1. **Chapter 01 – Python Fundamentals**
   1. What is Python & Key Features
   2. How python is different from other languages
   3. Software’s Required
   4. Install Python & VS Code
   5. Setup Environment
   6. First Python Program
   7. Running Python Program (IDE & Terminal)
   8. Keywords, Identifiers & Comments
   9. Variables & Data Types
   10. Type Casting
   11. Input & Output
   12. Operators
   13. Strings & String Methods
   14. Create venv (Virtual Environment)
   15. Requirements.txt file
2. **Chapter 02 – Control Flow & Loops**
   1. Conditional Statements (if / elif / else)
   2. Match Statement
   3. For Loop
   4. While Loop
   5. break, continue & pass
3. **Chapter 03 – Functions & Exceptions**
   1. Functions & Syntax
   2. Function Arguments & Return
   3. Scope (Local vs Global)
   4. Lambda Functions
   5. Exception Handling (try / except / else / finally)
4. **Chapter 04 – Data Structures**
   1. Lists (CRUD + common methods)
   2. Tuples (CRUD + common methods)
   3. Sets (CRUD + common methods)
   4. Dictionaries (CRUD + common methods)
5. **Chapter 05 – OOP’s Concepts**
   1. OOP Overview
   2. Classes & Objects
   3. \_\_init\_\_ & self
   4. Inheritance
   5. Polymorphism
   6. Encapsulation
   7. Abstraction
6. **Chapter 06 – Files, Libraries & Problem Solving**
   1. Read & Write Files using “with open ()”
   2. Math, Random & Date Libraries
   3. Importing Modules
   4. Basic Problem Solving
      1. Reverse string
      2. Count words
      3. Find max/min
      4. Simple test automation style scripts, etc.

**1.1 What is Python & Key Features**

**Python** is a high-level, interpreted, easy-to-read programming language used for automation, software development/testing, data science, AI and ML etc.

**Interpreter Vs Compiler**

**Interpreter** executes code line-by-line at runtime, which is slower but allows for quicker debugging. **Ex: Python, JavaScript, Ruby**

A **compiler** translates an entire program into machine code before execution, creating a standalone executable file that runs faster. **Ex: C, C++, Java**

**Key Features**

* Easy syntax (English-like)
* Platform independent
* Huge library support
* Used in Software Development/Testing, Data Science & Analytics, AI, ML, Scientific & Numeric Computing, Game Development, Automation etc.

**1.2 How python is different from other languages**

### **Summary of Key Differences:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Python** | **Other Languages** |
| **Syntax** | Readable, indentation-based | Uses braces {} or keywords |
| **Typing** | Dynamically typed | Statically typed (e.g., Java, C++) |
| **Memory Management** | Automatic (garbage collection) | Manual (C/C++) or automatic (Java) |
| **Compilation** | Interpreter | Compiler |
| **Programming Paradigm** | Multiparadigm (OOP, FP, procedural) | Mostly OOP (Java, C++) |
| **Standard Library** | Extensive "batteries-included" | Varies (more minimal in C) |
| **Performance** | Slower (interpreted, dynamic typing) | Faster (compiled languages) |
| **Best For** | Data Science, Web Development, Scripting, AI/ML etc. | System-level, performance-critical apps |

