LEVEL 1 – Getting Started With Python

1. What is Python?

Python is a simple and powerful programming language that people all over the world use to build amazing things.
It's called a *high-level and human-friendly language* because its words and symbols are easy to read and understand — just like English!

Big companies such as **Google, NASA, Meta, Microsoft, Netflix, and Tesla (for Autopilot AI)** use Python to create websites, apps, data tools, and even artificial intelligence systems.
Many top universities and institutes like **IITs and MIT** also use Python to teach students and develop new technologies.

That's why Python is often called the **language of the future** — it's easy to learn, fun to use, and powerful enough to do almost anything!

High-Level Language (Python, Java, C#)

A **high-level programming language** is designed to be easy for humans to read, write, and understand. It allows programmers to focus on solving problems using logical steps rather than worrying about complex hardware instructions.

Key Features

- Simple and readable — looks similar to English sentences
- Easy to learn and understand, even for beginners
- Automatically handles memory management
- Focuses on problem-solving and logic instead of hardware details

Example

```
print("Hello")
```

This program displays the word *Hello* on the screen.

Main Characteristics

- Syntax resembles plain English
- Requires fewer lines of code to achieve tasks
- Easier to debug and maintain

- Automatically manages system memory
- Emphasizes logic and ideas rather than machine operations

Common Uses

- Web and app development
- Artificial Intelligence and Machine Learning
- Data analysis and visualization
- Game development
- Robotics (software control and automation)

Analogy

Using a high-level language is like **talking to a person in simple English** — for example, saying *"Turn on the fan."* The computer understands your instruction clearly without needing detailed hardware-level commands.

Low-Level Language (Assembly, Machine Code)

A **low-level programming language** is much closer to the computer's hardware. It provides very little abstraction, meaning the programmer must give detailed instructions that directly control how the computer's hardware works.

These languages are powerful and fast but **difficult for humans to read, write, and understand**.

Key Features

- Extremely fast execution speed
- Requires manual memory management
- Hardware-specific (designed for a particular type of processor)
- Complex and error-prone to write

Examples

Assembly Language

```
MOV AX, 09

ADD BX, AX
```

Machine Language (Binary)

```
10110001 11001010
```

Used For

- Microcontrollers and embedded systems
- Operating system kernels
- Robotics hardware programming
- Device drivers and performance-critical software

Main Characteristics

- Very fast and efficient
- Gives full control over hardware components
- Hard to learn and understand
- Time-consuming to develop and debug
- Not portable — programs depend on specific hardware

Analogy

Using a low-level language is like **giving exact technical instructions to a robot**, for example:
"Rotate motor 1 by 15° at 3.3V."
You must specify every small detail for the computer to follow your command correctly.

Human-Friendly Language (Python)

A human-friendly programming language is one that looks and feels like natural human language. Python is considered the most human-friendly language because it uses simple words, minimal symbols, and clean indentation — making it easy for both students and professionals to learn.

Features

- Very simple syntax
- Minimal use of symbols
- Indentation makes code clean and readable
- Perfect for students and professionals

Python Example

```
if temperature > 30:
```

```
    print("It's hot outside!")
```

Why Python is Human-Friendly

- Uses simple English words such as *if*, *for*, *print*, *while*
- No confusing symbols or punctuation
- No need to handle memory manually
- No complex syntax or long code
- Indentation keeps programs neat and structured
- Lets you focus on logic and creativity instead of rules

Common Uses

- Artificial Intelligence (AI) and Data Science
- Automation and Robotics
- Web and App Development
- Education and Research

Analogy

Using Python is like talking to a younger sibling in simple instructions:
"If it rains, take the umbrella."
The computer understands your logic clearly without extra complexity.

Python is Famous Because

- It looks and reads like English
- Very easy to learn and use
- Shortest code for complex work
- Works smoothly on Windows, Mac, and Linux
- Connects with AI, Robots, IoT, Databases, Web, and Apps

Real-Life Analogy – Python as Human Language

Think of Python as talking to a very intelligent robot:

- If you speak clearly → it understands

- If your grammar is wrong → it gets confused

Example:
Correct: *"Robot, pick up the bottle."*
Incorrect: *"Pick bottle robot the."*

Python behaves the same way —

- *Correct syntax works perfectly,* but *wrong syntax gives an error***.**

2. Installing Python / Jupyter / VS Code

2.1 Installing Python

Go to: **python.org → Downloads → Install**

This includes:

✓ Python Interpreter (brain)
✓ IDLE editor
✓ PIP package manager

Real-life Connection:

Python is like installing the "language pack" your computer needs to understand your commands.

