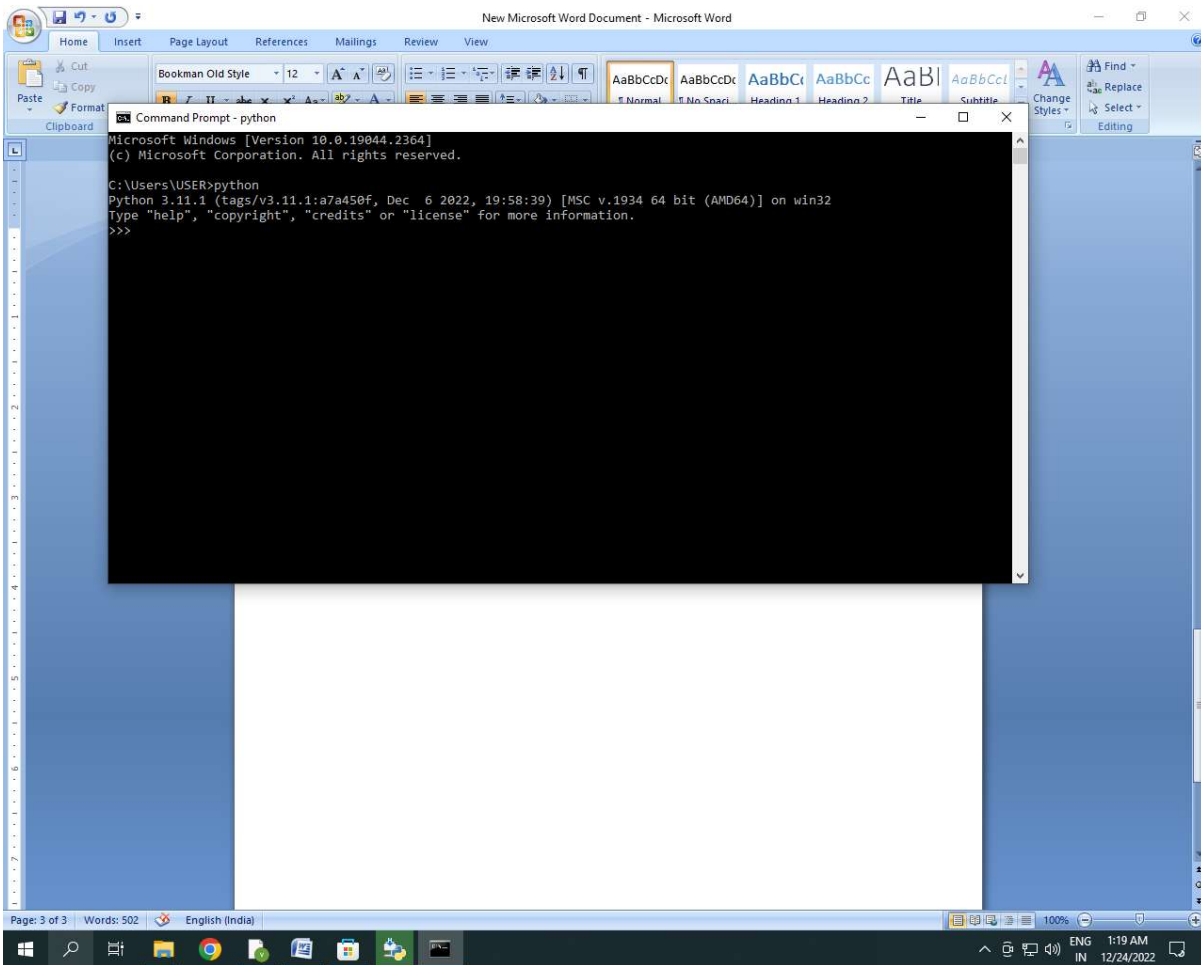
- The simplest way is by **starting the interpreter interactively**, entering one line of Python at a time for execution.
- Another way to start Python is by **running a script written in Python**. This is accomplished by invoking the interpreter on your script application.
- Finally, you can run from a graphical user interface (GUI) from within an **integrated development environment (IDE)**. IDEs typically feature additional tools such as an integrated debugger, text editor, and support for a wide range of source code control tools such as CVS.

Starting Python in a DOS/command window

- To add Python to your search path, you need to **edit the C:\autoexec.bat file and add the full path to where your interpreter is installed**.
- It is usually either **C:\Python** or **C:\Program Files \Python** (or its short DOS name equivalent **C:\Progra~1\Python**).

EX:

C:\Users\USER>python

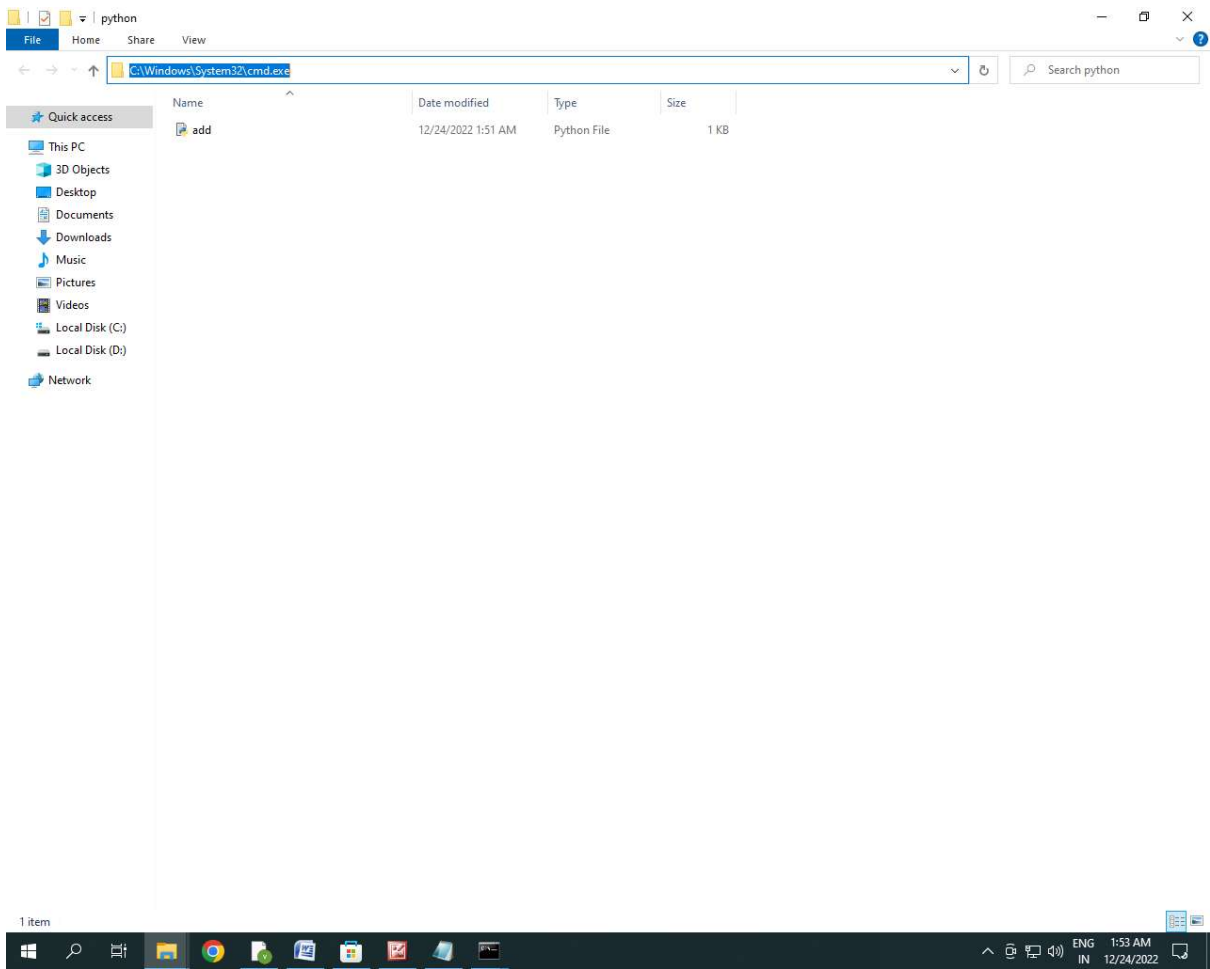


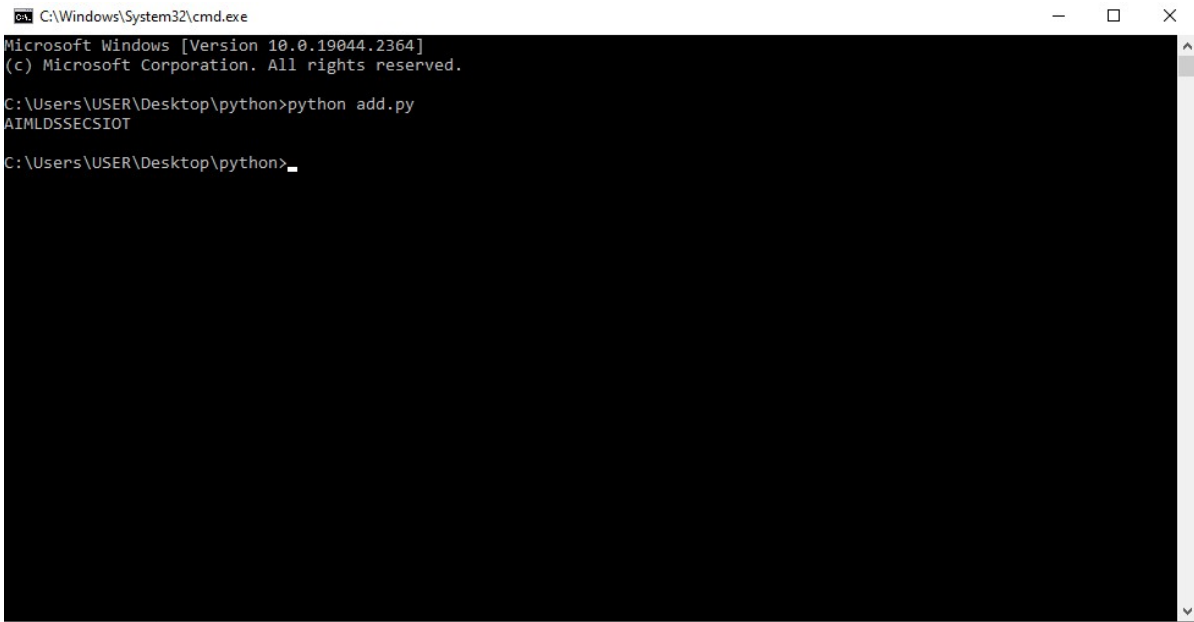
Command-Line Options When starting Python from the command-line, additional options may be provided to the interpreter. Here are some of the options to choose from:

- d Provide debug output
- O Generate optimized bytecode (resulting in .pyo files)
- S Do not run importsite to look for Python paths on startup
- v Verbose output (detailed trace on import statements)
- m mod run (library) module as a script
- Q opt division options (see documentation)
- c cmd Run Python script sent in as cmd string
- file Run Python script from given file (see later)

As a Script from the Command Line

- Open Notepad
- Type script/program (print("Hello AIML DS SECSIOT"))
- Save notepad in one Folder (Python) with **.py**
- Then click on path
- Type cmd
- Then one command will open in that type **python filename.py**
- Click on enter



A screenshot of a Windows Command Prompt window. The title bar shows 'C:\Windows\System32\cmd.exe'. The window content displays the following text: 'Microsoft Windows [Version 10.0.19044.2364] (c) Microsoft Corporation. All rights reserved. C:\Users\USER\Desktop\python>python add.py AIMLDSSECSIOT C:\Users\USER\Desktop\python>'. The prompt is at the end of the last line, waiting for input.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2364]
(c) Microsoft Corporation. All rights reserved.

C:\Users\USER\Desktop\python>python add.py
AIMLDSSECSIOT

C:\Users\USER\Desktop\python>
```

In an Integrated Development Environment

You can run Python from a graphical user interface (GUI) environment as well. All you need is a GUI application on your system that supports Python. If you have found one, chances are that it is also an IDE (integrated development environment). IDEs are more than just graphical interfaces. They typically have source code editors and trace and debugging facilities.

Python Documentation

Introduction

In all interactive examples, you will see the

- Python primary (`>>>`) The primary prompt is a way for the interpreter to let you know **that it is expecting the next Python statement**,
- secondary (`...`) prompts.

while the secondary prompt indicates that the **interpreter is waiting for additional input to complete the current statement**.

```
>>> print 'Hello World!'
Hello World!
```

- Expressions, on the other hand, do not use keywords. They can be simple equations that you use with mathematical operators, or can be functions which are called with parentheses.
- They may or may not take input, and they may or may not return a (meaningful) value. (Functions that do not explicitly return a value by the programmer automatically return `None`, Python's equivalent to `NULL`.)
- An example of a function that takes input and has a return value is the `abs()` function, which takes a number and returns its absolute value is:

```
>>> abs(4)
4
>>> abs(-4)
4
```

Program Output, the print Statement:

- In some languages, such as C, displaying to the screen is accomplished with a function, e.g., **printf()**, while with Python and most interpreted and scripting languages, it is a statement.
- Many shell script languages use an **echo** command for program output.

print Statement :

```
>>> myString = 'AIMLDS SECSIOT '
>>> print myString
SECSIOT
>>> myString
'AIMLDS SECSIOT '
```

The underscore (`_`) also has special meaning in the interactive interpreter: the last evaluated expression. So after the code above has executed, `_` will contain the string:

```
>>> _
SECSIOT
```

- Python's print statement, paired with the **string format operator (%)**, supports string substitution, much like the `printf()` function in C:

```
>>> print ("%s is number %d!" % ("Python", 1) )
Python is number 1!
```

Input Statement:

```
user =input('Enter login name: ')
Enter login name: root
print("your login is :",user)
your login is : root
```

```
num =input('Enter number: ')
Enter number: 1024
print("Doubling your number: %d" % (int(num) * 2))
Doubling your number: 2048
```

The `int()` function converts the string `num` to an integer so that the mathematical operation can be performed

```
>>> input
<built-in function input>
```

Comments:

The hash or pound (#) sign signals that a comment begins from the # and continues until the end of the line.

Single line comment :

```
>>> # one comment
>>> print ('Hello World!') # another comment
Hello World!
```

Multi line comments :

We may use hashtags (#) multiple times to construct multiline comment

```
# This
# Is
# Comment
```

We can observe that on running this code, there will be no output; thus, we utilize the strings inside triple quotes("""") as multi-line comments.

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

```
1. # Code to show how we use docstrings in Python
2.
3. def add(x, y):
4.     """This function adds the values of x and y"""
5.     return x + y
6.
7. # Displaying the docstring of the add function
8. print( add.__doc__ )
```

Output:

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

```
def foo():
    "This is a doc string."
    return True
```

Algorithm: Algorithm is a step-by-step process of solving a well-defined computational problem. In practice, in order to solve any complex real life problems, first we have to define the problem and then, design algorithm to solve it. Generally, algorithms are used to simplify the program implementation. Also, defined step-by-step procedure to solve the problem is called algorithm.

Example 1:

Write an algorithm to how to Make “Maggi Noodles”?

Step 1: Start

Step 2: Take pan with water Step

3: Put pan on the burner Step 4:

Switch on the gas/burner Step 5:

Put maggi and masala Step 6: Give

two minutes to boil Step 7: Take

off the pan

Step 8: Take out the maggi with the help of fork/spoon Step 9:

Put the maggi on the plate and serve it

Step 10: Stop.

Example 2:

Write an algorithm to print “Good Morning”.

Step 1: Start

Step 2: Print “Good Morning”

Step 3: Stop

Example 3:

Write an algorithm to find area of a rectangle.

Step 1: Start

Step 2: Take length and breadth and store them as L and B? Step

3: Multiply by L and B and store it in area

Step 4: Print area

Step 5: Stop

Program: A program is a collection instructions that can do a particular task or problem. Programming is the process of taking an algorithm and writing it in a particular programming language, so that it can be executed by a computer. Programmers can use any of the programming languages to write programs.

Python Programming Introduction:

Introduction:

Python is a high-level, interpreted, interactive, scripting, object-oriented and a reliable language. Python was developed by Guido van Rossum in the late 80's and early 90's at the National Research Institute for Mathematics and Computer Science in the Netherlands. It has derived from many other languages such as ABC, C, C++, Small Talk, Unix Shell and other scripting languages. The first version of language was released in 1991. Python has emerged as a very powerful and popular language.

Why the name Python? Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

Features of Python Programming

- 1) A **simple** language which is easier to learn Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#.
2. **Versatile** Python supports development of a wide range of applications ranging from simple text processing to WWW browser to games.
3. **Free and open-source** Python is an open source software. Therefore, anyone can freely distribute it, read the source code, edit it, and even use the code to write new programs.

4. **Portability** You can move Python programs from one platform to another, and run it without any changes. It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.
5. **Extensible and Embeddable** Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code. This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.
6. **A high-level, interpreted language** Unlike C/C++, you don't have to worry about difficult tasks like memory management, garbage collection and so on. Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.
7. **Large standard libraries** to solve common tasks Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself.
8. **Object-oriented** Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem very easily. With OOP, you are able to divide these complex problems into smaller sets by creating objects.

Applications of Python

Python is a high level general purpose programming language that is used to develop a wide range of applications including image processing, text processing, web, and enterprise level applications using scientific and numeric data from network. Some of the key applications of python includes:

- ❖ In operations of Google search engine, youtube, etc.
- ❖ Bit Torrent peer to peer file sharing is written using Python
- ❖ Intel, Cisco, HP, IBM, etc use Python for hardware testing.
- ❖ Maya provides a Python scripting API
- ❖ i-Robot uses Python to develop commercial Robot.
- ❖ NASA and others use Python for their scientific programming task.
- ❖ Python is used as an Embedded scripting language.
- ❖ GUI-based Desktop applications
- ❖ Games: Python support development of games.
- ❖ Business applications
- ❖ Operating systems
- ❖ Python is used for Network programming.

Interpreter : An interpreter is a program that converts program written in high-level language(Source code) into machine code understood by the computer. Interpreter executes one statement at a time.



scripting language: A **scripting language** is a programming language designed for integrating and communicating with other programming languages. Some of the most widely used scripting languages are JavaScript, VBScript, PHP, Perl, Python, Ruby, ASP and Tcl. Since a scripting language is normally used in conjunction with another programming language, they are often found alongside HTML, Java or C++.

First Python Program:

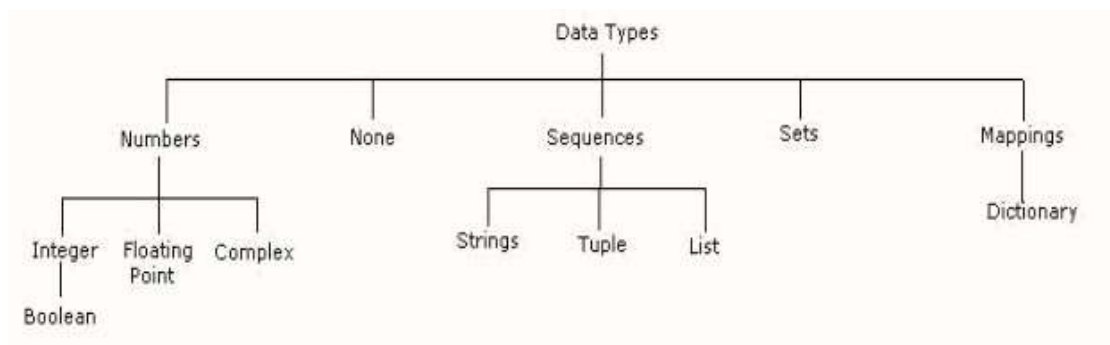
Example: To print a message on the screen

