* **Introduction to Python**

**Introduction to Python and its Features (simple, high-level, interpreted language).**

Python is a **simple, high-level, and interpreted** programming language widely used for various applications, including web development, data science, artificial intelligence, and automation. Created by **Guido van Rossum** and first released in **1991**, Python is known for its **readability and ease of use**, making it an excellent choice for beginners and experienced developers.

**Simple and Easy to Learn**

* Python uses a clean and readable syntax similar to the English language, making it easy to learn and understand.

**High-Level Language**

* Python allows developers to write programs without worrying about low-level details like memory management.

**Interpreted Language**

* Python code is executed **line by line** by the Python interpreter, making debugging easier.

**History and evolution of Python.**

Python is a widely used programming language known for its **simplicity, readability, and versatility**. It was created by **Guido van Rossum** in the late 1980s and officially released in **1991**. Over the years, Python has evolved through multiple versions, adding powerful features while maintaining its ease of use.

**1. The Birth of Python (1980s - 1991)**

* Guido van Rossum, a Dutch programmer, started working on Python in **1985** at the **Centrum Wickenden & Informatica (CWI)** in the Netherlands.
* He wanted to create a language that was easy to read, powerful, and free from unnecessary complexity.
* Python was influenced by the **ABC programming language**, but it was designed to be more extensible and practical.
* The first official version, **Python 1.0**, was released on **February 20, 1991**.

**2. Python 2.x Series (2000 - 2010)**

* **Python 2.0** was released on **October 16, 2000** with improvements like:
  + List comprehensions
  + Garbage collection with reference counting
* Despite its popularity, Python 2 had **backward compatibility issues**, making it difficult to transition to newer versions.

**3. Python 3.x Series (2008 - Present)**

* **Python 3.0** was released on **December 3, 2008**, bringing major improvements:
  + Better Unicode support
  + Improved integer division
  + Print statement changed to print() function
* **Python 2 was officially discontinued on January 1, 2020**, marking a complete shift to Python 3.

**4. Modern Python (2010 - Present)**

* Python has grown significantly with new versions, including:
  + **Python 3.6 (2016):** f-strings for better string formatting
  + **Python 3.7 (2018):** Data classes and performance improvements
  + **Python 3.8 (2019):** Walrus operator (:=) for assignment expressions
  + **Python 3.9 (2020):** Dictionary union operations
  + **Python 3.10 (2021):** Pattern matching and better error messages
  + **Python 3.11 (2022):** Major speed improvements
  + **Python 3.12 (2023):** Further performance enhancements and syntax improvements

**Advantages of using Python over other programming languages.**

Python is one of the most popular programming languages due to its **simplicity, versatility, and efficiency**.

Compared to other languages like **C, C++, Java, and JavaScript**, Python offers several advantages, making it a preferred choice for developers across various domains.

**1. Simple and Easy to Learn**

* Python has a **clean and readable syntax** similar to English, making it easier to learn than C++ or Java.
* It requires fewer lines of code to accomplish tasks compared to other languages.

**2. High-Level and Interpreted**

* Python is a **high-level language**, meaning developers do not need to manage low-level system details like memory allocation.
* It is an **interpreted language**, meaning the code is executed line-by-line, making debugging easier.

**3. Cross-Platform Compatibility**

* Python is **platform-independent**, meaning code written on one operating system (Windows, macOS, Linux) can run on another without modification.

**4. Extensive Standard Library**

* Python comes with a **rich set of built-in libraries** for handling file operations, networking, data science, web development, and more.
* Examples include **NumPy** (scientific computing), **Pandas** (data analysis), **TensorFlow** (machine learning), and **Django** (web development).

**5. Dynamically Typed Language**

* Unlike C++ or Java, Python does not require explicit declaration of variable types. The interpreter assigns data types dynamically, reducing code complexity.

**6. Object-Oriented and Procedural Support**

* Python supports both **object-oriented programming (OOP)** and **procedural programming**, providing flexibility to developers.

