# Learn Python Basics

Last Updated : 12 Apr, 2025

Python is a versatile, high-level programming language known for its readability and simplicity. Whether you're a beginner or an experienced developer, Python offers a wide range of functionalities that make it a popular choice in various domains such as web development, data science, artificial intelligence, and more. In this article, we'll cover the foundational concepts of Python programming to help you get started.

## Writing First Python Program

To begin coding in Python, we'll need to have **Python installed** on our system. You can download the latest version from the official Python website. Once installed, we can write and execute Python code using an Integrated Development Environment **(IDE)** like PyCharm, Vs Code (requires installing Python extension), or even a simple text editor.

```
print("Hello Geeks, Welcome to Python Basics")
```

**Output**

```
Hello Geeks, Welcome to Python Basics
```

**Explanation: print()** is a built-in function that outputs text or variables to the console. In this case, it displays the string "Hello, Geeks! Welcome to Python Basics".

## Comments in Python

Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program. Also, Comments enhance the readability of the code and help the programmers to understand the code very carefully.

```
# This is a single-line comment

"""
This is a
multi-line comment
or docstring.
"""
```

**Explanation:**

- **#**: Denotes a single-line comment.
- **""" """** or **''' '''**: Triple quotes are used for multi-line comments or docstrings.

## Variables in Python

Python Variable is a container that store values. Python is not “statically typed”. An Example of a Variable in Python is a representational name that serves as a pointer to an object. Once an object is assigned to a variable, it can be referred to by that name.

- Must start with a letter (a-z, A-Z) or an underscore (_).
- Cannot start with a number.
- Can only contain alphanumeric characters and underscores.
- Case-sensitive (name, Name, and NAME are different variables).
- The reserved words(keywords) in Python cannot be used to name the variable in Python.

**Example:**

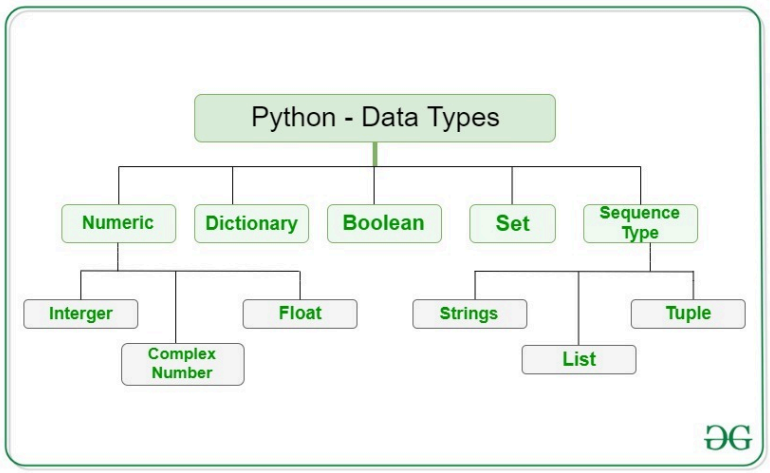
```
# Integer assignment
age = 45

# Floating-point assignment
salary = 1456.8

# String assignment
name = "Geek"

print(age)
print(salary)
print(name)
```

**Output**

```
45
1456.8
Geek
```

## Data Types in Python

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. Since **everything** is an **object** in Python programming, data types are classes and variables are instances (objects) of these classes.




**Example:** This code assigns variable **'x'** different values of various data types in Python.

```
x = "Hello World" # string
x = 50  # integer
x = 60.5  # float
x = 3j  # complex
x = ["geeks", "for", "geeks"]  # list
x = ("geeks", "for", "geeks")  # tuple
x = {"name": "Suraj", "age": 24} # dict
```

## Python Input/Output

Python provides simple functions for input and output operations.

### Input

The **input()** function allows user input, example:

```
val = input("Enter your value: ")
print("You entered:", val)
```

**Output:**

```
Enter your value: 11
You entered: 11
```

The **input()** function in Python always returns data as a string, regardless of what the user enters. If we want to store the input as another data type (like **int**, **float**, etc.), we need to explicitly convert (**typecast**) it.

**Example:**

```
name = input("Enter your name: ")
print(type(name))

age = int(input("Enter your age: "))
print(type(age))
```

**Output:**

```
Enter your name: Geeks
<class 'str'>
Enter your age: 8
<class 'int'>
```

**Explanation:**

- **name** stores the input as a string (default behavior of **input**()).
- **age** stores the input as an integer using **int()** for **typecasting**.

```
# Python program show input and Output
val = input("Enter your value: ")
print(val)
```

## Python Operators

In Python programming, Operators in general are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations. In this article, we will look into different types of Python operators.