```
>>> print ("Hello World! ! !")
```

Hello World

Data Types:

Variables can hold values of different types called data types. The standard data types supported by Python includes -



- 1) Numbers
 - a. Integers
 - b. Floating Point Numbers
 - c. Complex Numbers
- 2) None

3) Sequences

- a. Strings
- b. Tuple
- c. List

4) Sets

5) Mappings – Dictionary

1) Numbers: Number refers Numerical Values. This data type is immutable i.e. its value cannot be changed. These are of three different types:

- a) Integers
- b) Float/floating point
- c) Complex

a) **Integers:** Numbers like 5 or other whole numbers are referred to as integers. Bigger whole numbers are called long integers. For example, 56958558585855L is a long integer. Note that a long integer must have 'l' or 'L' as the suffix.

Integers contain Boolean Type which is a unique data type, consisting of two constants, True & False. A Boolean True value is Non-Zero, Non-Null and Non-empty.

Example

```
>>> flag = True
>>> type(flag)
<type 'bool'>
```

b) **Floating point numbers:** Numbers with fractions or decimal point are called floating point numbers.

Example

```
y= 12.36
```

c) **Complex Numbers:** Numbers of $a+bi$ form are complex numbers.

2) None: This is special data type with single value. It is used to signify the absence of value/false in a situation. It is represented by None.

3) Sequence:

A sequence is an ordered collection of items, indexed by positive integers. It is combination of mutable and non mutable data types. Three types of sequence data type available in Python are Strings, Lists & Tuples.

i) **Strings:** String is an ordered sequence of letters/characters. String is also defined as a group or set of characters. They are enclosed in single quotes (' ') or double (" "). Strings are immutable data types.

Example

```
>>> a = 'Ram'
```

ii) **List:** List is data type available in python. It is a sequence in which elements are written as a list of comma separated values between square brackets. List is mutable data type which means the value of its elements can be changed.

Syntax: list_variable=[var1,var2,]

Example

```
rgukt = ["spam", 20.5,5]
```

iii) **Tuples:** Tuples is data type available in python. Tuples are sequence of elements separated by comma enclosed in parenthesis. They are immutable data type.

Example

```
my_tuple = (4, 2, 5, 8)
```

4) Sets: Set is an unordered collection of values, of any type, with no duplicate entry. Sets are immutable.

Example

```
s = set ([1,2,3,4])
```

5) Mapping: This data type is unordered and mutable. A dictionary comes under Mappings.

Dictionary: Dictionary is a data type available in python which store values as a pair of key and value. Each key is separated from its value by a colon (:), and consecutive items are separated by commas. The entire items in a dictionary are enclosed in curly brackets {}. Dictionary keys must be of immutable type.

Example: d = {1:'a',2:'b',3:'c'}

Mutable and Immutable Variables

Variable: Variables are reserved memory locations that stores values. Variables play a very important role in most programming languages, and Python is no exception. You can store any piece of information in a variable. Variables are nothing but just parts of your computer's memory where information is stored. To be identified easily, each variable is given an appropriate name. Every variable is assigned a name which can be used to refer to the value in the program.

A **mutable variable** is one whose value may change , whereas in an **immutable** variable the value cannot be change.

Example: Initializing values to variables

```
val= "hello"
```

```
num=75
```

```
amt=12.65
```

```
print (val)
```

```
print (amt)
```

```
print (num)
```

Output:

```
"hello"
```

```
75
```

```
12.65
```

Multiple assignments

Python allows you to assign a single value to several variables simultaneously. For

example: a = b = c = 1

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location.

You can also assign multiple objects to multiple variables. For example – Example:

```
a,b,c = 1,2,"john"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable

Expression and Statements

An expression is a combination of values, variable and operators that represents a value.

Example:

```
sum=a+b      (Expression)
```

a and b operands and + is operator.

Operators are the constructs that are used to manipulate the value of operands. Some basic operators include +, -, *, /.

Operands are the values.

A Python **statement** is a unit of code that the Python interpreter can execute.

Example of statement are:

```
>>> x=5
```

```
>>> area=x**2 #assignment statement
```

```
>>>print x #print statement 5
```

```
>>>print area
```

```
25
```

```
>>> print x, area
```

```
5 25
```


Value:

In mathematics and computer programming, a **value** is data element that may be stored in a variable. The value can be numeric (such as an integer, or a floating point number), or alphanumeric (such as a character, or a string).

Example: 1, 2.4, "Hello"

Python Identifiers:

Identifier is the name given to entities like class, functions, variables etc. For naming identifier, there are some basic rules that you must follow. These rules are

- 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 (_).
- Keywords cannot be used as identifiers.
- An identifier cannot start with a digit. 1variable is invalid, but variable1 is valid.
- Identifier names are case sensitive. For example myvar and myVar are not the same.
- Punctuation characters such @,\$,£ and % are not allowed within identifiers.
- Does not include space character in the identifier name

Example:

Valid identifier names are: sum, _my_var, num1, r, var_33, first etc

Invalid identifier names are: 1num, my-var, %check, basic sal etc

Comments

Comments are the non-executable statements in a program. They are just added to describe the code of the program. Comments make the program easily readable and understandable by the programmer as well as other users who are seeing the code. The interpreter simply ignores the comments.

In python, a has sign (#) symbol writing as a comment.

Example

```
#This is a comment
#print out Hello
print('Hello')
#program ends here
```

Multi-line comments

If we have comments that extend multiple lines, We can use triple quotes, either `'''` or `"""`.

Example

```
"""This is also a  
perfect example  
of multi-line comments"""
```

Note: A program can have any number of comments.

Keywords in Python

Keywords are the reserved words in Python. We cannot use a keyword as variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language. In Python, keywords are case sensitive. There are 33 keywords in Python. All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

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

Indentation:

White space at the beginning of the line is called ***indentation***. These whitespaces or the indentation are very important in python. Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

Leading white space (spaces and tabs) at the beginning of each statement, which is used to determine the group of statement, is known as “indentation”.

Example

If $A > B$:

```
    print “A is Big”    # Block1
```

else:

```
    print “B is Big”    # Block2
```

In the above example, if statements are a type of code block. If the “if” expression evaluates to true, then Block1 is executed, otherwise, it executes Block2. Obviously, blocks can have multiple lines. As long as they are all indented with the same amount of spaces, they constitute one block.

Operators and Operands in Python

Operators: Operators are special symbols that are used to manipulate the value of operands. They are applied on operand(s), which can be values or variables. Operators when applied on operands form an expression. Operators are categorized as Arithmetic, Relational, Logical and Assignment.

Operands: Values and variables are known as operands.

Example:

```
result=a+b
```

(a, b operands and + is operator)

Python supports different types of operators which are as follows:

- 1) Arithmetic operators
- 2) Relational or Comparison operators
- 3) Logical Operators
- 4) Assignment Operators
- 5) Unary Operators
- 6) Bitwise Operators
- 7) Membership operators
- 8) Identity operators

1) Arithmetic operators

Some basic arithmetic operators are +, -, *, /, %, ** and //. You can apply these operators on numbers as well as on variables to perform corresponding operations.

Symbol/Operator	Description	Example 1	Example 2
+	Addition	>>>55+45 100	>>> "Good" + "Morning" GoodMorning
-	Subtraction	>>>55-45 10	>>>30-80 -50
*	Multiplication	>>>55*45 2475	>>> "Good"* 3 GoodGoodGood
/	Division	>>>17/5 3 >>>17/5.0 3.4 >>> 17.0/5 3.4	>>>28/3 9
%	Remainder/ Modulo	>>>17%5 2	>>> 23%2 1
**	Exponentiation	>>>2**3 8 >>>16**.5 4.0	>>>2**8 256
//	Integer Division	>>>7.0//2 3.0	>>>3// 2 1

2) Relational or Comparison operators

Comparison operators also known as relational operators are used to compare the values on its either sides and determines the relation between them. For example, assume a=100 and b = 200, we can use the comparison operators on them.

Symbol/Operator	Description	Example 1	Example 2
<	Less than	>>>7<10 True >>> 7<5 False	>>>"Hello"< "Goodbye" False >>>'Goodbye'< 'Hello' True

		<pre>>>> 7<10<15 True >>>7<10 and 10<15 True</pre>	
>	Greater than	<pre>>>>7>5 True >>>10<10 False</pre>	<pre>>>>”Hello”> “Goodbye” True >>>'Goodbye'> 'Hello' False</pre>
<=	less than equal to	<pre>>>> 2<=5 True >>> 7<=4 False</pre>	<pre>>>>”Hello”<= “Goodbye” False >>>'Goodbye' <= 'Hello' True</pre>
>=	greater than equal to	<pre>>>>10>=10 True >>>10>=12 False</pre>	<pre>>>>”Hello”>= “Goodbye” True >>>'Goodbye' >= 'Hello' False</pre>
!=	not equal to	<pre>>>>10!=11 True >>>10!=10 False</pre>	<pre>>>>”Hello”!= “HELLO” True >>> “Hello” != “Hello” False</pre>
==	equal to	<pre>>>>10==10 True >>>10==11 False</pre>	<pre>>>>”Hello” == “Hello” True >>>”Hello” == “Good Bye” False</pre>

3) Logical Operators

Python supports three logical operators logical and, logical or and logical not. Normally, the logical expressions are evaluated from left to right.

