

Module 1

Introduction: Python programming, history, features, Installing Python, Python Interpreter, IDLE, Keywords, Variables, Constants, Literals, Data Types, Data type conversion, Indentation, Comments, expression, statements, Input & Output functions, range() function , Operators: assignment (single value, multiple values), arithmetic, relational, logical.

Introduction to Python Programming:

Python is a **general-purpose, high-level, object-oriented, interpreted programming language**. It is widely known for its **simplicity, readability, and versatility**. Python can be used to build a wide range of applications such as:

- Web applications
- Desktop software
- Games
- Machine Learning (ML) models
- Artificial Intelligence (AI) solutions
- Data Science and Data Analytics projects
- Automation scripts

Python was **created by Guido van Rossum** and **first released in 1991**. Over the years, it has grown into one of the most popular programming languages across the world because of its:

- **Beginner-friendly syntax** (similar to English)
- **Cross-platform support** (runs on Windows, Linux, macOS, etc.)
- **Large community support** and vast number of libraries

Today, Python is considered a **must-learn language** for beginners as well as professionals in fields like software development, data science, machine learning, and automation.

History of Python:



- Python was conceived in the late **1980s** and was named after the *BBC TV show Monty Python's Flying Circus*.
- **Guido van Rossum** started implementing **Python at CWI(Centrum Wiskunde & Informatica) in the Netherlands in December of 1989**.
- Officially released in **1991 (Python 0.9.0)**.
- This was a successor to the ABC programming language which was capable of **exception handling and interfacing** with the Amoeba operating system.
- On **October 16 of 2000, Python 2.0 released** with many new features.
- Then **Python 3.0 was released on December 3, 2008**.
- Latest version of Python is 3.13.2, released on February 4, 2025

Features of Python:

1. **Simple and Easy** – Syntax close to English.
2. **Interpreted** – Runs directly without compilation.
3. **Cross-Platform** – Works on Windows, macOS, Linux.
4. **Free and Open Source**.
5. **Object-Oriented and Functional** programming styles supported.
6. **Extensive Libraries** for web, AI, ML, data science, etc.
7. **Portable** – Write once, run anywhere.



a. Easy

Python is very easy to learn and understand; any beginner can learn Python easily. When writing code in Python, you need fewer lines of code compared to languages like Java.

b. Interpreted

It is interpreted(executed) line by line. This makes it easy to test and debug.

c. Object-Oriented

The Python programming language supports classes and objects and hence it is object-oriented.

d. Free and Open Source

The language and its source code are available to the public for free; there is no need to buy a costly license.

e. Portable

Since Python is open-source, you can run it on Windows, Mac, Linux or any other platform. Your programs will work without any need to change it for every machine.

f. GUI Programming

Python is useful to develop GUI (Graphical User Interface) using its **Tkinter library**.

g. Large Python Library

Python provides you with a large standard library.

You can use it to implement a variety of functions without the need to reinvent the wheel every time. Just pick the code you need and continue.

In this Python introduction, we have provided a short description. You must read them in detail at [Python Programming Features](#)

h. Cross-Platform and Dynamic Typing

Python Runs on Windows, macOS, Linux, even phones. There is no need to declare variable types in advance.

Applications of Python

Python is easy to pick-up even if you come from a non-programming background. You can look at the code and tell what's going on.

Talking of Python applications, some of the cool things that you can do are –

- Build a website using Python
- Develop a game in Python
- Perform Computer Vision (Facilities like face-detection and color-detection)
- Implement Machine Learning (Give a computer the ability to learn)
- Enable Robotics with Python
- Perform Web Scraping (Harvest data from websites)
- Perform Data Analysis using Python
- Automate a web browser
- Perform Scripting in Python
- Perform Scientific Computing using Python
- Build Artificial Intelligence
- Create Audio and Video Applications
- Design Computer Aided Design Applications

Python isn't limited to these applications. If you've ever used services from brands like YouTube, Dropbox, and Netflix, then you've been a consumer of Python.

Installing Python:

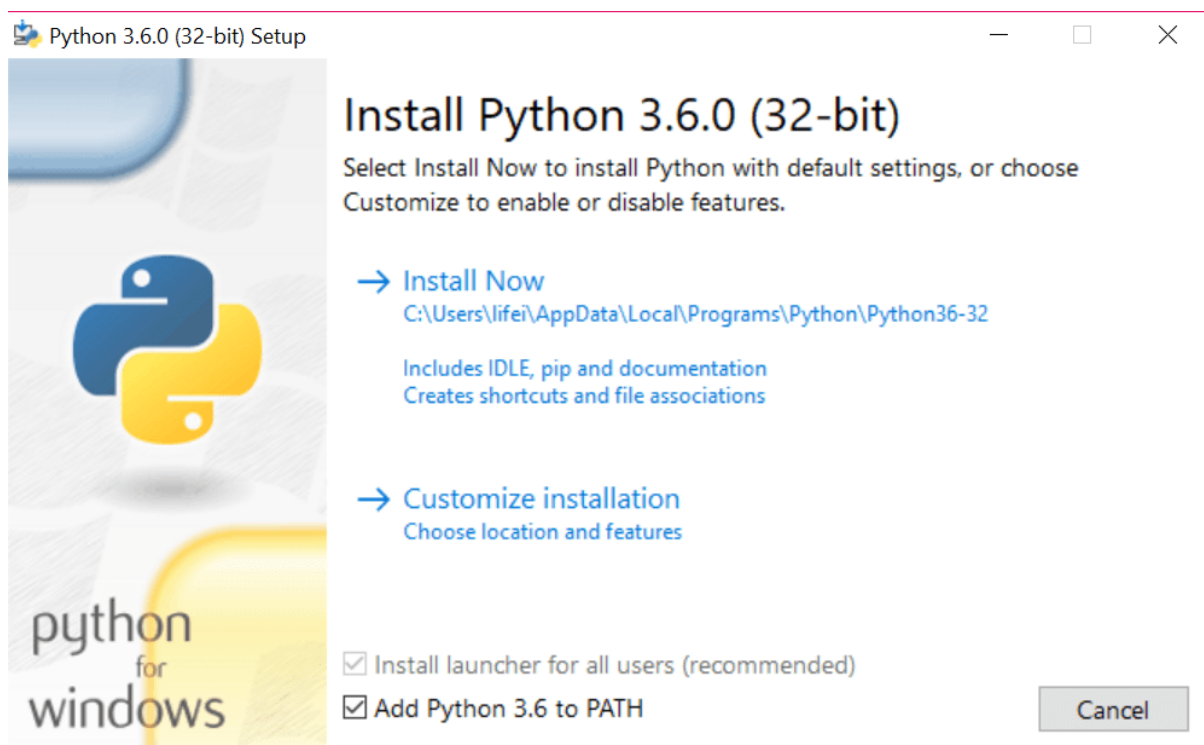
1. Download from python.org.
2. Run the installer and check “**Add Python to PATH**” option.
3. Installation includes:
 - Python **interpreter**
 - **IDLE (Integrated Development and Learning Environment)**

How to Install Python on Windows?

Installing and running Python on your personal computer is no difficult task. It involves just a few simple steps:

- 1: Download Python binaries from python.org
- 2: Install the binaries
- 3: Add Python to system environment variables
- 4: Install pip
- 5: [Optional] Install virtualenv using pip

1. Download binaries from python.org



Firstly, to install Python Windows you need to download the required Python binaries. We recommend you to download Python latest (Python 3.6.3, currently) installer for Windows.

If you have a different OS, download binaries accordingly. You may choose an x86-64 installer if you have a 64-bit system.

Choose an x86 installer if you have a 32-bit system. But you can also click on the download button in the page header.

2. Install Python binaries

Next, you need to install the binaries you downloaded.

Run the installer. It will show you two options.

Install PIP Windows

This is the default option. It also includes the IDLE (Integrated Development Environment), pip, and the official documentation. It also creates shortcuts.