### Arithmetic Operators

Python Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication, and division. Types of arithmetic operators: **+, -, *, /, //, %, ****

The precedence of Arithmetic Operators in Python is as follows:

1. P – Parentheses
2. E – Exponentiation
3. M – Multiplication (Multiplication and division have the same precedence)
4. D – Division
5. A – Addition (Addition and subtraction have the same precedence)
6. S – Subtraction

**Example:**

```
a = 9
b = 4
add = a + b

sub = a - b

mul = a * b

mod = a % b

E = a ** b
print(add)
print(sub)
print(mul)
print(mod)
print(E)
```

**Output**

```
13
5
36
1
6561
```

## Comparison Operators

Comparison operators are used to compare two values. They return a Boolean value — either True or False — depending on whether the comparison is correct. These operators are often used in conditional statements like if, while, and loops.

**Example:**

```
a = 10
b = 20

print(a == b)   # False, because 10 is not equal to 20
print(a != b)   # True, because 10 is not equal to 20
print(a > b)    # False, 10 is not greater than 20
print(a < b)    # True, 10 is less than 20
print(a >= b)   # False, 10 is not greater than or equal to 20
print(a <= b)   # True, 10 is less than or equal to 20
```

**Output**

```
False
True
False
True
False
True
```

**Explanation:**

- Each **print()** checks a condition.
- The result is either **True** or **False** depending on whether the comparison holds.
- These operators are commonly used to control program flow in **if** statements, loops, etc.

## Logical Operators

Python Logical operators perform **Logical AND** , **Logical OR** , and **Logical NOT** operations. It is used to combine conditional statements. Types of logicsl operators: and, or, not.

```
a = True
b = False
print(a and b)
print(a or b)
print(not a)
```

**Output**

```
False
True
False
```

## Bitwise Operators

Python Bitwise operators act on bits and perform bit-by-bit operations. These are used to operate on binary numbers. Types of bitwise operators: &, |, ^, ~, <<, >>

```
a = 10
b = 4
print(a & b)
print(a | b)
print(~a)
print(a ^ b)
print(a >> 2)
print(a << 2)
```

**Output**

```
0
```

```
2
40
```

### Assignment Operators

Python Assignment operators are used to assign values to the variables. Types of assignment operators: =, +=, -=, *=, /=, %=, **=, //=, &=, |=, ^=, >>=, <<=.

```
a = 10
b = a
print(b)
b += a
print(b)
b -= a
print(b)
b *= a
print(b)
b <<= a
print(b)
```

**Output**

```
10
20
10
100
102400
```

## Python If Else

In Python, the if statement is used to run a block of code only when a specific condition is true. If the condition is false and you want to run a different block of code, you can use the else statement. This allows your program to make decisions and respond differently based on conditions.

### Example 1: Python IF-Else

```
i = 20
if (i < 15):
    print("i is smaller than 15")
    print("i'm in if Block")
else:
    print("i is greater than 15")
    print("i'm in else Block")
print("i'm not in if and not in else Block")
```

**Output**

```
i is greater than 15
i'm in else Block
i'm not in if and not in else Block
```

- The condition **i < 15** is False, so the else block is executed.
- The last **print()** runs no matter what because it's outside the **if-else** structure.

### Example 2: Python if-elif-else ladder

Sometimes, weneed to check multiple conditions. In such cases, Python provides the **if-elif-else** structure.

```
i = 20
if (i == 10):
    print("i is 10")
elif (i == 15):
    print("i is 15")
elif (i == 20):
    print("i is 20")
else:
    print("i is not present")
```

**Output**

```
i is 20
```

**Explanation:**

- Python checks conditions from top to bottom.
- Once a condition is **True**, it executes that block and skips the rest.
- If none of the conditions match, it executes the **else** block.