Symbol	Description
and (&)	If both the operands are true, then the condition becomes true. Ex: x and y
or ()	If any one of the operand is true, then the condition becomes true. Ex: x or y
Not (!)	True if operand is false (complements the operand) Ex: not x

4) Assignment Operators

Assignment operators are used in Python to assign values to variables or operands. It is also known as shortcut operators that includes +=, -=, *=, /=, %=, //= and **= etc.

Symbol/Operator	Description	Example 1	Example 2
=	Assigned values from right side operands to left variable	x=12	c=a, assigns value of a to the variable c
+=	added and assign back the result to left operand	x+=2	x+=2 is same as x=x+2
-=	subtracted and assign back the result to left operand	x-=2	x-=2 is same as x=x-2
=	multiplied and assign back the result to left operand	x=2	x*=2 is same as x=x*2
/=	divided and assign back the result to left operand	x/=2	x/=2 is same as x=x/2
%=	taken modulus using two operands and assign the result to left operand	x%=2	x%=2 is same as x=x%2
=	performed exponential (power) calculation on operators and assign value to the left operand	x=2	x**=2 is same as x=x**2
//=	performed floor division on operators and assign value to the left operand	x /= 2	x//=2 is same as x=x//2

5) Unary Operators

Unary operators act on single operands. Python supports Unary minus operator. An operand is preceded by a minus sign, the unary operator negates its value.

For example, if a number is positive, it becomes negative when preceded with a unary minus operator. Similarly, if the number is negative, it becomes positive after applying the unary minus operator.

Ex: $b=10$

$a = -(b)$

The result of the expression is $a = -10$, because variable b has a positive value. After applying unary minus operator $(-)$ on the operand b , the value becomes -10 , which indicates it as a negative value.

6) Bitwise Operators

Bitwise operators perform operations at the bit level. These operators include bitwise AND, bitwise OR, bitwise XOR, and shift operators.

Bitwise AND (&)

Bitwise AND $\&$ will give 1 only if both the operands are 1 otherwise, 0.

The truth table for bitwise AND.

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

In the following example, we have two integer values 1 and 3 and we will perform bitwise AND operation and display the result.

Example:

```
# variables
x = 1
y = 3

# bitwise operation
result = x & y
print("result:", result)          # result: 1
```

Bitwise OR

Bitwise OR $|$ will give 0 only if both the operands are 0 otherwise, 1. The truth table for bitwise OR.

A	B	A B
0	0	0
0	1	1

1	0	1
1	1	1

In the following example we have two integer values 1 and 2 and we will perform bitwise OR operation and display the result.

Example:

```
# variables
x = 1
y = 2

# bitwise operation
result = x | y

print("result:", result)      # result: 3
```

Bitwise XOR

Bitwise XOR ^ will give 1 for odd number of 1s otherwise, 0.

The truth table for bitwise XOR.

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

In the following example we have two integer values 2 and 7 and we will perform bitwise XOR operation and display the result. Example,

```
# variables
x = 2
y = 7

# bitwise operation
result = x ^ y

print("result:", result)      # result: 5
```

Shift Operators:

Python supports two bitwise shift operators. They are shift left (<<) and shift right (>>). These operations are used to shift bits to the left or to the right.

i) << Binary Left Shift: The left operands value is moved left by the number of bits specified by the right operand.

Example:

```
a=60
```

```
a<<2
```

```
output: 240
```

ii) >> Binary Right Shift: The left operands value is moved right by the number of bits specified by the right operand.

Example:

```
a=60
```

```
a >> 2
```

```
output: 15
```

7) Membership operators

Python supports two types of membership operators – in and not in. These operators used to test a value or variable is found in a sequence. (strings, list or tuple)

Symbol/Operator	Description	Example
In	True if value/variable is found in the sequence	>>>x=[1,2, 4,5,7,9] >>>5 in x True
not in	True if value/variable is not found in the sequence	>>>x=[1,2, 4,5,7,9] >>>10 not in x True

Example:

```
>>>a="Hello World"
```

```
>>>print ('H' in a)
```

```
True
```

```
>>>print ('m' in a)
```

```
False
```

8) Identity operators

Python supports two types of identity operators. These operators compare the memory locations of two objects.

is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Symbol/Operator	Description	Example
is	True if the operands/values point(refer) to the same object	>>>x=5 >>>y=5 >>>x is y True
is not	True if the operands/values do not point(refer) to the same object	>>>a=[1,2, 4,5,7,9] >>>b=[1,2, 4,5,7,9] >>> x is not y True

Above given example, see that x and y are integers of same values, so they are equal as well as identical. But a and b are list. They are equal but not identical. Since list are mutable (can be changed), interpreter locates them separately in memory although they are equal.

Operators Precedence and Associativity

When an expression has more than one operator, then it is the relative priorities of the operators with respect to each other that determine the order in which the expression will be evaluated. The given table shows the lists of all operators from highest precedence to lowest.

Operator	Description
**	Exponentiation (raise to the power)
+, -	unary plus and minus
*, /, %, //	Multiply, divide, modulo and floor division
+, -	Addition and subtraction
>>, <<	Right and Left bitwise shift operators
&	Bitwise AND

<, <=, >, >=	Comparison operators
==, !=	Equality operators
% =, / =, // =, - =, + =, * =	Assignment operators
is, is not	Identity operators
in, not in	Membership operators
not, and, or	Logical operators

Operator precedence table is important as it affects how an expression is evaluated. For example

```
>>>10+3*5
```

```
160
```

This is because * has higher precedence than +, so it first multiplies 30 and 5 and then adds 10.

Input and Output:

Input Operation: The input() function prompts the user to provide(ask) some information on which the program can work and give the result. Always, the input function takes user's input as a string. So whether input a number or string, it is treated as a string only.

Example:

```
>>>x=input("Enter your name")
```

Enter your name: ABC

```
#Program to read variables from the user
name=input("What's your name?")
age=input("Enter your age:")
print (name + ", you are " + age + "years old")
```

Output:

What's your name? Ramesh

Enter your age: 20

Ramesh, you are 20 years old

Print Statement

Syntax: print expression/constant/variable

Print evaluates the expression before printing it on the monitor. Print statement outputs an entire (complete) line and then goes to next line for subsequent output (s). To print more than one item on a single line, comma (,) may be used.

Example

```
>>> print
“Hello” Hello
>>> print 5.5
5.5
>>> print 4+6
10
>>a=5
>>b=10
>>print (a,b)
```

Type Conversion or Type Casting

Convert a value from one data type to another data type known as type conversion or type casting. The conversion can be done explicitly (programmer specifies the conversions) or implicitly (Interpreter automatically converts the data type). The list of conversion built-in functions has given below.

Function	Description
int(x)	Converts x to an integer
long(x)	Converts x to a long integer
float(x)	Converts x to a floating point number
str(x)	Converts x to a string
tuple(x)	Converts x to a tuple
list(x)	Converts x to a list
set(x)	Converts x to a set

ord(x)	Converts a single character to its integer value
oct(x)	Converts an integer to an octal string
hex(x)	Converts an integer to a hexadecimal string
chr(x)	Converts an integer to a character
unichr(x)	Converts an integer to a Unicode character
dict(x)	Creates a dictionary if x forms a (key-value) pair

Errors

An error is a term used to describe any issue that arises unexpectedly that cause a computer to not function properly. There are three types of errors generally occur during running (execution) of a program. They are (i) Syntax error; (ii) Logical error; and (iii) Runtime error.

(i) **Syntax error:** Every programming language has its own rules and regulations (syntax). If we overcome the particular language rules and regulations, the syntax error will appear (i.e. an error of language resulting from code that does not conform to the syntax of the programming language). It can be recognized during compilation time.

Example

```
a = 0
while a < 10
    a = a +
    1 print a
```

In the above statement, the second line is not correct. Since the while statement does not end with “:”. This will flash a syntax error.

(ii) **Logical error:** Programmer makes errors while writing program that is called “logical error”. It is an error in a program's source code that results in incorrect or unexpected result. It is a type of runtime error that may simply produce the wrong output or may cause a program to crash while running. The logical error might only be noticed during runtime, because it is often hidden in the source code and are typically harder to find and debug.

```
a = 100
```

```
while a < 10:
```

```
    a = a +
```

```
    1 print a
```

In the above example, the while loop will not execute even a single time, because the initial value of “a” is 100.

(iii) **Runtime error:** A runtime error is an error that causes abnormal termination of program during running time. In general, the dividend is not a constant but might be a number typed by you at runtime. In this case, division by zero is illogical. Computers check for a "division by zero" error during program execution, so that you can get a "division by zero" error message at runtime, which will stop your program abnormally. This type of error is called runtime error.

Example

(a)

```
A=10
```

```
B=0
```

```
print (A/B)
```

Exercise

1. Write a program that asks two people for their names; stores the names in variables called name1 and name2; says hello to both of them.
2. Write a Python program which accepts the radius of a circle from the user and compute the area.

Sample Output :

```
r = 1.1
```

```
Area = 3.8013271108436504
```

3. Write a Python program which accepts the user's first and last name and print them in reverse order with a space between them.
4. Write a Python program that will accept the base and height of a triangle and compute the area.
5. Write a Python program to solve $(x + y) * (x + y)$.

