

## CHAPTER-06

# MORE ON PYTHON

### Introduction to Python

In the previous section, you learnt about the different methodologies for programming. A programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output.

In simple Words, a programming language is a vocabulary and set of grammatical rules for instructing a computer to perform specific tasks. Though there are many different programming languages such as BASIC, Pascal, C, C++, Java, Haskell, Ruby, Python, etc. we will study Python in this course.

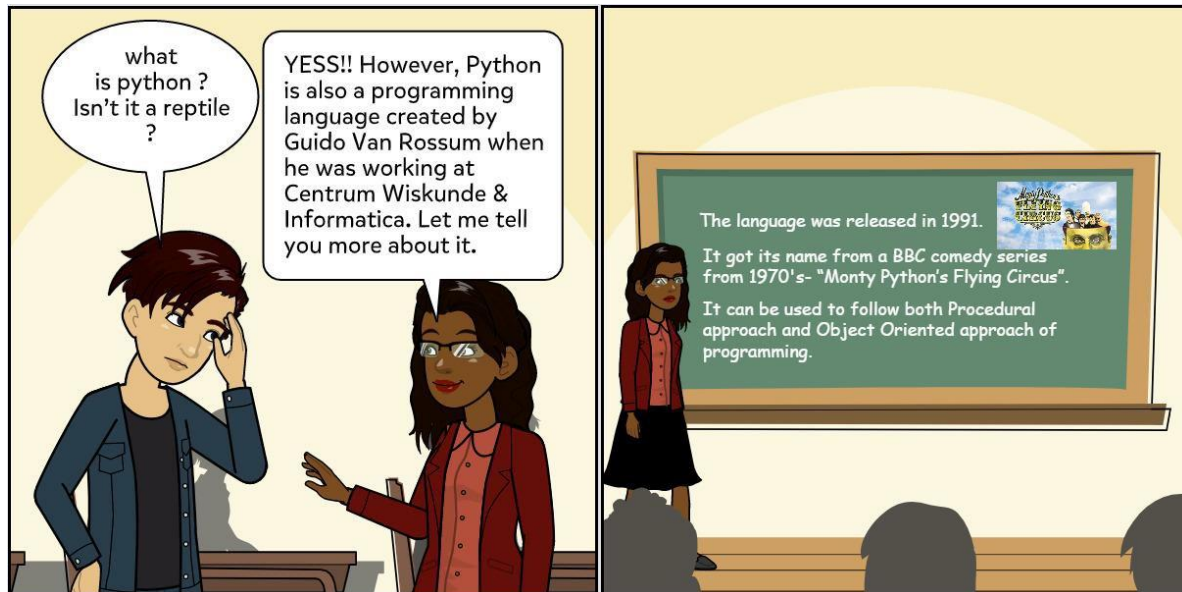
*Changing your Tomorrow*

### *What is a program?*

A computer program is a collection of instructions that perform a specific task when executed by a computer. It is usually written by a computer program in a programming language.

Before getting into understanding more about Python, we need to first understand what is Python and why we need to use Python?

### What is Python?



### Getting started with Python

Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows, MacOS, Linux and has even been ported to the Java and .NET virtual machines.

To write and run Python program, we need to have Python interpreter installed in our computer.

#### *Downloading and Setting up Python for use*

- Download Python from **python.org** using link **python.org/downloads**
- Select appropriate download link as per Operating System  
[Windows 32Bit/64 Bit, Apple iOS]
- FOR EXAMPLE, for Windows 64 Bit OS
- Select the following link

▪ [Download Windows x86-64 executable installer](#)

