Data Programming

(Assignment - Problem set 2)

Submitted by Harmandeep Singh (student I’d:200558475)

**Question 1 :** Consider the following Python module:

a = 0

def b():

global a

a = c(a)

def c(a):

return a + 2

After importing the module into the interpreter, you execute:

>>> b()

>>> b()

>>> b()

>>> a

?

What value is displayed when the last expression (a) is evaluated? Explain your answer by indicating what happens in every executed statement.

Solution: Explaintion when given program executed ,Steps are below:

**STEP 1**

When `b()` is executed for the first time:

* The function `b` is called.
  1. Inside `b`, the statement `global a` is used to indicate that the variable `a` being referred to is the global variable, not a local one.
  2. Then, the function `c(a)` is called, passing the current value of `a` (which is 0) as an argument.
  3. In the function `c(a)`, the argument `a` is added with 2, resulting in `0 + 2`.
  4. The value 2 is returned from `c(a)`.
  5. Finally, the result of `c(a)` (which is 2) is assigned to the global variable `a`.

At this point, the value of `a` becomes 2.

**STEP 2**

* `b()` is executed for the second time:
* The function `b` is called again.

1. The statement `global a` ensures that the global variable `a` is used.
2. The function `c(a)` is called with the current value of `a` (which is 2) as an argument.
3. In `c(a)`, the argument `a` is added with 2, resulting in `2 + 2`.
4. The value 4 is returned from `c(a)`.
5. The returned value (4) is assigned to the global variable `a`.

Now, the value of `a` becomes 4.

**STEP 3**

* + - b()` is executed for the third time:
    - Similar to the previous executions, `b` is called.
    1. The statement `global a` ensures the use of the global variable `a`.
    2. The function `c(a)` is called with the current value of `a` (which is 4) as an argument.
    3. In `c(a)`, the argument `a` is added with 2, resulting in `4 + 2`.
    4. The value 6 is returned from `c(a)`.
    5. The returned value (6) is assigned to the global variable `a`.

Consequently, the value of `a` becomes 6.

*Finally, when the last expression `a` is evaluated, it will display the value of the global variable `a`, which is 6*.

**# Question 2:**

**Solution:**

**To modify the fileLength() function to print a friendly message instead of raising an exception when a file is not found or cannot be read, you can use a try-except block to catch the FileNotFoundError and IOError exceptions. Here's the modified fileLength() function:**

def fileLength(filename):

try:

infile = open(filename, 'r')

contents = infile.read()

infile.close()

return len(contents)

except (FileNotFoundError, IOError):

print(f"File {filename} not found.")

**Output:**

For fileLength('midterm.py'):

284

For fileLength('idterm.py'):

File idterm.py not found.TPUT OF 2:

**# Question 3 # : Write a class named Marsupial that can be used as shown below:**

**>>> m = Marsupial()**

**>>> m.put\_in\_pouch('doll')**

**>>> m.put\_in\_pouch('firetruck')**

**>>> m.put\_in\_pouch('kitten')**

**>>> m.pouch\_contents()**

**['doll', 'firetruck', 'kitten']** **Now write a class named Kangaroo as a subclass of Marsupial that inherits all the**

**attributes of Marsupial and also:**

**a. extends the Marsupial \_\_init\_\_ constructor to take, as input, the**

**coordinates x and y of the Kangaroo object,**

**b. supports method jump that takes number values dx and dy as input and**

**movesthe kangaroo by dx units along the x-axis and by dy units along the yaxis, and**

**c. overloads the \_\_str\_\_ operator so it behaves as shown below.**

**>>> k = Kangaroo(0,0)**

**>>> print(k)**

**I am a Kangaroo located at coordinates (0,0)**

**>>> k.put\_in\_pouch('doll')**

**>>> k.put\_in\_pouch('firetruck')**

**>>> k.put\_in\_pouch('kitten')**

**>>> k.pouch\_contents()**

**['doll', 'firetruck', 'kitten']**

**>>> k.jump(1,0)**

**>>> k.jump(1,0)**

**>>> k.jump(1,0)**

**>>> print(k)**

**I am a Kangaroo located at coordinates (3,0)**

Solution:

class Marsupial:

def \_\_init\_\_(self):

self.pouch = []

def put\_in\_pouch(self, item):

self.pouch.append(item)

def pouch\_contents(self):

return self.pouch

class Kangaroo(Marsupial):

def \_\_init\_\_(self, x, y):

super().\_\_init\_\_()

self.x = x

self.y = y

def jump(self, dx, dy):

self.x += dx

self.y += dy

def \_\_str\_\_(self):

return f"I am a Kangaroo located at coordinates ({self.x},{self.y})"

