

---

## Unit 1.5: Python-I

---

### **1. Fundamental Concepts of Python**

#### **1.1 Introduction to Python**

Python is a popular programming language. Python is an interpreted, high-level, general-purpose programming language. It was created by **Guido van Rossum** and released in 1991. Python is a widely used programming language known for its simplicity and readability, extensive features, and library support. Its clean and straightforward syntax makes it beginner-friendly, while its powerful libraries and frameworks make it perfect for developers.

Python has gained immense popularity in the software development community. Its straightforward and human-readable syntax makes it ideal for beginners and experienced programmers alike. Thus, Python is a foundational language for those venturing into the world of programming.

#### **❖ Important Features of Python**

- Easy to Learn & Readable: Simple syntax, easy to understand.
- High-Level Language: No need to manage hardware details.
- Cross-Platform: Runs on Windows, Mac, Linux, Raspberry Pi.
- English-Like Syntax: Readable and intuitive code.
- Concise Code: Requires fewer lines than other languages.
- Interpreter-Based: Runs code instantly for quick testing.
- Multiple Paradigms: Supports procedural, OOP, and functional programming.
- Open Source: Free to use, modify, and distribute.
- Strong Community Support: Large active user base.
- Object-Oriented: Enables code reusability and modularity.
- General-Purpose: Used in web development, AI, data science, automation, etc.

#### **❖ Getting Started with Python:**

Python is a versatile, high-level language that is widely supported across all major operating systems. You can run Python on your computer using the following two methods:

#### **Run Python Online:**

To execute Python code, you need to have a Python interpreter installed on your computer. However, if you want to start immediately, you can use a free online Python editor.

The online editor enables you to run Python code directly in your browser, no installation required.

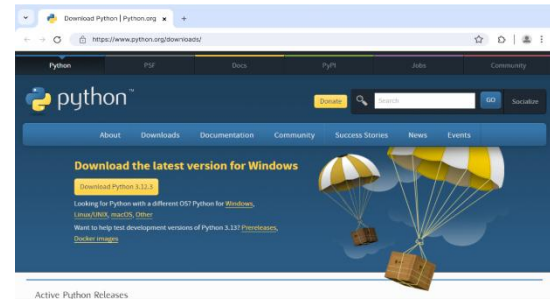
#### **Install Python on your computer:**

For those who prefer to install Python on your computer, this guide will walk you through the installation process on Windows.

**Here is a detailed explanation of each of the steps:**

### Step 1: Download the Python Installer

Go to the official Python website and download the latest version for Windows. The website automatically detects your operating system and gives you the right insane installer.



### Step 2: Run the installer

- Now, go to your Downloads folder and run the installer. You just downloaded. Depending on your security settings, you might be prompted to allow access.
- Simply allow it and proceed.

### Step 3: Install Python

Once you have run the installer, you will come across this screen.



On the Screen, you will see two options: **Install Now** and **Customize Installation**. We suggest you skip all customization options and simply click **Install Now**.

- Check on **Add python.exe to PATH** as it ensures Python is added to our system's PATH variable.
- Click **Install Now**, as it will include all necessary files needed later.

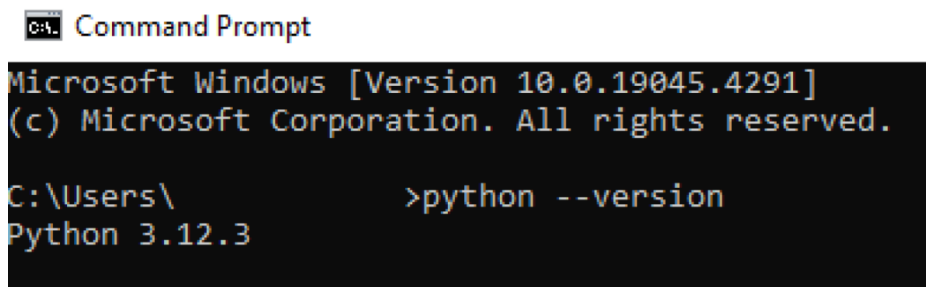
This makes it easier to run a Python program from the command prompt(cmd) directly without specifying the full path of the Python executable.