#### **Run in the Integrated Development Environment (IDE)**

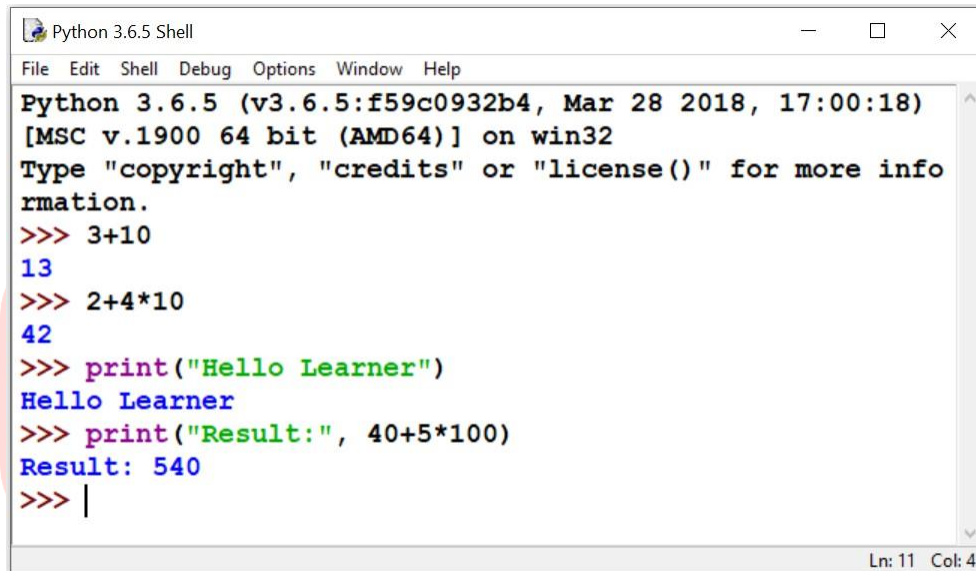
When we install Python, an IDE named **IDLE** is also installed. We can use it to run Python on our computer.

IDLE (GUI integrated) is the standard, most popular Python development environment. IDLE is an acronym of Integrated Development Environment. It lets one edit, run, browse and debug Python Programs from a single interface. This environment makes it easy to write programs.

Python shell can be used in two ways, viz., interactive mode and script mode. Where Interactive Mode, as the name suggests, allows us to interact with OS; script mode lets us create and edit Python source file.

### *Interactive Mode*

ODM EDUCATIONAL GROUP  
Changing your Tomorrow



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18)
[MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more info
rmation.
>>> 3+10
13
>>> 2+4*10
42
>>> print("Hello Learner")
Hello Learner
>>> print("Result:", 40+5*100)
Result: 540
>>> |
```

## EDUCATIONAL GROUP

You can see the above example, Python IDLE Shell account has `>>>` as Python prompt, where simple mathematical expressions and single line Python commands can be written and can be executed simply by pressing enter.

The first expression `3+10` written on the first Python prompt shows 13 as output in the next line.

The second expression  $2+4*10$  written on the second Python prompt shows 42 as output in the next line.

The third statement `print("Hello Learner")` written on the third Python prompt shows Hello Learner as output in the next line.

The third statement `print("Result:", 40+5*100)` written on the fourth Python prompt shows Result: as output in the next line.

Try Yourself

1. Find the result of  $(75+85+65)/3$  i.e. the average of three marks
2. Find the result of  $22/7 * 5 * 5$  i.e. the area of circle having radius as 5
3. Find the result of "RAVI"+"Kant"
4. Find the result of "###" \* 3

*Script Mode*

*Changing your Tomorrow*

In script mode, we type Python program in a file and then use the interpreter to execute the content from the file. Working in interactive mode is convenient for beginners and for testing small pieces of code, as we can test them immediately. But for coding more than few lines, we

should always save our code so that we may modify and reuse the code.

**Note:** Result produced by Interpreter in both the modes, viz., Interactive and script mode is exactly the same.

**Python Script/Program:** Python statements written in a particular sequence to solve a problem is known as Python Script/Program.

Try Yourself

5. Find the result of  $(75+85+65)/3$  i.e. the average of three marks
6. Find the result of  $22/7 * 5 * 5$  i.e. the area of circle having radius as 5
7. Find the result of "RAVI"+"Kant"
8. Find the result of "###" \* 3

*Script Mode*

In script mode, we type Python program in a file and then use the interpreter to execute the content from the file. Working in interactive

mode is convenient for beginners and for testing small pieces of code, as we can test them immediately. But for coding more than few lines, we should always save our code so that we may modify and reuse the code.

**Note:** Result produced by Interpreter in both the modes, viz., Interactive and script mode is exactly the same.

**Python Script/Program:** Python statements written in a particular sequence to solve a problem is known as Python Script/Program.

To write a Python script/program, we need to open a new file - **File >> New File**, type a sequence of Python statements for solving a problem, save it with a meaningful name - **File**

>> **Save**, and finally **Run** the program to view the output of the program.

Code Explanation:



Line 1 in the above code starting with # is a comment line, which means the line is non-executable and it is only for the programmer's reference.

Line 2 will simply display

Hello

Line 3 will assign a string value "Sam" to a variable **Name**

Line 4 will display **Learner** and will allow the next output to get

displayed in the same lineLine 5 will display **Sam** in the same line as

**Learner**

Line 6 will assign an integer **50** to a

variable **A** Line 7 will assign an

integer **300** to a variable **B** Line 8

will display **50** times **300** is **15000**

So, the complete output of the Python Code will be

