m3_case_study_1

October 22, 2024

0.0.1 Module 3 – OOP Packages Modules Try-Except

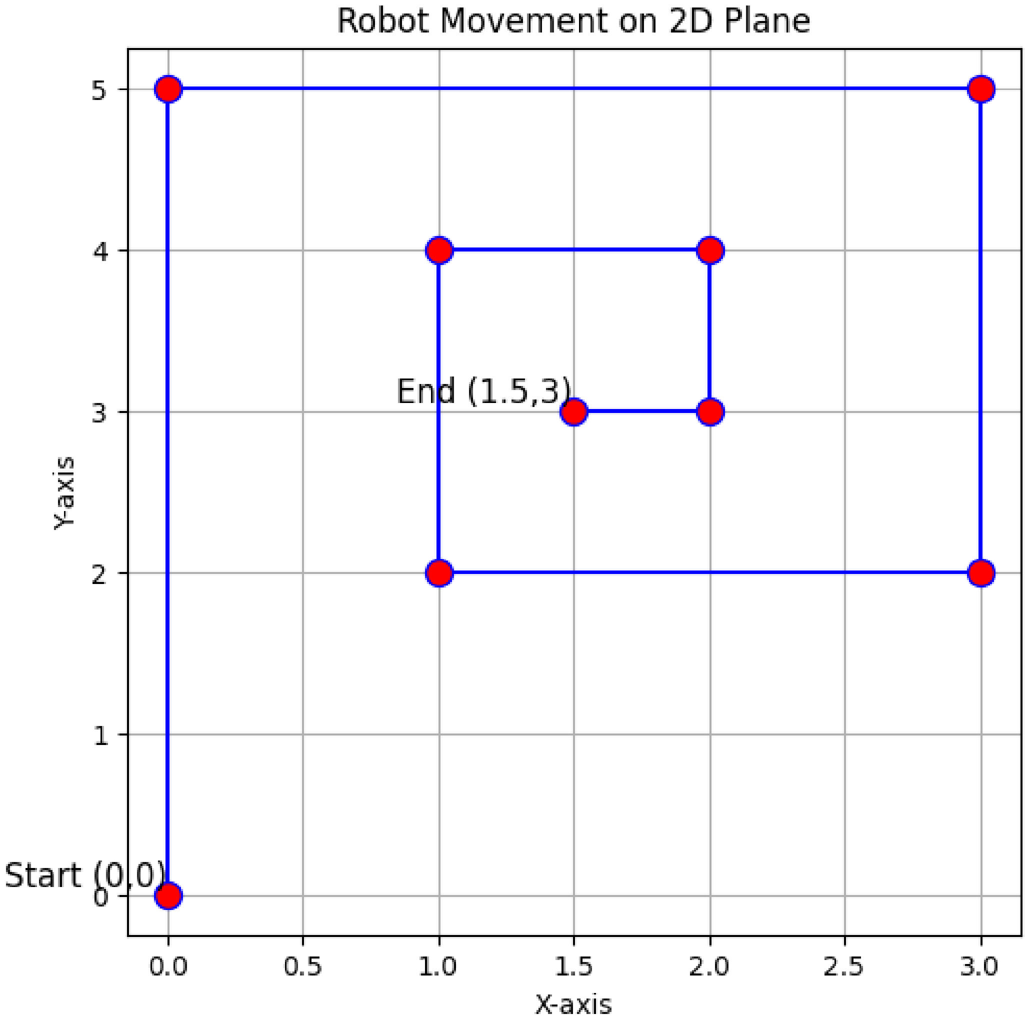
0.1 Case Study – 1

1. A Robot moves in a Plane starting from the origin point (0,0). The robot can move UP, DOWN, LEFT, or RIGHT. The trace of Robot movement is as given following: UP 5 DOWN 3 LEFT 3 RIGHT 2 The numbers after directions are steps. Write a program to compute the distance current position after a sequence of movements. Hint: Use the math module.

