Python Fundamentals

YOUR FIRST PROGRAM

Hi, I'm so excited to see you back here. I appreciate your decision to continue learning the Python language. In this module you will learn the fundamentals of Python language with hands-on practices. Let's look at the agenda of this module.

The first program will be covered in this module, along with some basic python language fundamentals such as comments & its types.

You will learn all these topics with plenty of examples.

So, let's get started!
This is the first Python program you've ever written.
Before moving on further, First let's get started with the most popular 'Hello World' program.
print("Hello, Python!")
See how simple it is to use a print statement to display the text!!!
This single line of code will give you the confidence to start o the Python journey.

Python has 3 fundamental concepts called statements, indentation and comments.
Comments are classified into three di erent types such as Single-line, Multi-line and Docstring.
Then you will learn about Indentation - Python uses indentation to indicate a block of code.
Statements, there are various types of - Empty, Simple, and Compound that will be discussed in detail later.

Don't be worried about the terms used here; you'll pick them up quickly.

Let's start!
Why do we use comments?

Wondering why a hash is used instead of a question mark?

Well the computer ignores any text written after the hash.

It won't compute it.

Let us see, what does a comment do?

Comments are used to make the source code more readable and understandable by the users.
You can write anything relevant to the code in the comment or just explain the logic of the code.
In computer programming, a comment is a programmer-readable explanation or annotation in the source code of a computer program. They are added with the purpose of **making the source code easier for humans to understand**, and are generally ignored by compilers and interpreters.

You will now learn how to write comments.
Let's start with a single line comment.

It starts with a hash symbol (#).
The hash sign must appear at the start of each line comment.

For example,

```
 # Single line comment
```

When Python reads a line that starts with a hash mark, it assumes that the rest of the line can be ignored. Single line comments are typically written in the code itself.

If you want to say 10 di erent lines of comments then just add a hash at the beginning of each line and share all that you want.

```
# THIS IS A MULTILINE
# COMMENT IN PYTHON
# USING THE SINGLE LINE
# COMMENT CONSECUTIVELY
```

The second type of comment is Multi-line comment.
It is called so because it can span within multiple lines.
When the Python sees three free-flowing quotation marks in a row it knows that it should ignore everything until it comes across three more quotations. This is also called Docstring.
A docstring can be a triple single or triple double quotation..
Example

```
“””You can write docstring like this”””
```

‘’’You can also write docstring like this’’’
You often use a multiline comment at the beginning of a program to describe what a program does or who wrote it.
You must always create comments as a programmer for various reasons.
Assume that the programmes are frequently completed by groups of people rather than by a single person.
Your teammates will be able to build on the programmes more easily if you leave comments.
Second, it's likely that the project you're working on will be passed down to others in the future.
These individuals must be aware of what your code accomplishes.

Finally, comments make it easier to understand your own code. Especially if you take a break from it and come back to it later.

Make sure that your comments help to make your application more readable.
That's simple to comprehend! Right?

INDENTATION

I hope you're still enjoying your Python fundamentals.
You will learn about indentation, how to apply it, and why it's important in this Python language.

Indentation is a very important concept of Python because, without proper indentation in the Python code, you will end up seeing an ‘Indentation error’ and the code will not get compiled.

That is a major issue! What exactly is indentation?
Python uses it **to highlight the blocks of code**. Whitespace is used for indentation in Python. All statements with the same distance to the right belong to the same block of code. If a block has to be more deeply nested, it is simply indented further to the right.

In simple terms indentation refers to adding the 4 spaces at the beginning of a code line.
Similarly for each level 4 spaces are added further.

But it brings the question: is it even necessary?
Python uses indentation to indicate a block of code. Well, a block is a several lines of code having the indentation at the same level.
Take a look at the meme card, which shows a developer checking indentation with a scale to avoid making a mistake.
Isn't it amusing?
Let me explain this with an example.

The expressions ‘Print' and ‘For' are separate. They are the owner statements.
As a result, they are aligned in the same level.
The ‘If’ statement is now controlled by the ‘for’ statement. To do so, you'll need to add four spaces.
Following this statement, the ‘If' statement controls the subsequent statements.
For indentation, you'll need to add four more spaces.
All statements within the if statement must be aligned.
Then this print represents the end of the if statement, and the following print represents the end of the for statement.
If there is no indentation, Python has no idea which statement to run next or which statement goes to which block. The flow of execution is guided by indentation.

I hope you have understood the concept.

We will learn about data types such as numbers in this module. But what exactly does data type imply?

