**PYTHON**

1. LIST , SET, TUPLE

* **List**: **Ordered** collection — the items maintain the order in which they were added.
* **List**: **Allows** duplicates.

my\_list = [1, 2, 3]

print(my\_list[0]) # Output: 1

* **Set**: **Unordered** collection — there is **no guarantee of order**, and you cannot access elements by index.
* **Set**: **Does NOT allow** duplicates — any duplicates are automatically removed.

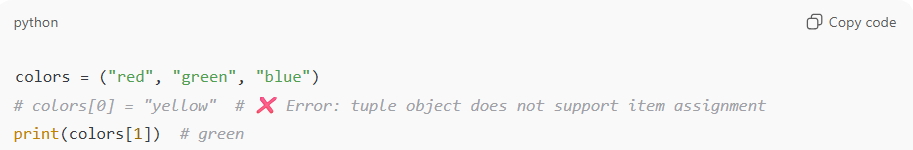
my\_set = {1, 2, 3}

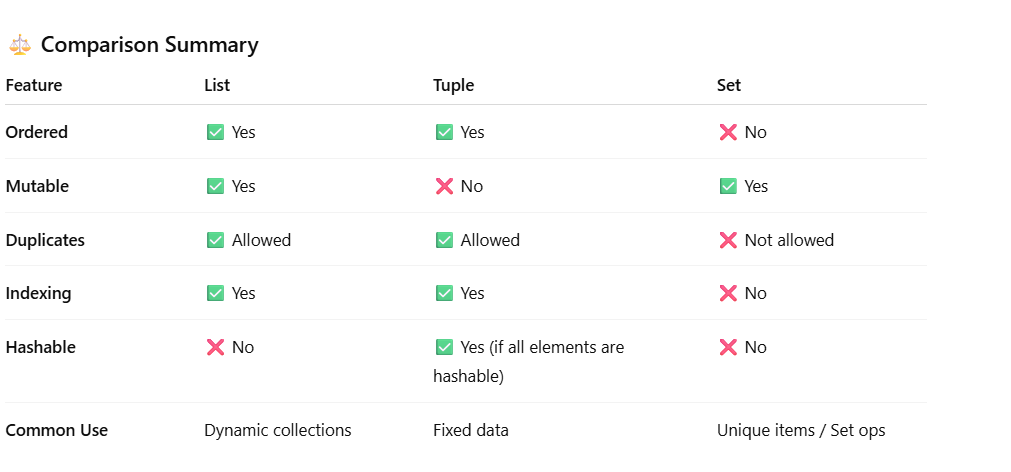
# my\_set[0] # ❌ Error: sets don't support indexing

my\_set = {1, 2, 2, 3}

print(my\_set) # Output: {1, 2, 3}

* Both **lists** and **sets** are **mutable** — you can add or remove items.
* **Tuple** : An **ordered**, **immutable** collection that can hold duplicate elements.
* **Tuple** : tuple1 = (1, 2, 3, 2, "apple")
* **Tuple**: **Mutable:** ❌ No (cannot modify once created)



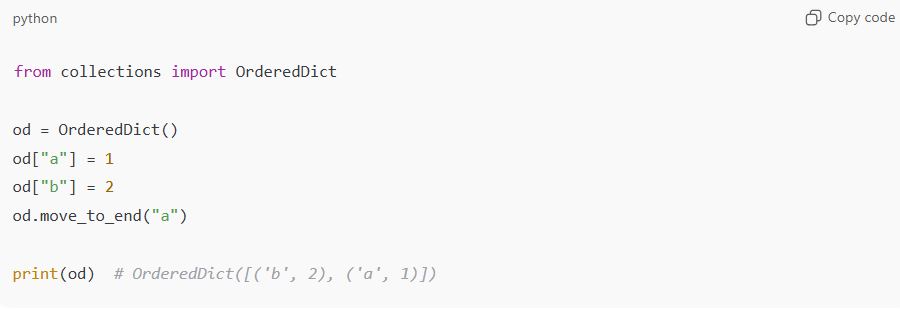


1. Dictionaries

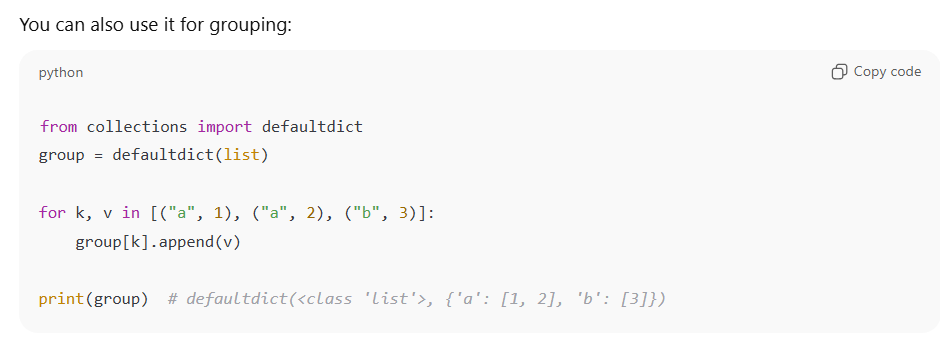
A **dictionary (dict)** is an **unordered collection** of **key–value pairs**.  
Each **key** must be **unique** and **immutable** (like a string, number, or tuple), while the **value** can be **any object**

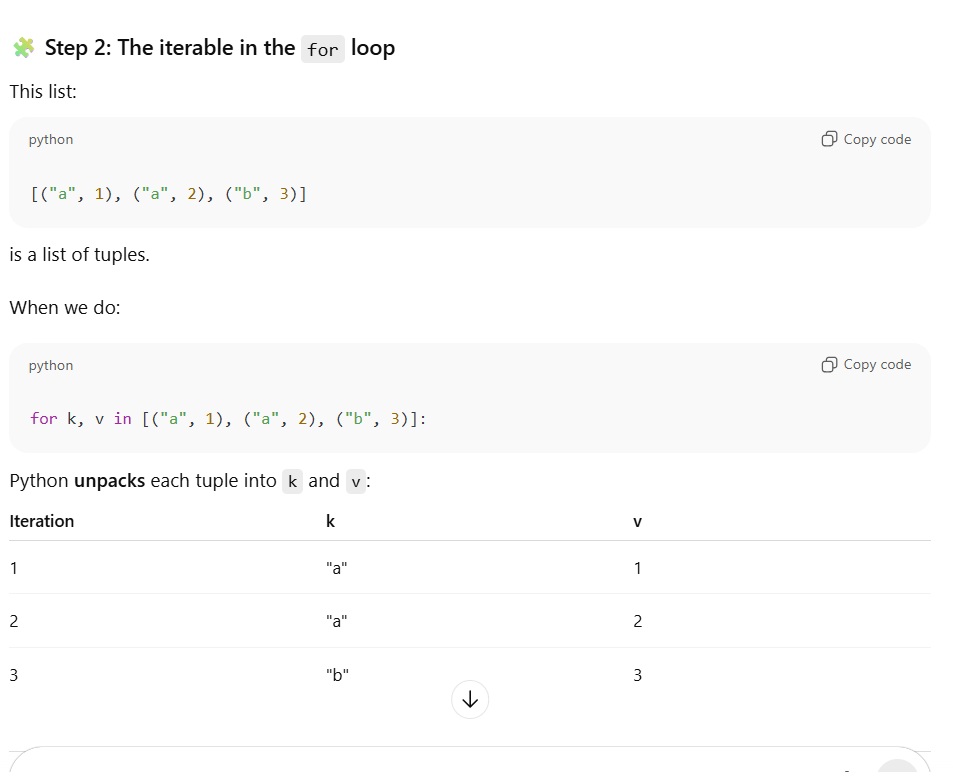


* 1. **OrderedDict** (collections.OrderedDict)



* 1. **DefaultDict** (collections.defaultdict)





* 1. **ChainMap** (collections.ChainMap)
* Used to **combine multiple dictionaries** into a single view without merging them.
* Lookups search through the maps in order — **like a layered view**.

