UNIT-4 Python Programming

Background and Context

"Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language."

—Peter Norvig, Director of Search Quality at Google, Inc.

You can understand and appreciate the importance of Python as a language and how much it is in demand globally. If you are well equipped in Python, a host of opportunities open up for you and employability increases significantly. For designing and implementing Artificial Intelligence and Machine Learning projects, Python programming is a critical skill and you must master it.

We will start with the basics of Python language and explore how it can be implemented in AI/ML projects.

Programming Language is a language to write programs. It is a set of instructions that produces various outputs.

A **high-level language (HLL)** is a programming language that enables a programmer to write programs that are independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from machine languages.

WHAT IS PYTHON

Python Programming Language is a high-level programming language created by Guido van Rossum in 1989. It resulted in a great general-purpose language capable of creating anything from desktop software to web applications and frameworks.

Why was Python created

In his own words, Guido revealed that he started working on it as a weekend project utilizing his free time during Christmas of December 1989. He originally wanted to create an interpreter which later turned out to be Python, gradually evolving into a full-fledged programming language.

How the name Python came about

Guido initially believed that the UNIX/C hackers were the target users of his project. Also, he was fond of watching the famous comedy series—*The Monty Python's Flying Circus.* The name Python captured his mind as not only did it appeal to his taste but also his target users.

A **general-purpose programming language** is a programming language designed to be used for writing software in the widest variety of application domains. Conversely, a domain-specific programming language is one designed to be used within a specific application domain.

PYTHON PROGRAMMING DOMAINS

- ### Web Application Development

  Python is majorly used in the field of web development, being the preferred language of many large projects. Key web application frameworks include Django, Flask, CherryPy and Bottle, which are commonly used and have a large developer community.

  These frameworks are very handy when it comes to simplifying tasks related to content management and configuration, accessing a database, and handling network protocols like HTTP, SMTP, FTP and POP.

- ### Data Science and Machine Learning

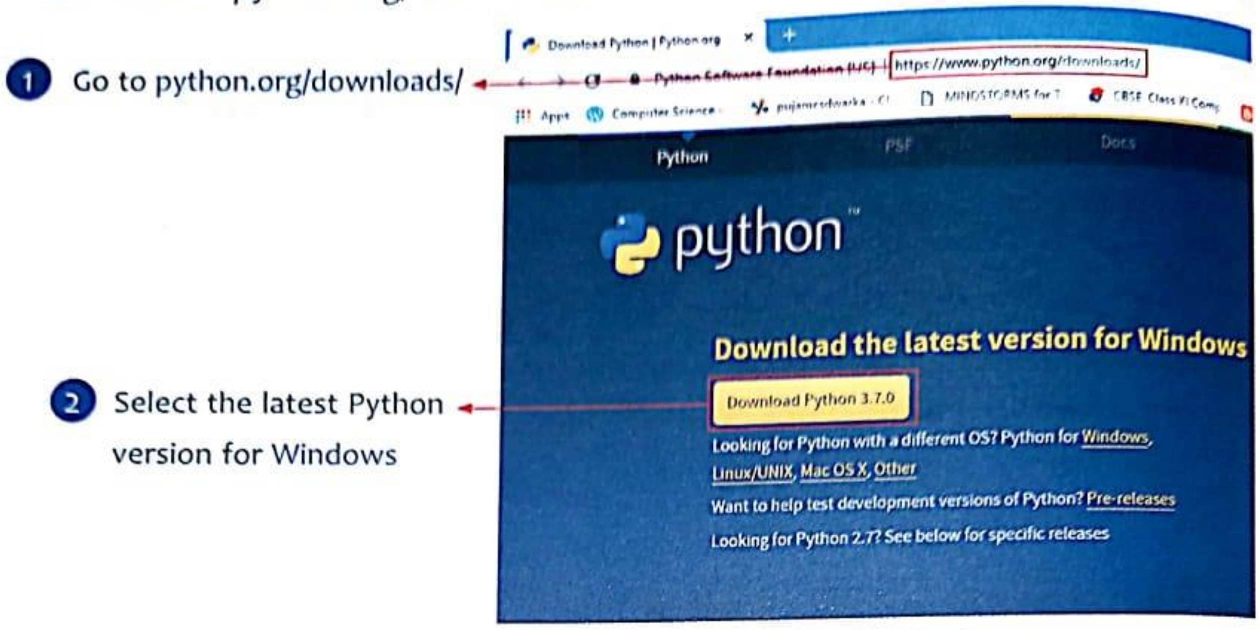
  As you have read in Chapter 1, Data Scientists are in so much demand globally that 40% vacancies are still lying vacant in AI/ML jobs. Pattern Recognition using sophisticated algorithms and making sense of data are hot skills in demand today. Python makes it extremely easy to get started with readymade libraries and large community support.

  It has tools and models for web scraping, data cleaning and standard algorithms.

PYTHON INSTALLATION

Let us now learn how to install Python in Windows OS.

1. Go to *www.python.org/downloads/*


1 Go to python.org/downloads/
2 Select the latest Python version for Windows
https://www.python.org/downloads/
python
Download Python 3.7.0

Download the latest version for Windows

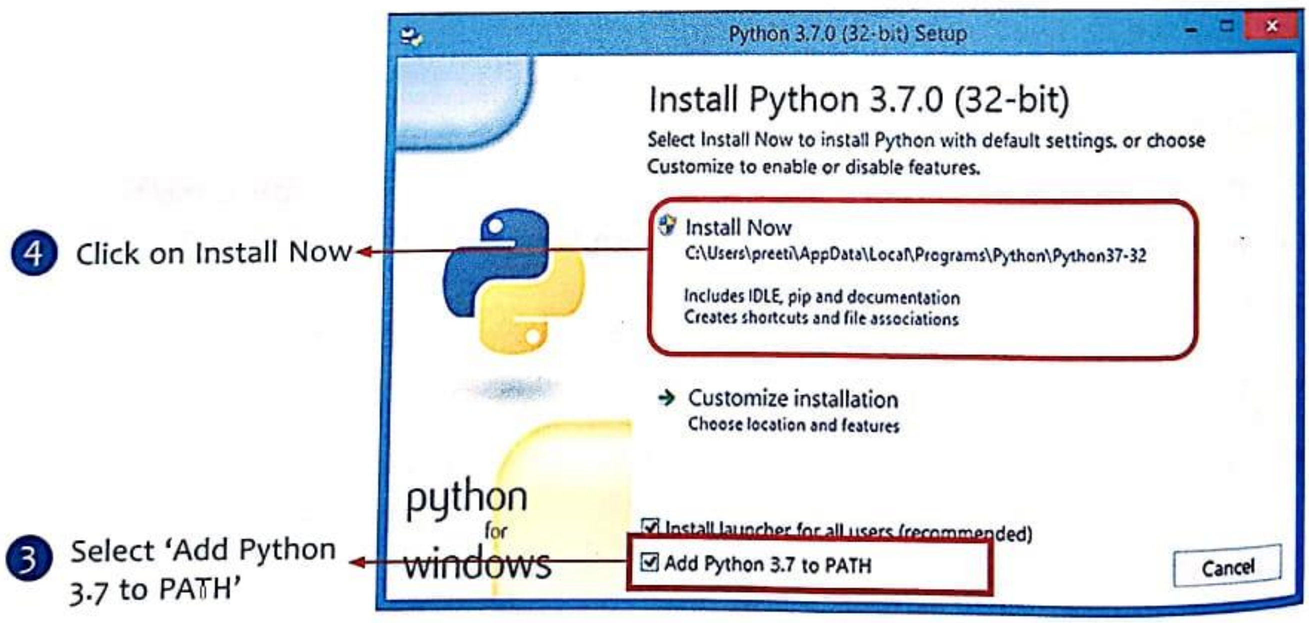
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