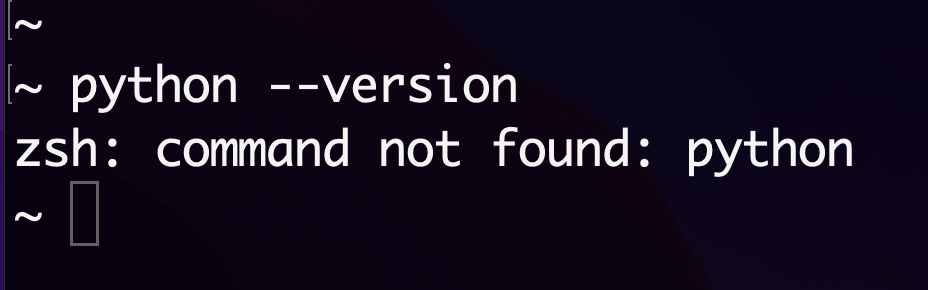
**1.3 Software's Required**

* OS: MAC/Windows/Linux
* Python
* VS Code
* Terminal/Command Prompt

**1.4 Install Python**

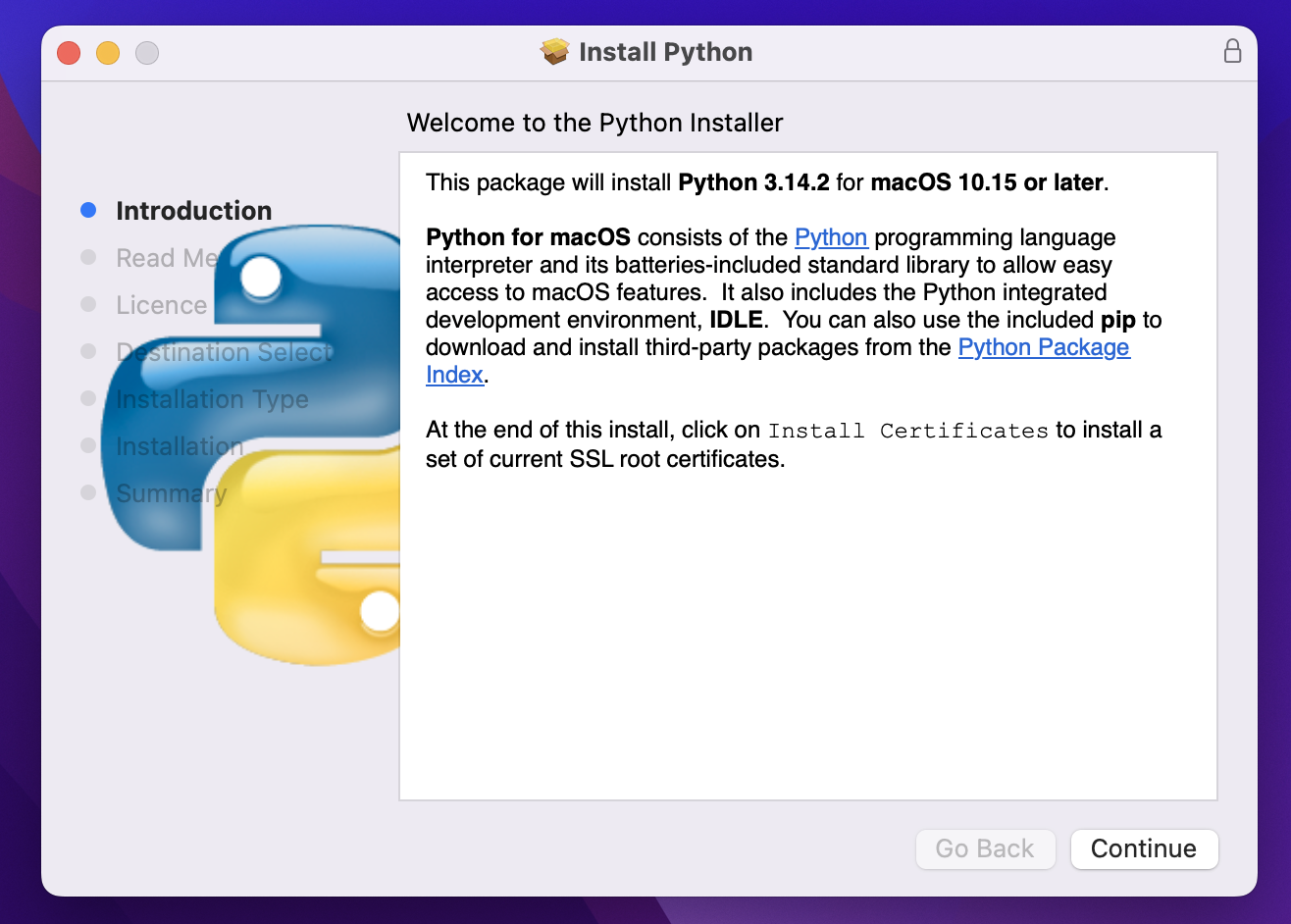
**OS: MAC**

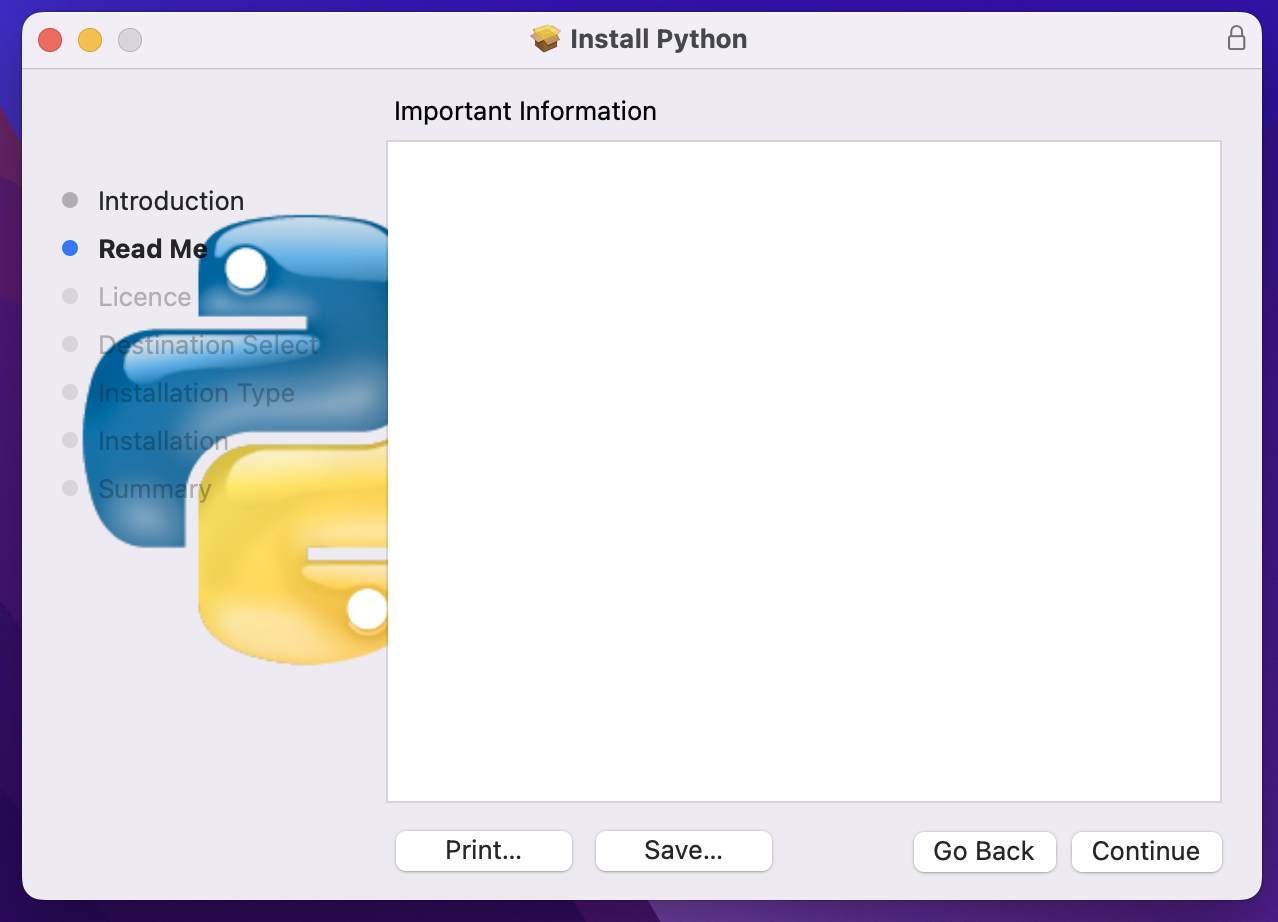
**Step1:** python --version or python3 --version



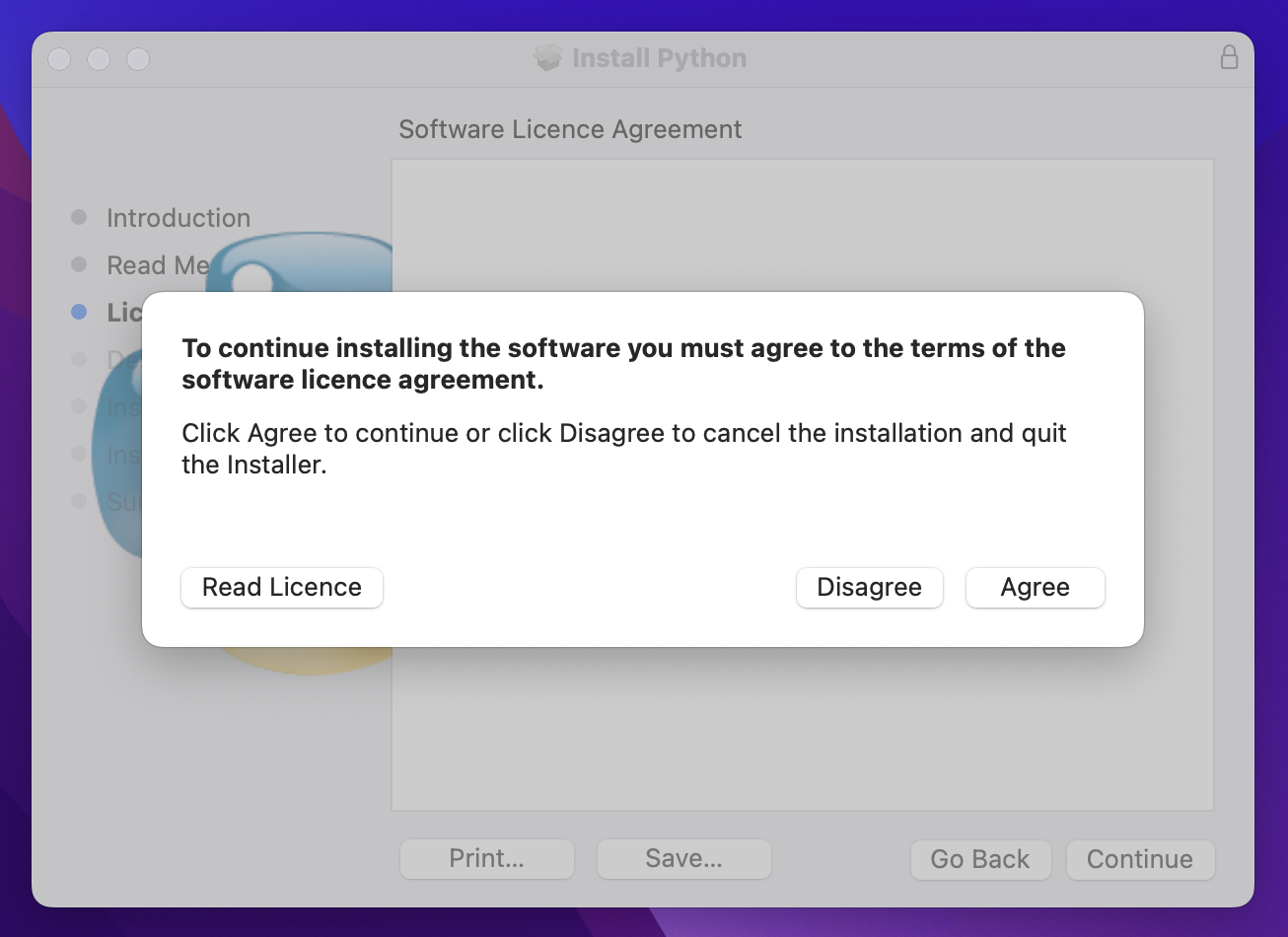
**Step2:** Download python - <https://www.python.org/downloads/>