A screenshot of a computer

AI-generated content may be incorrect.

1. What are the different ways to access dictionaries elements?

person = {"name": "Alice", "age": 25, "city": "London"}

A screenshot of a computer

AI-generated content may be incorrect.

A white box with black text

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.



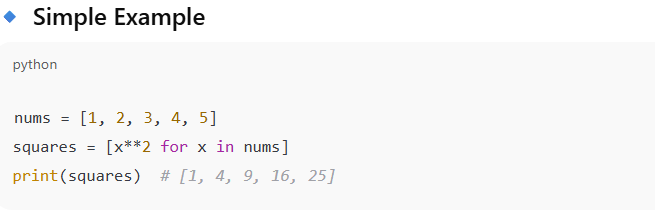
A screenshot of a computer program

AI-generated content may be incorrect.

1. List and Dict comprehension

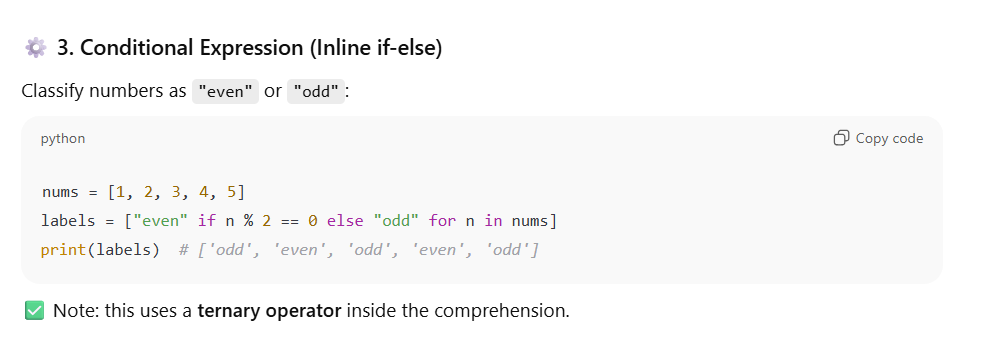
List comprehension provides a **concise, one-line way** to create new lists by transforming or filtering existing iterables.

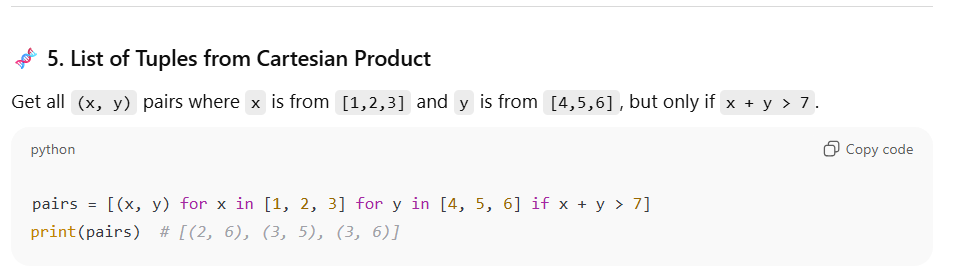
[expression for item in iterable if condition]



A screenshot of a computer

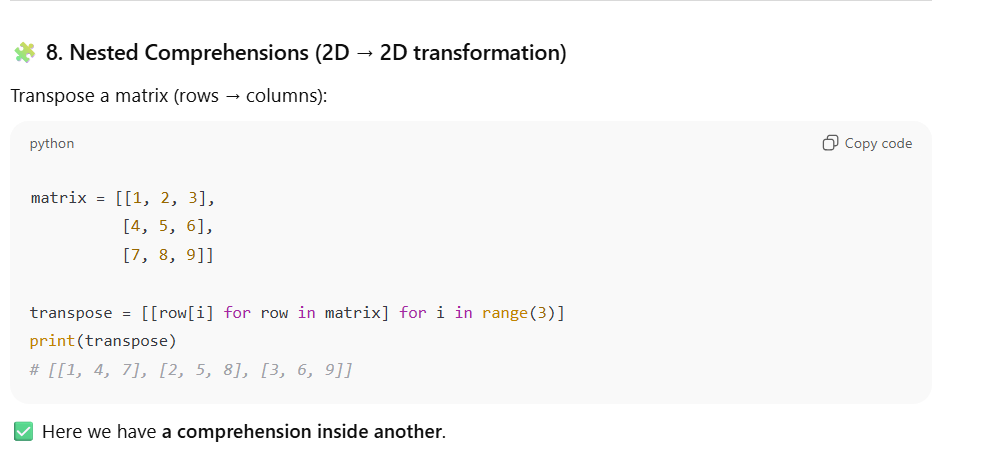
AI-generated content may be incorrect.





A white background with black text

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

1. Lambda Function

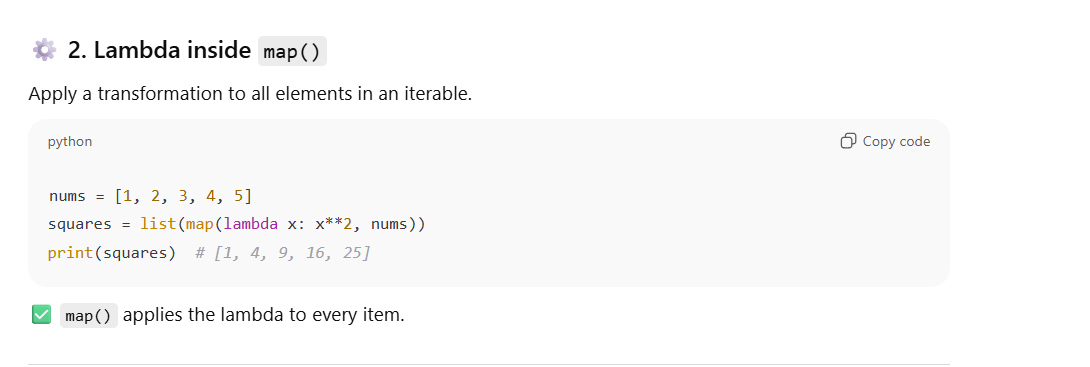
A **lambda function** is a **small anonymous (unnamed)** function defined in a single line using the lambda keyword.  
It’s often used for **short, one-time operations**, especially where defining a full function with def would be overkill.