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data.
To begin with, there are multiple data types such as numbers, strings, and booleans.

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data.

So let’s start with the numbers.
What are numbers?

There are three kinds of numeric data types such as, Integers, Floats, and Complex used to represent the number.
Let us see the first numeric data type is 'integers'.
Integers are numbers, it can be either positive, negative, or zero. But it cannot have fractions or decimal points. For example, 50, 0, -2 are treated as integers.
We can perform simple calculations with integers such as addition in the example.

```
>>> 6 + 4   (Six plus four results with 10)
10
>>> 1 + 2 + 4 + 10 - 3
14
>>> print(1 + 2 +  3 + 4 + 5 + 10)
25
```

Second is **Float**
Now what is the '**Float' data type**?
'Float' data type is similar to integers except that float data types will have a decimal point. For example -3.2, 0.0, 4.5652 are treated as float data types. Note that the decimal component can be zero such as 0.0. Or you tell your height in decimals such as 5.3 feet.

The third numeric data type is Complex numbers.
Complex numbers are constituted by 2 numbers. They are represented as " x + yj ".
They consist of real and imaginary parts, where x is a real number and y is an imaginary number. The real imaginary part can be an integer or floating point.

In engineering and research, complex numbers are extremely essential.
Control theory, signal analysis, relativity, and fluid dynamics are just a few of the fields where they're useful.

In upcoming modules, we'll conduct some interesting activities using numbers.

Primary Data Type

Good day, everyone! you'll learn about two more di erent types of data types in this module: strings and booleans.

What is a string?
It is a bunch or sequence of characters. It can be letters, any special symbols such as exclamation signs, percentages or it could be the numbers.
Strings are declared using quotation marks.
Python recognizes both single quotes (') and double quotes (") as valid ways to use 'strings'.

You can't do arithmetic with a number that's been contained in a single or double quote as a string.

For Example '19' in a single quote is treated as a string.

If you're trying to add numbers, it should be an integer or a float.

The following code just adds the two numbers supplied to it.

```
>>> 5 + 10
15
```

Whereas, with the string, it does the string concatenation or just puts the two values in the quotes together. For example

```
>>> '51' + '20'
'5120'
>>> 'Python' + 'Programming'
'PythonProgramming'
```

String data types in Python are immutable.
This means that a string value can't be changed.
It has use in security, synchronisation and concurrency, caching, and class loading, among others. Now that's simple.

As mentioned earlier, the Boolean data type is the next topic you'll learn
about. True or False is the only value that a Boolean data type could
represent.

You can think of it as an answer to a ‘yes’ or ‘no’ question. In programming, you often need to know if an expression is True or False.

In Python, you may evaluate any expression and get one of two results: True or False.
Here's an example of how Boolean can be used in programming.
print(10 > 9)
Answer: True

print(10 == 9)
Answer: False

print(10 < 9)
Answer: False

Right, the Boolean data type appeared to be amazing!!

String

Greetings, friends! I hope your learning of the Python language has gone well so far.
You will learn about statements, values, and variables in this module, as well as how to use them.

Let’s get started with statements.
‘Statements’ are simple codes written for execution.
There are three types of statements.
One is anempty statement.
The next is asimple statement.This consists of a signal line and does not a ect the flow of the program. For example, a=10, return, break, pass

And the last one iscompound statements.

A collection of statements working together is said to be a compound statement. They a ect or control the execution of those other statements in some way. Example: if, for, while, try, etc. Within the compound statement indentation is important to consider.

Next, we'll look at one of Python's most important concepts: values and variables.
So what is a variable?
Variables are containers or Memory Placeholders for storing data values. Or in simple terms, a variable is something that stores information that can be used later.
It's like a box holding a character, an alphanumeric value, or another variable that you can change later.
For example a = 5. Here **a** is the variable and **5** is the value.
Variables are assigned values by use of the assignment operator =.
Variables can have values such as integers, characters, and strings.
Such as a = 5.2, name = "Sam", greeting = "Hello World!"

However, there are some restrictions that must be followed while assigning variables. Let's take a look at some of the dos and don'ts. Dos.

- A variable name must start with an alphabet or an underscore
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Variable names are case-sensitive (age, Age, and AGE are three di erent variables)

Donts

- No other Special character or symbols are allowed
- A variable name cannot start with a number

You've learned how to make a variable. Let's conduct a little activity to see which variables are valid and which are invalid, as well as why they are valid and invalid.