[6]:

```python
# Here’s a Python program to calculate the distance of the robot from the origin after a sequence of movements:
import math
import matplotlib.pyplot as plt

def calculate_and_plot_movements(movements):
    """
    This function calculates the final distance of the robot from the origin and
    also plots the path of the robot's movements on a 2D plane.

    Parameters:
    movements (list): A list of tuples where each tuple contains a direction and the number of steps.

    Returns:
    float: The distance from the origin after completing the movements.
    """

    # Starting coordinates at origin (0, 0)
    x, y = 0, 0

    # Lists to store the x and y coordinates for plotting
    x_coords = [x]
    y_coords = [y]

    # Processing each movement in the list
    for direction, steps in movements:
        if direction == 'UP':
            y += steps
        elif direction == 'DOWN':
            y -= steps
        elif direction == 'LEFT':
            x -= steps
        elif direction == 'RIGHT':
            x += steps

        # Append the new coordinates after the movement
        x_coords.append(x)
        y_coords.append(y)

    # Calculate the Euclidean distance from the origin (0, 0)
    distance = math.sqrt(x**2 + y**2)

    # Plotting the movements
    plt.figure(figsize=(6,6))
    plt.plot(x_coords, y_coords, marker='o', color='b', linestyle='-',␣
 ↪markersize=10, markerfacecolor='red')

    # Mark the origin and the final point
    plt.text(0, 0, "Start (0,0)", fontsize=12, verticalalignment='bottom',␣
 ↪horizontalalignment='right')
    plt.text(x_coords[-1], y_coords[-1], f"End␣
 ↪({x_coords[-1]},{y_coords[-1]})", fontsize=12, verticalalignment='bottom',␣
 ↪horizontalalignment='right')

    # Set labels and title
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Robot Movement on 2D Plane')

    # Display the grid
    plt.grid(True)

    # Show the plot
    plt.show()

    return distance

# List of movements: (direction, steps)
movements = [('UP', 5), ('RIGHT', 3), ('DOWN', 3), ('LEFT', 2),('UP', 2),␣
 ↪('RIGHT', 1), ('DOWN', 1), ('LEFT', 0.5) ]

# Calculate the distance and plot the movement
distance = calculate_and_plot_movements(movements)

# Print the final distance
```

```
print(f"The distance from the origin is: {round(distance, 2)} units")
```



```
The distance from the origin is: 3.35 units
```

2. Data of XYZ company is stored in the sorted list. Write a program for searching specific data from that list. Hint: Use if/elif to deal with conditions.

   To search for a specific data point in a sorted list, you can implement binary search as it is more efficient for sorted data. Binary search works by repeatedly dividing the search interval in half, checking if the target element is in the left or right half of the list. This approach is efficient for large, sorted datasets because binary search has a time complexity of O(log n), making it faster than linear search.

[7]:
```
def binary_search(sorted_list, target):
    """
    This function performs binary search to find the target value in a sorted␣
↪list.

    Parameters:
    sorted_list (list): The sorted list in which to search for the target value.
    target (int/str): The value to search for in the list.

    Returns:
    int: The index of the target value if found, else -1.
    """

    # Initialize the starting and ending indices
    low = 0
    high = len(sorted_list) - 1

    # Loop until the range is valid
    while low <= high:
        # Calculate the middle index
        mid = (low + high) // 2

        # Check if the target is at the mid index
        if sorted_list[mid] == target:
            return mid  # Target found

        # If the target is less than the mid element, search in the left half
        elif sorted_list[mid] > target:
            high = mid - 1

        # If the target is greater, search in the right half
        else:
            low = mid + 1

    # Target not found
    return -1

# Example sorted list (XYZ company data)
company_data = [101, 105, 123, 135, 142, 155, 178, 200, 245, 300]

# Input: Asking the user to enter the value to search
target_value = int(input("Enter the value to search for: "))

# Perform the search
result_index = binary_search(company_data, target_value)

# Output the result
if result_index != -1:
    print(f"Value {target_value} found at index {result_index}.")
```

```
else:
    print(f"Value {target_value} not found in the list.")
```

```
Enter the value to search for:  142

Value 142 found at index 4.
```

3. Weather forecasting organization wants to show whether is it day or night. So, write a program for such an organization to find whether is it dark outside or not. Hint: Use the time module.

   To determine whether it is currently day or night, you can use Python's time module to check the current hour and compare it to a typical day-night range (e.g., daytime might be between 6 AM and 6 PM).

