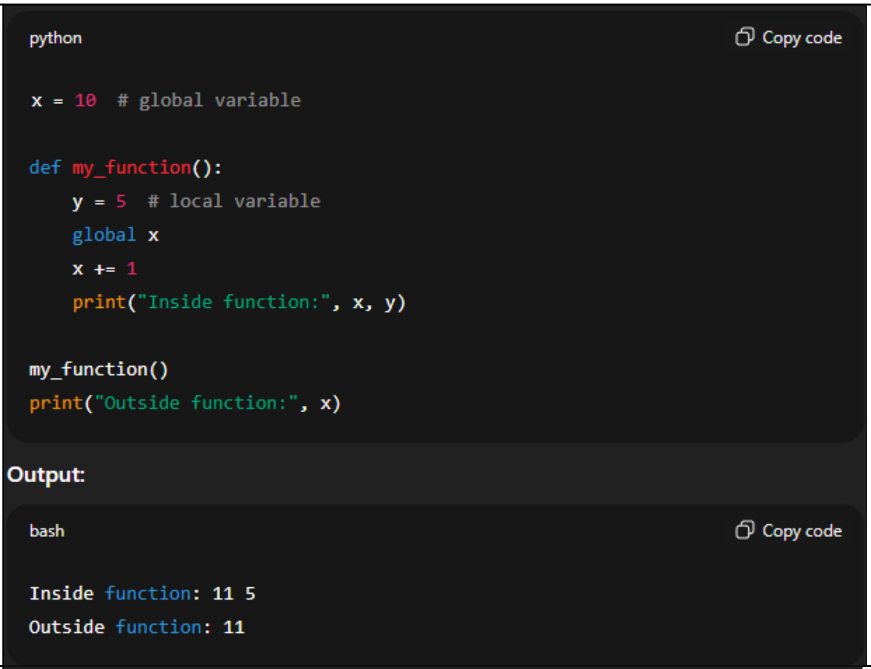
Python		
Basic	What is Python? Enlist some of its benefits. Python is a high-level, interpreted, and general-purpose programming language. It is known for its simple and readable syntax, which makes it easy to learn and use. Some benefits of Python include: - Easy to learn and read – clean syntax that resembles English. - Versatile and cross-platform – works on Windows, macOS, Linux, etc. - Large standard library and ecosystem – many built-in modules and third-party packages. - Supports multiple programming paradigms – procedural, object-oriented, and functional programming. - Strong community support – many tutorials, forums, and resources available.	
	Can you tell us if Python is object-oriented or functional programming? Python is a multi-paradigm programming language, which means it supports both object-oriented programming (OOP) and functional programming. - Object-oriented programming: Python allows you to define classes, create objects, and use inheritance and polymorphism. - Functional programming: Python supports functions as first-class objects, higher-order functions, map, filter, reduce, and lambda expressions. So, Python is both object-oriented and functional, depending on how you write your code.	

What rules govern local and global variables in Python?

In Python, variable scope determines where a variable can be accessed:
1. Local variables are defined inside a function and can only be accessed within that function. They are created when the function starts and destroyed when it ends.
2. Global variables are defined outside any function and can be accessed anywhere in the file.
3. To modify a global variable inside a function, you must use the global keyword.



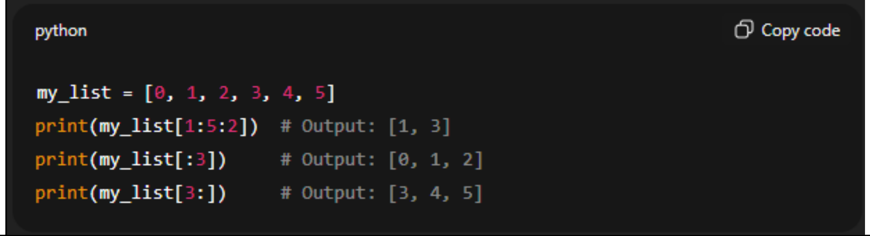
```python
x = 10  # global variable

def my_function():
    y = 5  # local variable
    global x
    x += 1
    print("Inside function:", x, y)

my_function()
print("Outside function:", x)
```

Output:

```bash
Inside function: 11 5
Outside function: 11
```

Can you tell us what is slicing in Python?

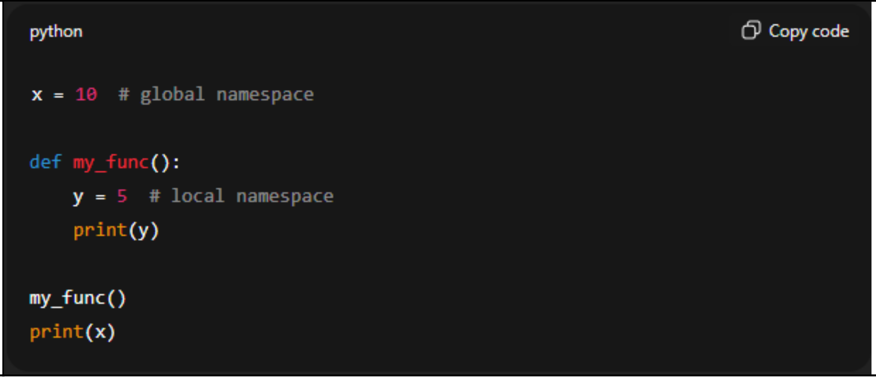
Slicing in Python is a way to extract a portion of a sequence, like a list, string, or tuple, by specifying a start index, stop index, and optional step.



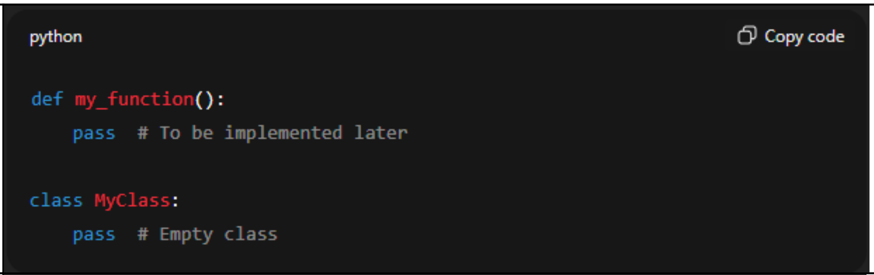
```python
my_list = [0, 1, 2, 3, 4, 5]
print(my_list[1:5:2])  # Output: [1, 3]
print(my_list[:3])     # Output: [0, 1, 2]
print(my_list[3:])     # Output: [3, 4, 5]
```

What is namespace in Python?

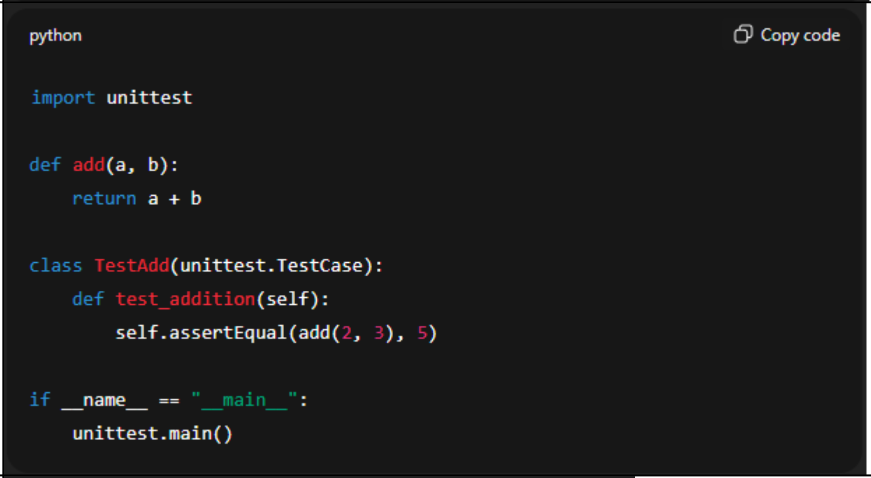
A namespace in Python is a container that holds names (identifiers) and maps them to objects. In other words, it is a system that keeps track of all the variable names and their corresponding objects in memory.
Types of namespaces:
1. Local namespace – inside a function.
2. Global namespace – at the module level.
3. Built-in namespace – Python's built-in functions and exceptions.



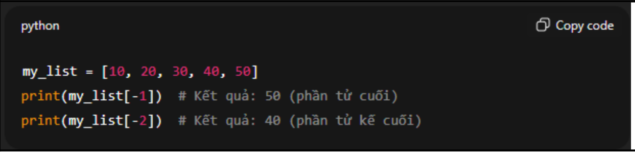
```python
x = 10  # global namespace

def my_func():
    y = 5  # local namespace
    print(y)

my_func()
print(x)
```

	What is pass in Python? "pass" is a null statement that allows your code to run without errors even if the function or class has no content yet.		

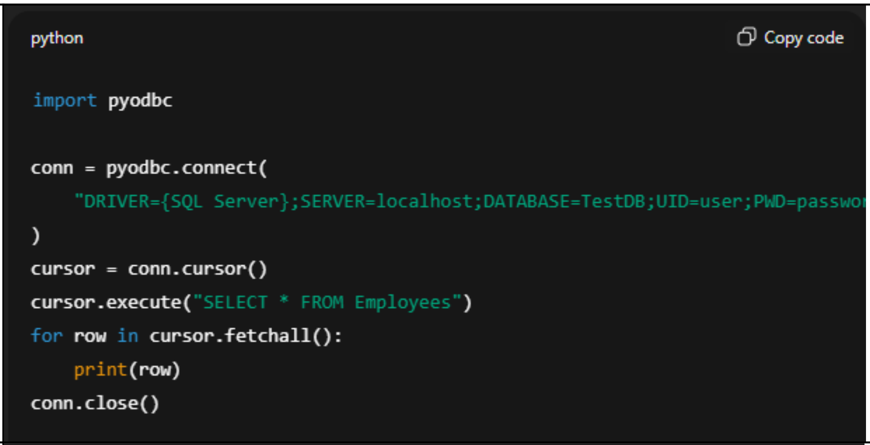
```python
def my_function():
    pass  # To be implemented later

class MyClass:
    pass  # Empty class
```

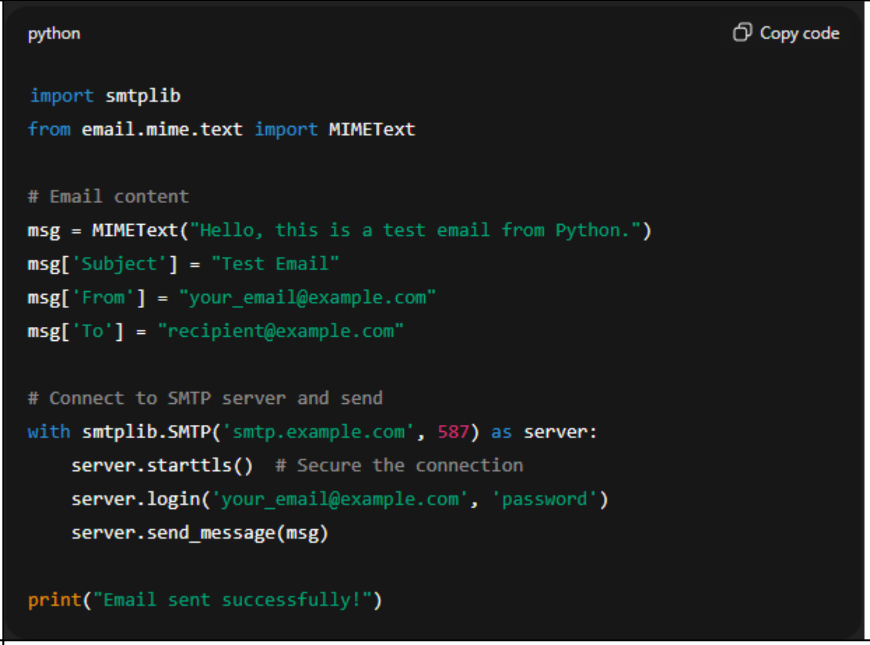
	Can you explain what is unittest in Python? "unittest is Python's built-in framework for writing and running unit tests to ensure that individual parts of the code work correctly."		

```python
import unittest

def add(a, b):
    return a + b

class TestAdd(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(add(2, 3), 5)

if __name__ == "__main__":
    unittest.main()
```

	What are negative indexes in Python? "Negative indexes in Python let you access elements from the end of a sequence. -1 is the last element, -2 the second last, and so on, so you don't need the sequence length to access items from the end."		

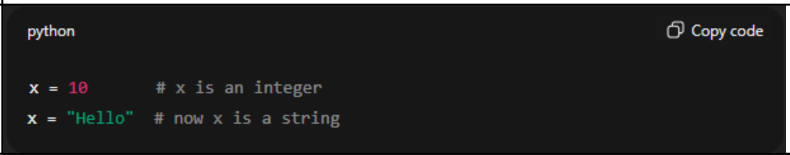
```python
my_list = [10, 20, 30, 40, 50]
print(my_list[-1])  # Kết quả: 50 (phần tử cuối)
print(my_list[-2])  # Kết quả: 40 (phần tử kế cuối)
```

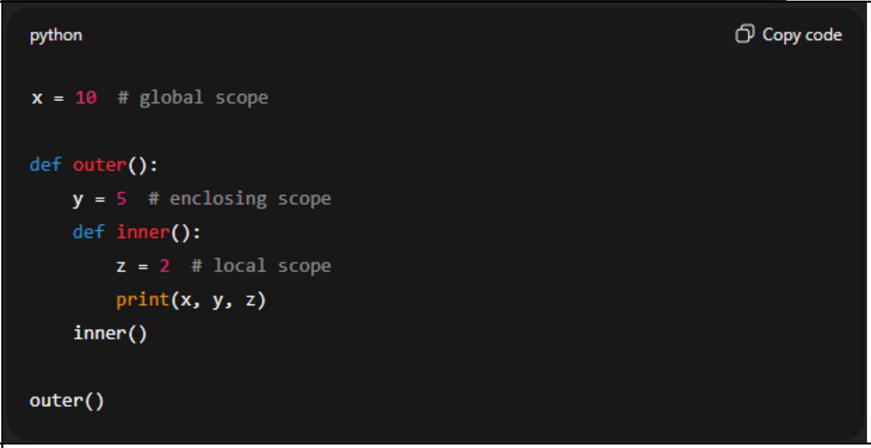
	What are ODBC modules in Python?
	
	"ODBC modules in Python let programs connect to databases via the ODBC standard, run SQL queries, and manage data. Common ones are pyodbc and pypyodbc."
		
		python Copy code
		`import pyodbc`
		
		`conn = pyodbc.connect(`
		`    "DRIVER={SQL Server};SERVER=localhost;DATABASE=TestDB;UID=user;PWD=passwo`
		`)`
		`cursor = conn.cursor()`
		`cursor.execute("SELECT * FROM Employees")`
		`for row in cursor.fetchall():`
		`    print(row)`
		`conn.close()`

	How will you send an email from a Python Script?
	
	"You can send emails in Python using smtplib to connect to an SMTP server and email to format the message."
		
		python Copy code
		`import smtplib`
		`from email.mime.text import MIMEText`
		
		`# Email content`
		`msg = MIMEText("Hello, this is a test email from Python.")`
		`msg['Subject'] = "Test Email"`
		`msg['From'] = "your_email@example.com"`
		`msg['To'] = "recipient@example.com"`
		
		`# Connect to SMTP server and send`
		`with smtplib.SMTP('smtp.example.com', 587) as server:`
		`    server.starttls()  # Secure the connection`
		`    server.login('your_email@example.com', 'password')`
		`    server.send_message(msg)`
		
		`print("Email sent successfully!")`

	What is PEP 8 and its importance? "PEP 8 is Python's style guide for writing clean, readable code. It ensures consistency, improves collaboration, and makes code easier to maintain."	

	What are the key features of Python? Python has several key features: 1. Easy to learn and readable – simple syntax, close to English. 2. Interpreted language – runs code line by line. 3. Dynamically typed – no need to declare variable types. 4. Cross-platform – runs on Windows, macOS, Linux, etc. 5. Extensive standard library – many modules for different tasks. 6. Object-oriented and functional – supports multiple programming paradigms. 7. Open-source – free to use with strong community support.	

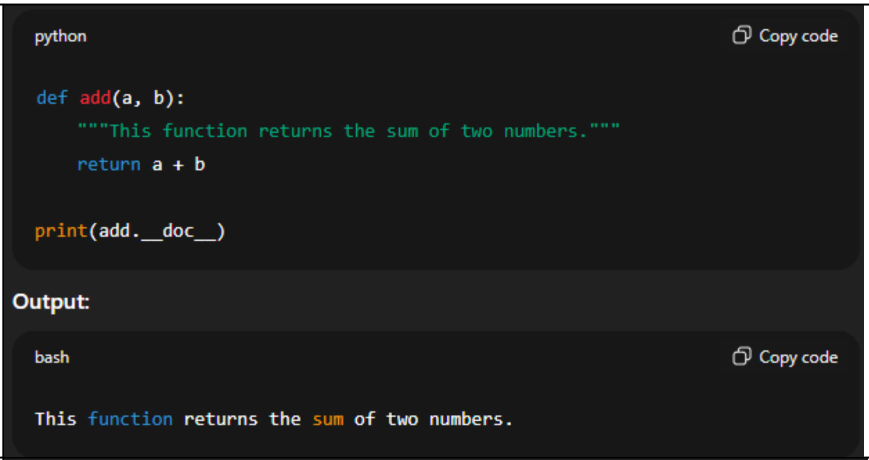
	What does it mean to be dynamically typed in Python?
	
	"Dynamically typed means you do not need to say the type of a variable. Python finds it when the program runs, and you can change the type later."
		
		`x = 10       # x is an integer`
		`x = "Hello"  # now x is a string`

	What is a scope in Python?
	
	"Scope in Python is where a variable can be used. Types: local (inside a function), enclosing (nested functions), global (whole file), built-in (Python’s own names)."
		
		`x = 10  # global scope`
		
		`def outer():`
		`    y = 5  # enclosing scope`
		`    def inner():`
		`        z = 2  # local scope`
		`        print(x, y, z)`
		`    inner()`
		
		`outer()`

	How can Python script be executable on Unix? To make a Python script executable on Unix: 1. Add the shebang line at the top of the script to specify the Python interpreter: #!/usr/bin/env python3 2. Make the script executable using the chmod command: chmod +x script.py 3. Run the script directly from the terminal: ./script.py This allows the script to be executed like a standalone program.	

What is Docstring in Python?

"A docstring is a special string in Python used to explain a module, class, or function. It goes in triple quotes at the start and can be seen with __doc__."



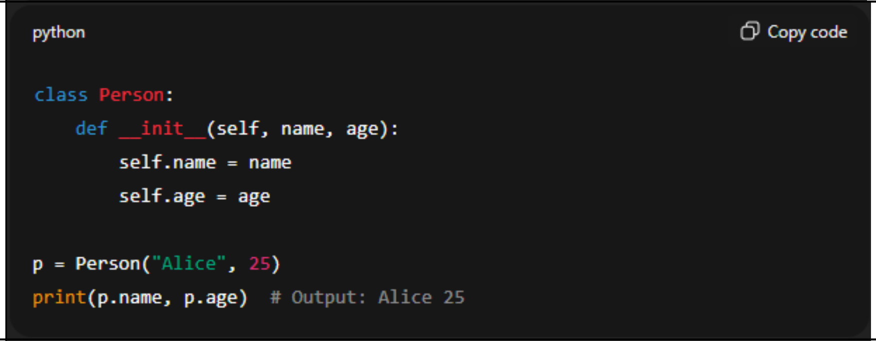
```python
def add(a, b):
    """This function returns the sum of two numbers."""
    return a + b

print(add.__doc__)
```

Output:

```bash
This function returns the sum of two numbers.
```

What is init in Python?

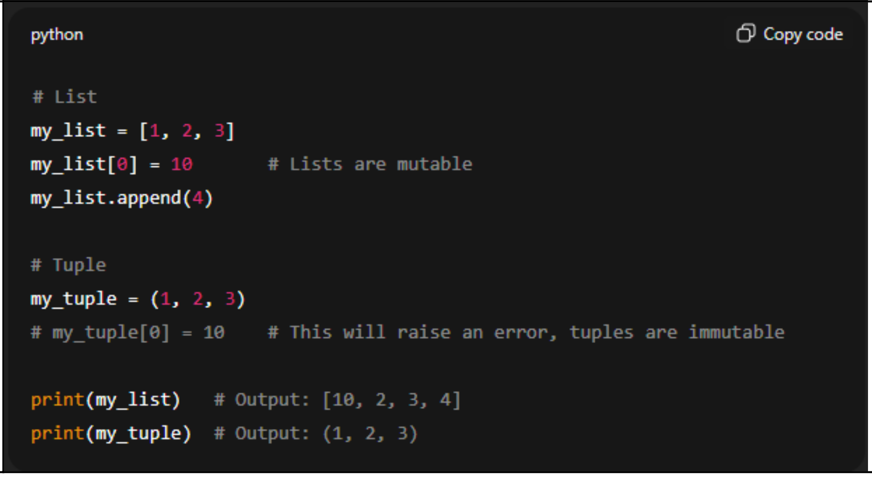
"__init__ is a special method in a class that runs when a new object is made. It sets up the object's attributes."



```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Alice", 25)
print(p.name, p.age)  # Output: Alice 25
```

What are lists and tuples in Python?

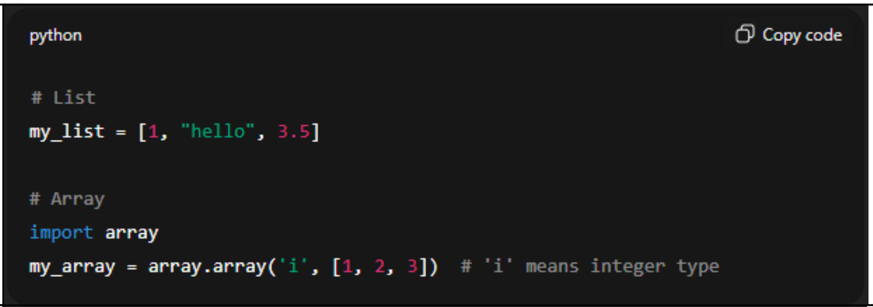
In Python:
- List is an ordered, mutable collection of items. You can change, add, or remove elements. Lists are defined using square brackets [].
- Tuple is an ordered, immutable collection of items. Once created, its elements cannot be changed. Tuples are defined using parentheses ().



```python
# List
my_list = [1, 2, 3]
my_list[0] = 10       # Lists are mutable
my_list.append(4)

# Tuple
my_tuple = (1, 2, 3)
# my_tuple[0] = 10    # This will raise an error, tuples are immutable

print(my_list)   # Output: [10, 2, 3, 4]
print(my_tuple)  # Output: (1, 2, 3)
```

What is the difference between Arrays and lists in Python?

Lists can hold different types and are flexible. Arrays hold one type, use less memory, and are faster for large numeric data.

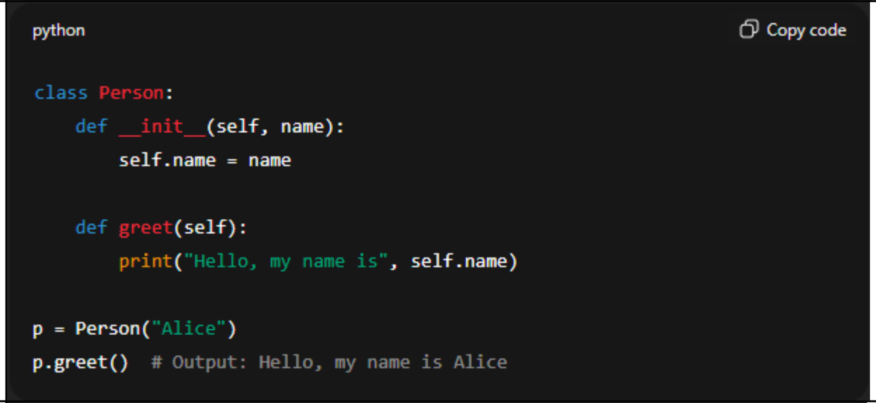


python Copy code

```python
# List
my_list = [1, "hello", 3.5]

# Array
import array
my_array = array.array('i', [1, 2, 3])  # 'i' means integer type
```

What is Self-used for in Python?

"Self is a reference to the current object in a class. It lets you access the object’s attributes and methods inside the class."

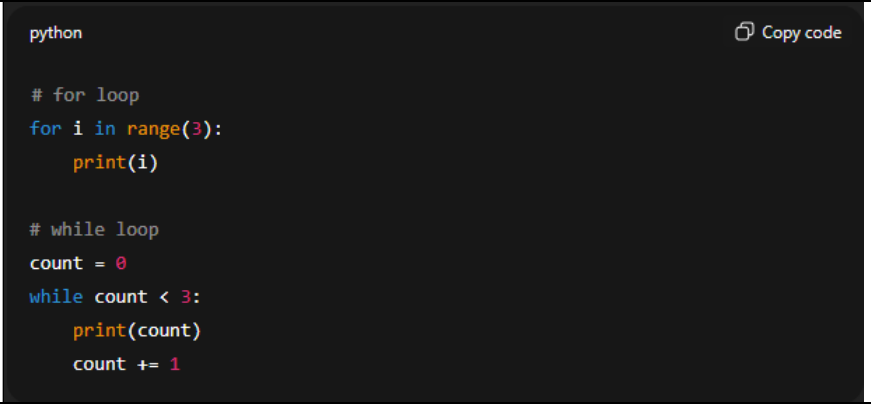


python Copy code

```python
class Person:
    def __init__(self, name):
        self.name = name

    def greet(self):
        print("Hello, my name is", self.name)

p = Person("Alice")
p.greet()  # Output: Hello, my name is Alice
```

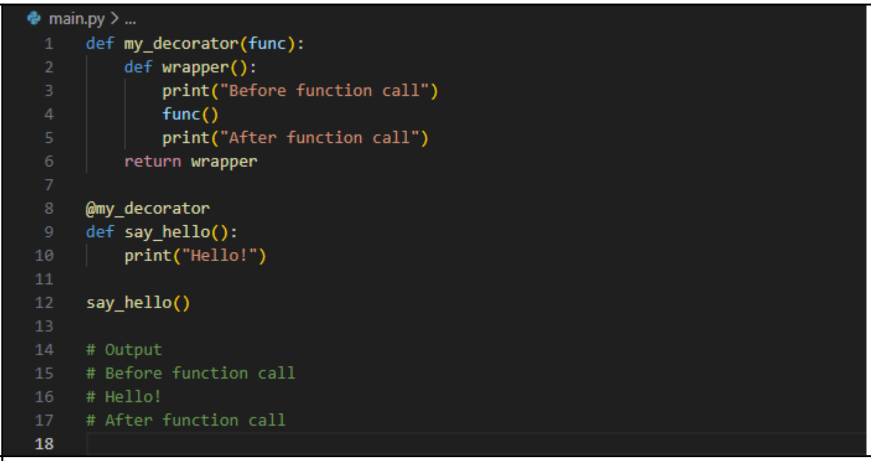
What are the major two-loop statements in Python?

"Python has two main loops: for loops to go through items in a sequence, and while loops to repeat code while a condition is true."

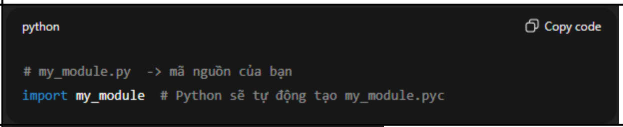


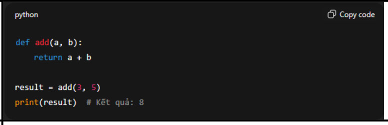
python Copy code

```python
# for loop
for i in range(3):
    print(i)

# while loop
count = 0
while count < 3:
    print(count)
    count += 1
```

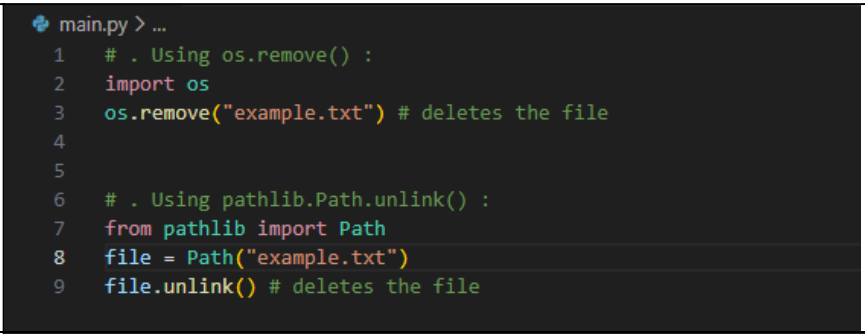
	What are Decorators in Python?
	
	A decorator in Python is a function that modifies or enhances another function or method without changing its code. Decorators are often used for logging, access control, timing, or caching. They are applied using the @decorator_name syntax above a function definition.
		main.py > ...
		`def my_decorator(func):`
		`    def wrapper():`
		`        print("Before function call")`
		`        func()`
		`        print("After function call")`
		`    return wrapper`
		
		`@my_decorator`
		`def say_hello():`
		`    print("Hello!")`
		
		`say_hello()`
		
		`# Output`
		`# Before function call`
		`# Hello!`
		`# After function call`

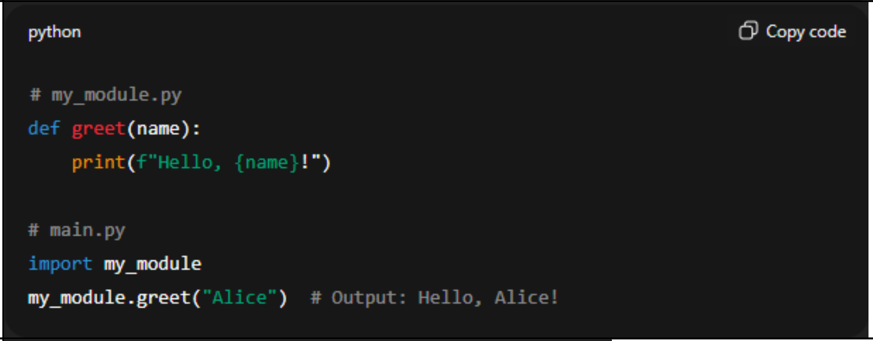
	What built-in types are available in Python? Python provides several built-in data types. The main ones include: 1. Numeric types: int, float, complex 2. Sequence types: list, tuple, range 3. Text type: str 4. Set types: set, frozenset 5. Mapping type: dict 6. Boolean type: bool 7. Binary types: bytes, bytearray, memoryview These types allow you to store and manipulate different kinds of data.	

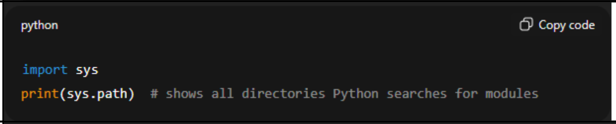
	How do you differentiate between .py and .pyc files in Python?
	
	"A .py file is the Python source code you write. A .pyc file is the compiled bytecode Python makes to run the program faster."
		python Copy code
		`# my_module.py  -> mã nguồn của bạn`
		`import my_module  # Python sẽ tự động tạo my_module.pyc`

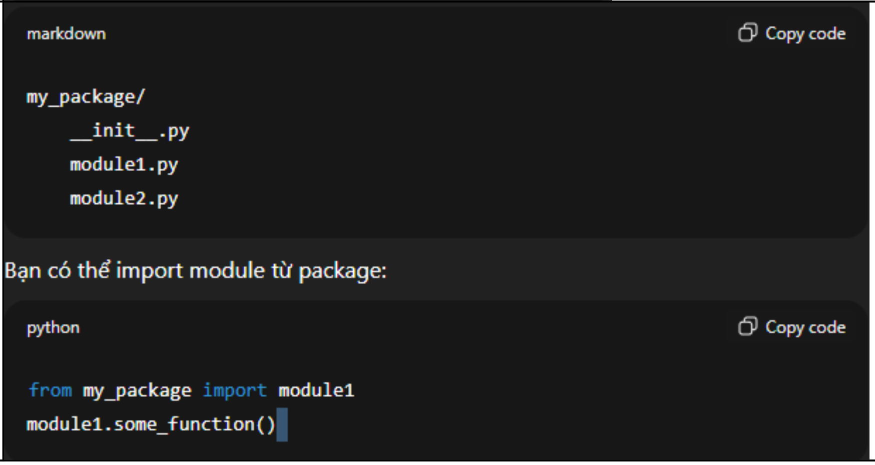
	How do you create a Python function?
	
	"Use def to make a function, add a name and inputs, write the code inside, and use return to give back a value."
		python Copy code
		`def add(a, b):`
		`    return a + b`
		
		`result = add(3, 5)`
		`print(result)  # Kết quả: 8`

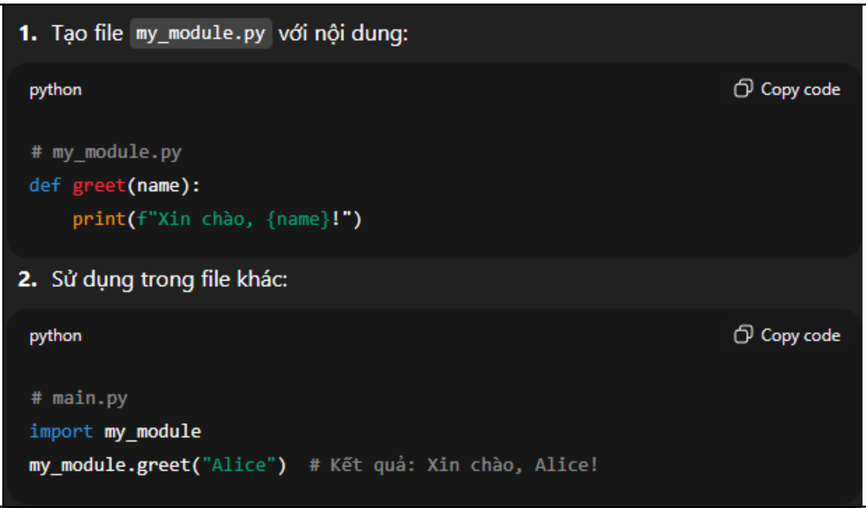
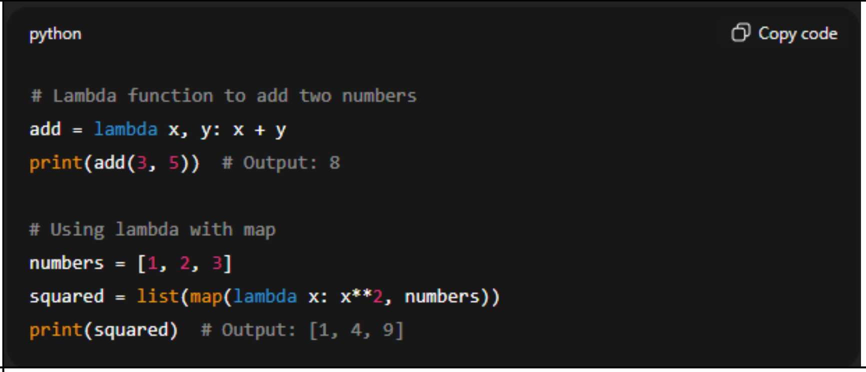
	How does a function return values in Python? "A Python function gives back a value using return. Without return, it gives back None."	

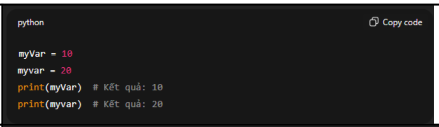
	What commands are used to delete Python files?
	
	In Python, you can delete files using the os or pathlib modules.
	Using os.remove():
	Using pathlib.Path.unlink():
		main.py > ...
		`# . Using os.remove() :`
		`import os`
		`os.remove("example.txt") # deletes the file`
		
		`# . Using pathlib.Path.unlink() :`
		`from pathlib import Path`
		`file = Path("example.txt")`
		`file.unlink() # deletes the file`

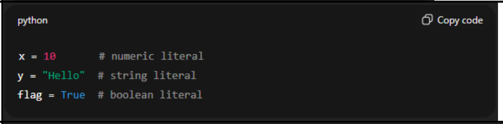
	What are modules in Python?
	