## Python Loops

### For Loop

Python For loop is used for sequential traversal i.e. it is used for iterating over an iterable like String, Tuple, List, Set, or Dictionary. Here, we will see a "for" loop in conjunction with the range() function to generate a sequence of numbers starting from 0, up to (but not including) 10, and with a step size of 2. For each number in the sequence, the loop prints its value using the print() function.

```
for i in range(0, 10, 2):
    print(i)
```

**Output**

```
0
2
4
6
8
```

**Explanation:**

### While Loop

A while loop continues to execute as long as a condition is True. In this example, the condition for while will be True as long as the counter variable (count) is less than 3.

```
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
```

#### Output

```
Hello Geek
Hello Geek
Hello Geek
```

#### Explanation:

- The loop runs **3** times because **count** goes from 0 - 1 - 2.
- Once count becomes 3, the condition count < 3 becomes False and the loop stops.

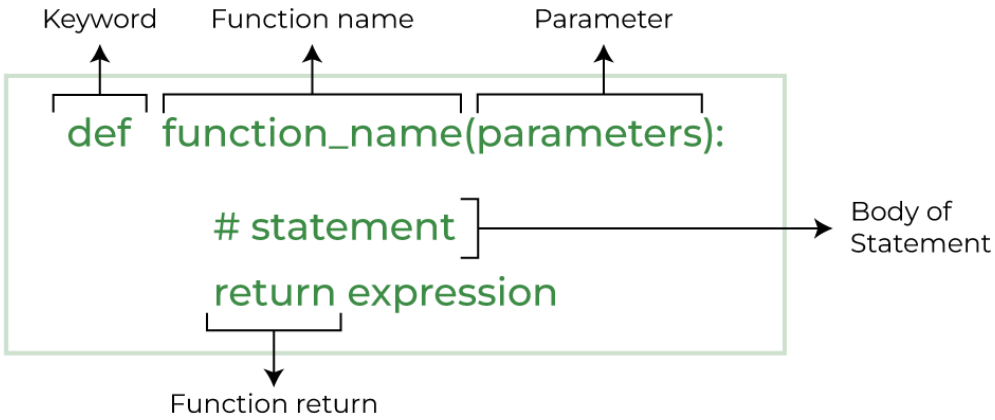
*Also see: Use of break, continue and pass in Python*

## Python Functions

Python Function is a block of reusable code that performs a specific task. Functions help make your code modular, readable, and easier to debug.

There are two main types of functions in Python:

1. **Built-in** functions like **print**(), **len**(), **type**()
2. **User-defined** functions created using the **def** keyword




### Example: User-Defined Function to Check Even/Odd

```
def evenOdd(x):
    if x % 2 == 0:
        print("even")
    else:
```

```
evenOdd(3)
```

**Output**

```
even
odd
```

**Explanation:**

- The function **evenOdd(x)** checks if **x** is divisible by 2.
- If it is, it prints "even"; otherwise, "odd".

## What's Next

After understanding the Python Basics, there are several paths you can explore to further enhance your skills and delve deeper into the language:

1. Continuous Python Learning : This article offers a well-organized, in-depth tutorial on Python, guiding learners from the basics to more advanced topics.
2. Advanced Python Concepts : This article covers some advance Python concepts that distinguishes Python from any other language such as list comprehenison, lambda function, etc.
3. Python Packages and Frameworks : Python has a rich ecosystem of libraries and frameworks for a wide range of tasks such as web development with frameworks like Django or Flask, data analysis and visualization with libraries like Pandas and Matplotlib, machine learning and artificial intelligence with TensorFlow or PyTorch, or automation with libraries like Selenium or BeautifulSoup This article explores them in detail.
4. Build Python Projects : Learn how to build real-world applications using Python. This section includes a variety of projects to help you apply your skills and create meaningful solutions.



## Similar Reads

**Python Fundamentals**

**Python Data Structures**

**Advanced Python**

**Data Science with Python**

**Web Development with Python**

**Python Practice**

GeeksforGeeks
Sanchhaya Education Private Limited

**Corporate & Communications Address:**
A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

**Registered Address:**
K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305



**Company**
About Us
Legal
Privacy Policy
Contact Us
Advertise with us
GFG Corporate Solution
Campus Training Program

**Explore**
POTD
Job-A-Thon
Community
Videos
Blogs
Nation Skill Up

**Tutorials**
Programming Languages
DSA
Web Technology
AI, ML & Data Science
DevOps
CS Core Subjects
Interview Preparation
GATE
Software and Tools

**Courses**
IBM Certification
DSA and Placements
Web Development
Programming Languages
DevOps & Cloud
GATE
Trending Technologies

**Videos**
DSA
Python
Java
C++
Web Development
Data Science
CS Subjects

**Preparation Corner**
Aptitude
Puzzles
GfG 160
DSA 360
System Design