# 11-06-2025

**SQL Server Practical Assignment (30 Minutes)**

Section A: Managing Databases (10 mins)

**1. List all system databases in SQL Server.**

SELECT name FROM sys.databases WHERE database\_id < 5;

**2. List physical file paths for all databases.**

SELECT name, physical\_name FROM sys.master\_files;

**3. Create a new user-defined database named TeamDB.**

CREATE DATABASE TeamDB;

**4. Rename the database TeamDB to ProjectDB.**

ALTER DATABASE TeamDB MODIFY NAME = ProjectDB;

**5. Drop the ProjectDB database.**

DROP DATABASE ProjectDB;

Section B: Managing Tables (10 mins)

**1. Create a table Employees with the following columns:**

EmpID INT (Primary Key)

Name VARCHAR(50)

Department VARCHAR(30)

JoiningDate DATE

IsActive BIT

Salary DECIMAL(10,2)

CREATE TABLE Employees (

EmpID INT PRIMARY KEY,

Name VARCHAR(50),

Department VARCHAR(30),

JoiningDate DATE,

IsActive BIT,

Salary DECIMAL(10,2)

);

**2. Add a column Salary (DECIMAL) to the table.**

ALTER TABLE Employees ADD Salary DECIMAL(10,2);

**3. Rename table Employees to TeamMembers.**

EXEC sp\_rename 'Employees', 'TeamMembers';

**4. Drop the table TeamMembers.**

DROP TABLE TeamMembers;

Section C: DML Operations (10 mins)

**1. Insert three rows into Employees.**

INSERT INTO Employees VALUES

(1, 'Amit', 'HR', '2022-01-01', 1, 50000),

(2, 'Sneha', 'IT', '2021-06-15', 1, 75000),

(3, 'John', 'Finance', '2020-10-10', 0, 65000);

**2. Update salary of 'Sneha' to 80000.**

UPDATE Employees SET Salary = 80000 WHERE Name = 'Sneha';

**3. Delete employee with IsActive = 0.**

DELETE FROM Employees WHERE IsActive = 0;

**4. Retrieve names and departments of all employees.**

SELECT Name, Department FROM Employees;

**5. Fetch employees from 'IT' department with salary above 70000.**

SELECT \* FROM Employees WHERE Department = 'IT' AND Salary > 70000;

**6. Apply filtering using LIKE, BETWEEN, and IN.**

SELECT \* FROM Employees WHERE Name LIKE 'S%';

SELECT \* FROM Employees WHERE Salary BETWEEN 60000 AND 80000;

SELECT \* FROM Employees WHERE Department IN ('IT', 'Finance');

# 12-06-2025

**Medium-Level Practical SQL Questions**

**1. Insert and Update with Integrity:**

Create a 'students' table with constraints (NOT NULL, UNIQUE). Insert 5 records. Then, update a

student's marks ensuring data integrity is maintained.

**CREATE TABLE students (**

**student\_id INT PRIMARY KEY,**

**name VARCHAR(100) NOT NULL UNIQUE,**

**marks INT NOT NULL**

**);**

**INSERT INTO students VALUES**

**(1, 'Alice', 85),**

**(2, 'Bob', 78),**

**(3, 'Charlie', 92),**

**(4, 'David', 88),**

**(5, 'Eva', 90);**

**UPDATE students SET marks = 95 WHERE name = 'Bob';**

**2. String Function Challenge:**

Given a 'customers' table with a 'full\_name' column, write a query to display:

- First name

- Last name

- Length of each name

**-- Create the customers table**

**CREATE TABLE customers (**

**customer\_id INTEGER PRIMARY KEY AUTOINCREMENT,**

**full\_name TEXT NOT NULL**

**);**

**-- Insert sample data**

**INSERT INTO customers (full\_name) VALUES**

**('John Doe'),**

**('Jane Smith'),**

**('Emily Johnson');**

**-- String function query to extract first and last names and their lengths**

**SELECT**

**full\_name,**

**SUBSTR(full\_name, 1, INSTR(full\_name, ' ') - 1) AS first\_name,**

**SUBSTR(full\_name, INSTR(full\_name, ' ') + 1) AS last\_name,**

**LENGTH(SUBSTR(full\_name, 1, INSTR(full\_name, ' ') - 1)) AS first\_name\_length,**

**LENGTH(SUBSTR(full\_name, INSTR(full\_name, ' ') + 1)) AS last\_name\_length**

**FROM customers;**

**3. Date Function Usage:**

From a 'sales' table with a 'sale\_date' column, write a query to:

- Extract the month name and year

- Display how many days ago the sale happened

**CREATE TABLE sales (**

**sale\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**sale\_date DATE NOT NULL**

**);**

**INSERT INTO sales (sale\_date) VALUES**

**('2025-06-10'),**

**('2025-05-20'),**

**('2025-04-01');**

**SELECT**

**sale\_id,**

**sale\_date,**

**MONTHNAME(sale\_date) AS month\_name,**

**YEAR(sale\_date) AS year,**

**DATEDIFF(CURDATE(), sale\_date) AS days\_ago**

**FROM sales;**

**4. Mathematical Functions on Salary:**

In an 'employees' table, calculate:

- Salary after a 10% hike

- Round the salary to the nearest hundred

**CREATE TABLE employees (**

**emp\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100) NOT NULL,**

**salary DECIMAL(10, 2) NOT NULL**

**);**

**INSERT INTO employees (name, salary) VALUES**

**('Alice', 48350.75),**

**('Bob', 55990.00),**

**('Charlie', 61240.25),**

**('Diana', 70000.00);**

**SELECT**

**emp\_id,**

**name,**

**salary,**

**ROUND(salary \* 1.10, 2) AS salary\_after\_10\_percent\_hike,**

**ROUND(salary, -2) AS rounded\_to\_nearest\_100**

**FROM employees;**

**5. System Function Check:**

Retrieve:

- Current date and time

- Database name and logged-in user

**SELECT**

**NOW() AS current\_datetime,**

**DATABASE() AS current\_database,**

**USER() AS logged\_in\_user;**

**6. Demo: Custom Result Set:**

From the 'products' table, write a query that:

- Returns product name in uppercase

- Replaces any NULL prices with 'Not Available'

**CREATE TABLE products (**

**product\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**product\_name VARCHAR(100) NOT NULL,**

**price DECIMAL(10, 2) DEFAULT NULL**

**);**

**INSERT INTO products (product\_name, price) VALUES**

**('Laptop', 75000.00),**

**('Tablet', NULL),**

**('Smartphone', 35000.00),**

**('Headphones', NULL);**

**SELECT**

**UPPER(product\_name) AS product\_name\_upper,**

**IFNULL(CAST(price AS CHAR), 'Not Available') AS price\_display**

**FROM products;**

**7. Aggregate Functions Practice:**

From a 'transactions' table, get:

- Total sales

- Average sale value

- Maximum and minimum sale on a single transaction

**CREATE TABLE transactions (**

**transaction\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**amount DECIMAL(10, 2)**

**);**

**INSERT INTO transactions (amount) VALUES**

**(250.00), (499.99), (120.75), (780.50), (350.25);**

**SELECT**

**SUM(amount) AS total\_sales,**

**AVG(amount) AS average\_sale\_value,**

**MAX(amount) AS max\_sale,**

**MIN(amount) AS min\_sale**

**FROM transactions;**

**8. Grouping with Aggregation:**

From a 'sales' table:

- Group by product category

- Show total sales and number of transactions in each category

**CREATE TABLE sales (**

**sale\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**product\_category VARCHAR(50),**

**sale\_amount DECIMAL(10, 2)**

**);**

**INSERT INTO sales (product\_category, sale\_amount) VALUES**

**('Electronics', 1200.50),**

**('Clothing', 750.00),**

**('Electronics', 500.00),**

**('Clothing', 300.25),**

**('Furniture', 1500.00);**

**SELECT**

**product\_category,**

**SUM(sale\_amount) AS total\_sales,**

**COUNT(\*) AS number\_of\_transactions**

**FROM sales**

**GROUP BY product\_category;**

**9. Inner Join for Orders and Customers:**

Join 'orders' and 'customers' to show:

- Customer name

- Order amount

- Only for customers who made orders

**CREATE TABLE customers (**

**customer\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**name VARCHAR(100)**

**);**

**CREATE TABLE orders (**

**order\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**customer\_id INT,**

**order\_amount DECIMAL(10, 2),**

**FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)**

**);**

**INSERT INTO customers (name) VALUES**

**('Alice'), ('Bob'), ('Charlie');**

**INSERT INTO orders (customer\_id, order\_amount) VALUES**

**(1, 500.00),**

**(2, 1200.75),**

**(1, 300.00); -- Charlie didn't order**

**10. Left Join for Products with or without Orders:**

Show all products with:

- Their order details (if available)