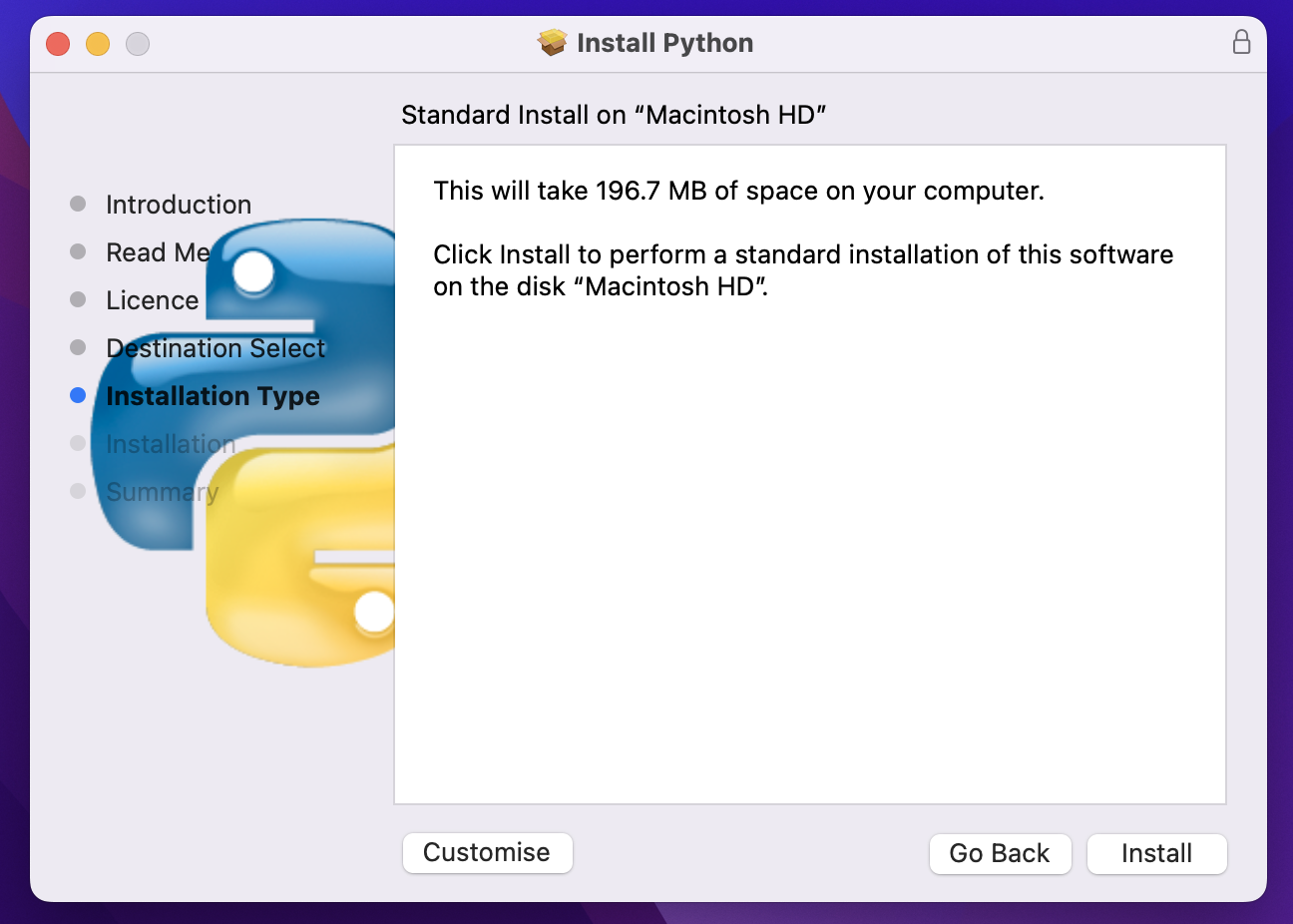
**Step3:** Double click on .pkg

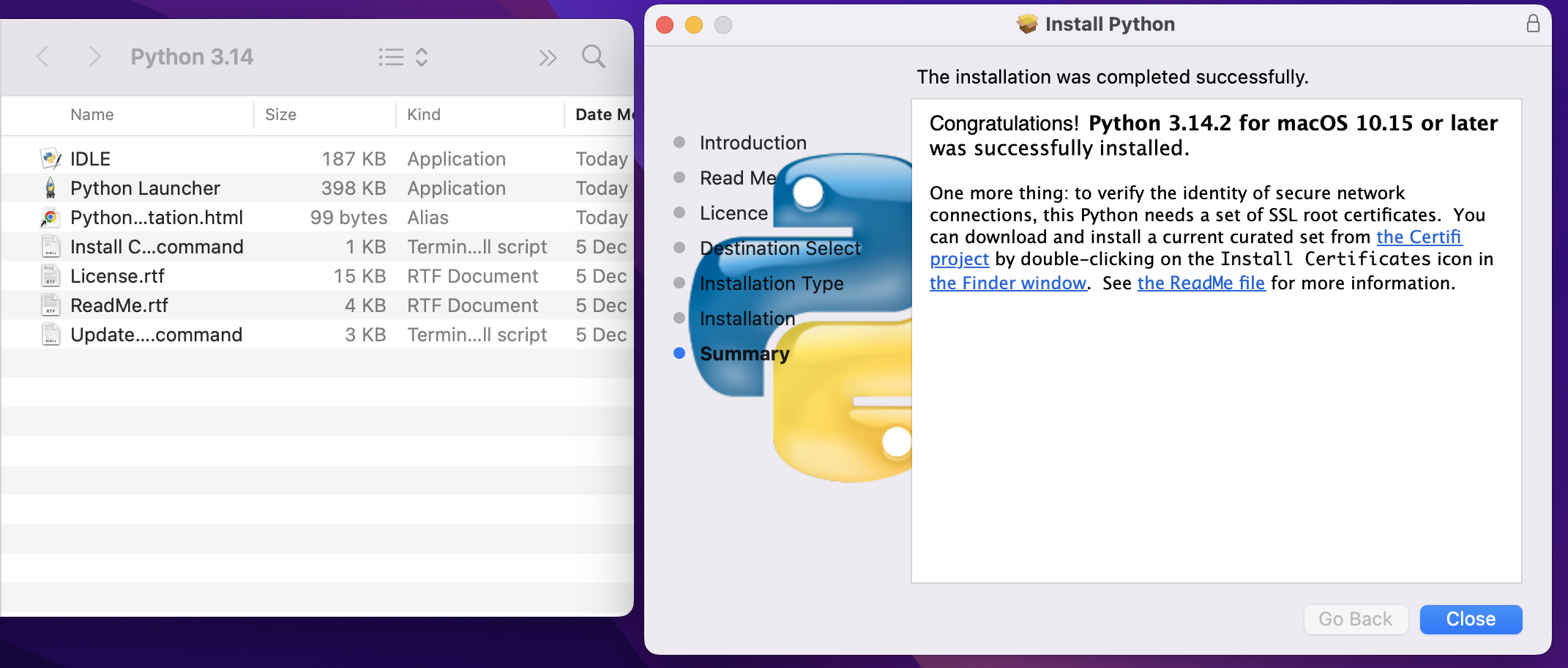










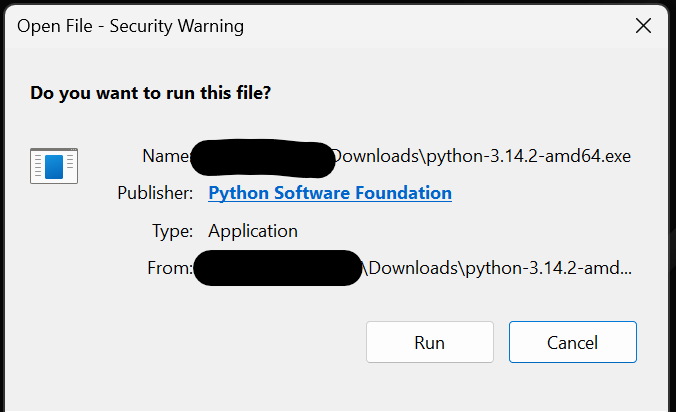


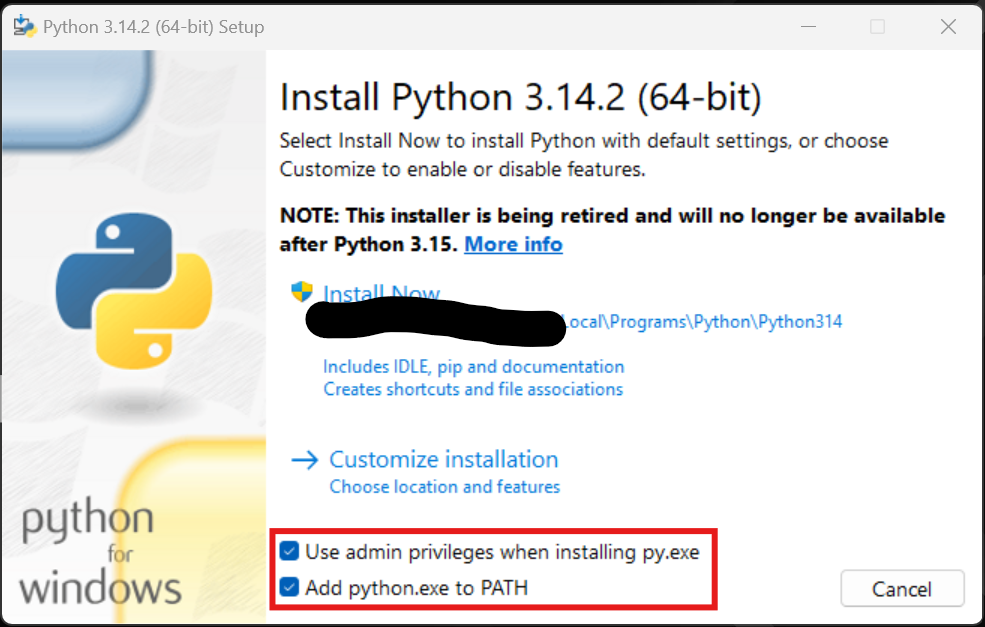