After using this option, Python will be successfully installed on your device.

### Step 4: Verify your Installation

After the installation is complete, you can verify whether Python is installed correctly by using the following command in the command prompt.

Python --version



```
C:\> Command Prompt

Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\>python --version
Python 3.12.3
```

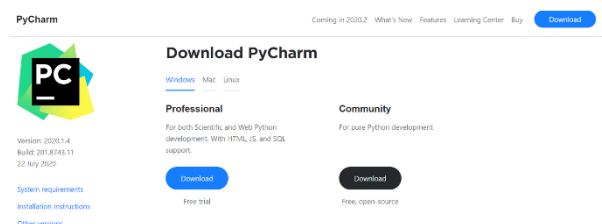
**Note:** The version number might differ from the one above, depending on your installed version.

Now, you are all set to run Python Programs on your device.

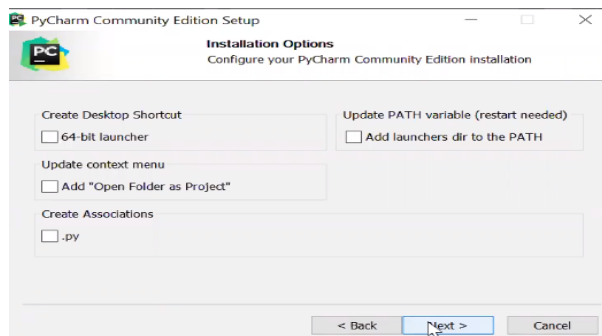
### ❖ Installing PyCharm on Windows

Python is installed, but now we will require coding software to be installed on your computer. For the courses that Educademy offers, we will use PyCharm.

**Step 1:** Go to <https://www.jetbrains.com/pycharm/download/#section=windows>. This will display the latest version of PyCharm. Underneath Community, download the free, open-source version of PyCharm.

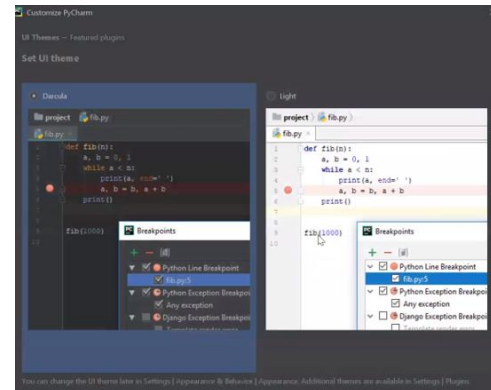


**Step 2:** Click next and go with PyCharm's default settings. PyCharm should then start installing itself.



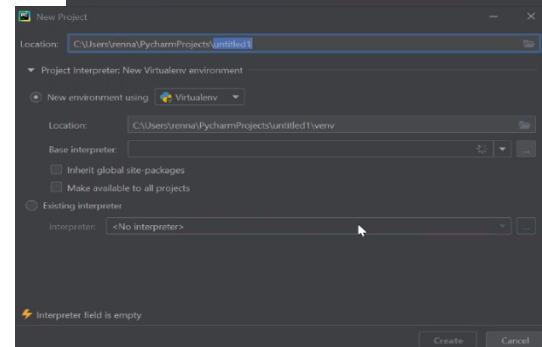
**Step 3:** We will now test if PyCharm works by running the program. You will be prompted to read the JetBrains Privacy Policy. Once you have read it, confirm that you have the User Agreement.

**Step 4:** You can now choose between different colors for your code editing software. Usually, black is easier on the eyes, but this can be changed later. Click your option and then click “skip remaining and set defaults”.



**Step 5:** A screen like this should now appear. Allow PyCharm to set up everything and then click “create”. PyCharm will then set itself up internally.

Now that you have set everything up to run Python programs on your computer.