2.2 Installing Jupyter Notebook

Jupyter is used for:

- AI & ML
- Data Science
- Step-by-step coding
- Writing code + notes together

Install via:

```
pip install notebook

jupyter notebook
```

2.3 Installing VS Code

VS Code is the **professional choice**.

Benefits:

✓ Autocomplete
✓ Debugging
✓ Extensions
✓ GitHub integration
✓ Project management

Used by developers worldwide.

3. Compiler vs Interpreter

3.1 Why Do We Need Them?

When we write Python code, we use words and symbols that humans can read — but computers only understand **binary (0s and 1s)**.
To make the computer understand our instructions, the code must be **translated into machine language**.
This translation is done by special programs called a **Compiler** or an **Interpreter**.

3.2 What Is a Compiler?

A **compiler** translates the **entire program at once** before running it.
If there are any errors, it shows them all together after checking the whole code.
Once compiled, it creates an **executable file** that can run many times without translating again.

Example Languages: C, C++, Java

Analogy:
A compiler is like a translator who reads an entire book, translates it completely, and then gives you the finished version to read anytime.

3.3 What Is an Interpreter?

An **interpreter** translates and runs the program **line by line**.
It reads one instruction, converts it into machine language, executes it, and then moves to the next line.
If an error appears, it stops immediately so you can fix it.

Python uses an interpreter.

Example Languages: Python, JavaScript, Ruby

Analogy:
An interpreter is like a live translator who listens to you and translates each sentence as you speak.

3.4 Why Python Uses an Interpreter

Python focuses on **simplicity** and **interactivity**.
By using an interpreter, you can:

- Run one line at a time
- See instant results
- Find and fix errors easily
- Learn and experiment quickly

This is why Python is considered one of the **most beginner-friendly programming languages**.

4. Running Python – Interactive Mode vs Script Mode

Interactive Mode

You interact directly with Python like a chat conversation.

Example:

```
>>> 10 + 5

15

>>> print("Hello")

Hello
```

✓ Best for testing small pieces
✓ Instant results

Script Mode

Write code in a .py file.

Example: hello.py

```
print("Welcome to Python!")
```

Run the script:

```
python hello.py
```

✓ Best for apps
✓ Best for projects
✓ Best for assignments

5. Basic Syntax Rules (COMPLETE THEORY)

Syntax = Grammar of Python.

Python is clean and human-like, but strict.

Rule 1: Indentation (VERY IMPORTANT)

Python uses **indentation** to define blocks of code.

❌ **WRONG**

```
if True:
print("Hello")
```

✅ **CORRECT**

```
if True:
    print("Hello")
```

- Recommended: **4 spaces**
- **Why Indentation Matters**

  Imagine writing answers OUTSIDE the exam box → teacher won't check.
  Python won't run code that's "outside" the block.

Real-Life Analogy

- Indentation = Paragraphs in English
- If everything is in one line, → confusing
- Proper spacing → readable

Rule 2: Case Sensitivity

Python differentiates uppercase and lowercase:

name ≠ Name ≠ NAME

Examples:

❌ Wrong

```
Print("Hello")
```

✅ Correct

```
print("Hello")
```

Rule 3: Keywords Cannot Be Variable Names

Keywords = reserved words → Python's own special vocabulary.

Examples:

if, else, for, while, True, False, None, import, class, def, return

❌ Wrong

```
for = 5
```

✅ Correct

```
loop_count = 5
```

6. Comments, Indentation, Keywords — COMPLETE EXPLANATION

6.1 Comments

Comments are notes written inside your Python code that the interpreter completely ignores. They are used to explain the logic, improve readability, and help others (and yourself) understand what the code does.

Comments are especially useful for:

- Explaining complex logic or calculations
- Temporarily disabling lines of code during debugging
- Adding author names, dates, or version details

1. Single-Line Comments

- A single-line comment begins with the # symbol.
- Everything written after # on that line is ignored by Python.

Example:

```
# This program calculates the area of a circle

radius = 7

area = 3.14 * radius ** 2

print("Area:", area)
```

2. Multi-Line Comments

When you need to write longer explanations or documentation, use triple quotes (""" """ or ''' ''').

Example:

```
"""
This program demonstrates
how to use multi-line comments in Python.
Author: Diwakar Kumar
"""
print("Learning Comments in Python")
```

Real-Life Analogy

Think of comments as sticky notes inside your notebook — they help you remember why you wrote something, but they don't affect the actual content or answers.

6.2 Indentation

Indentation refers to the spaces at the beginning of a line in Python code.
Unlike other languages that use braces {}, Python uses indentation to define code blocks.

