# 1. You are developing a program to store geographical coordinates (latitude and longitude)

# of various cities. The coordinates should be immutable so that they don'39;t get accidentally

# changed.

# Question: How would you use tuples to store the coordinates for five cities? Demonstrate how

# you would access the longitude of the third city in the list.

cities = [

    (40.7128, -74.0060),  # New York

    (34.0522, -118.2437),  # Los Angeles

    (37.7749, -122.4194),  # San Francisco

    (41.8781, -87.6298),  # Chicago

    (29.7604, -95.3698)   # Houston

]

third\_city\_longitude = cities[4][1]

print(third\_city\_longitude)

**OUTPUT : --**



# 2. You are working with employee data in a system where each employee has an ID, name,

# and department. You have a tuple of employee information (101, '39;John Doe'39;,

# '39;Engineering'39;).

# Question: How would you unpack the tuple into separate variables for the ID, name, and

# department? Demonstrate this in Python.

# Define the tuple of employee information

employee\_info = (101, 'John Doe', 'Engineering')

# Unpack the tuple into separate variables

employee\_id, name, department = employee\_info

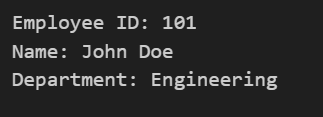
# Print the values of the separate variables

print("Employee ID:", employee\_id)

print("Name:", name)

print("Department:", department)

**OUTPUT : --**



# 3. You are working with a dataset where each record contains information about a student’s

# grades across multiple subjects. For example, the dataset looks like this:

# ('39;John Doe'39;, ('39;Math'39;, 85), ('39;Science'39;, 90), ('39;English'39;, 78)).

# Question: How would you access John Doe’s grade in Science using Python indexing?

def get\_grade(student\_record, subject):

    # Extract the subject-grade pairs

    subject\_grades = student\_record[1:]

    # Iterate over the subject-grade pairs to find the index of the subject

    for i, (sub, grade) in enumerate(subject\_grades):

        if sub == subject:

            # Return the grade if the subject is found

            return grade

    # Return None if the subject is not found

    return None

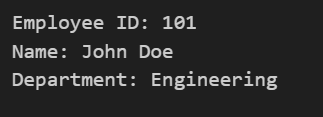
student\_record = ('John Doe', ('Math', 85), ('Science', 90), ('English', 78))

subject = 'Science'

grade = get\_grade(student\_record, subject)

print(f"John Doe's grade in {subject} is: {grade}")

**OUTPUT : --**

****

# 4. You are tasked with keeping track of sports team scores in a tournament. Each team has a

# name and a score.

# Task: Create a list of tuples to store the team names and their corresponding scores. Write a

# Python function to:

# • Add a new team&#39;s score.

# • Update the score of an existing team.

# • Find and display the team with the highest score.

# Initialize the list of teams and scores

teams = []

def add\_team(name, score):

    """Add a new team's score"""

    teams.append((name, score))

def update\_score(name, new\_score):

    """Update the score of an existing team"""

    for i, (team\_name, \_) in enumerate(teams):

        if team\_name == name:

            teams[i] = (name, new\_score)

            break

def find\_highest\_score():

    """Find and display the team with the highest score"""

    highest\_score\_team = max(teams, key=lambda x: x[1])

    print(f"The team with the highest score is {highest\_score\_team[0]} with a score of {highest\_score\_team[1]}")

# Example usage:

add\_team("Team A", 10)

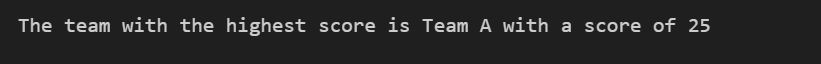
add\_team("Team B", 20)

add\_team("Team C", 15)

update\_score("Team A", 25)

find\_highest\_score()

**OUTPUT : --**

****

# 5. You are developing a flight information system where each flight has a flight number,

# destination, and departure time.

# Task: Write a Python function that:

# • Adds a new flight to the list.

# • Sorts the flights by departure time.

# • Finds all flights headed to a particular destination

class Flight:

    def \_\_init\_\_(self, flight\_number, destination, departure\_time):

        self.flight\_number = flight\_number

        self.destination = destination

        self.departure\_time = departure\_time

class FlightInformationSystem:

    def \_\_init\_\_(self):

        self.flights = []

    def add\_flight(self, flight\_number, destination, departure\_time):

        new\_flight = Flight(flight\_number, destination, departure\_time)

        self.flights.append(new\_flight)

        self.flights.sort(key=lambda x: x.departure\_time)

    def find\_flights\_by\_destination(self, destination):

        return [flight for flight in self.flights if flight.destination == destination]

# Example usage:

flight\_system = FlightInformationSystem()

flight\_system.add\_flight("UA101", "New York", "08:00")

flight\_system.add\_flight("UA102", "Los Angeles", "09:00")

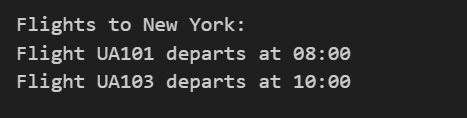
flight\_system.add\_flight("UA103", "New York", "10:00")

print("Flights to New York:")

for flight in flight\_system.find\_flights\_by\_destination("New York"):

    print(f"Flight {flight.flight\_number} departs at {flight.departure\_time}")

**OUTPUT : --**

****

# 6. Write a Python function that takes a string and returns a compressed version of the string.

# The compression should represent consecutive repeated characters as the character

# followed by the number of repetitions. If the compressed string is not shorter than the

# original, return the original string instead.

# Input:

# • A string s containing only lowercase letters (e.g., "aaabbccccde").

# Output:

# • The compressed version of the string if it&#39;s shorter, otherwise the original string.

# Examples:

# • Input: "aaabbccccde"

# Output: "a3b2c4de"

# • Input: "abcd"

# Output: "abcd"

# • Input: "aabcccccaaa"

# Output: "a2bc5a3"

def compress\_string(s):

    """

    Compress a string by representing consecutive repeated characters as the character

    followed by the number of repetitions. If the compressed string is not shorter than

    the original, return the original string instead.

    Args:

        s (str): The input string containing only lowercase letters.

    Returns:

        str: The compressed version of the string if it's shorter, otherwise the original string.

    """

    if not s:

        return s  # Return the original string if it's empty

    compressed = []

    count = 1  # Start with a count of 1 for the first character

    # Iterate over the string starting from the second character

    for i in range(1, len(s)):

        if s[i] == s[i - 1]:

            count += 1  # Increment count for consecutive characters

        else:

            # Append the previous character and its count to the compressed list

            compressed.append(s[i - 1] + str(count))

            count = 1  # Reset count for the new character

    # Append the last character and its count

    compressed.append(s[-1] + str(count))

    compressed\_str = "".join(compressed)  # Join the list into a string

    # Return compressed string if it's shorter; otherwise, return the original

    return compressed\_str if len(compressed\_str) < len(s) else s

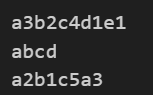
# Example usages

print(compress\_string("aaabbccccde"))  # Output: "a3b2c4de"

print(compress\_string("abcd"))          # Output: "abcd"

print(compress\_string("aabcccccaaa"))   # Output: "a2bc5a3"

**OUTPUT : --**

****