Looking for Python 2.7? See below for specific releases

Fig. 7.3: Python Download

2. After the application file is downloaded, we can install it by opening it. This will open the **Python 3.7.0 (32-bit)** Setup window as shown below. Put check on "Add Python 3.7 to PATH".



Python 3.7.0 (32-bit) Setup

Install Python 3.7.0 (32-bit)

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

Install Now
C:\Users\preeti\AppData\Local\Programs\Python\Python37-32

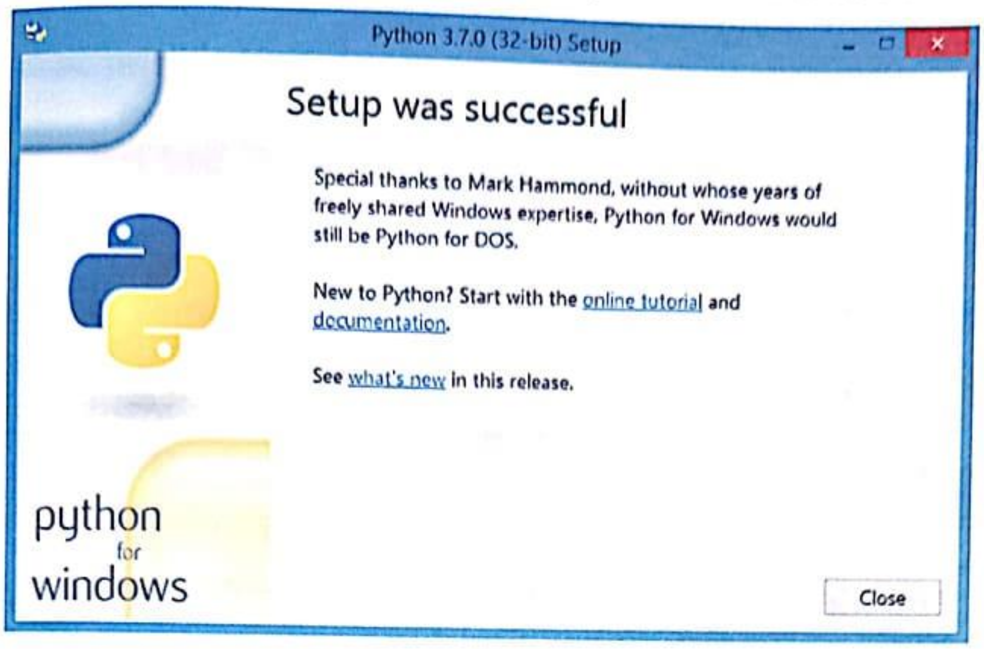
Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

Fig. 7.4: Install Python

3. When the installation is complete, you will see a screen displayed as under.



Python 3.7.0 (32-bit) Setup

Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

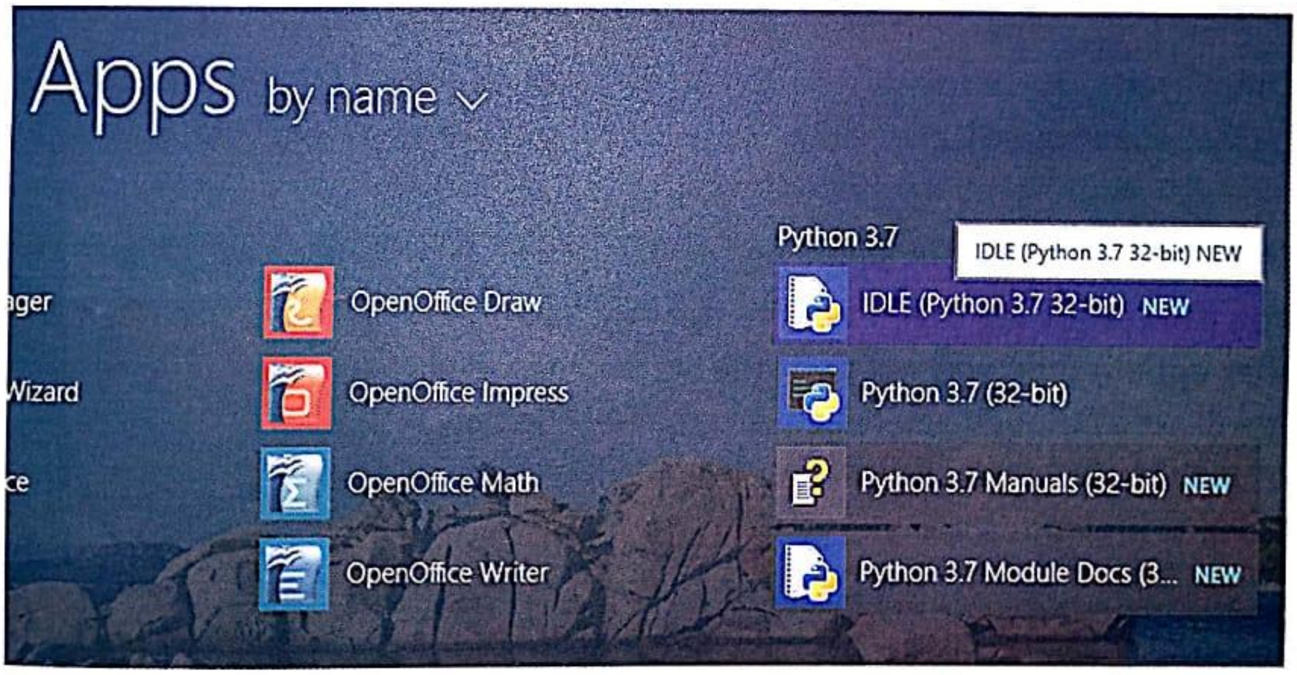
New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Fig. 7.5: Python Setup

4. Click **Close** to finish.

5. As you will see, the computer interface, IDLE (Python 3.7 32-bit), gets installed (as shown in the figure below) and is ready to use now. You can now write Python code in the newly installed Python Shell (Editor) or Script window.