Valid Variable(s)	Invalid Variable(s)	Reason
Num	Number 1	Because of space
NUM	Num 1	Because of space
Num1	1Num	Starting with number

_Num	Num.1	Because of dot
Num_1	Program to do addition	Because of spaces
IF	if	It is a keyword
ELSE	else	It is a keyword

It's time to put the principles and methods for assigning value to variables into practise now that you've learned them.

```
# Let us consider three variables such as height, weight, and BMI in this
program height =4.79 weight = 68.7
bmi = weight / height * height
print(bmi)
```

I'm sure you're curious about the potential of using values and variables to make your life easier.

Statement

Good day, everyone! As we move on to new areas, I hope your confidence in the programming language grows, motivating you to take on new tasks.

In this module you will study about identifiers, assignment operators, and their types with examples.

An identifier is a sequence of one or more characters used to name a given program element. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

NOTE: Variable is only a kind of identifier, other kinds of identifiers are function names, class names, structure names, etc. So it can be said that all variables are identifiers whereas, vice versa is not true.

This is an example of identifier - first_name, last_name, phone_startwith_91

You may apply what you've learned about variables and identifiers to the following topic, called an assignment statement, now that you've grasped the concept and limitations.

In Python, the assignment operator is used to create or alter variables. The following assignment statement will be used:

Variable Name = Value
(or)
Variable Name = Expression
Example : a = 10
The name of the variable is on the left, and the value assigned to it is on the right.

Any value can be assigned to a variable. A number, characters, alphanumeric values, and strings can all be used.

Assignment can be of di erent types such as:

1. Simple Assignment
a=100
print(a)
Output 100

Here a simple variable is assigned a value.

2. You can assign multiple values to multiple variables at the same timeFor example
a, b = 100, 200 print(a) Output **100 print(b)**
Output **200**

3. You can assign more than three variables at a time. Moreover, it is also possible toassign di erent types of values.
Such as the use of float numbers, integers and even strings like in this example. **a, b, c = 0.1, 100, 'Python'**
print(a) Output
0.1 print(b)
Output **100**
print(c) Output
'Python'

4. You can assign the same value to multiple variables such as assigning a, b, c = 2021
In this case, all the variables a, b, and c will be assigned with the same value i.e., 2021.

Want to swap or exchange two variables' values!!!
You can swap or interchange the values of two variables using assignment statements.

With this example, you will see what I mean.

The conventional way of doing variable exchange is using temporary variable temp:

```
a=10
b=20
temp =
a a = b b
= temp
print(a)
print(b)
```

Python swaps or exchanges the value of two variables in a single assignment instead of three di erent assignments as follows.

```
a=10
b=20
a,b  =  b,a      #  Python  does  it  for  us  in  a  single  assignment  print(a)  print(b)
```

KEYWORDS & BUILT-iN FUNCTIONS

Good day, everyone! I hope you're all having pleasure learning the fundamentals of the Python programming language through easy examples that reinforce the concepts in your mind.

You will learn about keywords or reserved words, and built-in functions for the number data type in this module.

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other regular identifiers.
There are 33 keywords available in Python.
For example, if we try to assign number 10 to the keyword and, it will notify an error.
and = 10
Depending on how they're used, these keywords are divided into several categories. Let's have a look at what those categories are.

- Value Keywords: True, False, None.
- Operator Keywords: and, or, not, in, is.
- Control Flow Keywords: if, elif, else.
- Iteration Keywords: for, while, break, continue, else.
- Structure Keywords: def, class, with, as, pass, lambda.
- Returning Keywords: return, yield.
- Import Keywords: import, from, as.

Let us move on to our next topic called built-in functions.
What are built-in functions?
As of now, there are 68 built-in functions that Python can interpret.
Many built-in functions support mathematical operations.
Many useful functions can be used to speed up the coding process for a variety of tasks.
Let's have a look at a few of the examples.

Math Functions	Description
abs()	This Returns absolute value of a number abs(-11) Answer: 11
divmod()	This Returns quotient and remainder of integer division divmod(15, 2) Answer: (7, 1)

max()	This Returns the largest of the given arguments or items in an iterable max(100, 20, 400) Answer: 400
min()	This Returns the smallest of the given arguments or items in an iterable min(100, 20, 400) Answer: 20
pow()	Raises a number to a power pow(10, 2) Answer: 100
round()	Rounds a floating-point value round(100.56) Answer: 101 round(100.46) 100
sum()	Sums the items of an iterable sum(10, 20, 30) Answer: 60

Isn't it amazing that all of those arithmetic computations could be accomplished with just a few lines of code?

Well, in this module you have learned many of the amazing features of python. So stay updated for the next module.