**Step4:** Verify python using “python3 --version”

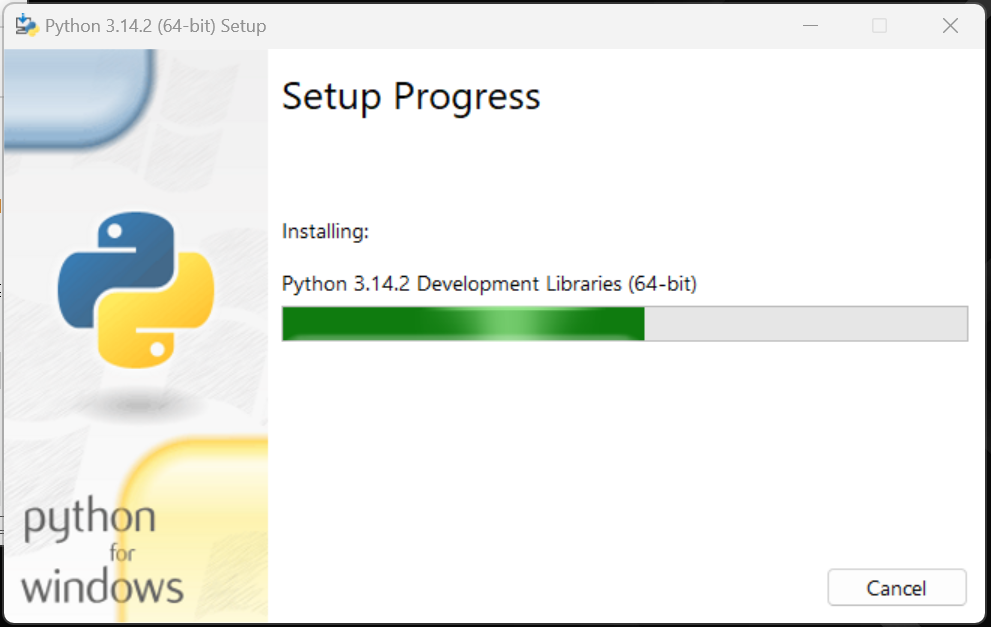


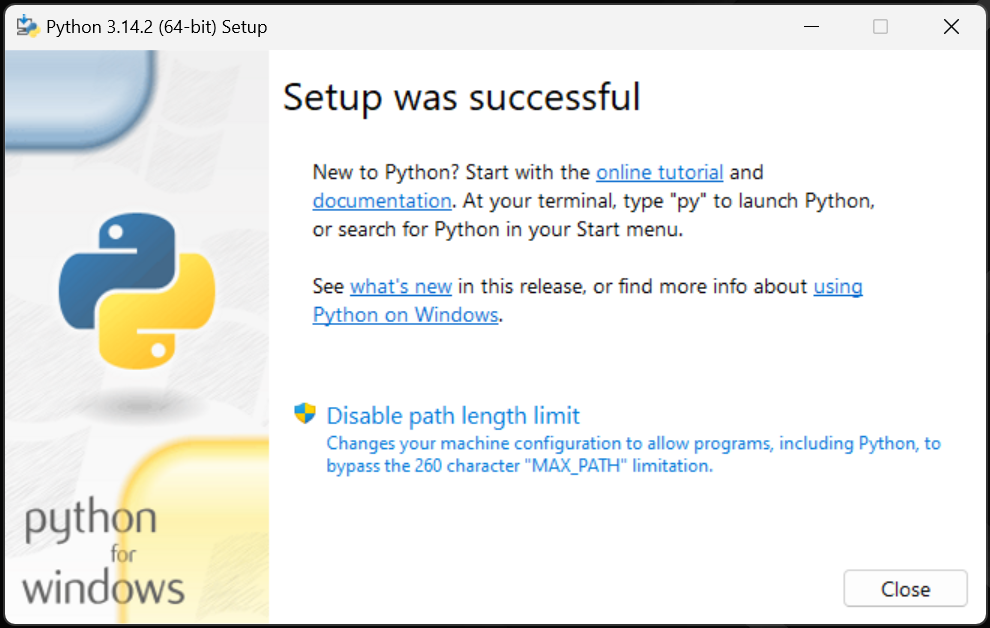
**OS: Windows**

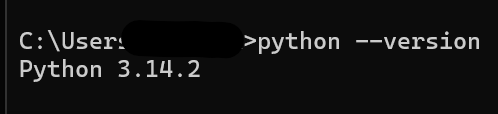
Download python - <https://www.python.org/downloads/>