```
Hello  
Learner Sam
```

### Python Statement and Comments

In this section we will learn about Python statements, why indentation is important and how to use comments in programming.

#### *Python Statement* *Changing your Tomorrow*

Instructions written in the source code for execution are called statements. There are different types of statements in the Python programming language like Assignment statement, Conditional statement, Looping statements etc. These help the user to get the required output. For example, `n = 50` is an assignment statement.

### Multi-line statement

In Python, end of a statement is marked by a newline character.

However, Statements in Python can be extended to one or more lines using parentheses (), braces {}, square brackets [], semi-colon (;), continuation character slash (\). When we need to do long calculations and cannot fit these statements into one line, we can make use of these characters.

Examples:

Type of Multi-line Statement	Usage
Using Continuation Character (/)	<pre>s = 1 + 2 + 3 + \     4 + 5 + 6 + \     7 + 8 + 9</pre>
Using Parentheses ()	<pre>n = (1 * 2 * 3 + 4 - 5)</pre>
Using Square Brackets []	<pre>footballer = ['MESSI',               'NEYMAR',               'SUAREZ']</pre>

Using braces {}	<code>x = {1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9}</code>
Using Semicolons ( ; )	<code>flag = 2; ropes = 3; pole = 4</code>

### Python Comments

A **comment** is text that doesn't affect the outcome of a code, it is just a piece of text to let someone know what you have done in a program or what is being done in a block of code.

In Python, we use the hash (#) symbol to start writing a comment.



### Single line Comments

#Defining a variable to store number.

```
a = 10
```

```
fb = "messi" #Store Messi as value in fb
```

### Multi Line Comments

```
"""
```

```
This is an example code.
```

```
This is to learn Python for AI
```

```
"""
```

```
'''
```

```
This is an example code.
```

```
This is to learn Python for AI
```

```
'''
```

EDUCATIONAL GROUP

*Python Keywords and Identifiers*

*Changing your Tomorrow*

In this section, we will learn about keywords (reserved words in Python) and identifiers (names given to variables, functions, etc.).

Keywords



Keywords are the reserved words in Python used by Python interpreter to recognize the structure of the program.

### 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	

— An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

E

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 \_.

An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.

Keywords cannot be used as identifiers.

We cannot use special symbols like !, @, #, \$, % etc. in our identifier.

Identifier can be of any length.

Note:

Python is a case-sensitive language. This means, `Variable` and `variable` are not the same. Always name identifiers that make sense.

While, `c = 10` is valid. Writing `count = 10` would make more sense and it would be easier to figure out what it does even when you look at your code after a long gap.

Multiple words can be separated using an underscore, for example `this_is_a_long_variable`

*Variables and Datatypes*

*Variables*  *Changing your Tomorrow* 

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. For example,



```
x = 42
```

```
y = 42
```

### *Constants:*

A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.

Non technically, you can think of constant as a shoe box with a fixed size of shoe kept inside which cannot be changed after that.

### *Assigning Value to a constant in Python*

In Python, constants are usually declared and assigned on a module. Here, the module means a new file containing variables, functions etc. which is imported to the main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Example : Declaring and assigning value to a constant

- Create a info.py