A screenshot of a computer

AI-generated content may be incorrect.

A white rectangular object with a black stripe

AI-generated content may be incorrect.







A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

1. “For” Loop and “While” Loop.



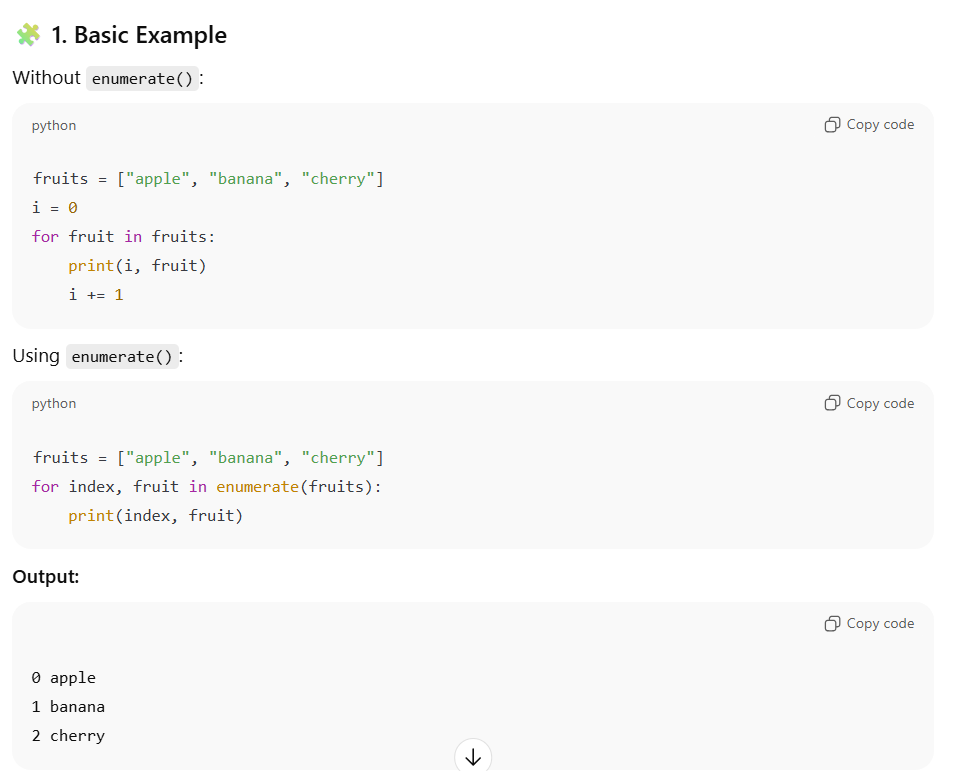
A screenshot of a computer program

AI-generated content may be incorrect.



1. Python Enumerate()

The **enumerate()** function adds a **counter (index)** to an iterable (like a list, tuple, or string) and returns it as an **enumerate object** — which can be directly used in a loop.



A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Python generators

A **generator** is a **special type of function** that allows you to **iterate over data without storing it all in memory** at once.  
It’s created using the **yield** keyword instead of return.

A screenshot of a computer program

AI-generated content may be incorrect.

1. Python Decorators.

A **decorator** is a function that **takes another function as input**, **adds extra functionality** to it, and then **returns a new function** — *without modifying the original function’s code.*



A close-up of a person

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

1. Usage \*args and \*\*kwargs

They are **special syntax** used in Python function definitions to **pass a variable number of arguments**.

A close-up of words

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



1. **Python – Threading, Asyncio and Multiprocessing .**
   1. **Threading**

**Threading is used when you have I/O-bound tasks like:**

* **Downloading files**
* **Reading web pages**
* **Waiting for I/O (network or disk)**

**I/O-bound tasks are computing tasks that spend most of their time waiting for input/output operations to complete, such as reading from or writing to a disk, network, or database.  Unlike**[**CPU-bound tasks**](https://www.google.com/search?q=CPU-bound+tasks&sca_esv=07b4fd533e682f0b&sxsrf=AE3TifMk6nZ2VAFlJauu-gWLTok8CWSQPw%3A1760804045780&source=hp&ei=zbzzaLLDLeyRseMPwKzbuAQ&iflsig=AOw8s4IAAAAAaPPK3YPz3Nao915pIzu4A3ZUU-S5MSb9&oq=what+is+I%2FO-bound&gs_lp=Egdnd3Mtd2l6IhF3aGF0IGlzIEkvTy1ib3VuZCoCCAAyBRAAGIAEMgUQABiABDIGEAAYFhgeMgYQABgWGB4yBhAAGBYYHjIGEAAYFhgeMgYQABgWGB4yBhAAGBYYHjIGEAAYFhgeMgYQABgWGB5IySZQAFjzDnAAeACQAQCYAdEBoAHLC6oBBTAuOC4xuAEByAEA-AEB-AECmAIJoALrC8ICBxAjGPAFGCfCAgQQIxgnwgIIEAAYgAQYsQPCAg4QLhiABBixAxjRAxjHAcICCxAuGIAEGLEDGIMBwgIREC4YgAQYsQMY0QMYgwEYxwHCAgsQABiABBixAxiDAcICChAjGIAEGCcYigXCAg4QABiABBixAxiDARiKBcICCxAAGIAEGLEDGIoFmAMAkgcFMC44LjGgB6BhsgcFMC44LjG4B-sLwgcFMC42LjPIBxY&sclient=gws-wiz&mstk=AUtExfDM4Pf8OKOGA8dJF30tdUBQOBSiDMMeWEcXAF0xGN3GWsgR3on0lPTISnkw7fzPQq-RJQyMyBRTIqcvrsp8kWPOdjPxqb-FEJAMPhSTJIGTm0j8x9ACq_jGPVlSgTCrtpr8malaNsK1m0cNh2AMYEOBTkjbXpvrfxrd_RLJyU2UN9U&csui=3&ved=2ahUKEwi5-NCgkq6QAxUkTGwGHTDODJcQgK4QegQIARAE)**that are limited by the CPU's processing power, an I/O-bound task's speed is bottlenecked by the speed of the I/O subsystem.**

****

**A white rectangular object with a black border

AI-generated content may be incorrect.**

**🧠 Explanation:**

* **Each thread runs in parallel (sort of — due to GIL, they don’t run CPU tasks simultaneously, but I/O can overlap).**
* **.start() begins execution.**
* **.join() waits for thread completion.**

**🔹 Concept**

* **Uses real OS threads (threading.Thread).**
* **Each thread runs independently — Python schedules them via the OS.**
* **However, due to the GIL (Global Interpreter Lock), only one thread executes Python bytecode at a time.**

**🔹 When to use**

**✅ Ideal for I/O-bound tasks  
✅ Good when code is blocking (like waiting for a network, file, or API)  
🚫 Not great for CPU-bound tasks (like data processing or computation-heavy work)**