**7. Large and Active Community**

* Python has a **huge global community** that continuously contributes to its development, providing extensive documentation, tutorials, and support forums.

**8. Strong Support for AI, Machine Learning, and Data Science**

* Python is the **most preferred language** for artificial intelligence, data science, and machine learning due to its powerful libraries like **Scikit-learn, TensorFlow, and PyTorch**.

**9. Automation and Scripting Capabilities**

* Python is widely used for **automation and scripting**, making repetitive tasks (like file handling, web scraping, and testing) more efficient.

**10. Integration with Other Languages**

* Python can be easily integrated with **C, C++, Java, and .NET** using libraries like **Cython** and **Jython**, making it highly adaptable.

**Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).**

1. Install Python

**Option 1: Install Python from the Official Website**

1. Go to the [official Python website](https://www.python.org/downloads/).
2. Download the latest Python version for your operating system (Windows, macOS, or Linux).
3. Run the installer and **check the box** that says **"Add Python to PATH"** before installing.
4. Verify the installation by opening a terminal or command prompt and running:

python –version

**Option 2: Install Python via Anaconda**

If you prefer a bundled package with data science tools, install **Anaconda** instead of standard Python.

1. Download Anaconda from here.
2. Install it by following the on-screen instructions.
3. After installation, open **Anaconda Navigator** or use **Anaconda Prompt** to manage environments.

To check if Anaconda is installed, run:

conda –version

**2. Choose a Development Environment**

You can use **PyCharm, VS Code, or Jupyter Notebook** depending on your preference.

**Option 1: PyCharm (Best for Large Projects)**

1. Download PyCharm from [JetBrains](https://www.jetbrains.com/pycharm/download/).
2. Install and open it.
3. Create a new project and configure the Python interpreter (Settings > Project > Python Interpreter).
4. Choose a virtual environment or Anaconda if installed.

**Option 2: VS Code (Best for Versatility)**

1. Download **VS Code** from [here](https://code.visualstudio.com/).
2. Install the **Python extension** from the Extensions Marketplace.
3. Open VS Code, create a .py file, and select your Python interpreter (Ctrl + Shift + P > Python: Select Interpreter).

**Option 3: Jupyter Notebook (Best for Data Science)**

1. Install Jupyter using pip or Conda:

pip install jupyter

1. Launch Jupyter by running:

jupyter notebook

**3. Verify Everything**

* Run a simple Python script in your chosen editor:

print("Hello, Python!")

**Writing and executing your first Python program.**

**1. Writing Your First Python Program**

**Option 1: Using a Text Editor (VS Code, PyCharm, or Notepad++)**

1. **Open your preferred text editor** (VS Code, PyCharm, or any other).
2. **Create a new file** and name it hello.py.
3. **Write the following Python code**:

print("Hello, Python!")

1. **Save the file**.

**2. Running Your Python Program**

**Option 1: Using the Terminal or Command Prompt**

1. Open a terminal (**Command Prompt, PowerShell, or Bash**).

cd path/to/your/python/script

1. Run the script by typing:

python hello.py

**Option 2: Running in IDLE (Python’s Built-in IDE)**

1. Open **IDLE** (comes installed with Python).
2. Click **File > Open**, select your hello.py file.
3. Click **Run > Run Module (F5)**.

**Option 3: Running in VS Code**

1. Open VS Code and open your hello.py file.
2. Click the **Run Python File** button or press Ctrl + Shift + P, then select **Run Python File in Terminal**.

**Option 4: Running in PyCharm**

1. Open PyCharm and create a **new Python project**.
2. Inside the project, create a new Python file (hello.py).
3. Write the print("Hello, Python!") code.
4. Click the **Run** button or press Shift + F10.

**3. Verifying the Output**

Once executed, you should see the following output in the terminal:

Hello, Python!