## ❖ Application of Python

Python is a Versatile and widely used programming language known for its simplicity, readability, and a broad range of applications, some of which are mentioned below:

- **Web Development:** Used for backend development with frameworks like Django and Flask.
- **Data Science & ML:** Key language for data analysis, visualization, and AI with libraries like NumPy and pandas.
- **AI & NLP:** Supports deep learning and text processing with TensorFlow, NLTK, and spaCy.
- **Scientific Computing:** Used in research with SciPy and SymPy for simulations and modeling.
- **Game Development:** Enables 2D game creation using Pygame.
- **Databases:** Supports MySQL, SQLite, and other databases for data storage and retrieval.
- **IoT:** Programs microcontrollers and handles sensor data with MicroPython and CircuitPython.
- **Cybersecurity:** Used in penetration testing and encryption with Scapy and PyCrypto.

## 1.2 PYTHON SYNTAX AND STRUCTURE

The Python syntax defines a set of rules used to create a Python Program. The Python Programming Language Syntax is similar to Perl, C, and Java Programming Languages. However, there are some definite differences between the languages.

Or by creating a Python file on the server, using the .py file extension, and running it in the Command Line:

### First Python Program:

Let us execute a Python program to print "Hello, World!" in two different Python Programming modes. (a) Interactive Mode Programming (b) Script Mode Programming.

#### ❖ Python - Interactive Mode Programming

We can invoke a Python interpreter from the command line by typing **python** at the command prompt as follows –

```
>>> python3
Python 3.10.6 (main, Mar 10 2023, 10:55:28) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Here >>> denotes a Python Command Prompt where you can type your commands. Let's type the following text at the Python prompt and press the Enter

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

If you are running an older version of Python, like Python 2.4.x, then you would need to use a print statement without parentheses as in **print "Hello, World!"**. However, in Python version 3.x, this produces the following result.

*Hello World*

#### ❖ Python Indentation:

Indentation refers to the spaces at the beginning of a code line.

In other programming languages, code indentation is used only for readability, but in Python, it is very important.

Python uses indentation to indicate a block of code

```
if 5 > 2:
    print("Five is greater than two")
Python will give you an error if you skip the indentation:
if 5 > 2:
print("Five is greater than two")
```

#### ❖ Interactive mode:

allows you to execute Python commands line-by-line in the Python shell or REPL(Read-Eval-Print Loop). This mode is ideal for testing individual statements or experimenting with Python code snippets.

### Example:

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

```
>>>x = 10  
>>>y=5  
>>>x+y  
15
```

### Benefits of Interactive Mode:

- Instant feedback for each command executed.
- Perfect for beginners to explore Python features interactively.
- Great for testing small snippets before putting them in larger scripts.

#### ❖ Script mode:

Is used for writing, saving, and executing entire Python programs. Programs are saved in .py files and executed using the Python interpreter.