- Use LEFT JOIN

**CREATE TABLE products (**

**product\_id INT PRIMARY KEY,**

**product\_name VARCHAR(100)**

**);**

**CREATE TABLE orders (**

**order\_id INT PRIMARY KEY,**

**product\_id INT,**

**quantity INT,**

**FOREIGN KEY (product\_id) REFERENCES products(product\_id)**

**);**

**INSERT INTO products VALUES**

**(1, 'Laptop'), (2, 'Phone'), (3, 'Tablet');**

**INSERT INTO orders VALUES**

**(101, 1, 2), (102, 2, 1); -- No order for Tablet**

**SELECT**

**p.product\_name,**

**o.order\_id,**

**o.quantity**

**FROM products p**

**LEFT JOIN orders o ON p.product\_id = o.product\_id;**

**11. Right Join for Customer Contacts:**

Use a RIGHT JOIN between 'contacts' and 'customers' to display:

- All customers, even if they don't have contact info

**CREATE TABLE customers (**

**customer\_id INT PRIMARY KEY,**

**name VARCHAR(100)**

**);**

**CREATE TABLE contacts (**

**contact\_id INT PRIMARY KEY,**

**customer\_id INT,**

**email VARCHAR(100),**

**FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)**

**);**

**INSERT INTO customers VALUES**

**(1, 'Alice'), (2, 'Bob'), (3, 'Charlie');**

**INSERT INTO contacts VALUES**

**(201, 1, 'alice@mail.com'), (202, 2, 'bob@mail.com'); -- Charlie has no contact**

**SELECT**

**c.customer\_id,**

**c.name,**

**ct.email**

**FROM contacts ct**

**RIGHT JOIN customers c ON c.customer\_id = ct.customer\_id;**

**12. Full Outer Join for Suppliers and Products:**

Use a FULL OUTER JOIN to list:

- All suppliers and products

- Match supplier to product, or show NULLs where not available

**CREATE TABLE suppliers (**

**supplier\_id INT PRIMARY KEY,**

**supplier\_name VARCHAR(100)**

**);**

**CREATE TABLE products (**

**product\_id INT PRIMARY KEY,**

**product\_name VARCHAR(100),**

**supplier\_id INT**

**);**

**INSERT INTO suppliers VALUES**

**(1, 'Supplier A'), (2, 'Supplier B');**

**INSERT INTO products VALUES**

**(10, 'Laptop', 1), (11, 'Monitor', NULL);**

**-- Left join**

**SELECT**

**s.supplier\_name,**

**p.product\_name**

**FROM suppliers s**

**LEFT JOIN products p ON s.supplier\_id = p.supplier\_id**

**UNION**

**-- Right join**

**SELECT**

**s.supplier\_name,**

**p.product\_name**

**FROM suppliers s**

**RIGHT JOIN products p ON s.supplier\_id = p.supplier\_id;**

**13. Cross Join for Offers:**

Suppose you have tables 'products' and 'offers'.

Write a CROSS JOIN to show:

- All possible combinations of products and offers

**CREATE TABLE offers (**

**offer\_id INT PRIMARY KEY,**

**offer\_name VARCHAR(50)**

**);**

**-- Assume 'products' table already exists**

**INSERT INTO offers VALUES**

**(1, '10% Off'), (2, 'Buy 1 Get 1');**

**SELECT**

**p.product\_name,**

**o.offer\_name**

**FROM products p**

**CROSS JOIN offers o;**

**14. Join with Aggregation:**

Join 'orders' and 'products', then group by product category and:

- Show total quantity sold and average price per category

**ALTER TABLE products ADD COLUMN category VARCHAR(50);**

**UPDATE products**

**SET category = CASE**

**WHEN product\_name = 'Laptop' THEN 'Electronics'**

**WHEN product\_name = 'Phone' THEN 'Electronics'**

**WHEN product\_name = 'Tablet' THEN 'Electronics'**

**ELSE 'General'**

**END;**

**SELECT**

**p.category,**

**SUM(o.quantity) AS total\_quantity\_sold,**

**AVG(p\_price.price) AS average\_price**

**FROM orders o**

**JOIN products p ON o.product\_id = p.product\_id**

**JOIN (**

**SELECT product\_id, 50000 AS price FROM products**

**) AS p\_price ON o.product\_id = p\_price.product\_id**

**GROUP BY p.category;**

**15. Demo: Join with Grouping and Filter:**

Join 'students' and 'marks' tables.

