CH2. Basics of Python

-Introduction to python

Python is a **high-level, interpreted, general-purpose programming language**. It was created by **Guido van Rossum in 1991** and is maintained by the Python Software Foundation.

It is popular because it emphasizes **readability and simplicity**, allowing developers to focus on solving problems rather than worrying about complex syntax.

Python supports **multiple programming paradigms**:

- **Procedural programming** (step-by-step code execution)
- **Object-Oriented Programming (OOP)** (using classes and objects)
- **Functional programming** (using functions as first-class objects, e.g., map, filter, lambda). **Features of Python**

-Python is widely used due to its powerful features:

1. **Simple & Easy to Learn**
   - Its syntax is close to English, making it beginner-friendly.
   - Example:
   - print("Hello, World!")
2. **Interpreted Language**
   - No need for compilation; code runs directly via the interpreter.
3. **Dynamically Typed**
   - No need to declare variable types.
   - x = 10   # integer
   - x = "hi" # now string
4. **Extensive Standard Library**
   - Built-in modules for math, file handling, networking, etc.
5. **Portable & Cross-Platform**
   - Runs on Windows, macOS, Linux, and even mobile/embedded devices.
6. **Object-Oriented**
   - Everything in Python is an object.
7. **Supports Integration**
   - Can integrate with C, C++, Java, .NET, etc.
8. **Large Community & Open Source**
   - Strong support and countless third-party libraries (e.g., NumPy, Django, TensorFlow).

-Applications of Python Programming

Python is one of the most versatile languages, used in almost every domain:

1. **Web Development**

   - Frameworks: Django, Flask, FastAPI
   - Example: Instagram and Pinterest use Python on the backend.

2. **Data Science & Analytics**
   - Libraries: Pandas, NumPy, Matplotlib, SciPy
   - Used for data cleaning, analysis, and visualization.
3. **Artificial Intelligence & Machine Learning**
   - Libraries: TensorFlow, PyTorch, Scikit-learn, Keras
   - Powers chatbots, recommendation engines, self-driving cars, etc.
4. **Automation & Scripting**
   - Automate repetitive tasks like file handling, web scraping (BeautifulSoup, Selenium), and system tasks.
5. **Game Development**
   - Libraries: Pygame, Panda3D
   - Example: "Civilization IV" used Python for scripting.
6. **Desktop GUI Applications**
   - Tkinter, PyQt, Kivy
   - Used to build cross-platform desktop apps.
7. **Networking & Cybersecurity**
   - Used for building network tools, penetration testing, ethical hacking scripts.
8. **Scientific & Numeric Computing**
   - Libraries: SciPy, SymPy
   - Used in research, simulations, and mathematical computations.
9. **Embedded Systems & IoT**
   - MicroPython, CircuitPython used in devices like Raspberry Pi.
10. **Finance & Business Applications**

- Risk management, algorithmic trading, financial data analysis.

Python Installation

Download Python

- Go to the official Python website: https://www.python.org/downloads/
- Choose the latest stable version (e.g., Python 3.12).
- Python automatically suggests the correct installer for your operating system.

- **Install on Windows**

1. Run the downloaded .exe file.

2. ✅ **Important**: Check the box **“Add Python to PATH”** before installing.
3. Click **Install Now**.
4. After installation, verify by typing:

```
python –version
```

Installing IDE/Code Editor

You need an editor to write Python programs:

- **IDLE** → comes with Python by default.
- **VS Code** (lightweight, widely used).
- **PyCharm** (powerful IDE for large projects).
- **Jupyter Notebook** (for Data Science).

Installing pip (Python Package Manager)

- Pip is installed by default with Python 3.4+
- Check pip version:
  - pip --version
- Example: installing a package:
- pip install numpy

2.3 Basic structure of python program

A Python program generally contains:

- **Comments**
- **Import statements (libraries/modules)**
- **Function definitions**
- **Main program logic**
- **Output**

Example with code

```
# 1. Comment: This program calculates the sum of two numbers


# 2. Import (if needed)

import math   # Importing math library (not used here, just example)


# 3. Function definition

def add_numbers(a, b):

   """This function adds two numbers and returns the result"""

   return a + b
```

```
# 4. Main program logic
if __name__ == "__main__":
    # Taking input from user
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))

    # Calling function
    result = add_numbers(num1, num2)

    # 5. Output
    print("The sum is:", result)
```

Python Comments

A comment in Python is text in your code that is ignored by the Python interpreter.

- It is used to explain code, make it readable, and provide documentation.
- Comments are only for humans, not for execution**.**

Types of Comments in Python

1. Single-Line Comment

- Written using the # symbol.
- Everything after # on that line is ignored.