[10]:
```
import time

def is_it_dark():
    """
    This function checks the current time and determines if it's day or night␣
 ↪based on typical time ranges.

    Returns:
    str: "It's day." or "It's night."
    """

    # Get the current hour (24-hour format)
    current_hour = time.localtime().tm_hour

    # Define day time between 6 AM (06:00) and 6 PM (18:00)
    if 6 <= current_hour < 18:
        return "It's day."
    else:
        return "It's night."

# Example usage
result = is_it_dark()
print(result)
```

```
It's night.
```

[1]:
```
# 4.Write a program to find the distancebetween two locations when their␣
 ↪latitude and longitudes are given.
# Hint: Use the math module.

import math

# Function to calculate the distance between two locations
def haversine(lat1, lon1, lat2, lon2):
    # Radius of the Earth in kilometers
    R = 6371.0

    # Convert latitude and longitude from degrees to radians
    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)

    # Difference in coordinates
    dlat = lat2_rad - lat1_rad
    dlon = lon2_rad - lon1_rad

    # Haversine formula
    a = math.sin(dlat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.
 ↪sin(dlon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    # Distance in kilometers
    distance = R * c

    return distance

# Input coordinates of the two locations
lat1 = float(input("Enter the latitude of the first location: "))
lon1 = float(input("Enter the longitude of the first location: "))
lat2 = float(input("Enter the latitude of the second location: "))
lon2 = float(input("Enter the longitude of the second location: "))

# Calculate the distance
distance = haversine(lat1, lon1, lat2, lon2)

# Print the distance
print(f"The distance between the two locations is {distance:.2f} kilometers.")
```

```
Enter the latitude of the first location:  52.2296756
Enter the longitude of the first location:  21.0122287
Enter the latitude of the second location:  41.8919300
Enter the longitude of the second location:  12.5113300

The distance between the two locations is 1315.51 kilometers.
```

[2]:

```
# 5. Design software for bank systems. There should be options like cash␣
 ↪withdrawal, cash credit, and a change password.
# According to user input, the software should provide the required output.
#Hint: Use if else statements and functions.

# Bank system software
```

```python
# Simulated database for user details
user_balance = 10000  # Initial balance
user_password = "1234"  # Initial password

# Function to handle cash withdrawal
def cash_withdrawal():
    global user_balance
    amount = float(input("Enter the amount to withdraw: "))
    if amount > user_balance:
        print("Insufficient balance!")
    else:
        user_balance -= amount
        print(f"Withdrawal successful! Your new balance is: {user_balance}")

# Function to handle cash credit (deposit)
def cash_credit():
    global user_balance
    amount = float(input("Enter the amount to deposit: "))
    user_balance += amount
    print(f"Deposit successful! Your new balance is: {user_balance}")

# Function to handle password change
def change_password():
    global user_password
    current_password = input("Enter your current password: ")
    if current_password == user_password:
        new_password = input("Enter your new password: ")
        confirm_password = input("Confirm your new password: ")
        if new_password == confirm_password:
            user_password = new_password
            print("Password changed successfully!")
        else:
            print("Passwords do not match!")
    else:
        print("Incorrect current password!")

# Main function to display options and handle user input
def bank_system():
    print("Welcome to the bank system!")
    while True:
        print("\nPlease select an option:")
        print("1. Cash Withdrawal")
        print("2. Cash Credit")
        print("3. Change Password")
        print("4. Exit")
        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            cash_withdrawal()
        elif choice == '2':
            cash_credit()
        elif choice == '3':
            change_password()
        elif choice == '4':
            print("Thank you for using the bank system. Goodbye!")
            break
        else:
            print("Invalid choice! Please select a valid option.")

# Run the bank system
bank_system()
```

```
Welcome to the bank system!

Please select an option:
1. Cash Withdrawal
2. Cash Credit
3. Change Password
4. Exit

Enter your choice (1-4):  1
Enter the amount to withdraw:  500

Withdrawal successful! Your new balance is: 9500.0

Please select an option:
1. Cash Withdrawal
2. Cash Credit
3. Change Password
4. Exit

Enter your choice (1-4):  3
Enter your current password:  1234
Enter your new password:  5678
Confirm your new password:  5678

Password changed successfully!

Please select an option:
1. Cash Withdrawal
2. Cash Credit
3. Change Password
4. Exit

Enter your choice (1-4):  4
```

```
Thank you for using the bank system. Goodbye!
```