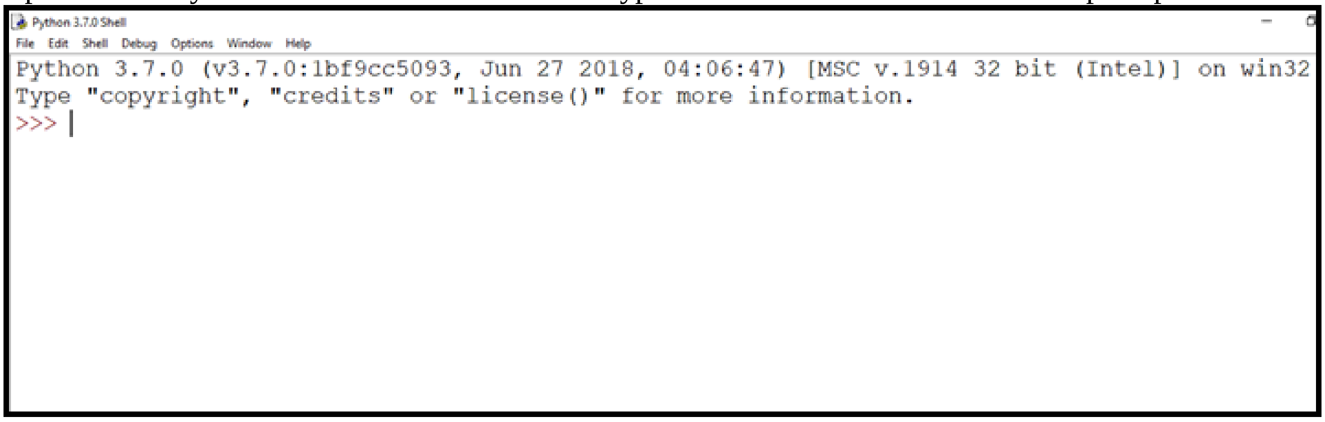


Apps by name
ager
Wizard
ce
OpenOffice Draw
OpenOffice Impress
OpenOffice Math
OpenOffice Writer
Python 3.7
IDLE (Python 3.7 32-bit) NEW
IDLE (Python 3.7 32-bit) NEW
Python 3.7 (32-bit)
Python 3.7 Manuals (32-bit) NEW
Python 3.7 Module Docs (3... NEW

Fig. 7.6: Start Python IDLE

Working with Python

To write and run (execute) a Python program, we need to have a Python interpreter installed on our computer or we can use any online Python interpreter. The interpreter is also called *Python shell*. A sample screen of Python interpreter is shown in Figure. Here, the symbol >>> is called Python prompt, which indicates that the interpreter is ready to receive instructions. We can type commands or statements on this prompt for execution.



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |
```

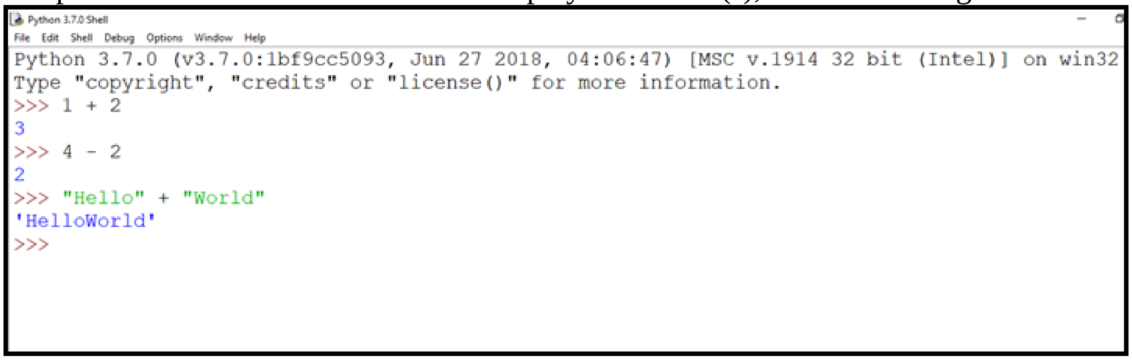
Execution Modes

There are two ways to run a program using the Python interpreter:

1. Interactive mode
2. Script mode

(A) Interactive Mode

In the interactive mode, we can type a Python statement on the >>> prompt directly. As soon as we press enter, the interpreter executes the statement and displays the result(s), as shown in Figure



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 1 + 2
3
>>> 4 - 2
2
>>> "Hello" + "World"
'HelloWorld'
>>>
```

Working in the interactive mode is convenient for testing a single line code for instant execution. But in the interactive mode, we cannot save the statements for future use and we have to retype the statements to run them again.

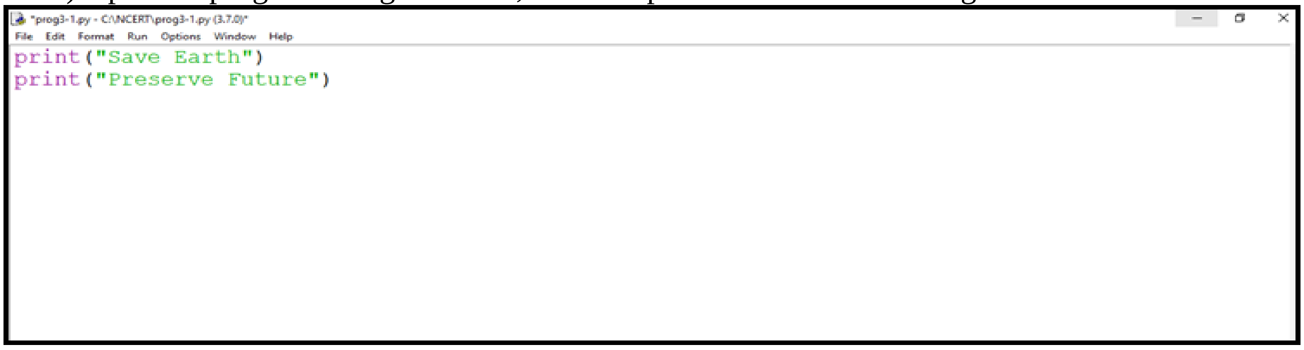
(B) Script Mode

In the script mode, we can write a Python program in a file, save it and then use the interpreter to execute the program from the file. Such program files have a .py extension and they are also known as scripts. Python has a *built-in editor* called IDLE which can be used to create programs. After opening the IDLE, we can click File>New File to create a new file, then write our program on that file and save it with a desired name. By default, the Python scripts are saved in the Python installation folder.