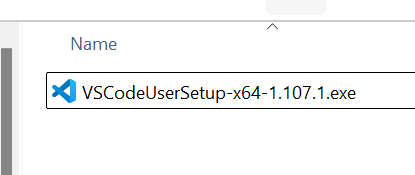


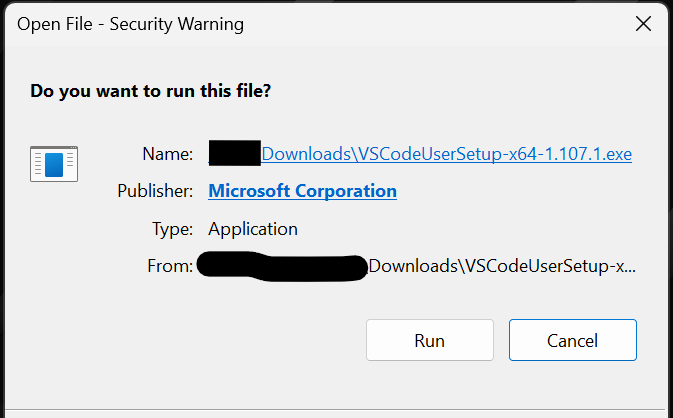
**Install VS Code -** [**https://code.visualstudio.com/download**](https://code.visualstudio.com/download)

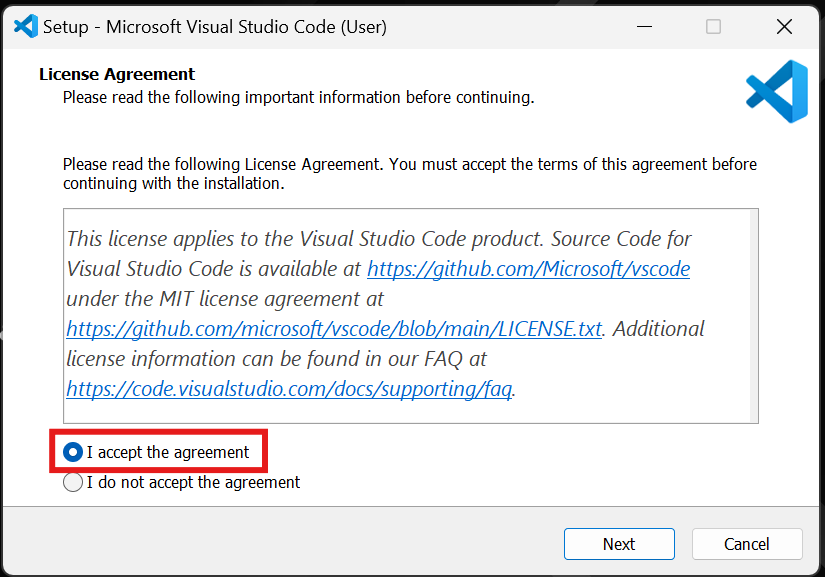
OS: MAC

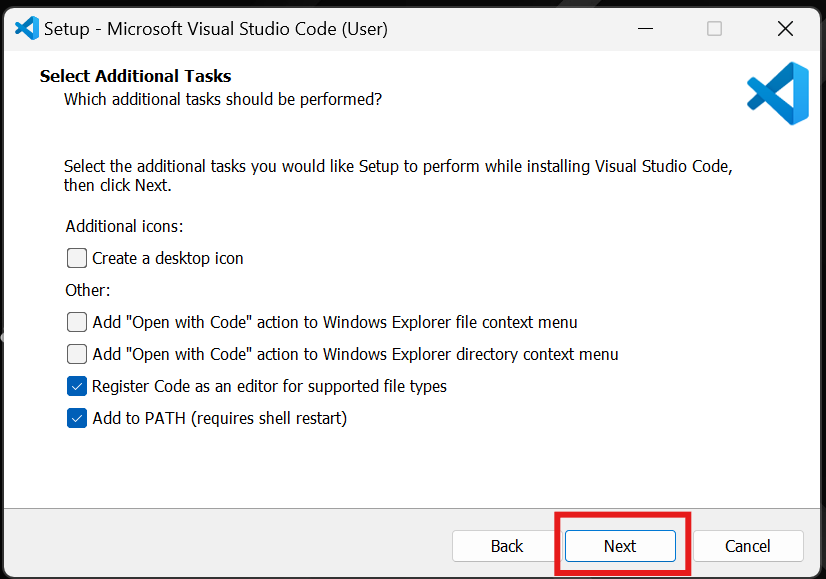
**OS:** Windows

* Double click on the downloaded .exe file

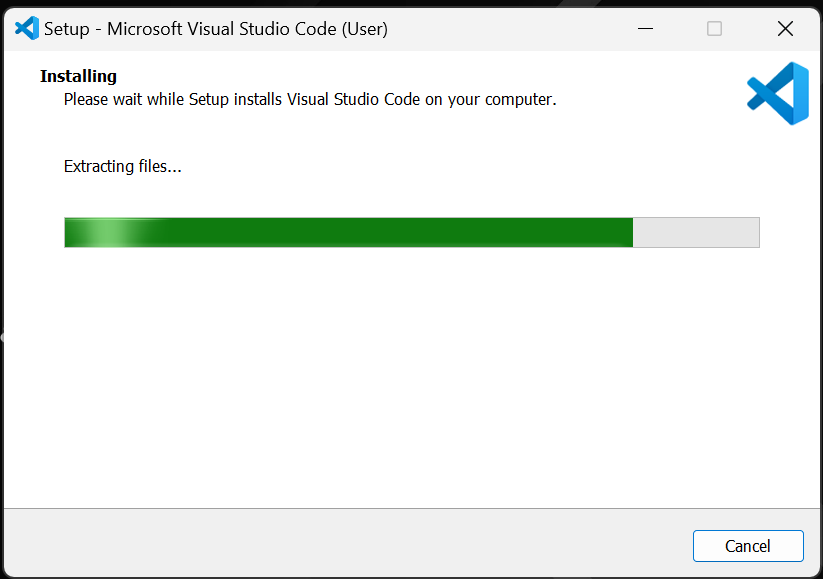
















**1.5 Setup Environment**

Extensions:

1. Python (Microsoft)
2. Python Debugger (Microsoft)

**1.6 First Python Program**

- Create .py file & print something...

- Run python program using VS Code IDE

**1.7 Running Python Program (IDE & Terminal & Command Prompt)**

**1.8 Keywords, Identifiers & Comments**

**- Keywords** are reserved words in Python.  
 They have predefined meanings and cannot be used as variable names.

**Example**: if, else, for, while, def, return, True, False, None, class, import etc.

**1.8 Keywords, Identifiers & Comments**

**- Identifiers** are names given to variables, functions, classes, or objects.

**📌 Rules for Identifiers**

* Must start with a letter or underscore (\_)
* Cannot start with a number
* Can contain letters, numbers, and underscores
* Case-sensitive (age and Age are different)
* Cannot use Python keywords

**1.8 Keywords, Identifiers & Comments**

**- Comments** are used to explain code and are ignored by the Python interpreter.

- Single line comment (# text)

- Multi-line comment (“”” text “””)

**1.9 Variables & Data Types**

**What are Variables?**

A **variable** is a name that refers to a value stored in memory.  
 In Python, you **don’t need to declare the type** of variable explicitly - Python figures it out automatically.

**Data Types:**

Data types define the **kind of value** a variable can hold.

- string

- integer

- floating/decimal point

- boolean

**1.10 Type Casting**

- Type casting from string to integer

- Type casting from integer to string

**1.11 Input & Output**

- How to accept command line input & perform operations.

**1.12 Operators**

- Arithmetic Operators

- Comparison Operator

- Logical Operator

**1.13 Strings & String Methods**

- strip ()

- len()

- startswith(‘substr’)

- endswith(‘substr’)

- lower ()

- upper ()

- title ()

- capitalize ()

- index(‘character’)

- replace (‘original’, ‘new value’)

- split (‘ ’)

- find(‘substr’)

**Checking methods**

* isdigit()
* isalpha()
* isalnum()

- substrings

**1.14 Virtual Environment(venv)**

**When:**

* Immediately after creating the project folder
* Before installing any third-party libraries

**Why:**

* Python libraries are project-specific
* Prevents version conflicts between projects

**Commands:**

* python3 -m venv venv
* source venv/bin/activate // **macOS/Linux**
* Venv\Scripts\Activate.ps1 **// powershell**
* venv\Scripts\activate **// CMD**

**1.15 requirements.txt**

- Create requirements.txt after installing project dependencies

- pip install playwright pytest

- pip3 install -r requirements.txt // Throws error when file doesn’t exist

- pip3 freeze > requirements.txt

**If any issues, replace pip3 with pip**

**2.1 Conditional Statements (if / elif / else)**

- if, elif, and else are conditional statements

- Used Execute different blocks of code based on conditions.