	"A module is a Python file with code (functions, classes, variables) that you can import and use in other files."
		python Copy code
		
		`# my_module.py`
		`def greet(name):`
		`    print(f"Hello, {name}!")`
		
		`# main.py`
		`import my_module`
		`my_module.greet("Alice")  # Output: Hello, Alice!`

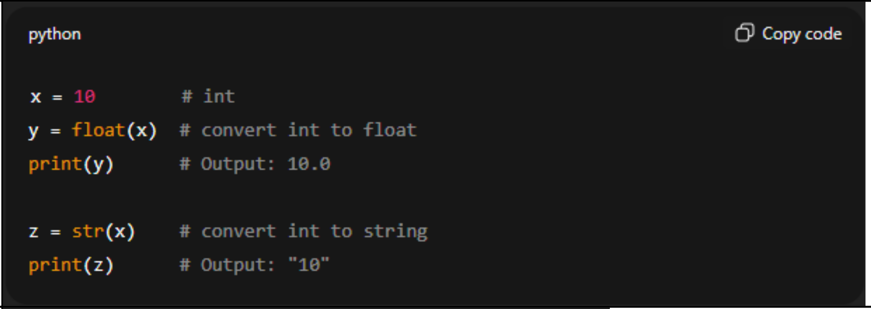
	What is a Python PATH?
	
	"Python PATH is a list of folders where Python looks for modules when you use import. It’s in sys.path and you can add folders if needed."
		python Copy code
		
		`import sys`
		`print(sys.path)  # shows all directories Python searches for modules`

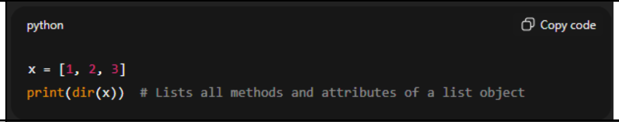
	What is a Package in Python?
	
	"A package is a folder with Python modules and an __init__.py file that organizes them together."
		markdown Copy code
		
		`my_package/`
		`    __init__.py`
		`    module1.py`
		`    module2.py`
		
		Bạn có thể import module từ package:
		
		python Copy code
		
		`from my_package import module1`
		`module1.some_function()`

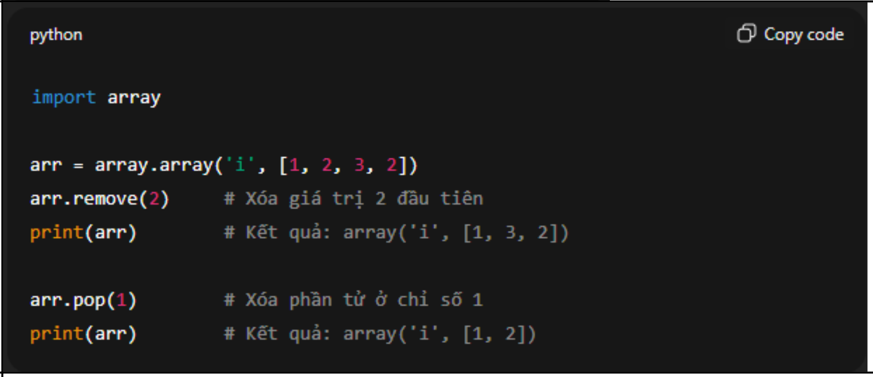
	Create a module in Python. In Python, a module is simply a Python file with a .py extension that contains code such as functions, variables, or classes. You can then import it in another Python file to use its contents. "A module is a Python file with functions or classes that can be imported in another file."	 **1.** Tạo file `my_module.py` với nội dung: python Copy code `# my_module.py` `def greet(name):` `    print(f"Xin chào, {name}!")` **2.** Sử dụng trong file khác: python Copy code `# main.py` `import my_module` `my_module.greet("Alice")  # Kết quả: Xin chào, Alice!`
	What is lambda in Python? "A lambda is a small, anonymous Python function with only one expression that can take multiple arguments."	 python Copy code `# Lambda function to add two numbers` `add = lambda x, y: x + y` `print(add(3, 5))  # Output: 8` `# Using lambda with map` `numbers = [1, 2, 3]` `squared = list(map(lambda x: x**2, numbers))` `print(squared)  # Output: [1, 4, 9]`
	How memory can be managed in Python? "Python manages memory automatically using reference counting (tracks object use), garbage collection (removes unused objects), and dynamic allocation from its private heap. You usually don't need to free memory yourself."	
	What are keywords in Python? "Keywords are reserved words in Python that have special meanings and cannot be used as variable or function names." Keywords: if, else, for, while, def, class, import, return, try, except, finally,...	

	Is Python a case-sensitive language?
	
	"Yes, Python is case-sensitive, so names like myVar and myvar are different."
		
		python Copy code
		`myVar = 10`
		`myvar = 20`
		`print(myVar)  # Kết quả: 10`
		`print(myvar)  # Kết quả: 20`

	What are literals in Python?
	
	"A literal is a fixed value written in the code, like numbers (10), strings ("Hello"), or Booleans (True)."
		
		python Copy code
		`x = 10       # numeric literal`
		`y = "Hello"  # string literal`
		`flag = True  # boolean literal`

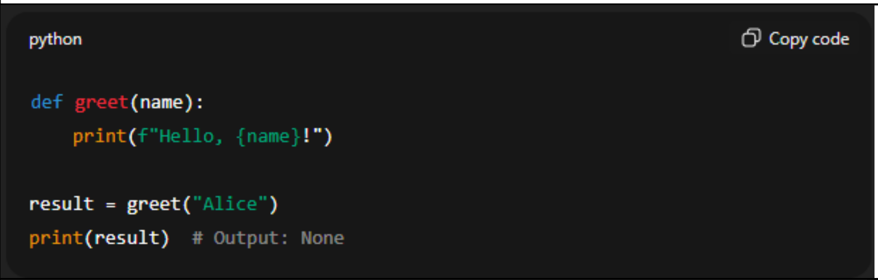
	What is Type Conversion in Python?
	
	Type conversion is changing a value from one type to another using functions like int(), float(), or str().
		
		python Copy code
		`x = 10        # int`
		`y = float(x)  # convert int to float`
		`print(y)      # Output: 10.0`
		
		`z = str(x)    # convert int to string`
		`print(z)      # Output: "10"`

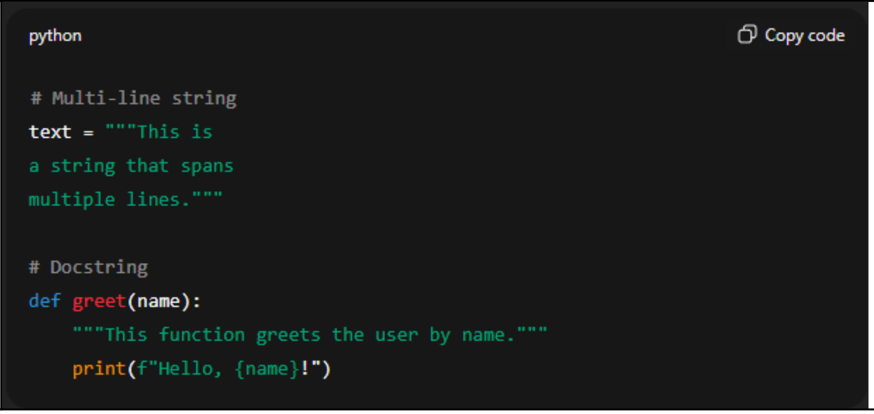
	What do you think is the use of dir () function in Python?
	
	"The dir() function shows all attributes and methods of an object, class, or module."
		
		python Copy code
		`x = [1, 2, 3]`
		`print(dir(x))  # Lists all methods and attributes of a list object`

	How can you remove values from an array in Python?
	
	"In Python arrays, use remove() to delete a value or pop() to delete by index (last element if no index)."
		
		python Copy code
		`import array`
		
		`arr = array.array('i', [1, 2, 3, 2])`
		`arr.remove(2)     # Xóa giá trị 2 đầu tiên`
		`print(arr)        # Kết quả: array('i', [1, 3, 2])`
		
		`arr.pop(1)        # Xóa phần tử ở chỉ số 1`
		`print(arr)        # Kết quả: array('i', [1, 2])`

	How is list stored in memory in Python?	
	How is tuple stored in memory in Python?	

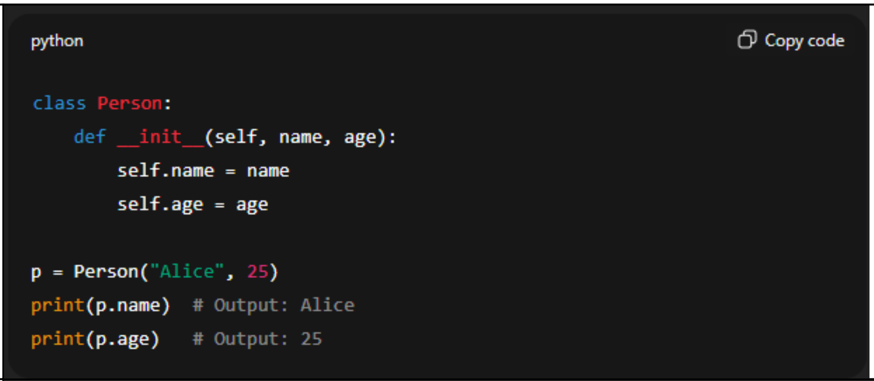
	How are list, tuple, set, array, and dictionary stored in memory in Python? List, tuple, set, array và dictionary trong Python được lưu trữ trong bộ nhớ như thế nào?	

Intermediate	Is it possible for a function not to have a return statement and is it valid?
	
	"Python function can have no return. It is valid and will return None by default."
		
		python
		Copy code
		`def greet(name):`
		`    print(f"Hello, {name}!")`
		
		`result = greet("Alice")`
		`print(result)  # Output: None`

	When should Python use triple quotes as a delimiter?
	
	"Use triple quotes in Python for multi-line strings or for docstrings to document modules, classes, or functions."
		
		python
		Copy code
		`# Multi-line string`
		`text = """This is`
		`a string that spans`
		`multiple lines."""`
		
		`# Docstring`
		`def greet(name):`
		`    """This function greets the user by name."""`
		`    print(f"Hello, {name}!")`

What is the main role of the init method? Give a code block example.

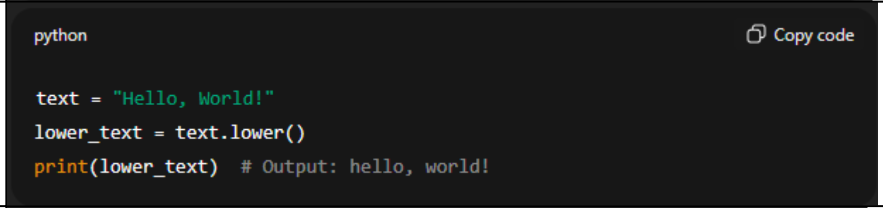
The __init__ method is used to set up a class's attributes when a new object is created.



```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Alice", 25)
print(p.name)  # Output: Alice
print(p.age)   # Output: 25
```

How do you convert string to lowercase in Python?

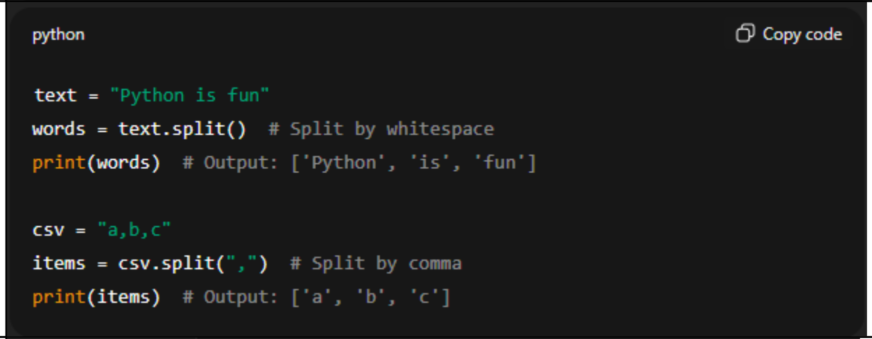
In Python, you can convert a string to lowercase using the lower() method of the string.



```python
text = "Hello, World!"
lower_text = text.lower()
print(lower_text)  # Output: hello, world!
```

How do you use the split method in Python?

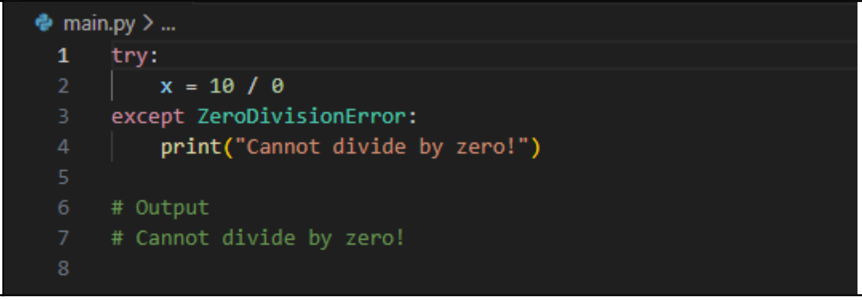
In Python, the split() method is used to divide a string into a list of substrings based on a separator. By default, it splits by whitespace.



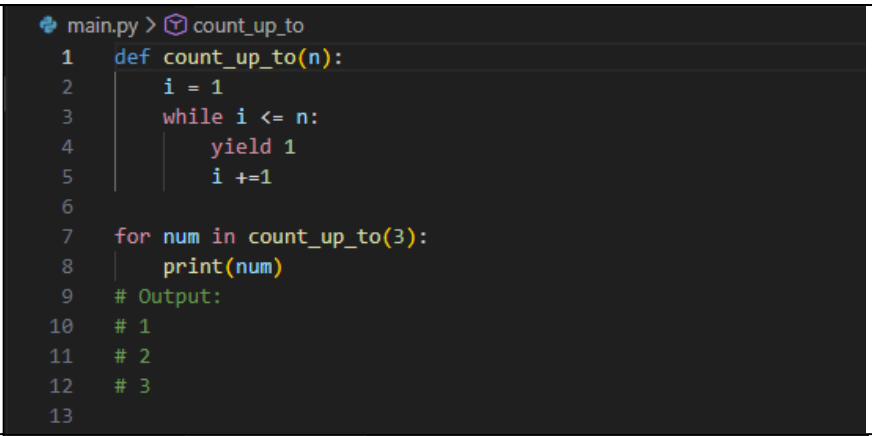
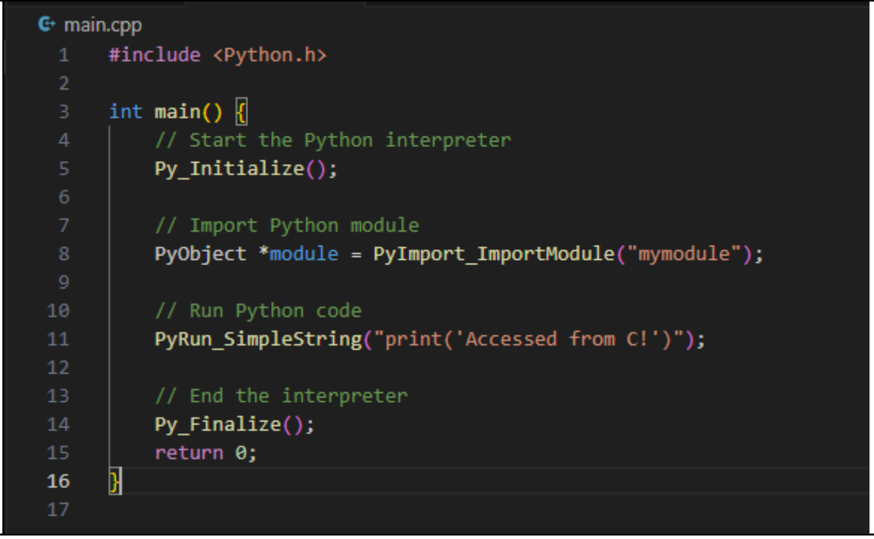
```python
text = "Python is fun"
words = text.split()  # Split by whitespace
print(words)  # Output: ['Python', 'is', 'fun']

csv = "a,b,c"
items = csv.split(",")  # Split by comma
print(items)  # Output: ['a', 'b', 'c']
```

What is a Try Block?

A try block is used to run code that may cause an error. If an error happens, it can be handled in an except block so the program doesn't crash.



```python
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")

# Output
# Cannot divide by zero!
```

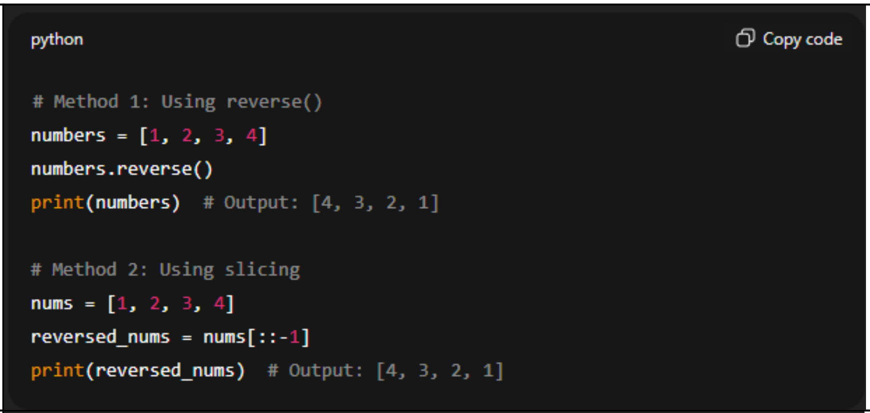
	What are generators in Python? A generator is a function that gives values one by one using yield, which helps save memory.	
	How can a module written in Python be accessed from C? "You can access a Python module from C by embedding the Python interpreter and using the Python/C API."	

```python
def count_up_to(n):
    i = 1
    while i <= n:
        yield 1
        i +=1

for num in count_up_to(3):
    print(num)
# Output:
# 1
# 2
# 3
```

```cpp
#include <Python.h>

int main() {
    // Start the Python interpreter
    Py_Initialize();

    // Import Python module
    PyObject *module = PyImport_ImportModule("mymodule");

    // Run Python code
    PyRun_SimpleString("print('Accessed from C!')");

    // End the interpreter
    Py_Finalize();
    return 0;
}
```

How a list can be reversed in Python?

You can reverse a list by using list.reverse() (changes the list) or list[::-1] (makes a reversed copy).

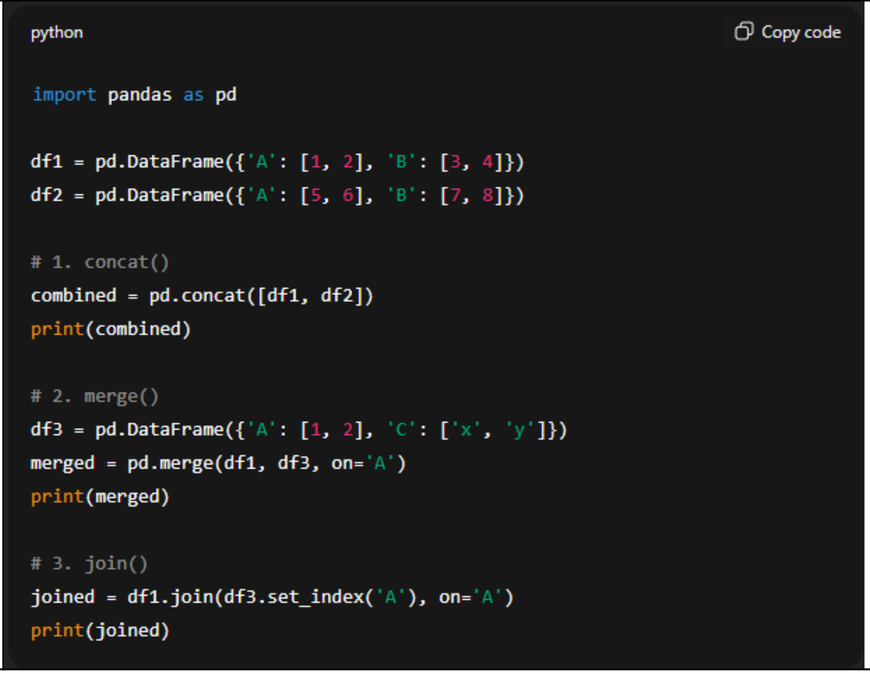


```python
# Method 1: Using reverse()
numbers = [1, 2, 3, 4]
numbers.reverse()
print(numbers)  # Output: [4, 3, 2, 1]

# Method 2: Using slicing
nums = [1, 2, 3, 4]
reversed_nums = nums[::-1]
print(reversed_nums)  # Output: [4, 3, 2, 1]
```

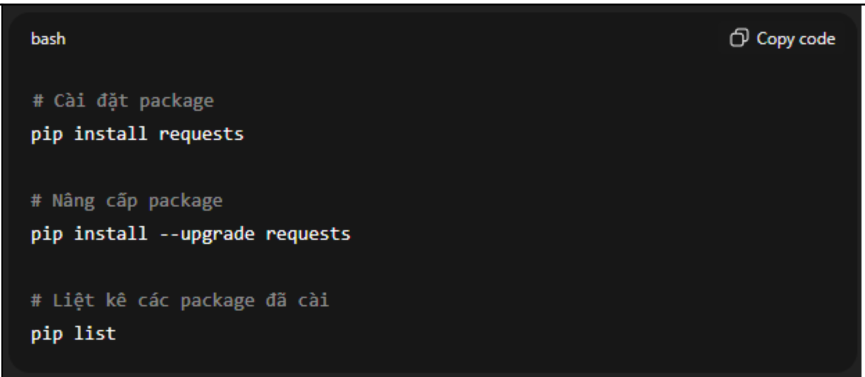
What are ways to combine dataframes in Python?

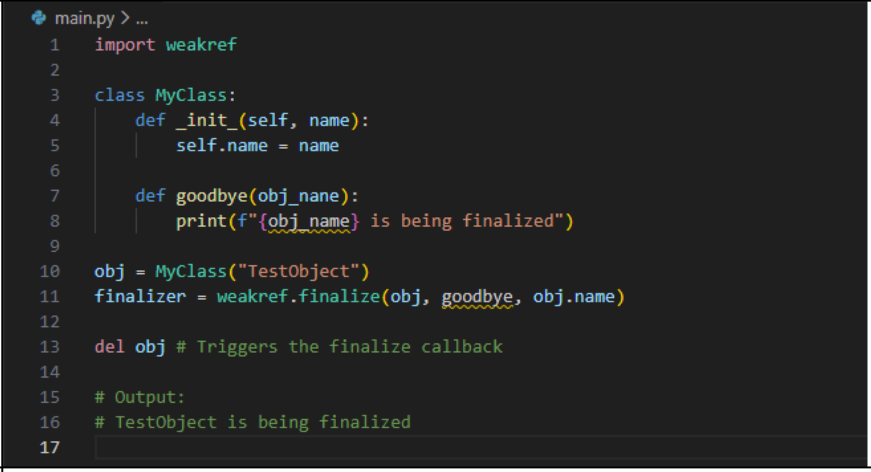
In Python (using pandas), there are several ways to combine DataFrames depending on how you want to merge the data:
1. concat() – Combines DataFrames along rows or columns.
2. merge() – Combines DataFrames based on common columns or keys (like SQL joins).
3. join() – Combines DataFrames based on their indexes.

"We can combine DataFrames using concat() for stacking, merge() for SQL-style joins, and join() for index-based merging."



```python
import pandas as pd

df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})

# 1. concat()
combined = pd.concat([df1, df2])
print(combined)

# 2. merge()
df3 = pd.DataFrame({'A': [1, 2], 'C': ['x', 'y']})
merged = pd.merge(df1, df3, on='A')
print(merged)

# 3. join()
joined = df1.join(df3.set_index('A'), on='A')
print(joined)
```

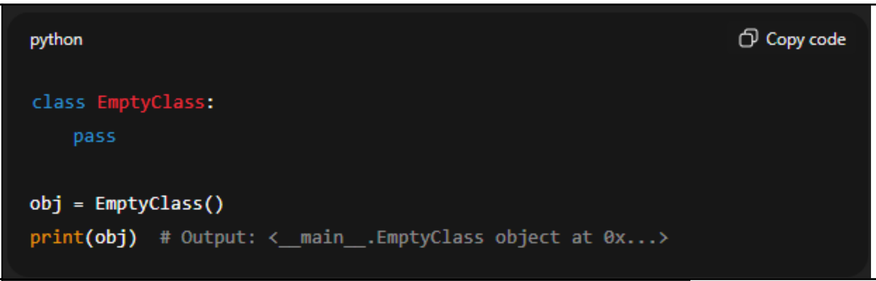
	What is a PIP?
	
	"PIP is Python’s package installer, used to install and manage third-party libraries."
		
		bash
		`# Cài đặt package`
		`pip install requests`
		
		`# Nâng cấp package`
		`pip install --upgrade requests`
		
		`# Liệt kê các package đã cài`
		`pip list`

	Why is finalize used in Python?
	
	In Python, finalize (from the weakref module) is used to register a callback function that will be called when an object is about to be garbage collected. It is helpful for cleaning up resources like closing files or network connections when an object is no longer needed, without relying on __del__().
	
	"finalize is used to register a callback for cleanup when an object is about to be garbage collected."
		
		main.py > ...
		`import weakref`
		
		`class MyClass:`
		`    def __init__(self, name):`
		`        self.name = name`
		
		`    def goodbye(obj_nane):`
		`        print(f"{obj_name} is being finalized")`
		
		`obj = MyClass("TestObject")`
		`finalizer = weakref.finalize(obj, goodbye, obj.name)`
		
		`del obj # Triggers the finalize callback`
		
		`# Output:`
		`# TestObject is being finalized`

	Differentiate between override and new modifiers. The difference between override and new modifiers (common in languages like C#; Python doesn’t have explicit modifiers but conceptually similar behavior can exist) is: 1. Override: - Used when a derived class wants to provide a new implementation for a method already defined in the base class. - Ensures polymorphic behavior, so the derived class method is called even when using a base class reference. 2. New: - Used when a derived class defines a method with the same name as in the base class but does not override it. - It hides the base class method; the base class version is called when the object is referenced as the base type.	

In Python, what would you do to create an empty class?

In Python, to create an empty class, you define the class with the class keyword and use pass inside to indicate no content.

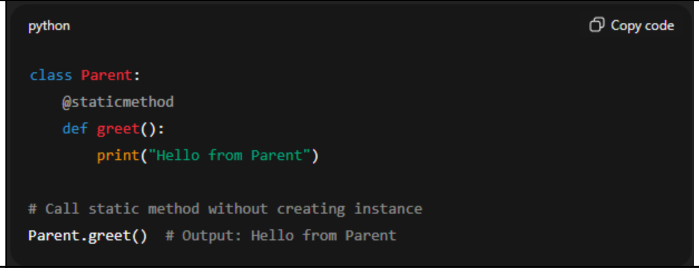


```python
class EmptyClass:
    pass

obj = EmptyClass()
print(obj)  # Output: <__main__.EmptyClass object at 0x...>
```

Do you think you can call the parent class without its instance creation?

In Python, you can call methods or access attributes of a parent class without creating an instance of it if the method is a class method (@classmethod) or a static method (@staticmethod). For regular instance methods, you normally need an instance of the parent class.

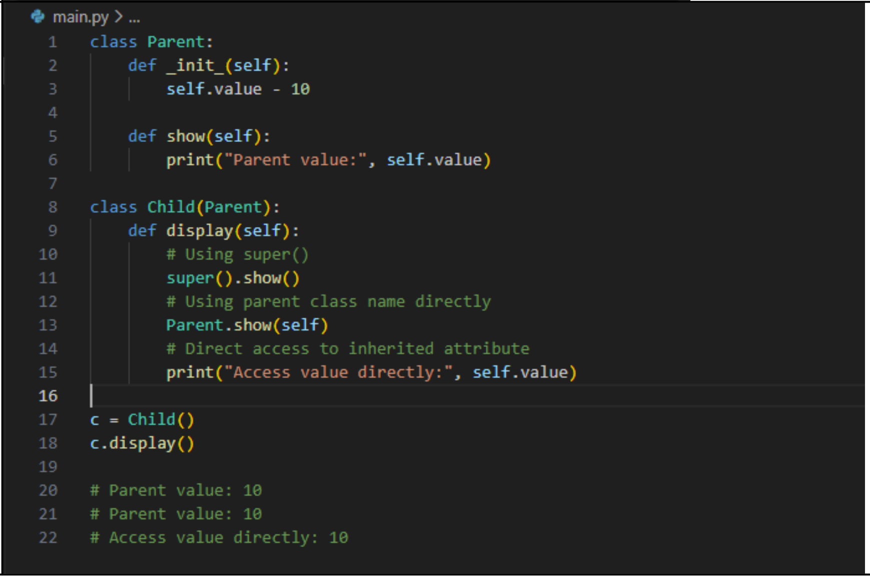
"You can call a parent class without an instance only if the method is a class method or static method. For normal methods, you need an object."



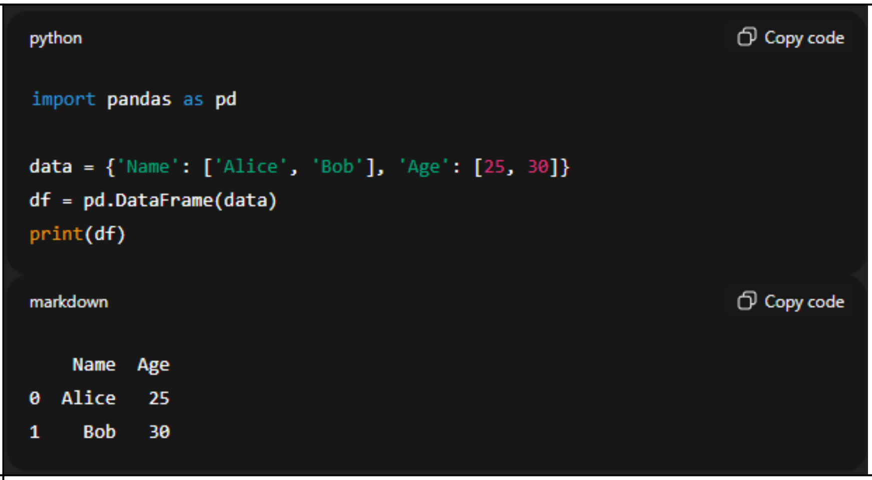
```python
class Parent:
    @staticmethod
    def greet():
        print("Hello from Parent")

# Call static method without creating instance
Parent.greet()  # Output: Hello from Parent
```

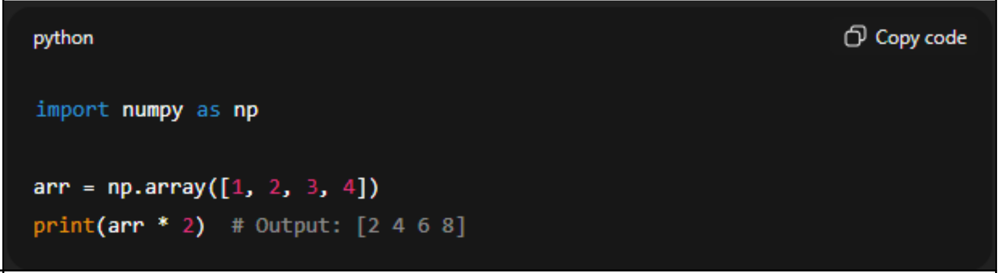
In what ways can parent members in a child class be accessed?

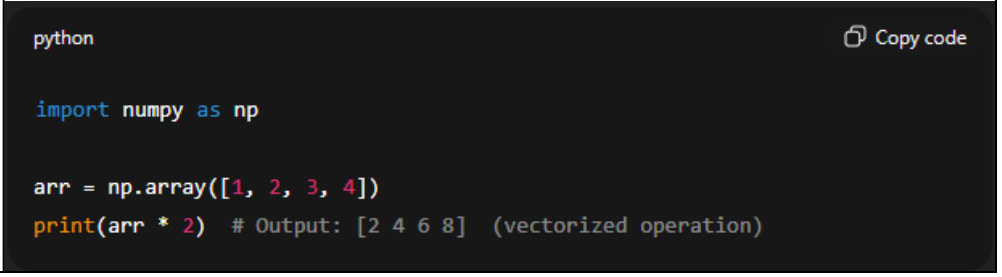
In Python, parent class members (attributes or methods) can be accessed in a child class in several ways:
1. Using super() – Calls parent methods or accesses parent attributes in a safe and flexible way.
2. Directly using the parent class name – Calls parent methods or accesses attributes explicitly.
3. Inherited automatically – If the member is public, the child class can use it directly.



```python
class Parent:
    def _init_(self):
        self.value - 10

    def show(self):
        print("Parent value:", self.value)

class Child(Parent):
    def display(self):
        # Using super()
        super().show()
        # Using parent class name directly
        Parent.show(self)
        # Direct access to inherited attribute
        print("Access value directly:", self.value)

c = Child()
c.display()

# Parent value: 10
# Parent value: 10
# Access value directly: 10
```

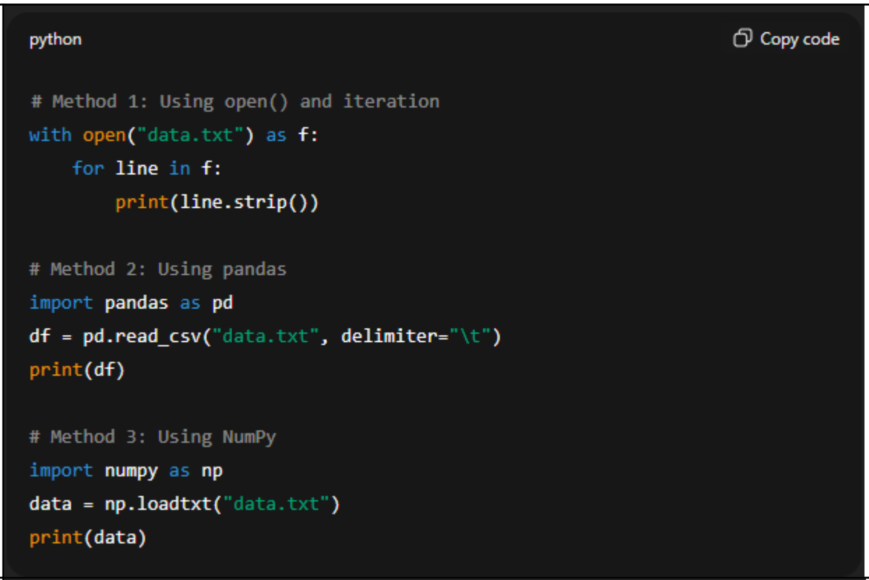
	What are Pandas in Python?
	
	Pandas is a Python library used for data manipulation and analysis. It provides data structures like DataFrame and Series that make it easy to handle tabular or time-series data, perform filtering, aggregation, and statistical operations efficiently.
	
	"Pandas is a Python library for data manipulation and analysis, using DataFrame and Series structures."
		
		python Copy code
		`import pandas as pd`
		
		`data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]}`
		`df = pd.DataFrame(data)`
		`print(df)`
		
		markdown Copy code
		`    Name  Age`
		`0  Alice   25`
		`1    Bob   30`

	What is a NumPy?
	
	NumPy (Numerical Python) is a Python library for numerical computing. It provides a high-performance multidimensional array object (ndarray) and functions to perform mathematical, logical, and statistical operations efficiently. It is widely used in scientific computing, data analysis, and machine learning.
	
	"NumPy is a Python library for numerical computing with high-performance arrays and mathematical functions."
		
		python Copy code
		`import numpy as np`
		
		`arr = np.array([1, 2, 3, 4])`
		`print(arr * 2)  # Output: [2 4 6 8]`

	Why is NumPy preferred over Python lists?
	