# IDLE : Integrated Development and Learning Environment

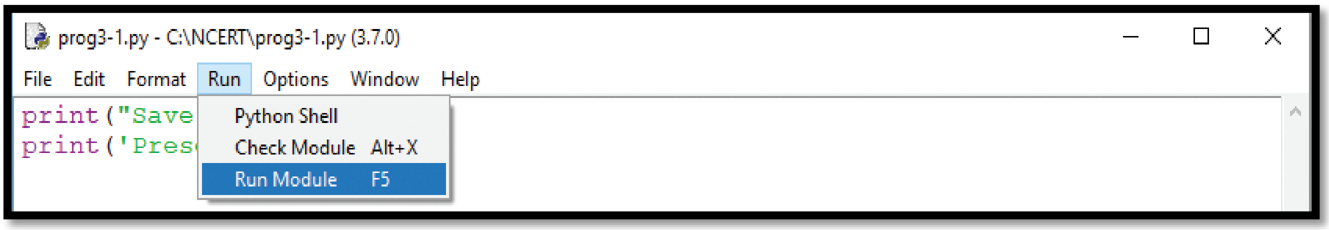
To execute a Python program in script mode,

a) Open the program using an editor, for example IDLE as shown in Figure


prog3-1.py - C:\NCERT\prog3-1.py (3.7.0)
File Edit Format Run Options Window Help

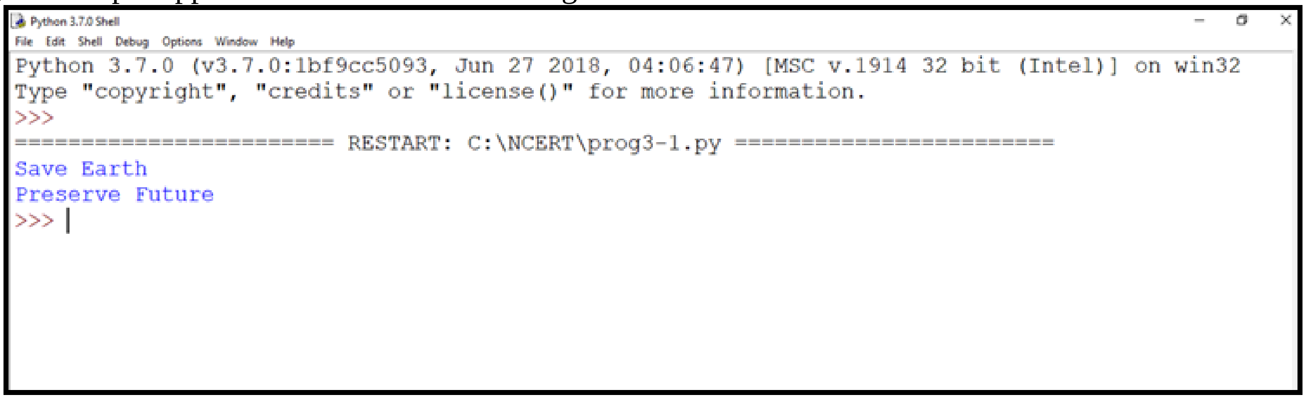
```
print("Save Earth")
print("Preserve Future")
```

b) In IDLE, go to [Run]->[Run Module] to execute the prog3-1.py as shown in Figure


prog3-1.py - C:\NCERT\prog3-1.py (3.7.0)
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5

```
print("Save
print('Pres
```