* 1. **Asynchronous Programming with asyncio**

**asyncio is used for asynchronous (non-blocking) programming.  
It uses coroutines (functions declared with async def) that yield control using await.**

**✅ Perfect for:**

* **Network calls**
* **Database queries**
* **I/O without blocking the main thread**

****

**A white background with black text

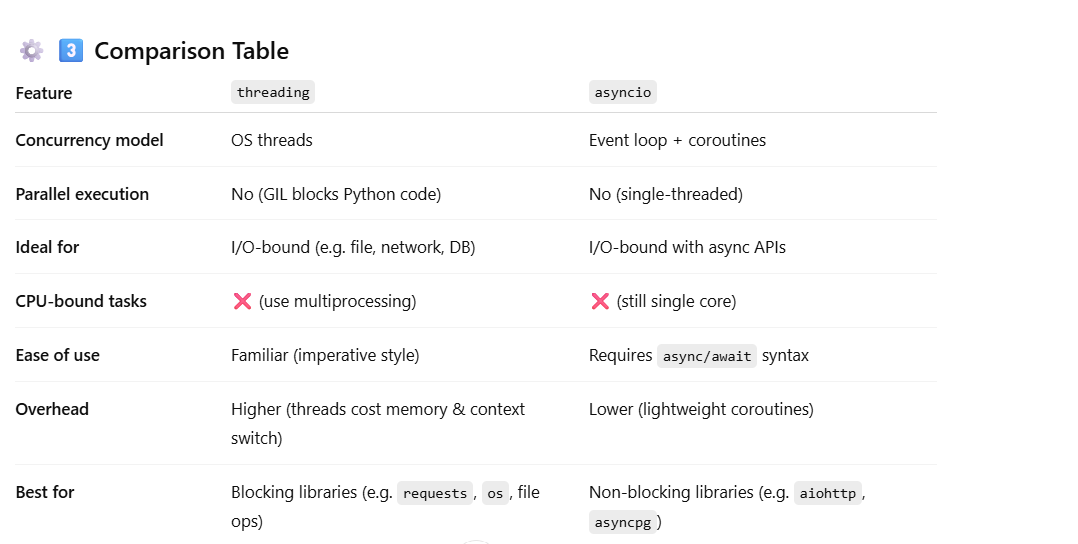
AI-generated content may be incorrect.**

**🧠 Explanation:**

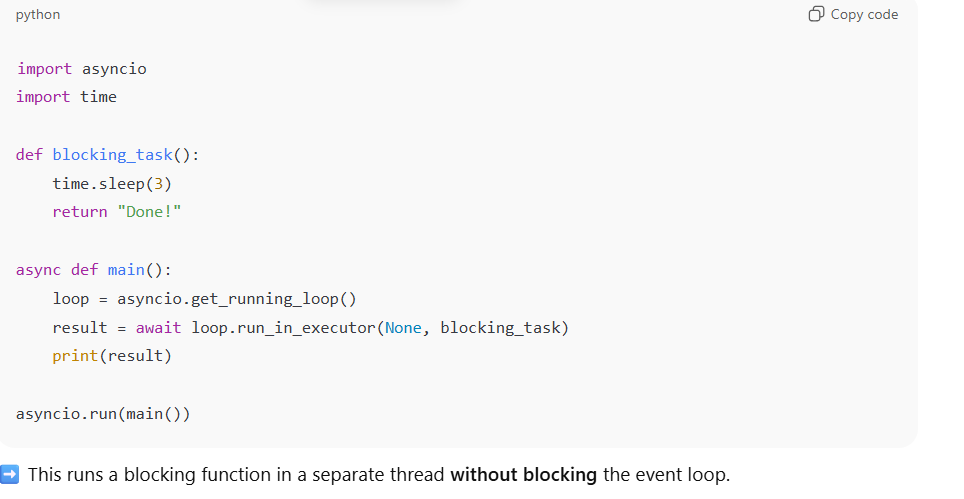
* **async def → defines a coroutine.**
* **await → pauses until the awaited task completes.**
* **asyncio.gather() → runs all coroutines concurrently.**
* **asyncio.run() → starts the event loop.**

**🔹 Concept**

* **Uses cooperative multitasking (no real threads).**
* **Tasks yield control explicitly using await.**
* **Runs on a single thread and single event loop.**
* **Non-blocking I/O is handled asynchronously using coroutines**
* **🔹 When to use**
* ✅ Excellent for **high-performance I/O-bound** programs (many concurrent tasks, like 10,000 requests)  
  🚫 Not for **CPU-bound** tasks (since it's still single-threaded)



For example, if you have an async web server (like FastAPI) that needs to run blocking code:



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

* 1. **Complex Example — Mixing Async + ThreadPoolExecutor**

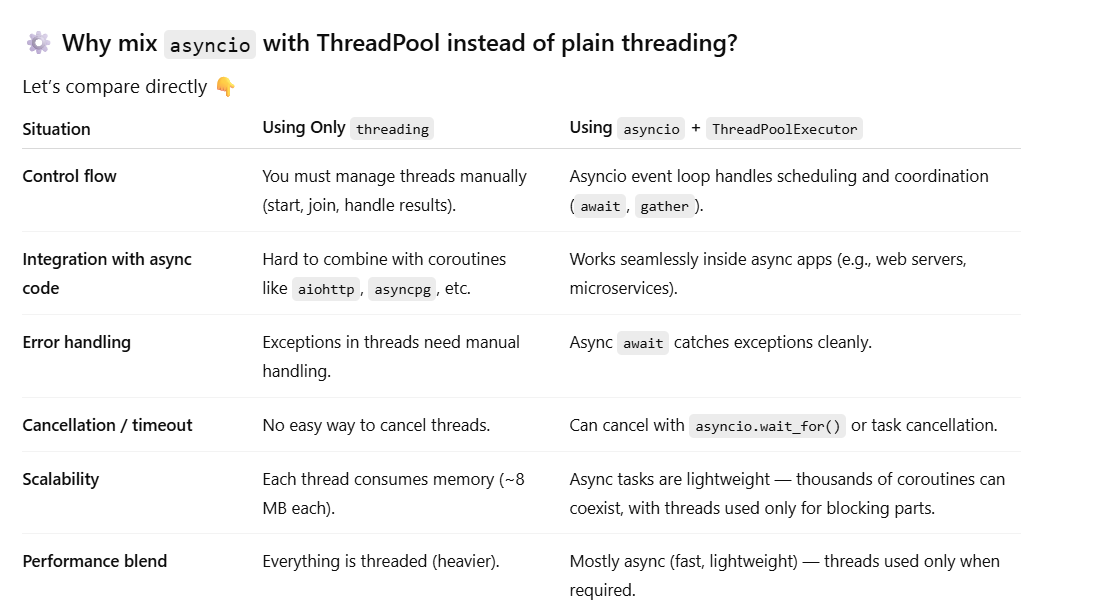
You can run **blocking code** (like CPU-bound or non-async functions) inside an async function using **executors**.

A computer screen shot of a program code

AI-generated content may be incorrect.