* **Programming Style**

**Understanding Python’s PEP 8 guidelines**

**Python’s PEP 8 Guidelines (Code Style Best Practices)**

PEP 8 ensures **clean, readable, and consistent** Python code.

✅ **Key Rules:**

1. **Indentation** – Use **4 spaces**, no tabs.
2. **Max Line Length** – **79 characters** (72 for docstrings).
3. **Blank Lines** – **2 lines** between top-level functions/classes.
4. **Imports** – One module per line, properly ordered.
5. **Whitespace** – No extra spaces inside parentheses or before commas.
6. **Naming Conventions** –
   * Variables & functions: **snake\_case** (my\_function)
   * Classes: **PascalCase** (MyClass)
   * Constants: **UPPER\_CASE** (PI = 3.14)
7. **Comments** – Keep them **short and meaningful**.

Following PEP 8 makes your code **more readable and professional**!

**Indentation, comments, and naming conventions in Python.**

✅ **Indentation** (for code blocks)

* Use **4 spaces per indentation level** (no tabs).
* Required in loops, functions, classes, etc.
* def greet():
* print("Hello, Python!") # Indented correctly

✅ **Comments** (for code clarity)

* **Single-line comment:** Use #
* # This is a single-line comment
* print("Hello, World!")
* **Multi-line comment:** Use triple quotes """ """ or ''' '''
* """
* This is a multi-line comment.
* It explains the code in detail.
* """

✅ **Naming Conventions** (for readability)

* **Variables & Functions:** snake\_case → my\_variable, calculate\_sum()
* **Classes:** PascalCase → MyClass
* **Constants:** UPPER\_CASE → PI = 3.14

Following these practices improves **code readability and maintainability**!

**Writing readable and maintainable code.**

**Tips for Readable & Maintainable Python Code**

**Follow PEP 8** – Proper indentation, spacing, and naming.  
**Use Meaningful Names** – Clear variable & function names.  
**Keep Functions Short** – Each function does one task.  
**Write Comments & Docstrings** – Explain complex logic.  
**Use List Comprehensions** – For cleaner loops.  
**Avoid Hardcoding** – Use constants instead.  
**Modular Code** – Use functions & classes for reusability.

Clean code is **easy to read, debug, and maintain!**

* **Core Python Concepts**

**Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets**

**Python Data Types (Quick Guide)**

**int** – Whole numbers → x = 10  
**float** – Decimals → y = 3.14  
**str** – Text → name = "Python"  
**list** – Ordered, mutable → fruits = ["apple", "banana"]  
**tuple** – Ordered, immutable → coords = (10, 20)  
**dict** – Key-value pairs → student = {"name": "John"}  
**set** – Unique, unordered → unique\_nums = {1, 2, 3}

Each type serves a **specific purpose**!

**Python variables and memory allocation.**

**Python Variables & Memory (Easy Explanation)**

**What is a Variable?**  
A variable is like a **container** that holds data.

x = 10 # x stores the number 10

**How Does Python Store Data?**

* **Stack** → Keeps track of variable names.
* **Heap** → Stores actual values (like numbers, lists, etc.).
* **Garbage Collector** → Removes unused data automatically.

**Example:**

a = [1, 2, 3] # A list is created in memory

b = a # `b` also points to the same list

del a # The list still exists because `b` is using it

Python **handles memory automatically**, so you don’t have to worry!

**Python operators: arithmetic, comparison, logical, bitwise.**

**Python Operators (Quick Guide)**

✅ **1. Arithmetic Operators** (Math operations)

+ (Addition) → x + y

- (Subtraction) → x - y

\* (Multiplication) → x \* y

/ (Division) → x / y

% (Modulus) → x % y # Remainder

\*\* (Exponentiation) → x \*\* y # Power

// (Floor Division) → x // y # Whole number division

✅ **2. Comparison Operators** (Check values)