Example:

```
if True:
  print("This is properly indented!")
```

Incorrect Example:

```
if True:
print("This will cause an error")  # No indentation!
```

Rules for Indentation

- Always use 4 spaces per indentation level.
- Tabs and spaces should not be mixed.
- All statements within a block must have the same indentation.

Why Indentation Matters:
It defines the structure of your program. Without proper indentation, Python will raise an IndentationError.

Analogy:
Indentation in Python is like writing answers neatly inside boxes in an exam —
if you write outside the box, it won't be accepted.

6.3 Keywords

Definition:
Keywords are special reserved words in Python that have predefined meanings.
They form the core syntax of the language and cannot be used as variable names or identifiers.

Example:

 # Wrong

```
for = 10  # ❌ Error: 'for' is a reserved keyword
```

 # Correct

```
count = 10  # ✅ Works perfectly
```

Full List of Python Keywords (for Level 1)

False	**await**	**else**	**import**	**pass**
None	**break**	**except**	**in**	**raise**
True	**class**	**finally**	**is**	**return**
and	**continue**	**for**	**lambda**	**try**
as	**def**	**from**	**nonlocal**	**while**
assert	**del**	**global**	**not**	**with**
async	**elif**	**if**	**or**	**yield**

These are fixed by Python and cannot be redefined.

✓ **Use the following command in Python to view the current list of keywords:**

```
import keyword
```

```
print(keyword.kwlist)
```

6.4 Identifiers

Identifiers are names given to elements in your Python program.
They help identify variables, functions, classes, or modules uniquely.

Example:

```
name = "Aarav"

total_marks = 92
```

Here, name and total_marks are identifiers.

Rules for Naming Identifiers

1. Must begin with a letter (A–Z or a–z) or an underscore (_).
2. Cannot begin with a number.
3. Cannot include special symbols like @, #, %, -, space.
4. Keywords cannot be used as identifiers.
5. Python is case-sensitive (Name, name, and NAME are all different).

Examples

Valid Identifiers:

```
student_name

totalAmount

age

is_present
```

Invalid Identifiers:

```
2value    # starts with a number

total-amt  # contains a hyphen

for      # reserved keyword

student name  # contains a space
```

7. print() Function & Escape Sequences

7.1 Understanding the print() Function

The print() function is one of the most commonly used commands in Python.
It displays output on the screen — helping you test, explain, or debug your code.

Example:

```
print("Hello Python!")
```

Output:

Hello Python!

Here,

- print() is a built-in Python function.
- The text inside the quotes " " is called a **string**.

Using Variables with print()

Variables can also be displayed using the print() function.

Example:

```
name = "Riya"
print("Welcome", name)
```

Output:

Welcome Riya

Note: Python automatically adds a space between multiple arguments in print() when separated by commas.

7.2 Escape Sequences

Escape sequences are **special symbols** that help you format your text output in creative ways. They begin with a **backslash (\)** and perform specific actions inside strings.

Escape Code	Meaning	Example Output
\n	New line	Moves text to next line

Escape Code	Meaning	Example Output
\t	Tab space	Adds horizontal spacing
\"	Double quote	Displays quotes inside quotes
\'	Single quote	Displays apostrophes
\\	Backslash	Displays a single backslash
\b	Backspace	Removes last character
\r	Carriage return	Overwrites the current line

Example:

```
print("Hello\nWorld")
print("Name:\tDiwakar")
print("He said \"Hi\"")
```

Output:

```
Hello
World
Name:   Diwakar
He said "Hi"
```

7.3 Real-Life Connection

- \n is like pressing the **Enter key** on your keyboard.
- \t is like hitting the **Tab key** to align text neatly.
- \" helps you include quotes inside strings like:
  → *He said "Hi"*
- \\ is used for **file paths** like C:\\Users\\Riya\\Documents.

8. Debugging Basics

8.1 What Is Debugging?

Debugging means finding and fixing errors in your code.
It's a key skill that turns confusion into clarity and mistakes into learning.

Even expert programmers make mistakes — **debugging helps you learn how to think logically** and fix problems step by step.

8.2 Common Beginner Errors

Type of Error	Example	What Happens
Missing quote	print("Hello)	Causes SyntaxError

Wrong indentation	<pre>if True:
	print("Hi")</pre>
		Python shows IndentationError

Wrong keyword	If 5 > 3:	Python is case-sensitive, should be if

8.3 Tips for Debugging

- Read the **error message carefully** — it usually tells you the line number.
- Check for missing symbols like ", :, or ().
- Make sure indentation (spacing) is consistent.
- Fix one issue at a time, then re-run the code.
- Practice reading your code out loud — it often reveals logic errors.

8.4 Real-Life Analogy

Debugging is like proofreading an essay.
You read line by line, fix typos, and improve clarity until everything works perfectly.