	NumPy is preferred over Python lists for several reasons:
	1. Performance: NumPy arrays are implemented in C and are much faster than Python lists for large-scale numerical computations.
	2. Memory efficiency: NumPy arrays use less memory because they store elements in a homogeneous and contiguous block.
	3. Vectorized operations: NumPy allows element-wise operations without explicit loops, making code cleaner and faster.
	4. Rich mathematical functions: NumPy provides a wide range of built-in mathematical, statistical, and linear algebra functions.
	
	"NumPy is faster, more memory-efficient, supports vectorized operations, and has rich mathematical functions compared to Python lists."
		
		python Copy code
		`import numpy as np`
		
		`arr = np.array([1, 2, 3, 4])`
		`print(arr * 2)  # Output: [2 4 6 8]  (vectorized operation)`

How can we efficiently load data from a text file?

In Python, data from a text file can be efficiently loaded depending on the file size and format:
1. Using with open() – Reads the file safely and automatically closes it.
2. Using readlines() or iterating line by line – Efficient for large files to avoid loading everything into memory.
3. Using pandas.read_csv() – For structured data (like CSV or tab-delimited), this is fast and convenient.
4. Using numpy.loadtxt() – For numerical data, it's efficient and returns NumPy arrays.



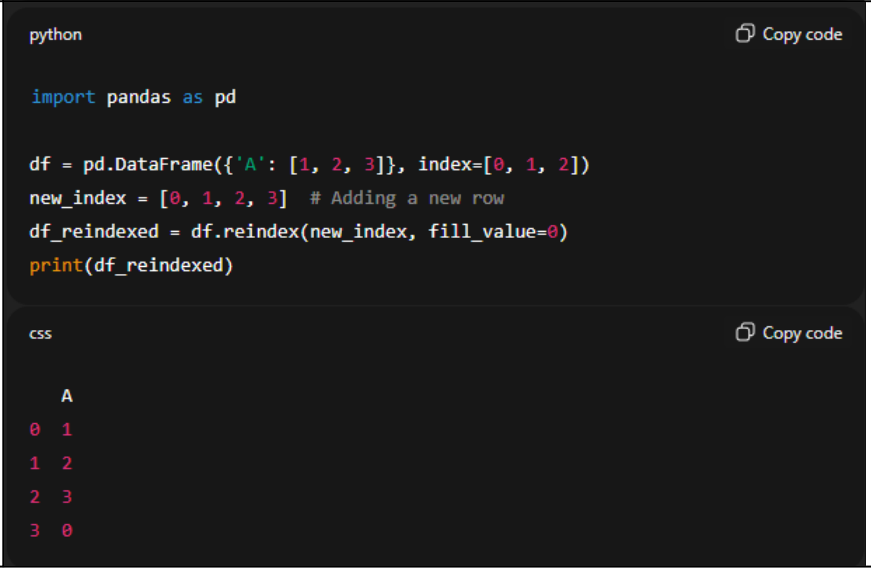
python Copy code

```python
# Method 1: Using open() and iteration
with open("data.txt") as f:
    for line in f:
        print(line.strip())

# Method 2: Using pandas
import pandas as pd
df = pd.read_csv("data.txt", delimiter="\t")
print(df)

# Method 3: Using NumPy
import numpy as np
data = np.loadtxt("data.txt")
print(data)
```

What is reindexing in Pandas?

In Pandas, reindexing means changing the row or column labels of a DataFrame or Series to a new set of labels. This can be used to add new rows/columns, remove some, or rearrange the order. When new labels are introduced, the missing values are filled with NaN by default or a specified value.

"Reindexing in Pandas changes the row or column labels, adding or rearranging them and filling missing values."

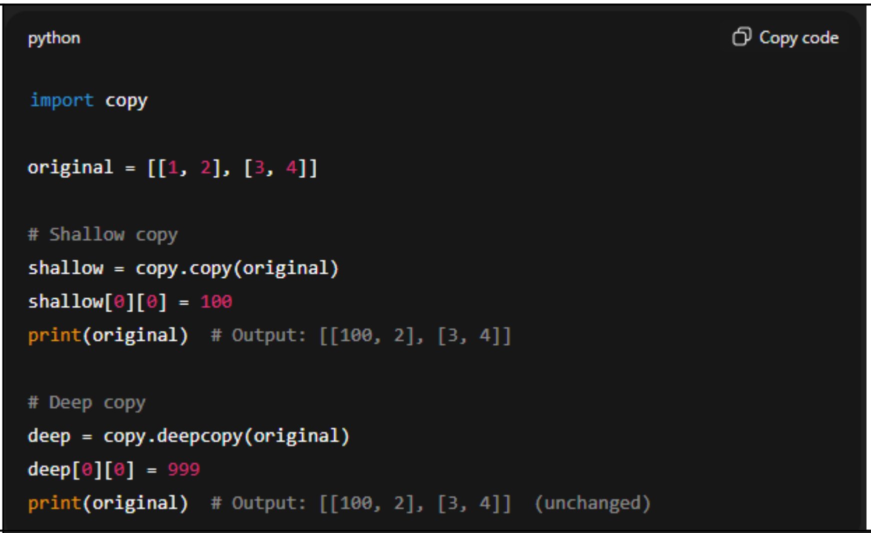


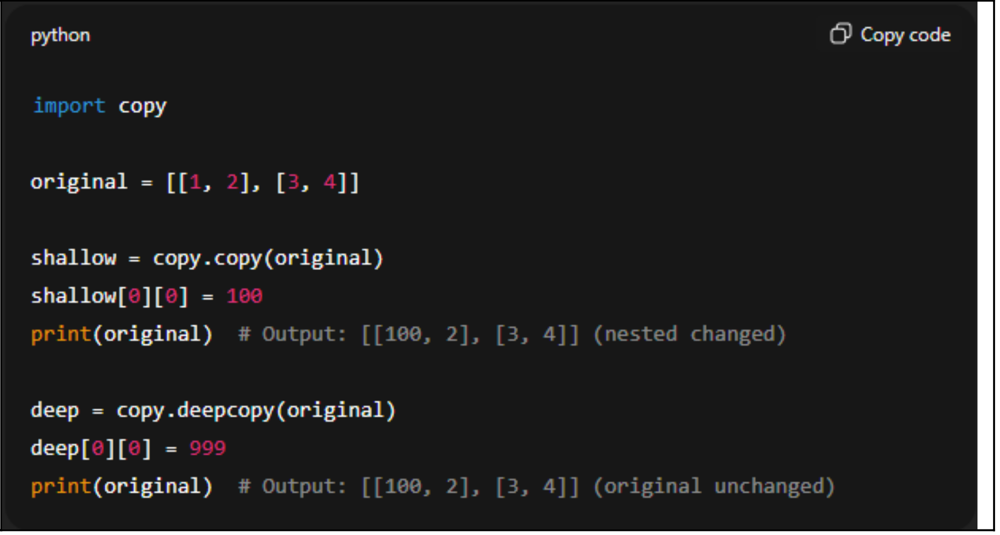
python Copy code

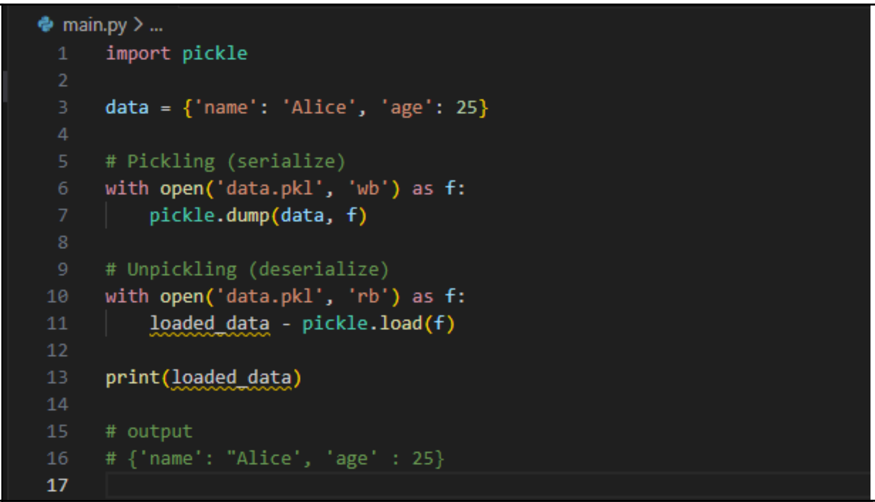
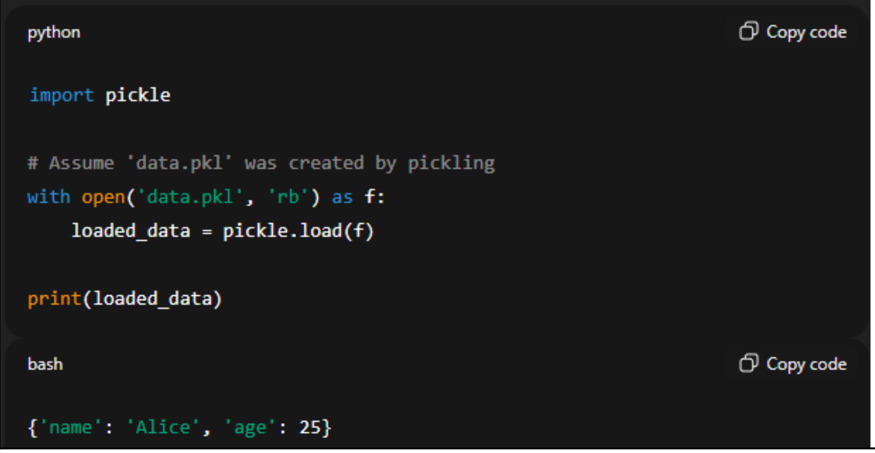
```python
import pandas as pd

df = pd.DataFrame({'A': [1, 2, 3]}, index=[0, 1, 2])
new_index = [0, 1, 2, 3]  # Adding a new row
df_reindexed = df.reindex(new_index, fill_value=0)
print(df_reindexed)
```

css Copy code

```
   A
0  1
1  2
2  3
3  0
```

	How can you copy an object in Python?
	
	In Python, you can copy an object in two main ways:
	1. Shallow copy – Creates a new object, but nested objects are shared. Use copy.copy().
	2. Deep copy – Creates a new object with all nested objects copied recursively. Use copy.deepcopy().
	
	"Use copy.copy() for shallow copy and copy.deepcopy() for deep copy of objects."
		python Copy code
		`import copy`
		
		`original = [[1, 2], [3, 4]]`
		
		`# Shallow copy`
		`shallow = copy.copy(original)`
		`shallow[0][0] = 100`
		`print(original)  # Output: [[100, 2], [3, 4]]`
		
		`# Deep copy`
		`deep = copy.deepcopy(original)`
		`deep[0][0] = 999`
		`print(original)  # Output: [[100, 2], [3, 4]]  (unchanged)`

	What is shallow and deep copying in Python?
	
	The difference between shallow copy and deep copy in Python is:
	1. Shallow copy:
	- Creates a new object, but the nested objects inside are shared with the original.
	- Changes to nested objects affect both the original and the copied object.
	- Use copy.copy().
	2. Deep copy:
	- Creates a completely independent copy, including all nested objects recursively.
	- Changes in the copied object do not affect the original object.
	- Use copy.deepcopy().
	
	"Shallow copy copies only the top-level object, sharing nested objects, while deep copy copies everything recursively, making an independent object."
		python Copy code
		`import copy`
		
		`original = [[1, 2], [3, 4]]`
		
		`shallow = copy.copy(original)`
		`shallow[0][0] = 100`
		`print(original)  # Output: [[100, 2], [3, 4]] (nested changed)`
		
		`deep = copy.deepcopy(original)`
		`deep[0][0] = 999`
		`print(original)  # Output: [[100, 2], [3, 4]] (original unchanged)`

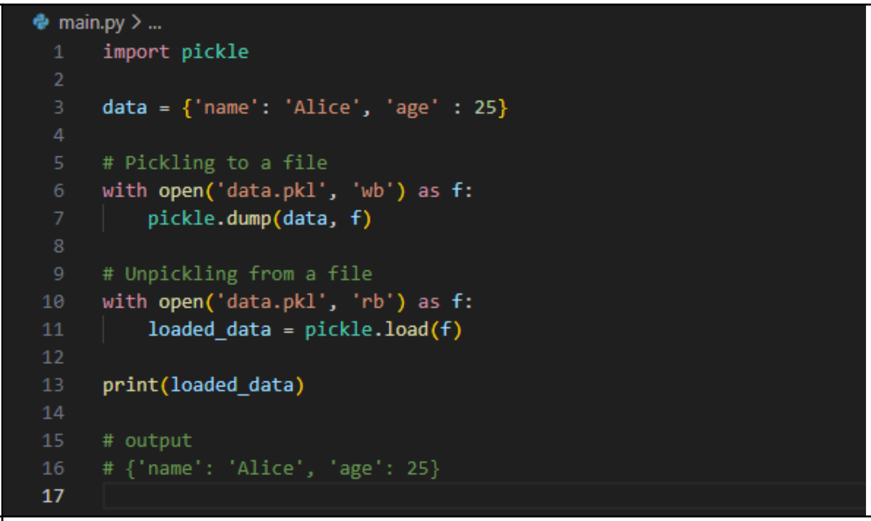
	What are Pickling? In Python, pickling is the process of converting a Python object into a byte stream so that it can be saved to a file or sent over a network. The reverse process, converting the byte stream back to a Python object, is called unpickling. Pickling is commonly used for persisting Python objects. "Pickling is serializing a Python object to a byte stream for storage or transmission; unpickling restores it."		
	How can you define Unpickling in Python? In Python, unpickling is the process of converting a pickled (serialized) byte stream back into the original Python object. It is the reverse of pickling and is used to load objects that were previously saved or transmitted. "Unpickling converts a pickled byte stream back into the original Python object."		

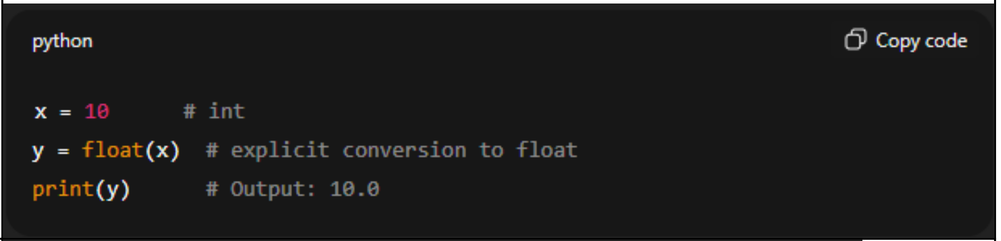
main.py > ...

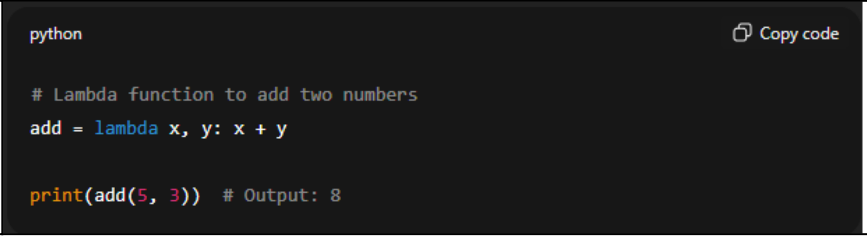
```python
import pickle

data = {'name': 'Alice', 'age': 25}

# Pickling (serialize)
with open('data.pkl', 'wb') as f:
    pickle.dump(data, f)

# Unpickling (deserialize)
with open('data.pkl', 'rb') as f:
    loaded_data - pickle.load(f)

print(loaded_data)

# output
# {'name': "Alice", 'age' : 25}
```

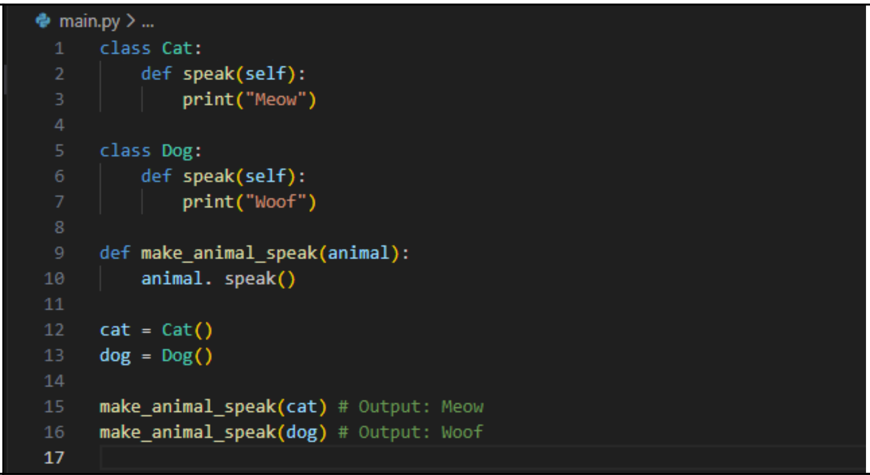
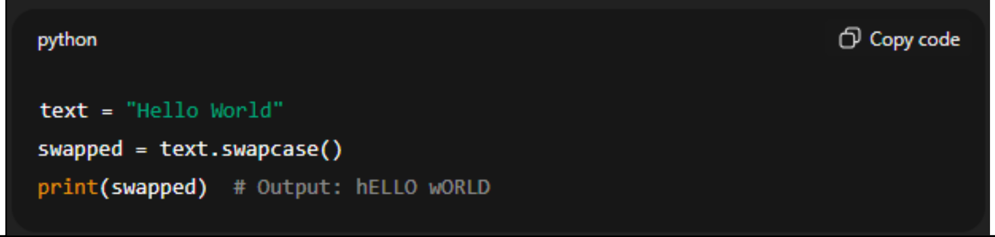
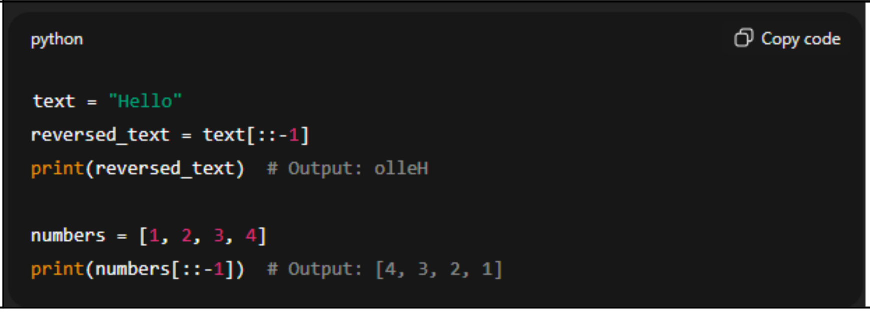
```python
import pickle

# Assume 'data.pkl' was created by pickling
with open('data.pkl', 'rb') as f:
    loaded_data = pickle.load(f)

print(loaded_data)
```

```bash
{'name': 'Alice', 'age': 25}
```

	What function is used for Pickling and Unpickling?
	
	In Python, the pickle module provides functions for pickling and unpickling:
	1. Pickling (serialization): Use pickle.dump() to write an object to a file or pickle.dumps() to get a byte stream.
	2. Unpickling (deserialization): Use pickle.load() to read an object from a file or pickle.loads() to load from a byte stream.
		 main.py > ...
		`import pickle`
		
		`data = {'name': 'Alice', 'age' : 25}`
		
		`# Pickling to a file`
		`with open('data.pkl', 'wb') as f:`
		`    pickle.dump(data, f)`
		
		`# Unpickling from a file`
		`with open('data.pkl', 'rb') as f:`
		`    loaded_data = pickle.load(f)`
		
		`print(loaded_data)`
		
		`# output`
		`# {'name': 'Alice', 'age': 25}`

	What are some types of Type Conversion in Python?
	
	In Python, type conversion is changing a variable from one data type to another. There are two main types:
	1. Implicit Type Conversion (Type Casting): Python automatically converts one type to another. Example: converting int to float.
	2. Explicit Type Conversion: You manually convert types using built-in functions like:
	- int() – converts to integer
	- float() – converts to float
	- str() – converts to string
	- list() – converts to list
	- tuple() – converts to tuple
	- dict() – converts to dictionary
		 python Copy code
		
		`x = 10       # int`
		`y = float(x)  # explicit conversion to float`
		`print(y)      # Output: 10.0`

	Give an example of a lambda function.
	
	A lambda function in Python is an anonymous, single-line function defined using the lambda keyword.
	
	"A lambda function is an anonymous single-line function defined with lambda."
		 python Copy code
		
		`# Lambda function to add two numbers`
		`add = lambda x, y: x + y`
		
		`print(add(5, 3))  # Output: 8`

	In Python, what is Polymorphism? In Python, polymorphism is the ability of different objects to be accessed through the same interface, allowing them to perform actions in different ways depending on their type. It is commonly seen in method overriding, operator overloading, and duck typing. "Polymorphism allows different objects to be accessed through the same interface, performing actions in their own way."	
	How would you define the Swapcase () in Python? In Python, the swapcase() method is a string method that converts all uppercase letters to lowercase and all lowercase letters to uppercase in a string. It does not change non-alphabetic characters. "swapcase() converts uppercase letters to lowercase and lowercase letters to uppercase in a string."	
Advanced	What does [::-1] do in Python and give an example? In Python, the [::-1] slice notation is used to reverse a sequence such as a string, list, or tuple. It means: start from the end towards the first element with a step of -1. "[::-1] reverses a sequence in Python, like a string or list."	

main.py > ...

```python
class Cat:
    def speak(self):
        print("Meow")

class Dog:
    def speak(self):
        print("Woof")

def make_animal_speak(animal):
    animal. speak()

cat = Cat()
dog = Dog()

make_animal_speak(cat) # Output: Meow
make_animal_speak(dog) # Output: Woof

```

python

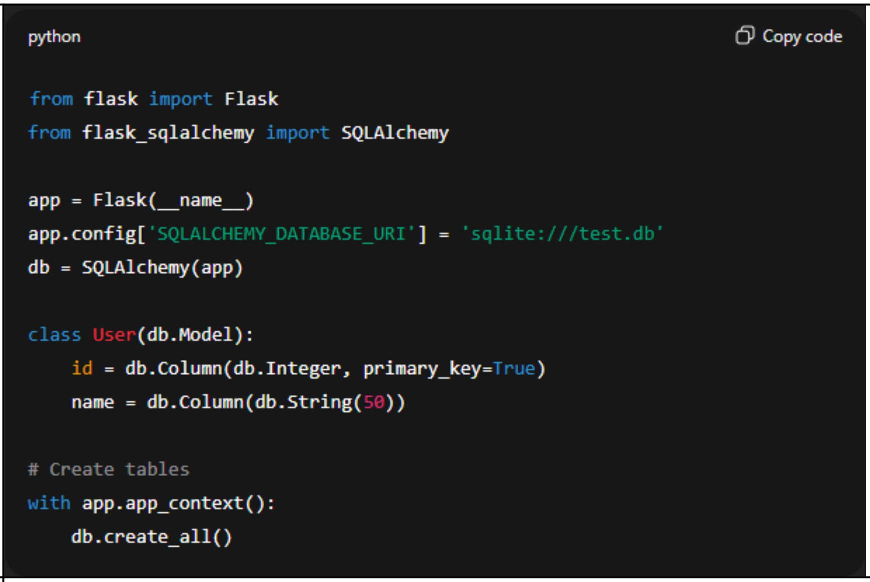
Copy code

```python
text = "Hello World"
swapped = text.swapcase()
print(swapped)  # Output: hELLO wORLD
```

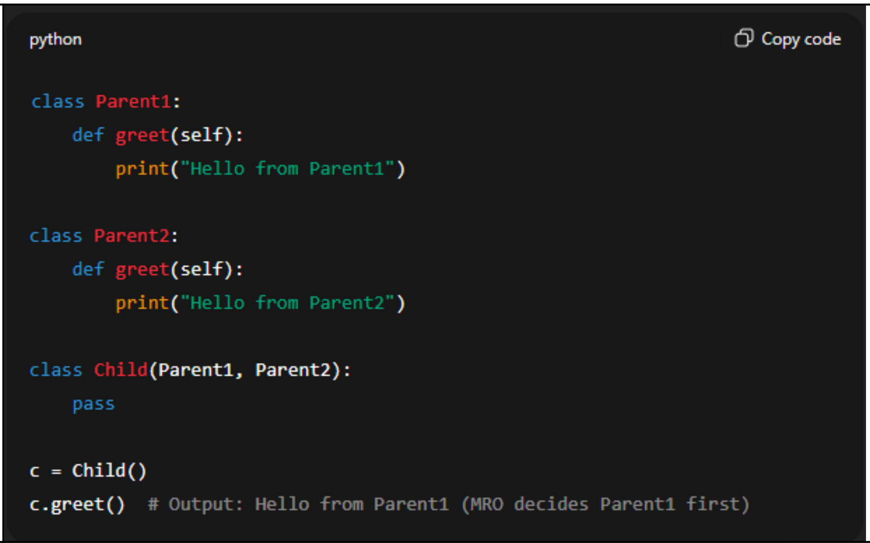
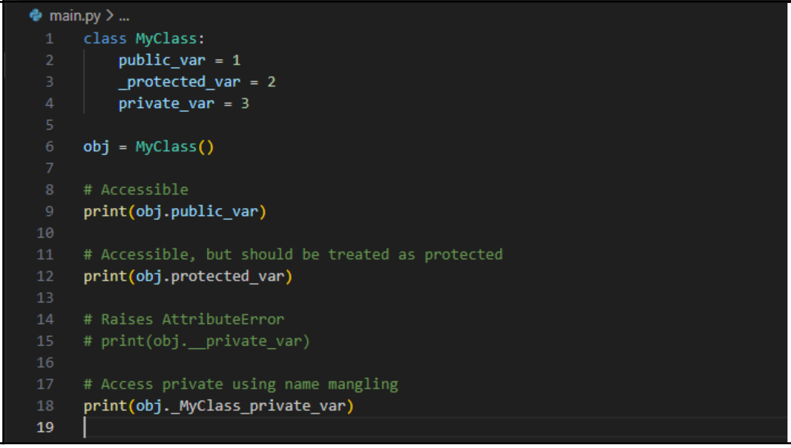
python

Copy code

```python
text = "Hello"
reversed_text = text[::-1]
print(reversed_text)  # Output: olleH

numbers = [1, 2, 3, 4]
print(numbers[::-1])  # Output: [4, 3, 2, 1]
```

	Explain database connection in Python Flask. In Flask, connecting to a database involves using a database driver or ORM (Object Relational Mapper). Common approaches include: 1. Using Flask extensions like Flask-SQLAlchemy: - Provides ORM support and simplifies database operations. - You define the database URI, create a SQLAlchemy object, and interact with the database through models. 2. Using direct database drivers (like sqlite3 or pymysql): - You manually open a connection, create a cursor, execute queries, and close the connection.	
	What is the Dogpile effect? The Dogpile effect occurs when multiple processes or threads try to regenerate the same cached data simultaneously after it expires. This can overload the database or backend system because all requests trigger the same expensive computation at once. Prevention: - Use cache locks or stale-while-revalidate techniques to ensure only one process regenerates the cache while others wait. Example (conceptual): Imagine a web cache for a popular product price. When the cache expires, hundreds of users request the price at the same time, causing many queries to the database simultaneously — this is the Dogpile effect. *"Dogpile effect happens when multiple requests try to regenerate expired cache simultaneously, causing overload."*	

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))

# Create tables
with app.app_context():
    db.create_all()
```

	Are multiple inheritances supported in Python? Yes, Python supports multiple inheritance, which means a class can inherit attributes and methods from more than one parent class. Python uses the Method Resolution Order (MRO) to determine the order in which base classes are searched when calling a method. "Yes, Python supports multiple inheritance and uses MRO to determine method lookup order."	
	Does Python make use of access specifiers? Python does not have strict access specifiers like public, private, or protected found in languages such as Java or C++. Instead, it relies on naming conventions: 1. Public: Any attribute or method without underscores is public and accessible anywhere. 2. Protected: Prefix a single underscore _ (e.g., _var) to indicate it should be treated as protected, but it's only a convention. 3. Private: Prefix a double underscore __ (e.g., __var) triggers name mangling, making it harder (but not impossible) to access from outside the class. "Python uses naming conventions instead of strict access specifiers: public, protected (_), and private (__ with name mangling)."	

python Copy code

```python
class Parent1:
    def greet(self):
        print("Hello from Parent1")

class Parent2:
    def greet(self):
        print("Hello from Parent2")

class Child(Parent1, Parent2):
    pass

c = Child()
c.greet()  # Output: Hello from Parent1 (MRO decides Parent1 first)
```

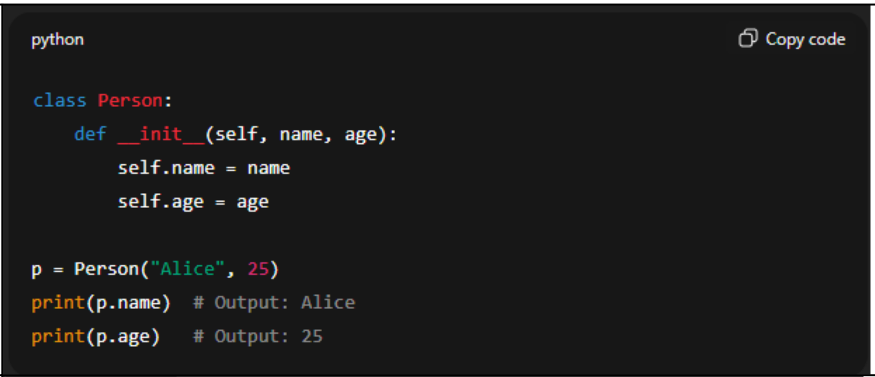
main.py > ...

```python
class MyClass:
    public_var = 1
    _protected_var = 2
    private_var = 3

obj = MyClass()

# Accessible
print(obj.public_var)

# Accessible, but should be treated as protected
print(obj.protected_var)

# Raises AttributeError
# print(obj.__private_var)

# Access private using name mangling
print(obj._MyClass_private_var)
```

How do you create a constructor in Python?

In Python, a constructor is a special method called __init__() that is automatically invoked when an object of a class is created. It is used to initialize the attributes of the object.

"A constructor in Python is the __init__() method used to initialize object attributes."

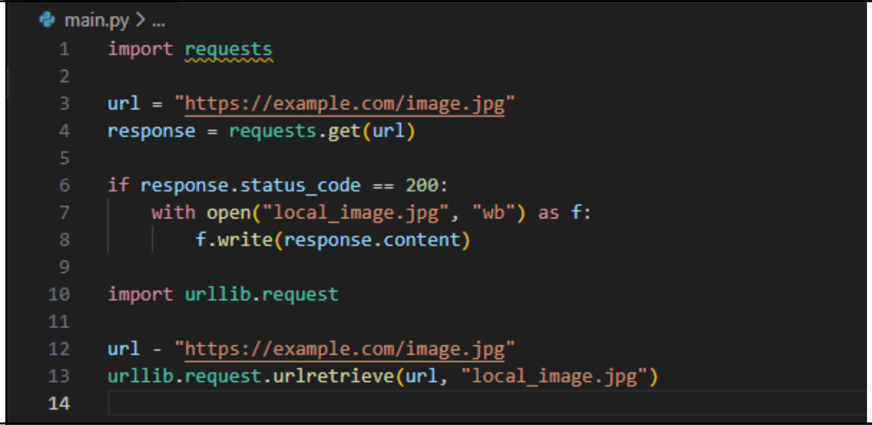


```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Alice", 25)
print(p.name)  # Output: Alice
print(p.age)   # Output: 25
```

How to save an image in Python locally when we know the URL address?

In Python, you can save an image from a URL using the requests library or urllib. The idea is to download the image content and write it to a local file in binary mode.

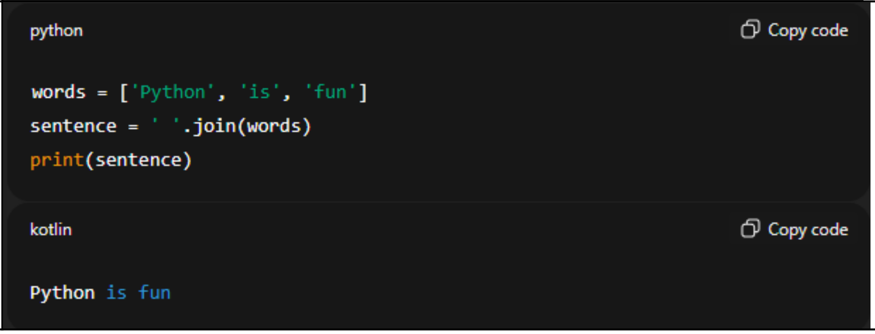
"Use requests.get() or urllib.request.urlretrieve() to download an image from a URL and save it locally."