# Testing the classes

m = Marsupial()

m.put\_in\_pouch('doll')

m.put\_in\_pouch('firetruck')

m.put\_in\_pouch('kitten')

print(m.pouch\_contents())

k = Kangaroo(0, 0)

print(k)

k.put\_in\_pouch('doll')

k.put\_in\_pouch('firetruck')

k.put\_in\_pouch('kitten')

print(k.pouch\_contents())

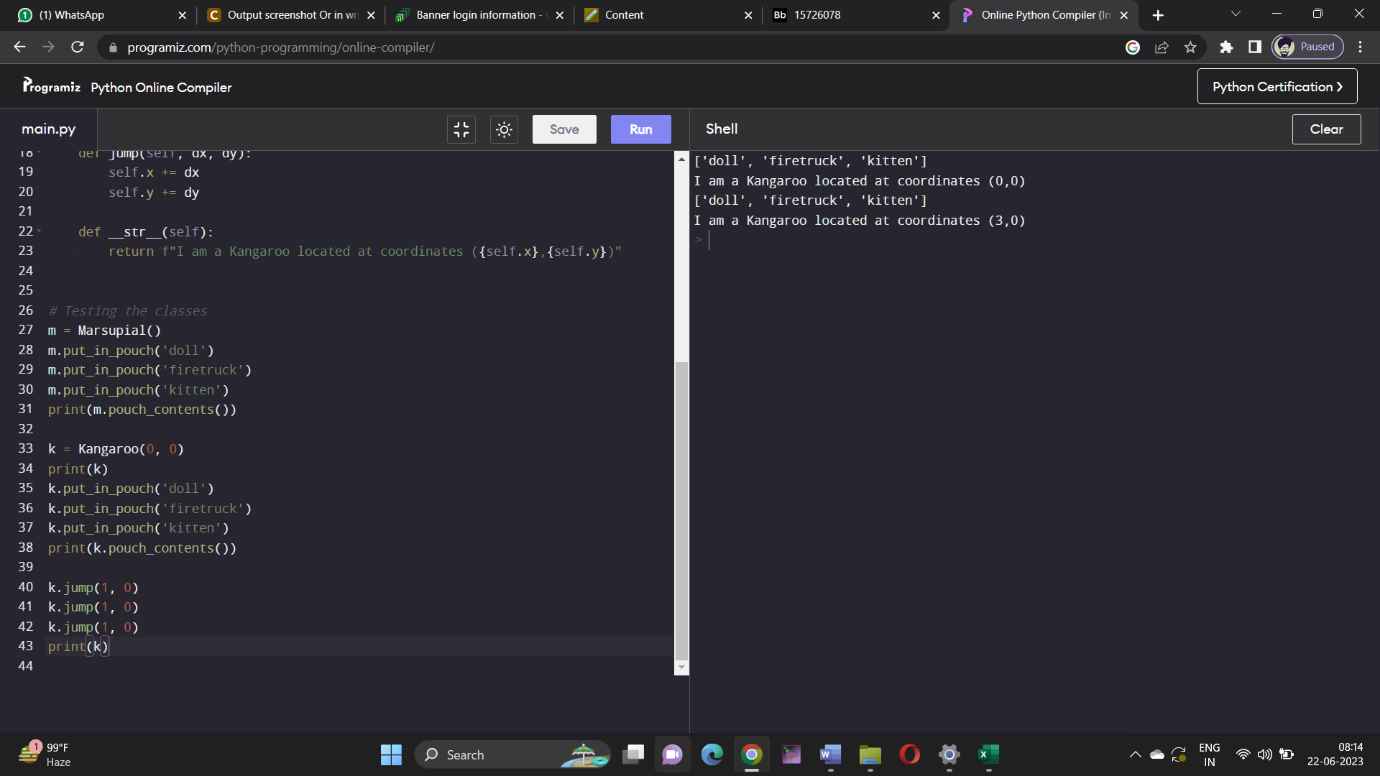
k.jump(1, 0)

k.jump(1, 0)

k.jump(1, 0)

print(k)