```
#file: my_script.py  
def greet(name):  
    print(f'Hello,{name}!')  
greet("world")  
#output when run:  
#Hello, world!
```

### Benefits of Scripts Mode:

- Suitable for larger programs and applications.
- Code can be reused and shared.
- Ideal for production-ready projects.

#### ❖ Interactive Mode vs Scripts Mode

Feature	Interactive Mode	Script Mode
Execution	Executes line-by-line	Executes the entire file
Purpose	Testing small code snippets	Writing full programs
File Saving	The file is not saved	Saves code in .py files
Convenience	Good for experimentation	Good for production

## ❖ Program Structure in Python:

- **Importing Modules:** In Python, you use libraries or modules to extend functionality. For example, to calculate the square root of a number, you first import the math module using the command `import math`.
- **Function and Methods:** A Python program often includes reusable blocks of code called functions. For instance, if you need to greet someone by name, you define a function like `def greet(name)` followed by logic to print the greeting.
- **Conditional Statements:** Decision-making in Python uses `if`, `elif`, and `else` statements. For example, if a number is positive, the program can print a message using `if x > 0:` followed by the action.
- **Loops:** Python supports `for` and `while` loops for iteration. For instance, you might loop through numbers from 1 to 5 using `for i in range(1,6):` and print each number.
- **Blocks of code:** Indentation is crucial in Python to define blocks of code for loops, functions, or conditional statements. Proper indentation ensures that the program's logic is correctly interpreted by Python.

## ❖ Errors In Python

- **Syntax Errors:** These occur when the Python interpreter encounters invalid syntax in your code. For instance, forgetting to close a parenthesis in `print("Hello World!")` will prevent the program from running.
- **Logical Errors:** These occur when the code runs without crashing but produces incorrect results. For example, if you mistakenly divide two numbers instead of adding them, the output will be wrong despite the program running.
- **Runtime Errors:** These happen during the program's execution when the program encounters something it cannot handle. For instance, trying to divide a number by zero (`x/0`) will throw a `ZeroDivisionError`.
- **Linking Errors:** These occur when an external library or file the program relies on is not found or is improperly linked. For instance, if you attempt to import a module that doesn't exist, Python will throw an `ImportError`.

## ❖ Debugging Python Programs:

- **Using Print Statements:**
  - i. One of the simplest ways to debug is by inserting `print()` statements in your code to check variable values or flow.
  - ii. For instance, if a loop isn't behaving as expected, adding `print(i)` inside the loop can help you track its execution.
- **Using Python Debugger (pdb):**
  - i. Python has a built-in debugger called `pdb`. You can set breakpoints in your code using `pdb.set_trace()` to pause execution and inspect variables step by step.
  - ii. This is especially useful for locating issues in complex programs.

- **Exception Handling:**

To handle errors gracefully, you can use try and except blocks. For example, if you expect a division operation to fail, you can wrap it in a try block and catch a `ZeroDivisionError` in the except block to prevent the program from crashing.

❖ **Code Review and Testing:**

- Regularly review your code for potential issues and test small portions before integrating them into a larger program.
- Use tools like `unittest` or `pytest` for automated testing

❖ **Input/output:**

- **Basic output**

The `print()` function displays output to the user. Output is the information or result produced by a program. The `sep` and `end` options can be used to customize the output. Table 1.1 shows examples of `sep` and `end`. Multiple values, separated by commas, can be printed in the same statement. By default, each value is separated by a space character in the output. The `sep` option can be used to change this behavior. By default, the `print()` function adds a newline character at the end of the output. A newline character tells the display to move to the next line. The `end` option can be used to continue printing on the same line.

- **Basic input:**

Computer programs often receive input from the user. Input is what a user enters into a program. An input statement, `variable = input("prompt")`, has three parts:

- A variable refers to a value stored in memory. In the statement above, the variable can be replaced with any name the programmer chooses.
- The `input()` function reads one line of input from the user. A function is a named, reusable block of code that performs a task when called. The input is stored in the computer's memory and can be accessed later using the variable.
- A prompt is a short message that indicates the program is waiting for input. In

Code	Output
<code>print("Today is Monday.")</code> <code>print("I like string \t beans.")</code> # tab space	Today is Monday. I like string    beans.
<code>print("Today is", "\n", "Monday")</code> #new line <code>print("Today is Monday, ",end= "...")</code>	Today is Monday Today is Monday...
<code>print("Today is Monday, ",end=" ")</code> <code>print("I like string beans.")</code>	Today is Monday, I like string beans.
<code>print("Today", "is", "Monday",sep="? ",end= "!!")</code> <code>print("I like string beans.")</code>	Today? is? Monday!! I like string beans.

the statement above, "prompt" can be omitted or replaced with any message



### ❖ Sample

#### Print your name and age

```
print("Name: John Doe")  
print("Age: 14")
```

#### Take user input and print it

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

#### Use comments in a program

```
# This is a comment  
print("Comments help in understanding the code.")
```

#### Check the type of variable

```
x = 10  
print(type(x)) # Output: <class 'int'>
```

#### Swap two numbers

```
a, b = 5, 10  
a, b = b, a  
print(a, b) # Output: 10 5
```

#### Perform basic arithmetic operations