```python
import requests

url = "https://example.com/image.jpg"
response = requests.get(url)

if response.status_code == 200:
    with open("local_image.jpg", "wb") as f:
        f.write(response.content)

import urllib.request

url - "https://example.com/image.jpg"
urllib.request.urlretrieve(url, "local_image.jpg")

```

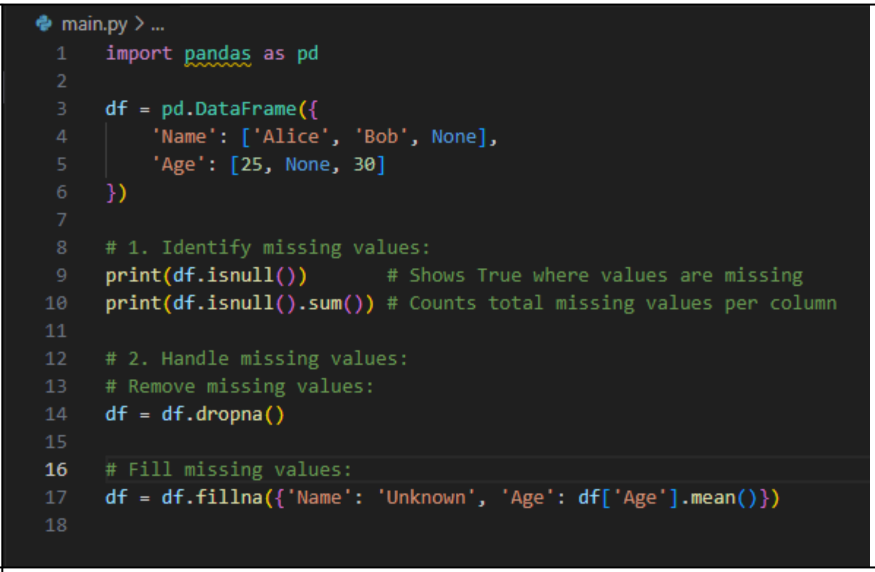
Explain how the join () function in Python?

The join() function in Python is a string method used to combine (concatenate) elements of an iterable (like a list or tuple) into a single string, with a specified separator between each element.

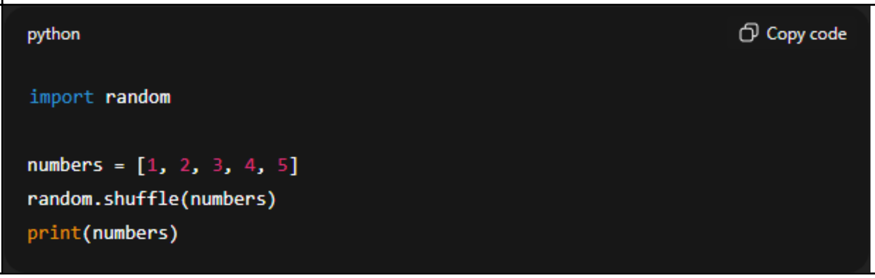


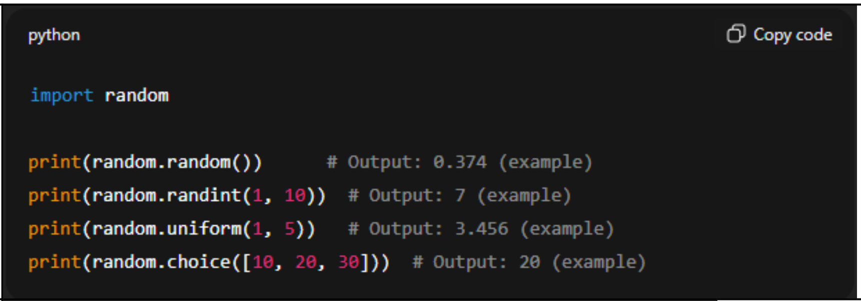
```python
words = ['Python', 'is', 'fun']
sentence = ' '.join(words)
print(sentence)
```

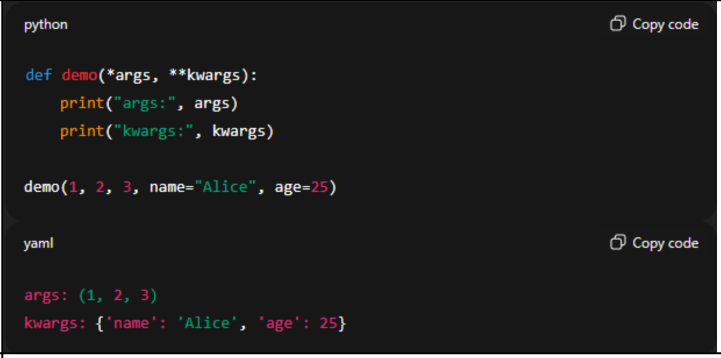
```kotlin
Python is fun
```

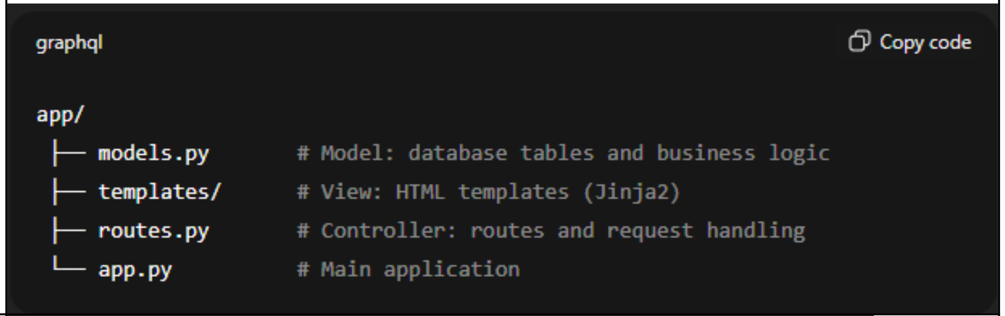
	How can you identify and deal with missing values in a dataframe?
	
	In Python’s Pandas, missing values are usually represented as NaN (Not a Number). You can identify and handle them using built-in functions.
		main.py > ...
		import pandas as pd
		
		df = pd.DataFrame({
		    'Name': ['Alice', 'Bob', None],
		    'Age': [25, None, 30]
		})
		
		# 1. Identify missing values:
		print(df.isnull()) # Shows True where values are missing
		print(df.isnull().sum()) # Counts total missing values per column
		
		# 2. Handle missing values:
		# Remove missing values:
		df = df.dropna()
		
		# Fill missing values:
		df = df.fillna({'Name': 'Unknown', 'Age': df['Age'].mean()})	

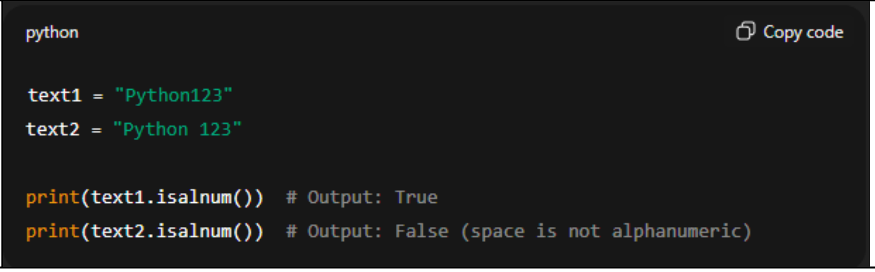
	What is the use of manage.py in Python? In Django (a popular Python web framework), the manage.py file is an auto-generated command-line utility that helps you interact with your Django project. It allows you to perform various administrative tasks, such as: - Running the development server (python manage.py runserver) - Applying database migrations (python manage.py migrate) - Creating new apps (python manage.py startapp appname) - Creating superusers (python manage.py createsuperuser) - Running tests (python manage.py test) "manage.py acts as a wrapper around Django’s django-admin tool and helps manage your project more easily."		

	Explain the shuffle method and give an example.
	
	The shuffle() method in Python is part of the random module. It is used to randomly rearrange (shuffle) the elements of a list in place — meaning it modifies the original list instead of creating a new one.
		python Copy code
		
		import random
		
		numbers = [1, 2, 3, 4, 5]
		random.shuffle(numbers)
		print(numbers)	

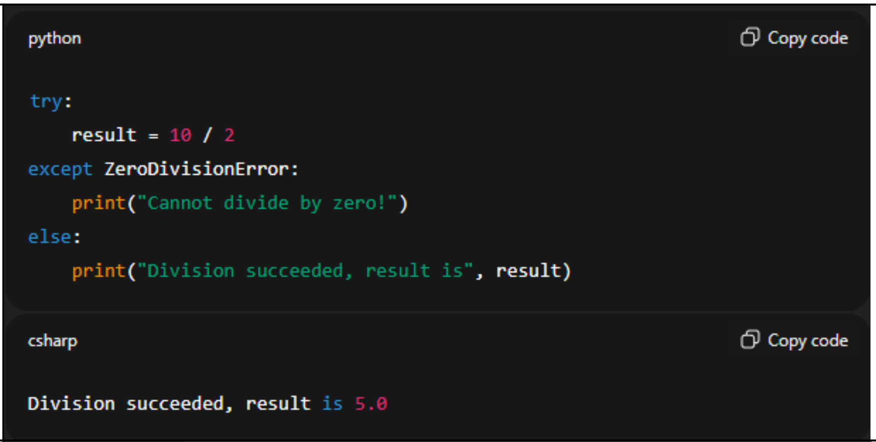
	What method can be used to generate random numbers in Python?
	
	In Python, you can generate random numbers using functions from the random module. Some commonly used methods are:
	1. random.random() – Returns a random float between 0.0 and 1.0.
	2. random.randint(a, b) – Returns a random integer between a and b (inclusive).
	3. random.uniform(a, b) – Returns a random float between a and b.
	4. random.choice(sequence) – Returns a random element from a sequence (like list or tuple).
		
		python Copy code
		`import random`
		
		`print(random.random())      # Output: 0.374 (example)`
		`print(random.randint(1, 10))  # Output: 7 (example)`
		`print(random.uniform(1, 5))   # Output: 3.456 (example)`
		`print(random.choice([10, 20, 30]))  # Output: 20 (example)`

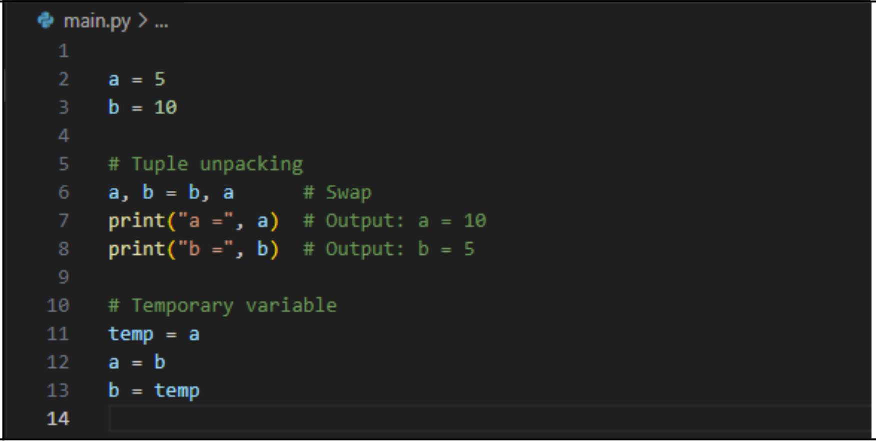
	What does *args, ** kwargs mean in Python?
	
	In Python, *args and **kwargs are used in function definitions to allow variable numbers of arguments.
	1. *args (Non-keyword arguments):
	- Collects extra positional arguments passed to a function into a tuple.
	- Useful when you don't know in advance how many positional arguments will be provided.
	
	2. **kwargs (Keyword arguments):
	- Collects extra keyword arguments passed to a function into a dictionary.
	- Useful when you want to handle named arguments dynamically.
		
		python Copy code
		`def demo(*args, **kwargs):`
		`    print("args:", args)`
		`    print("kwargs:", kwargs)`
		
		`demo(1, 2, 3, name="Alice", age=25)`
		
		yaml Copy code
		`args: (1, 2, 3)`
		`kwargs: {'name': 'Alice', 'age': 25}`

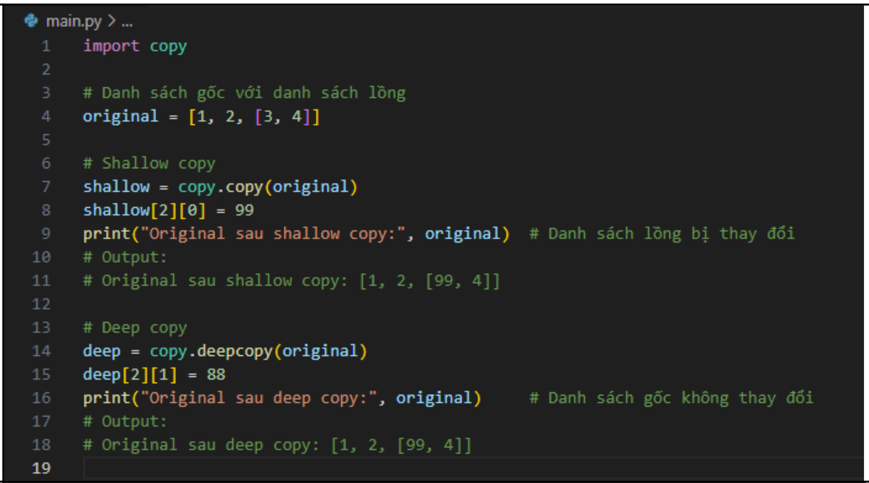
	Is Flask an MVC model? If true, justify this using the MVC pattern.
	
	Flask can be used to implement the MVC (Model-View-Controller) pattern, but it is not strictly enforced. Flask is a micro-framework, so it gives developers flexibility to structure their application in an MVC way.
	- Model: Handles data and business logic. In Flask, this is usually done using ORMs like SQLAlchemy or by directly interacting with databases.
	- View: Represents the user interface, typically HTML templates rendered with Jinja2 in Flask.
	- Controller: Handles application logic and routing, implemented via Flask route functions that take requests, process data (using models), and return rendered views.
		
		graphql Copy code
		`app/`
		`├── models.py      # Model: database tables and business logic`
		`├── templates/     # View: HTML templates (Jinja2)`
		`├── routes.py      # Controller: routes and request handling`
		`└── app.py         # Main application`

	How can we check if all characters in a string are alphanumeric?
	
	In Python, you can use the isalnum() string method to check if all characters in a string are alphanumeric (letters and numbers only, no spaces or special characters). It returns True if all characters are alphanumeric, otherwise False.
	
	"Use the isalnum() method; it returns True if all characters in a string are letters or digits.
		
		python Copy code
		`text1 = "Python123"`
		`text2 = "Python 123"`
		
		`print(text1.isalnum())  # Output: True`
		`print(text2.isalnum())  # Output: False (space is not alphanumeric)`

	What are some of the most used built-in modules in Python? Python comes with many built-in modules that provide useful functionality without requiring installation. Some of the most commonly used built-in modules are: 1. os – Interact with the operating system (files, directories, environment variables). 2. sys – Access system-specific parameters and functions. 3. math – Perform mathematical operations (trigonometry, logarithms, etc.). 4. random – Generate random numbers and perform random operations. 5. datetime – Work with dates and times. 6. json – Encode and decode JSON data. 7. re – Perform regular expression operations. 8. collections – Specialized container datatypes like Counter, deque, OrderedDict. "Common built-in modules include os, sys, math, random, datetime, json, re, and collections."	
	What are the tools for debugging and performing static analysis in Python? In Python, there are several tools for debugging and static analysis: 1. Debugging tools: - pdb – The built-in Python debugger, allows stepping through code, setting breakpoints, and inspecting variables. - ipdb – An enhanced version of pdb with IPython integration. - IDE debuggers – Most IDEs like PyCharm, VS Code, and Spyder provide integrated debugging with breakpoints, watches, and call stack inspection. 2. Static analysis tools: - pylint – Checks for coding standards, errors, and potential bugs. - flake8 – Linter for style guide enforcement (PEP 8) and code errors. - mypy – Performs type checking based on type hints. - pyflakes – Detects errors without performing style checks. *"For debugging use pdb, ipdb, or IDE debuggers; for static analysis use pylint, flake8, mypy, or pyflakes."*	

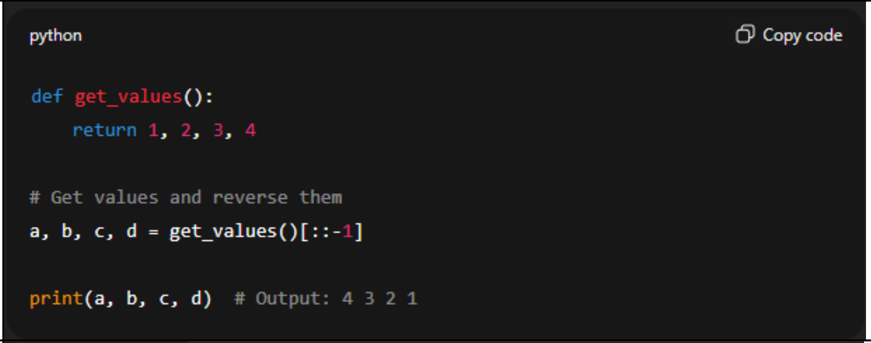
	When will the else part of try-except-else be executed?
	
	"The else part runs only when the try block finishes with no errors."
		python Copy code
		`try:`
		`    result = 10 / 2`
		`except ZeroDivisionError:`
		`    print("Cannot divide by zero!")`
		`else:`
		`    print("Division succeeded, result is", result)`
		
		csharp Copy code
		`Division succeeded, result is 5.0`	

	Write a code to swap two numbers in Python.
	
	In Python, you can swap two numbers in a very simple way using tuple unpacking, without needing a temporary variable.
	
	"You can swap two numbers using a, b = b, a or with a temporary variable."
		main.py > ...
		`a = 5`
		`b = 10`
		
		`# Tuple unpacking`
		`a, b = b, a       # Swap`
		`print("a =", a)  # Output: a = 10`
		`print("b =", b)  # Output: b = 5`
		
		`# Temporary variable`
		`temp = a`
		`a = b`
		`b = temp`	

	Show code examples of deep and shallow copying. In Python, shallow copy and deep copy are used to copy objects, but they behave differently: - Shallow copy: Copies the object itself, but nested objects are still references to the original. - Deep copy: Copies the object and all nested objects, creating a completely independent copy. "Shallow copy copies the outer object but keeps references to nested objects, while deep copy duplicates everything including nested objects."		

main.py > ...

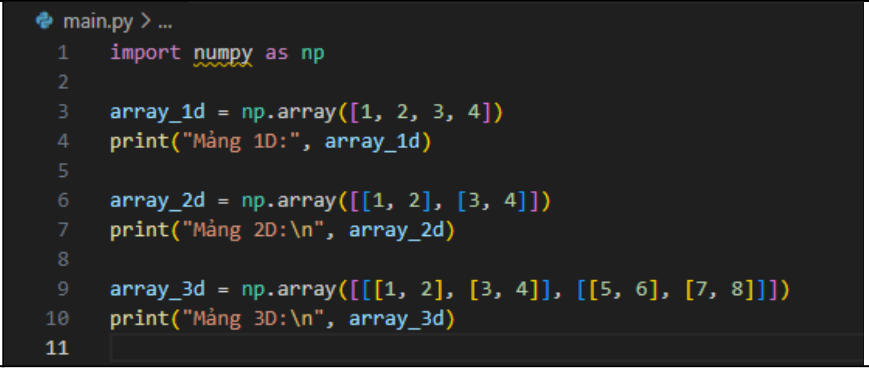
```python
import copy

# Danh sách gốc với danh sách lồng
original = [1, 2, [3, 4]]

# Shallow copy
shallow = copy.copy(original)
shallow[2][0] = 99
print("Original sau shallow copy:", original)  # Danh sách lồng bị thay đổi
# Output:
# Original sau shallow copy: [1, 2, [99, 4]]

# Deep copy
deep = copy.deepcopy(original)
deep[2][1] = 88
print("Original sau deep copy:", original)     # Danh sách gốc không thay đổi
# Output:
# Original sau deep copy: [1, 2, [99, 4]]

```

	Write a code to reverse multiple values from functions. In Python, a function can return multiple values as a tuple, and you can reverse them using tuple unpacking or the [::-1] slicing. "Return multiple values as a tuple and use slicing [::-1] to reverse them."		

python

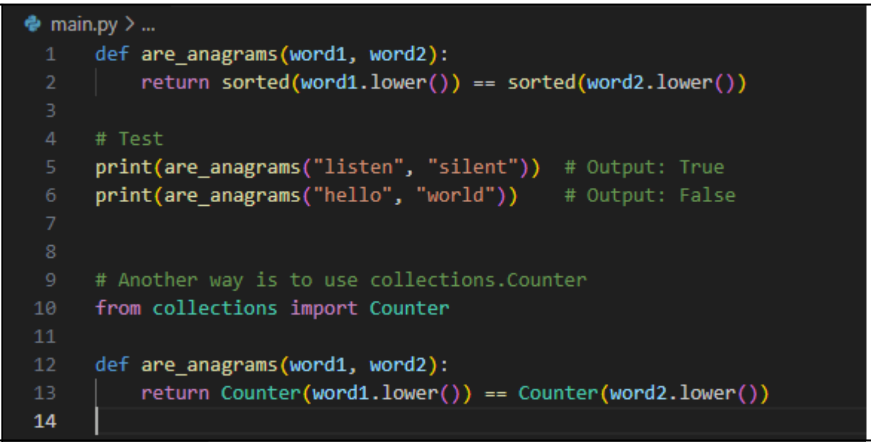
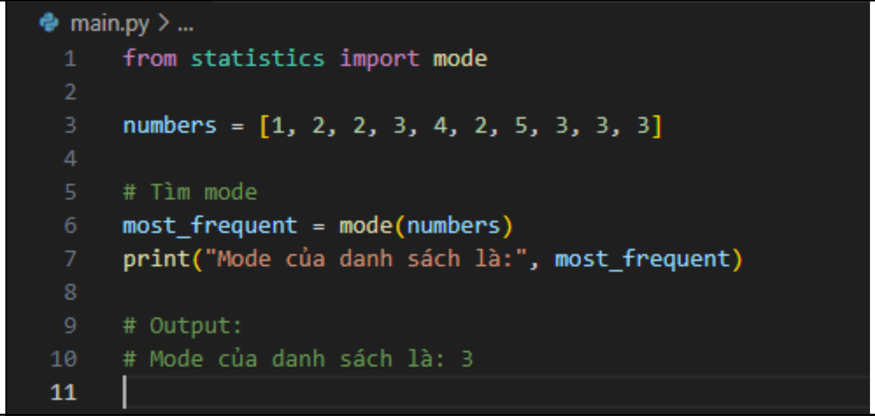
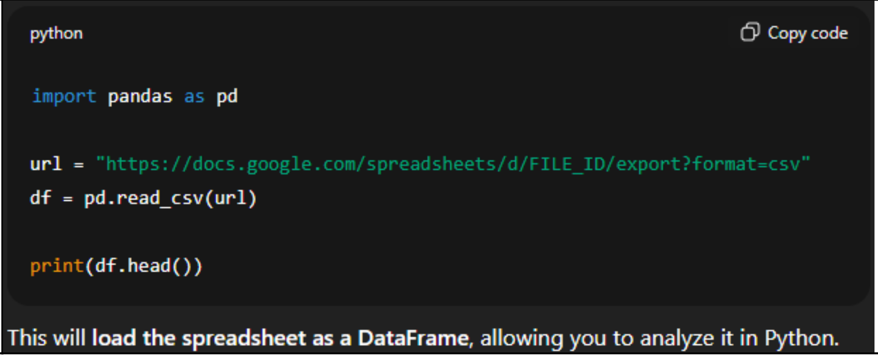
Copy code

```python
def get_values():
    return 1, 2, 3, 4

# Get values and reverse them
a, b, c, d = get_values()[::-1]

print(a, b, c, d)  # Output: 4 3 2 1
```

	What are the steps to create 3D, 2D, and 1D arrays? In Python, you can create 1D, 2D, and 3D arrays using the NumPy library.		

main.py > ...

```python
import numpy as np

array_1d = np.array([1, 2, 3, 4])
print("Mảng 1D:", array_1d)

array_2d = np.array([[1, 2], [3, 4]])
print("Mảng 2D:\n", array_2d)

array_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("Mảng 3D:\n", array_3d)

```

	Write a code to check whether two words are anagrams or not. In Python, two words are anagrams if they contain the same letters with the same frequency, regardless of order. One simple way is to sort the letters and compare.	
	Write a code to test for the mode in a list of numbers. In Python, the mode of a list is the number that appears most frequently. You can use the statistics module to find it. Note: If multiple numbers have the same highest frequency, statistics. mode() will return the first one encountered.	
	Show how to access the dataset of a publicly shared spreadsheet in the format of CSV stored in Google Drive. To access a publicly shared Google Spreadsheet in CSV format, you can use Python's pandas library with the file's CSV export link. Steps: 1. Make sure the spreadsheet is public. 2. Get the CSV export link. For a Google Sheet with URL like: https://docs.google.com/spreadsheets/d/FILE_ID/edit#gid=0 The CSV link will be: https://docs.google.com/spreadsheets/d/FILE_ID/export?format=csv 3. Use pandas.read_csv() to load it:	

main.py > ...

```python
def are_anagrams(word1, word2):
    return sorted(word1.lower()) == sorted(word2.lower())

# Test
print(are_anagrams("listen", "silent"))  # Output: True
print(are_anagrams("hello", "world"))    # Output: False


# Another way is to use collections.Counter
from collections import Counter

def are_anagrams(word1, word2):
    return Counter(word1.lower()) == Counter(word2.lower())

```

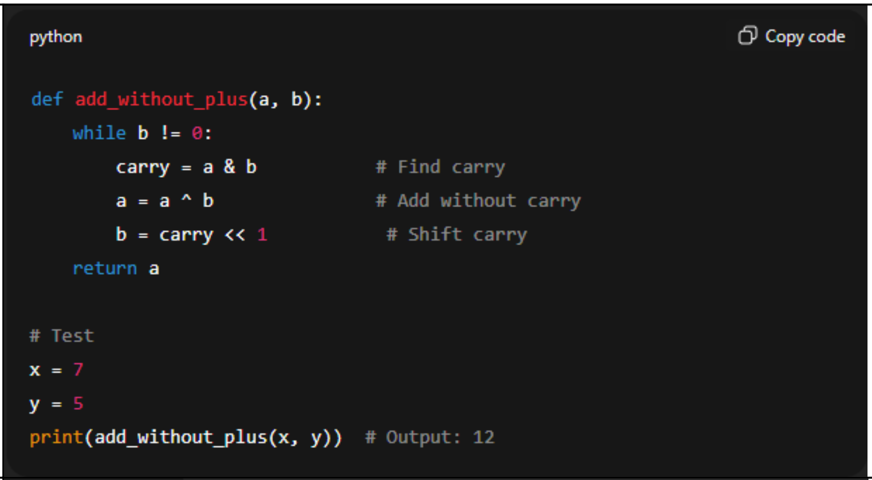
main.py > ...

```python
from statistics import mode

numbers = [1, 2, 2, 3, 4, 2, 5, 3, 3, 3]

# Tìm mode
most_frequent = mode(numbers)
print("Mode của danh sách là:", most_frequent)

# Output:
# Mode của danh sách là: 3

```

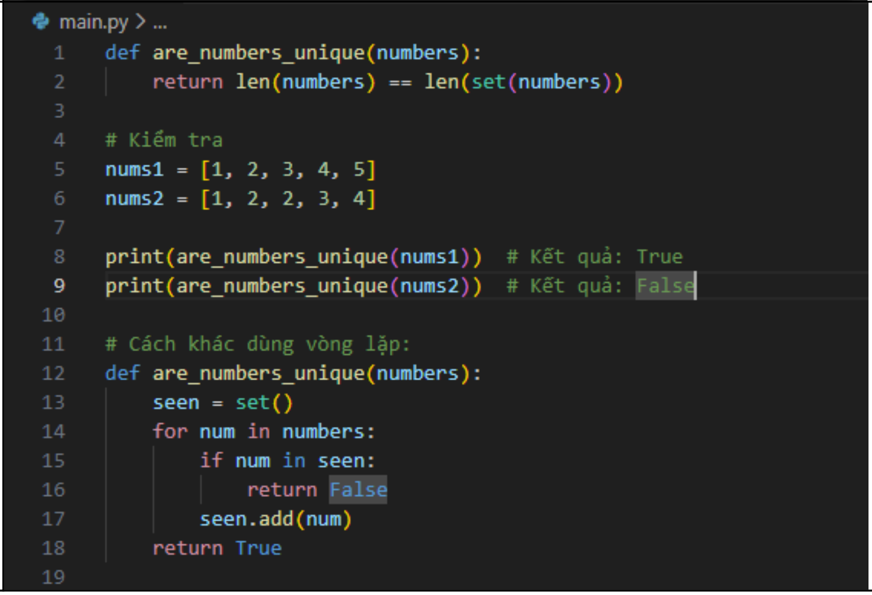
python Copy code

```python
import pandas as pd

url = "https://docs.google.com/spreadsheets/d/FILE_ID/export?format=csv"
df = pd.read_csv(url)

print(df.head())
```

This will **load the spreadsheet as a DataFrame**, allowing you to analyze it in Python.

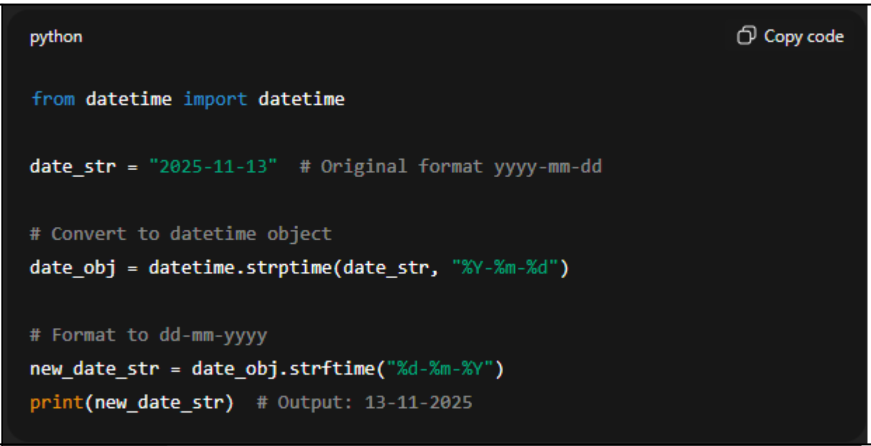
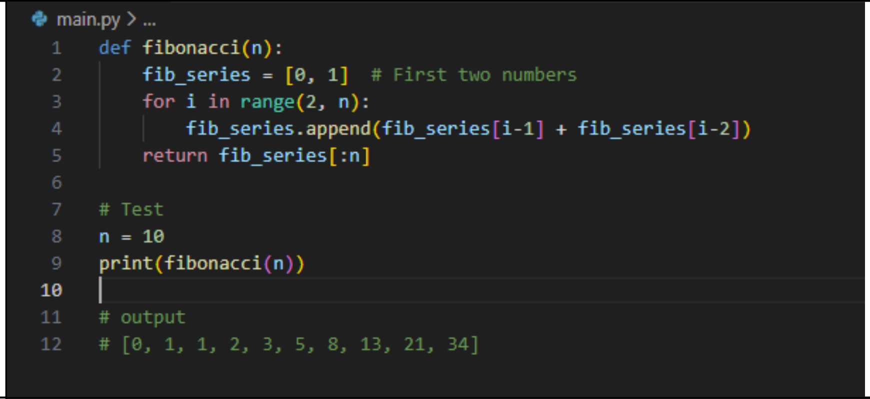
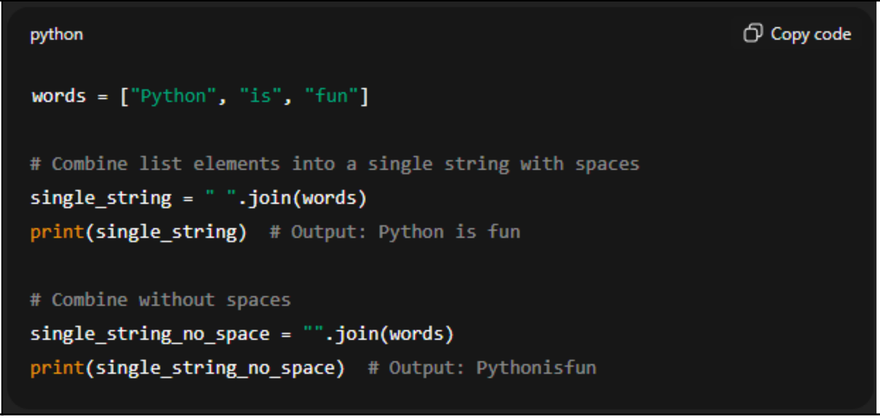
	Write a code to add two integers without using the '+' operator when the numbers are greater than zero. You can add two positive integers without using the + operator by using bitwise operations. The idea is to use XOR (^) for addition without carry and AND (&) shifted left for the carry, repeating until there is no carry left. "Use bitwise XOR for addition and AND shifted left for carry, loop until carry is zero."		

```python
def add_without_plus(a, b):
    while b != 0:
        carry = a & b           # Find carry
        a = a ^ b               # Add without carry
        b = carry << 1           # Shift carry
    return a


# Test
x = 7
y = 5
print(add_without_plus(x, y))  # Output: 12
```

	Write a code that is given a sequence of numbers and it checks if the numbers are unique. In Python, you can check if numbers in a sequence are unique by converting the sequence to a set and comparing its length with the original sequence. "Convert the sequence to a set and compare lengths, or check each number with a set to ensure uniqueness."		

main.py > ...

```python
def are_numbers_unique(numbers):
    return len(numbers) == len(set(numbers))

# Kiểm tra
nums1 = [1, 2, 3, 4, 5]
nums2 = [1, 2, 2, 3, 4]

print(are_numbers_unique(nums1))  # Kết quả: True
print(are_numbers_unique(nums2))  # Kết quả: False

# Cách khác dùng vòng lặp:
def are_numbers_unique(numbers):
    seen = set()
    for num in numbers:
        if num in seen:
            return False
        seen.add(num)
    return True

```

	Write a program to convert the date format from yyyy-mm-dd to dd-mm-yyyy. In Python, you can convert a date string from yyyy-mm-dd to dd-mm-yyyy using the datetime module`. "Use datetime.strptime() to parse yyyy-mm-dd, then strftime() to format dd-mm-yyyy."	
	Write a program in Python to create a Fibonacci series. In Python, you can create a Fibonacci series using a loop or recursion. "Start with [0,1] and use a loop to append the sum of the last two numbers to build the Fibonacci series."	
	Write a code to create a single string from elements in a list. In Python, you can create a single string from a list of elements using the join() method.	

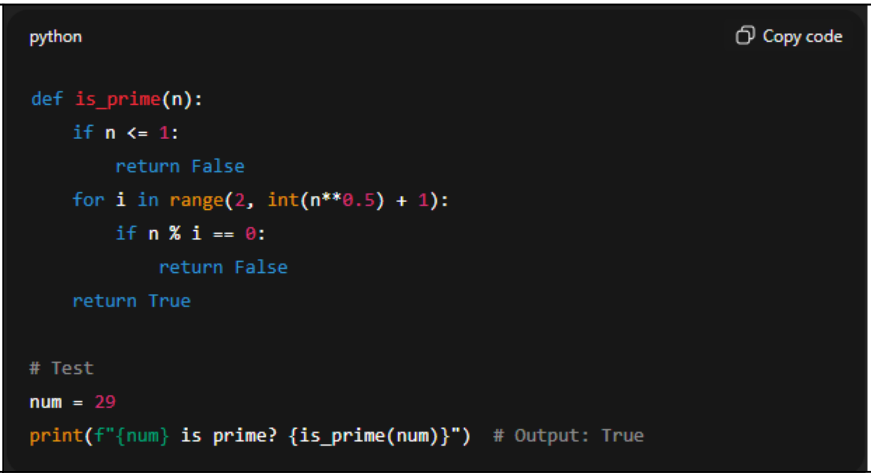
```python
from datetime import datetime

date_str = "2025-11-13"  # Original format yyyy-mm-dd

# Convert to datetime object
date_obj = datetime.strptime(date_str, "%Y-%m-%d")

# Format to dd-mm-yyyy
new_date_str = date_obj.strftime("%d-%m-%Y")
print(new_date_str)  # Output: 13-11-2025
```

```python
def fibonacci(n):
    fib_series = [0, 1]  # First two numbers
    for i in range(2, n):
        fib_series.append(fib_series[i-1] + fib_series[i-2])
    return fib_series[:n]

# Test
n = 10
print(fibonacci(n))

# output
# [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```python
words = ["Python", "is", "fun"]

# Combine list elements into a single string with spaces
single_string = " ".join(words)
print(single_string)  # Output: Python is fun

# Combine without spaces
single_string_no_space = "".join(words)
print(single_string_no_space)  # Output: Pythonisfun
```

Write a program to check if a number is a prime.

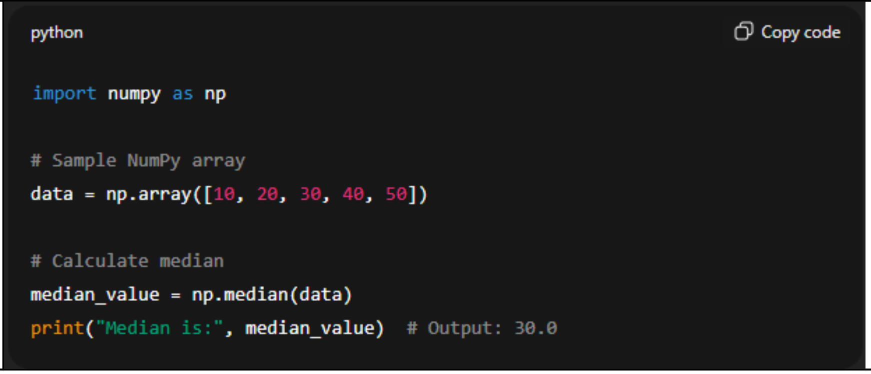
In Python, a prime number is a number greater than 1 that has no divisors other than 1 and itself. You can check it using a simple loop.