Display:

- Student name

- Average marks

- Filter to show only students with average marks > 75

**CREATE TABLE students (**

**student\_id INT PRIMARY KEY,**

**name VARCHAR(100)**

**);**

**CREATE TABLE marks (**

**mark\_id INT PRIMARY KEY,**

**student\_id INT,**

**subject VARCHAR(50),**

**score INT,**

**FOREIGN KEY (student\_id) REFERENCES students(student\_id)**

**);**

**INSERT INTO students VALUES**

**(1, 'Ravi'), (2, 'Neha'), (3, 'John');**

**INSERT INTO marks VALUES**

**(1, 1, 'Math', 80), (2, 1, 'Science', 90),**

**(3, 2, 'Math', 70), (4, 2, 'Science', 60),**

**(5, 3, 'Math', 95), (6, 3, 'Science', 85);**

**SELECT**

**s.name AS student\_name,**

**AVG(m.score) AS average\_marks**

**FROM students s**

**JOIN marks m ON s.student\_id = m.student\_id**

**GROUP BY s.student\_id**

**HAVING AVG(m.score) > 75;**

# 13-06-2025

**Querying Data by Using Subqueries - Examples**

**Sample Table: Employees**

CREATE TABLE Employees (EmpID INT, Name VARCHAR(50), Department VARCHAR(50), Salary INT);

INSERT INTO Employees VALUES (1, 'Alice', 'HR', 5000);

INSERT INTO Employees VALUES (2, 'Bob', 'IT', 7000);

INSERT INTO Employees VALUES (3, 'Charlie', 'Finance', 6000);

INSERT INTO Employees VALUES (4, 'David', 'IT', 8000);

INSERT INTO Employees VALUES (5, 'Eva', 'HR', 5500);

INSERT INTO Employees VALUES (6, 'Frank', 'Finance', 6200);

**Querying Data by Using Subqueries**

Query:

SELECT Name FROM Employees WHERE Salary > (SELECT AVG(Salary) FROM Employees);

**Querying Data by Using Subqueries Using the EXISTS**

Query:

SELECT Name FROM Employees e WHERE EXISTS (SELECT 1 FROM Employees WHERE Department = 'IT' AND e.Department = Department);

**Querying Data by Using Subqueries using ANY**

Query:

SELECT Name FROM Employees WHERE Salary > ANY (SELECT Salary FROM Employees WHERE Department = 'HR');

**Querying Data by Using Subqueries using ALL Keywords**

Query:

SELECT Name FROM Employees WHERE Salary > ALL (SELECT Salary FROM Employees WHERE Department = 'HR');

**Querying Data by Using Subqueries using Nested Subqueries**

Query:

SELECT Name FROM Employees WHERE Salary = (SELECT MAX(Salary) FROM Employees WHERE Department = (SELECT Department FROM Employees WHERE Name = 'Charlie'));

**Querying Data by Using Subqueries Using Correlated Subqueries**

Query:

SELECT Name FROM Employees e1 WHERE Salary > (SELECT AVG(Salary) FROM Employees e2 WHERE e1.Department = e2.Department);

**Querying Data by Using Subqueries Using UNION**

Query:

SELECT Name FROM Employees WHERE Department = 'HR' UNION SELECT Name FROM Employees WHERE Salary > 7000;

**Querying Data by Using Subqueries using INTERSECT**

Query:

SELECT Name FROM Employees WHERE Department = 'IT' INTERSECT SELECT Name FROM Employees WHERE Salary > 7000;

**Querying Data by Using Subqueries using EXCEPT**

Query:

SELECT Name FROM Employees WHERE Department = 'IT' EXCEPT SELECT Name FROM Employees WHERE Salary > 7000;

**Querying Data by Using Subqueries using MERGE**

Query:

MERGE INTO Employees AS target USING (SELECT 2 AS EmpID, 'Bob' AS Name) AS source ON target.EmpID = source.EmpID WHEN MATCHED THEN UPDATE SET Salary = 7500;

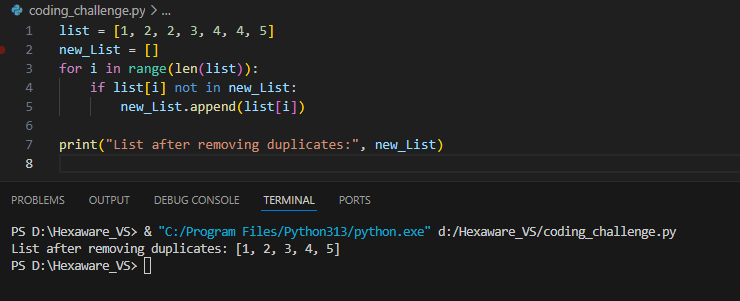
# 18-06-2025

Python Coding Challenge

Q1. Write a Python program to remove all duplicates from a list without using the set() function.

Input Example: [1, 2, 2, 3, 4, 4, 5]

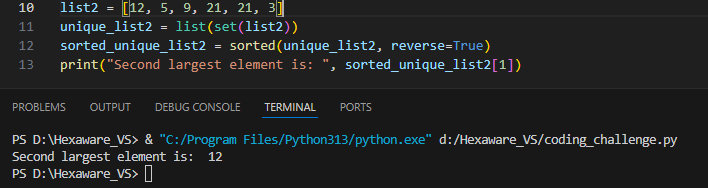
Output: [1, 2, 3, 4, 5]



Q2. Given a list of integers, write a program to find the second highest unique number.