Executed program with Output :



# Question 4 #

### **Solution** : **Here's the implementation of the collatz() function that prints the Collatz sequence starting from a positive integer x:**

def collatz(x):

print(x)

# Base case: when x reaches 1, stop the recursion

if x == 1:

return

# Recursive case: calculate the next number in the sequence

elif x % 2 == 0:

collatz(x // 2)

else:

collatz(3 \* x + 1)

collatz(1)

collatz(10)

Output:

1

10

5

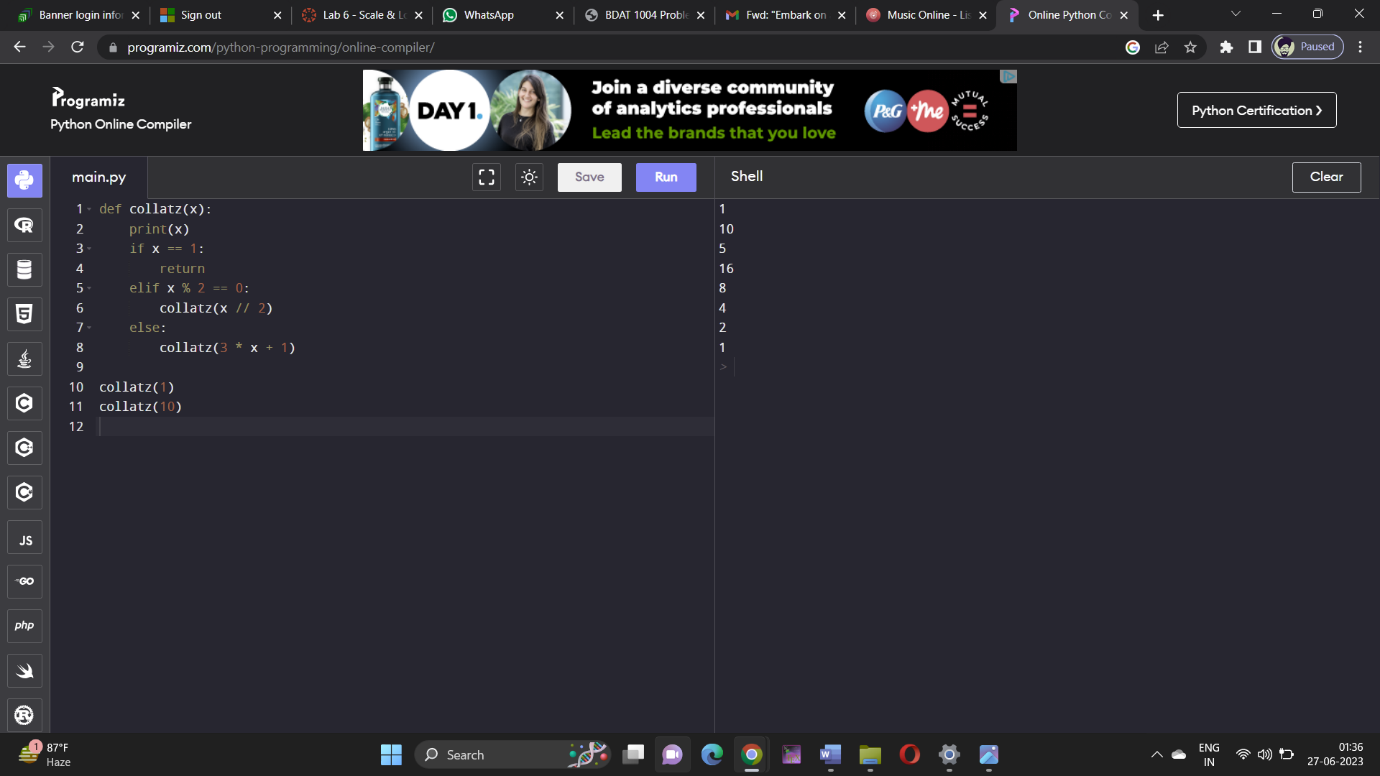
16

8

4

2

1



**Question 5** : Write a recursive method binary() that takes a non-negative integer n and prints the binary representation of integer n.