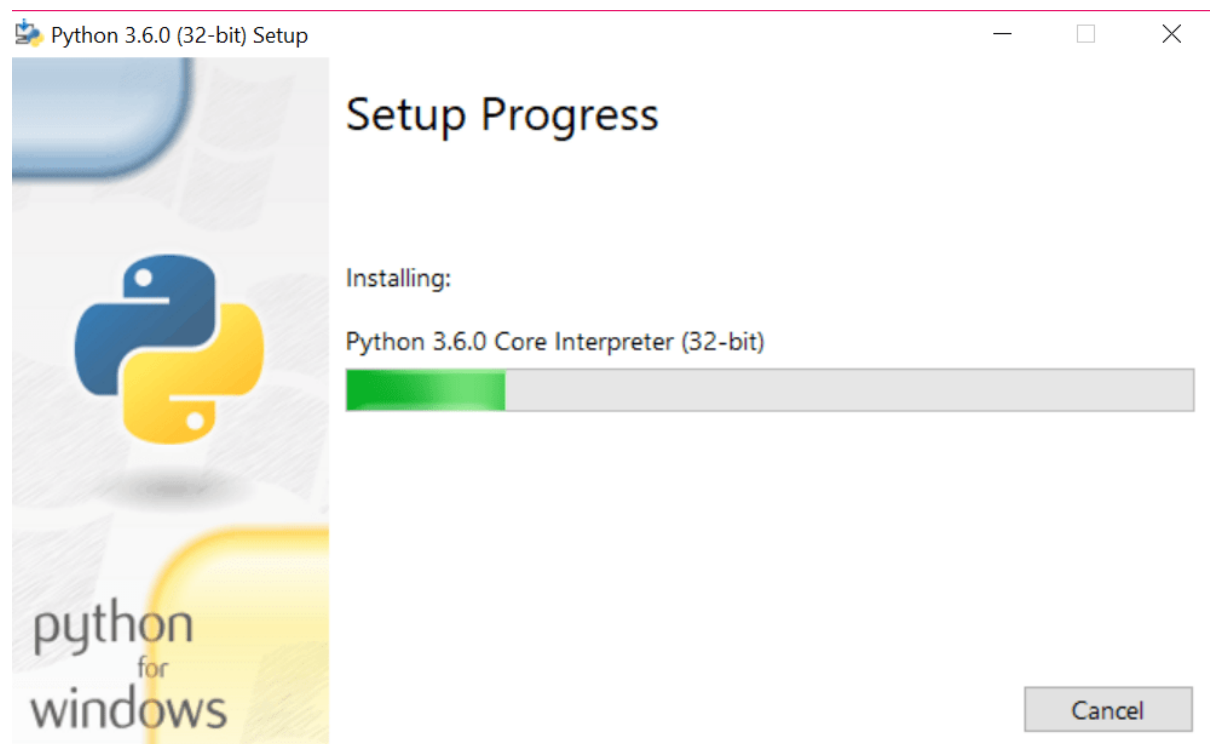
Customize Python Installation

This option allows you to choose the features that you want.

Firstly, to install Python Windows you need to download the required Python binaries. We recommend you to download Python latest (Python 3.6.3, currently) installer for Windows.

If you have a different OS, download binaries accordingly. You may choose an x86-64 installer if you have a 64-bit system.

Choose an x86 installer if you have a 32-bit system. But you can also click on the download button in the page header.



Python Interpreter:

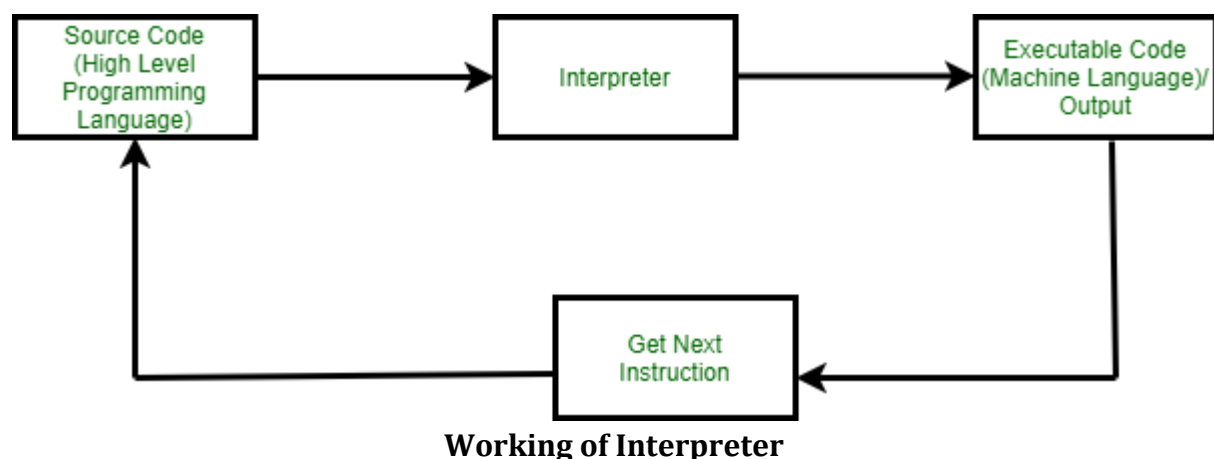
- Python is an interpreted language developed by Guido van Rossum in the year of 1991. As we all know Python is one of the most high-level languages used today because of its massive versatility and portable library & framework features. It is an **interpreted** language because it executes **line-by-line instructions**.
- There are actually two way to execute python code one is in **Interactive mode** and another thing is having Python prompts which is also called **script mode**.
- Python does not convert high level code into low level code as many other programming languages do rather it will scan the entire code into something called **bytecode**.
- Every time when Python developer runs the code and start to execute the compilation part execute first and then it generate an byte code which get converted by PVM Python Virtual machine that understand the analogy and give the desired output.

Interpreted Languages: Perl, BASIC, Python, JavaScript, Ruby, PHP.

Compiled Languages: C, C++, C#, COBOL and CLEO.

What are Interpreters?

Interpreters are the computer program that will convert the source code or an high level language into intermediate code (machine level language). It is also called translator in programming terminology. Interpreters executes each line of statements slowly. This process is called Interpretation. For example Python is an interpreted language, PHP, Ruby, and JavaScript.



Advantages of using Interpreter

- It is flexible and error localization is easier.
- It is smaller in size.
- It executes line by line execution.