Test Data : x = 4, y = 3

Expected Output : $(4 + 3) ^ 2 = 49$

6. Write a python program to sum of the first n positive integers.

7. Write a Python program to swap two variables.
8. Write a program to enter two integers and then perform all arithmetic operations on them.
9. Write a program to perform string concatenation.
10. Write a program to print the ASCII value of a character.
11. Write a program to swap two numbers without using temporary variable.
12. Write a program to calculate simple interest.
13. Write a program that prompts users to enter two integers x and y. The program then calculates and display x^y .
14. Write a program that calculates numbers of seconds in a day.
15. Write a program to calculate salary of an employee given his basic pay(to be entered by the user), HRA =10 percent of basic pay, TA=5 percent of basic pay. Define HRA and TA as constants and use them to calculate the salary of the employee.
16. Write a program to print the digit at one's place of Number.
17. Write a program to calculate average of two numbers. Print their deviation.
18. Write a program to covert a floating point number into the corresponding integer.
19. Write a program to convert a integer into the corresponding floating point number.
20. Write a program to calculate area of triangle using Heron's formula. (Hint: Heron's formula is :
$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

Assignment-I

1. Define datatype. What are the datatypes available in Python? Explain in detail.
2. What is an Identifier? List out the rules of an identifier.
3. What is a Python? Write notes on Python history, features and Applications.
4. What is indentation? Explain with an example.
5. Define the following terms.
 - i) Algorithm ii) Interpreter iii) variable iv) Expression v) Value vi) comments

Assignment-II

1. What is an operator? Explain all the operators available in Python.
2. What is the type conversion or Type casting? Explain type casting functions along with examples.

Control Statements:

A control statement is a statement that determines the control flow of a set of instructions. It decides the sequence in which the instructions in a program are to be executed. A control statement can either comprise of one or more instructions.

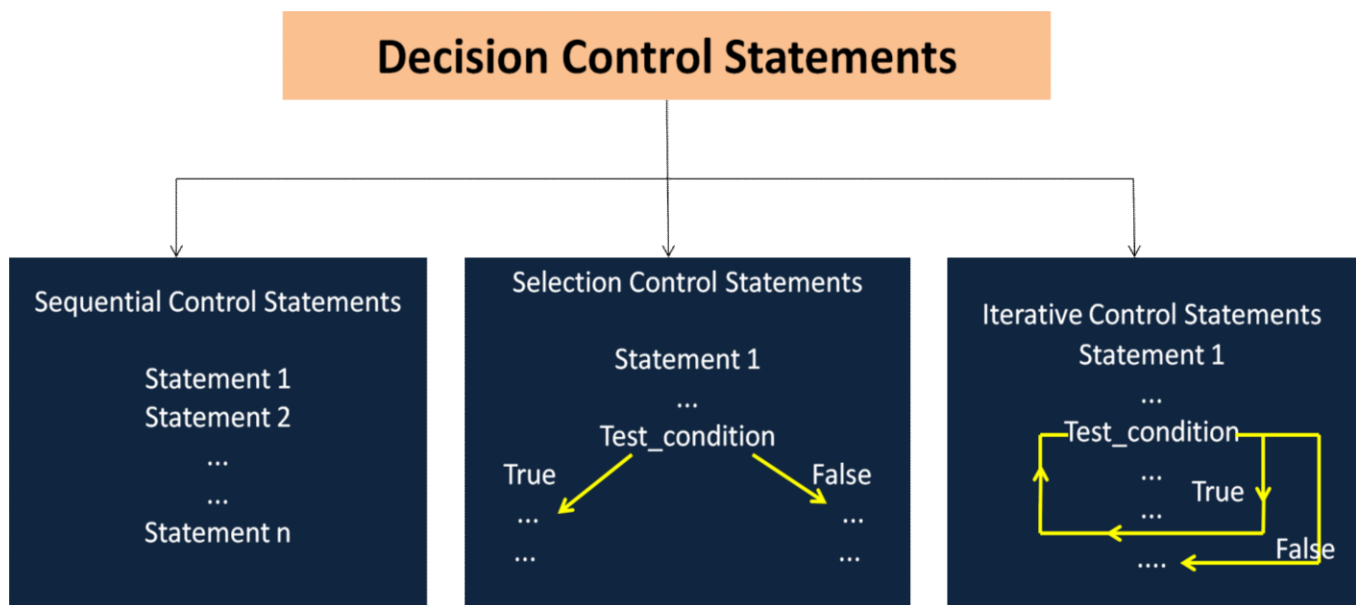
The three fundamental methods of control flow in a programming language are

- i) Sequential Control Statements
- ii) Selection Control Statements
- iii) Iterative Control Statements

i) Sequential Control Statements: The code of Python program is executed sequentially from the first line of the program to its last line. That is, the second statement is executed after the first, the third statement is executed after the second, so on and so forth. This method is known as sequential Control flow and these statements called as sequential control statements.

ii) Selection Control Statements: Some cases, execute only a set of statements called as selection control statements. This method is known as selection control flow.

iii) Iterative Control Statements: execute a set of statements repeatedly called as Iterative control statements. This method is known as Iterative control flow.



SELECTION/CONDITIONAL CONTROL STATEMENTS

The decision control statements usually jumps from one part of the code to another depending on whether a particular condition is satisfied or not. That is, they allow you to execute statements selectively on certain decisions. Such type of decision control statements are known as selection control statements or conditional branching statements. Python supports different types of conditional branching statements which are as follows:

- If statement
- If –else statement
- Nested if statement
- If-elif-else statement

If Statement: An if statement is a selection control statement based on the value of a given Boolean expression. The if structure may contain 1 statement or n statements enclosed within the if block. First, the test expression(Boolean expression) is evaluated. If the test expression is True, the statement of if block are executed, otherwise these statements will be skipped and jump to next statement which is outside of the if block.

Syntax of if statement

```
if test_expression:
    Statement1
    Statement2
    .....
    Statement n
Statement x
```

Example 1: Program to increment a number if it is positive.

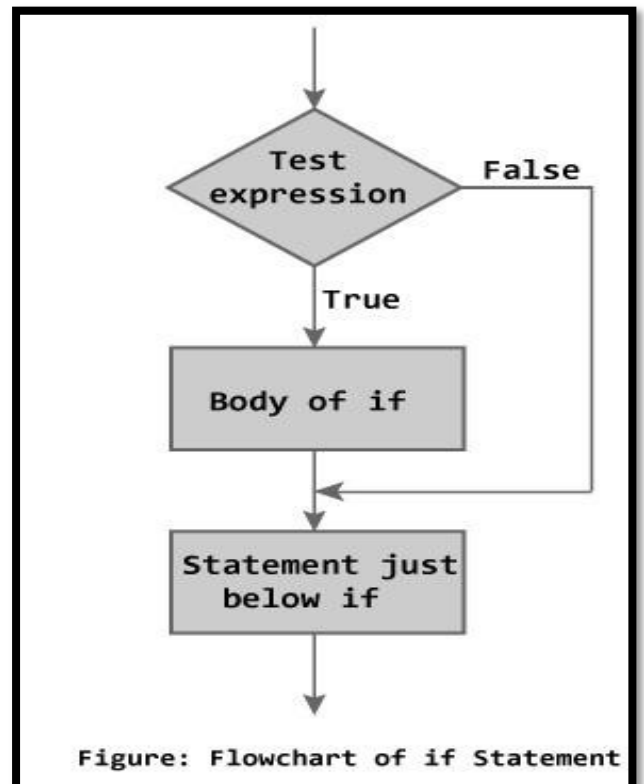
```
x=10
if (x>0):
    x=x+
1 print (x)
```

Output:

```
x=11
```

Example 2: Write a program to determine whether a person is eligible to vote.

```
age=int(input("Enter the age: "))
```



```
if (age>18):  
    print ("You are eligible to vote")
```

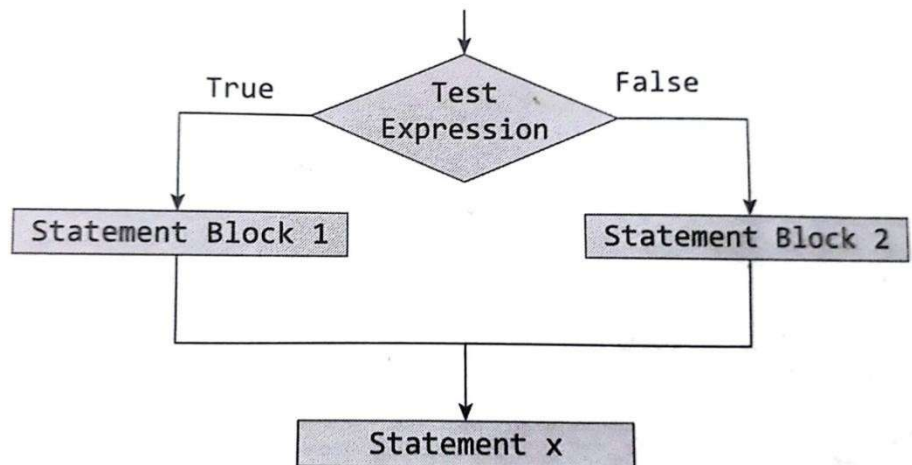
Output:

Enter the age: 35
You are eligible to vote

if –else Statement: An if-else construct, first the test expression (Boolean expression) is evaluated. If the expression is True, statement block 1 is executed and statement block 2 is skipped. Otherwise, if the expression is False, statement block 2 is executed and statement block 1 is ignored.

Syntax of if statement

```
if test_expression:  
    Statement block 1  
else:  
    Statement block 2  
Statement x
```



Example 3: Write a program to determine whether a person is eligible to vote or not. If he is not eligible, display how many years are left to be eligible.

```
age=int(input("Enter the age:"))
```

```
if (age>=18):  
    print ("You are eligible to vote")
```

```
else:
```

```
    yrs=18-age
```

```
    print ("You have to wait for another"+ str(yrs)+ "years to  
    cast your vote")
```

Output:

Enter the age: 10
You have to wait for another 8 years to cast your vote.

