Chapter 1 — Getting Started with Python

Chapter 1 — Getting Started with Python

Step 1: Write your first program (Hello World)

1. Create a folder named `python_work`.
2. Create a file named `hello_world.py` inside `python_work`.
3. Write this code in the file:

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

Step 2: Install Python 3

Windows

1. Download Python 3 and run the installer.
2. Enable **Add Python to PATH** during installation.
3. Verify in Command Prompt:

```
python --version
```

If it doesn't work:

```
py --version
```

macOS

1. Open Terminal and verify:

```
python3 --version
```

Linux

1. Open Terminal and verify:

```
python3 --version
```

Step 3: Install VS Code

1. Download and install Visual Studio Code.
2. Open VS Code.

Step 4: Set up VS Code for Python

1. Open VS Code → **Extensions**
2. Search **Python** (Microsoft) → Install
3. Open Command Palette:
    - Windows/Linux: `Ctrl+Shift+P`
    - macOS: `Cmd+Shift+P`
4. Run:

```
Python: Select Interpreter
```

5. Select your Python 3 interpreter.

Step 5: Run `hello_world.py` in VS Code

1. Open the folder `python_work` in VS Code (**File** → **Open Folder...**).
2. Open `hello_world.py`.
3. Run:
    - Click **Run Python File in Terminal**, or
    - Right-click → **Run Python File in Terminal**
4. Expected output:

```
Hello Python world!
```

Step 6: Run Python program from Terminal

Linux / macOS

```
cd ~/Desktop/python_work
ls
python3 hello_world.py
```

Windows

```
cd Desktop\python_work
dir
python hello_world.py
```

If needed:

```
py hello_world.py
```

Step 7: Run Python in the interpreter (snippets)

Start interpreter

- Linux/macOS:

```
python3
```

- Windows:

```
python
```

(or `py`)

Try a snippet

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

Exit interpreter

- Linux/macOS: `Ctrl+D` or `exit()`
- Windows: `Ctrl+Z` then Enter, or `exit()`

Step 8: Troubleshooting

- Recheck: `print`, quotes, parentheses.
- In VS Code: re-run `Python: Select Interpreter` and choose Python 3.
- Windows “python not recognized”: reinstall with **Add Python to PATH** or use `py`.

Make an account on GitHub

1. Open **github.com**
2. Click **Sign up**
3. Enter: **email** → **password** → **username**
4. Verify your email (GitHub will send a verification mail)
5. After login, keep your **username** noted (you’ll use it later in VS Code)

---

2) Create a repo for “Python Programming”

1. In GitHub (top-right), click **+** → **New repository**
2. Fill:
    - **Repository name:** `python-programming` (or `Python-Programming`)

   - **Description (optional):** “Python Programming practice and notes”
   - Choose **Public** (or **Private**)
3. Tick:
   - ✅ **Add a README file**
   - (Optional) ✅ **Add .gitignore** → choose **Python**
4. Click **Create repository**

---

3) Download GitHub Desktop

1. Open **desktop.github.com**
2. Download and install **GitHub Desktop**
3. Open GitHub Desktop → **Sign in to GitHub.com**
4. Complete the browser login/authorization, then return to GitHub Desktop

---

4) Link VS Code to GitHub using web authorization

1. Open **VS Code**
2. Press **Ctrl + Shift + P**
3. Type and select: **GitHub: Sign in**
4. Choose **Sign in with browser**
5. Your browser opens GitHub → click **Authorize** (approve VS Code access)
6. After success, VS Code will show you are signed in (Accounts icon/profile in the bottom-left or top-right depending on layout)

That’s it—your GitHub account is now connected to VS Code via web authorization, and your “Python Programming” repo is ready to use.

15/01/2026 11:44

Lecture 1

Introduction to Python

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python’s simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed."

Python is a high-level, general-purpose programming language designed with an emphasis on readability and developer productivity. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is widely adopted in both academic and industrial environments due to its clean syntax, extensive standard library, and strong third-party ecosystem.

Python is commonly used in the following domains:

- Software Development and Automation
- Web Development (server-side applications and APIs)
- Data Science and Analytics
- Artificial Intelligence and Machine Learning
- Cloud Engineering and DevOps Tooling
- Scientific and Research Computing

Key characteristics of Python include:

- Interpreted execution model
- Dynamic typing
- Automatic memory management
- Cross-platform compatibility
- Large ecosystem of libraries and frameworks

Working with Python

Working with Python typically involves installing the Python interpreter and using an editor or integrated development environment (IDE) to write and execute programs.

Python Interpreter

The Python interpreter is responsible for executing Python code. It can be invoked from the command line to run scripts or to start an interactive session.

Typical commands include:

- Checking version:

```
python --version
```

- Running a Python script:

```
python filename.py
```

Development Tools

Python can be used with a variety of editors and IDEs. Common tools include:

- Visual Studio Code (VS Code)
- PyCharm
- Jupyter Notebook
- IDLE (bundled with Python)

The choice of tool generally depends on the type of work being performed (software development, scripting, data analysis, etc.).

Interactive Mode and Scripting Mode

Python programs can be executed in two primary modes: interactive mode and scripting mode.

Interactive Mode

Interactive mode allows users to execute Python commands line-by-line and immediately view results. This mode is useful for quick testing, experimentation, and learning.