Disadvantages of using Interpreters

- It takes lot of time to translate.
- It is slower.
- It uses lot of storage.

Difference between Compilers and Interpreters

Compiler	Interpreter
It translates a program on a single run	It executes a program line by line.
It is an Fast Process	It is an Slow Process
It generates output in the form of .(exe)	It does not generate any form of outputs.
It utilizes CPU more.	It takes less utilization of CPU
It is used in Production environment	It is maximum used in Programming and development environment.
C, C++, C# are the compiled languages	Python, Ruby, PHP are the interpreted languages.
It takes lot of time to analyze the code structure	It takes less time compared to compilers.

IDLE:

Python IDLE is the default integrated development environment (IDE) that comes bundled with every Python installation, helping you to start coding right out of the box. In this tutorial, you'll explore how to interact with Python directly in IDLE, edit and execute Python files, and even customize the environment to suit your preferences.

By the end of this tutorial, you'll understand that:

- **Python IDLE** is completely **free** and comes packaged with the Python language itself.
- **Python IDLE** is an IDE included with Python installations, designed for basic editing, execution, and debugging of Python code.
- You **open IDLE** through your system's application launcher or terminal, depending on your operating system.
- You can **customize IDLE** to make it a useful tool for writing Python.

Understanding the basics of Python IDLE will allow you to write, test, and debug Python programs without installing any additional software.

Keywords:

Python Keywords are reserved words with fixed meanings that define Python's syntax. Cannot be used as names.

Python Identifiers are user-defined names for variables, functions, or classes. Must follow naming rules (no digits at start, only _ allowed).

Keywords in Python

- Predefined and reserved words with special meanings.
- Used to define the syntax and structure of Python code.
- Cannot be used as identifiers, variables, or function names.
- Written in lowercase, except True and False.
- Python 3.11 has 35 keywords.
- The keyword module provides:
 - **iskeyword()** → checks if a string is a keyword.
 - **kwlist** → returns the list of all keywords.

Rules for Keywords in Python

- Python keywords cannot be used as identifiers.
- All the keywords in Python should be in lowercase except True and False.

Variables:

In Python, variables are used to store data that can be referenced and manipulated during program execution. A variable is essentially a name that is assigned to a value. Unlike many other programming languages, Python variables do not require explicit declaration of type. The type of the variable is inferred based on the value assigned. Variables act as placeholders for data. They allow us to store and reuse values in our program.

Example:

```
# Variable 'x' stores the integer value 10
x = 5

# Variable 'name' stores the string "Prashant"
name = "Prashant"

print(x)
print(name)
```



```
5
Prashant
```

Variable Names:

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- 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 different variables)
- A variable name cannot be any of the Python keywords.

Example:

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Constants:

In Python, constants are variables whose values are intended to remain unchanged throughout a program. They are typically defined using uppercase letters to signify

their fixed nature, often with words separated by underscores (e.g., MAX_LIMIT). Let's understand with the help of example:

```
# Mathematical constant
PI = 3.14159

# Acceleration due to gravity
GRAVITY = 9.8

print(PI)
print(GRAVITY)
```

Output

```
3.14159
9.8
```

Rules while declaring a Constant

- Python constant and variable names can include:
 - Lowercase letters (a-z)
 - Uppercase letters (A-Z)
 - Digits (0-9)
 - Underscores (_)
- Naming rules for constants:
 - Use **UPPERCASE** letters for constant names (e.g., CONSTANT = 65).
 - Do not start a constant name with a digit.
 - Only the underscore (_) is allowed as a special character; other characters (e.g., !, #, ^, @, \$) are not permitted.
- Best practices for naming constants:
 - Use meaningful and descriptive names (e.g., VALUE instead of V) to make the code clearer and easier to understand.

Example of Constant

There are multiple use of constants here are some of the examples:

1. Mathematical Constant:

```
PI = 3.14159  
E = 2.71828  
GRAVITY = 9.8
```

2. Configuration settings:

```
MAX_CONNECTIONS = 1000  
TIMEOUT = 15
```

3. UI Color Constants:

```
BACKGROUND_COLOR = "#FFFFFF"  
TEXT_COLOR = "#000000"  
BUTTON_COLOR = "#FF5733"
```

Literals:

Literals in Python are fixed values written directly in the code that represent constant data. They provide a way to store numbers, text, or other essential information that does not change during program execution. Python supports different types of literals, such as numeric literals, string literals, Boolean literals, and special values like None. **For example:**

- 10, 3.14, and 5 + 2j are numeric literals.
- 'Hello' and "Python" are string literals.
- True and False are Boolean literals.