Example 4: Write a program to find whether the given number is even or odd.

```
num=int(input("Enter any number :"))
```

```
if (num%2==0):  
    print (num, "is even")
```

```
else:
```

```
    print (num, "is odd")
```

Output:

Enter any number: 125

125 is odd

Nested if statements: To perform more complex check, if statement can be nested, that is can be placed one inside the other. In such a case, the inner if statement is the statement part of the outer one. Nested if statements are used to check if more than one condition is satisfied.

Example 5: Write a python program to find the given number is positive or negative number, if positive number then compare with 100, if number is greater than 100, output display as “high” otherwise display as “low”.

```
num = float(input("Enter a number: "))
```

```
if num > 0:
```

```
    if num > 100:
```

```
        print("High")
```

```
    else:
```

```
        print("Low")
```

```
else:
```

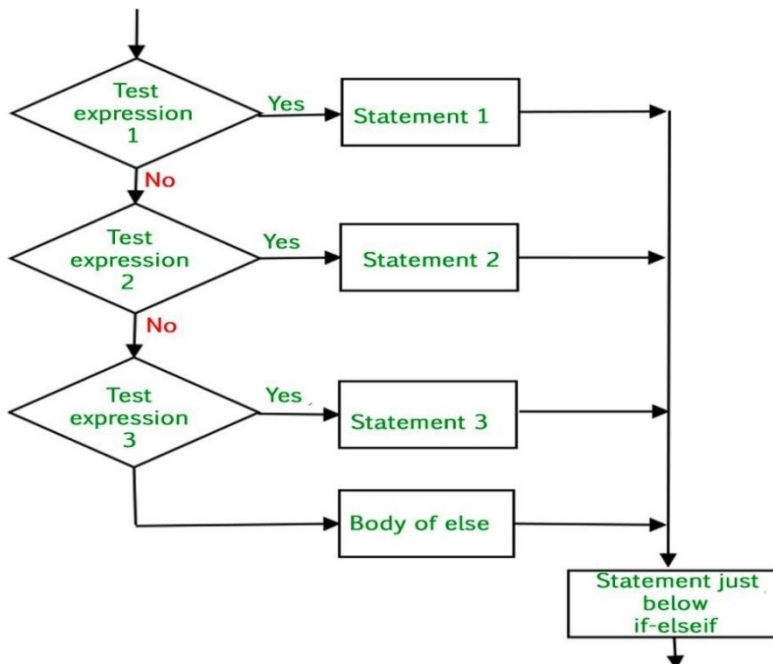
```
    print("Negative number")
```

Output:

Enter a number: 5

low

if-elif-else Statement: The elif statement allows to check multiple expressions and executes block of statements wherever the condition returns as True. From there, it exit from the entire *if-elif-else* block. If any of the expression not returns as true, then it executes the *else* block code.



Syntax of if-elif-else Statement

```
if ( test expression 1)
    statement block 1
elif ( test expression 2)
    statement block 2
.....
elif (test expression N)
    statement block N
else
    Statement Block X
Statement Y
```

Example 6: Program to test whether a number entered by the user is negative, positive or equal to Zero?

```
num=int(input("Enter any number"))
if (num==0):
    print ("The value is equal to zero")
elif (num>0):
    print ("The number is positive")
else:
    print ("The number is negative")
```

Output:

Enter any number : -10
The number is negative

Exercise:

1. Write a program to find larger of two numbers.
2. A Company decides to give bonus to all its employees on Diwali. A 5% bonus on salary is given to the male workers and 10% bonus on salary to the female workers. Write a Python program to enter the salary of the employee and gender of the employee. If the salary of the employee is less than Rs/- 10,000 then the employee gets an extra 2% bonus on salary. Calculate the bonus that has to be given to the employee and display the salary that the employee will get.
3. Write a program to find whether a given year is a leap year or not.
4. Write a program to determine whether the character entered is a vowel or not.
5. Write a program to find the greatest number from the three numbers.
6. Write a program that prompts the user to enter a number between 1-7 and then displays the corresponding day of the week.
7. Write a program to calculate tax given the following conditions:
If income is less than 1,50,000 then no tax
If taxable income is 1,50,001 – 3,00,000 then charge 10% tax
If taxable income is 3,00,001 – 5,00,000 then charge 20% tax
If taxable income is above 5,00,001 then charge 30% tax
8. Write a program to enter the marks of a student in four subjects. Then calculate the total and aggregate, and display the grade obtained by the student. If the student scores an aggregate greater than 75%, then the grade is Distinction. If aggregate is $60 \geq$ and < 75 , then the grade is First Division. If aggregate is $50 \geq$ and < 60 , then the grade is Second Division. If aggregate is $40 \geq$ and < 50 , then the grade is Third Division. Else the grade is fail.
9. Write a program to calculate roots of a quadratic equation.
10. Write a program to take input from the user and then check whether it is a number or a character. If it is a character, determine whether it is in uppercase or lowercase.
11. Write a program that prompts users to enter a character (O, A,B,C,F). Then using if-elif-else construct display Outstanding, Very Good, Good, Average and Fail respectively.
12. Write a program to read two numbers. Then find out whether the first number is a multiple of the

second number.

ITERATIVE CONTROL STATEMENTS/LOOPING STATEMENTS

Iterative statements are decision control statements that are used to repeat the execution of a list/set/group of statements. Python language supports two types of iterative statements **While** loop and **for** loop.

While loop:

The while loop provides a mechanism to repeat one or more statements while a particular condition is True.

While loop, the condition is tested before any of the statements in the statement block is executed. If the condition is True, only then the statements will be executed otherwise if the condition is False, it is jump to next statement which is outside of the while loop.

Note: If the condition of a while loop is never updated and condition never become False, then the computer will run into an infinite loop.

Syntax of While Loop:

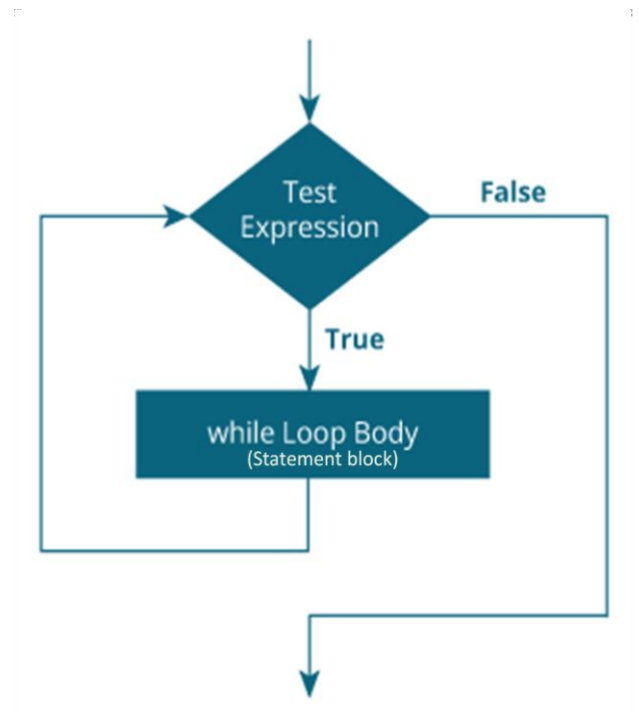
```
statement x while  
(condition):  
    Statement block  
Statement y
```

Example 7: Write a python program to print first 10 numbers using a while loop. $i=0$

```
while (i<=10):  
    print (i)  
    i=i+1
```

Output:

1
2
3
4
5
6
7
8



9
10

Example 8: Write a python program to print first 10 numbers using a while loop on the same line. i=0

```
while (i<=10):  
    print(i, end= ' ')  
    i=i+1
```

Output:

1 2 3 4 5 6 7 8 9 10

Example 9: Program to display the Fibonacci sequence up to n-th term where n is provided by the user nterms =

```
10  
n1 = 0  
n2 = 1  
count = 0  
if nterms <= 0:  
    print("Please enter a positive integer")  
elif nterms == 1:  
    print("Fibonacci sequence upto",nterms,":") print(n1)  
else:
```

p
r
i
n
t
(
"
F
i
b
o
n
a
c
c
i

s
e
q
u
e
n
c
e

u
p
t
o
"
,
n
t
e
r
m
s
,
"
:
"
)

w
h
i
l
e

c
o
u
n
t

<

n
t
e
r
m
s
:

p
r
i
n
t

(
n
1
,
e
n
d
=
,
,
,
)

n
t
h

=

n
1

+

n
2
n
1

=

n
2

n
2

=

n
t
h

count += 1

Output:

Fibonacci sequence upto 10 :

0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

Note: In simple terms, end specifies the values which have to be printed after the print statement has been executed. In above example, the values printed on the same line, used end with a separator. You can specify any separator like tab (\t), space, comma, etc., with end.

Exercise:

1. Write a program to calculate the sum and average of first 10 numbers.

2. Write a program to print 20 horizontal asterisks(*).
3. Write a program to calculate the sum of numbers from m to n.
4. Write a program to read the numbers until -1 is encountered. Also count the negative, positive and zeros entered by user.
5. Write a program to read the numbers until -1 is encountered. Find the average of positive number and negative numbers entered by the user.
6. Write a program to find whether the given number is an Armstrong Number or not.
7. Write a program to enter a decimal number. Calculate and display the binary equivalent of this number.
8. Write a program to enter a binary number and convert it into decimal number.
9. Write a program to read a character until a * is encountered. Also count the number of uppercases, lowercase and numbers entered by the users.
10. Write a program to enter a number and then calculate the sum of the digits.
11. Write a program to calculate GCD of two numbers.
12. Write a program to print the reverse of a number.
13. Write a program to read a number from the user and find whether it is a palindrome number or not.
14. Write program using a while loop that asks the user for a number and prints a countdown from that number to zero.
15. Write a program that prompts users to enter numbers. Once the user enters -1, it displays the count, sum and average of even numbers and that of odd numbers.