Input Example: [12, 5, 9, 21, 21, 3]

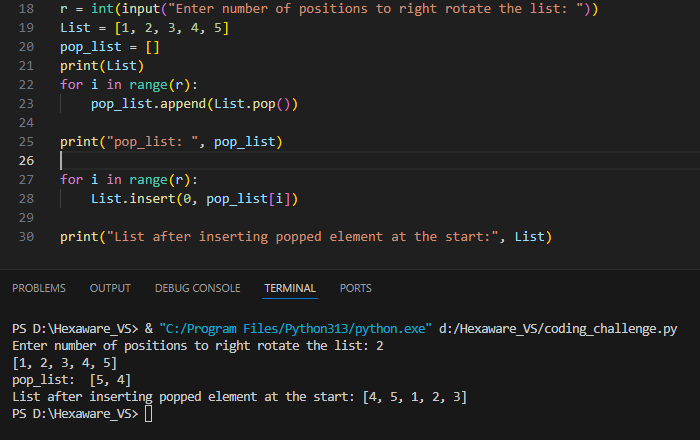
Output: 12



Q3. Rotate a list to the right by k positions.

Input: List = [1, 2, 3, 4, 5], k = 2

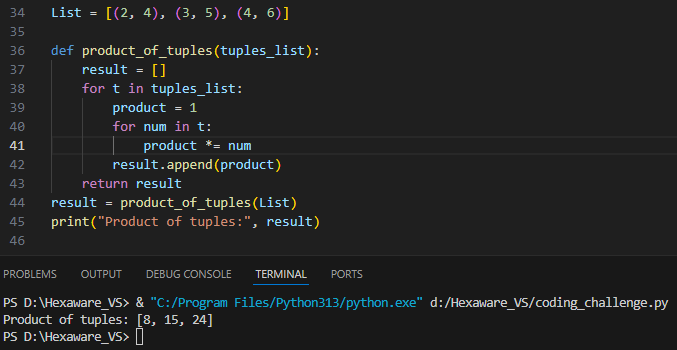
Output: [4, 5, 1, 2, 3]



Q4. Write a Python program to multiply the elements of each tuple in a list of tuples and return a new list.

Input: [(2, 4), (3, 5), (4, 6)]

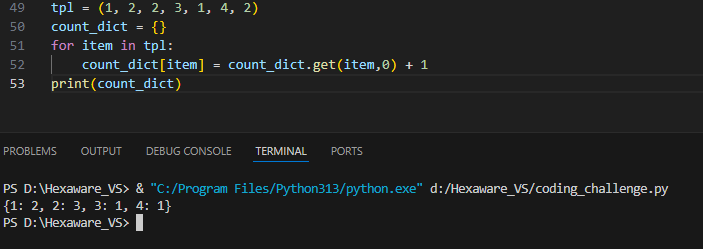
Output: [8, 15, 24]



Q5. Given a tuple of integers, write a program to count how many times each element occurs.

Input: (1, 2, 2, 3, 1, 4, 2)

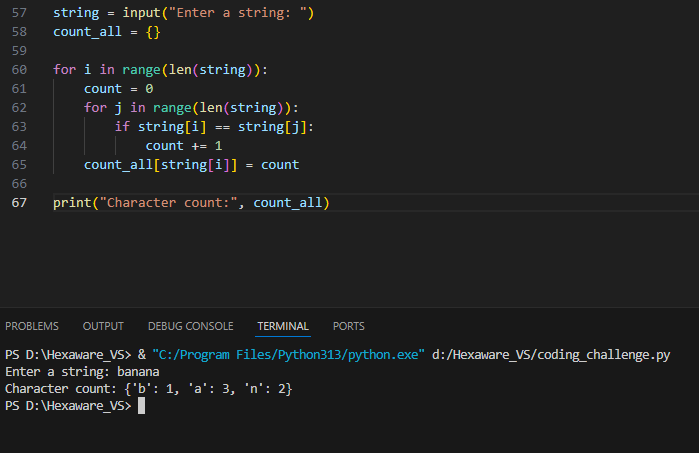
Output: {1: 2, 2: 3, 3: 1, 4: 1}



Q6. Write a Python program to count the frequency of each character in a string using a dictionary.