Numeric Literals

Numeric literals represent numbers and are classified into three types:

- **Integer Literals** – Whole numbers (positive, negative, or zero) without a decimal point. **Example:** 10, -25, 0
- **Floating-point (Decimal) Literals** – Numbers with a decimal point, representing real numbers. **Example:** 3.14, -0.01, 2.0
- **Complex Number Literals** – Numbers in the form $a + bj$, where a is the real part and b is the imaginary part. **Example:** 5 + 2j, 7 - 3j

```
# Integer literals
a = 100
b = -50

# Floating-point literals
c = 3.14
d = -0.005

# Complex number literals
e = 4 + 7j
f = -3j

print(a, b, c, d, e, f)
```

Output

```
100 -50 3.14 -0.005 (4+7j) (-0-3j)
```

String Literals

String literals are sequences of characters enclosed in quotes. They are used to represent text in Python.

Types of String Literals:

- **Single-quoted strings** – Enclosed in single quotes (' '). Example: 'Hello, World!'
- **Double-quoted strings** – Enclosed in double quotes (" "). Example: "Python is fun!"
- **Triple-quoted strings** – Enclosed in triple single (''' ''') or triple double (""" """) quotes, generally used for multi-line strings or docstrings. Example:
'''This is
a multi-line
string'''

Boolean Literals

Boolean literals represent truth values in Python. They help in decision-making and logical operations. Boolean literals are useful for controlling program flow in conditional statements like if, while, and for loops.

Types of Boolean Literals:

- **True** – Represents a positive condition (equivalent to 1).
- **False** – Represents a negative condition (equivalent to 0).

```
# Boolean literals
a = True
b = False

print(a, b)          # Output: True False
print(1 == True)     # Output: True
print(0 == False)    # Output: True
print(True + 5)      # Output: 6 (1 + 5)
print(False + 7)     # Output: 7 (0 + 7)
```

Explanation:

- True is treated as 1, and False is treated as 0 in arithmetic operations.
- Comparing `1 == True` and `0 == False` returns True because Python considers True as 1 and False as 0.

Data Types in Python:**Built-in Data Types**

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:

`str`

Numeric Types:

`int`, `float`, `complex`

Sequence Types:

`list`, `tuple`, `range`

Mapping Type:

`dict`

Set Types:

`set`, `frozenset`

Boolean Type:

`bool`

Binary Types:

`bytes`, `bytearray`, `memoryview`

None Type:

`NoneType`

In Python, a data type is a classification that specifies the kind of values a variable can hold and determines what operations can be performed on that data. Since Python is an object-oriented language, data types are fundamentally classes, and variables are instances (objects) of these classes.

Here are the key aspects of data types in Python:

- **Classification of Data:**

Data types categorize data items, such as numbers, text, or collections of items. This classification helps the interpreter understand how to store and manipulate the data effectively.

- **Dynamic Typing:**

Python is a dynamically typed language. This means you do not explicitly declare the data type of a variable when you create it. Instead, Python automatically infers the data type at runtime based on the value assigned to the variable.

- **Operations and Behavior:**

The data type of a variable dictates the set of operations that can be applied to it. For example, you can perform arithmetic operations on numeric types (integers, floats), but not directly on string types.

Data type conversion:

Data type conversion in Python involves changing a value from one data type to another. This process is also known as type casting. Python supports two main types of data type conversion: Implicit Type Conversion (Coercion).

This occurs automatically by the Python interpreter when different data types are involved in an operation. Python will convert one data type to another to prevent data loss. For example, when an integer and a float are added, the integer is implicitly converted to a float before the addition.

Explicit Type Conversion (Type Casting).

This is performed manually by the programmer using built-in functions to convert a value to a specific data type.

Common functions for explicit type conversion include:

- `int()`: Converts a value to an integer. It truncates decimal parts when converting from float and requires a valid numeric string when converting from string.

Python



```
float_num = 3.14
int_num = int(float_num)
print(int_num)

str_num = "123"
int_from_str = int(str_num)
print(int_from_str)
```

- `float()` : Converts a value to a floating-point number.

Python



```
int_val = 10
float_val = float(int_val)
print(float_val)

str_float = "4.56"
float_from_str = float(str_float)
print(float_from_str)
```

Indentation in Python

-

In Python, indentation is used to define blocks of code. It tells the [Python interpreter](#) that a group of statements belongs to a specific block. All statements with the same level of indentation are considered part of the same block. [Indentation](#) is achieved using whitespace (**spaces or tabs**) at the beginning of each line.