[3]:

```
# 6. Write a program that will find all such numbers which are divisible by 7␣
 ↪but are not a multiple of 5, between 2000 and 3200 (both included).
# The numbers obtained should be printed in a comma-separated sequence on a␣
 ↪single line.

# Function to find and print numbers divisible by 7 but not a multiple of 5
def find_numbers():
    result = []

    # Loop through the range from 2000 to 3200
    for num in range(2000, 3201):
        if num % 7 == 0 and num % 5 != 0:
            result.append(str(num))

    # Join the result list with commas and print the final output
    print(", ".join(result))

# Call the function to display the result
find_numbers()
```

```
2002, 2009, 2016, 2023, 2037, 2044, 2051, 2058, 2072, 2079, 2086, 2093, 2107,
2114, 2121, 2128, 2142, 2149, 2156, 2163, 2177, 2184, 2191, 2198, 2212, 2219,
2226, 2233, 2247, 2254, 2261, 2268, 2282, 2289, 2296, 2303, 2317, 2324, 2331,
2338, 2352, 2359, 2366, 2373, 2387, 2394, 2401, 2408, 2422, 2429, 2436, 2443,
2457, 2464, 2471, 2478, 2492, 2499, 2506, 2513, 2527, 2534, 2541, 2548, 2562,
2569, 2576, 2583, 2597, 2604, 2611, 2618, 2632, 2639, 2646, 2653, 2667, 2674,
2681, 2688, 2702, 2709, 2716, 2723, 2737, 2744, 2751, 2758, 2772, 2779, 2786,
2793, 2807, 2814, 2821, 2828, 2842, 2849, 2856, 2863, 2877, 2884, 2891, 2898,
2912, 2919, 2926, 2933, 2947, 2954, 2961, 2968, 2982, 2989, 2996, 3003, 3017,
3024, 3031, 3038, 3052, 3059, 3066, 3073, 3087, 3094, 3101, 3108, 3122, 3129,
3136, 3143, 3157, 3164, 3171, 3178, 3192, 3199
```

[4]:

```
# 7. Write a program that can compute the factorial of a given numb. Use␣
 ↪recursion to find it.
# Hint: Suppose the following input is supplied to the program: 8
# Then, the output should be: 40320

# Function to compute factorial using recursion
def factorial(n):
    # Base case: factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1
    else:
        # Recursive case: n * factorial of (n-1)
        return n * factorial(n - 1)
```

```python
# Input: Get the number from the user
num = int(input("Enter a number to find its factorial: "))

# Output: Compute and display the factorial
result = factorial(num)
print(f"The factorial of {num} is: {result}")
```

```
Enter a number to find its factorial:  8
```

```
The factorial of 8 is: 40320
```

[5]:

```python
# 8. Write a program that calculates and prints the value according to the␣
 ↪given formula:
# Q = Square root of [(2 * C * D)/H]
# Following are the fixed values of C and H: C is 50. H is 30.
# D is the variable whose values should be input to your program in a comma␣
 ↪separated sequence.
# Example:
# Let us assume the following comma-separated input sequence is given to the␣
 ↪program: 100,150,180
# The output of the program should be: 18,22,24

import math

# Fixed values of C and H
C = 50
H = 30

# Function to calculate Q for a given D
def calculate_Q(D):
    return int(math.sqrt((2 * C * D) / H))

# Input: Get comma-separated values of D
input_values = input("Enter comma-separated values for D: ")

# Split the input into a list of strings, convert to integers
D_values = [int(d) for d in input_values.split(",")]

# Calculate the corresponding Q values using the formula
Q_values = [calculate_Q(D) for D in D_values]

# Output: Print the Q values as a comma-separated sequence
print(", ".join(map(str, Q_values)))
```

```
Enter comma-separated values for D:  100,150,180
```

```
18, 22, 24
```

[6]:

```
# 9. Write a program that takes 2 digits, X, Y as input and generates a␣
 ↪2-dimensional array.
# The element value in the i-th row and j-th column of the array should be i*j.
# Note: i=0,1.., X-1; j=0,1,¡-Y-1.
# Example:
# Suppose the following inputs are given to the program: 3,5
# Then, the output of the program should be: [[0, 0, 0, 0, 0], [0, 1, 2, 3, 4],␣
 ↪[0, 2, 4, 6, 8]]

# Function to generate 2D array
def generate_2d_array(X, Y):
    # Create a 2D array using list comprehension
    array = [[i * j for j in range(Y)] for i in range(X)]
    return array

# Input: Get X and Y from the user
X, Y = map(int, input("Enter two digits X and Y (comma-separated): ").
 ↪split(','))

# Generate the 2D array
result_array = generate_2d_array(X, Y)

# Output: Print the 2D array
print(result_array)
```

```
Enter two digits X and Y (comma-separated):  3,5
```

```
[[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]
```