**🔍 What’s happening**

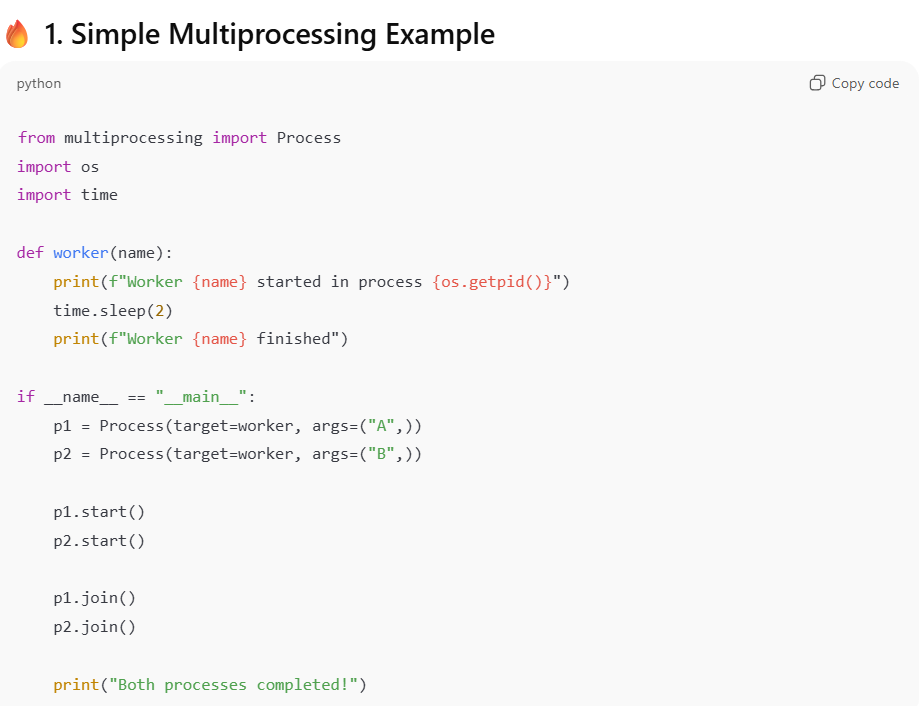
* You have an **async event loop** (via asyncio).
* You offload **blocking** work (time.sleep) to a ThreadPoolExecutor.
* The event loop **submits** the blocking function to background threads but remains **free to handle other async tasks** (like HTTP requests, DB I/O, etc.).
* You await the results using asyncio.gather() → meaning you still stay within async style.
* So — the blocking tasks run in **threads**, but they are **orchestrated** by the **async event loop**.



* 1. **Multiprocessing in Python**

The **multiprocessing** module allows you to run **multiple processes simultaneously**, each with its **own Python interpreter and memory space**.

✅ Unlike threads, **processes bypass the Global Interpreter Lock (GIL)** — so they can actually run **in parallel** on multiple CPU cores.  
That’s why multiprocessing is ideal for **CPU-bound tasks** (like computation, image processing, or data crunching).



**🧠 Explanation:**

* Each Process() runs independently in its own memory space.
* start() → launches the new process.
* join() → waits until that process finishes.
* os.getpid() → shows the unique process ID.

**Output:**

A white background with black and white clouds

AI-generated content may be incorrect.



A screen shot of a computer

AI-generated content may be incorrect.

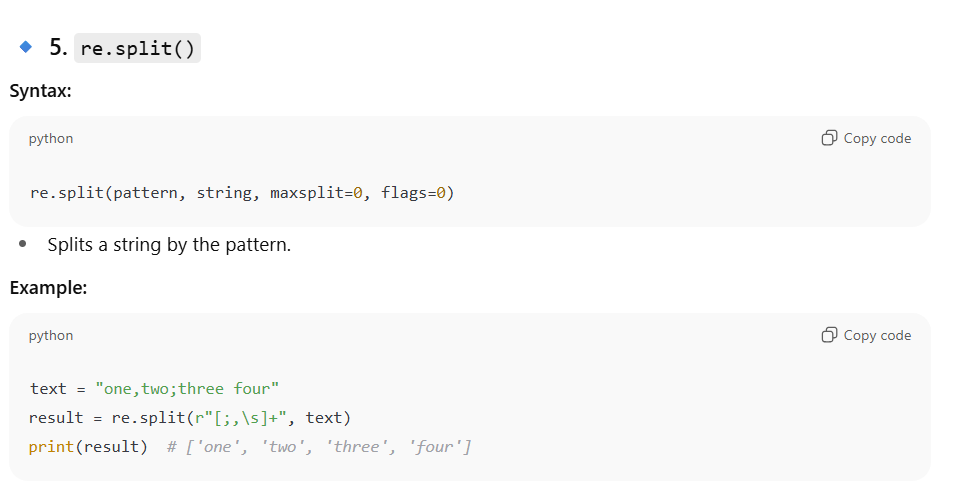
1. Python – Regex handling .



A screenshot of a computer code

AI-generated content may be incorrect.