For Example:

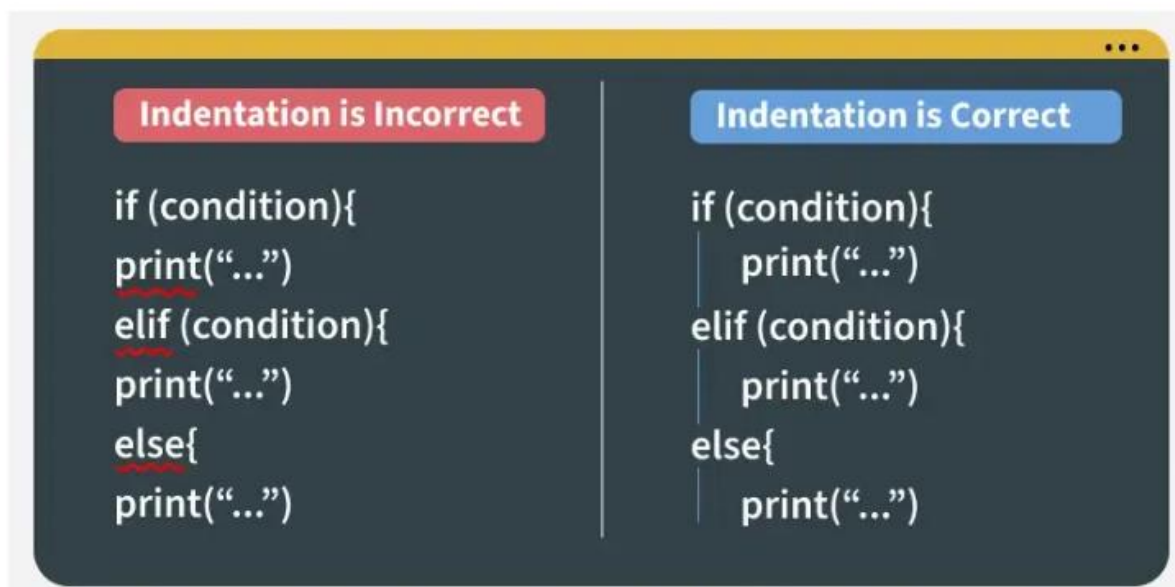
```
if 10 > 5:
    print("This is true!")
    print("I am tab indentation")

print("I have no indentation")
```

Output

```
This is true!
I am tab indentation
I have no indentation
```

- The first two print statements are indented by 4 spaces, so they belong to the if block.
- The third print statement is not indented, so it is outside the if block.



Comments in Python:

Python comments start with the hash symbol # and continue to the end of the line. [Comments in Python](#) are useful information that the developers provide to make the reader understand the source code. It explains the logic or a part of it used in the code. Comments in Python are usually helpful to someone maintaining or enhancing your code when you are no longer around to answer questions about it. These are often cited as useful programming convention that does not take part in the output of the program but improves the readability of the whole program. Comments in Python are identified with a hash symbol, #, and extend to the end of the line.

Types of comments in Python

A comment can be written on a single line, next to the corresponding line of code, or in a block of multiple lines. Here, we will try to understand examples of comment in Python one by one:

Single-line comment in Python

Python single-line comment starts with a hash symbol (#) with no white spaces and lasts till the end of the line. If the comment exceeds one line then put a hashtag on the next line and continue the comment. Python's single-line comments are proved useful for supplying short explanations for variables, function declarations, and expressions. See the following code snippet demonstrating single line comment:

Example 1:

Python allows comments at the start of lines, and Python will ignore the whole line.

```
# This is a comment
# Print "GeeksforGeeks" to console
print("GeeksforGeeks")
```

Output

```
GeeksforGeeks
```

If we Skip Indentation, Python will throw error.

```
if 10>5:
print("GeeksforGeeks")
```

```
Hangup (SIGHUP)
File "Solution.py", line 2
    print("GeeksforGeeks")
    ^
IndentationError: expected an indented block
```

Error

Example 2:

Python also allows comments at the end of lines, ignoring the previous text.

```
a, b = 1, 3 # Declaring two integers
sum = a + b # adding two integers
print(sum) # displaying the output
```

Output

4

Multiline comment in Python

Use a hash (#) for each extra line to create a [multiline comment](#). In fact, Python multiline comments are not supported by Python's syntax. Additionally, we can use Python multi-line comments by using multiline strings. It is a piece of text enclosed in a delimiter ("""") on each end of the comment. Again there should be no white space between delimiter ("""). They are useful when the comment text does not fit into one line; therefore need to span across lines. Python Multi-line comments or paragraphs serve as documentation for others reading your code. See the following code snippet demonstrating a multi-line comment:

Example :

In this example, we are using an extra # for each extra line to create a Python multiline comment.

```
# This is a comment
# This is second comment
# Print "GeeksforGeeks" to console
print("GeeksforGeeks")
```

Output

GeeksforGeeks

- `str()` : Converts a value to a string.

Python



```
num = 123
str_num = str(num)
print(str_num)
print(type(str_num))
```

- `bool()` : Converts a value to a boolean. Zero, `None`, empty sequences (like `""`, `[]`, `()`, `{}`), and empty sets are converted to `False`; all other values are converted to `True`.