c) The output appears on shell as shown in Figure



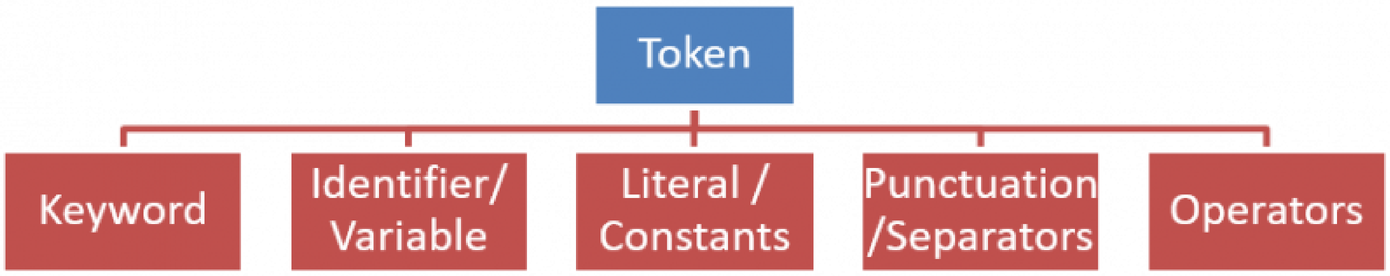
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
======================== RESTART: C:\NCERT\prog3-1.py ========================
Save Earth
Preserve Future
>>> |
```

PYTHON CHARACTER SET

Character set is a set of valid characters recognized by Python. A character represents any letter, digit or any other symbol. Python uses the traditional ASCII character set. However, the latest version recognizes the Unicode character set. The ASCII character set is a subset of the Unicode character set. Python supports the following character sets:
Letters: A-Z, a-z
Digits: 0-9
Special Symbols: space + -/**( ) { } [ ] //=l= == <>,"" , ;: % !# ? $&^ <= >= @
Whitespaces: Blank space (‘ ‘) , tabs (\t), carriage return, newline, formfeed
Other Characters: All other 256 ASCII and Unicode characters

TOKENS

A token is the smallest element of a Python script that is meaningful to the interpreter.
The following categories of tokens exist:
Identifiers, keywords, literals, operators and delimiters/punctuators.



Python Keywords

Keywords are reserved words. Each keyword has a specific meaning to the Python interpreter. As Python is case sensitive, keywords must be written exactly as given in Table

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	

Identifiers

In programming languages, *identifiers* are names used to identify a variable, function, or other entities in a program. The rules for naming an identifier in Python are as follows:

1. The name should begin with an uppercase or a lowercase alphabet or an underscore sign (_). This may be followed by any combination of characters a-z, A-Z, 0-9 or underscore (_). Thus, an identifier cannot start with a digit.
2. It can be of any length. (However, it is preferred to keep it short and meaningful).
3. It should not be a keyword or reserved word given in Table 3.1.
4. We cannot use special symbols like !, @, #, $, %, etc. in identifiers.

Variables

Variable is an identifier whose value can change. For example variable age can have different value for different person. Variable name should be unique in a program. Value of a variable can be string (for example, ‘b’, ‘Global Citizen’), number (for example 10,71,80.52) or any combination of alphanumeric (alphabets and numbers for example ‘b10’) characters. In Python, we can use an assignment statement to create new variables and assign specific values to them.

gender = 'M'
message = "Keep Smiling"
price = 987.9

Variables must always be assigned values before they are used in the program, otherwise it will lead to an error. Wherever a variable name occurs in the program, the interpreter replaces it with the value of that particular variable.

Literals/Constants

A fixed numeric or non-numeric value is called a literal. It can be defined as a number, text, or other data that represents values to be stored in variables. They are also known as constants. Python supports the following types of literals:

1. **String Literals** for example, a = "abc", b ="Independence", etc. alphanumeric/ text values
2. **Numeric Literals** for example, p = 2, a =1000, etc. integer values
3. **Floating Literals** - for example, salary = 15000.00, area =1.2,etc. decimal values
4. **Boolean Literals** – for example, value = True, value2 = False, etc.
5. **Collection Literals** -for example, list1 = [1,2,3,4,5], group = (10,20,30,40,50), etc.

Therefore, literals are data items that have fixed value.

Operators

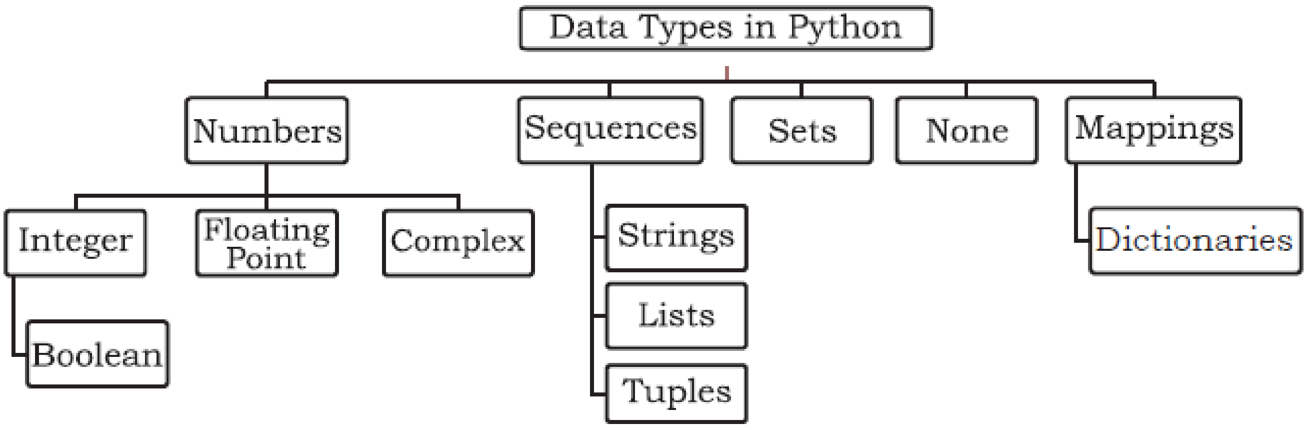
An operator is a symbol or a word that performs some kind of operation on the given values and returns the result. Examples of operators are: +,**,/, etc. Python operators have been discussed briefly later in the chapter.

Punctuators/Delimiters

Delimiters are the symbols which can be used as separators of values or to enclose some values.
Examples of delimiters are: ( ) { } [ ] , ; :
Note: # symbol used to insert comments is not a token. Any comment itself is not a token.

Data Types

Every value belongs to a specific data type in Python. Data type identifies the type of data which a variable can hold and the operations that can be performed on those data. Following figure enlists the data types available in Python.



Number

Number data type stores numerical values only. It is further classified into three different types: int, float and complex.

Numeric data types

Type/ Class	**Description**	**Examples**
int	integer numbers	-12, -3, 0, 123, 2
float	floating point numbers	-2.04, 4.0, 14.23
complex	complex numbers	3 + 4i, 2 - 2i

Boolean data type (bool) is a subtype of integer. It is a unique data type, consisting of two constants, True and False. Boolean True

Sequence

A Python sequence is an ordered collection of items, where each item is indexed by an integer value. Three types of sequence data types available in Python are Strings, Lists and Tuples. A brief introduction to these data types is as follows:

(A) String

String is a group of characters. These characters may be alphabets, digits or special characters including spaces. String values are enclosed either in single quotation marks (for example 'Hello') or in double quotation marks (for example "Hello"). The quotes are not a part of the string, they are used to mark the beginning and end of the string for the interpreter. **For example,**

```
>>> str1 = 'Hello Friend'
>>> str2 = "452"
```

We cannot perform numerical operations on strings, even when the string contains a numeric value. For example str2 is a numeric string.

(B) List

List is a sequence of items separated by commas and items are enclosed in square brackets [ ]. Note that items may be of different data types.

For example,

```
#To create a list
>>> list1 = [5, 3.4, "New Delhi", "20C", 45] #print the elements of the list list1
>>> list1
[5, 3.4, 'New Delhi', '20C', 45]
```

(C) Tuple

Tuple is a sequence of items separated by commas and items are enclosed in parenthesis ( ). This is unlike list, where values are enclosed in brackets [ ]. Once created, we cannot change items in the tuple. Similar to List, items may be of different data types.

For example,

```
#create a tuple tuple1
>>> tuple1 = (10, 20, "Apple", 3.4, 'a')
#print the elements of the tuple tuple1
>>> print(tuple1)
(10, 20, "Apple", 3.4, 'a')
```

Mapping

Mapping is an unordered data type in Python. Currently, there is only one standard mapping data type in Python called Dictionary.

Dictionary

Dictionary in Python holds data items in key-value pairs and Items are enclosed in curly brackets { }. dictionaries permit faster access to data. Every key is separated from its value using a colon (:) sign. The key value pairs of a dictionary can be accessed using the key. Keys are usually of string type and their values can be of any data type. In order to access any value in the dictionary, we have to specify its key in square brackets [ ].

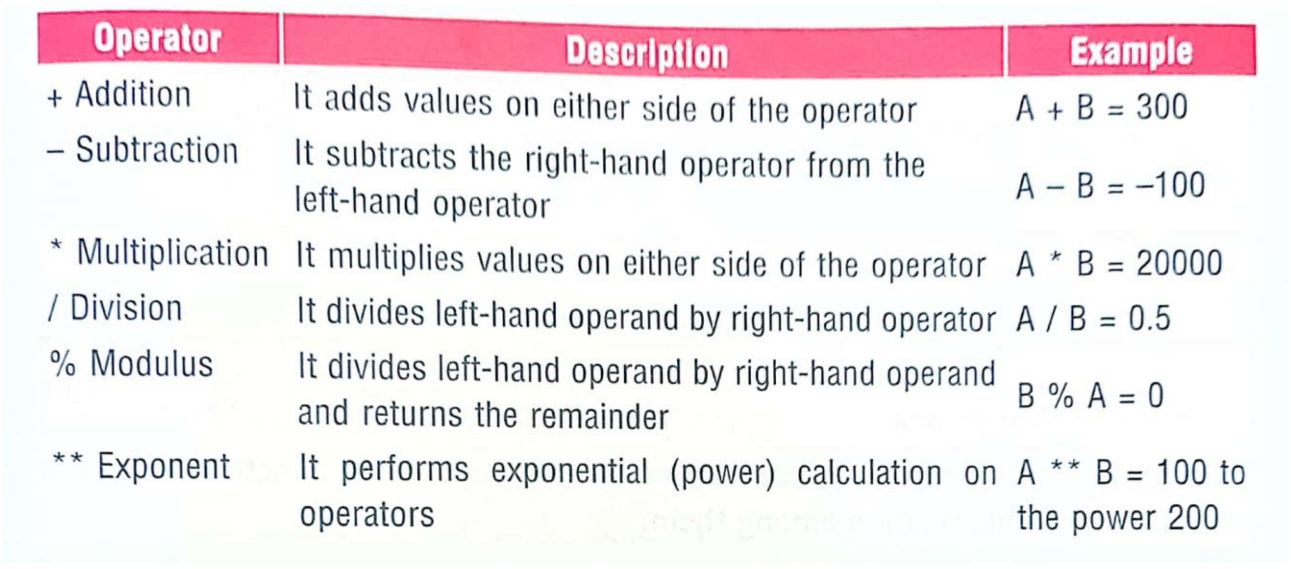
For example,