For loop:

The for loop provides a mechanism to repeat a task(set of statements) until a particular condition is true.

The For loop is known as a determine loop because the programmer knows exactly how many times the loop will repeat. The number of times the loop has to be executed can be determined mathematically checking the logic of the loop.

The for in statement is a looping statement used in python to iterate over a sequece(list, tuple, string) of objects i.e., go through each item in a sequence. Sequence means an ordered collection of itmes.

Syntax of for Loop

for val in sequence:

 Body of for (Statement Block)

Note: The for loop is widely used to execute a single or a group of statements.

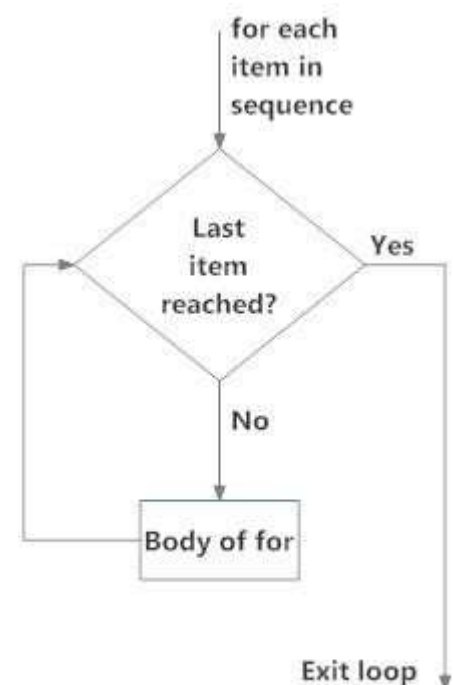


Fig: operation of for loop

The range function():

The range() is a built-in function in Python that is used to iterate over a sequence of numbers. The syntax of range() is

```
range(beginning, ending, [step size])
```

The range() produces a sequence of numbers starting with beginning (inclusive) and ending with one less than the number end. The step argument is optional. By default, every number in range is incremented by 1 but we can specify a different increment using step.

Note: Step size can be either positive or negative but it cannot be equal to zero.

Example 10:

Program:

```
for i in range (1,5):  
    print (i, end=" ")
```

Output: 1 2 3 4

Example 11:

Program:

```
for i in range(1, 10, 2):  
    print (i, end= " ")
```

Output: 1 3 5 7 9

Example 12:

Program:

```
for i in range(10):  
    print (i, end= " ")
```

Output: 0 1 2 3 4 5 6 7 8 9

Example 13:

Program:

```
for i in range(1, 15):  
    print (i, end= " ")
```

Output: 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Example 14:

Program:

```
for i in range(1, 20,3): print
    (i,          end=“ ”)
```

Output: 1 4 7 10 13 16 19

Exercise – For Loop:

1. Write a program using for loop to calculate the average of first n natural numbers.
2. Write a program to print the multiplication table of n, where n is entered by the user.
3. Write a program using for loop to print all the numbers from m-n thereby classifying them as even or odd.
4. Write a program using for loop to calculate factorial of a number.
5. Write a program using for loop to calculate $\text{pow}(x,n)$
6. Write a program that display all leap years from 1800 – 1900
7. Write a program to sum the series – $1 + 1/2 + 1/3 + \dots + 1/n$.
8. Write a program to sum the series – $1 + 1/2^2 + 1/3^2 + \dots + 1/n^2$.
9. Write a program to sum the series – $1/2 + 2/3 + 3/4 + \dots + n/(n+1)$.
10. Write a program to sum the series – $1/1 + 2^2/2 + 3^2/3 + \dots + n^2/n$.
11. Write a program to calculate sum of cubes of numbers from 1-n.
12. Write a program to sum of squares of even numbers.

Nested Loops:

Nested loops, that is, loops can be placed inside other loops. This feature works any loop like while loop as well as for loop, but it is most commonly used with the for loop, because this is easiest to control.

Exercise:

1. Write a program to print the following pattern.

```
RGUKT 1 – 1 2 3 4 5
RGUKT 2 – 1 2 3 4 5
RGUKT 3 – 1 2 3 4 5
RGUKT 4 – 1 2 3 4 5
RGUKT 5 – 1 2 3 4 5
```

2. Write a program to print the following pattern.

```
*****
*****
*****
*****
*****
```

3. Write a program to print the following pattern.

```
*  
**  
***  
****  
*****
```

5. Write a program to print the following pattern. 1

```
1 2  
1 2 3  
1 2 3 4  
1 2 3 4 5
```

6. Write a program to print the following pattern. 1

```
22  
333  
4444  
55555
```

7. Write a program to print the following pattern. 1

```
2 3  
4 5 6  
7 8 9 10
```

8. Write a program to print the following pattern. 1

```
12  
123  
1234  
12345
```

9. Write a program to print the following pattern. 1

```
1 2 1  
1 2 3 2 1  
1 2 3 4 3 2 1
```

1 2 3 4 5 4 3 2 1

10. Write a program to print the following pattern. 1

```
  2  2
 3 3 3
4 4 4 4
5 5 5 5 5
```

11. Write a program to print the following pattern.

```
* * * * *
*       *
*       *
*       *
* * * * *
```

12. Write a program to print the following pattern.

```
$ * * * *
* $       *
*  $     *
*    $  *
* * * * $
```

break STATEMENT:

Break can be used to unconditionally jump out of the loop. It terminates the execution of the loop. Break can be used in while loop and for loop. Break is mostly required, when because of some external condition, we need to exit from a loop.

Example 15: Program to using the break statement i=1

while (i<=10):

 print (i, end=" ")

 if (i==5):

 break

 i=i+1

print ()

print ("completed")

Output:
1 2 3 4 5
completed

Continue Statement:

This statement is used to tell Python to skip the rest of the statements of the current loop block and to move to next iteration, of the loop. Continue will return back the control to the beginning of the loop. This can also be used with both while and for statement.

Example 16: Program to using the continue statement

```
for i in range(1,11):  
    if (i==5):  
        continue  
    print (i, end=" ")  
print ("\n completed")
```

Output:
1 2 3 4 6 7 8 9 10

Exercise:

1. Write a program to classify a given number as prime or composite.
2. Write a program to read the numbers until -1 is encountered. Count the number of prime numbers and composite numbers entered by the user.
3. Write a Python program that prints all the numbers from 0 to 6 except 3 and 6.
4. Write a Python program to Print first 100 prime numbers.

Pass Statement:

The pass statement is used when a statement is required syntactically but no command or code has to be executed. It specifies a null operation or No Operation (NOP) statement. Nothing happens when the pass statement is executed.

Example17: Program to demonstrate pass statement for letter in "Hello":

```
pass          #The statement is doing nothing  
print ("PASS:", letter)  
print ("Done")
```

Output:
PASS:
H
PASS:
E
PASS:
L

PASS:
L PASS:
O
Done

The pass statement is used as a placeholder. For example, if we have a loop that is not implemented yet, but we may wish to write some piece of code in it in the future. In such cases, pass statement can be written because we cannot have an empty body of the loop. Though the pass statement will not do anything but it will make the program syntactically correct.

Note: The difference between comment and pass statements is, pass is null statement that is executed by python interpreter, comment is non-executable statement that is ignored(not executed) by python interpreter.

The else STATEMENT used with Loops:

Unlike C and C++, in python have the **else** statement associated with a loop statements.

1) In for-else statement, the **else** statement is executed when the loop has completed iterating. **Break** statement can be used to stop a for loop. In such case, the else statement is ignored. Remaining cases, for loop's **else** part is executed if no break occurs.

Example 18:

```
for i in range(1,11):  
    print (i, end="")  
else:  
    print ("\n Done")
```

Output:

1 2 3 4 5 6 7 8 9 10
Done

Example 19:

```
for i in range(1,20):  
    if (i==5):  
        break  
    print (i, end="")  
else:  
    print ("\n Done")
```

Output:

1 2 3 4

2) **else** statement with the while loop, the else statement is executed when the conditions becomes False. The while loop can be terminated with a break statement. In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

Example 20:

```
i=1
while (i<0):
    print (i)
    i=i-1
else:
    print (i, "is not negative so loop did not execute")
```

Output:

1 is not negative so loop did not execute

Example 21:

```
i=1
while (i<10):
    if (i==6):
        break
    print (i, end=" ")
    i=i+1
else:
    print ("Completed")
```

Output:

1 2 3 4 5

Assignment-III

1. Write a detail notes on conditional/selectional branching statements supported by python?
2. Explain *for* and *while* loop in python along with syntax, flowchart and an example?
3. Write a program to print the following pattern.

```
PYTHON 1 – 1 2 3 4 5
PYTHON 2 – 1 2 3 4 5
PYTHON 3 – 1 2 3 4 5
PYTHON 4 – 1 2 3 4 5
PYTHON 5 – 1 2 3 4 5
```

4. Write a program to print the following pattern.

```
*****
*****
```



```
*****
*****
*****
*****
```

5. Write a program to print the following pattern.

```
*
**
***
****
*****
```

6. Write a program to print the following pattern. 1

```
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

7. Write a program to print the following pattern. 1

```
22
333
4444
55555
```

8. Write a program to print the following pattern. 1

```
2 3
4 5 6
7 8 9 10
```

9. Write a program to print the following pattern. 1

```
12
123
1234
12345
```

10. Write a program to print the following pattern. 1

```

    1 2 1
  1 2 3 2 1
1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1

```

11. Write a program to print the following pattern. 1

```

    2 2
  3 3 3
4 4 4 4
5 5 5 5 5

```

12. Write a program to print the following pattern.