```python
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

# Test
num = 29
print(f"{num} is prime? {is_prime(num)}")  # Output: True
```

Write a program to calculate the median in Python using the NumPy arrays.

In Python, you can calculate the median of a dataset using NumPy with the np.median() function.

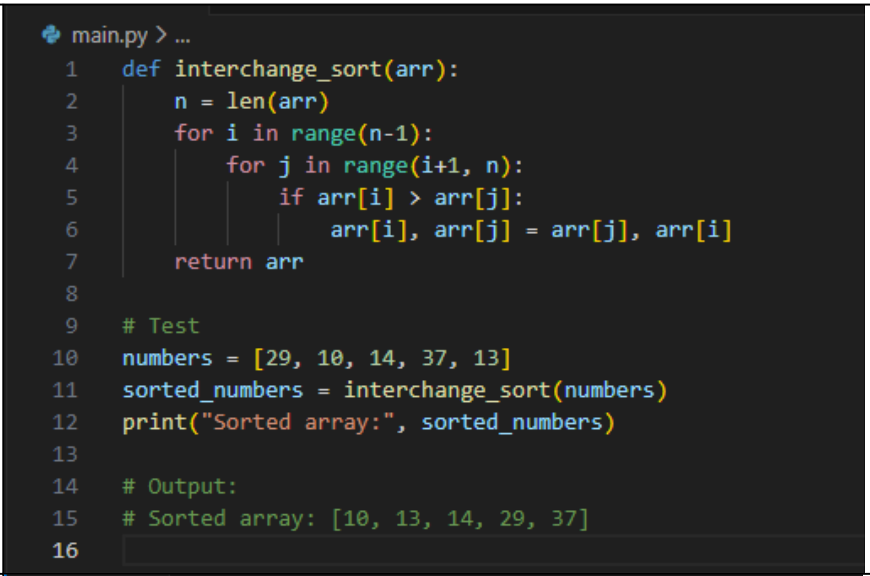


```python
import numpy as np

# Sample NumPy array
data = np.array([10, 20, 30, 40, 50])

# Calculate median
median_value = np.median(data)
print("Median is:", median_value)  # Output: 30.0
```

Write a program to execute the interchange sort algorithm.

Interchange Sort is a simple sorting algorithm similar to bubble sort, where every element is compared with all other elements and swapped if they are out of order.

"Compare each element with all subsequent elements and swap if out of order."



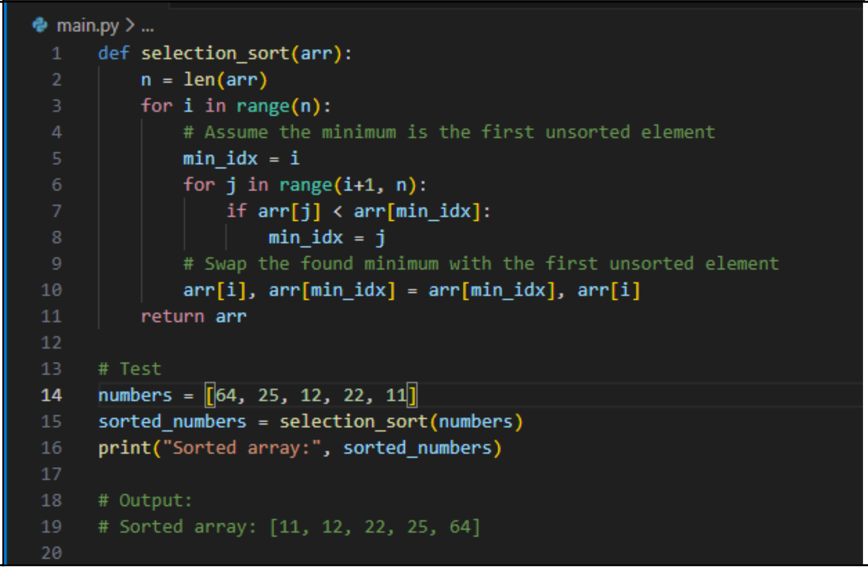
main.py > ...

```python
def interchange_sort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(i+1, n):
            if arr[i] > arr[j]:
                arr[i], arr[j] = arr[j], arr[i]
    return arr

# Test
numbers = [29, 10, 14, 37, 13]
sorted_numbers = interchange_sort(numbers)
print("Sorted array:", sorted_numbers)

# Output:
# Sorted array: [10, 13, 14, 29, 37]

```

Write a program to execute the selection sort algorithm.Write a program to execute the selection sort algorithm.

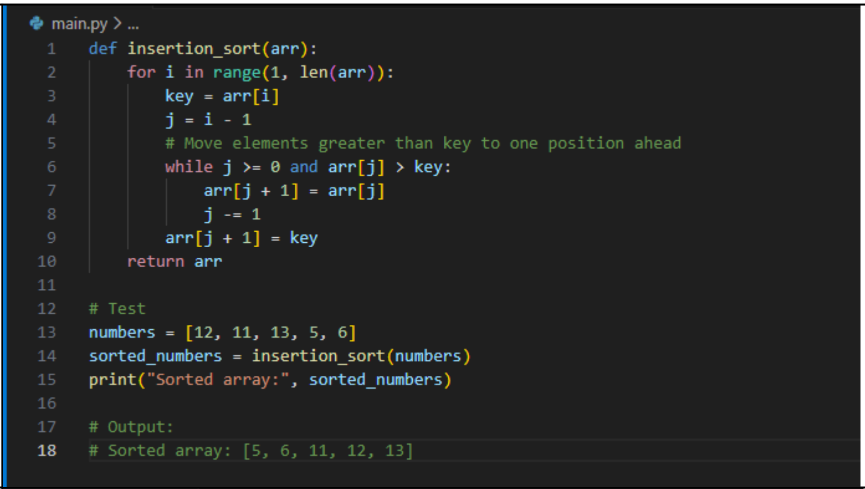
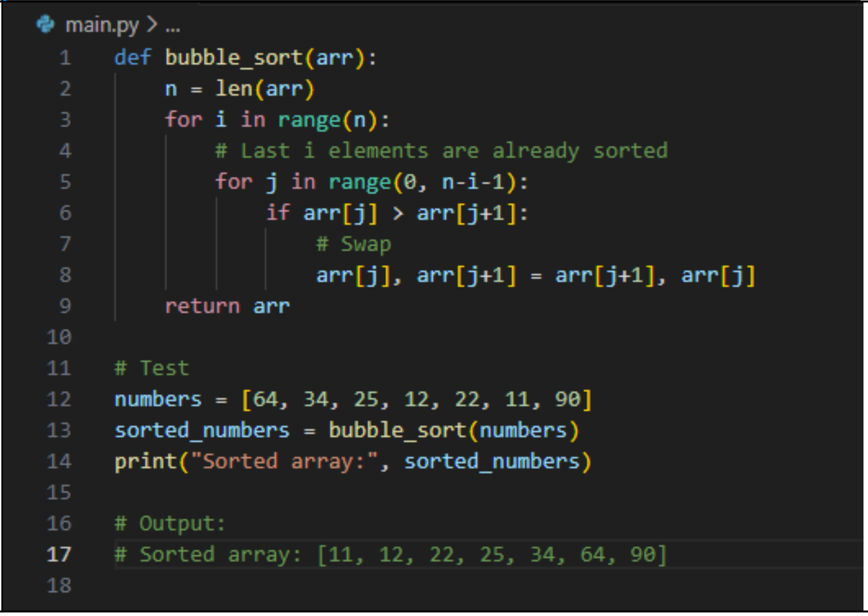
Selection Sort is a sorting algorithm where the smallest (or largest) element is repeatedly selected from the unsorted portion and swapped with the first unsorted element.

"Repeatedly select the minimum element from the unsorted part and swap it with the first unsorted element."



main.py > ...

```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        # Assume the minimum is the first unsorted element
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        # Swap the found minimum with the first unsorted element
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

# Test
numbers = [64, 25, 12, 22, 11]
sorted_numbers = selection_sort(numbers)
print("Sorted array:", sorted_numbers)

# Output:
# Sorted array: [11, 12, 22, 25, 64]

```

	Write a program to execute the insertion sort algorithm. Insertion Sort is a simple sorting algorithm where elements are picked one by one and inserted into their correct position in the sorted portion of the array. "Pick each element and insert it into the sorted part of the array by shifting larger elements to the right."		
	Write a program to execute the bubble sort algorithm. Bubble Sort is a simple sorting algorithm where adjacent elements are repeatedly swapped if they are in the wrong order. "Loop through the list multiple times, swapping adjacent elements if they’re in the wrong order until sorted."		

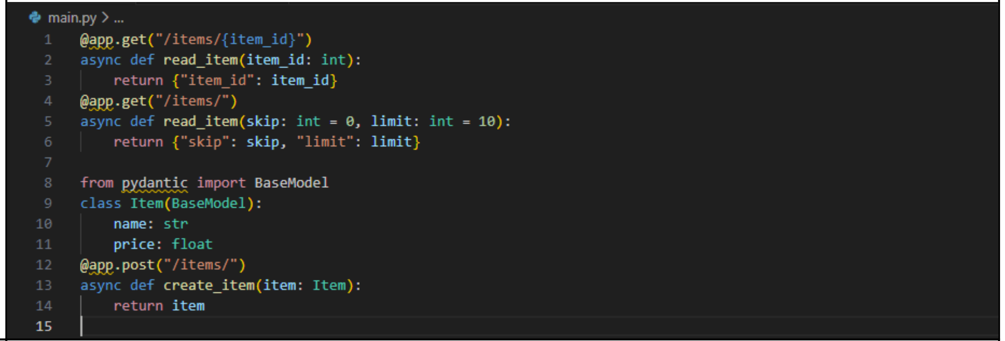
main.py > ...
```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        # Move elements greater than key to one position ahead
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

# Test
numbers = [12, 11, 13, 5, 6]
sorted_numbers = insertion_sort(numbers)
print("Sorted array:", sorted_numbers)

# Output:
# Sorted array: [5, 6, 11, 12, 13]
```

main.py > ...
```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Last i elements are already sorted
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                # Swap
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# Test
numbers = [64, 34, 25, 12, 22, 11, 90]
sorted_numbers = bubble_sort(numbers)
print("Sorted array:", sorted_numbers)

# Output:
# Sorted array: [11, 12, 22, 25, 34, 64, 90]

```

Flask	1. Flask là gì? Tại sao nó được gọi là micro-framework? Flask là một micro-framework của Python để xây dựng ứng dụng web nhanh và nhẹ. Nó gọi là “micro” vì rất tối giản: không có sẵn ORM, hệ thống xác thực hay form handling. Flask chỉ cung cấp những phần cơ bản như routing, request/response, và template engine Jinja2. Các tính năng nâng cao như database hay authentication do lập trình viên thêm bằng extensions (ví dụ Flask-SQLAlchemy, Flask-Login…). Nhờ vậy, ứng dụng vừa nhẹ, vừa linh hoạt, dễ mở rộng theo nhu cầu.	Flask is a Python micro-framework for building web applications quickly and lightly. It’s called “micro” because it is very minimal: it doesn’t include ORM, authentication, or form handling by default. Flask only provides core parts like routing, request/response handling, and the Jinja2 template engine. Advanced features like database or authentication can be added with extensions (e.g., Flask-SQLAlchemy, Flask-Login…). This keeps apps light, flexible, and easy to extend.
	2. Flask định nghĩa route như thế nào, và cơ chế routing hoạt động ra sao? Trong Flask, route được định nghĩa bằng decorator @app.route() trên một hàm. Khi Flask khởi động, decorator này đăng ký URL và map đến hàm đó. Khi client gửi request đến URL: Flask duyệt bảng routing để tìm route phù hợp. Gọi hàm xử lý (view function). Nhận dữ liệu trả về từ hàm và biến thành HTTP response gửi lại cho client. Tóm lại: route → ánh xạ URL đến một hàm → Flask gọi hàm → trả response.	In Flask, a route is defined using the @app.route() decorator on a function. When Flask starts, the decorator registers the URL and maps it to the function. When a client sends a request: Flask looks in the routing table to find the matching route. Calls the handler function (view function) Takes the return value and converts it into an HTTP response to send back to the client. In short: route → maps URL to a function → Flask calls the function → returns response.
	3. Sự khác nhau giữa request, session, và g trong Flask là gì? Trong Flask, request, session, và g đều thuộc context, nhưng dùng cho mục đích khác nhau: request: chứa tất cả dữ liệu của HTTP request hiện tại (URL, method, headers, body…). Chỉ tồn tại trong một request, dùng để đọc dữ liệu client gửi lên. session: lưu dữ liệu tạm cho người dùng, tồn tại qua nhiều request nhờ signed cookie. Dùng để lưu trạng thái ngắn hạn như user_id, theme, giỏ hàng… g: viết tắt “global for one request”, lưu dữ liệu tạm thời trong một request, ví dụ kết nối database mở sẵn. Không lưu giữa các request. Tóm lại: request → đọc dữ liệu HTTP request. session → lưu trạng thái user qua nhiều request. g → chia sẻ dữ liệu tạm trong một request duy nhất.	In Flask, request, session, and g are all part of the context, but they serve different purposes: request: contains all data of the current HTTP request (URL, method, headers, body…). It exists only during one request and is used to read what the client sent. session: stores temporary data for a user, persists across multiple requests using a signed cookie. Used for short-term state like user_id, theme, or shopping cart. g: stands for “global for one request”, stores temporary data during a single request, e.g., a database connection. It does not persist between requests. In short: request → read HTTP request data. session → store user state across requests. g → share temporary data in one request only.

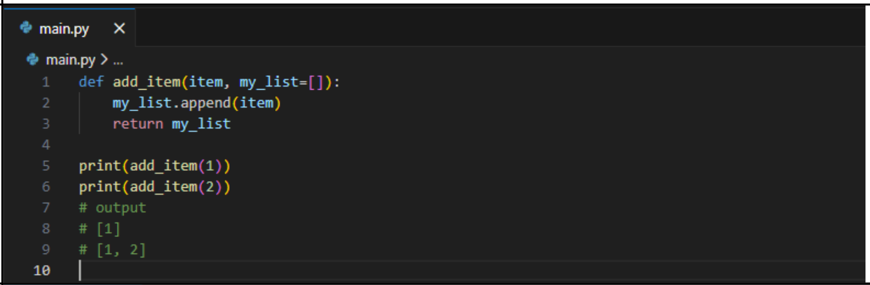
	4. Giải thích cơ chế "Application Context" và "Request Context" trong Flask. Trong Flask, context giúp quản lý các biến gắn với ứng dụng hoặc request mà không cần truyền qua từng hàm. Application Context (app context) Chứa thông tin liên quan đến ứng dụng, ví dụ current_app và g. current_app cho phép truy cập app hiện tại mà không cần truyền biến app. g là nơi lưu trữ dữ liệu tạm thời dùng chung trong vòng đời ứng dụng. App context có thể tồn tại độc lập với request và push thủ công khi chạy code ngoài request. Request Context (request context) Chứa thông tin của một request cụ thể, ví dụ request và session. Flask tự động push request context khi request đến và pop khi kết thúc. Cho phép truy cập request hay session mà không cần truyền vào từng hàm. Tóm lại: App context liên quan đến ứng dụng nói chung, request context liên quan đến từng request. Cơ chế này giúp Flask thread-safe và tránh xung đột giữa các request chạy song song.	In Flask, context manages variables linked to the app or a request without passing them to every function. Application Context (app context) Holds info related to the application, like current_app and g. current_app lets you access the current app without passing it around. g is a place to store temporary data for the app's lifetime. App context can exist outside of a request and can be pushed manually. Request Context (request context) Holds info for a specific request, like request and session. Flask automatically pushes request context when a request comes and pops it when it ends. Lets you access request or session without passing them to each function. In short: App context is for the app itself; request context is for each request. This makes Flask thread-safe and avoids conflicts between simultaneous requests.
	5. Làm sao để cấu hình Flask thành production? Dùng WSGI nào? "Để chạy Flask production, không nên dùng app.run() vì server này chỉ dành cho phát triển, không tối ưu về hiệu năng và bảo mật. Thay vào đó, thường dùng WSGI server như: Gunicorn: phổ biến, dễ cấu hình, hỗ trợ nhiều worker và thread. uWSGI: mạnh, tùy biến cao, nhiều worker, caching, logging. Waitress: nhẹ, dễ dùng, phù hợp Windows hoặc môi trường đơn giản. Thường Flask chạy sau reverse proxy như Nginx hoặc Apache để xử lý SSL, load balancing, static files, caching. Quy trình chuẩn: Client → Nginx → Gunicorn/uWSGI → Flask → Response Cần chú ý cấu hình logging, error handling, số worker, timeout để ứng dụng ổn định."	To run Flask in production, don't use app.run() because it's only for development and not optimized for performance or security. Use a WSGI server instead: Gunicorn: popular, easy to configure, supports multiple workers and threads. uWSGI: powerful, highly configurable, supports many workers, caching, logging. Waitress: lightweight, easy to use, good for Windows or simple setups. Flask usually runs behind a reverse proxy like Nginx or Apache to handle SSL, load balancing, static files, and caching. Standard flow: Client → Nginx → Gunicorn/uWSGI → Flask → Response Also, configure logging, error handling, number of workers, and timeouts to keep the app stable.

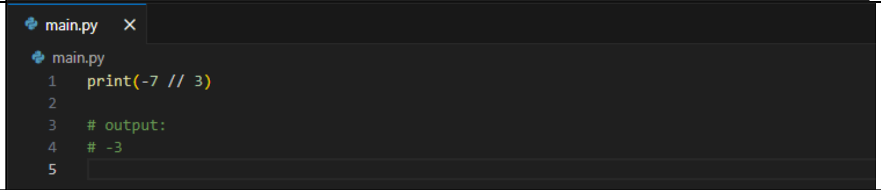
FastAPI	1. FastAPI dựa trên chuẩn ASGI là gì? Khác gì so với WSGI của Flask? ASGI (Asynchronous Server Gateway Interface) là chuẩn giao tiếp giữa server và ứng dụng Python hỗ trợ async I/O, giúp xử lý nhiều kết nối đồng thời một cách hiệu quả. FastAPI dựa trên ASGI, cho phép viết các endpoint bất đồng bộ (async def) để xử lý I/O-bound tasks như gọi API, đọc/ghi database, hay thao tác file mà không block các request khác, từ đó đạt concurrency cao. Ngược lại, Flask dùng WSGI (Web Server Gateway Interface) – synchronous, chỉ xử lý một request tại một thời điểm trong mỗi worker, nên cần nhiều worker để tăng throughput. Khi triển khai FastAPI, thường dùng ASGI server như Uvicorn hoặc Hypercorn để tận dụng async, trong khi Flask thường chạy qua WSGI server như Gunicorn hoặc uWSGI. "ASGI là chuẩn cho ứng dụng web Python hỗ trợ bất đồng bộ (async). Nhờ vậy FastAPI có thể xử lý nhiều request cùng lúc, đặc biệt tốt cho các tác vụ I/O-bound như gọi API, truy vấn DB, đọc file… WSGI là chuẩn cũ, chỉ hỗ trợ synchronous. Flask chạy theo kiểu mỗi worker chỉ xử lý được 1 request tại một thời điểm, muốn tăng throughput thì phải tăng số worker."	"ASGI is a newer Python web standard that supports async code. This lets FastAPI handle many requests at the same time, which is great for I/O-bound tasks like calling external APIs, database queries, or reading files. WSGI is an older, synchronous standard. Flask uses WSGI, so each worker can handle only one request at a time. To increase throughput, you need to run more workers."
	2. FastAPI hỗ trợ khai báo request parameters như thế nào? Trong FastAPI, request parameters có thể được khai báo trực tiếp trong định nghĩa hàm endpoint với kiểu dữ liệu rõ ràng (type hints), và framework sẽ tự động parse và validate: - Path parameters: lấy giá trị từ URL. - Query parameters: lấy từ query string. - Request body: dùng Pydantic model để parse JSON body và validate dữ liệu. FastAPI tự động kiểm tra type, bắt lỗi nếu không đúng, và sinh thông báo lỗi chuẩn JSON, giúp giảm code boilerplate và tăng tính an toàn. "FastAPI cho khai báo request parameters ngay trong hàm endpoint bằng type hints. Nhờ vậy FastAPI tự hiểu và tự validate luôn. Path params: lấy từ URL. Query params: lấy từ query string. Request body: thường dùng Pydantic model để đọc JSON và kiểm tra dữ liệu. Điểm hay là mình không phải tự parse hay tự validate nữa — FastAPI làm hết và trả lỗi dạng JSON rất rõ ràng."	FastAPI lets you declare request parameters directly in the endpoint function using type hints. FastAPI will parse and validate them automatically. Path parameters: values from the URL path. Query parameters: values from the query string. Request body: usually defined with a Pydantic model to read and validate JSON. The good thing is you don't need to manually parse or validate data — FastAPI does it for you and returns clear JSON error messages.

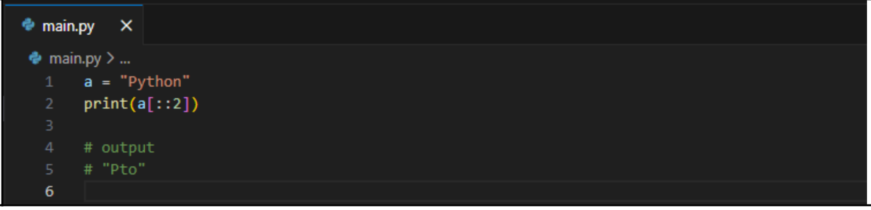
	3. Pydantic dùng để làm gì trong FastAPI? Pydantic trong FastAPI được dùng để định nghĩa các model dữ liệu với type hints, giúp: 1. Validate dữ liệu đầu vào (data validation): Kiểm tra kiểu dữ liệu, giá trị bắt buộc, định dạng (ví dụ email, URL, số dương…). Nếu dữ liệu không hợp lệ, FastAPI tự động trả lỗi chuẩn JSON. 2. Serialize/deserialize: Chuyển đổi dữ liệu Python → JSON khi trả response, và JSON → Python object khi nhận request body. 3. Tạo schema OpenAPI tự động: FastAPI dùng Pydantic models để sinh tài liệu Swagger / ReDoc, giúp client biết chính xác cấu trúc request/response mà không cần viết thêm code. Tóm lại: Pydantic giúp type-safe, giảm lỗi runtime, và tự động tạo tài liệu API, làm FastAPI rất mạnh trong việc xây dựng API hiện đại và đáng tin cậy.	
	4. FastAPI tự sinh tài liệu API (Swagger / Redoc) như thế nào? "FastAPI tự tạo tài liệu API (Swagger UI và ReDoc) dựa trên type hints và Pydantic models mà bạn định nghĩa trong endpoint. Cách hoạt động: FastAPI lấy thông tin về type và validation từ function và Pydantic model. Sinh ra schema OpenAPI mô tả các endpoint, request/response, kiểu dữ liệu, giá trị mặc định, và mô tả. Cung cấp giao diện web: /docs → Swagger UI, test API trực tiếp trong trình duyệt. /redoc → ReDoc, tài liệu đẹp, dễ đọc. Ưu điểm là bạn không cần viết tài liệu thủ công, mọi thứ đồng bộ với code và luôn cập nhật theo API."	FastAPI automatically creates API docs (Swagger UI and ReDoc) using the type hints and Pydantic models in your endpoints. How it works: FastAPI collects type and validation info from your functions and Pydantic models. It generates an OpenAPI schema describing endpoints, request/response data, types, default values, and descriptions. It provides web pages: /docs → Swagger UI, you can test API in the browser. /redoc → ReDoc, shows nice, easy-to-read documentation. The advantage is you don't need to write docs manually; everything stays synced with your code and updates automatically.

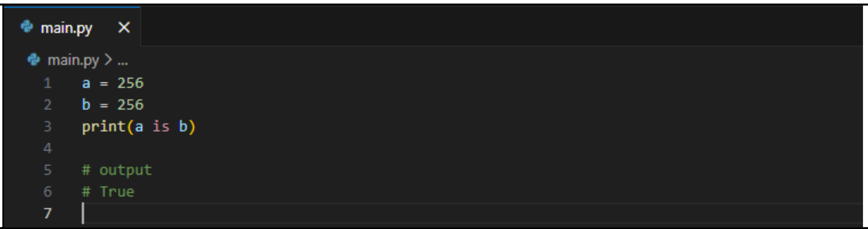
main.py > ...
```python
@app.get("/items/{item_id}")
async def read_item(item_id: int):
    return {"item_id": item_id}
@app.get("/items/")
async def read_item(skip: int = 0, limit: int = 10):
    return {"skip": skip, "limit": limit}

from pydantic import BaseModel
class Item(BaseModel):
    name: str
    price: float
@app.post("/items/")
async def create_item(item: Item):
    return item

```

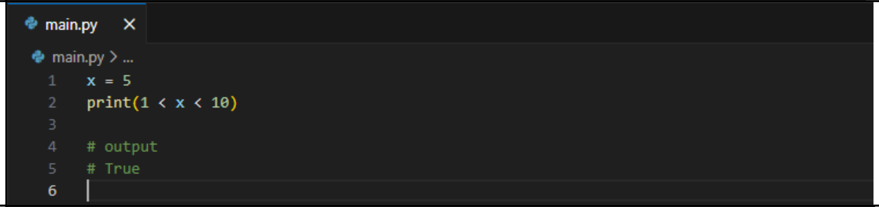
	5. Sự khác nhau giữa function sync và async trong FastAPI? Khi nào nên dùng async? Trong FastAPI, endpoint có thể là sync (def) hoặc async (async def): Sync function (def): chạy tuần tự, blocking. Mỗi request phải chờ hàm xong mới xử lý request khác. Thích hợp với CPU-bound tasks hoặc các tác vụ nhanh, ít I/O. Async function (async def): dùng await cho các tác vụ I/O-bound như gọi API, truy vấn database, đọc/ghi file. Nhiều request có thể chạy đồng thời trong cùng một worker, tăng concurrency. Khi nào dùng async: Endpoint làm nhiều I/O chậm (network, DB…). Muốn xử lý nhiều request cùng lúc mà không cần tăng worker. Lưu ý: Async không giúp CPU-bound tasks, những tác vụ nặng CPU vẫn cần worker riêng hoặc task queue (ví dụ Celery). Tóm lại: Async → tận dụng event loop cho I/O-bound tasks. Sync → nhanh, CPU-bound, ít I/O.	In FastAPI, an endpoint can be sync (def) or async (async def): Sync function (def): runs sequentially and blocks. Each request waits for the function to finish before the worker can handle another request. Good for CPU-bound tasks or fast operations with little I/O. Async function (async def): uses await for I/O-bound operations like API calls, database queries, or reading/writing files. Multiple requests can run at the same time in one worker, increasing concurrency. When to use async: The endpoint does slow I/O tasks (network, DB). You want to handle many requests at the same time without adding more workers. Note: Async does not help CPU-bound tasks; heavy CPU tasks still need separate workers or a task queue like Celery.

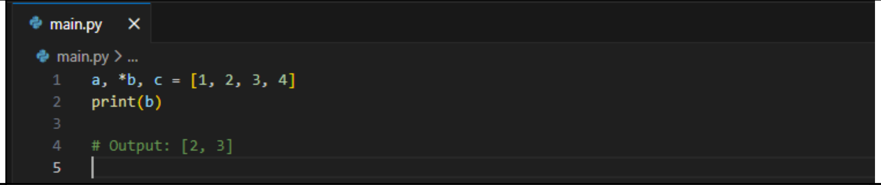
Trick	Giải thích:
	
	- my_list=[] là mutable, được tạo 1 lần khi function định nghĩa.
	- Các lần gọi sau cùng dùng list cũ, không tạo mới.
		
		main.py ×
		main.py > ...
		`def add_item(item, my_list=[]):`
		`    my_list.append(item)`
		`    return my_list`
		
		`print(add_item(1))`
		`print(add_item(2))`
		`# output`
		`# [1]`
		`# [1, 2]`

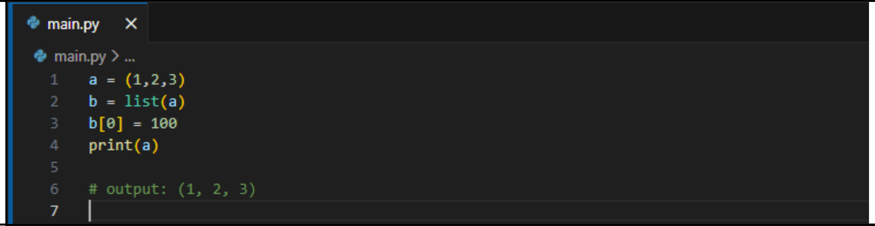
	Giải thích:
	
	Python làm floor division → làm tròn xuống số nguyên gần nhất.
		
		main.py ×
		main.py
		`print(-7 // 3)`
		
		`# output:`
		`# -3`

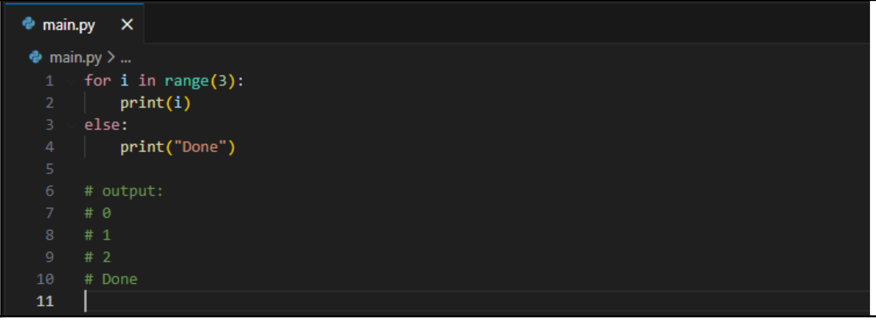
	Giải thích:
	
	start:end:step, step=2 → lấy 1 ký tự 1 lần nhảy 2.
		
		main.py ×
		main.py > ...
		`a = "Python"`
		`print(a[::2])`
		
		`# output`
		`# "Pto"`

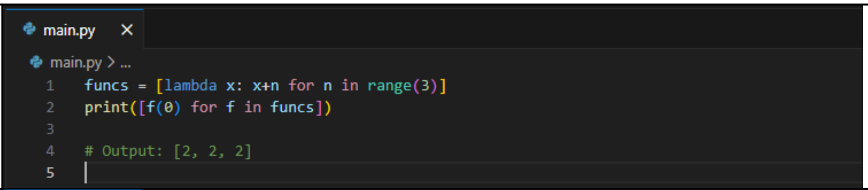
	Giải thích:
	
	- Python cache số nguyên nhỏ từ -5 đến 256.
	- Nếu a = 257; b = 257 → False.
		
		main.py
		`a = 256`
		`b = 256`
		`print(a is b)`
		
		`# output`
		`# True`

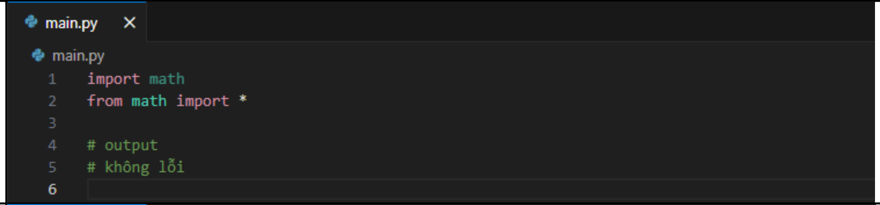
	Giải thích:
	
	Python cho phép so sánh “chuỗi”: 1 < x and x < 10.
		
		main.py
		`x = 5`
		`print(1 < x < 10)`
		
		`# output`
		`# True`

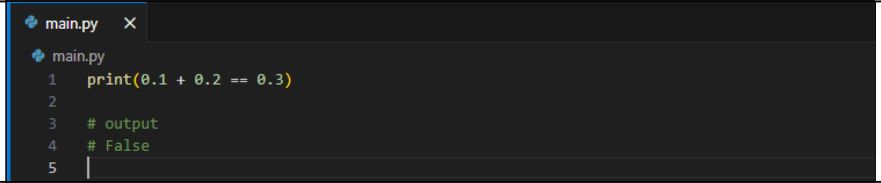
	Giải thích:
	
	*b lấy tất cả phần “giữa” giữa a và c.
		
		main.py
		`a, *b, c = [1, 2, 3, 4]`
		`print(b)`
		
		`# Output: [2, 3]`

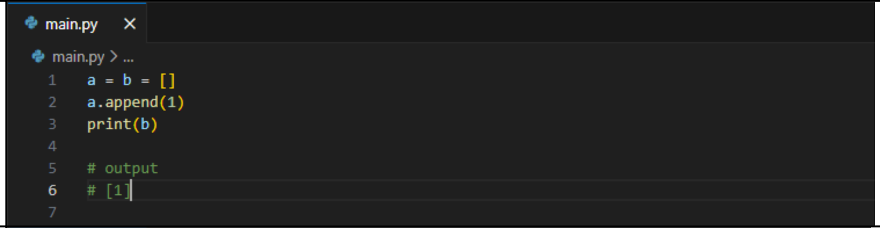
	Giải thích:
	
	tuple immutable, list copy không ảnh hưởng tuple gốc.
		
		main.py
		`a = (1,2,3)`
		`b = list(a)`
		`b[0] = 100`
		`print(a)`
		
		`# output: (1, 2, 3)`

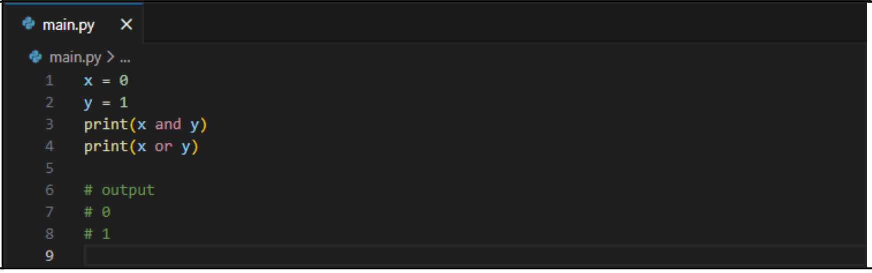
	Giải thích:
	
	else chạy khi loop không bị break.
		
		main.py
		`for i in range(3):`
		`    print(i)`
		`else:`
		`    print("Done")`
		
		`# output:`
		`# 0`
		`# 1`
		`# 2`
		`# Done`

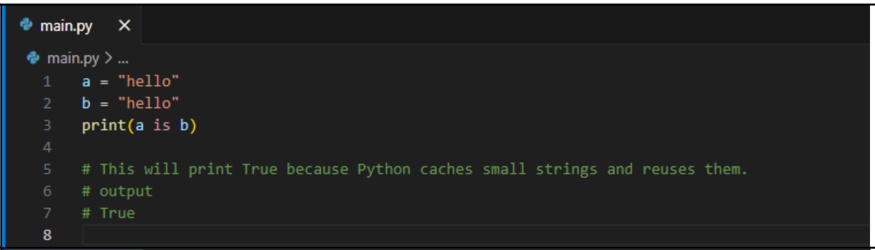
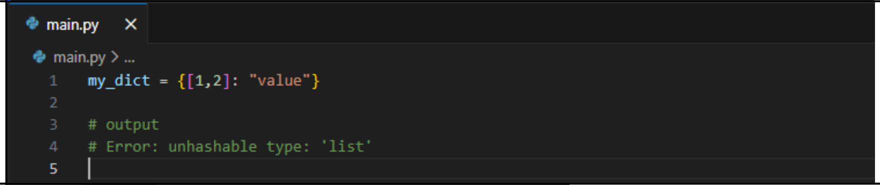
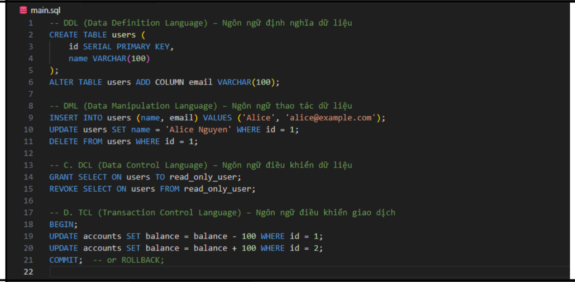
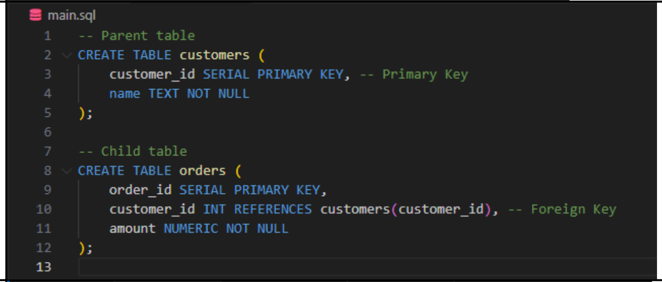
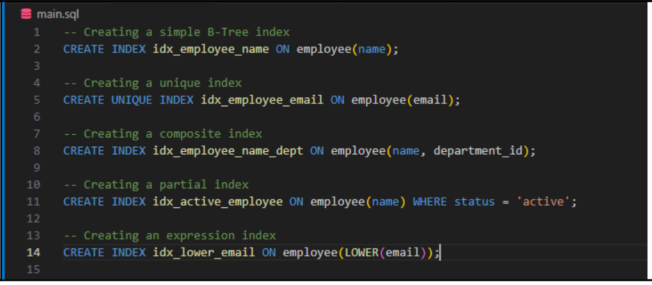
	Giải thích:
	
	Lambda capture biến theo reference, lúc gọi n=2 (lớp cuối cùng).
	Nếu muốn fix: lambda x, n=n: x+n.
		
		`funcs = [lambda x: x+n for n in range(3)]`
		`print([f(0) for f in funcs])`
		
		`# Output: [2, 2, 2]`	

	Đáp án:
	
	Không lỗi, nhưng ghi đè tên đã import, dễ gây nhầm lẫn.
		
		`import math`
		`from math import *`
		
		`# output`
		`# không lỗi`	

	Giải thích:
	
	Floating point không chính xác tuyệt đối, 0.1+0.2 ≈ 0.30000000000000004.
		
		`print(0.1 + 0.2 == 0.3)`
		
		`# output`
		`# False`	

	Giải thích:
	
	a và b cùng trỏ tới 1 list, thay đổi cùng ảnh hưởng.
		
		`a = b = []`
		`a.append(1)`
		`print(b)`
		
		`# output`
		`# [1]`	

	Giải thích:
	
	- and trả về giá trị đầu False hoặc giá trị cuối True
	- or trả về giá trị đầu True hoặc giá trị cuối False
		
		`x = 0`
		`y = 1`
		`print(x and y)`
		`print(x or y)`
		
		`# output`
		`# 0`
		`# 1`	

	Giải thích: Python intern small strings, tiết kiệm bộ nhớ.	
	Giải thích: Key trong dict phải immutable, list mutable không dùng được.	
	What is SQL? Differentiate between DDL, DML, DCL, and TCL. SQL (Structured Query Language) is a standard language used to interact with relational databases. You use SQL to store, retrieve, update, and manage data. Almost all relational database systems—like PostgreSQL, MySQL, SQL Server, Oracle—use SQL as their core language. DDL is used to define or modify the structure of the database, such as creating or altering tables. DML is used to work with the actual data inside tables (insert, update	
	What is the difference between a Primary Key and a Foreign Key? A Primary Key is a column (or set of columns) that uniquely identifies each row in a table. It guarantees two things: 1. Uniqueness – no two rows can have the same primary key value. 2. Non-null – a primary key cannot contain NULL values. A Foreign Key, on the other hand, is a column (or set of columns) in one table that refers to the primary key of another table. Its purpose is to maintain referential integrity, meaning it ensures the relationship between the two tables remains valid – for example, you cannot insert a value in	
	What is an Index? What types of indexes exist? An index in SQL is a database structure that improves the speed of data retrieval operations on a table at the cost of additional storage and slightly slower write operations. Think of it like an index in a book: instead of scanning every page to find a topic, you can go straight to the page number listed in the index. There are several types of indexes in SQL: - Primary Key Index: Automatically created when you define a primary key, ensures uniqueness.	

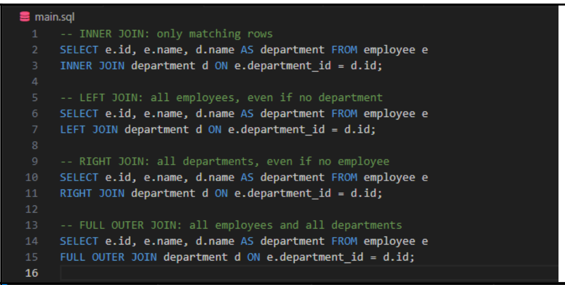
main.py > ...
```python
a = "hello"
b = "hello"
print(a is b)