A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

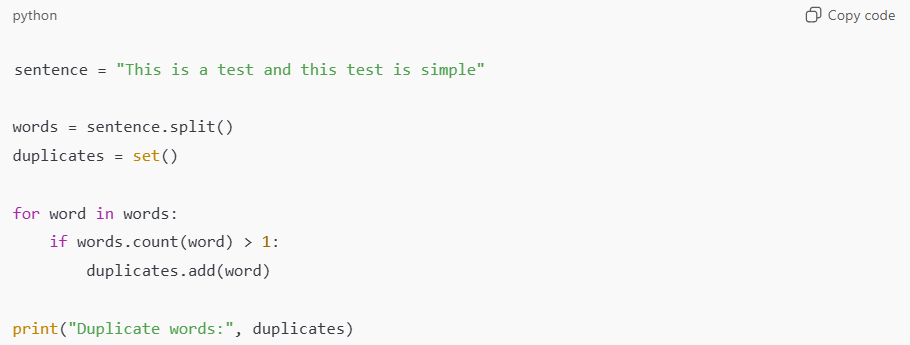
A screenshot of a computer

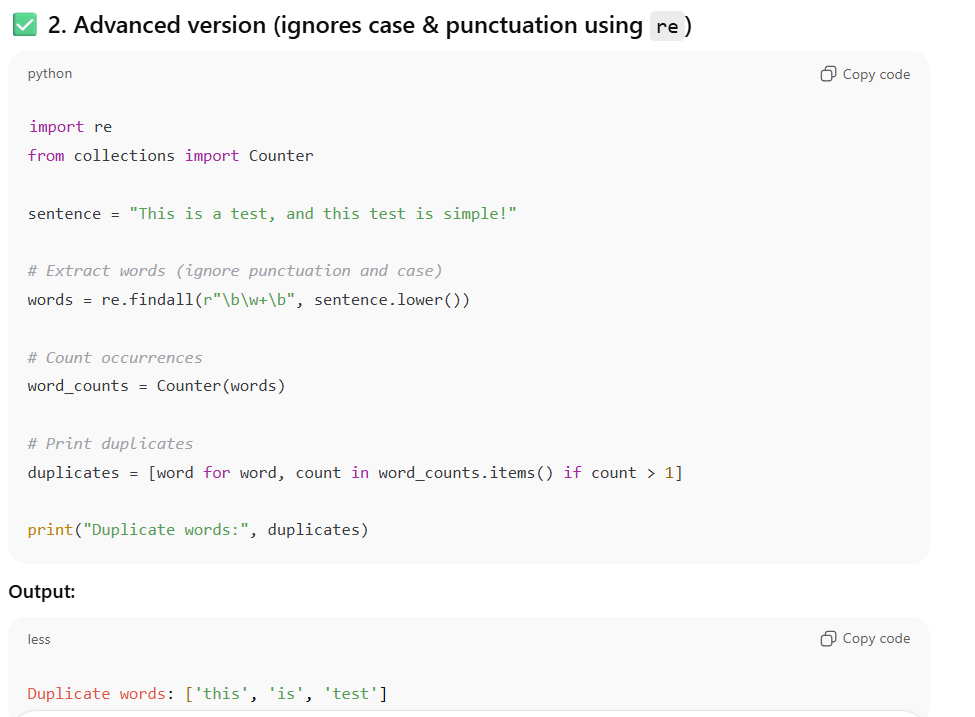
AI-generated content may be incorrect.

1. Python program to count word occurrences using a dictionary



to find and print **duplicate words** in a sentence [ Using count()]







A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Python – Class, Objects and Methods.**

**14.1 What is the difference between class and objects?**

🧩 What is a Class?

A class is a blueprint or template for creating objects.

It defines:

* Attributes (data / variables)
* Methods (functions / behaviors)

Think of a class as a design — it doesn’t hold actual data yet.



**🧍‍♂️ What is an Object?**

An **object** is a **real instance** of a class — created from that blueprint.

When you create an object, you fill in the actual data values.

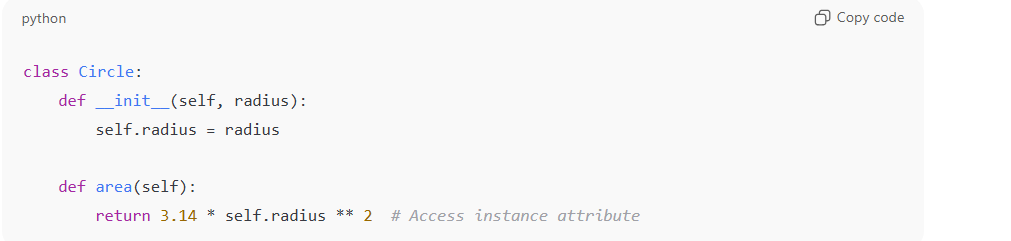
A screenshot of a computer

AI-generated content may be incorrect.

14.2 **Different Types of Methods**

**14.2.1 Instance Methods**

* **Works on an instance of the class.**
* **Can access instance attributes via self.**
* **Most common type.**

****

**14.2.2 Class methods**

* **Works on the class itself, not instance.**
* **Defined using the decorator @classmethod.**
* **First argument is cls (class reference).**
* **Can access class variables.**

****

**14.2.3 Static Methods**

* **Doesn’t access instance or class attributes.**
* **Defined using @staticmethod.**
* **Acts like a normal function inside class namespace.**

**A white background with a black border

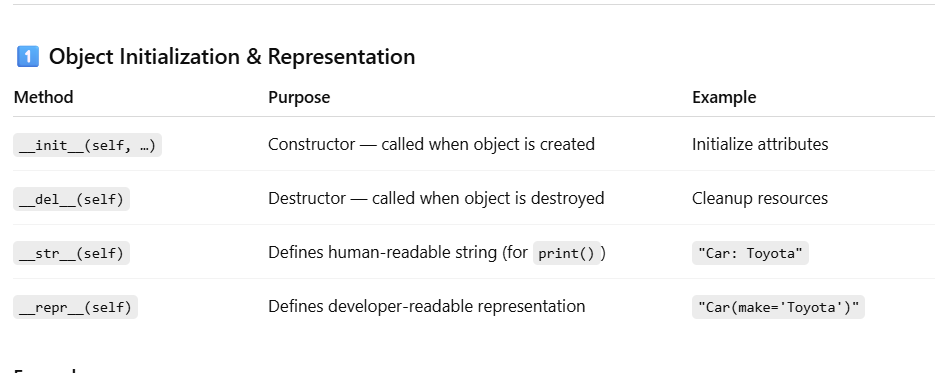
AI-generated content may be incorrect.**

1. **What Are Dunder (Special) Methods?**

Special methods let your objects behave like built-in types — by implementing them, you can:

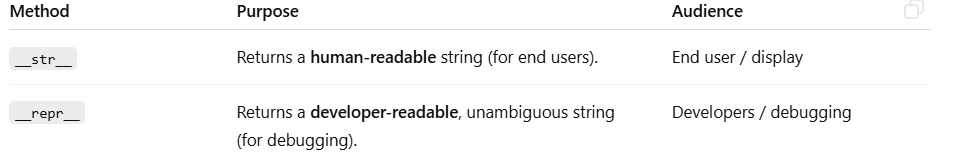
* Initialize objects
* Represent them as strings
* Compare them
* Add or multiply them (+, \* operators)
* Make them iterable, callable, etc.

They’re automatically called by Python’s built-in syntax, not directly by you.

****

**What Are \_\_str\_\_ and \_\_repr\_\_?**

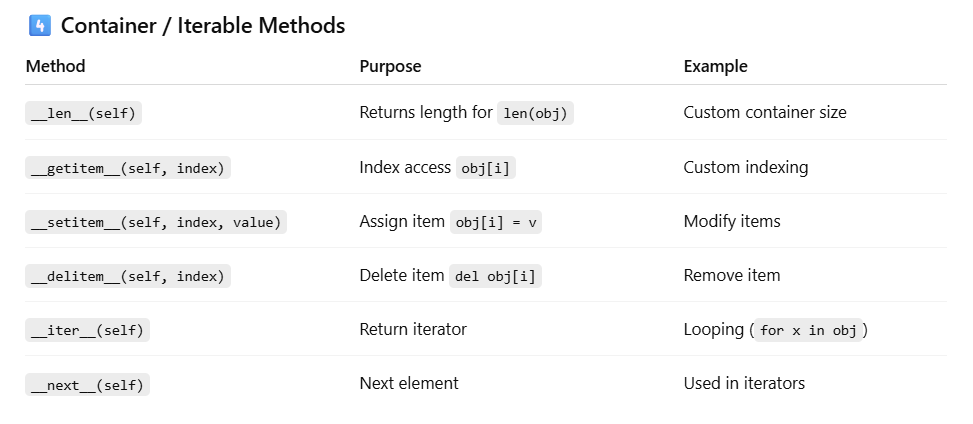
**Used to represent an object as a string .**

