PIYUSH KUMAR MISHRA

230957212

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)

```python
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]
```