== (Equal) → x == y

!= (Not equal) → x != y

> (Greater than) → x > y

< (Less than) → x < y

>= (Greater/equal) → x >= y

<= (Less/equal) → x <= y

✅ **3. Logical Operators** (Check conditions)

and (Both True) → x > 5 and x < 10

or (At least one True) → x > 5 or x < 3

not (Reverse condition) → not(x > 5)

✅ **4. Bitwise Operators** (Work on binary values)

& (AND) → x & y

| (OR) → x | y

^ (XOR) → x ^ y

~ (NOT) → ~x

<< (Left shift) → x << 2

>> (Right shift) → x >> 2

Python operators help **perform calculations and make decisions!**

* **Conditional Statements**

**Introduction to conditional statements: if, else, elif**

* **Conditional Statements in Python**
* Conditional statements allow **decision-making** in Python.
* ✅ **1. if Statement** (Executes if condition is true)
* x = 10
* if x > 5:
* print("x is greater than 5")
* ✅ **2. if-else Statement** (Executes one of two blocks)
* x = 3
* if x > 5:
* print("x is greater than 5")
* else:
* print("x is 5 or less")
* ✅ **3. elif Statement** (Checks multiple conditions)
* x = 10
* if x > 10:
* print("Greater than 10")
* elif x == 10:
* print("Equal to 10")
* else:
* print("Less than 10")
* **Conditional statements help control the flow of a program!**

**Nested if-else conditions.**

**Nested if-else in Python**

A **nested if-else** means an if inside another if.

✅ **Example:**

x = 10

if x > 5:

print("x is greater than 5")

if x > 8:

print("x is also greater than 8")

else:

print("x is between 5 and 8")

else:

print("x is 5 or less")

🔹 **How it Works?**

* If the **first if** is true, the inner condition is checked.
* If the first if is false, the outer else runs.

Used for **complex decision-making!**

* **Looping**

**Introduction to for and while loops.**

**Loops in Python (for & while)**

Loops are used to **repeat tasks** in Python.

✅ **1. for Loop** (Iterates over a sequence)

for i in range(5):

print(i) # Output: 0, 1, 2, 3, 4

✅ **2. while Loop** (Repeats while condition is true)

x = 0

while x < 5:

print(x)

x += 1 # Increases x to avoid infinite loop

🔹 **for → Best for fixed loops**  
🔹 **while → Best for loops with conditions**

Loops help automate **repetitive tasks!**

**How loops work in Python.**

**How Loops Work in Python**

**1. for Loop**

for i in range(3):

print(i)

**2. while Loop**

x = 0

while x < 3:

print(x)

x += 1

**Using loops with collections (lists, tuples, etc.).**

**Using Loops with Collections in Python**

✅ **1. for Loop with Lists**

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

✅ **2. for Loop with Tuples**

numbers = (1, 2, 3)

for num in numbers:

print(num)

✅ **3. for Loop with Dictionaries**

student = {"name": "John", "age": 25}

for key, value in student.items():

print(key, value)

✅ **4. for Loop with Sets**

unique\_numbers = {10, 20, 30}

for num in unique\_numbers:

print(num)

* **Functions and Methods**

**Defining and calling functions in Python.**

**Defining and Calling Functions in Python**

✅ **Defining a Function**

def greet(name):

return f"Hello, {name}!"

✅ **Calling a Function**

message = greet("Alice")

print(message)

**Function arguments (positional, keyword, default).**

**Function Arguments in Python**

**1. Positional Arguments** (Order matters)

def greet(name, age):

print(f"Hello, {name}. You are {age} years old.")

greet("Alice", 25)

**2. Keyword Arguments** (Order doesn't matter)

greet(age=25, name="Alice")

**3. Default Arguments** (Uses default if not provided)

def greet(name, age=18):

print(f"Hello, {name}. You are {age} years old.")

greet("Bob") # Uses default age = 18

Functions allow **flexible argument passing!**

**Scope of variables in Python.**

**Scope of Variables in Python**

**1. Local Scope** (Exists inside a function)

def my\_function():

x = 10

print(x) # x is accessible here

my\_function()