```
NAME = "Ajay"  
AGE = 24
```

### Rules and Naming convention for variables and constants

Create a name that makes sense.

Suppose, vowel makes more sense than v.

Use camelCase notation to declare a variable. It starts with lowercase letter. For example: myName

Use capital letters where possible to declare a constant. For example: PI

Never use special symbols like !, @, #, \$, %, etc.

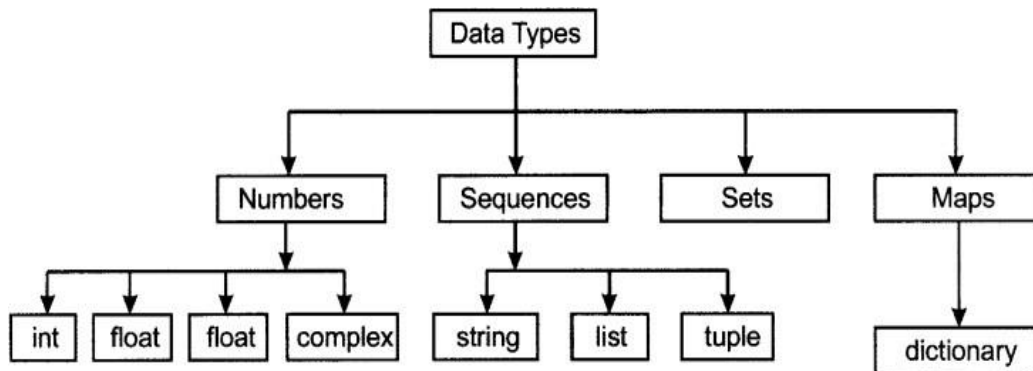
Constant and variable names should have combination of letters in lowercase or uppercase or digits or an underscore (\_).

## Datatypes

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are mentioned below in the image

ODM  
EDUCATIONAL GROUP  
*Changing your Tomorrow*



Python Data types

Number data type stores Numerical Values. These are of three different types:

- a) Integer & Long
- b) Float / floating point

#### *Integer & Long Integer*

Range of an integer in Python can be from -2147483648 to 2147483647, and longinteger has unlimited range subject to available memory.

Integers are the whole numbers consisting of + or – sign with decimal digits

like 100000, -99,0, 17. While writing a large integer value, don't use commas to separate digits. Also, integers should not have leading zeros.

### *Floating Point:*

Numbers with fractions or decimal point are called floating point numbers.

A floating-point number will consist of sign (+,-) sequence of decimals digits and a dot such as 0.0, -21.9, 0.98333328, 15.2963. These numbers can also be used to represent a number in engineering/ scientific notation.

$-2.0 \times 10^5$  will be represented as -2.0e5

$2.0 \times 10^{-5}$  will be 2.0E-5

#### 1) *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**.

#### 2) *Sequence*

A sequence is an ordered collection of items, indexed by positive integers. It is a combination of mutable and non-mutable data types. Three types of sequence data type available in Python are:

- a) Strings
- b) Lists
- c) Tuples

### *String*

String is an ordered sequence of letters/characters. They are enclosed in single quotes ( ' ') or double ( " "). The quotes are not part of string. They only tell the computer where the string constant begins and ends. They can have any character or sign, including space in them.

### *Lists*

List is also a sequence of values of any type. Values in the list are called elements / items. These are indexed/ordered. List is enclosed in square brackets.

## Conditional Statements In Python

In programming languages, most of the time in large projects we have to control the flow of execution of our program and we want to execute some set of statements only if the given condition is satisfied, and a different set of statements when it's not satisfied.

Conditional statements are also known as decision-making statements. We need to use these conditional statements to execute the specific block of code if the given condition is true or false.

**In Python we can achieve decision making by using the following statements:**

- E • if statements
- if-else statements
- elif statements
- • Nested if and if-else statements
- elif ladder

In this tutorial, we will discuss all the statements in detail with some real-time examples.

### #1) if statements

Python if statement is one of the most commonly used conditional statements in programming languages. It decides whether certain statements need to be executed or not. It checks for a given condition, if the condition is true, then the set of code present inside the "if" block will be executed otherwise not.

The if condition evaluates a Boolean expression and executes the block of code only when the Boolean expression becomes TRUE.

**Syntax:**