# This will print True because Python caches small strings and reuses them.
# output
# True
```

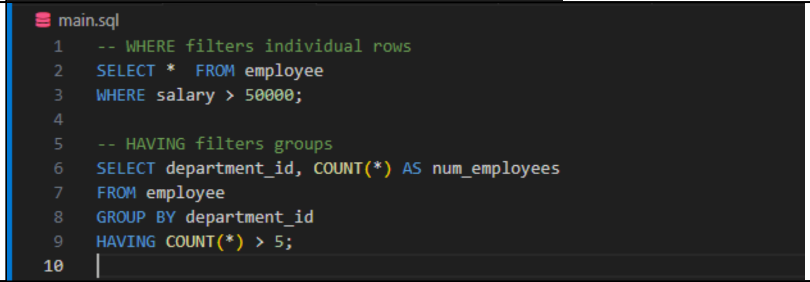
main.py > ...
```python
my_dict = {[1,2]: "value"}

# output
# Error: unhashable type: 'list'
```

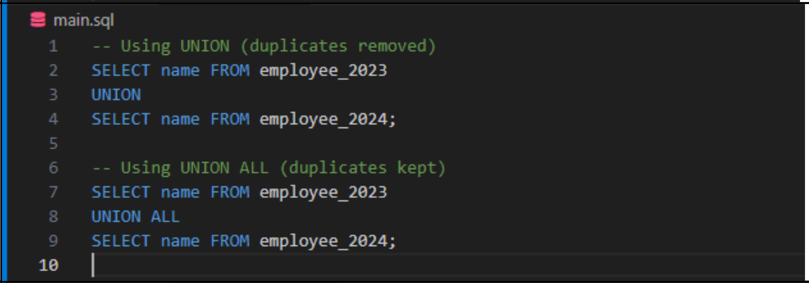
main.sql
```sql
-- DDL (Data Definition Language) - Ngôn ngữ định nghĩa dữ liệu
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100)
);
ALTER TABLE users ADD COLUMN email VARCHAR(100);

-- DML (Data Manipulation Language) - Ngôn ngữ thao tác dữ liệu
INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
UPDATE users SET name = 'Alice Nguyen' WHERE id = 1;
DELETE FROM users WHERE id = 1;

-- C. DCL (Data Control Language) - Ngôn ngữ điều khiển dữ liệu
GRANT SELECT ON users TO read_only_user;
REVOKE SELECT ON users FROM read_only_user;

-- D. TCL (Transaction Control Language) - Ngôn ngữ điều khiển giao dịch
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;  -- or ROLLBACK;
```

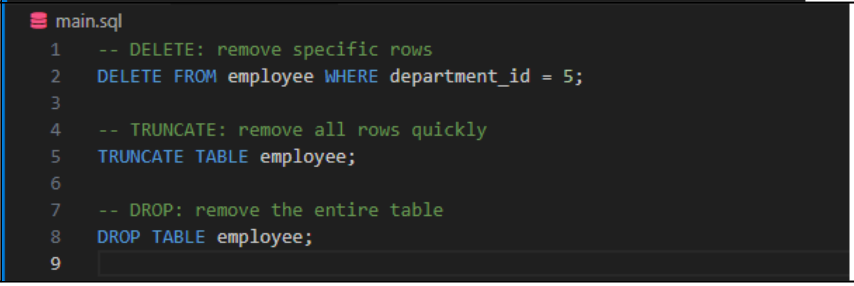
main.sql
```sql
-- Parent table
CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY, -- Primary Key
    name TEXT NOT NULL
);

-- Child table
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    customer_id INT REFERENCES customers(customer_id), -- Foreign Key
    amount NUMERIC NOT NULL
);
```

main.sql
```sql
-- Creating a simple B-Tree index
CREATE INDEX idx_employee_name ON employee(name);

-- Creating a unique index
CREATE UNIQUE INDEX idx_employee_email ON employee(email);

-- Creating a composite index
CREATE INDEX idx_employee_name_dept ON employee(name, department_id);

-- Creating a partial index
CREATE INDEX idx_active_employee ON employee(name) WHERE status = 'active';

-- Creating an expression index
CREATE INDEX idx_lower_email ON employee(LOWER(email));
```

	What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?
	
	In SQL, joins are used to combine rows from two or more tables based on a related column.
	- INNER JOIN returns only the rows where there is a match in both tables.
	- LEFT JOIN returns all rows from the left table and the matching rows from the right table; if there is no match, the right side will have NULL.
	- RIGHT JOIN is the opposite: it returns all rows from the right table and
		
		main.sql
		`-- INNER JOIN: only matching rows`
		`SELECT e.id, e.name, d.name AS department FROM employee e`
		`INNER JOIN department d ON e.department_id = d.id;`
		
		`-- LEFT JOIN: all employees, even if no department`
		`SELECT e.id, e.name, d.name AS department FROM employee e`
		`LEFT JOIN department d ON e.department_id = d.id;`
		
		`-- RIGHT JOIN: all departments, even if no employee`
		`SELECT e.id, e.name, d.name AS department FROM employee e`
		`RIGHT JOIN department d ON e.department_id = d.id;`
		
		`-- FULL OUTER JOIN: all employees and all departments`
		`SELECT e.id, e.name, d.name AS department FROM employee e`
		`FULL OUTER JOIN department d ON e.department_id = d.id;`

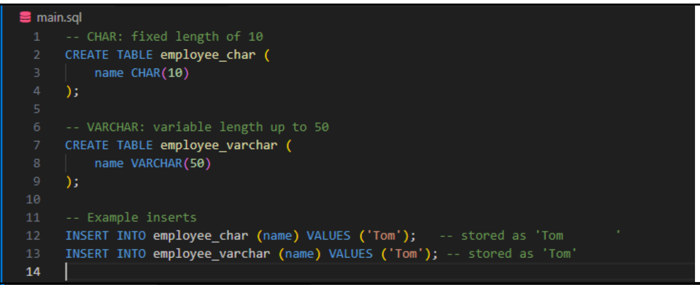
	What is the difference between WHERE and HAVING
	
	In SQL, WHERE is used to filter rows before any groupings are made, while HAVING is used to filter groups after GROUP BY has been applied. In other words, WHERE works on individual rows, and HAVING works on aggregated results like COUNT, SUM, AVG, etc.
		
		main.sql
		`-- WHERE filters individual rows`
		`SELECT *  FROM employee`
		`WHERE salary > 50000;`
		
		`-- HAVING filters groups`
		`SELECT department_id, COUNT(*) AS num_employees`
		`FROM employee`
		`GROUP BY department_id`
		`HAVING COUNT(*) > 5;`

	What is the difference between UNION and UNION ALL?
	
	In SQL, UNION combines the result sets of two or more queries and removes duplicate rows, while UNION ALL combines result sets including all duplicates. So, UNION ensures uniqueness but may be slightly slower because it needs to check for duplicates, whereas UNION ALL is faster because it just appends all rows.
		
		main.sql
		`-- Using UNION (duplicates removed)`
		`SELECT name FROM employee_2023`
		`UNION`
		`SELECT name FROM employee_2024;`
		
		`-- Using UNION ALL (duplicates kept)`
		`SELECT name FROM employee_2023`
		`UNION ALL`
		`SELECT name FROM employee_2024;`

	What is the difference between TRUNCATE, DELETE, and DROP?
	
	In SQL, DELETE, TRUNCATE, and DROP are used to remove data, but they work differently.
	- DELETE removes specific rows from a table based on a condition and can be rolled back if inside a transaction.
	- TRUNCATE removes all rows from a table quickly, resets identity counters, but usually cannot be rolled back in many databases.
	- DROP removes the entire table or database structure permanently, including all data, indexes, and constraints.
		
		main.sql
		`-- DELETE: remove specific rows`
		`DELETE FROM employee WHERE department_id = 5;`
		
		`-- TRUNCATE: remove all rows quickly`
		`TRUNCATE TABLE employee;`
		
		`-- DROP: remove the entire table`
		`DROP TABLE employee;`

What is the difference between CHAR and VARCHAR?

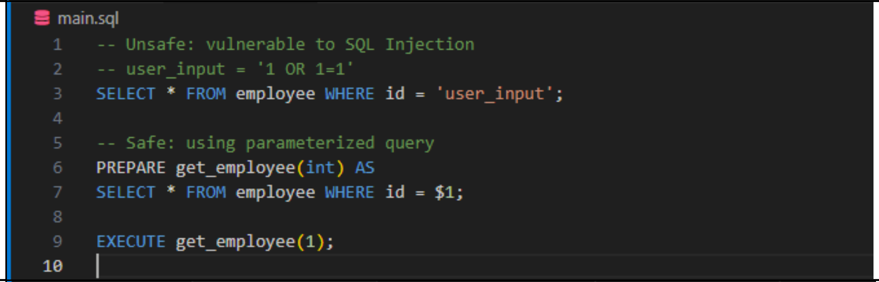
In SQL, CHAR and VARCHAR are used to store text, but they work differently. CHAR(n) is a fixed-length string, meaning it always uses n characters; if the input is shorter, it pads with spaces. VARCHAR(n) is a variable-length string, storing only the characters you input up to n, so it's more space-efficient. Use CHAR when the length of data is consistent, and VARCHAR when it varies.


main.sql

```sql
-- CHAR: fixed length of 10
CREATE TABLE employee_char (
    name CHAR(10)
);

-- VARCHAR: variable length up to 50
CREATE TABLE employee_varchar (
    name VARCHAR(50)
);

-- Example inserts
INSERT INTO employee_char (name) VALUES ('Tom');   -- stored as 'Tom       '
INSERT INTO employee_varchar (name) VALUES ('Tom'); -- stored as 'Tom'
```

What is SQL Injection? How can it be prevented?

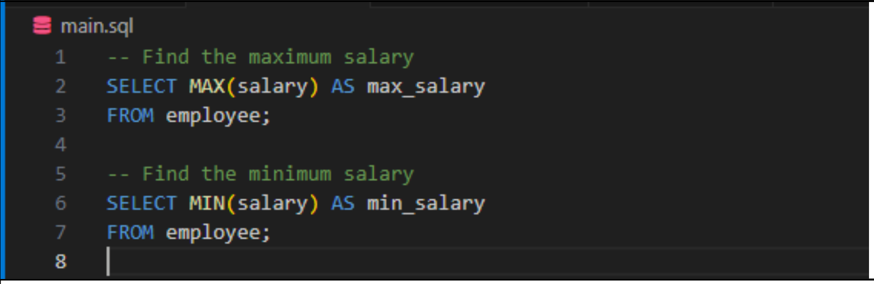
SQL Injection is a security vulnerability that occurs when an attacker can manipulate an SQL query by inserting malicious input, potentially allowing them to read, modify, or delete data in the database. It happens when user input is directly concatenated into SQL statements without proper validation.
Prevention methods:
- Use parameterized queries / prepared statements: never concatenate user input directly.


main.sql

```sql
-- Unsafe: vulnerable to SQL Injection
-- user_input = '1 OR 1=1'
SELECT * FROM employee WHERE id = 'user_input';

-- Safe: using parameterized query
PREPARE get_employee(int) AS
SELECT * FROM employee WHERE id = $1;

EXECUTE get_employee(1);
```

Write a query to find the maximum/minimum value in a column.

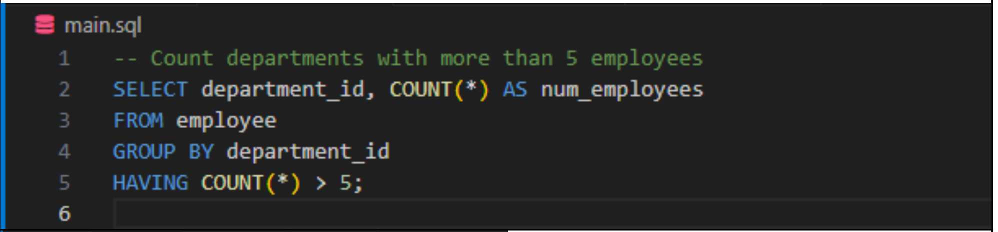
In SQL, you can use the MAX() function to find the largest value in a column, and MIN() to find the smallest value. These are aggregate functions that scan the column and return a single value.


main.sql

```sql
-- Find the maximum salary
SELECT MAX(salary) AS max_salary
FROM employee;

-- Find the minimum salary
SELECT MIN(salary) AS min_salary
FROM employee;
```

Write a query to count records by group.

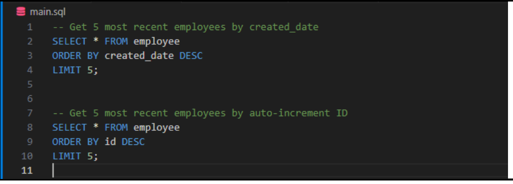
In SQL, to count records by group, you use the COUNT() aggregate function together with GROUP BY. This allows you to count how many rows belong to each category or group in a column.


main.sql

```sql
-- Count departments with more than 5 employees
SELECT department_id, COUNT(*) AS num_employees
FROM employee
GROUP BY department_id
HAVING COUNT(*) > 5;
```

How do you retrieve the 5 most recent records from a table?

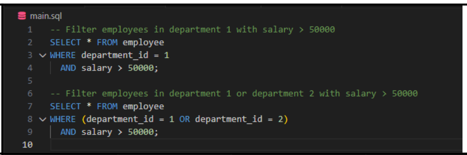
In SQL, to retrieve the most recent records, you typically need a column that indicates order, like a date, timestamp, or an auto-incrementing ID. You use ORDER BY to sort the records in descending order and LIMIT to restrict the number of rows returned.


main.sql

```sql
-- Get 5 most recent employees by created_date
SELECT * FROM employee
ORDER BY created_date DESC
LIMIT 5;


-- Get 5 most recent employees by auto-increment ID
SELECT * FROM employee
ORDER BY id DESC
LIMIT 5;
```

Write a query to filter data based on multiple conditions.

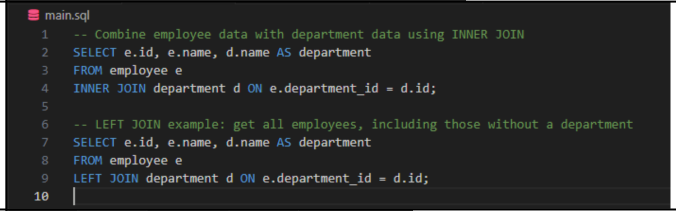
In SQL, to filter data based on multiple conditions, you use the WHERE clause with AND, OR, and parentheses () to combine conditions logically. This allows you to precisely select rows that meet complex criteria.



main.sql
```sql
-- Filter employees in department 1 with salary > 50000
SELECT * FROM employee
WHERE department_id = 1
  AND salary > 50000;

-- Filter employees in department 1 or department 2 with salary > 50000
SELECT * FROM employee
WHERE (department_id = 1 OR department_id = 2)
  AND salary > 50000;
```

How do you combine data from multiple tables?

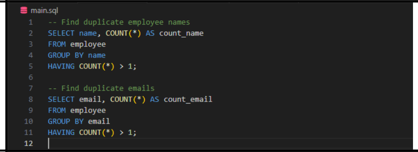
In SQL, to combine data from multiple tables, you use JOIN clauses. The most common types are INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN. JOINs allow you to match rows from one table with rows from another table based on a related column, usually a primary key and foreign key relationship.



main.sql
```sql
-- Combine employee data with department data using INNER JOIN
SELECT e.id, e.name, d.name AS department
FROM employee e
INNER JOIN department d ON e.department_id = d.id;

-- LEFT JOIN example: get all employees, including those without a department
SELECT e.id, e.name, d.name AS department
FROM employee e
LEFT JOIN department d ON e.department_id = d.id;
```

Write a query to find records that appear more than once.

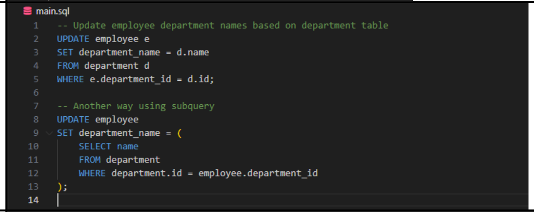
In SQL, to find records that appear more than once, you use GROUP BY on the column(s) you want to check and HAVING COUNT(*) > 1 to filter groups with duplicates.



main.sql
```sql
-- Find duplicate employee names
SELECT name, COUNT(*) AS count_name
FROM employee
GROUP BY name
HAVING COUNT(*) > 1;

-- Find duplicate emails
SELECT email, COUNT(*) AS count_email
FROM employee
GROUP BY email
HAVING COUNT(*) > 1;
```

Write a query to update data in one table based on data from another table.

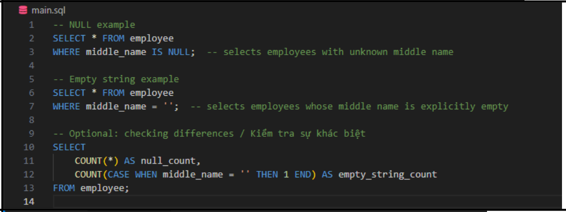
In SQL, to update data in one table based on data from another table, you use the UPDATE statement combined with a JOIN (or a subquery) to match rows between the tables. This lets you change values in one table according to corresponding values in another table.



main.sql
```sql
-- Update employee department names based on department table
UPDATE employee e
SET department_name = d.name
FROM department d
WHERE e.department_id = d.id;

-- Another way using subquery
UPDATE employee
SET department_name = (
    SELECT name
    FROM department
    WHERE department.id = employee.department_id
);
```

What is the difference between NULL and an empty string?

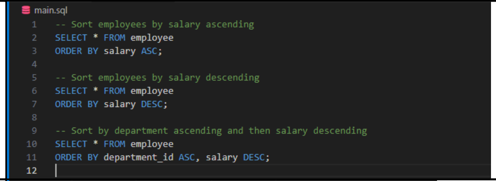
In SQL, NULL represents the absence of a value — it means the value is unknown or missing. An empty string (''), on the other hand, is a known value that just happens to have no characters. NULL and empty string are not the same, and comparisons with NULL require special handling using IS NULL or IS NOT NULL.



main.sql
```sql
-- NULL example
SELECT * FROM employee
WHERE middle_name IS NULL;  -- selects employees with unknown middle name

-- Empty string example
SELECT * FROM employee
WHERE middle_name = '';  -- selects employees whose middle name is explicitly empty

-- Optional: checking differences / Kiểm tra sự khác biệt
SELECT
    COUNT(*) AS null_count,
    COUNT(CASE WHEN middle_name = '' THEN 1 END) AS empty_string_count
FROM employee;
```

How do you sort query results in ascending and descending order?

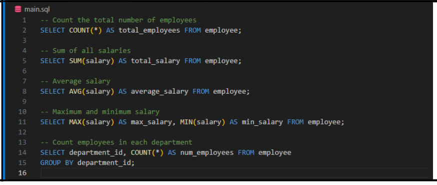
In SQL, to sort query results, you use the ORDER BY clause. By default, ORDER BY sorts in ascending order (ASC), and you can explicitly use DESC for descending order. You can also sort by multiple columns, each with its own order.



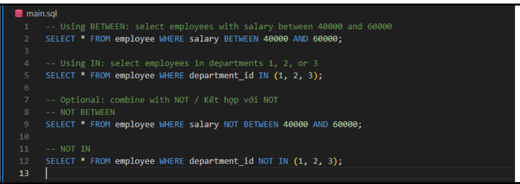
main.sql
```sql
-- Sort employees by salary ascending
SELECT * FROM employee
ORDER BY salary ASC;

-- Sort employees by salary descending
SELECT * FROM employee
ORDER BY salary DESC;

-- Sort by department ascending and then salary descending
SELECT * FROM employee
ORDER BY department_id ASC, salary DESC;
```

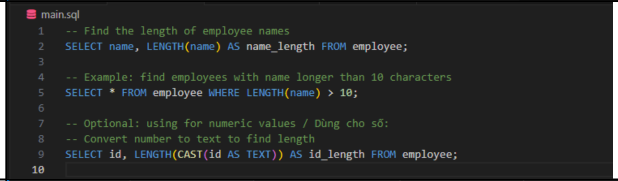
What are aggregate functions in SQL? Give examples.

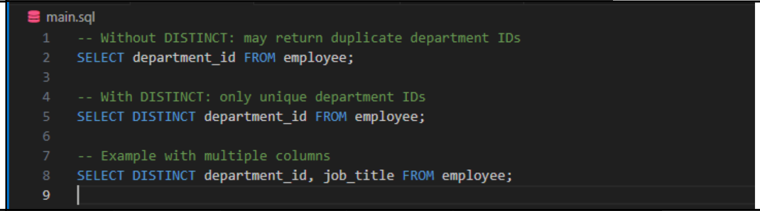
In SQL, aggregate functions are functions that perform calculations on a set of values and return a single summary value. They are often used with GROUP BY to summarize data by groups. Common aggregate functions include COUNT(), SUM(), AVG(), MAX(), MIN().

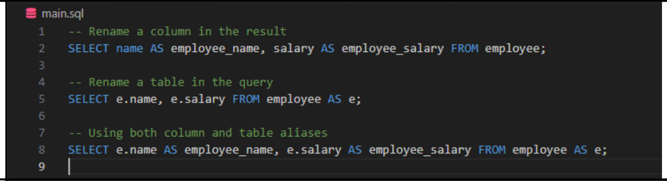


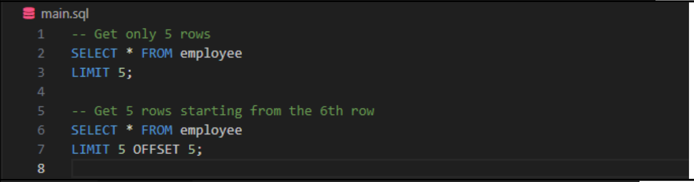
main.sql
```sql
-- Count the total number of employees
SELECT COUNT(*) AS total_employees FROM employee;

-- Sum of all salaries
SELECT SUM(salary) AS total_salary FROM employee;

-- Average salary
SELECT AVG(salary) AS average_salary FROM employee;

-- Maximum and minimum salary
SELECT MAX(salary) AS max_salary, MIN(salary) AS min_salary FROM employee;

-- Count employees in each department
SELECT department_id, COUNT(*) AS num_employees FROM employee
GROUP BY department_id;
```

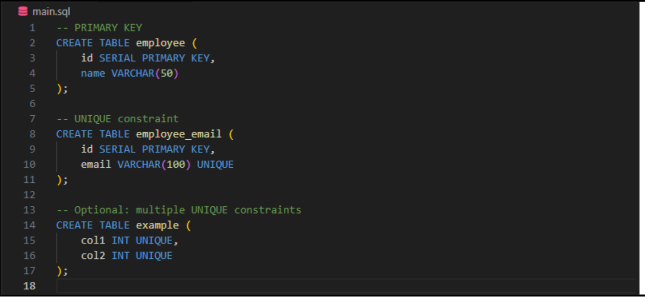
	What is the difference between BETWEEN and IN operators?
	
	In SQL, BETWEEN is used to filter values within a range, including the boundary values. IN is used to filter values that match any value in a list. BETWEEN is best for continuous ranges, while IN is best for discrete sets of values.
		
		main.sql
		`-- Using BETWEEN: select employees with salary between 40000 and 60000`
		`SELECT * FROM employee WHERE salary BETWEEN 40000 AND 60000;`
		
		`-- Using IN: select employees in departments 1, 2, or 3`
		`SELECT * FROM employee WHERE department_id IN (1, 2, 3);`
		
		`-- Optional: combine with NOT / Kết hợp với NOT`
		`-- NOT BETWEEN`
		`SELECT * FROM employee WHERE salary NOT BETWEEN 40000 AND 60000;`
		
		`-- NOT IN`
		`SELECT * FROM employee WHERE department_id NOT IN (1, 2, 3);`

	How do you find the length of a string or number of characters in a column?
	
	In SQL, to find the length of a string or the number of characters in a column, you use the LENGTH() function in PostgreSQL. This function returns the total number of characters, including spaces.
		
		main.sql
		`-- Find the length of employee names`
		`SELECT name, LENGTH(name) AS name_length FROM employee;`
		
		`-- Example: find employees with name longer than 10 characters`
		`SELECT * FROM employee WHERE LENGTH(name) > 10;`
		
		`-- Optional: using for numeric values / Dùng cho số:`
		`-- Convert number to text to find length`
		`SELECT id, LENGTH(CAST(id AS TEXT)) AS id_length FROM employee;`

	What is the difference between DISTINCT and no DISTINCT in a SELECT statement?
	
	In SQL, DISTINCT is used in a SELECT statement to return only unique values, removing duplicates from the result set. Without DISTINCT, the query returns all matching rows, including duplicates. Use DISTINCT when you want to avoid repeated values in your results.
		
		main.sql
		`-- Without DISTINCT: may return duplicate department IDs`
		`SELECT department_id FROM employee;`
		
		`-- With DISTINCT: only unique department IDs`
		`SELECT DISTINCT department_id FROM employee;`
		
		`-- Example with multiple columns`
		`SELECT DISTINCT department_id, job_title FROM employee;`

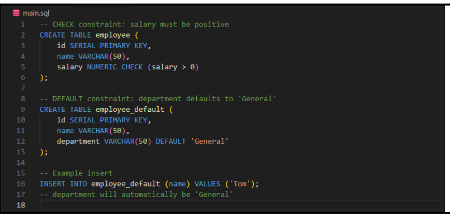
	How do you rename a column or table in a query result?
	
	In SQL, to rename a column or table in the query result, you use aliases with the AS keyword. Aliases do not change the actual table or column names in the database; they only rename them in the result set for readability.
		
		main.sql
		`-- Rename a column in the result`
		`SELECT name AS employee_name, salary AS employee_salary FROM employee;`
		
		`-- Rename a table in the query`
		`SELECT e.name, e.salary FROM employee AS e;`
		
		`-- Using both column and table aliases`
		`SELECT e.name AS employee_name, e.salary AS employee_salary FROM employee AS e;`

	How do you limit the number of rows returned in a query?
	
	In SQL, to limit the number of rows returned by a query, you use the LIMIT clause. This allows you to retrieve only a specific number of rows instead of the entire result set. In PostgreSQL, you can also combine it with OFFSET to skip a certain number of rows.
		
		main.sql
		`-- Get only 5 rows`
		`SELECT * FROM employee`
		`LIMIT 5;`
		
		`-- Get 5 rows starting from the 6th row`
		`SELECT * FROM employee`
		`LIMIT 5 OFFSET 5;`

	What is the difference between a PRIMARY KEY and a UNIQUE constraint?
	
	In SQL, both PRIMARY KEY and UNIQUE constraints enforce uniqueness for a column or a set of columns. The main differences are: a PRIMARY KEY uniquely identifies each row in a table and cannot accept NULL values, and a table can have only one primary key. A UNIQUE constraint also ensures uniqueness but can accept one NULL value per column in most databases, and a table can have multiple UNIQUE constraints.
		
		main.sql
		`-- PRIMARY KEY`
		`CREATE TABLE employee (`
		`    id SERIAL PRIMARY KEY,`
		`    name VARCHAR(50)`
		`);`
		
		`-- UNIQUE constraint`
		`CREATE TABLE employee_email (`
		`    id SERIAL PRIMARY KEY,`
		`    email VARCHAR(100) UNIQUE`
		`);`
		
		`-- Optional: multiple UNIQUE constraints`
		`CREATE TABLE example (`
		`    col1 INT UNIQUE,`
		`    col2 INT UNIQUE`
		`);`

What is the difference between CHECK and DEFAULT constraints?

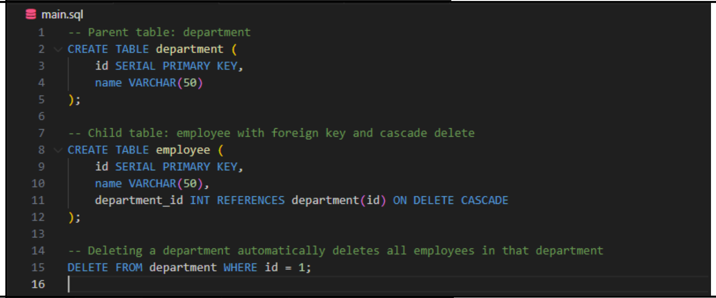
In SQL, CHECK and DEFAULT constraints serve different purposes. A CHECK constraint enforces a rule on the values that can be inserted into a column, ensuring data validity. A DEFAULT constraint automatically assigns a default value to a column when no value is provided during insertion.



```sql
-- CHECK constraint: salary must be positive
CREATE TABLE employee (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    salary NUMERIC CHECK (salary > 0)
);

-- DEFAULT constraint: department defaults to 'General'
CREATE TABLE employee_default (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    department VARCHAR(50) DEFAULT 'General'
);

-- Example insert
INSERT INTO employee_default (name) VALUES ('Tom');
-- department will automatically be 'General'
```

What is a cascade delete? When should it be used?

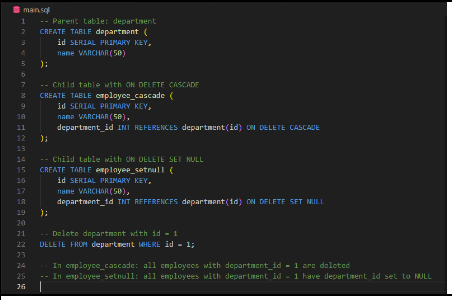
In SQL, a cascade delete is a foreign key option that automatically deletes child rows when the corresponding parent row is deleted. This ensures referential integrity without leaving orphaned records in related tables.
You should use cascade delete when you want related records to be automatically removed together with the parent, such as deleting an order and all its order items. Be careful using it, because it can delete large amounts of data unintentionally if not planned properly.



```sql
-- Parent table: department
CREATE TABLE department (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50)
);

-- Child table: employee with foreign key and cascade delete
CREATE TABLE employee (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    department_id INT REFERENCES department(id) ON DELETE CASCADE
);

-- Deleting a department automatically deletes all employees in that department
DELETE FROM department WHERE id = 1;
```

What is the difference between ON DELETE CASCADE and ON DELETE SET NULL?

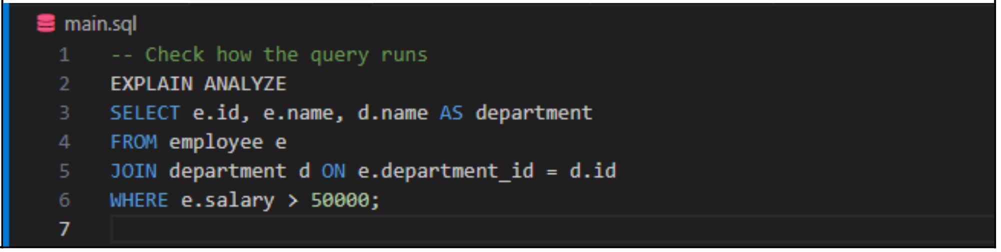
In SQL, both ON DELETE CASCADE and ON DELETE SET NULL are foreign key actions triggered when a parent row is deleted. ON DELETE CASCADE automatically deletes all child rows that reference the parent, ensuring no orphaned records remain. ON DELETE SET NULL does not delete child rows; instead, it sets the foreign key column in the child table to NULL, keeping the child rows but removing the reference to the deleted parent.



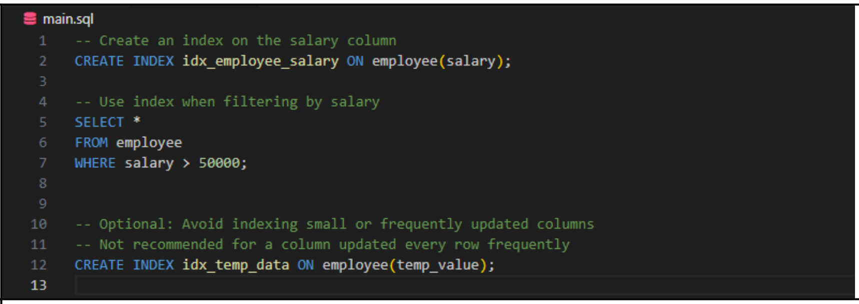
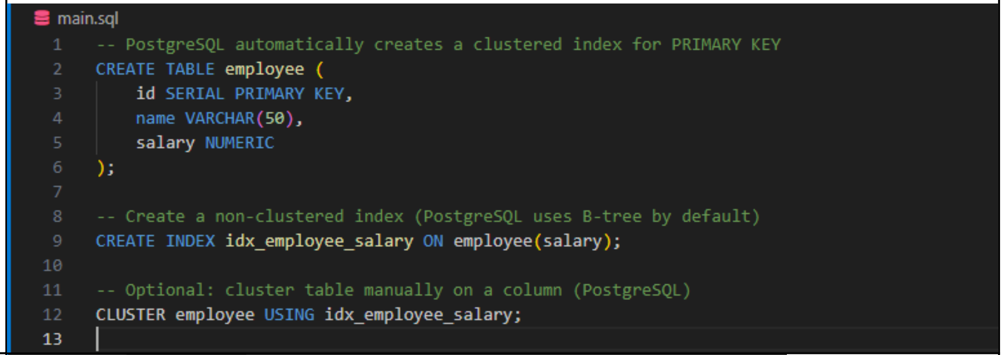
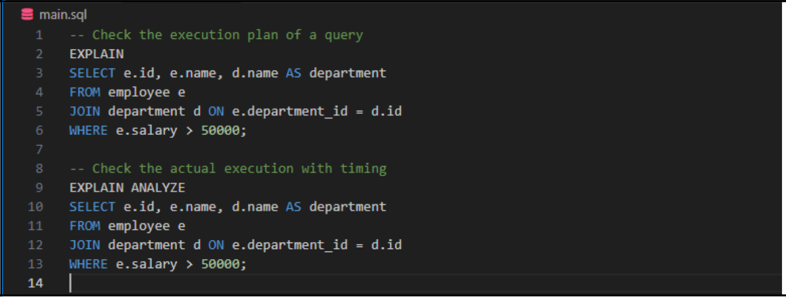
```sql
-- Parent table: department
CREATE TABLE department (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50)
);

-- Child table with ON DELETE CASCADE
CREATE TABLE employee_cascade (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    department_id INT REFERENCES department(id) ON DELETE CASCADE
);

-- Child table with ON DELETE SET NULL
CREATE TABLE employee_setnull (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    department_id INT REFERENCES department(id) ON DELETE SET NULL
);

-- Delete department with id = 1
DELETE FROM department WHERE id = 1;

-- In employee_cascade: all employees with department_id = 1 are deleted
-- In employee_setnull: all employees with department_id = 1 have department_id set to NULL
```

How can you optimize a slow query?

In SQL, to optimize a slow query, you can use several strategies. First, analyze and add indexes on columns used in WHERE, JOIN, ORDER BY, or GROUP BY clauses to speed up lookups. Second, **avoid SELECT *** and only retrieve the columns you need. Third, rewrite complex queries using joins, subqueries, or CTEs more efficiently. Fourth, check execution plans with EXPLAIN to find bottlenecks. Finally, consider caching, partitioning, or denormalization if appropriate.
Optional tips:
- Ensure indexes are used on filtering and joining columns.
- Avoid functions on indexed columns in WHERE clauses (e.g., LENGTH (name) > 5 may bypass index).
- Limit data scanned with proper WHERE conditions.