To start interactive mode:

```
python
```

Example:

```
>>> 10 + 20
30
>>> "Hello" + " Python"
'Hello Python'
```

Common use cases

- Testing small code snippets
- Performing quick calculations
- Exploring language behavior
- Debugging small logic blocks

Limitations

- Not suitable for building complete programs
- Code is not preserved unless manually saved
- Difficult to manage longer workflows

Scripting Mode

Scripting mode involves writing Python code in a file (typically with `.py` extension) and executing the complete program.

Example file (`hello.py`):

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

Execution:

```
python hello.py
```

Benefits of scripting mode

- Suitable for full-scale programs and projects
- Code can be version-controlled and reused
- Easier to maintain structured logic
- Supports modular and organized development

Dynamic Types in Python

Python is dynamically typed, meaning variable types are determined at runtime rather than being explicitly declared by the programmer. A variable in Python is a reference to an object, and the object itself has a datatype.

Example:

```
x = 10          # integer
x = "Python"    # string
x = 4.5         # floating point
```

At each reassignment, the variable points to a new object with a different datatype.

Type Inspection

The datatype of a value can be inspected using the `type()` function:

```
a = 100
print(type(a))   # <class 'int'>
```

Dynamic typing increases flexibility but requires careful handling in larger programs to avoid runtime type-related errors.

Mutable and Immutable Data Types

Python objects are categorized based on whether their state can be modified after creation.

Immutable Data Types

Immutable objects cannot be modified once created. If a change is required, Python creates a new object in memory.

Common immutable types:

- `int`
- `float`
- `bool`
- `str`
- `tuple`
- `frozenset`

Example (string immutability):

```python
s = "Python"
# s[0] = "J"   # Error: strings do not support item assignment
```

To “modify” a string, a new string must be created:

```python
s = "Jython"
```

Mutable Data Types

Mutable objects allow in-place modification without changing the object identity.

Common mutable types:

- `list`
- `dict`
- `set`

Example:

```python
numbers = [10, 20, 30]
numbers[0] = 99
print(numbers)   # [99, 20, 30]
```

Memory Identity Example

Immutable type:

```
x = 5
print(id(x))
x = x + 1
print(id(x))     # memory identity changes
```

Mutable type:

```
a = [1, 2]
print(id(a))
a.append(3)
print(id(a))     # memory identity remains the same
```

Basic Syntax

Python syntax is designed to be concise and readable. The language relies on indentation for block structure and avoids excessive punctuation.

Indentation-Based Blocks

Python uses indentation instead of braces `{}` to define code blocks.

Correct:

```
if 10 > 5:
    print("Condition is True")
```

Incorrect:

```
if 10 > 5:
print("Condition is True")   # IndentationError
```

Statements and Case Sensitivity

Python does not require semicolons at the end of statements:

```
x = 10
y = 20
print(x + y)
```

Python is case-sensitive:

```
name = "Aman"
Name = "Raj"
print(name)  # Aman
print(Name)  # Raj
```

Multi-Line Statements

Long statements may be split across multiple lines using parentheses:

```
total = (10 + 20 + 30 +
         40 + 50)
```

Comments in Python

Comments are used to document code and improve readability. They are ignored during execution.

Single-Line Comments

A single-line comment begins with `#`:

```
# This is a comment
print("Python")
```

Multi-Line Documentation (Docstrings)

Triple quotes are commonly used for multi-line documentation or descriptive text:

```
"""
This block is often used as documentation
or explanatory text inside programs.
"""
```

String Values in Python

A string is a sequence of characters enclosed in quotes. Strings are immutable in Python and support extensive operations such as indexing, slicing, searching, and formatting.

Creating Strings

Strings can be created using:

- single quotes `'...'`
- double quotes `"..."`
- triple quotes `"""..."""` for multi-line strings

Examples:

```
a = 'Hello'
b = "Python"
c = """This is a
multi-line string."""
```

Indexing and Negative Indexing

Strings are indexed starting at 0:

```
s = "Python"
print(s[0])   # P
print(s[3])   # h
```

Negative indexing counts from the end:

```
print(s[-1])  # n
print(s[-2])  # o
```

Slicing

Slicing extracts a substring:

```
s = "Python"
print(s[0:4])  # Pyth
print(s[2:])   # thon
print(s[:3])   # Pyt
print(s[::2])  # Pto
```

Common String Operations

Concatenation:

```
a = "Hello"
b = "World"
print(a + " " + b)
```

Repetition:

```
print("Hi " * 3)
```

Length:

```
print(len("Python"))  # 6
```

Common String Methods

```
s = "python programming"

print(s.upper())        # PYTHON PROGRAMMING
print(s.lower())        # python programming
print(s.title())        # Python Programming
print(s.replace("python", "java"))
print(s.split())        # ['python', 'programming']
```

Membership Checking

```
s = "Python"
print("Py" in s)      # True
print("Java" in s)    # False
```

String Formatting

f-string formatting (recommended)

```
name = "Vibhu"
age = 35
print(f"Name: {name}, Age: {age}")
```

format() method

```
print("Name: {}, Age: {}".format(name, age))
```

Conclusion

Python provides a simple yet powerful environment for building applications across a wide range of domains. Understanding execution modes, dynamic typing, mutability behavior, and core syntax rules forms the foundation for effective programming. Mastery of strings and documentation practices further improves code quality and maintainability in real-world development.