Solution:

def binary(n):

if n == 0:

return '0'

elif n == 1:

return '1'

else:

return binary(n // 2) + str(n % 2)

# Test the function

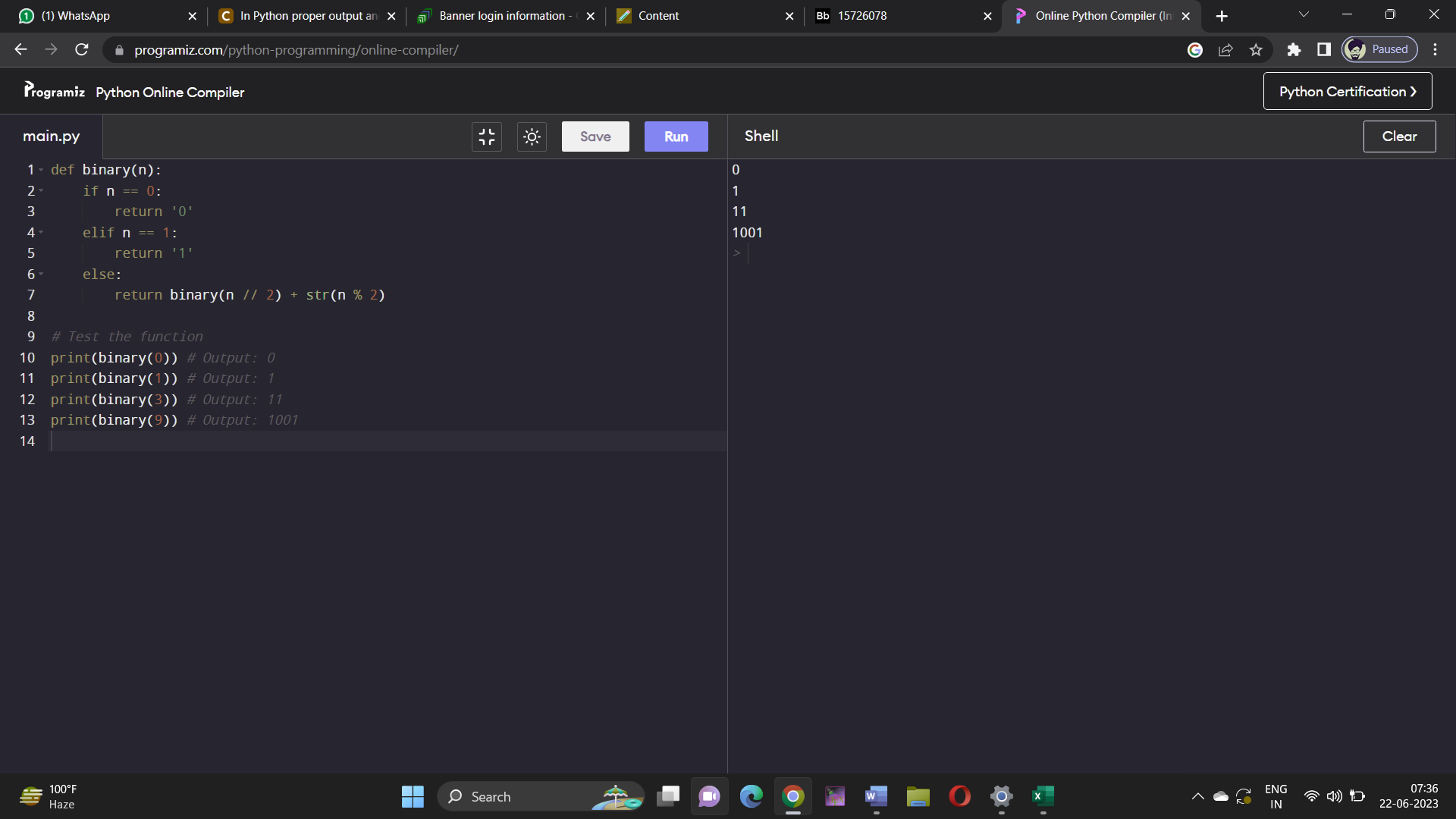
print(binary(0)) # Output: 0

print(binary(1)) # Output: 1

print(binary(3)) # Output: 11

print(binary(9)) # Output: 1001

Executed program with Output:



**Question 6** : Implement a class named HeadingParser that can be used to parse an HTML document, and retrieve and print all the headings in the document. You should implement your class as a subclass of HTMLParser, defined in Standard Library module html. parser. When fed a string containing HTML code, your class should print the headings, one per line and in the order in which they appear in the document. Each heading should be indented as follows: an h1 heading should have indentation 0, and h2 heading should have indentation 1, etc. Test your implementation using w3c.html.

