```
# 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 : --

```
-95.3698
```

```
# 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 : --

```
Employee ID: 101
Name: John Doe
Department: Engineering
```

```python
# 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 : --

```
Employee ID: 101
Name: John Doe
Department: Engineering
```

```python
# 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 : --

```
The team with the highest score is Team A with a score of 25
```

```python
# 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 : --

```
Flights to New York:
Flight UA101 departs at 08:00
Flight UA103 departs at 10:00
```

```python
# 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: "aabccccccaaa"
# 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 : --

```
a3b2c4d1e1
abcd
a2b1c5a3
```