```
If ( EXPRESSION == TRUE ):
```

```
    Block of code
```

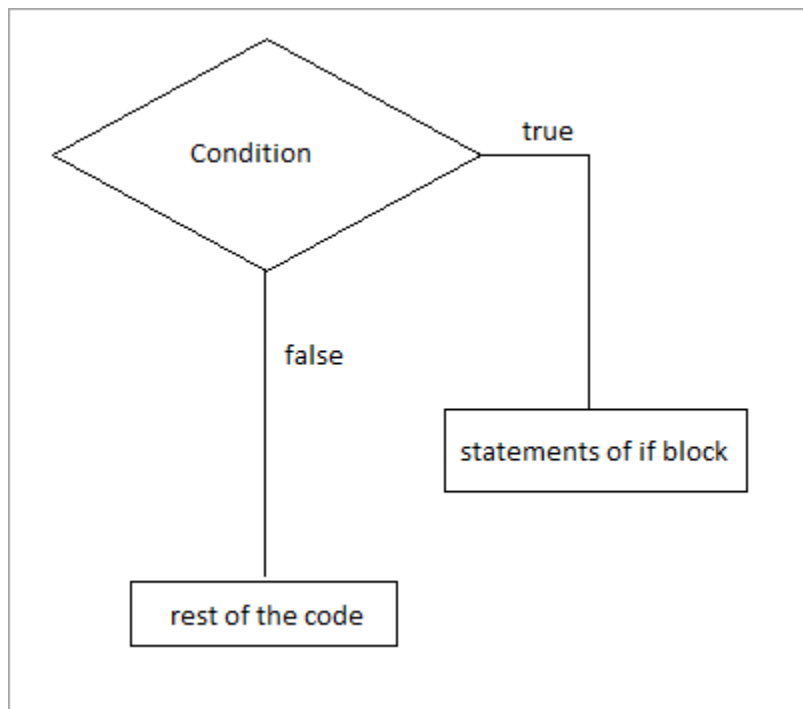
```
else:
```

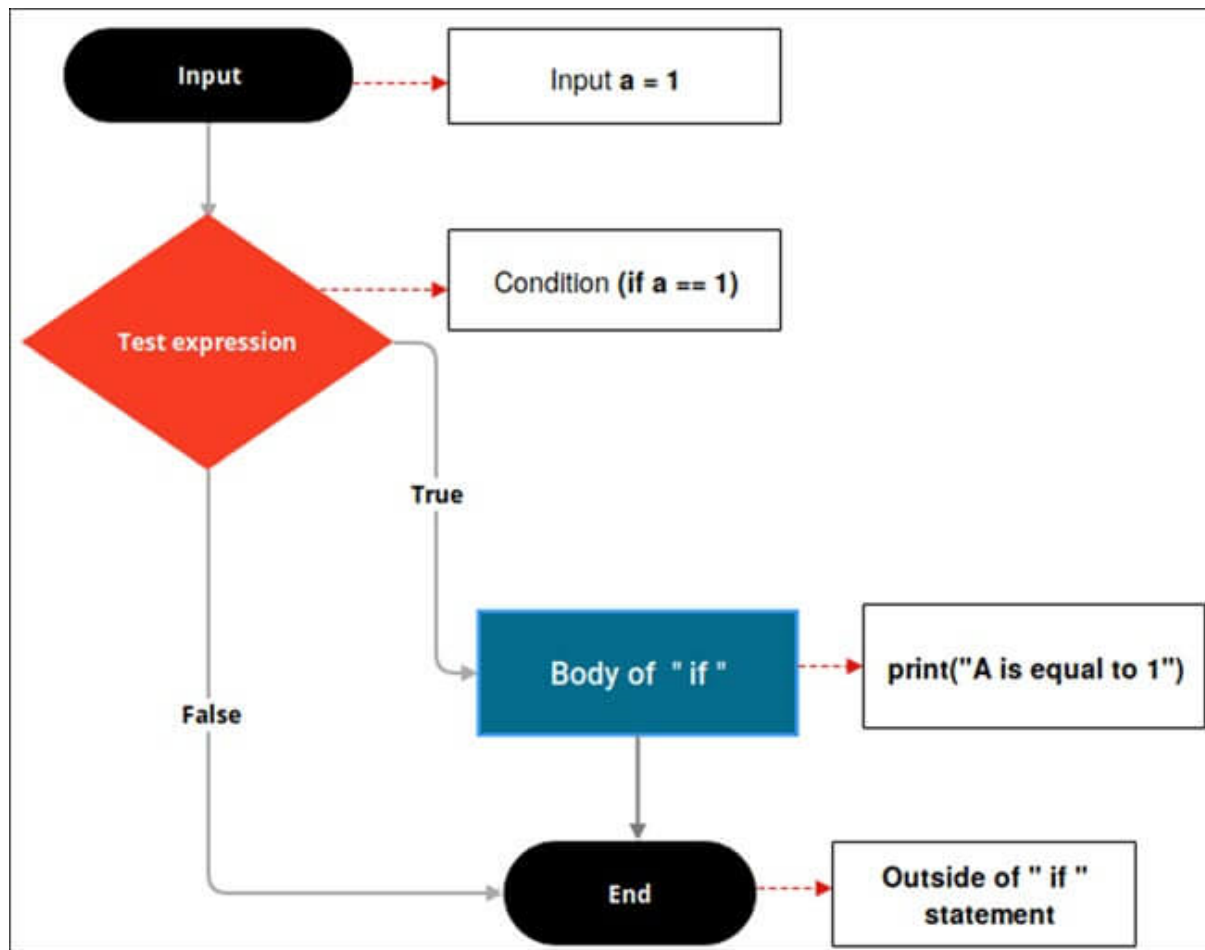
```
    Block of code
```

Here, the condition will be evaluated to a Boolean expression (true or false). If the condition is true, then the statement or program present inside the "if" block will be executed and if the condition is false, then the statements or program present inside the "else" block will be executed.



Let's see how it looks on a flow chart.





If you observe the above flow-chart, first the controller will come to an if condition and evaluate the condition if it is true, then the statements will be executed, otherwise the code present outside the block will be executed.

Let's see some examples of "if" statements.

**Example: 1**