**Syntax:**

**if** first\_condition:

# block runs if condition is True

**elif** second\_condition:

# runs if above condition is False and this is True

**else:**

# runs if all conditions are False

### **Important rules**

✔ if is mandatory  
✔ elif is optional (can have many)  
✔ else is optional  
✔ Indentation is **mandatory** in Python

**2.2 Match Statement**

It works like a **switch-case** statement but is **more powerful and flexible**.

It compares a value against multiple **patterns** and executes the matching block.

**Benefits:**

* Cleaner & more readable than multiple if-elif-else
* Reduces complex conditional logic
* Supports **pattern matching** (values, types, sequences, dictionaries)
* Improves maintainability in real-world applications

**Syntax:**

match variable:

case pattern1:

# code block

case pattern2:

# code block

case \_:

# default case

**Example:**

day = 3

match day:

case 1:

print("Monday")

case 2:

print("Tuesday")

case 3:

print("Wednesday")

case \_:

print("Invalid day")

**2.3 for loop & nested for loops**

“for” loop used to execute a block of code for a specified number of times or until condition is true.

**Syntax:**

for variable in sequence:

# code block

**Example:**

for i in range (1, 6, 2):

print(i)

**Range method:**

range(start, stop, step)

* Starting value (default 0)
* Ending value (excluded)
* Increment (default 1)

**2.4 while loop**

A while loop is used to execute a block of code as long as a condition is True

**When:**

* The number of iterations is **unknown**
* Loop should continue until the condition **changes.**

**Syntax:**

while condition:

# code block

**Note:** Forgetting to update condition in while → infinite loop

**2.5 break, continue & pass**

These are **loop control statements** that change the normal flow of execution in loops.

**break** is used to **terminate (exit) the loop immediately**, even if the loop condition is still True.

**Example:**

for i in range(1, 10):

if i == 5:

break

print(i)

**continue** is used to **skip the current iteration** and move to the **next iteration in the loop**.

**Example:**

for i in range(1, 10):

if i == 3:

continue

print(i)

**pass** is a **null statement** — it does nothing.

**Why:**

To act as a placeholder for future code

To create **empty blocks**

To avoid syntax errors during development

## **🧠 One-Liners Summary**

* **break**: *Terminates the loop immediately*
* **continue**: *Skips the current iteration*
* **pass**: *Acts as a placeholder with no operation*

**3.1 Functions, Syntax & Examples**

**function** is a **block of reusable code** that performs a **specific task**.

Instead of writing the same code again and again, you put it inside a function and **call it whenever needed**.

### **✅ Benefits of Functions**

1. **Code Reusability**  
    → Write once, use many times
2. **Clean & Readable Code**  
    → Break large programs into small logical blocks
3. **Easy Maintenance**  
    → Fix code in one place instead of everywhere
4. **Modularity**  
    → Each function does one job
5. **Testing Made Easy**  
    → Functions can be tested independently

**Syntax:**

def function\_name():

// code

**3.2 Function Arguments & Return**

def function\_name(**param1**, **param2**...):

// code

**return** value

**3.3 Scope (Local vs Global)**

- Global variables

- Local variables

### **✅ What is a Global Variable?**

A **global variable** is declared **outside of all the functions** and can be accessed **anywhere in the program**.

**Syntax:**

global\_variable = value

def function\_name():

print(global\_variable)

**Example:**

x = 100 # global variable

def show():

print(x)

show()

print(x)

✔ Accessible inside and outside function

### **✅ What is a Local Variable?**

A **local variable** is declared **inside a function** and can be accessed **only within that function**.

**Syntax:**

def function\_name():

local\_variable = value

**Example:**

def add():

a = 10 # local variable

b = 20 # local variable

print(a + b)

add()

✔ Accessible inside function only

**3.4 Lambda Functions**

**lambda** functions are created using lambda keyword, we can write less code when compared to function implementation.

**Syntax:**

lambda arguments: expression

**3.5 Exception Handling (try / except / else / finally)**

Exception handling allows you to **handle runtime errors gracefully** without crashing the program.

Python uses **try, except, else, finally** block to handle exceptions

**except** runs only when exceptions occur.

**else** runs only if no exception occurs.

**finally always executes**, used for cleanup.

**Syntax:**

try:

# risky code

except ExceptionType:

# handling code

else:

# code

finally:

# cleanup code

**4.1 Lists (CRUD Operation)**

* A **list** is a collection of items
* Ordered, **mutable (changeable)**
* Allows **duplicate values**
* Can store **different data types values**

**Syntax:**

my\_list = [element1, element2, element3....]

**Example:**

mixed = [1, "Hello", 3.5, True]

**4.2 Tuples**

* A **tuple** is a collection of items
* **Ordered**
* **Immutable** (cannot be changed)
* Allows **duplicate values**
* Can store **multiple data types**
* Supports for **packing & unpacking**

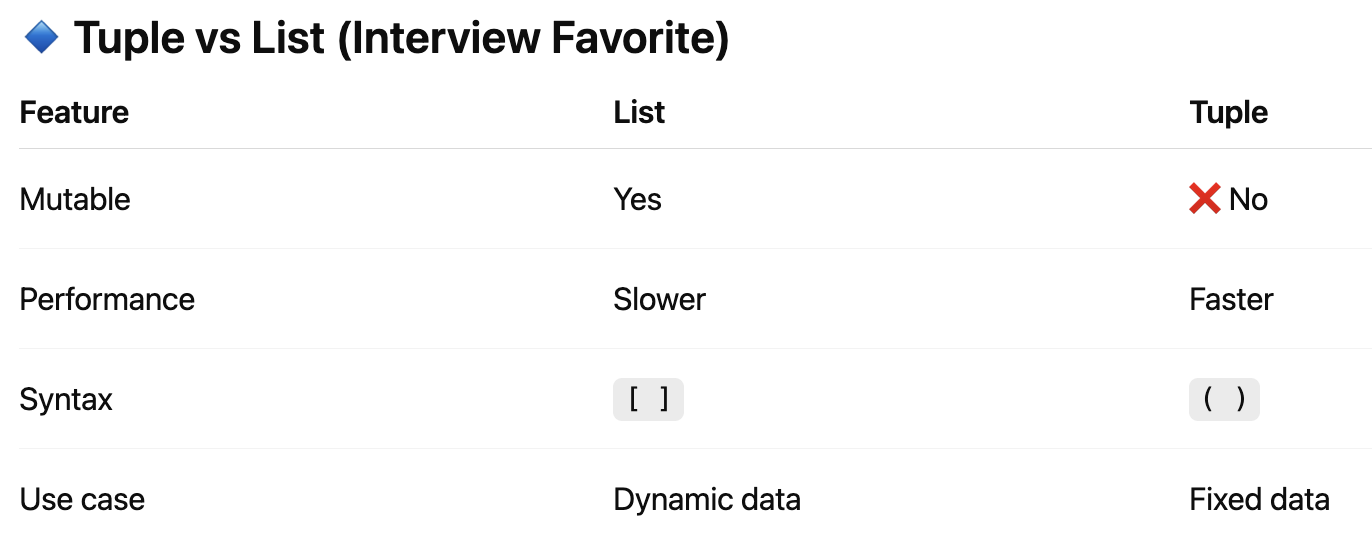
**Syntax:**

my\_tuple = (element1, element2, element3)