```
#create a dictionary
>>> dict1 = {'Fruit':'Apple', 'Climate':'Cold', 'Price(kg)':120}
>>> print(dict1)
{'Fruit': 'Apple', 'Climate': 'Cold', 'Price(kg)': 120}
```

Operators

An operator is used to perform specific mathematical or logical operation on values. The values that the operator works on are called *operands*. For example, in the expression 10 + num, the value 10, and the variable num are operands and the + (plus) sign is an operator.

Arithmetic Operators



Operator	Description	Example
+ Addition	It adds values on either side of the operator	A + B = 300
– Subtraction	It subtracts the right-hand operator from the left-hand operator	A – B = –100
* Multiplication	It multiplies values on either side of the operator	A * B = 20000
/ Division	It divides left-hand operand by right-hand operator	A / B = 0.5
% Modulus	It divides left-hand operand by right-hand operand and returns the remainder	B % A = 0
** Exponent	It performs exponential (power) calculation on operators	A ** B = 100 to the power 200

Comparison Operators/ RELATIONAL OPERATORS

These operators compare the values on either side of their function and then decide the possible relation among them.

Assume A = 100 and B = 80.

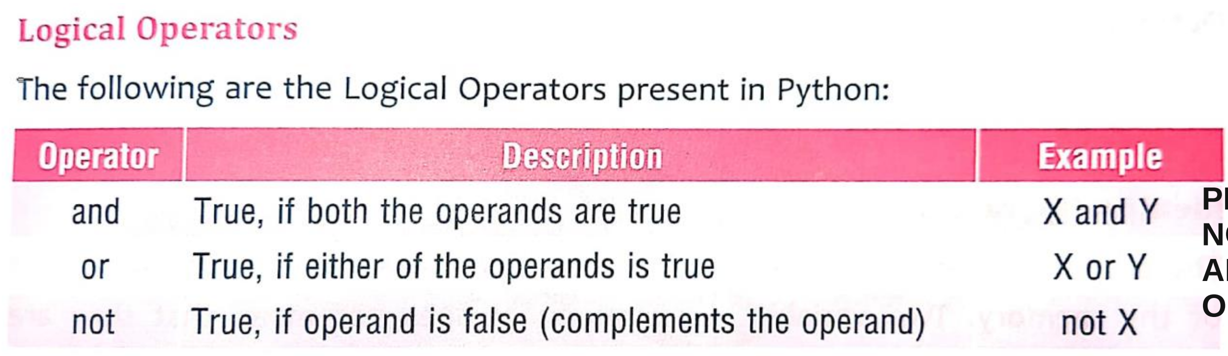
Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(A == B) is not true
!=	If values of two operands are not equal, then the condition becomes true.	(A != B) is true
>	If the value of left operand is greater than the value of right operand, then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

Assignment Operators

An Assignment Operator is the operator used to assign a new value to a variable.

Assume A = 100 and B = 80 for the below table.

Operator	Description	Example
=	Assigns values from right-side operands to left-side operand	C = A + B assigns value of A + B to C
+= Add AND	It adds right operand to the left operand and assigns the result to left operand	C += A is equivalent to C = C + A
−= Subtract AND	It subtracts right operand from the left operand and assigns the result to left operand	C −= A is equivalent to C = C − A
*= Multiply AND	It multiplies right operand with the left operand and assigns the result to left operand	C *= A is equivalent to C = C * A
/= Divide AND	It divides left operand by right operand and assigns the result to left operand	C /= A is equivalent to C = C / A
%= Modulus AND	It takes modulus using two operands and assigns the result to left operand	C %= A is equivalent to C = C % A
**= Exponent AND	It performs exponential (power) calculation on operators and assigns value to the left operand	C **= A is equivalent to C = C ** A

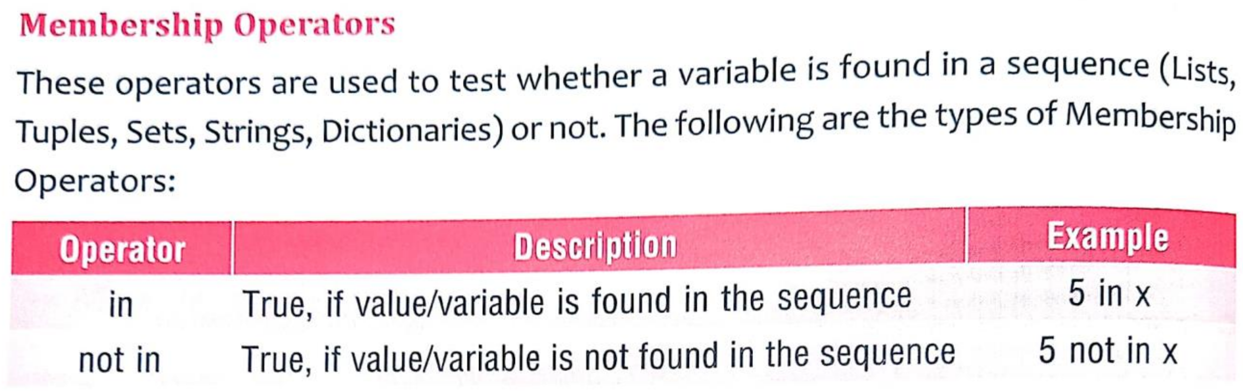


Logical Operators

The following are the Logical Operators present in Python:

Operator	Description	Example
and	True, if both the operands are true	X and Y
or	True, if either of the operands is true	X or Y
not	True, if operand is false (complements the operand)	not X

PRECENDENCE
NOT
AND
OR



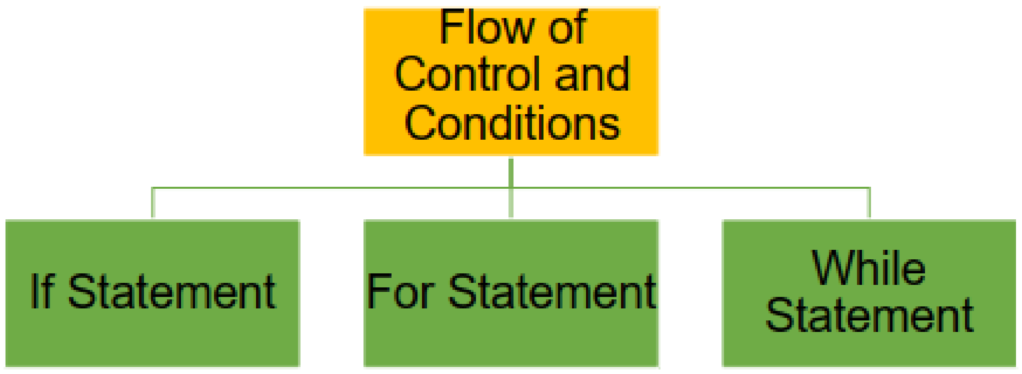
Membership Operators

These operators are used to test whether a variable is found in a sequence (Lists, Tuples, Sets, Strings, Dictionaries) or not. The following are the types of Membership Operators:

Operator	Description	Example
in	True, if value/variable is found in the sequence	5 in x
not in	True, if value/variable is not found in the sequence	5 not in x

Flow of Control and Conditions

In the programs we have seen till now, there has always been a series of statements faithfully executed by Python in exact top-down order. What if you wanted to change the flow of how it works? For example, you want the program to take some decisions and do different things depending on different situations, such as printing 'Good Morning' or 'Good Evening' depending on the time of the day?
As you might have guessed, this is achieved using control flow statements. There are three control flow statements in Python - if, for and while.



If Statement (SELECTION STATEMENTS)

On the occasion of World Health Day, one of the schools in the city decided to take an initiative to help students maintain their health and be fit. Let's observe an interesting conversation happening between the students when they come to know about the initiative.

Hey, guys did you hear the amazing news ?
No!! What is the news ?
From World Health Day onwards , Each students will have to run extra rounds after the morning assembly, on the basis of his/her BMI.

Mine is 27 and I will not have to run any rounds. What about you?
That's great !
My BMI is 20, But I don't know how many rounds ill have to run. ?
I heard the news too! Mine is 31 and Ill have to run 2 rounds daily.

There come situations in real life when we need to make some decisions and based on these decisions, we need to decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code.
Decision making statements in programming languages decide the direction of flow of program execution.
Decision making statements available in Python are:
- if statement
- if..else statements
- if-elif ladder

If Statement

The if statement is used to check a condition: *if* the condition is true, we run a block of statements (called the *if-block*).
Syntax:

```
if test expression:
statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.
If the text expression is False, the statement(s) is not executed.

Note:
1) In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.
2) Python interprets non-zero values as True. None and 0 are interpreted as False.

Python if Statement Flowchart

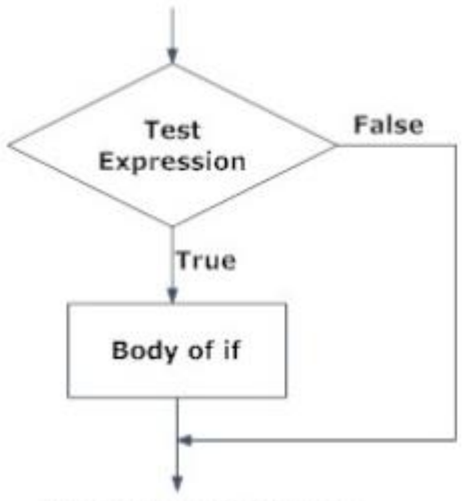


Fig: Operation of if statement

Example :

```
#Check if the number is positive, we print an appropriate message
num = 3
if num > 0:
        print(num, “is a positive number.”)
print(“this is always printed”)
num = -1
if num > 0:
        print(num, “is a positive number.”)
print(“this is always printed”)
```

When you run the program, the output will be:

```
3 is a positive number
This is always printed
This is also always printed.
```

In the above example, num > 0 is the test expression. The body of if is executed only if this evaluates to True.
When variable num is equal to 3, test expression is true and body inside body of if is executed.
If variable num is equal to -1, test expression is false and body inside body of if is skipped.
The print() statement falls outside of the if block (unindented). Hence, it is executed regardless of the test expression.

Python if...else Statement

Syntax of if...else

```
if test expression:
Body of if
else:
Body of else
```

The if..else statement evaluates test expression and will execute body of if only when test condition is True. If the condition is False, body of else is executed. Indentation is used to separate the blocks.

Python if..else Flowchart

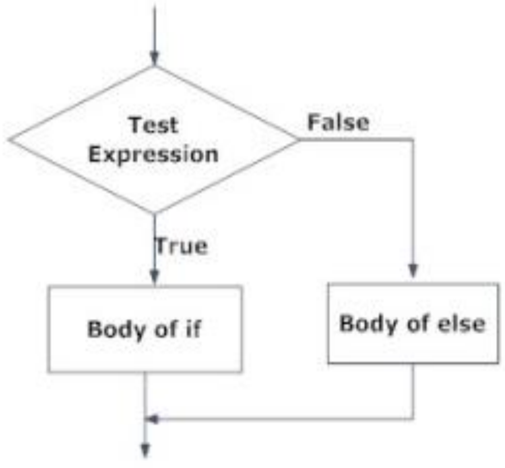


Fig: Operation of if...else statement

Example of if...else

```
#A program to check if a person can vote
age = input("Enter Your Age")
if age >= 18:
print("You are eligible to vote")
else:
print("You are not eligible to vote")
```

In the above example, when if the age entered by the person is greater than or equal to 18, he/she can vote. Otherwise, the person is not eligible to vote.

Python if...elif...else Statement

Syntax of if...elif...else

```
if test expression:
Body of if
elif test expression:
Body of elif
else:
Body of else
```

The elif is short for else if. It allows us to check for multiple expressions.
If the condition for if is False, it checks the condition of the next elif block and so on.
If all the conditions are False, body of else is executed.

Only one block among the several if...elif...else blocks is executed according to the condition.
The if block can have only one else block. But it can have multiple elif blocks.