```
num = 5
```

```
if (num < 10):
```

```
    print("Num is smaller than 10")
```

```
print("This statement will always be executed")
```

**Output:** Num is smaller than 10.

This statement will always be executed.

```
λ ~/Desktop/ python3 demo.py
Num is smaller than 10
This statement will always be executed
λ ~/Desktop/ █
```

In the above example, we declared a variable called 'Num' with the value as 5 and the "if" statement is checking whether the number is lesser than 10 or not. If the condition is true then a set of statements inside the if block will be executed.

### Example: 2

```
a = 7
```

```
b = 0
```

```
if (a > b):
```

```
    print("a is greater than b")
```

### **Output:**

```
a is greater than b
```

```
λ ~/Desktop/ python3 demo.py
a is greater than b
λ ~/Desktop/ █
```

In the above example, we are checking the relationship between a and b using the greater than (>) operator in the if condition. If “a” is greater than “b” then we will get the above output.

### Example: 3

```
a = 0
```

```
b = 7
```

```
if (b > a):
```

```
    print(“b is greater than a”)
```

### **Output:**

```
b is greater than a.
```

```
λ ~/Desktop/ python3 demo.py  
b is greater than a  
λ ~/Desktop/ █
```

**Example: 4**

```
a = 7
```

```
b = 0
```

```
if (a):
```

```
    print("true")
```

**Output:**

```
true
```

```
λ ~/Desktop/ python3 demo.py  
true  
λ ~/Desktop/ █
```

If you observe, in the above example, we are not using or evaluating any condition in the “if” statement. Always remember that in any programming language, the positive integer will be treated as true value and an integer which is less than 0 or equal to 0 will be treated as false.

Here the value of a is 7 which is positive, hence it prints true in the console output.

**Example: 5**

```
if ('Python' in ['Java', 'Python', 'C#']):
```

```
    print("true")
```

**Output:**

```
true
```

```
λ ~/Desktop/ python3 demo.py
true
λ ~/Desktop/ █
```

Here, we are verifying if the element 'Python' is present in the given list or not. Hence it prints true because " Python " is present in the given list.

**Let's take one real-life example where we will use the Python if statement.**

**For Example:** You have written an exam for a total score of 100 and if your score is above or equal to 60 then you will be considered as PASS in the exam.

Let's write the code for it.

**Example: 6**

```
passing_Score = 60
```

```
my_Score = 67
```

```
if(my_Score >= passing_Score):
```

```
    print("Congratulations! You have passed your exam")
```

**Output:**

Congratulations! You have passed your exam.



```
λ ~/Desktop/ python3 demo.py
Congratulations! You have passed your exam
λ ~/Desktop/ █
```

Remember to use the (:) operator at the end of the if statement, because whatever the code you write after the colon operator will be a part of “if block” and indentation is very important in Python.

### Example: 7

```
passing_Score = 60
```

```
my_Score = 67
```

```
if(my_Score >= passing_Score):
```

```
    print(“You passed the exam”)
```

```
print(“Congratulations!”)
```

### **Output:**

```
You passed the exam
```

Congratulations!