Input: 'banana'

Output: {'b': 1, 'a': 3, 'n': 2}



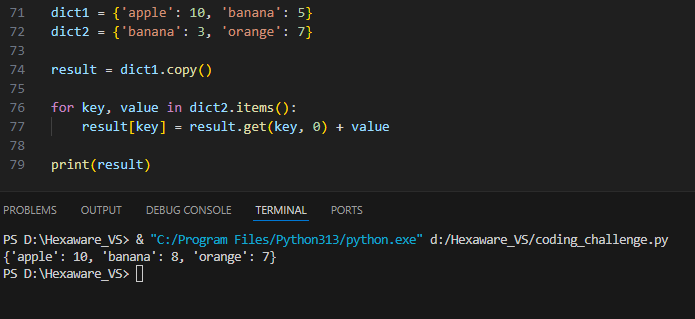
Python Coding Challenge

Topic: List, Tuple, Dictionary, Set | Total Questions: 10 | Time: 60 minutes

Q7. Merge two dictionaries such that common keys have their values summed.

Input: {'apple': 10, 'banana': 5}, {'banana': 3, 'orange': 7}

Output: {'apple': 10, 'banana': 8, 'orange': 7}

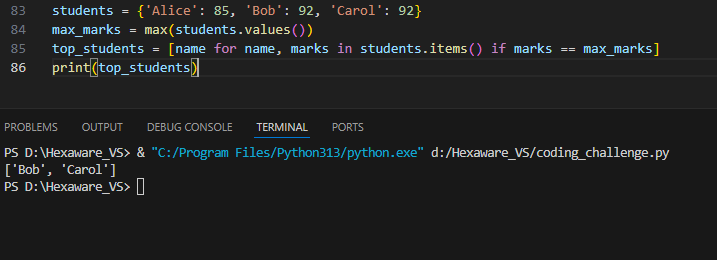


Q8. Given a dictionary of student names and their marks, print the name(s) of the student(s) with the highest

marks.

Input: {'Alice': 85, 'Bob': 92, 'Carol': 92}

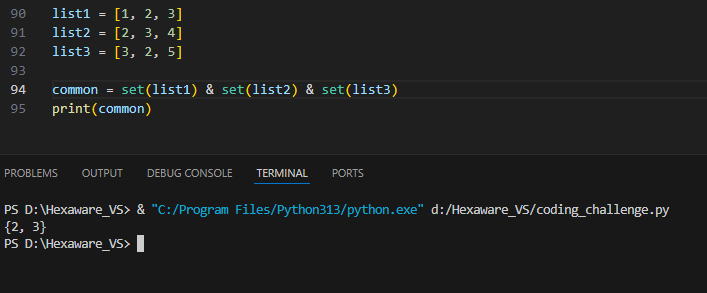
Output: ['Bob', 'Carol']



Q9. Write a Python program to find all common elements among three lists using set operations.

Input: [1, 2, 3], [2, 3, 4], [3, 2, 5]

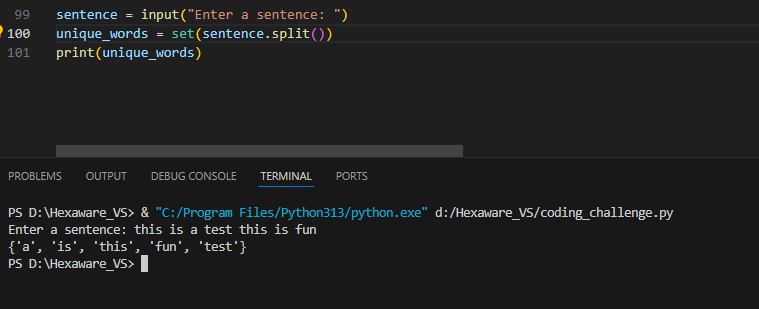
Output: {2, 3}



Q10. From a sentence entered by the user, extract and display all unique words using a set.

Input: 'this is a test this is fun'

Output: {'this', 'is', 'a', 'test', 'fun'}



Pattern Printing   
  
# Get number of rows from user

rows = int(input("Enter number of rows for pyramid: "))

# Print pyramid pattern

for i in range(1, rows + 1):

    print(' ' \* (rows-i) + '\* ' \* i)

# Number pyramid

print("Number Pyramid:")

for i in range(1, rows + 1):

    # Print spaces

    print(' ' \* (rows - i), end='')

    # Print numbers from 1 to (2\*i - 1)

    for j in range(1, 2 \* i):

        print(j, end='')

    print()

# Alphabet pyramid

print("\nAlphabet Pyramid:")

for i in range(1, rows + 1):

    # Print spaces

    print(' ' \* (rows - i), end='')

    # Print alphabet characters from A to the corresponding character

    for j in range(1, 2 \* i):

        print(chr(64 + j), end='')  # 65 is 'A'

