**WEEK 3:**

**1. Write a program that uses a 'while' loop along with a found flag to search through a list of powers of 2. Your program should identify the value corresponding to 2 raised to the fifth power (32).**

powerof2=[2\*\*i for i in range(10)]

found = False

index=0

target\_value=32

while index < len(powerof2):

if powerof2[index] == target\_value:

found = True

break

index += 1

if found :

print(f"The value {target\_value} was found at index {index}.")

else:

print(f"The value {target\_value} was not found in the list.")

**OUTPUT:**

The value 32 was found at index 5.

**2. Write a program to calculate the distance covered by a robot after a series of movements on a plane. A robot starts at the origin point (0,0) in a 2D plane.   
It can move in four directions: UP, DOWN, LEFT, and RIGHT.   
The robot's movements are defined as follows:   
a. UP - 5   
b. DOWN - 3   
c. LEFT - 3   
d. RIGHT - 2   
The number following each direction indicates the number of steps taken in that direction. Implement a program that:   
a. Tracks the robot's position after the given sequence of movements.   
b. Calculates the distance between the robot's final position and the origin (0,0).   
c. If the calculated distance is a floating-point number, round it to the nearest integer.   
  
Constraints:   
a. You are not allowed to use any external packages or libraries.   
b. Only basic Python functionalities should be used.**

movement\_distances = {

'UP' : 5,

'DOWN' : 3,

'LEFT' : 3,

'RIGHT' : 2

}

x,y = 0,0

movements = [

('UP',2),

('RIGHT',3),

('DOWN',1),

('LEFT',1)

]

for direction , steps in movements:

if direction == 'UP':

y+= movement\_distances['UP']\*steps

elif direction == 'DOWN' :

y-=movement\_distances['DOWN']\*steps

elif direction == 'LEFT' :

x-=movement\_distances['LEFT']\*steps

elif direction == 'RIGHT':

x += movement\_distances["RIGHT"]\*steps

distance = (x\*\*2 + y\*\*2) \*\* 0.5

distance\_rounded = round(distance)

print(f"Final Position: ({x},{y}")

print(f"Distance from origin (rounded): {distance\_rounded}")

**OUTPUT:**

Final Position: (3,7

Distance from origin (rounded): 8

**3. Write a python program that can accept two strings as input and print the string with maximum length in console. If two strings have the same length, then the program should print all strings in one line.**

string1 = input("Enter the first string:")

string2 = input("Enter the second string")

if len(string1) > len(string2):

print(string1)

elif len(string1) < len(string2):

print(string2)

else:

print(f"{string1} {string2}")

**OUTPUT:**

Enter the first string:Mayank

Enter the second stringAmishi

Mayank Amishi

**4. Write a program that computes the net amount in a bank account based on a series of transactions provided as input. Instructions: a. Your program should accept a transaction log as input, where each transaction is either a deposit or a withdrawal. b. The transaction log format is as follows: c. D 100 indicates a deposit of 100 units. d. W 200 indicates a withdrawal of 200 units. e. The program should calculate the net amount in the bank account after processing all transactions.**

net\_amount = 0

print("Enter transactions in the format 'D 100' for deposits or 'W 200' for withdrawals.")

print("Type 'done' to finish.")

while True:

transaction = input("Enter transaction: ")

if transaction.lower() == 'done':

break

parts = transaction.split()

if len(parts) != 2:

print("Invalid format. Please enter in the format 'D 100' or 'W 200'.")

continue

action, amount\_str = parts

try:

amount = float(amount\_str)

if action == 'D':

net\_amount += amount

elif action == 'W':

net\_amount -= amount

else:

print("Invalid action. Use 'D' for deposit and 'W' for withdrawal.")

except ValueError:

print("Invalid amount. Please enter a valid number.")

print(f"Net amount in the account: ${net\_amount:.2f}")

**OUTPUT:**

Enter transaction: D 1000

Enter transaction: W 200

Enter transaction: Done

Net amount in the account: $1600.00

**5. Write a python program to implement the binary search which searches an item in a sorted list. (Do not use any Packages)**

def binary\_search(sorted\_list, target):

left = 0

right = len(sorted\_list) - 1

while left <= right:

mid = (left + right) // 2

mid\_value = sorted\_list[mid]

if mid\_value == target:

return mid

elif mid\_value < target:

left = mid + 1

else:

right = mid - 1

return -1

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

sorted\_list = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

target = int(input("Enter the value to search for: "))

result = binary\_search(sorted\_list, target)

if result != -1:

print(f"Value {target} found at index {result}.")

else:

print(f"Value {target} not found in the list.")

**OUTPUT:**

Enter the value to search for: 5

Value 5 found at index 2.

**6. Write a Python 3 program that demonstrates how to calculate the summation of all possible combinations of elements in a list of tuples. Use nested for loops to iterate through the tuples and generate the combinations.**

def calculate\_combinations\_sum(tuples\_list):

if not tuples\_list:

return 0

total\_sum = 0

num\_combinations = 1

for t in tuples\_list:

num\_combinations \*= len(t)

def generate\_combinations(tuples\_list, current\_combination, index):

nonlocal total\_sum

if index == len(tuples\_list):

total\_sum += sum(current\_combination)

return

for element in tuples\_list[index]:

generate\_combinations(tuples\_list, current\_combination + [element], index + 1)

generate\_combinations(tuples\_list, [], 0)

return total\_sum

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

tuples\_list = [(1, 2), (3, 4), (5, 6)]

result = calculate\_combinations\_sum(tuples\_list)

print(f"Total sum of all possible combinations: {result}")

**OUTPUT:**

Total sum of all possible combinations: 84

**7. Write a Python3 program that demonstrates how to convert a tuple into a list, where each element in the list is the succeeding element of the original tuple.**

def convert\_tuple\_to\_succeeding\_list(tup):

return [tup[i + 1] if i + 1 < len(tup) else None for i in range(len(tup))]

# Example usage

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

tup = (1, 2, 3, 4, 5)

result = convert\_tuple\_to\_succeeding\_list(tup)

print(result)

**OUTPUT:**

[2, 3, 4, 5, None]