```
x, y = 8, 3  
print(x + y, x - y, x * y, x / y)
```

#### Write a multi-line string

```
text = """Python is fun!  
It is easy to learn."""  
print(text)
```

## 2. Data Types and Operators

### 1. 2.1 Data and Information

- ❖ **Data:** Data refers to raw facts and figures without meaning unless processed. It can be in different forms: numbers, text, images, audio, and video.

#### Types of Data:

1. **Structured Data** - Organized and stored in databases (e.g., tables, spreadsheets).
2. **Unstructured Data** - Data without a predefined format (e.g., emails, social media posts).
3. **Semi-structured Data** - A mix of structured and unstructured data (e.g., JSON, XML files).

- ❖ **Information:** Information is processed data that is meaningful and useful for decision-making. When data is analyzed or structured in a certain way, it becomes information.

### 2. 2.2 Data Types in Python

Python supports various data types that determine what kind of data a variable can hold.

## ❖ Numeric Data Types

Python provides three types of numeric data:

1. **Integer (int)** - Whole numbers (e.g., 5, -10, 1000)
2. **Floating-Point (float)** - Decimal numbers (e.g., 3.14, -0.001, 2.0)
3. **Complex (complex)** - Numbers with real and imaginary parts (e.g., 2+3j, -4-2j)

## ❖ String (str)

A sequence of characters enclosed in single ( ' ') or double ( " ") quotes.

Example:

```
name = "John Doe"
message = 'Hello, World!'
```

## ❖ Boolean (bool)

Represents two values: True or False. It is used in conditions and logical operations.

Example:

```
is_valid = True
is_done = False
```

## ❖ Type Conversion and Type-checking

Python allows converting one data type into another.

Type conversion, also known as type casting, is converting one data type into another. Python provides two types of type conversion:

1. Implicit Type Conversion (Automatic)
2. Explicit Type Conversion (Manual)

### 1. Implicit Type Conversion (Automatic Type Casting)

Python automatically converts a smaller data type into a larger one to prevent data loss.

Example:

```
x = 5      # Integer
y = 2.5    # Float
z = x + y  # Python automatically converts x (int) to float
print(z)   # Output: 7.5
print(type(z)) # Output: <class 'float'>
```

### ❖ Type Conversion (Type Casting):

Explicit type conversion is done manually using predefined functions like int(), float(), str(), etc.

Example:

```
x = 10      # Integer
y = float(x) # Converts to float
print(y)    # Output: 10.0
print(type(y)) # Output: <class 'float'>
```

### Common Type Casting Functions:

Function	Description
<code>int(x)</code>	Converts x to an integer
<code>float(x)</code>	Converts x to a floating-point number
<code>str(x)</code>	Converts x to a string
<code>bool(x)</code>	Converts x to a boolean (True or False)

#### ❖ Example:

```
a = 3.7
b = int(a) # Converts float to integer
print(b)   # Output: 3

num_str = "123"
num_int = int(num_str) # Converts string to integer
print(num_int) # Output: 123
```

#### ❖ Type Checking:

To check the type of a variable, use the `type()` function. Example:

```
num = 100
print(type(num)) # Output: <class 'int'>
text = "Hello"
print(type(text)) # Output: <class 'str'>
decimal = 3.14
print(type(decimal)) # Output: <class 'float'>
```

The `type()` function helps determine the data type, which is useful for debugging and ensuring

## 3. 2.3 Variables and Assignments

A variable is a named storage location used to store values.

#### ❖ Variable Declaration and Assignment:

```
age = 25
name = "Alice"
price = 99.99
```

#### ❖ Rules for Naming Variables:

- Must start with a letter (a-z, A-Z) or underscore (`_`)
- Can contain letters, digits (0-9), and underscores
- Cannot be a Python keyword (e.g., `if`, `else`, `for`)
- Case-sensitive (`age` and `Age` are different)