Python



```
print(bool(0))
print(bool(5))
print(bool(""))
print(bool("hello"))
```

- `list()`, `tuple()`, `set()` : Convert iterable objects into lists, tuples, or sets, respectively.

Python



```
my_string = "Python"
my_list = list(my_string)
print(my_list)

my_tuple = tuple(my_list)
print(my_tuple)
```

Statements:

A **statement** is an instruction that the Python interpreter can execute. We have only seen the assignment statement so far. Some other kinds of statements that we'll see shortly are **while** statements, **for** statements, **if** statements, and **import** statements. (There are other kinds too!)

A **Python statement** is an instruction that the [Python interpreter](#) can execute. There are different types of **statements in Python** language as Assignment statements, Conditional statements, Looping statements, etc. The token character NEWLINE is

used to end a statement in Python. It signifies that each line of a [Python](#) script contains a statement. These all help the user to get the required output.

Types of statements in Python?

The different types of Python statements are listed below:

- [Multi-Line Statements](#)
 - Python Conditional and Loop Statements
 - [Python If-else](#)
 - [Python for loop](#)
 - [Python while loop](#)
 - [Python try-except](#)
 - [Python with statement](#)

Expression:

An **expression** is a combination of values, variables, operators, and calls to functions. Expressions need to be evaluated. If you ask Python to **print** an expression, the interpreter **evaluates** the expression and displays the result.

An expression is a combination of operators and operands that is interpreted to produce some other value. In any programming language, an expression is evaluated as per the precedence of its operators. So that if there is more than one operator in an expression, their precedence decides which operation will be performed first. We have many different types of expressions in Python. Let's discuss all types along with some exemplar codes :

1. Constant Expressions: These are the expressions that have constant values only.

Example:

```
# Constant Expressions
x = 15 + 1.3

print(x)
```

Output

```
16.3
```

2. Arithmetic Expressions: An arithmetic expression is a combination of numeric values, operators, and sometimes parenthesis. The result of this type of expression is also a numeric value. The operators used in these expressions are arithmetic operators like addition, subtraction, etc. Here are some arithmetic operators in Python:

Operators	Syntax	Functioning
+	$x + y$	Addition
-	$x - y$	Subtraction
*	$x * y$	Multiplication
/	x / y	Division
//	$x // y$	Quotient
%	$x \% y$	Remainder
**	$x ** y$	Exponentiation

Let's see an exemplar code of arithmetic expressions in Python :

```
# Arithmetic Expressions
```

```
x = 40
```

```
y = 12
```

```
add = x + y
```

```
sub = x - y
```

```
pro = x * y
```

```
div = x / y
```

```
print(add)
```

```
print(sub)
```

```
print(pro)
```

```
print(div)
```

Output

```
52
28
480
3.3333333333333335
```

Input and Output in Python

Understanding input and output operations is fundamental to Python programming. With the `print()` function, we can display output in various formats, while the `input()` function enables interaction with users by gathering input during program execution. Taking input in Python

Python's `input()` function is used to take user input. By default, it returns the user input in form of a string.

Example:

```
name = input("Enter your name: ")
print("Hello,", name, "! Welcome!")
```

Output

```
Enter your name: GeeksforGeeks
Hello, GeeksforGeeks ! Welcome!
```

The code prompts the user to input their name, stores it in the variable "name" and then prints a greeting message addressing the user by their entered name.

Printing Output using `print()` in Python

At its core, printing output in Python is straightforward, thanks to the `print()` function. This function allows us to display text, variables and expressions on the console. Let's begin with the basic usage of the `print()` function:

In this example, "Hello, World!" is a string literal enclosed within double quotes. When executed, this statement will output the text to the console.

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

Output

```
Hello, World!
```

Printing Variables

We can use the `print()` function to print single and multiple variables. We can print multiple variables by separating them with commas. **Example:**

```
# Single variable
s = "Bob"
print(s)

# Multiple Variables
s = "Alice"
age = 25
city = "New York"
print(s, age, city)
```

Output

```
Bob
Alice 25 New York
```

Range() function:

The `range()` function in Python generates an immutable sequence of integers, which is commonly used for looping a specific number of times. A `range` object is memory-efficient because it generates numbers on demand, rather than storing the entire sequence in memory.


Syntax

The `range()` function can be called with one, two, or three integer arguments:

- `range(stop)` : Generates a sequence from `0` up to (but not including) `stop` . The step is `1` by default.
- `range(start, stop)` : Generates a sequence from `start` up to (but not including) `stop` . The step is `1` by default.
- `range(start, stop, step)` : Generates a sequence from `start` up to (but not including) `stop` , incrementing or decrementing by `step` .