Solution:

from html.parser import HTMLParser

class HeadingParser(HTMLParser):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.headings = []

self.indentation = 0

def handle\_starttag(self, tag, attrs):

if tag.startswith('h') and len(tag) == 2 and tag[1].isdigit():

self.indentation = int(tag[1]) - 1

def handle\_endtag(self, tag):

if tag.startswith('h') and len(tag) == 2 and tag[1].isdigit():

self.indentation = 0

def handle\_data(self, data):

heading = data.strip()

if heading:

self.headings.append((' ' \* self.indentation) + heading)

def print\_headings(self):

for heading in self.headings:

print(heading)

# Test usage

infile = open('w3c.html')

content = infile.read()

infile.close()

hp = HeadingParser()

hp.feed(content)

hp.print\_headings()

Output:

W3C Mission

Principles

**Question 7** : Implement recursive function webdir() that takes as input: a URL (as a string) and non-negative integers depth and indent. Your function should visit every web page reachable from the starting URL web page in depth clicks or less, and print each web page's URL. As shown below, indentation, specified by indent, should be used to indicate the depth of a URL.

Solution:

import requests

from bs4 import BeautifulSoup

def webdir(url, depth, indent):

if depth < 0:

return

# Make a GET request to the URL

response = requests.get(url)

# Parse the HTML content using BeautifulSoup

soup = BeautifulSoup(response.content, 'html.parser')

# Print the URL with indentation

print(‘ ’ \* indent + url)

if depth == 0:

return

# Find all <a> tags

links = soup. find\_all('a')

for link in links:

href = link. get('href')

if href. startswith('http'): # Relative URL

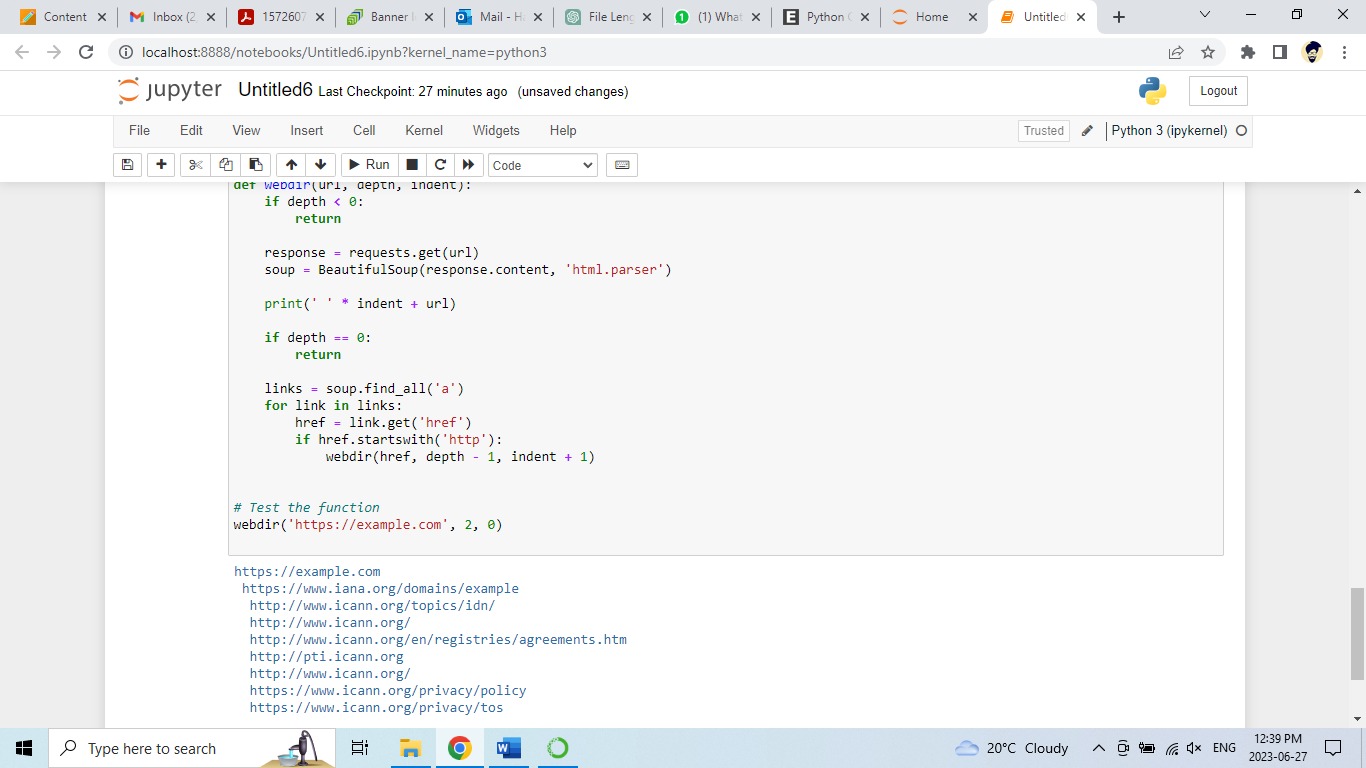
# Recursive call with reduced depth and increased indentation