**Example:**

**mixed = (1, "Hello", 3.5)**

**One line:** A tuple is an ordered, immutable collection used when data should not be modified.



**4.3 Sets**

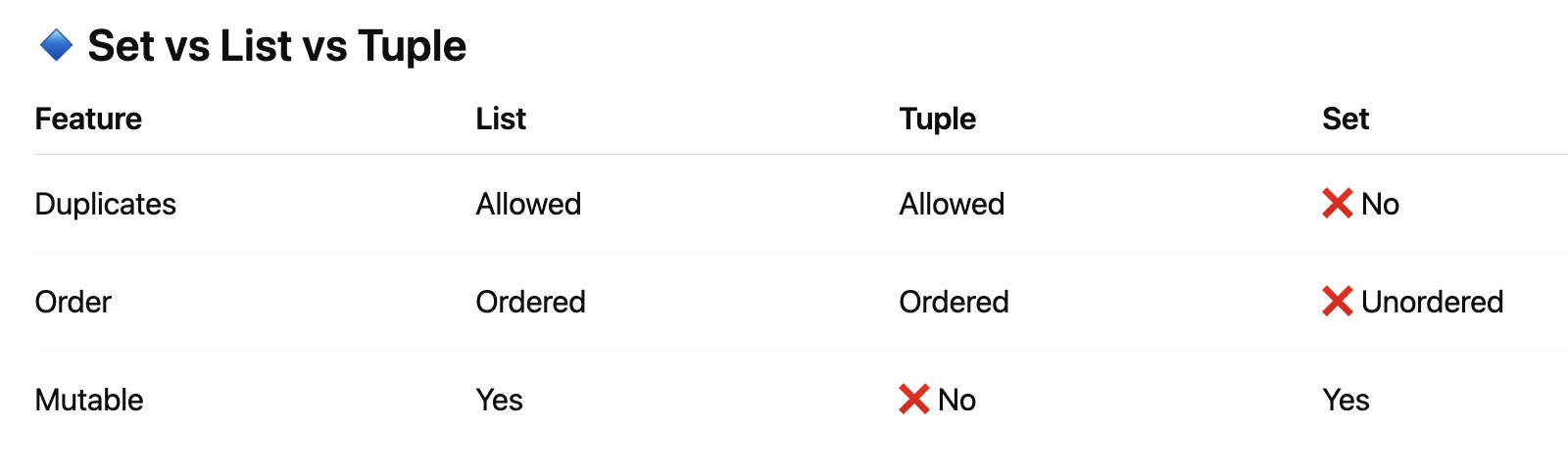
* A **set** is a collection of **unique elements**
* **Unordered**
* **Immutable**
* Does **not allow duplicates**
* Uses { }

**Syntax:**

my\_set = {element1, element2, element3}

**Example:**

mixed = {1, "Hello", 3.5}



**4.4 Dictionaries**

* A **dictionary** stores data as **key–value pairs**
* **Mutable**
* Keys must be **unique**
* Keys must be **immutable** (string, int, tuple)

**Syntax:**

my\_dict = {key1: value1, key2: value2...}

**Example:**

employee = {

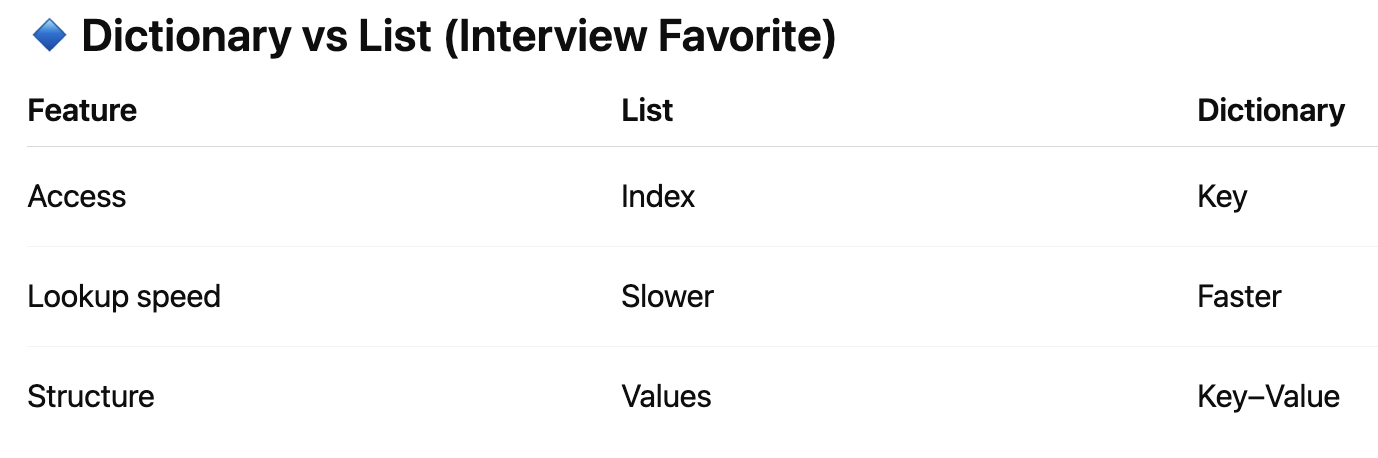
"name": "Jhon",

"age": 30,

"salary": 50000

}

**One line:** A dictionary is a mutable data structure that stores data in key–value pairs for fast lookup.



**5.1 OOP’s Overview**

**OOP (Object-Oriented Programming)**

* To create **reusable** code
* To create **Maintainable** software
* Enhance code organization
* Improve the security
* Scalability
* Better debugging in large complex application
* **Objects** that will have: **Attributes** (data) & **Methods** (behavior / functions)

**OOP’s concepts:** Class, Objects, Encapsulation, Inheritance, Abstraction & Polymorphism.

**5.2 Classes & Objects**

**Class:**

- One of the OOP’s concept.

- A **class** is a **blueprint or template** for creating objects.

- Class contains attributes & methods

**Syntax:**

class class\_name:

# attributes (variables)

# methods (functions)

**Object:**

- An **object** is an **instance of a class**.

**Syntax:**

object\_name = class\_name()

- Structure code using **classes and objects**.

**5.3 \_\_init\_\_ & self**

* **What is \_\_init\_\_?**
* **\_\_init\_\_** is a constructor
* Declared in class only
* Automatically **\_\_init\_\_** will be invoked when an object is created
* **Why \_\_init\_\_?**
* Used to initialize object variables

**Syntax:**

def \_\_**init\_\_**(**self**, param1, param2...):

self. param1 = param1

self. param2 = param2

**5.4 Inheritance**

**Inheritance** allows a class (**child / subclass**) to **reuse properties and methods** of another class (**parent / base class**).

📌 Benefits:

* Code reusability
* Maintainability
* Extensibility

**Syntax:**

class **ParentClass**:

# parent properties & methods

pass

class ChildClass(**ParentClass**):

# child properties & methods

pass

**What is Method Overriding?**

When a **child class provides its own implementation** of a method that already exists in the **parent class**.

**super()** used for calling parent class constructor & methods, variables

**Type of Inheritance with Examples**

1. Single Inheritance
2. Multi-level Inheritance
3. Multiple inheritance
4. Hierarchical Inheritance

**2. Multi-level Inheritance:**

Sub class Inherits properties from another sub class

print(‘Multi-level Inheritance’)

class A:

def funtion1(self):

print('Running function1...')

class B(A):

def funtion2(self):

print('Running function2...')

class C(B):

def funtion3(self):

print('Running function3...')

object1 = C()

object1.funtion1()

object1.funtion2()

object1.funtion3()

**3. Multiple inheritance:**

In **multiple inheritance**, a class inherits from **more than one parent class**.

print(“Multiple Inheritance”)

class A:

def function1(self):

print('Running function1...')

class B:

def function2(self):

print('Running function2...')

class C(A, B): # multiple inheritance

def function3(self):

print('Running function3...')

object1 = C()

object1.function1()

object1.function2()

object1.function3()

**4. Hierarchical Inheritance:**

Multiple child classes inherit from the same parent class

print(“Hierarchical Inheritance”)

class A:

def function1(self):

print('Running function1...')

class B(A): # Hierarchical Inheritance

def function2(self):

print('Running function2...')

class C(A): # Hierarchical Inheritance

def function3(self):

print('Running function3...')

obj\_b = B()

obj\_c = C()

obj\_b.function1()

obj\_b.function2()

obj\_c.function1()

obj\_c.function3()

**5.5 Polymorphism**

**Polymorphism** means **“one name, many forms”**.

In Python, it allows the **same method name** to behave **differently** based on the object calling to it.