```

$ * * * *
* $      *
*   $    *
*     $  *
* * * * $

```

Assignment-IV

1. Explain range() function along with an example?
2. Explain break statement with the help of an example?
3. Explain continue statement with the help of an example?
4. Explain about pass statement in Python?
5. Write short notes on for-else and while-else?
6. Write a Python program to print prime numbers between 500 to 600.

Functions

Function is a group of related statements that perform a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. It avoids repetition and makes code reusable.

Syntax of Function

```
def function_name(parameters):           #Keyword def marks the start of function header.  
    statement(s)
```

Basically, we can divide functions into the following two types:

1. Built-in functions - Functions that are built into Python.
2. User-defined functions - Functions defined by the users themselves.

Built-in Functions

Functions that come built into the Python language itself are called built-in functions and are readily available to us.

Functions like *print()*, *input()*, *len()* etc. that we have been using, are some examples of the built-in function. There are 68 built-in functions defined in Python 3.7.4

Examples:

Name	Description	Example
max(x, y, z,)	It returns the largest of its arguments: where x, y and z are numeric variable/expression.	>>>max(80, 100, 1000) 1000 >>>max(-80, -20, -10) -1
min(x, y, z,)	It returns the smallest of its arguments; where x, y, and z are numeric variable/expression.	>>> min(80, 100, 1000) 80 >>> min(-80, -20, -10) -80
cmp(x, y)	It returns the sign of the difference of two numbers: -1 if x < y, 0 if x == y, or 1 if x > y, where x and y are numeric variable/expression.	>>>cmp(80, 100) -1 >>>cmp(180, 100) 1

len (s)	Return the length (the number of items) of an object. The argument may be a sequence (string, tuple or list) or a mapping (dictionary).	<pre>>>> a= [1,2,3] >>>len (a) 3 >>> b= "Hello" >>> len (b) 5</pre>
range (start, stop[, step])	This is a versatile function to create lists containing arithmetic progressions.	<pre>>>> range(10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] >>> range(1, 11) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] >>> range(0, 30, 5) [0, 5, 10, 15, 20, 25] >>> range(0, 10, 3) [0, 3, 6, 9]</pre>

User-defined functions

Functions that we define ourselves to do certain specific task are referred as user-defined functions.

Functions that readily come with Python are called built-in functions. If we use functions written by others in the form of library, it can be termed as library functions.

All the other functions that we write on our own fall under user-defined functions. So, our user-defined function could be a library function to someone else.

Advantages of user-defined functions

1. User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
3. Programmers working on large project can divide the workload by making different functions.

Example 22:

Write a python script to print a message using user-defined function.

```
def message( ):
```

```
    print("Iam working on it,")
```

```
print("shall let you know once I finish it off") #
```

Call the message function.

```
message( )
```

Docstring

The first string after the function header is called the docstring and is short for documentation string. It is used to explain in brief, what a function does.

Although optional, documentation is a good programming practice.

Example 23:

```
def greet(name):  
    """This function greets to  
    the person passed in as  
    parameter"""  
    print("Hello, " + name + ". Good morning!")  
print(greet.__doc_)
```

Output:

```
This function greets to  
    the person passed into the  
    name parameter
```

Function Call (Calling a Function)

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

Example 24:

```
def greet(name):  
    """This function greets to  
    the person passed in as
```

```
parameter""  
print("Hello, " + name + ". Good morning!")
```

```
greet('Paul') #call a function
```

Example 25:

This program has two functions. First we define the main function. def

```
main( ):
```

```
    print('I have a message for you.')
```

```
    message( )
```

```
    print('Goodbye!')
```

Next we define the message function.

```
def message( ):
```

```
    print("Iam working on it,")
```

```
    print("shall let you know once I finish it off") #
```

```
    Call the main function.
```

```
main( )
```

Example 26:

#Providing Function Definition

```
def sum(x,y):
```

```
    """Going to add x and y"""
```

```
    s=x+y
```

```
    print("Sum of two numbers is")
```

```
    print(s)
```

#Calling the sum Function

```
sum(10,20)
```

```
sum(20,30)
```

The return statement

The return statement is used returns a value from the function and goes back to the place from where it was called with the expression. A return statement may contain a constant, variable, expression. If return is used without anything, it will return **None**.

Example 27:

Function definition is here

```
def changeme( x ):
```

```
    return x #return statement with a argument
```

Now you can call changeme function

```
x = 5
```

```
print changeme( x )
```

Example 28:

Function definition is here

```
def changeme( x ):
```

```
    return #return statement without argument
```

Now you can call changeme function

```
x = 5
```

```
print changeme( x )
```

Void Functions:

Function performs a task, and does not contain a return statement such functions are called void functions. Void functions do not return anything. 'None' is a special type in Python which is returned after a void function ends.

Example 29:

```
def check (num):
```

```
    if (num%2==0):
```

```
        print "True"
```

```
    else:
```

```
print "False"
```

```
result = check (29)
```

```
print (result)
```

Output:

False

None

Exercise:

1. Write a Python function to find the Min of three numbers.
2. Write a Python function to find the Max of three numbers.
3. Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.
4. Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters.
5. Write a Python function that takes a number as a parameter and check the number is prime or not.
6. Write a Python function to check whether a number is perfect or not.

Flow of Execution

Execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom. Function definition does not alter the flow of execution of program, as the statement inside the function is not executed until the function is called.

On a function call, instead of going to the next statement of program, the control jumps to the body of the function; executes all statements of the function in the order from top to bottom and then comes back to the point where it left off. This remains simple, till a function does not call another function. Similarly, in the middle of a function, program might have to execute statements of the other function and so on.

Parameters and Arguments:

There can be two types of data passed in the function.

The First type of data is the data passed in the function call. This data is called arguments. Arguments can be variables and expressions.

Argument is the actual value of this variable that gets passed to function.

The second type of data is the data received in the function definition(function header). This data is called parameters. Parameters must be variable to holding coming values.

Parameter is variable in the declaration of function.

More on Defining Functions-Passing parameters/Arguments:

Python Supports following types arguments.

1. Required Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable-length or Arbitrary Arguments

1. Required Arguments: In the required arguments, the arguments are passed to a function in correct positional order. The number of arguments in the function call should exactly match with the number of parameters specified in the function definition.

Example:

<pre>def display(): print ("hello") display("hi")</pre> <p>Output: TypeError</p>	<pre>def display(x): print (x) display()</pre> <p>Output: TypeError</p>	<pre>def display(x): print (x) display("hello")</pre> <p>Output: hello</p>
--	---	--

2. Keyword Arguments: Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed.

Example:

```
def display(name,x,y):  
    print ("The name of person is", name)  
    print ("The x value is", x)  
    print ("The Y value is", y)  
display(y=20,name="RAMESH",x=40)
```

3. Default Arguments: A default argument is an argument that assumes a default value which defined in

the called function if a value is not provided in the function call for that argument.

Example:

```
def display(name,course="BTECH"):

    print ("The name of person is", name)

    print ("Course: ", course)

display("RAJESH","MBA")

display(course="MSC",name="RAJU")

display(name="RAVI")
```

4. Variable-length or Arbitrary Arguments:

Sometimes, we do not know in advance the number of arguments that will be passed into a function. Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.

In the function definition we use an asterisk (*) before the parameter name to denote this kind of argument. Here is an example.

Example

```
def display(*names):

    for i in names:

        print("Hello",name)

display("Monica","Luke","Steve","John")
```

Scope of Variables:

Scope of variable refers to the part of the program in which a variable is accessible. Some of the variables may not exist for the entire duration of the program. In which part of the program can access a variable and which part of the program a variable exists depends on how the variable has been declared.

There are two types of scope of variables

- 1) Global variables (Global Scope)
- 2) Local variables (Local Scope)

1) **Global Variables** : The variables which are defined (declared) in the main body of the program called *global variables*. Those variables can be accessed anywhere in the program.

Example

```
x=50    #global variable
```

```
def test ( ):
```

```
    print ("Inside test x is" , x)
```

```
print ("Value of x is" , x)
```

on execution the above code will produce

Inside test x is 50

Value of x is 50

Example:

```
x=50    # global variable
```

```
def test ( ):
```

```
    x+= 10
```

```
    print ("Inside test x is", x)
```

```
print ("Value of x is", x)
```

will produce

Inside test x is 60

Value of x is 60

2) **Local variables:** The variables which are defined within a function is called *local variables*. Local variables can be accessed only within the function. It exists as long as the function is executing. Function parameters behave like local variables in the function.

Example

```
X=50 #global variable
```

```
def test ( ):
```

```
    y
```

```
test()
```

```
    =
```

```
    2
```

```
    0
```

```
    #local variable print ("Value of x is", X, "y is", y)
```

```
print ("Value of x is ", X, "y is" , y)
```

On executing the code we will get

Value of x is 50; y is 20

The next print statement will produce an error, because the variable y is not accessible outside the function body.

Example

```
x=50          #global variable
def test ( ):
    x=5        #local variable
    y=2        #local variable
    print ("Value of x & y inside the function are" , x , y)

test()
print ("Value of x outside the function is" , x)
```

This code will produce following output: Value of x & y inside the function are 5 2 Value of x outside the function is 50

Difference between Global and Local variables

Global Variables	Local Variables
1. They are defined in the main body of the program.	1. They are defined within a function and is local to that function.
2. They can be accessed throughout the program.	2. They can be accessed within a function only. We cannot access the variables outside of the function.
3. Global variables are accessible to all functions in the program.	3. They are not related in any way to other variables with the same name used outside the function.

Using the Global Keyword:

Want to modify local variable as global variable inside the function then add ‘global’ keyword before the variable name. This declares the local or the inner variable of the function to have module scope.

Example:

```
var= "Good" def show():
```

```
    show()
```

```
    print ("Outside function, var1 is –", var1)
```

```
    print ("var is ", var)
```

Output:

In function var is – Good

Outside function, var1 is – Morning var

is – Good

```
global var1
```

```
var1= "Morning"
```

```
print ("In function var is –", var)
```