Examples

`range(stop)`

If you only provide one argument, the sequence starts at 0. 


python

```
# Create a sequence of numbers from 0 to 4  
for i in range(5):  
    print(i)
```

Output:

```
0  
1  
2  
3  
4
```

`range(start, stop)`

Use two arguments to specify a different starting point. 


python

```
# Create a sequence of numbers from 3 to 6  
for i in range(3, 7):  
    print(i)
```

Output:

```
3  
4  
5  
6
```

`range(start, stop, step)`

The third argument changes the increment or decrement between numbers. 


python

```
# Create a sequence of even numbers from 0 to 8
for i in range(0, 10, 2):
    print(i)
```

Output:

```
0
2
4
6
8
```

Counting backward with a negative step

To create a descending sequence, use a negative `step` value. 

python

```
# Count down from 5 to 1
for i in range(5, 0, -1):
    print(i)
```

Output:

```
5
4
3
2
1
```

Key takeaways

- **Exclusive stop value:** The number specified as `stop` is never included in the sequence. For example, `range(5)` generates numbers from 0 up to 4.
- **Integers only:** `range()` only works with integer arguments. You cannot use floating-point numbers.
- **Zero step is an error:** The `step` value cannot be `0`. A `ValueError` will be raised if you try to use it.
- **Not a list:** `range()` returns a `range` object, not a list of numbers. To get a list, you can convert it explicitly with `list()`, for example, `list(range(5))`

Operators in Python:

In Python, operators are special symbols or keywords that perform specific operations on one or more values or variables, known as operands. They are fundamental to manipulating data, performing calculations, making comparisons, and controlling program flow.

Python divides the operators in the following groups:

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	$x = 5$	$x = 5$
+=	$x += 3$	$x = x + 3$
-=	$x -= 3$	$x = x - 3$
*=	$x *= 3$	$x = x * 3$
/=	$x /= 3$	$x = x / 3$
%=	$x \% = 3$	$x = x \% 3$
//=	$x //= 3$	$x = x // 3$
**=	$x ** = 3$	$x = x ** 3$

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
	OR	Sets each bit to 1 if one of two bits is 1	x y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x

Python Code:

Program to demonstrate all types of operators in Python

print("----- Arithmetic Operators -----")

a = 15

b = 4

print("a + b =", a + b) # Addition

print("a - b =", a - b) # Subtraction

print("a * b =", a * b) # Multiplication

print("a / b =", a / b) # Division (float)

print("a % b =", a % b) # Modulus (remainder)

print("a // b =", a // b) # Floor division (integer division)

print("a ** b =", a ** b) # Exponent (power)

print("\n----- Assignment Operators -----")

Single value assignment

x = 10

print("Single assignment: x =", x)

Multi-value assignment

p, q, r = 1, 2, 3

print("Multiple assignment: p =", p, ", q =", q, ", r =", r)

Compound assignment

x = 10

print("\nInitial x =", x)

x += 5 # x = x + 5

print("After x += 5 ->", x)

```
x -= 3 # x = x - 3
print("After x -= 3 ->", x)
x *= 2 # x = x * 2
print("After x *= 2 ->", x)
x /= 4 # x = x / 4
print("After x /= 4 ->", x)
x %= 3 # x = x % 3
print("After x %= 3 ->", x)
x **= 2 # x = x ** 2
print("After x **= 2 ->", x)
x //= 2 # x = x // 2
print("After x //= 2 ->", x)
```

print("\n----- Relational Operators -----")

```
m = 10
n = 20
print("m == n :", m == n) # Equal to
print("m != n :", m != n) # Not equal to
print("m > n :", m > n) # Greater than
print("m < n :", m < n) # Less than
print("m >= n :", m >= n) # Greater than or equal
print("m <= n :", m <= n) # Less than or equal
```

print("\n----- Logical Operators -----")

```
x = True
y = False
print("x and y :", x and y) # Logical AND
print("x or y :", x or y) # Logical OR
print("not x :", not x) # Logical NOT
print("not y :", not y)
```

Output:

```
----- Arithmetic Operators -----
```

```
a + b = 19
```

```
a - b = 11
```

```
a * b = 60
```

```
a / b = 3.75
```

```
a % b = 3
```

```
a // b = 3
```

```
a ** b = 50625
```

```
----- Assignment Operators -----
```

```
Single assignment: x = 10
```

```
Multiple assignment: p = 1 , q = 2 , r = 3
```

```
Initial x = 10
```

```
After x += 5 -> 15
```

```
After x -= 3 -> 12
```

```
After x *= 2 -> 24
```

```
After x /= 4 -> 6.0
```

```
After x %= 3 -> 0.0
```

```
After x **= 2 -> 0.0
```

```
After x //= 2 -> 0.0
```

```
----- Relational Operators -----
```

```
m == n : False
```

```
m != n : True
```

```
m > n : False
```

```
m < n : True
```

```
m >= n : False
```

```
m <= n : True
```

```
----- Logical Operators -----
```

```
x and y : False
```

```
x or y : True
```

```
not x : False
```

```
not y : True
```