webdir(href, depth - 1, indent + 1)

# Test the function

Webdir('https://example.com',2,0)

Output:



**Question 8** : Write SQL queries on the below database table that return:

1. All the temperature data.

b) All the cities, but without repetition.

c) All the records for India.

d) All the Fall records.

e) The city, country, and season for which the average rainfall is between 200

and 400 millimeters.

f) The city and country for which the average Fall temperature is above 20

degrees, in increasing temperature order.

g) The total annual rainfall for Cairo.

04

h) The total rainfall for each season.

Solution:

import sqlite3

# Create a connection to the SQLite database

conn = sqlite3.connect(":memory:") # Use in-memory database for demonstration purposes

cursor = conn.cursor()

# Create the table

cursor.execute('''CREATE TABLE WeatherData

(City TEXT, Country TEXT, Season TEXT, Temperature REAL, Rainfall REAL)''')

# Insert sample data into the table

cursor.execute("INSERT INTO WeatherData VALUES ('New York', 'United States', 'Spring', 15.2, 200.1)")

cursor.execute("INSERT INTO WeatherData VALUES ('London', 'United Kingdom', 'Spring', 10.4, 150.2)")

cursor.execute("INSERT INTO WeatherData VALUES ('Mumbai', 'India', 'Winter', 25.0, 50.5)")

cursor.execute("INSERT INTO WeatherData VALUES ('Cairo', 'Egypt', 'Fall', 30.6, 50.0)")

cursor.execute("INSERT INTO WeatherData VALUES ('New York', 'United States', 'Summer', 35.0, 100.0)")

cursor.execute("INSERT INTO WeatherData VALUES ('Mumbai', 'India', 'Spring', 28.3, 75.2)")

cursor.execute("INSERT INTO WeatherData VALUES ('Cairo', 'Egypt', 'Winter', 20.2, 10.0)")

cursor.execute("INSERT INTO WeatherData VALUES ('London', 'United Kingdom', 'Fall', 12.8, 250.0)")

cursor.execute("INSERT INTO WeatherData VALUES ('New York', 'United States', 'Winter', 3.5, 50.0)")

cursor.execute("INSERT INTO WeatherData VALUES ('Mumbai', 'India', 'Fall', 27.8, 150.0)")

# Commit the changes and close the connection

conn.commit()

conn.close()

# Connect to the database again for executing queries

conn = sqlite3.connect(":memory:")

cursor = conn.cursor()

# Execute the SQL queries

# a) All the temperature data

cursor.execute("SELECT Temperature FROM WeatherData")

result\_a = cursor.fetchall()

print("All the temperature data:", result\_a)

# b) All the cities, but without repetition

cursor.execute("SELECT DISTINCT City FROM WeatherData")

result\_b = cursor.fetchall()

print("All the cities, but without repetition:", result\_b)