****

**A screen shot of a computer code

AI-generated content may be incorrect.**

****

****

**A screenshot of a computer

AI-generated content may be incorrect.**

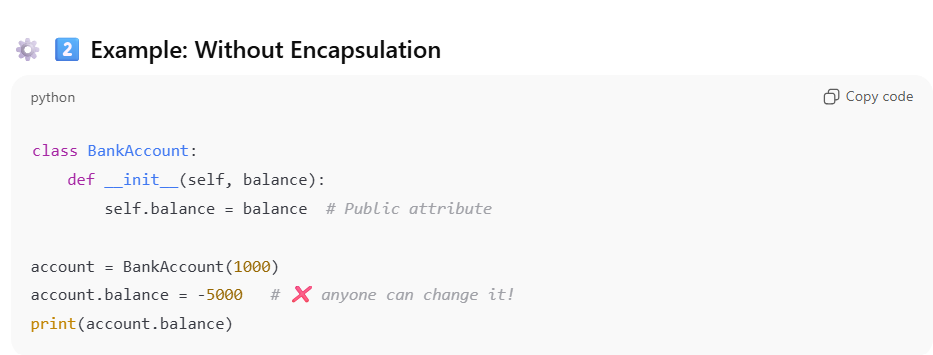
1. **What is encapsulation ?**

**Encapsulation** means **bundling data (attributes)** and **methods (functions)** that operate on that data **into a single unit — a class**.  
It also means **restricting direct access** to some parts of an object to protect the data from being modified accidentally.

**💬 In Simple Terms:**

Encapsulation = **data hiding + data protection**.

You expose only what’s necessary through **methods or properties**, and keep the rest **private**.

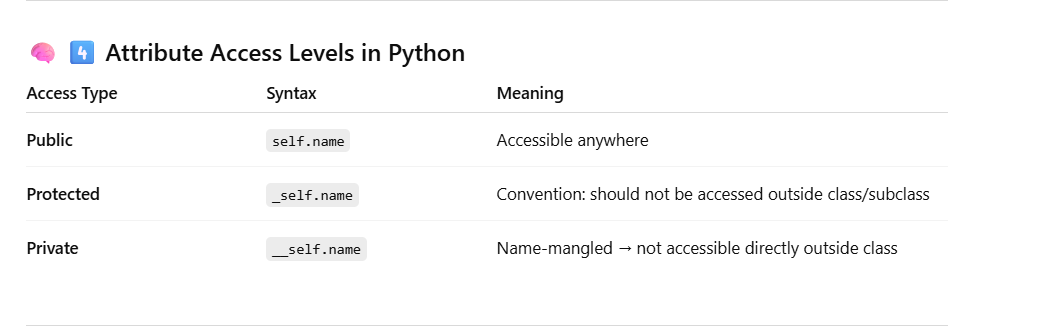


A screenshot of a computer

AI-generated content may be incorrect.

✅ Here:

* You can’t access account.\_\_balance directly (it’s hidden).
* Access is controlled through methods like deposit(), withdraw(), and get\_balance().



1. **@property** in Python is the **modern, Pythonic way** to implement **encapsulation** (data hiding + controlled access).



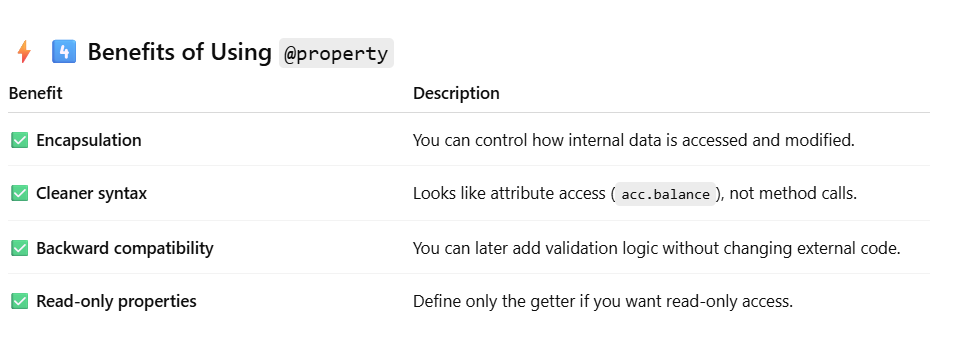
A screenshot of a computer program

AI-generated content may be incorrect.

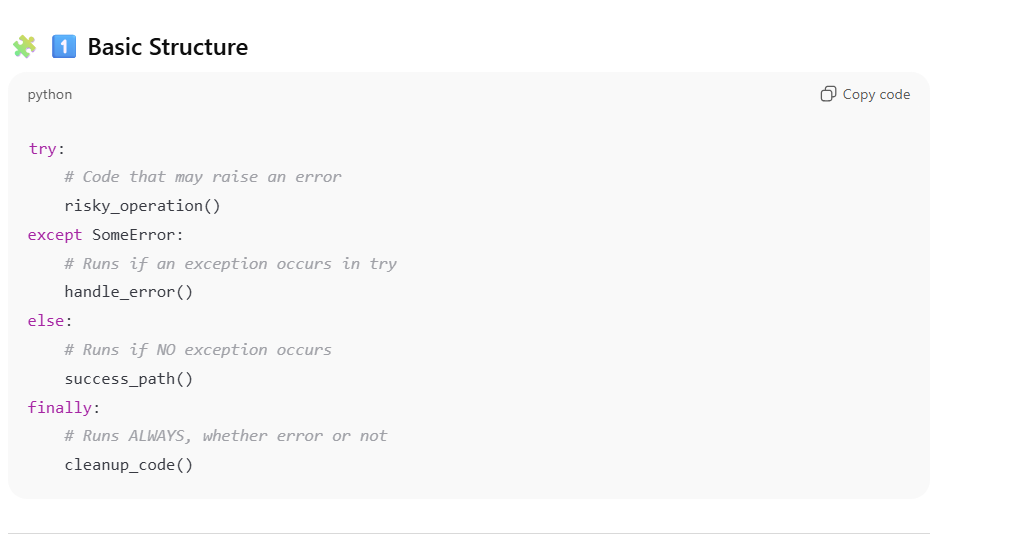
**🧠 3️⃣ What’s Happening Behind the Scenes**

* @property → Turns a **method** into a **read-only attribute**.
* @balance.setter → Adds a **setter** for that same property.
* @balance.deleter → (Optional) Allows controlled deletion.

So even though balance looks like a normal variable, it’s **actually managed by methods**.