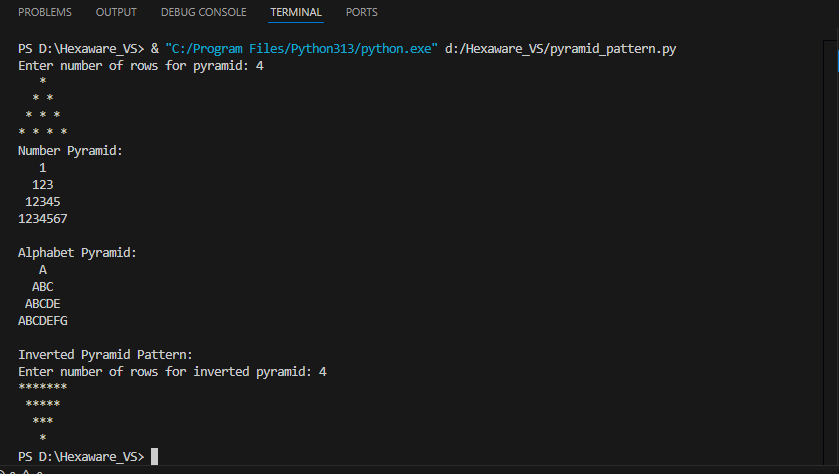
    print()

print("\nInverted Pyramid Pattern:")

rows = int(input("Enter number of rows for inverted pyramid: "))

for i in range(rows, 0, -1):

    print(' ' \* (rows - i) + '\*' \* (2 \* i - 1))



If elif else:

age = int(input("Enter your age: "))

if(age <= 12):

    print("You are a kid.")

elif(age <= 18):

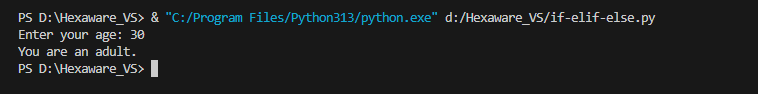
    print("You are a teenager.")

elif(age < 60):

    print("You are an adult.")

else:

    print("You are old.")



23-06-2025

Section A: Basic Understanding (Short Answer Questions)

**1) What is a Python package? How is it different from a module?**  
A Python package is a directory containing an \_\_init\_\_.py file and one or more modules or subpackages. A module is a single .py file, while a package is a collection of modules.

**2) What is the purpose of**\_\_init\_\_.py**in a package directory?**  
\_\_init\_\_.py marks the directory as a Python package and can contain initialization code for the package.

**3) What happens when you use**from package import \***in Python?**  
All names listed in the package’s \_\_all\_\_ variable (if defined) are imported. If \_\_all\_\_ is not defined, all modules and variables not starting with an underscore are imported.

**4) What is the effect of defining**\_\_all\_\_**in a package’s**\_\_init\_\_.py**file?**  
It controls which modules or names are imported when using from package import \*, restricting the import to only those listed in \_\_all\_\_.

**5) How can you create and use a subpackage in Python?**  
Create a subdirectory with an \_\_init\_\_.py file inside a package. Import modules from the subpackage using dot notation, e.g., from package.subpackage import module.

circle.py

def area\_of\_circle(radius):

    from math import pi

    return pi \* radius \* radius

rectangle.py

from .circle import area\_of\_circle

def area\_of\_rectangle(length, width):

    return length \* width

module\_a.py

from .module\_b import greet

def call\_greet():

    return greet()

if \_\_name\_\_ == "\_\_main\_\_":

    call\_greet()

module\_d.py

def greet():

    print("Hello from module\_b")

advanced\_ops.py

def power(a, b):

    return a \*\* b

def factorial(n):

    if n == 0 or n == 1:

        return 1

    return n \* factorial(n - 1)

basic\_ops.py

def add(a, b):

    return a + b

def subtract(a, b):

    return a - b

demo\_math\_utils.py

from math\_utils.basic\_ops import add, subtract

from math\_utils.advanced\_ops import power, factorial

print(add(2, 3))         # 5

print(subtract(5, 2))    # 3

print(power(2, 4))       # 16

print(factorial(5))      # 120

Exception handling questions

# Exception Handling – Question Paper

## Section A: Basic Try-Except (2 marks each)

1. Write a program to divide two numbers entered by the user. Handle ZeroDivisionError using try-except.
2. Write a program to convert a string to an integer. Handle ValueError if the input is not a valid number.
3. Accept two numbers from the user and perform addition. Use try-except to handle invalid input types.
4. Write a program to read an element from a list using an index entered by the user. Handle IndexError.

## Section B: Try-Except-Else (4 marks each)

1. Create a program that accepts a number from the user and prints its square. Use try-except-else to handle ValueError and ensure successful computation is shown only if there's no error.
2. Write a program to open a file and read contents. Use try-except-else to handle FileNotFoundError.
3. Write a Python program to convert a number to its binary format. Use try-except-else to handle any invalid input.