```sql
-- Check how the query runs
EXPLAIN ANALYZE
SELECT e.id, e.name, d.name AS department
FROM employee e
JOIN department d ON e.department_id = d.id
WHERE e.salary > 50000;
```

	When should you use an Index? When should you avoid using one? In SQL, you should use an index when you have frequent queries that filter, sort, or join on a column, especially for large tables. Indexes can drastically speed up SELECT queries by allowing the database to find rows without scanning the entire table. You should avoid using an index when: the table is small (full scan is fast enough), the column is updated very frequently (indexes slow down INSERT/UPDATE/DELETE), or there are too many indexes that increase storage and maintenance overhead.	
	What is the difference between a clustered and a non-clustered index? In SQL, a clustered index determines the physical order of data in a table. There can be only one clustered index per table, and the table rows are stored in the order of the clustered index. A non-clustered index does not change the physical order of the table; it creates a separate structure pointing to the actual data, and a table can have multiple non-clustered indexes. Clustered indexes are good for range queries, while non-clustered indexes are good for lookups on specific columns. Key points: - Clustered index = physical order of rows. - Non-clustered index = separate structure pointing to rows. - One table → 1 clustered index, many non-clustered indexes possible.	
	What is an Explain Plan? What is it used for? In SQL, an Explain Plan is a tool that shows how the database executes a query, including the steps, order of operations, and indexes used. It is mainly used to analyze and optimize queries by identifying bottlenecks or inefficient operations. Optional: - Helps see which indexes are used - Helps estimate query cost - Helps identify full table scans	

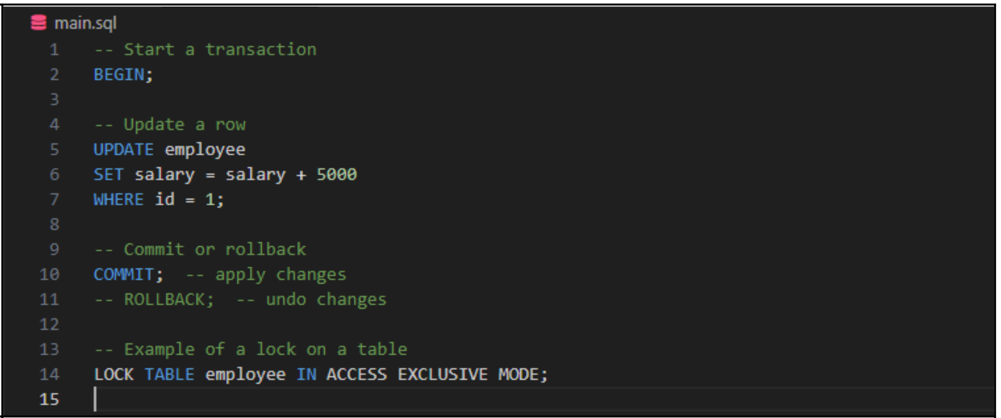
main.sql
```sql
-- Create an index on the salary column
CREATE INDEX idx_employee_salary ON employee(salary);

-- Use index when filtering by salary
SELECT *
FROM employee
WHERE salary > 50000;


-- Optional: Avoid indexing small or frequently updated columns
-- Not recommended for a column updated every row frequently
CREATE INDEX idx_temp_data ON employee(temp_value);
```

main.sql
```sql
-- PostgreSQL automatically creates a clustered index for PRIMARY KEY
CREATE TABLE employee (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    salary NUMERIC
);

-- Create a non-clustered index (PostgreSQL uses B-tree by default)
CREATE INDEX idx_employee_salary ON employee(salary);

-- Optional: cluster table manually on a column (PostgreSQL)
CLUSTER employee USING idx_employee_salary;
```

main.sql
```sql
-- Check the execution plan of a query
EXPLAIN
SELECT e.id, e.name, d.name AS department
FROM employee e
JOIN department d ON e.department_id = d.id
WHERE e.salary > 50000;

-- Check the actual execution with timing
EXPLAIN ANALYZE
SELECT e.id, e.name, d.name AS department
FROM employee e
JOIN department d ON e.department_id = d.id
WHERE e.salary > 50000;
```

What is the difference between a transaction and a lock?

In SQL, a transaction is a logical unit of work that consists of one or more SQL statements executed as a single unit. Transactions follow the ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure data integrity. A lock is a mechanism used by the database to control concurrent access to data, preventing conflicts like dirty reads or lost updates. While transactions define what work should be done atomically, locks are the tools the database uses internally to manage concurrency during transactions.
Quick notes:
- Transaction = logical unit of work
- Lock = mechanism to control concurrent access
- Transactions use locks internally to protect data

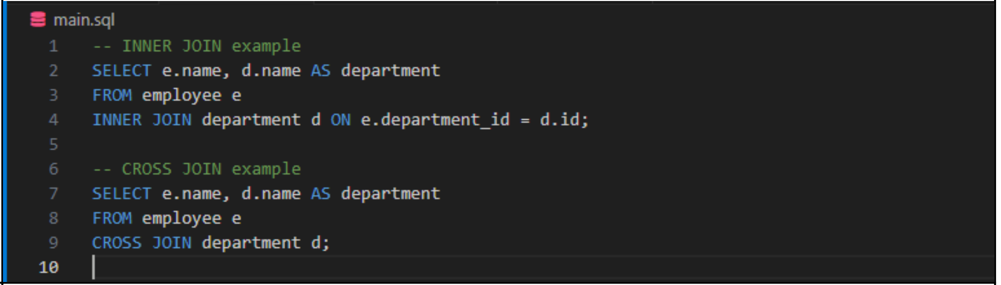

main.sql

```sql
-- Start a transaction
BEGIN;

-- Update a row
UPDATE employee
SET salary = salary + 5000
WHERE id = 1;

-- Commit or rollback
COMMIT;  -- apply changes
-- ROLLBACK;  -- undo changes

-- Example of a lock on a table
LOCK TABLE employee IN ACCESS EXCLUSIVE MODE;
```

Explain the difference between INNER JOIN and CROSS JOIN.

An INNER JOIN returns rows from two tables where there is a match based on a specified condition, usually involving primary and foreign keys. Only the rows satisfying the condition appear in the result.
A CROSS JOIN, on the other hand, returns the Cartesian product of the two tables, meaning every row from the first table is combined with every row from the second table, regardless of any condition. CROSS JOIN can produce a very large number of rows if both tables are big.

Quick notes:
- INNER JOIN = match rows based on condition
- CROSS JOIN = all combinations of rows
- Use CROSS JOIN carefully, may create large result sets

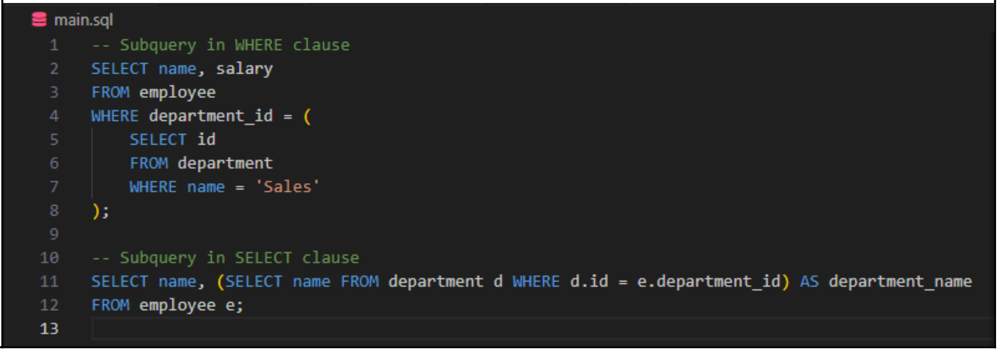

main.sql

```sql
-- INNER JOIN example
SELECT e.name, d.name AS department
FROM employee e
INNER JOIN department d ON e.department_id = d.id;

-- CROSS JOIN example
SELECT e.name, d.name AS department
FROM employee e
CROSS JOIN department d;
```

What is a subquery and when would you use one?

A subquery is a query nested inside another query, typically inside a SELECT, INSERT, UPDATE, or DELETE statement. Subqueries are used when you need to retrieve a value or set of values from one table to use in another query. They help break complex queries into manageable parts and can be used for filtering, calculating aggregates, or checking existence.

Quick notes:
Subquery = a query inside another query
Can return single value, list, or table
Useful for filtering, aggregating, or conditional checks

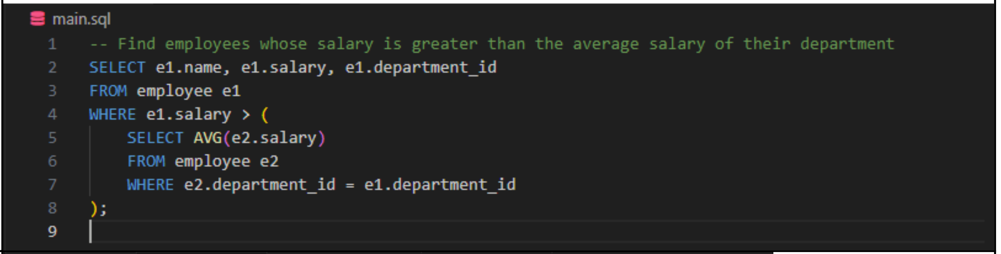

main.sql

```sql
-- Subquery in WHERE clause
SELECT name, salary
FROM employee
WHERE department_id = (
    SELECT id
    FROM department
    WHERE name = 'Sales'
);

-- Subquery in SELECT clause
SELECT name, (SELECT name FROM department d WHERE d.id = e.department_id) AS department_name
FROM employee e;
```

What is a correlated subquery? Give an example.

A correlated subquery is a subquery that depends on a column from the outer query. It is executed once for each row of the outer query, unlike a regular subquery which runs independently. Correlated subqueries are useful for row-by-row comparisons or calculations that reference the outer query's data.

Quick notes:
Correlated subquery references outer query columns
Executed once per row of the outer query
Useful for comparisons and conditional calculations



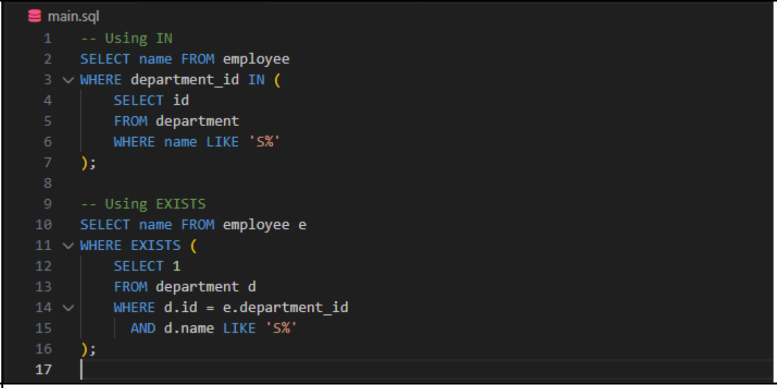
main.sql

```sql
-- Find employees whose salary is greater than the average salary of their department
SELECT e1.name, e1.salary, e1.department_id
FROM employee e1
WHERE e1.salary > (
    SELECT AVG(e2.salary)
    FROM employee e2
    WHERE e2.department_id = e1.department_id
);

```

Explain the difference between EXISTS and IN.

In SQL, both EXISTS and IN are used to filter rows based on another query, but they work differently. EXISTS checks if a subquery returns any rows and returns TRUE or FALSE for each row of the outer query; it is often more efficient with large datasets. IN checks if a column's value matches any value in a list or subquery. EXISTS is typically faster when the subquery returns many rows, while IN is more readable for small lists.

Quick notes:
EXISTS = TRUE/FALSE based on subquery result
IN = matches values in list or subquery
EXISTS often faster with large subqueries

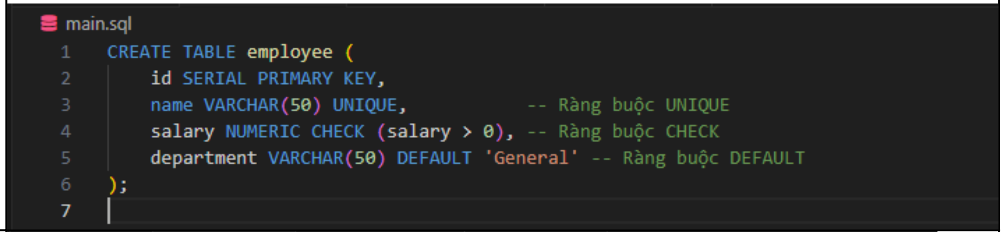


main.sql

```sql
-- Using IN
SELECT name FROM employee
WHERE department_id IN (
    SELECT id
    FROM department
    WHERE name LIKE 'S%'
);

-- Using EXISTS
SELECT name FROM employee e
WHERE EXISTS (
    SELECT 1
    FROM department d
    WHERE d.id = e.department_id
      AND d.name LIKE 'S%'
);

```

How can you enforce data integrity using constraints other than PRIMARY KEY and FOREIGN KEY?

In SQL, besides PRIMARY KEY and FOREIGN KEY, you can enforce data integrity using several other constraints: UNIQUE, CHECK, and DEFAULT. UNIQUE ensures that a column or combination of columns has distinct values. CHECK enforces a rule on column values, such as limiting salaries to be positive. DEFAULT automatically assigns a default value to a column if none is provided. These constraints help maintain valid, consistent, and predictable data in the database.

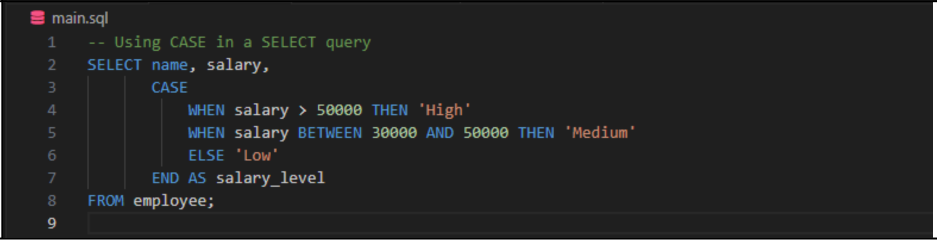


main.sql

```sql
CREATE TABLE employee (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE,             -- Ràng buộc UNIQUE
    salary NUMERIC CHECK (salary > 0), -- Ràng buộc CHECK
    department VARCHAR(50) DEFAULT 'General' -- Ràng buộc DEFAULT
);

```

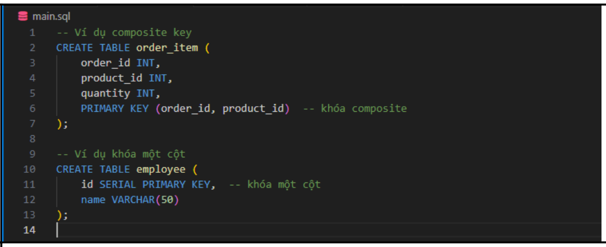
What is the difference between CASE and IF in SQL?

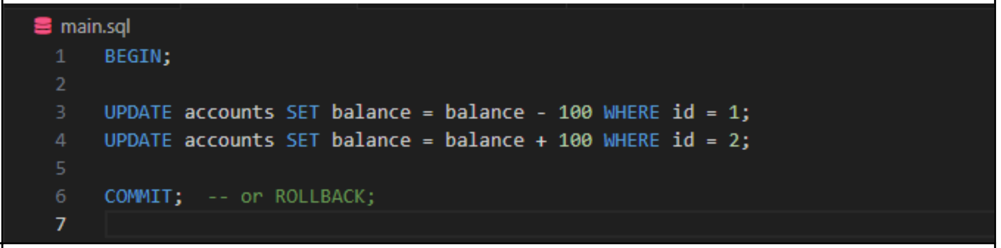
In SQL, CASE and IF are used for conditional logic, but they are different. CASE is part of the SQL standard and works in SELECT, UPDATE, ORDER BY, and other statements; it can evaluate multiple conditions and return different results based on them. IF is usually a procedural statement used in database programming languages like PL/pgSQL, T-SQL, or inside stored procedures, not in standard SELECT queries.

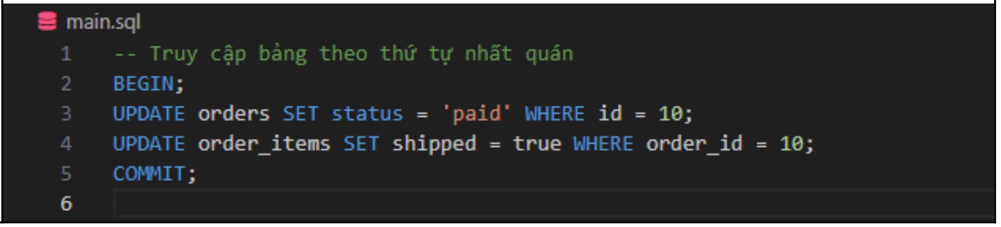


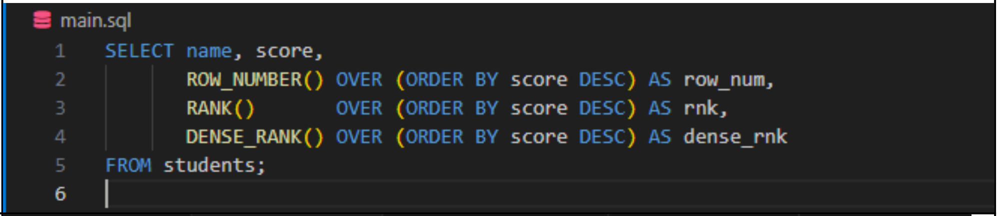
main.sql

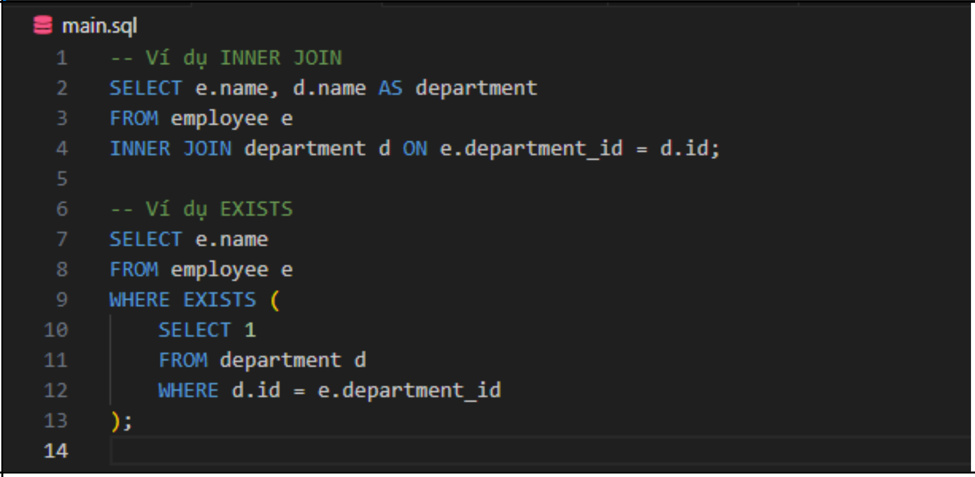
```sql
-- Using CASE in a SELECT query
SELECT name, salary,
    CASE
        WHEN salary > 50000 THEN 'High'
        WHEN salary BETWEEN 30000 AND 50000 THEN 'Medium'
        ELSE 'Low'
    END AS salary_level
FROM employee;

```

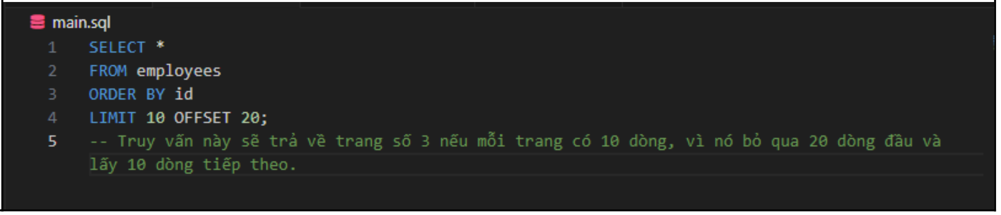
	What is a composite key? How is it different from a single-column key?
	
	A composite key is a primary key made up of two or more columns in a table, where the combination of values in these columns uniquely identifies each row. A single-column key (or simple primary key) uses only one column to uniquely identify rows. Composite keys are used when no single column can guarantee uniqueness, but a combination of columns can.
		 main.sql
		`-- Ví dụ composite key`
		`CREATE TABLE order_item (`
		`    order_id INT,`
		`    product_id INT,`
		`    quantity INT,`
		`    PRIMARY KEY (order_id, product_id)  -- khóa composite`
		`);`
		
		`-- Ví dụ khóa một cột`
		`CREATE TABLE employee (`
		`    id SERIAL PRIMARY KEY,  -- khóa một cột`
		`    name VARCHAR(50)`
		`);`

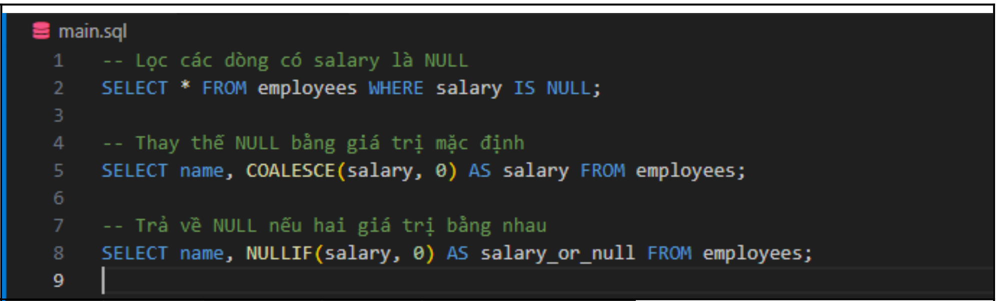
	What are transactions in SQL, and what are the ACID properties?
	
	In SQL, a transaction is a sequence of operations that are treated as a single logical unit of work, meaning either all operations succeed or none of them take effect. Transactions ensure data consistency, especially in cases of errors or system failures. The reliability of transactions is defined by the ACID properties:
	- Atomicity: all operations in the transaction must succeed or all must be rolled back.
	- Consistency: the database must move from one valid state to another.
	- Isolation: transactions should not interfere with each other; intermediate states are not visible.
	- Durability: once a transaction is committed, the changes are permanently saved even if the system crashes.
		 main.sql
		`BEGIN;`
		
		`UPDATE accounts SET balance = balance - 100 WHERE id = 1;`
		`UPDATE accounts SET balance = balance + 100 WHERE id = 2;`
		
		`COMMIT;  -- or ROLLBACK;`

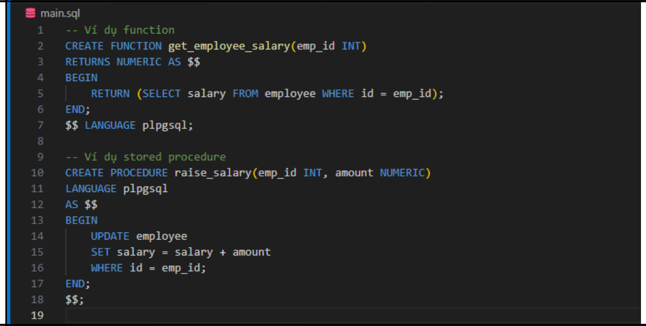
	How can you prevent deadlocks in a database?
	
	To prevent deadlocks in a database, you should follow good transaction and locking practices. The most important strategy is keeping transactions short and consistent so locks are held for minimal time. Always access tables and rows in a consistent order across all parts of the application to avoid circular waits. Use proper indexing to reduce full table scans that can escalate locks. Avoid unnecessary locking by selecting only the rows you need, and use lower isolation levels when possible to reduce locking intensity. Finally, monitor queries and use tools like EXPLAIN to detect slow operations that may cause lock contention.
		 main.sql
		`-- Truy cập bảng theo thứ tự nhất quán`
		`BEGIN;`
		`UPDATE orders SET status = 'paid' WHERE id = 10;`
		`UPDATE order_items SET shipped = true WHERE order_id = 10;`
		`COMMIT;`

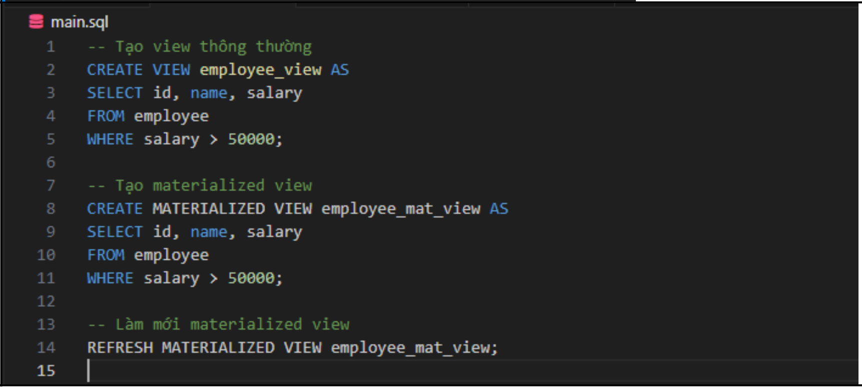
	What is the difference between ROW_NUMBER(), RANK(), and DENSE_RANK()?
	
	ROW_NUMBER(), RANK(), and DENSE_RANK() are window functions used to assign ordering numbers to rows, but they behave differently when duplicates appear. ROW_NUMBER() assigns a unique sequential number to each row, even if values are tied. RANK() assigns the same rank to tied rows but skips numbers afterward. DENSE_RANK() also assigns the same rank to tied rows but does not skip numbers, making the sequence continuous.
	
	Quick notes:
	- ROW_NUMBER(): no ties, always 1,2,3,…
	- RANK(): ties share rank, numbers are skipped
	- DENSE_RANK(): ties share rank, no skipping
		
		main.sql
		`SELECT name, score,`
		`       ROW_NUMBER() OVER (ORDER BY score DESC) AS row_num,`
		`       RANK()       OVER (ORDER BY score DESC) AS rnk,`
		`       DENSE_RANK() OVER (ORDER BY score DESC) AS dense_rnk`
		`FROM students;`

	What is the difference between INNER JOIN and EXISTS?
	
	The difference between INNER JOIN and EXISTS lies in how they filter data and how they perform. INNER JOIN combines rows from two tables based on matching values; it returns the actual columns from both tables. It is best when you need data from both sides. EXISTS, on the other hand, checks whether a subquery returns at least one row; it returns TRUE or FALSE for each row of the outer query and does not return columns from the subquery. EXISTS is usually more efficient when you only need to check the existence of related rows, especially when the subquery can stop early once a match is found.
	
	Quick notes:
	- INNER JOIN returns combined data from two tables.
	- EXISTS only checks for existence, no need to return columns.
	- EXISTS can be faster when the subquery has many rows.
		
		main.sql
		`-- Ví dụ INNER JOIN`
		`SELECT e.name, d.name AS department`
		`FROM employee e`
		`INNER JOIN department d ON e.department_id = d.id;`
		
		`-- Ví dụ EXISTS`
		`SELECT e.name`
		`FROM employee e`
		`WHERE EXISTS (`
		`    SELECT 1`
		`    FROM department d`
		`    WHERE d.id = e.department_id`
		`);`

	Write a query to paginate data.
	
	A common way to paginate data in SQL is to use the keywords LIMIT and OFFSET. LIMIT controls how many rows you want to return, while OFFSET tells the query how many rows to skip before starting to return results. This is useful when you want to display data page by page, for example in a web application. For page-based pagination, you usually calculate the offset as: (page_number - 1) * page_size.
		
		main.sql
		`SELECT *`
		`FROM employees`
		`ORDER BY id`
		`LIMIT 10 OFFSET 20;`
		`-- Truy vấn này sẽ trả về trang số 3 nếu mỗi trang có 10 dòng, vì nó bỏ qua 20 dòng đầu và lấy 10 dòng tiếp theo.`

	How do you handle NULL values in SQL?
	
	In SQL, NULL represents missing or unknown data. Handling NULLs properly is important to avoid errors or incorrect results. You can use:
	- IS NULL / IS NOT NULL to filter rows with or without NULL values.
	- COALESCE() to replace NULL with a default value.
	- NULLIF() to return NULL if two expressions are equal.
	- Aggregate functions like COUNT(column) ignore NULLs, but COUNT(*) counts all rows.
	Handling NULLs carefully ensures correct calculations and filtering.
		
		main.sql
		`-- Lọc các dòng có salary là NULL`
		`SELECT * FROM employees WHERE salary IS NULL;`
		
		`-- Thay thế NULL bằng giá trị mặc định`
		`SELECT name, COALESCE(salary, 0) AS salary FROM employees;`
		
		`-- Trả về NULL nếu hai giá trị bằng nhau`
		`SELECT name, NULLIF(salary, 0) AS salary_or_null FROM employees;`

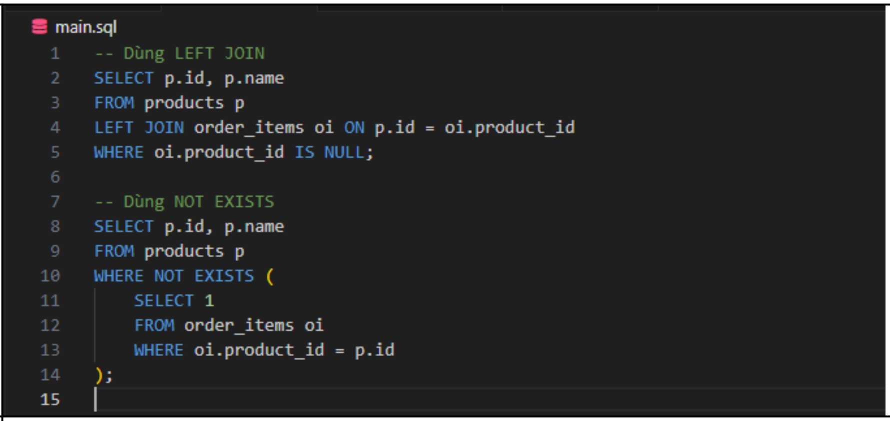
	What is the difference between a STORED PROCEDURE and a FUNCTION?
	
	In SQL, both stored procedures and functions are reusable blocks of code stored in the database, but they have key differences. A stored procedure can perform operations like INSERT, UPDATE, DELETE, or complex logic, and it does not have to return a value. It is called with the CALL statement. A function must return a value and can be used in SQL expressions, like in SELECT or WHERE clauses. Functions are generally used for calculations or transformations, while procedures are used for executing business logic or workflows.
		
		main.sql
		`-- Ví dụ function`
		`CREATE FUNCTION get_employee_salary(emp_id INT)`
		`RETURNS NUMERIC AS $$`
		`BEGIN`
		`    RETURN (SELECT salary FROM employee WHERE id = emp_id);`
		`END;`
		`$$ LANGUAGE plpgsql;`
		
		`-- Ví dụ stored procedure`
		`CREATE PROCEDURE raise_salary(emp_id INT, amount NUMERIC)`
		`LANGUAGE plpgsql`
		`AS $$`
		`BEGIN`
		`    UPDATE employee`
		`    SET salary = salary + amount`
		`    WHERE id = emp_id;`
		`END;`
		`$$;`

	What is the difference between a VIEW and a MATERIALIZED VIEW?
	
	A VIEW is a virtual table in SQL that is defined by a query. It does not store data physically, so each time you query it, the database executes the underlying query. A MATERIALIZED VIEW, on the other hand, stores the query results physically. This can improve performance for complex queries because the data is precomputed, but it may become stale if the underlying tables change, and may require manual refresh.
	
	Quick notes:
	- VIEW = virtual table, no physical storage
	- MATERIALIZED VIEW = stores results physically
	- Use materialized view for performance on heavy queries
		
		main.sql
		`-- Tạo view thông thường`
		`CREATE VIEW employee_view AS`
		`SELECT id, name, salary`
		`FROM employee`
		`WHERE salary > 50000;`
		
		`-- Tạo materialized view`
		`CREATE MATERIALIZED VIEW employee_mat_view AS`
		`SELECT id, name, salary`
		`FROM employee`
		`WHERE salary > 50000;`
		
		`-- Làm mới materialized view`
		`REFRESH MATERIALIZED VIEW employee_mat_view;`

How do you find products that have not been included in any orders?

To find products that have not been included in any orders, you can use either a LEFT JOIN with a NULL check or a NOT EXISTS subquery. The LEFT JOIN approach joins products with orders and checks for rows where the order does not exist. The NOT EXISTS approach checks for products where no matching order is found. Both methods work, but NOT EXISTS is often more readable for this case.

Quick notes:
- LEFT JOIN + NULL = find unmatched rows
- NOT EXISTS = check absence of related rows
- Both achieve the same result; choice depends on readability and query planner



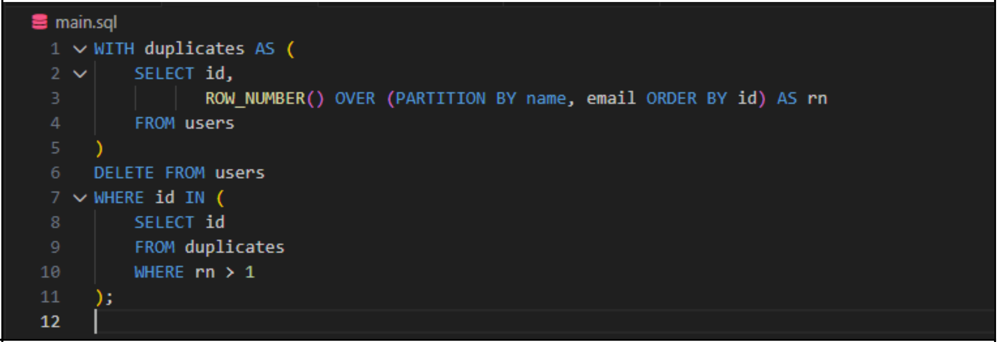
main.sql