1. Python – try, catch, else, finally



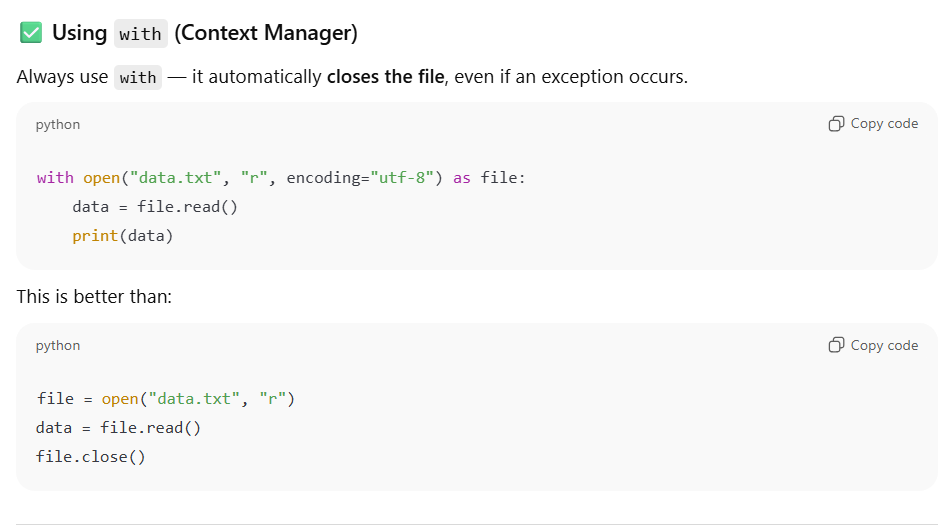
A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

1. **Advanced Python File Handling Basics**

****

**A white background with text and numbers

AI-generated content may be incorrect.**

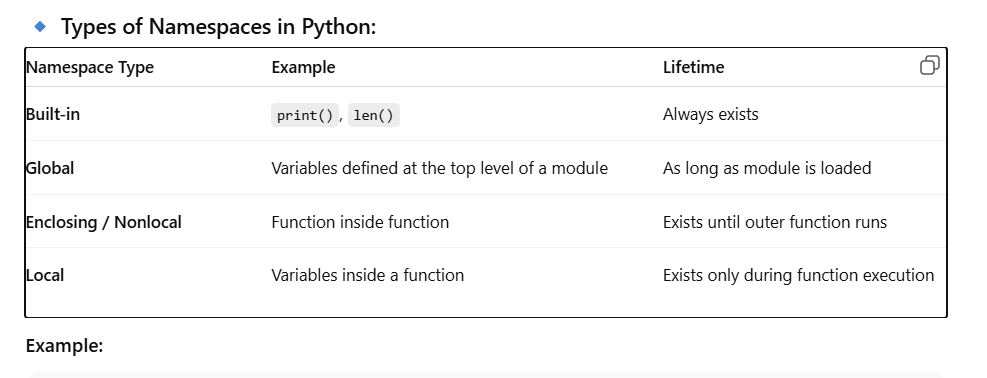
****

**A screenshot of a computer program

AI-generated content may be incorrect.**

1. **What are Python namespaces and scopes?**

**A namespace is like a mapping of names to objects — basically, it’s a container that holds all variable and function names in a program and their corresponding objects.**

****

**A white screen with black text

AI-generated content may be incorrect.**

**In Python, scope determines visibility of a variable.**

**🔹 LEGB Rule (Python’s Scope Resolution Order)**

**Python searches for a name in this order:**

**L – Local: Names defined inside the current function**

**E – Enclosing: Names in enclosing functions (nested functions)**

**G – Global: Names at the top level of module**

**B – Built-in: Names pre-defined by Python (len, print, etc.)**

**Global and Nonlocal Keywords**

* **global → Allows modifying a global variable inside a function**
* **nonlocal → Allows modifying a variable in enclosing function**

****

**A white background with a black and white flag

AI-generated content may be incorrect.**

1. **What is \_\_name\_\_?**

**In Python, every module (i.e., .py file) has a special built-in variable called \_\_name\_\_.**

* **When a module is run directly, \_\_name\_\_ is set to "\_\_main\_\_".**
* **When a module is imported into another module, \_\_name\_\_ is set to the module’s name.**

**What does if \_\_name\_\_ == "\_\_main\_\_": mean?**

**It is used to check whether a script is run directly or imported.**

****

**✅ Explanation:**

* **Because we ran module\_a.py directly, Python sets \_\_name\_\_ = "\_\_main\_\_".**
* **The if \_\_name\_\_ == "\_\_main\_\_": block executes.**

****

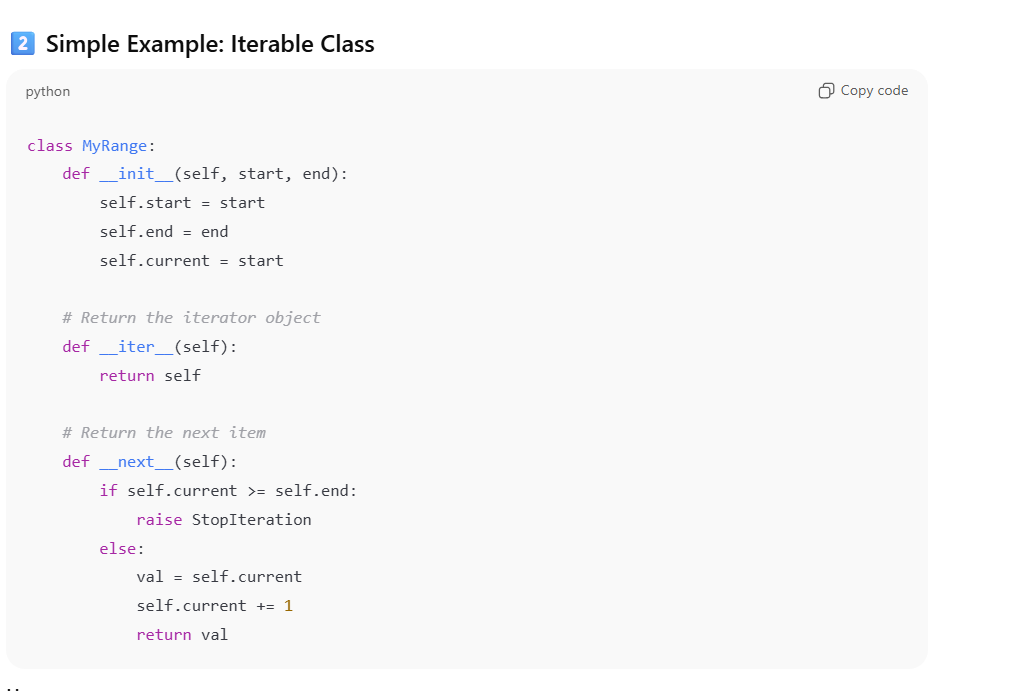
**✅ Explanation:**

1. **module\_a is imported**
   * **Python executes the code in module\_a.py to load the module.**
   * **But since it’s imported, \_\_name\_\_ in module\_a is set to "module\_a", not "\_\_main\_\_"**
   * **Therefore, the if \_\_name\_\_ == "\_\_main\_\_": block does NOT run.**
2. **main.py is run directly**
   * **\_\_name\_\_ in main.py is "\_\_main\_\_"**
   * **So any code inside if \_\_name\_\_ == "\_\_main\_\_": (if present) would execute.**
3. **How to make a class iterable?**

**A class is iterable if you can loop over its items using for.**

**Python requires two things:**

1. **\_\_iter\_\_() method → returns an iterator object (usually self)**
2. **\_\_next\_\_() method → returns the next value and raises StopIteration when done**

****