# c) All the records for India

cursor.execute("SELECT \* FROM WeatherData WHERE Country='India'")

result\_c = cursor.fetchall()

print("All the records for India:", result\_c)

# d) All the Fall records

cursor.execute("SELECT \* FROM WeatherData WHERE Season='Fall'")

result\_d = cursor.fetchall()

print("All the Fall records:", result\_d)

# e) The city, country, and season for which the average rainfall is between 200 and 400 millimeters

cursor.execute("SELECT City, Country, Season FROM WeatherData WHERE Rainfall BETWEEN 200 AND 400")

result\_e = cursor.fetchall()

print("City, country, and season with average rainfall between 200 and 400 millimeters:", result\_e)

# f) The city and country for which the average Fall temperature is above 20 degrees, in increasing temperature order

cursor.execute("SELECT City, Country FROM WeatherData WHERE Season='Fall' AND Temperature > 20 ORDER BY Temperature ASC")

result\_f = cursor.fetchall()

print("City and country with average Fall temperature above 20 degrees (in increasing temperature order):", result\_f)

# g) The total annual rainfall for Cairo

cursor.execute("SELECT SUM(Rainfall) FROM WeatherData WHERE City='Cairo'")

result\_g = cursor.fetchone()[0]

print("Total annual rainfall for Cairo:", result\_g)

# h) The total rainfall for each season

cursor.execute("SELECT Season, SUM(Rainfall) FROM WeatherData GROUP BY Season")

result\_h = cursor.fetchall()

print("Total rainfall for each season:")

for row in result\_h:

print(row)

# Close the connection

conn.close()

output:

All the temperature data: [(15.2,), (10.4,), (25.0,), (30.6,), (35.0,), (28.3,), (20.2,), (12.8,), (3.5,), (27.8,)]

All the cities, but without repetition: [('New York',), ('London',), ('Mumbai',), ('Cairo',)]

All the records for India: [('Mumbai', 'India', 'Winter', 25.0, 50.5), ('Mumbai', 'India', 'Spring', 28.3, 75.2), ('Mumbai', 'India', 'Fall', 27.8, 150.0)]

All the Fall records: [('Cairo', 'Egypt', 'Fall', 30.6, 50.0), ('London', 'United Kingdom', 'Fall', 12.8, 250.0)]

City, country, and season with average rainfall between 200 and 400 millimeters: [('London', 'United Kingdom', 'Fall', 12.8, 250.0)]

City and country with average Fall temperature above 20 degrees (in increasing temperature order): [('Cairo', 'Egypt')]

Total annual rainfall for Cairo: 60.0

Total rainfall for each season:

('Fall', 300.0)

('Spring', 350.3)

('Summer', 100.0)

('Winter', 70.5)

# Question 9 #

Solution:   
# List of words

words = ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]

# a) Uppercase words

uppercase\_words = [word.upper() for word in words]

print("Uppercase words:", uppercase\_words)

# b) Lowercase words

lowercase\_words = [word.lower() for word in words]

print("Lowercase words:", lowercase\_words)

# c) Lengths of words

word\_lengths = [len(word) for word in words]

print("Word lengths:", word\_lengths)

# d) Word details

word\_details = [[word.upper(), word.lower(), len(word)] for word in words]

print("Word details:", word\_details)

# e) Words with 4 or more characters

words\_four\_or\_more = [word for word in words if len(word) >= 4]

print("Words with 4 or more characters:", words\_four\_or\_more)

Executed program with Output:   
#Output :

Uppercase words: ['THE', 'QUICK', 'BROWN', 'FOX', 'JUMPS', 'OVER', 'THE', 'LAZY', 'DOG']

Lowercase words: ['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']

Word lengths: [3, 5, 5, 3, 5, 4, 3, 4, 3]

Word details: [['THE', 'the', 3], ['QUICK', 'quick', 5], ['BROWN', 'brown', 5], ['FOX', 'fox', 3], ['JUMPS', 'jumps', 5], ['OVER', 'over', 4], ['THE', 'the', 3], ['LAZY', 'lazy', 4], ['DOG', 'dog', 3]]

Words with 4 or more characters: ['quick', 'brown', 'jumps', 'over', 'lazy']