```
λ ~/Desktop/ python3 demo.py
You passed the exam
Congratulations!
λ ~/Desktop/ █
```

Here, `print("Congratulations!")` statement will always be executed even though the given condition is true or false.

The problem with the above code is the statement `'print("Congratulations!")'` will always be executed even if the condition is evaluated to true or false. But in real-time, if you pass the exam or if you fail in the exam, then the system will say Congratulations!!!.

In order to avoid this, Python provides one conditional statement called if-else.

## #2) if-else statements

The statement itself says if a given condition is true then execute the statements present inside the “if block” and if the condition is false then execute the “else” block.

The “else” block will execute only when the condition becomes false. It is the block where you will perform some actions when the condition is not true.

if-else statement evaluates the Boolean expression. If the condition is TRUE then, the code present in the “ if “ block will be executed otherwise the code of the “else“ block will be executed

**Syntax:**

```
If (EXPRESSION == TRUE):
```

```
    Statement (Body of the block)
```

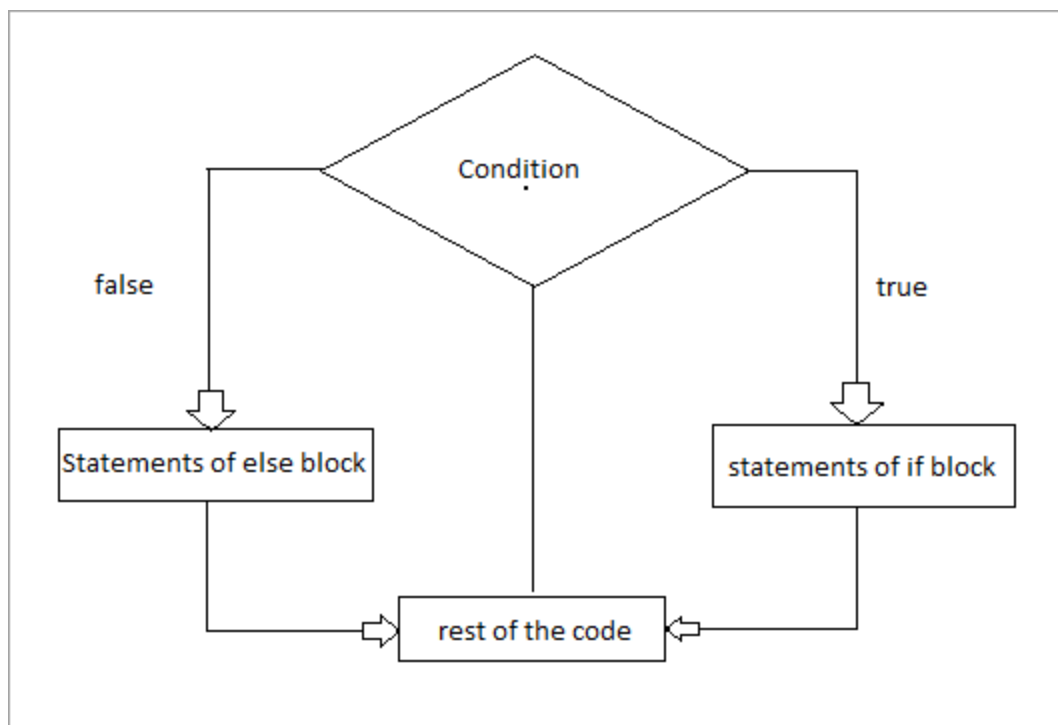
```
else:
```

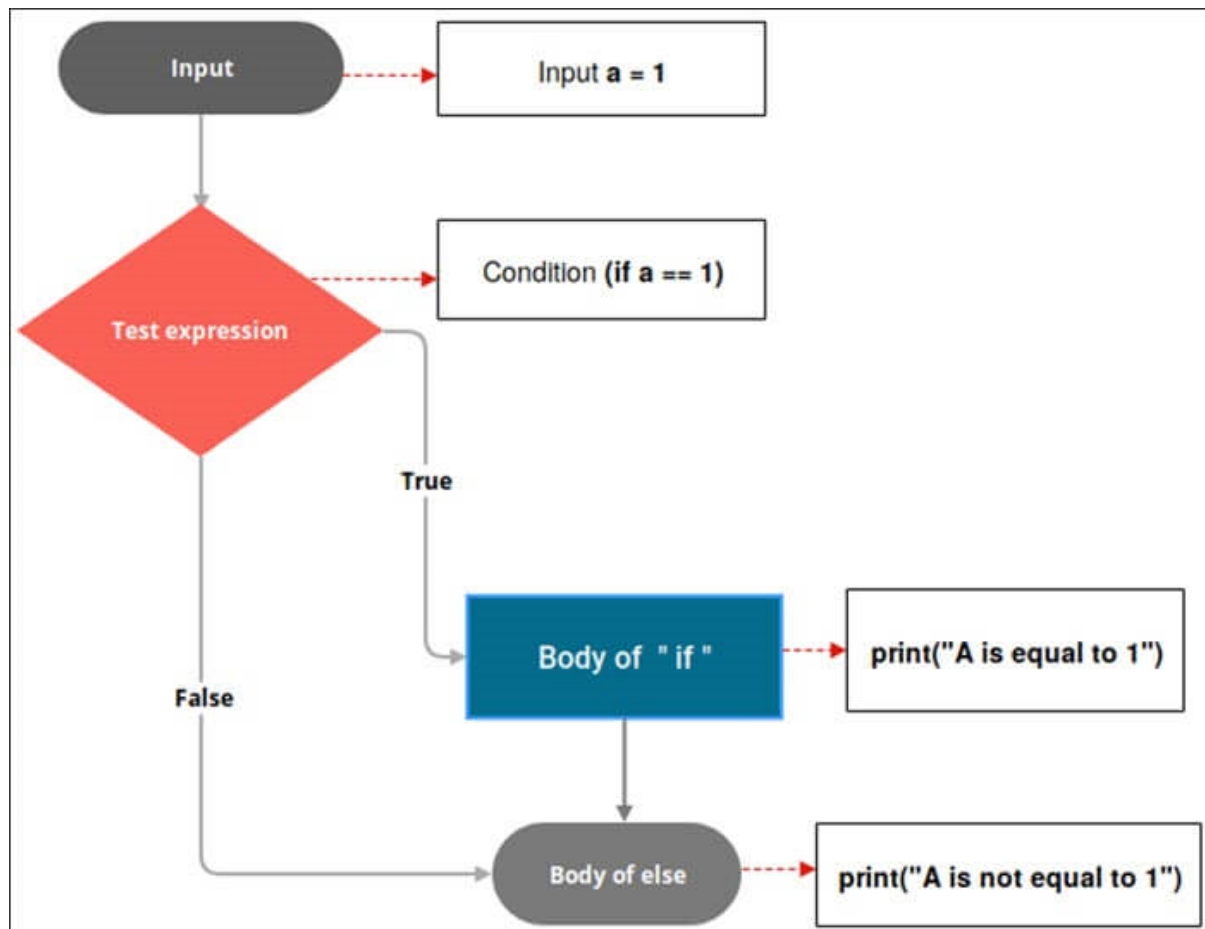
```
    Statement (Body of the block)
```

Here, the condition will be evaluated to a Boolean expression (true or false). If the condition is true then the statements or program present inside the “if” block will

be executed and if the condition is false then the statements or program present inside the “else” block will be executed.