Example:

```
# This is a single-line comment
print("Hello, Python!")  # This prints a message
```

2. Multi-Line Comment

Python doesn’t have a specific syntax for multi-line comments.
Two common ways:

(a) Multiple # lines

```
# This is line 1 of a comment
# This is line 2 of a comment
# This is line 3 of a comment
```

(b) Using Triple Quotes (''' or """)

Technically, triple quotes are for **multi-line strings**, but if they are not assigned to a variable, Python ignores them — so we use them as multi-line comments.

```
"""

This is a multi-line comment.

It can span multiple lines.

Useful for large explanations.

"""
```

Keywords in Python

- **Definition**: Keywords are reserved words in Python that have **special meaning** and cannot be used as variable names.
- Example keywords: if, else, for, while, True, False, None, def, class, import

Example:

```
# Using keywords

if True:

   print("This is a keyword example")
```

Output:

```
This is a keyword example
```

---

2. Identifiers in Python

- **Definition**: Names given to variables, functions, classes, etc.
- Rules:
    1. Can contain letters, digits, and underscore _
    2. Must **not** start with a digit
    3. Case-sensitive (name ≠ Name)
    4. Cannot be a keyword

Example:

```
my_var = 10       # valid identifier

Name = "Python"    # valid (case-sensitive)

# 1name = 5       # ❌ invalid (starts with digit)
```

---

3. Data Types in Python

Python is **dynamically typed** → you don't need to declare type explicitly.

Common Data Types:

- **Numeric** → int, float, complex
- **Sequence** → str, list, tuple
- **Mapping** → dict

- **Set types** → set, frozenset
- **Boolean** → True, False
- **None** → NoneType

Example:

```
x = 10          # int
y = 3.14        # float
z = "Hello"     # string
a = [1, 2, 3]   # list
b = (4, 5, 6)   # tuple
c = {"name": "John", "age": 25}  # dictionary
d = {1, 2, 3}   # set
e = True        # boolean
f = None        # NoneType

print(type(x), type(y), type(z))
```

Output:

```
<class 'int'> <class 'float'> <class 'str'>
```

---

4. Variables in Python

- **Definition**: A variable is a **named location in memory** used to store data.
- No need to declare type, Python assigns automatically.

Example:

```
name = "Alice"   # string
age = 21         # integer
height = 5.6     # float

print("Name:", name)
print("Age:", age)
print("Height:", height)
```

Output:

```
Name: Alice
Age: 21
Height: 5.6
```

---

5. Operators in Python

Operators are symbols that perform operations on variables/values.

Types of Operators:

Arithmetic Operators

+, -, *, /, % (modulus), // (floor division), ** (power)

1. a, b = 10, 3
2. print(a + b)   # 13
3. print(a - b)   # 7
4. print(a * b)   # 30
5. print(a / b)   # 3.333...
6. print(a // b) # 3
7. print(a % b)   # 1
8. print(a ** b) # 1000

Comparison Operators

==, !=, >, <, >=, <=

9. print(10 > 5)   # True
10. print(10 == 5) # False

Logical Operators

and, or, not

11. print(True and False) # False
12. print(True or False)   # True
13. print(not True)   # False

Assignment Operators

=, +=, -=, *=, /=, etc.

14. x = 5
15. x += 3   # x = x + 3
16. print(x) # 8

Bitwise Operators

- &, |, ^, ~, <<, >>

17. print(5 & 3) # 1 (AND)
18. print(5 | 3) # 7 (OR)

Membership Operators

in, not in

19. nums = [1, 2, 3, 4]

```
print(3 in nums)     # True
print(5 not in nums) # True
```

Type Conversation

What is Type Conversion?

In Python, type conversion means changing the data type of a value (e.g., from int to float, or from string to int).

There are two kinds of type conversion:

1. **Implicit Type Conversion (Type Casting done by Python)**
    - Also called Type Promotion.
    - Python automatically converts a smaller data type into a bigger data type without losing information.
2. **Explicit Type Conversion (Type Casting done by Programmer)**
    - We manually convert one data type to another using functions like int(), float(), str(), list(), etc.

🔸 1. Implicit Type Conversion

Python handles this automatically when mixing different types in expressions.

```
# Example of Implicit Type Conversion
a = 5       # int
b = 2.5     # float

result = a + b   # int + float → float

print("a:", a, type(a))
print("b:", b, type(b))
print("result:", result, type(result))
```

✅ Output:

```
a: 5 <class 'int'>
b: 2.5 <class 'float'>
result: 7.5 <class 'float'>
```

👉 Here, Python automatically converted int to float to avoid data loss.

🔸 2. Explicit Type Conversion

We force the conversion using built-in functions.

Example 1: Converting String to Int/Float

```
num_str = "100"

num_int = int(num_str)    # string → int

num_float = float(num_str)   # string → float

print("String:", num_str, type(num_str))

print("Int:", num_int, type(num_int))

print("Float:", num_float, type(num_float))
```

✅ Output:

```
String: 100 <class 'str'>

Int: 100 <class 'int'>

Float: 100.0 <class 'float'>
```