```sql
-- Dùng LEFT JOIN
SELECT p.id, p.name
FROM products p
LEFT JOIN order_items oi ON p.id = oi.product_id
WHERE oi.product_id IS NULL;

-- Dùng NOT EXISTS
SELECT p.id, p.name
FROM products p
WHERE NOT EXISTS (
    SELECT 1
    FROM order_items oi
    WHERE oi.product_id = p.id
);
```

How do you delete all duplicate records but keep one unique record?

To delete duplicate records but keep one unique row, you can use a CTE (Common Table Expression) with the ROW_NUMBER() window function. The idea is to assign a unique sequential number to duplicates within each group of identical values and then delete rows where this number is greater than 1. This ensures that one row is kept while all duplicates are removed.

Explanation:
- PARTITION BY name, email groups duplicates based on these columns.
- ROW_NUMBER() assigns 1 to the first row in each group.
- DELETE removes rows where rn > 1, keeping only the first row.



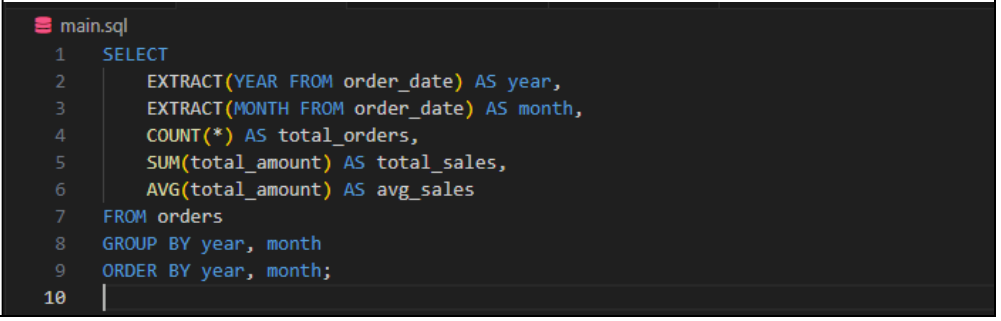
main.sql

```sql
WITH duplicates AS (
    SELECT id,
           ROW_NUMBER() OVER (PARTITION BY name, email ORDER BY id) AS rn
    FROM users
)
DELETE FROM users
WHERE id IN (
    SELECT id
    FROM duplicates
    WHERE rn > 1
);
```

Write a query to calculate sales statistics by month and year.

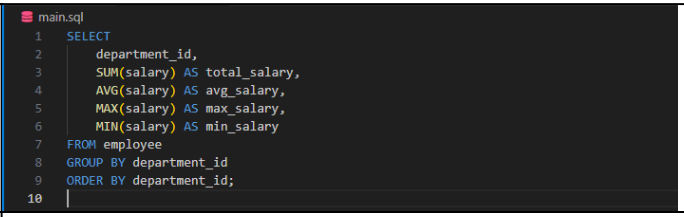
To calculate sales statistics by month and year, you can use the EXTRACT() function in PostgreSQL to extract the year and month from a date column, then group by these values. You can use aggregate functions like SUM() for total sales, COUNT() for the number of orders, and AVG() for average sales. This is useful for monthly or yearly reporting.

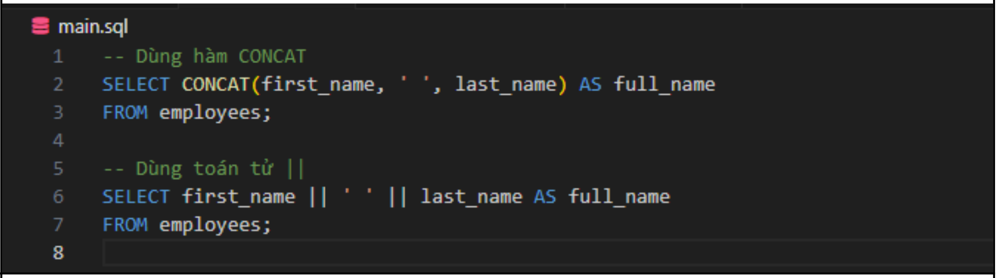
Quick notes:
EXTRACT(YEAR FROM ...) and EXTRACT(MONTH FROM ...) get year and month.
Aggregate functions summarize the data.
GROUP BY allows monthly/yearly aggregation.

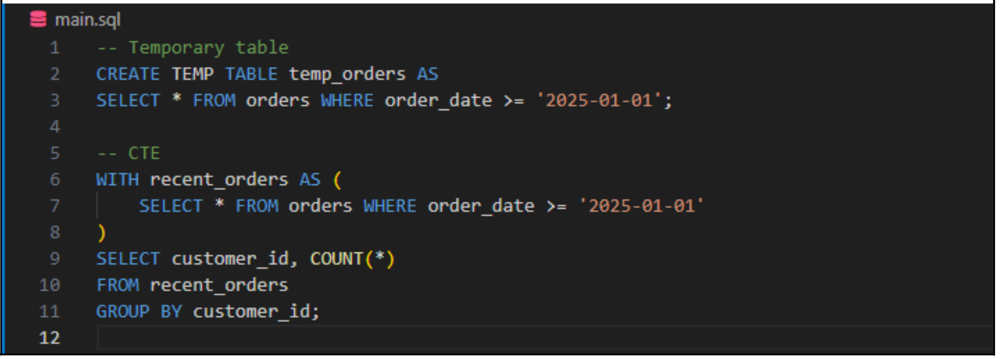


main.sql

```sql
SELECT
    EXTRACT(YEAR FROM order_date) AS year,
    EXTRACT(MONTH FROM order_date) AS month,
    COUNT(*) AS total_orders,
    SUM(total_amount) AS total_sales,
    AVG(total_amount) AS avg_sales
FROM orders
GROUP BY year, month
ORDER BY year, month;
```

	Write a query to calculate the total, average, maximum, and minimum for each group.
	
	To calculate total, average, maximum, and minimum for each group, you use SQL aggregate functions (SUM(), AVG(), MAX(), MIN()) together with a GROUP BY clause. This allows you to summarize data per group, for example per department, category, or any grouping column.
		
		main.sql
		`SELECT`
		`    department_id,`
		`    SUM(salary) AS total_salary,`
		`    AVG(salary) AS avg_salary,`
		`    MAX(salary) AS max_salary,`
		`    MIN(salary) AS min_salary`
		`FROM employee`
		`GROUP BY department_id`
		`ORDER BY department_id;`

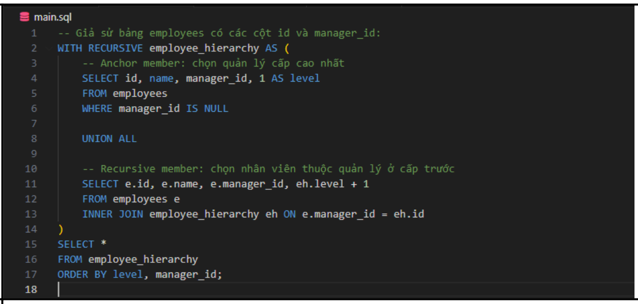
	How do you combine multiple columns into a single column?
	
	To combine multiple columns into a single column in SQL, you can use concatenation functions. In PostgreSQL, you can use the || operator or the CONCAT() function. This is useful when you want to merge, for example, first name and last name into a full name column. You can also include separators like spaces or commas.
	
	Quick notes:
	- CONCAT() = merges multiple columns/values
	- || = PostgreSQL operator for string concatenation
	- Include separators (space, comma) as needed
		
		main.sql
		`-- Dùng hàm CONCAT`
		`SELECT CONCAT(first_name, ' ', last_name) AS full_name`
		`FROM employees;`
		
		`-- Dùng toán tử ||`
		`SELECT first_name || ' ' || last_name AS full_name`
		`FROM employees;`

	Explain the differences between temporary tables, table variables, and CTEs (Common Table Expressions).
	
	In SQL, temporary tables, table variables, and CTEs (Common Table Expressions) are all ways to store intermediate results, but they differ in scope, lifetime, and usage.
	- Temporary tables (CREATE TEMP TABLE) are stored in the database like regular tables but are automatically dropped at the end of the session or optionally at the end of a transaction. They can have indexes, constraints, and statistics, and can be referenced multiple times in queries. They are useful for large datasets or multiple-query processing.
	- Table variables (e.g., in T-SQL DECLARE @table TABLE) are variables that store a table structure in memory. They have limited scope (usually the batch or procedure) and generally do not maintain statistics or allow indexes except primary keys. They are best for small datasets and lightweight temporary storage.
	- CTEs (Common Table Expressions) are temporary result sets defined using WITH. They exist only for the duration of a single query and are not physically stored. They are mainly used for readability, recursion, or simplifying complex queries.
		
		main.sql
		`-- Temporary table`
		`CREATE TEMP TABLE temp_orders AS`
		`SELECT * FROM orders WHERE order_date >= '2025-01-01';`
		
		`-- CTE`
		`WITH recent_orders AS (`
		`    SELECT * FROM orders WHERE order_date >= '2025-01-01'`
		`)`
		`SELECT customer_id, COUNT(*)`
		`FROM recent_orders`
		`GROUP BY customer_id;`

How do you implement recursive queries in SQL?

In SQL, you implement recursive queries using CTEs (Common Table Expressions) with the WITH RECURSIVE clause. Recursive queries are useful for hierarchical data, like organizational charts, bill-of-materials, or tree structures. A recursive CTE consists of two parts:
1. Anchor member: the base query that provides the starting row(s).
2. Recursive member: references the CTE itself to repeatedly fetch child or related rows.

The recursion stops when no new rows are returned.



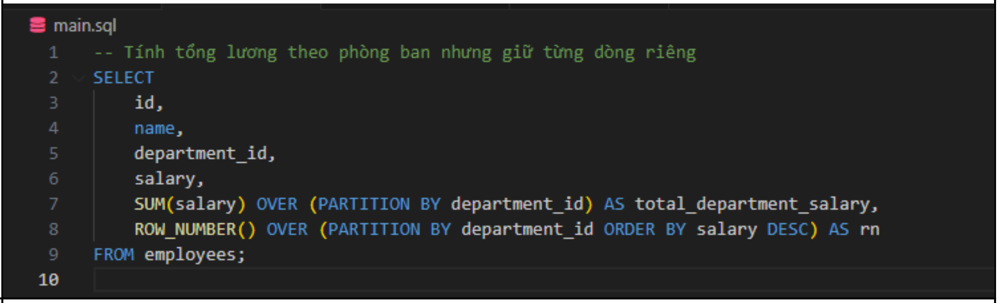
main.sql

```sql
-- Giả sử bảng employees có các cột id và manager_id:
WITH RECURSIVE employee_hierarchy AS (
    -- Anchor member: chọn quản lý cấp cao nhất
    SELECT id, name, manager_id, 1 AS level
    FROM employees
    WHERE manager_id IS NULL

    UNION ALL

    -- Recursive member: chọn nhân viên thuộc quản lý ở cấp trước
    SELECT e.id, e.name, e.manager_id, eh.level + 1
    FROM employees e
    INNER JOIN employee_hierarchy eh ON e.manager_id = eh.id
)
SELECT *
FROM employee_hierarchy
ORDER BY level, manager_id;
```

Windowing trong SQL là gì, và nó khác gì so với GROUP BY?

Windowing trong SQL là các hàm cửa sổ (window functions), dùng để thực hiện tính toán trên một tập hợp các dòng liên quan đến dòng hiện tại, không gộp các dòng thành một kết quả duy nhất. Khác với GROUP BY, gộp tất cả dòng thành một kết quả cho mỗi nhóm, window functions giữ nguyên số dòng gốc đồng thời cung cấp tổng hợp, xếp hạng, hoặc trung bình động trên "cửa sổ" các dòng.
Các hàm window phổ biến: ROW_NUMBER(), RANK(), SUM() OVER (...), AVG() OVER(...), LEAD()/LAG(). Câu lệnh OVER() xác định cửa sổ, có thể chia nhóm (PARTITION BY) và sắp xếp (ORDER BY).

Lưu ý nhanh:
GROUP BY = gộp các dòng thành một dòng mỗi nhóm
Window function = tính toán không giảm số dòng
Dùng cho xếp hạng, tổng lũy kế, trung bình động, v.v.



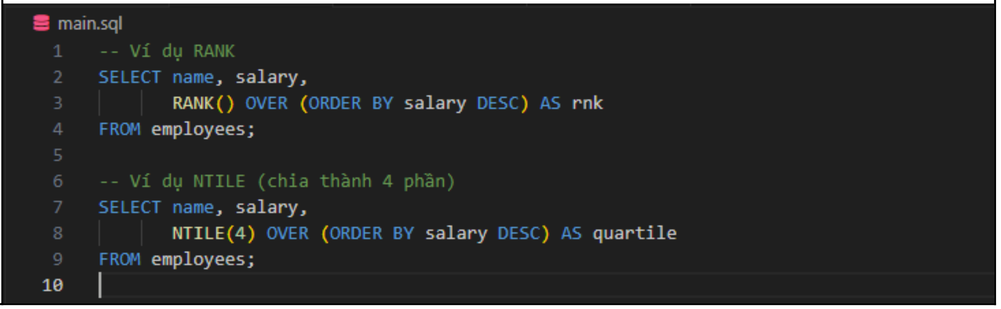
main.sql

```sql
-- Tính tổng lương theo phòng ban nhưng giữ từng dòng riêng
SELECT
    id,
    name,
    department_id,
    salary,
    SUM(salary) OVER (PARTITION BY department_id) AS total_department_salary,
    ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn
FROM employees;
```

Explain the difference between RANK() and NTILE().

Both RANK() and NTILE() are window functions in SQL used to assign numbers to rows, but they behave differently.
- RANK() assigns a rank to each row based on the ordering of a column. Rows with the same value get the same rank, and gaps appear in the ranking sequence after ties.
- NTILE(n) divides the ordered rows into n approximately equal groups (buckets) and assigns a bucket number to each row. It does not care about ties but ensures all rows are distributed into the specified number of tiles.

Quick notes:
RANK = ranking with possible gaps for ties
NTILE = splits rows into equal-sized buckets
Use RANK for positions, NTILE for grouping into percentiles/quartiles

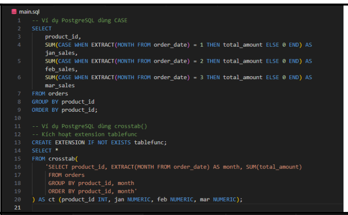


main.sql

```sql
-- Ví dụ RANK
SELECT name, salary,
       RANK() OVER (ORDER BY salary DESC) AS rnk
FROM employees;

-- Ví dụ NTILE (chia thành 4 phần)
SELECT name, salary,
       NTILE(4) OVER (ORDER BY salary DESC) AS quartile
FROM employees;
```

How can you pivot rows into columns in SQL?

To pivot rows into columns in SQL, you transform row values into column headers. In PostgreSQL, you can use the crosstab() function from the tablefunc extension or simulate it using CASE with GROUP BY. Pivoting is useful for reporting or summarizing data, for example turning monthly sales into separate columns for each month.

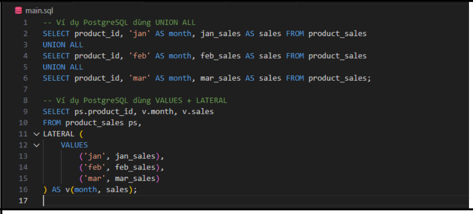


main.sql

```sql
-- Ví dụ PostgreSQL dùng CASE
SELECT
    product_id,
    SUM(CASE WHEN EXTRACT(MONTH FROM order_date) = 1 THEN total_amount ELSE 0 END) AS jan_sales,
    SUM(CASE WHEN EXTRACT(MONTH FROM order_date) = 2 THEN total_amount ELSE 0 END) AS feb_sales,
    SUM(CASE WHEN EXTRACT(MONTH FROM order_date) = 3 THEN total_amount ELSE 0 END) AS mar_sales
FROM orders
GROUP BY product_id
ORDER BY product_id;

-- Ví dụ PostgreSQL dùng crosstab()
-- Kích hoạt extension tablefunc
CREATE EXTENSION IF NOT EXISTS tablefunc;
SELECT *
FROM crosstab(
    'SELECT product_id, EXTRACT(MONTH FROM order_date) AS month, SUM(total_amount)
     FROM orders
     GROUP BY product_id, month
     ORDER BY product_id, month'
) AS ct (product_id INT, jan NUMERIC, feb NUMERIC, mar NUMERIC);
```

How can you unpivot columns into rows in SQL?

To unpivot columns into rows in SQL, you transform multiple column values into a single column with multiple rows. In PostgreSQL, you can use the UNION ALL approach or the jsonb/lateral methods. Unpivoting is useful when you need to normalize data for analysis, like converting monthly sales columns into a single column with month and sales values.

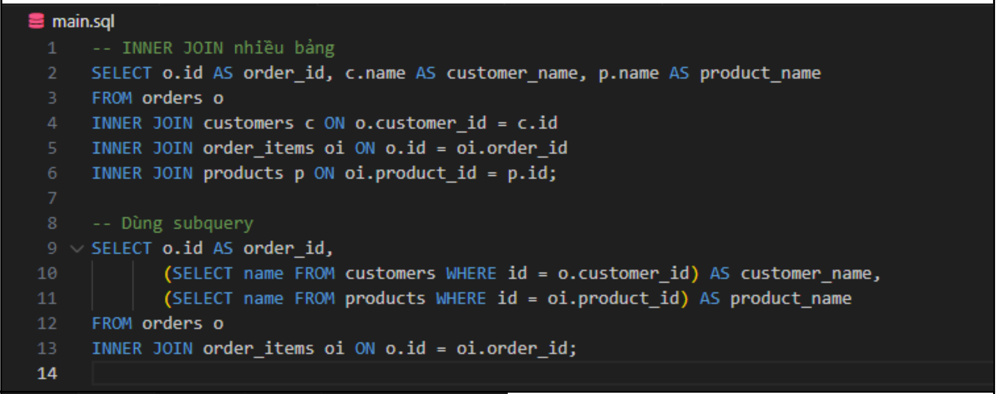


main.sql

```sql
-- Ví dụ PostgreSQL dùng UNION ALL
SELECT product_id, 'jan' AS month, jan_sales AS sales FROM product_sales
UNION ALL
SELECT product_id, 'feb' AS month, feb_sales AS sales FROM product_sales
UNION ALL
SELECT product_id, 'mar' AS month, mar_sales AS sales FROM product_sales;

-- Ví dụ PostgreSQL dùng VALUES + LATERAL
SELECT ps.product_id, v.month, v.sales
FROM product_sales ps,
LATERAL (
    VALUES
        ('jan', jan_sales),
        ('feb', feb_sales),
        ('mar', mar_sales)
) AS v(month, sales);
```

Explain the differences between INNER JOIN with multiple tables versus multiple subqueries.

When retrieving data from multiple tables, you can either use INNER JOINs or multiple subqueries, but there are key differences.
- INNER JOIN with multiple tables combines rows from all tables based on matching keys in a single query. It is usually more efficient, easier to read, and allows the query planner to optimize joins effectively. It returns a single result set with columns from all joined tables.
- Multiple subqueries involve executing separate queries, often in WHERE clauses or in the SELECT statement. This can be less efficient, especially if the subqueries run repeatedly for each row, and can be harder to read. Subqueries are useful when you need aggregated or filtered results independently before joining them to the main query.

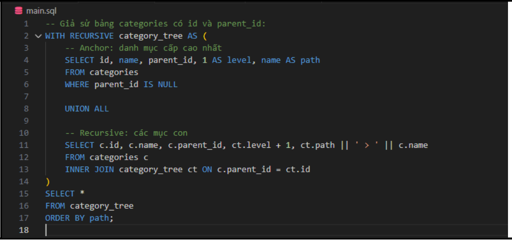


main.sql

```sql
-- INNER JOIN nhiều bảng
SELECT o.id AS order_id, c.name AS customer_name, p.name AS product_name
FROM orders o
INNER JOIN customers c ON o.customer_id = c.id
INNER JOIN order_items oi ON o.id = oi.order_id
INNER JOIN products p ON oi.product_id = p.id;

-- Dùng subquery
SELECT o.id AS order_id,
       (SELECT name FROM customers WHERE id = o.customer_id) AS customer_name,
       (SELECT name FROM products WHERE id = oi.product_id) AS product_name
FROM orders o
INNER JOIN order_items oi ON o.id = oi.order_id;
```

How do you handle hierarchical or tree-structured data in SQL?

Hierarchical or tree-structured data in SQL, like organizational charts or categories, can be handled using recursive queries with CTEs (Common Table Expressions). The idea is to start with root nodes (top-level items) as the anchor, then recursively retrieve child nodes until all levels are fetched. You can also use parent_id/child_id relationships and window functions for path or level calculations.



main.sql

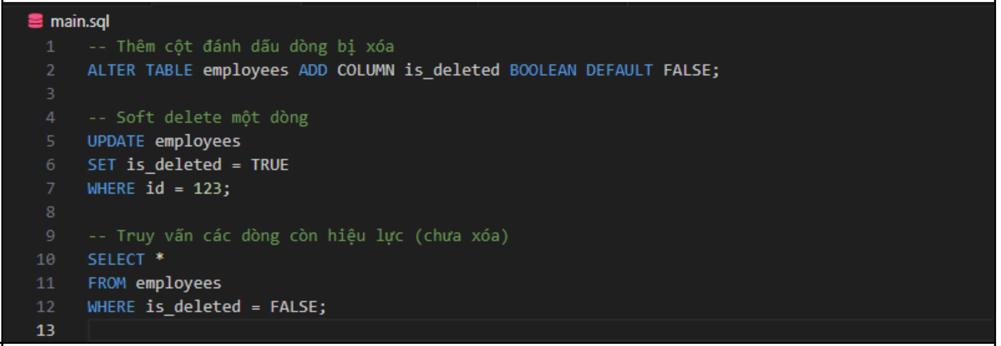
```sql
-- Giả sử bảng categories có id và parent_id:
WITH RECURSIVE category_tree AS (
    -- Anchor: danh mục cấp cao nhất
    SELECT id, name, parent_id, 1 AS level, name AS path
    FROM categories
    WHERE parent_id IS NULL

    UNION ALL

    -- Recursive: các mục con
    SELECT c.id, c.name, c.parent_id, ct.level + 1, ct.path || ' > ' || c.name
    FROM categories c
    INNER JOIN category_tree ct ON c.parent_id = ct.id
)
SELECT *
FROM category_tree
ORDER BY path;
```

How can you implement soft delete (logical delete) in a table?

A soft delete (logical delete) means you mark a row as deleted instead of physically removing it from the table. This is usually done by adding a column like is_deleted or deleted_at. Queries then filter out deleted rows by checking this column. Soft delete is useful for auditing, recovery, or preserving historical data.

Alternative approach: use a timestamp column deleted_at instead of a boolean to record when a row was deleted.

Quick notes:
Soft delete = mark row, do not physically delete
Filter queries to exclude deleted rows
Allows recovery and auditing


main.sql

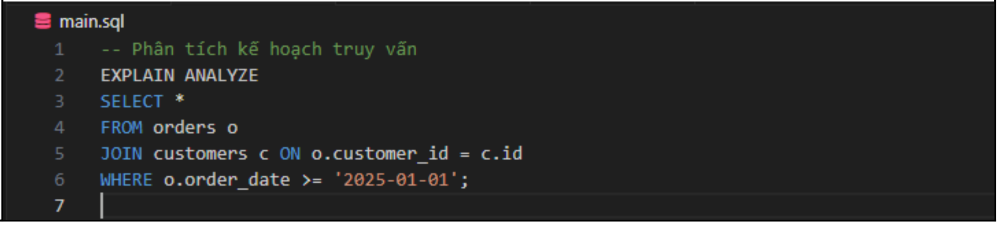
```sql
-- Thêm cột đánh dấu dòng bị xóa
ALTER TABLE employees ADD COLUMN is_deleted BOOLEAN DEFAULT FALSE;

-- Soft delete một dòng
UPDATE employees
SET is_deleted = TRUE
WHERE id = 123;

-- Truy vấn các dòng còn hiệu lực (chưa xóa)
SELECT *
FROM employees
WHERE is_deleted = FALSE;
```

How can you identify and optimize queries with high I/O usage?

Queries with high I/O usage can slow down your database because they read or write a large amount of data from disk. To identify them, you can use execution plans, database statistics, or monitoring tools. Common signs include sequential scans on large tables, large joins, or frequent table scans instead of index usage.

To optimize high I/O queries:
1. Use indexes on columns frequently used in WHERE, JOIN, or ORDER BY clauses.
2. Limit scanned data with proper filtering conditions.
3. Avoid unnecessary SELECT *; fetch only needed columns.
4. Break complex queries into smaller steps or pre-aggregate data if possible.
5. Analyze query plans (EXPLAIN) to see which operations cause high I/O.

Quick notes:
Look for sequential scans on large tables
Check the estimated rows vs actual rows in EXPLAIN
Adding indexes or rewriting queries can reduce I/O

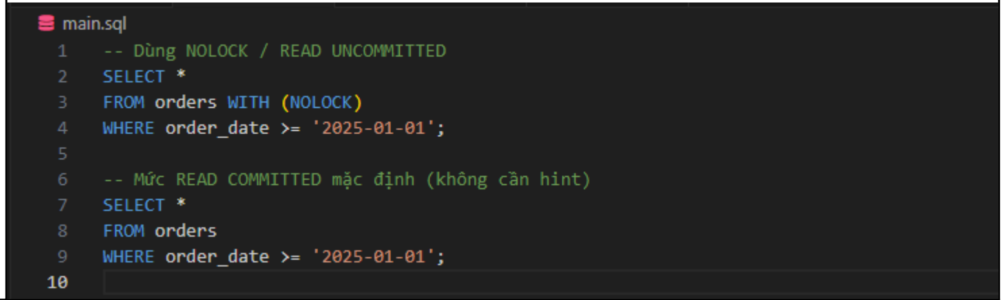

main.sql

```sql
-- Phân tích kế hoạch truy vấn
EXPLAIN ANALYZE
SELECT *
FROM orders o
JOIN customers c ON o.customer_id = c.id
WHERE o.order_date >= '2025-01-01';
```

Explain the difference between NOLOCK (or READ UNCOMMITTED) and default isolation levels.

NOLOCK (or READ UNCOMMITTED) is a transaction isolation level that allows a query to read data without acquiring shared locks. This means it can read uncommitted or "dirty" data from other transactions. It improves performance and reduces blocking but can result in inaccurate or inconsistent results.
The default isolation level in most databases (e.g., READ COMMITTED in PostgreSQL, SQL Server) ensures that a query only reads committed data, preventing dirty reads. This is safer but can cause locks and blocking if many transactions are running concurrently.

Quick notes:
NOLOCK = faster, non-blocking, may read dirty data
Default isolation = safer, reads only committed data, may block
Use NOLOCK carefully; avoid for critical financial or audit data

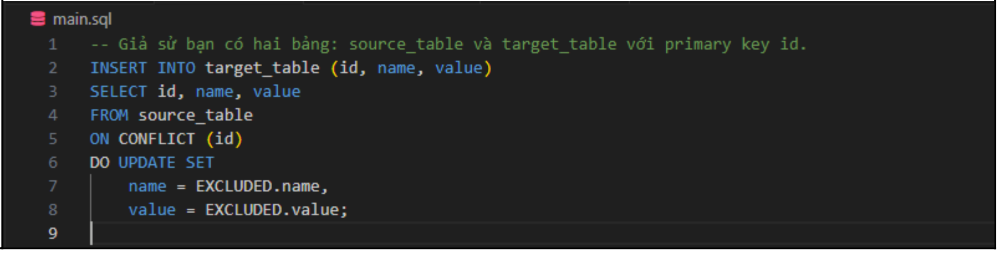

main.sql

```sql
-- Dùng NOLOCK / READ UNCOMMITTED
SELECT *
FROM orders WITH (NOLOCK)
WHERE order_date >= '2025-01-01';

-- Mức READ COMMITTED mặc định (không cần hint)
SELECT *
FROM orders
WHERE order_date >= '2025-01-01';
```

How do you merge data from two tables (UPSERT) in SQL?

UPSERT in SQL is a combination of INSERT and UPDATE: it inserts a new row if it doesn’t exist, or updates the existing row if it does. In PostgreSQL, this can be achieved using INSERT ... ON CONFLICT, specifying the unique constraint or primary key to detect conflicts. UPSERT is useful for synchronizing or merging data between tables.

Quick notes:
UPSERT = insert if new, update if exists
ON CONFLICT (key) detects duplicate keys
EXCLUDED refers to the row that would have been inserted


main.sql

```sql
-- Giả sử bạn có hai bảng: source_table và target_table với primary key id.
INSERT INTO target_table (id, name, value)
SELECT id, name, value
FROM source_table
ON CONFLICT (id)
DO UPDATE SET
    name = EXCLUDED.name,
    value = EXCLUDED.value;
```

	Explain how indexing affects INSERT, UPDATE, and DELETE performance. Indexing improves SELECT performance by allowing the database to quickly locate rows, but it also affects INSERT, UPDATE, and DELETE operations: - INSERT: When inserting a new row, all relevant indexes must also be updated. More indexes = more overhead, which can slow down inserts. - UPDATE: Updating indexed columns requires updating the index as well. If many indexes exist on the updated column(s), the operation becomes slower. - DELETE: Deleting a row requires removing entries from all indexes that reference that row, which adds overhead. In summary, indexes improve read performance but add extra work for write operations. It's important to balance between query speed and write performance. Quick notes: More indexes = faster reads, slower writes Only index columns used frequently in queries Consider partial or filtered indexes if appropriate	
	Question: What is the result of NULL = NULL? Trick: Nhiều người nghĩ là TRUE. Thực tế: kết quả là NULL (UNKNOWN). So sánh phải dùng IS NULL.	
	Question: What is the difference between COUNT(column_name) and COUNT(*)? Trick: COUNT(column_name) chỉ đếm các giá trị không NULL, COUNT(*) đếm tất cả dòng.	
	Question: Can you use HAVING without GROUP BY? Trick: Có thể, nhưng hiếm. Nhiều người sẽ trả lời là không ngay lập tức. SQL cho phép HAVING mà không có GROUP BY (tương đương một nhóm ảo).	
	Question: What happens if a subquery in WHERE returns multiple rows with = operator? Trick: Nó sẽ lỗi. Để tránh phải dùng IN thay vì =.	

	Question: Can you select a column in SQL that is not part of GROUP BY and not an aggregate? Trick: Một số hệ quản trị (Oracle strict mode) sẽ lỗi; MySQL cho phép nhưng kết quả không xác định.	
	Question: What is the difference between INNER JOIN ON 1=1 and CROSS JOIN? Trick: Nhiều người không nhận ra rằng INNER JOIN ON 1=1 thực chất tạo ra Cartesian product như CROSS JOIN.	
	Question: What is the purpose of ORDER BY NULL? Trick: Nó tắt việc sắp xếp trong một query, thường dùng để tối ưu aggregate queries.	
	Question: What happens if you UPDATE a table using a JOIN that matches multiple rows per record? Trick: Kết quả có thể bị cập nhật nhiều lần hoặc không xác định, tùy DBMS.	
	Question: If two transactions update the same row concurrently, which value will persist? Trick: Phụ thuộc vào isolation level; nhiều người trả lời trực quan nhưng quên rằng isolation level quyết định lock/rollback.	
	Question: What happens if two tables in a JOIN have the same column name and you don’t alias? Trick: Một số DBMS lỗi ngay, một số trả về không xác định.	
	Question: What is the result of '5' + 1 in SQL? Trick: Tùy DBMS: MySQL auto convert, SQL Server lỗi hoặc convert khác. Rất nhiều người quên về kiểu dữ liệu.	
	Question: Which is faster: EXISTS or IN? Trick: Không có câu trả lời chung; phụ thuộc DBMS, số lượng bản ghi, và NULL trong danh sách. Nhiều người trả lời vội vàng là EXISTS luôn nhanh hơn.	

	Question: What is the result of this query? SELECT COUNT(*) FROM employees WHERE salary > 5000 OR salary < 5000; Trick: Nhiều người nghĩ là toàn bộ nhân viên, nhưng nếu có salary IS NULL, họ không được tính. Vì NULL không so sánh được bằng toán tử > hoặc <.	
	Question: What does this return? SELECT * FROM employees WHERE NOT (salary > 5000); Trick: Không tương đương với salary <= 5000 vì các hàng có NULL cũng bị loại bỏ.	
	Question: What is the output of: SELECT 1 / 0; Trick: Một số DBMS sẽ báo lỗi (division by zero), nhưng MySQL trong chế độ mặc định có thể trả NULL.	
	Question: Can you use GROUP BY without any aggregate function? Answer: Có thể. Nó chỉ nhóm các hàng mà không tính toán, thường dùng để loại bỏ trùng (như DISTINCT).	
	Question: What does SELECT DISTINCT NULL, NULL; return? Trick: Chỉ 1 dòng — DISTINCT coi tất cả NULL là "bằng nhau".	
	Question: Which returns more rows, INNER JOIN or LEFT JOIN? Trick: Không có câu trả lời tuyệt đối — LEFT JOIN có thể nhiều hơn, nhưng nếu mọi hàng có match, thì số dòng bằng nhau.	
	Question: Does HAVING filter before or after aggregation? Trick: Sau khi nhóm (GROUP BY) và tính toán aggregate xong mới lọc.	
	Question: What happens when you compare NULL with anything using !=? Trick: Luôn trả UNKNOWN. NULL không bao giờ bằng hay khác bất kỳ giá trị nào.	

	Question: Does ORDER BY guarantee consistent ordering if there are duplicate values in the sorting column? Trick: Không. Thứ tự các hàng có cùng giá trị có thể khác nhau giữa các lần chạy nếu không có tiêu chí phụ.	
	Question: Is DELETE FROM table; the same as TRUNCATE TABLE table;? Trick: Không. DELETE có thể rollback và kích hoạt trigger; TRUNCATE thì không (tùy DBMS).	
	Question: Can NULL be a value in a UNIQUE column? Trick: Có thể. SQL coi NULL là “unknown”, nên hai NULL không bị coi là trùng (MySQL, PostgreSQL).	
	Question: Which one executes faster — SELECT * or SELECT specific_columns? Trick: SELECT * có thể chậm hơn vì phải fetch tất cả các cột, đặc biệt với nhiều cột hoặc I/O lớn.	
	Question: What will this return? SELECT SUM(CASE WHEN dept_id = 10 THEN 1 END) AS count_dept10 FROM employees; Trick: Có thể trả NULL nếu không có dòng nào thỏa dept_id = 10, vì SUM (NULL) là NULL, không phải 0.	
	Question: Can you use an aggregate function inside a WHERE clause? Trick: Không. WHERE chạy trước aggregation. Phải dùng HAVING hoặc subquery.	

	Question: What is the difference between:
	
	SELECT COUNT(*) FROM table;
	SELECT COUNT(1) FROM table;
	
	Trick: Thường không khác biệt về hiệu năng. Một số người tin COUNT(1) nhanh hơn, nhưng hầu hết DBMS tối ưu như nhau.	

	Question: Will this query run? SELECT * FROM employees WHERE salary = ALL (SELECT salary FROM employees); Trick: Có thể trả rỗng vì điều kiện nghĩa là “salary bằng tất cả các salary khác” — chỉ đúng nếu mọi người cùng lương.	

	Question: What happens if you update a table that is used in its own subquery?
	
	UPDATE employees
	SET salary = salary * 1.1
	WHERE emp_id IN (SELECT manager_id FROM employees);
	
	Trick: Một số DBMS (như MySQL) không cho phép subquery đọc và ghi cùng bảng (error 1093).	

	Question: What’s wrong with this query? SELECT name, AVG(salary) FROM employees; Trick: Không có GROUP BY name → lỗi (trừ khi MySQL trong mode lỏng).	
	Question: What’s the result of SELECT NULL + 5;? Trick: NULL. Mọi phép toán với NULL → NULL.	
	Question: What happens when you insert a row with no value into an IDENTITY (auto-increment) column? Trick: DBMS tự tạo giá trị mới, nhưng nếu bạn chèn NULL hoặc DEFAULT, hành vi có thể khác tùy hệ quản trị.	