## Section C: Try-Finally (5 marks each)

1. Write a program that opens a file and ensures it gets closed, whether or not an exception occurs. Use try-finally.
2. Simulate a login process where the user input is handled in a try block and a log message is printed in finally regardless of success or failure.
3. Write a program that divides two numbers, catching errors with try-except, and printing a clean-up message using finally.

## Section D: Combined Exception Handling (6 marks each)

1. Create a program that handles multiple exceptions: ZeroDivisionError, ValueError, and always prints "Execution complete" using finally.
2. Write a program to simulate bank withdrawal. Use try-except-else-finally to handle incorrect amount input, and always print a message whether the transaction succeeded or failed.

# Section A: Basic Try-Except

# 1. Divide two numbers, handle ZeroDivisionError

def divide\_numbers():

    try:

        a = float(input("Enter numerator: "))

        b = float(input("Enter denominator: "))

        result = a / b

        print("Result:", result)

    except ZeroDivisionError:

        print("Error: Cannot divide by zero.")

# 2. Convert string to integer, handle ValueError

def string\_to\_int():

    try:

        s = input("Enter a number: ")

        num = int(s)

        print("Integer:", num)

    except ValueError:

        print("Error: Invalid number.")

# 3. Addition with input validation

def add\_numbers():

    try:

        x = float(input("Enter first number: "))

        y = float(input("Enter second number: "))

        print("Sum:", x + y)

    except ValueError:

        print("Error: Invalid input type.")

# 4. List index access, handle IndexError

def list\_element\_by\_index():

    lst = [10, 20, 30, 40, 50]

    try:

        idx = int(input("Enter index: "))

        print("Element:", lst[idx])

    except IndexError:

        print("Error: Index out of range.")

# Section B: Try-Except-Else

# 5. Square of a number with try-except-else

def square\_number():

    try:

        n = int(input("Enter a number: "))

    except ValueError:

        print("Error: Invalid input.")

    else:

        print("Square:", n \* n)

# 6. Open and read file with try-except-else

def read\_file():

    filename = input("Enter filename: ")

    try:

        f = open(filename, 'r')

    except FileNotFoundError:

        print("Error: File not found.")

    else:

        print("File contents:\n", f.read())

        f.close()

# 7. Convert number to binary with try-except-else

def number\_to\_binary():

    try:

        n = int(input("Enter a number: "))

    except ValueError:

        print("Error: Invalid input.")

    else:

        print("Binary:", bin(n))

# Section C: Try-Finally

# 8. Open file and ensure close with try-finally

def open\_file\_finally():

    f = None

    try:

        f = open("sample.txt", "r")

        print(f.read())

    finally:

        if f:

            f.close()

            print("File closed.")

# 9. Simulate login with finally log message

def login\_simulation():

    try:

        username = input("Username: ")

        password = input("Password: ")

        if username == "admin" and password == "1234":

            print("Login successful.")

        else:

            print("Login failed.")

    finally:

        print("Login attempt logged.")

# 10. Divide numbers with try-except-finally

def divide\_with\_cleanup():

    try:

        a = float(input("Enter numerator: "))

        b = float(input("Enter denominator: "))

        print("Result:", a / b)

    except Exception as e:

        print("Error:", e)

    finally:

        print("Clean-up complete.")

# Section D: Combined Exception Handling

# 11. Handle multiple exceptions and always print finally

def multiple\_exceptions():

    try:

        a = float(input("Enter numerator: "))

        b = float(input("Enter denominator: "))

        print("Result:", a / b)

    except ZeroDivisionError:

        print("Error: Division by zero.")

    except ValueError:

        print("Error: Invalid input.")

    finally:

        print("Execution complete.")

# 12. Bank withdrawal with try-except-else-finally

def bank\_withdrawal():

    balance = 1000

    try:

        amount = float(input("Enter withdrawal amount: "))

        if amount > balance:

            print("Insufficient funds.")

        elif amount < 0:

            print("Invalid amount.")

        else:

            balance -= amount

    except ValueError:

        print("Error: Invalid input.")

    else:

        print("Withdrawal successful. Remaining balance:", balance)

    finally:

        print("Transaction processed.")

# Uncomment to test individual functions

# divide\_numbers()

# string\_to\_int()

# add\_numbers()

# list\_element\_by\_index()

# square\_number()

# read\_file()

# number\_to\_binary()

# open\_file\_finally()

# login\_simulation()

# divide\_with\_cleanup()

# multiple\_exceptions()

# bank\_withdrawal()