# print(x) # Error: x is not accessible outside the function

**2. Global Scope** (Accessible everywhere)

x = 20

def my\_function():

print(x) # Can access global variable

my\_function()

**3. Enclosing (Nonlocal) Scope** (Used in nested functions)

def outer():

y = 30

def inner():

nonlocal y

y += 5

print(y)

inner()

outer()

**4. Built-in Scope** (Python’s predefined names)

print(len("Hello")) # `len` is built-in

**Built-in methods for strings, lists, etc.**

**Built-in Methods for Strings, Lists, etc.**

**1. String Methods**

text = "hello world"

print(text.upper())

print(text.replace("world", "Python"))

**2. List Methods**

fruits = ["apple", "banana"]

fruits.append("cherry")

fruits.remove("banana")

print(fruits)

**3. Tuple Methods**

numbers = (1, 2, 3, 2)

print(numbers.count(2))

print(numbers.index(3))

**4. Dictionary Methods**

student = {"name": "Alice", "age": 25}

print(student.keys())

print(student.values())

**5. Set Methods**

unique\_numbers = {1, 2, 3}

unique\_numbers.add(4)

unique\_numbers.discard(2)

print(unique\_numbers)

* **Control Statements**

**Understanding the role of break, continue, and pass in Python loops.**

In Python loops, break, continue, and pass control the flow of execution:

1. **break** – Exits the loop immediately.
2. **continue** – Skips the current iteration and moves to the next one.
3. **pass** – A placeholder that does nothing (used for syntactic purposes).

**Example:**

for i in range(5):

if i == 2:

break # Stops the loop when i is 2

print(i)

# Output: 0, 1

for i in range(5):

if i == 2:

continue # Skips printing 2

print(i)

# Output: 0, 1, 3, 4

for i in range(5):

if i == 2:

pass # Does nothing, loop continues

print(i)

# Output: 0, 1, 2, 3, 4

break stops the loop, continue skips an iteration, and pass is just a placeholder.

**Practical Example: 1) Write a Python program to skip 'banana' in a list using the continue statement. List1 = ['apple', 'banana', 'mango']**

Here's a Python program that skips 'banana' using the continue statement:

List1 = ['apple', 'banana', 'mango']

for fruit in List1:

if fruit == 'banana':

continue # Skips 'banana'

print(fruit)

**Output:**

apple

mango

**Practical Example: 2) Write a Python program to stop the loop once 'banana' is found using the break statement.**

Here's a Python program that stops the loop once 'banana' is found using break:

List1 = ['apple', 'banana', 'mango']

for fruit in List1:

if fruit == 'banana':

break # Stops loop when 'banana' is found

print(fruit)

**Output:**

apple

* **String Manipulation**

**Understanding how to access and manipulate strings.**

Here’s the cleaned-up version without comments:

text = "Hello"

print(text[0])

print(text[-1])

print(text[1:4])

print(text[:3])

print(text[::2])

text = "hello world"

print(text.upper())

print(text.lower())

print(text.replace("world", "Python"))

print(text.split())

print(" ".join(["Hello", "Python"]))

name = "Alice"

age = 25

print(name + " is " + str(age) + " years old.")

print(f"{name} is {age} years old.")

print("Hello" in "Hello World")

print("Python" not in "Hello World")

**Basic operations: concatenation, repetition, string methods (upper(), lower(), etc.).**

str1 = "Hello"

str2 = " World"

result = str1 + str2

print(result)

text = "Python "

print(text \* 3)

text = "Hello World"

print(text.upper())

print(text.lower())

print(text.title())

print(text.strip())

print(text.replace("World", "Python"))

print(text.split())

print(" - ".join(["Hello", "Python"]))

**String slicing**

**String Slicing in Python**

text = "Hello World"

print(text[0:5])

print(text[:5])

print(text[6:])

print(text[-5:])

print(text[::2])

print(text[::-1])