**Let’s see the flowchart of if-else**





If you observe the above flow chart, first the controller will come to if condition and evaluate the condition if it is true and then the statements of if block will be

executed otherwise “else” block will be executed and later the rest of the code present outside “if-else” block will be executed.

**Example: 1**

```
num = 5
```

```
if(num > 10):
```

```
    print("number is greater than 10")
```

```
else:
```

```
    print("number is less than 10")
```

```
print ("This statement will always be executed" )
```

**Output:**

number is less than 10.

This statement will always be executed.

```
λ ~/Desktop/ python3 demo.py
number is less than 10
This statement will always be executed
λ ~/Desktop/ █
```

In the above example, we have declared a variable called 'num' with the value as 5 and in the "if" statement we are checking if the number is greater than 5 or not.

If the number is greater than 5 then, the block of code inside the "if" block will be executed and if the condition fails then the block of code present inside the "else" block will be executed.

### **Example: 2**

```
a = 7
```

```
b = 0
```

```
if (a > b):
```

```
    print("a is greater than b")
```

```
else:
```

```
print("b is greater than a")
```

**Output:**

a is greater than b

```
λ ~/Desktop/ python3 demo.py
a is greater than b
λ ~/Desktop/ █
```

In the above code if “a” is greater than “b” then the statements present inside the “if” block will be executed and the statements present inside the “else” block will be skipped.

EDUCATIONAL GROUP  
Changing your Tomorrow