Flowchart of if...elif...else

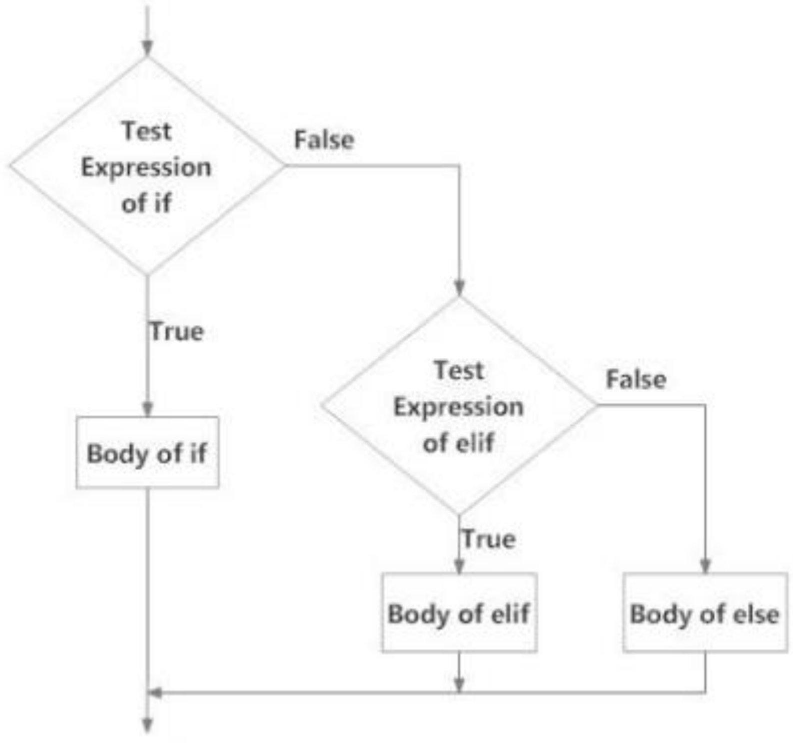


Fig: Operation of if...elif...else statement

Example of if...elif...else

```
#To check the grade of a student
Marks = 60
if marks > 75:
        print("You get an A grade")
elif marks > 60:
        print("You get a B grade")
else:
        print("You get a C grade")
```

Python Nested if statements

We can have an if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.
Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if it can be.

Python Nested if Example

```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message
# This time we use nested if
num = float(input("Enter a number: "))
if num >= 0:
        if num == 0:
                print("Zero")
        else:
                print("Positive number")
else:
        print("Negative number")
```

When you run the above program
Output 1

```
Enter a number: 5
Positive number
```

Output 2

```
Enter a number: -1
Negative number
```

Output 3

```
Enter a number: 0
Zero
```

LOOPING STATEMENTS

The For Loop

The for..in statement is another looping statement which *iterates* over a sequence of objects i.e. go through each item in a sequence. A sequence is just an ordered collection of items.

Syntax of for Loop

```
for val in sequence:
            Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.
Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Flowchart of for Loop

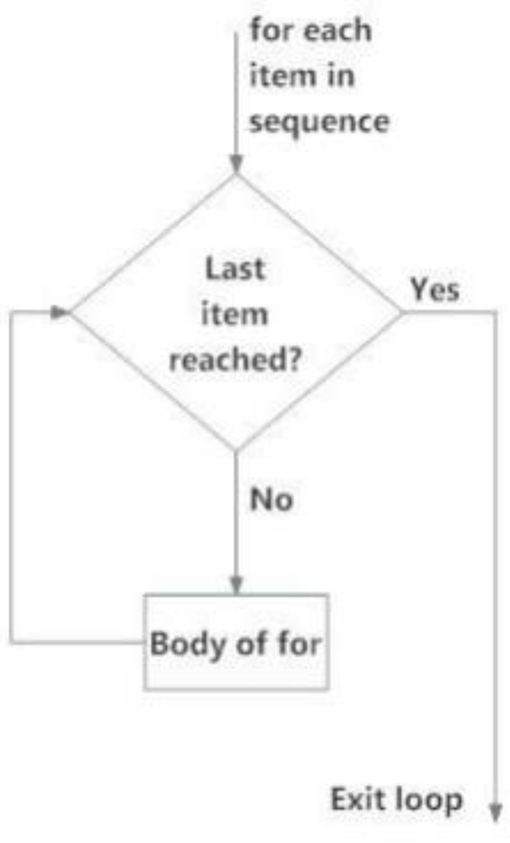


Fig: operation of for loop

Example: Python for Loop

```
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]


# variable to store the sum
sum = 0


# iterate over the list
for val in numbers:
        sum = sum+val


# Output: The sum is 48
print("The sum is", sum)
```

when you run the program, the output will be:

```
The sum is 48
```

The while Statement

The while statement allows you to repeatedly execute a block of statements as long as a condition is true. A while statement is an example of what is called a *looping* statement. A while statement can have an optional else clause.

Syntax of while Loop in Python

```
while test_expression:
        Body of while
```

In while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False. In Python, the body of the while loop is determined through indentation. Body starts with indentation and the first unindented line marks the end. Python interprets any non-zero value as True. None and 0 are interpreted as False.

Flowchart of while Loop

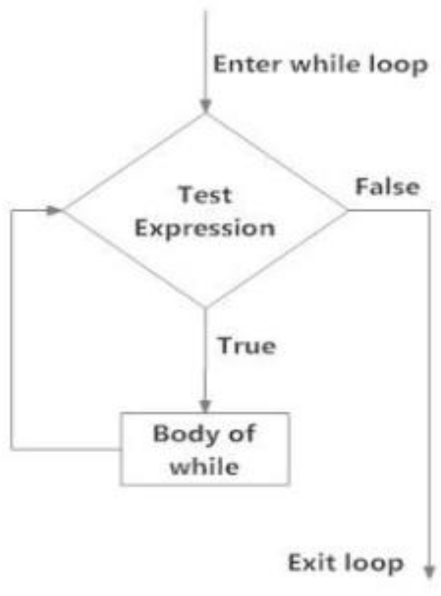


Fig: operation of while loop

Example: Python while Loop

```
# Program to add natural
# numbers upto
# sum = 1+2+3+...+n
# To take input from the user,
# n = int(input("Enter n: "))
n = 10
# initialize sum and counter
sum = 0
i = 1
while i <= n:
sum = sum + i
i = i+1 # update counter
# print the sum
print("The sum is", sum)
```

When you run the program, the output will be:

```
Enter n: 10
The sum is 55
```

In the above program, the test expression will be True as long as our counter variable i is less than or equal to n (10 in our program).

We need to increase the value of the counter variable in the body of the loop. This is very important (and mostly forgotten). Failing to do so will result in an infinite loop (never ending loop).

Finally, the result is displayed.