**One line:** Same function name, different behavior

**Polymorphism can be achieved using:**

* Method Overriding
* Duck Typing
* Operator Overloading

**5.6 Encapsulation**

**Encapsulation** means **binding data (variables) and methods together** and **restricting direct access** to object’s data.

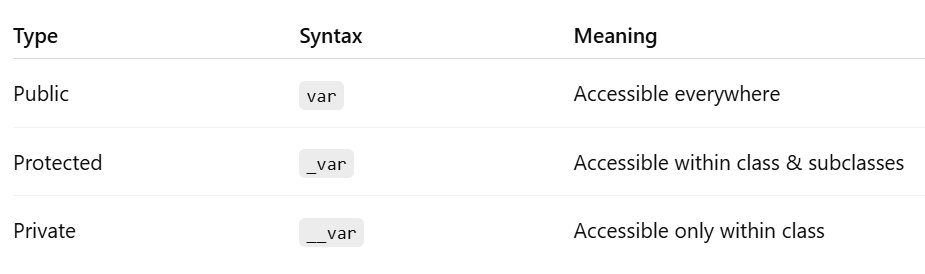
In simple words:  
 **“Data hiding + controlled access”**

**🔹 Why Encapsulation?**

* Protects data from accidental modification
* Improves security
* Makes code maintainable
* Provides controlled access via methods

Encapsulation can be achieved usingaccess modifiers **(public, protected, private)**

Python doesn’t have strict access modifiers like Java, but uses **naming conventions**:



**Examples:**

**print("=== public ===")**

class Employee: def **init**(self, salary): self.salary = salary # public

emp = Employee(50000) print(emp.salary)

**print("=== protected ===")**

class Employee:

def **init**(self, salary):

self.\_salary = salary # protected

def show\_salary(self):   
 print(self.\_salary)

emp = Employee(50000)

emp.show\_salary()

print(emp.\_salary) # accessible, but NOT recommended

**print("=== private ===")**

class Employee: def **init**(self, salary): self.\_\_salary = salary # private

def get\_salary(self):  
 return self.\_\_salary  
  
def set\_salary(self, amount):  
 if amount > 0:  
 self.\_\_salary = amount

emp = Employee(50000) print(emp.get\_salary()) # allowed

emp.set\_salary(60000) print(emp.get\_salary()) # allowed

#print(emp.\_\_salary) # Error

**5.7 Abstraction**

**Abstraction** means **hiding implementation details** and showing **only what is necessary** to the user.

👉 Focus on **what an object does**, not **how it does it**.

**Why do we need Abstraction?**

✔ Reduce complexity  
✔ Improve code readability  
✔ Enforce common structure  
✔ Helps in large-scale applications/frameworks

**Key Points:**

✔ Abstraction can be achieved by creating abstract class

✔ Uses abc(Abstract Base Class) module  
✔ Abstract method has no body

✔ Abstract class **can have normal methods**  
✔ Child class must override  
✔ Cannot instantiate abstract class

**Example:**

**6.1 Read & Write Files using “with open ()” usage**

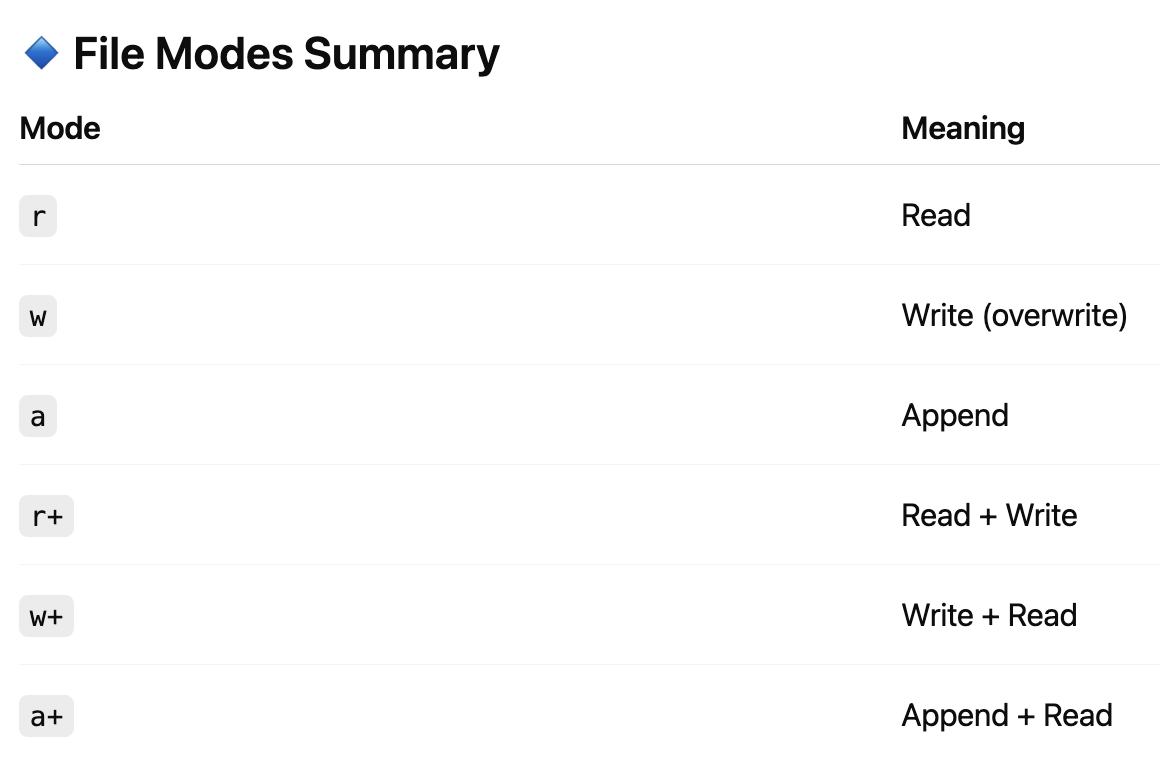
with open is used for **file handling** and it **automatically closes the file** after use.

It uses something called a **context manager**.

Use “with open ()” instead of “open ()” method

**Advantages:**

✅ No need to call close()  
✅ Safer (prevents memory leaks)  
✅ Cleaner code



**6.3 Math, Random & DateTime Libraries**

**Math** and **Random** libraries are very commonly used in Python and frequently asked in interviews.

The **math** module provides **mathematical functions**.

Firstly, you need to import - “import math”

* Square root
* Power
* Round down
* Round up
* Factorial
* GCD
* Pi
* E, etc.

The **random module** is used to **generate random values**.

Firstly, you need to import – “import random”

* Floating point
* Random integer
* Random choice
* Shuffle list etc.

The **datetime module** is used to **work with dates and times**.

Firstly, you need to import - “import datetime”

* Date + Time
* Date
* Time
* Format (/, -)
* Future date
* Difference between dates

**6.4 Importing Modules**

**6.5 Basic Problem Solving**

**6.5.1 Reverse string**

**6.5.2 Count words**

**6.5.3 Find max/min etc.**