[7]:

```
# 10.Write a program that accepts a comma-separated sequence of words as input␣
 ↪and prints the words in a comma-separated sequence after sorting them␣
 ↪alphabetically.
# Suppose the following input is supplied to the program: ␣
 ↪without,hello,bag,world
# Then, the output should be:  bag,hello,without,world

# Input: Get a comma-separated sequence of words from the user
input_words = input("Enter a comma-separated sequence of words: ")

# Split the input into a list of words
words_list = input_words.split(',')

# Sort the list of words alphabetically
sorted_words = sorted(words_list)

# Join the sorted list back into a comma-separated string
result = ",".join(sorted_words)
```

```
# Output: Print the sorted words
print(result)
```

```
Enter a comma-separated sequence of words:  without,hello,bag,world
```

```
bag,hello,without,world
```

[8]:

```
# 11.Write a program that accepts sequence of lines as input and prints the␣
 ↪lines after making all characters in the sentence capitalized.
# Suppose the following input is supplied to the program:
# Hello world
# Practice makes perfect

# Then, the output should be:
# HELLO WORLD
# PRACTICE MAKES PERFECT

# Function to accept multiple lines as input and return them capitalized
def capitalize_lines():
    lines = []
    print("Enter your lines of text (type 'done' when finished):")

    # Accept multiple lines of input from the user
    while True:
        line = input()
        if line.lower() == 'done':  # Stop when 'done' is typed
            break
        lines.append(line.upper())  # Convert the line to uppercase

    # Output: Print each line capitalized
    for line in lines:
        print(line)

# Call the function to get and display the capitalized lines
capitalize_lines()
```

```
Enter your lines of text (type 'done' when finished):
```

```
 Hello world
 Practice makes perfect
 done
```

```
HELLO WORLD
PRACTICE MAKES PERFECT
```

[10]:

```
# 12.Write a program that accepts a sequence of whitespace-separated words as␣
 ↪input and prints the words after removing all duplicate words and sorting␣
 ↪them alphanumerically.
# Suppose the following input is supplied to the program:
```

```
# hello world and practice makes perfect and hello world again
# Then, the output should be:
# again and hello makes perfect practice world

# Function to process input, remove duplicates, and sort the words
def process_words():
    # Input: Get a sequence of whitespace-separated words from the user
    input_words = input("Enter a sequence of whitespace-separated words: ")

    # Split the input into a list of words
    words_list = input_words.split()

    # Remove duplicates by converting the list to a set
    unique_words = set(words_list)

    # Sort the unique words alphanumerically
    sorted_words = sorted(unique_words)

    # Output: Print the sorted words as a space-separated sequence
    print(" ".join(sorted_words))

# Call the function to execute the program
process_words()
```

```
Enter a sequence of whitespace-separated words:  hello world and practice makes
perfect and hello world again
```

```
again and hello makes perfect practice world
```

[11]:

```
# 13.Write a program that accepts a sequence of comma separated 4 digit binary␣
 ↪numbers as its input and then check whether they are divisible by 5 or not.
# The numbers that are divisible by 5 are to be printed in a comma-separated␣
 ↪sequence.
# Example: 0100,0011,1010,1001
# Then the output should be: 1010

# Function to check if binary numbers are divisible by 5
def check_divisible_by_5():
    # Input: Get a sequence of comma-separated binary numbers from the user
    input_numbers = input("Enter comma-separated 4-digit binary numbers: ")

    # Split the input into a list of binary numbers
    binary_numbers = input_numbers.split(',')

    # List to store numbers that are divisible by 5
    divisible_by_5 = []

    # Iterate through each binary number
    for binary in binary_numbers:
        # Convert binary to decimal
        decimal_value = int(binary, 2)

        # Check if the decimal value is divisible by 5
        if decimal_value % 5 == 0:
            divisible_by_5.append(binary)

    # Output: Print the binary numbers divisible by 5 as a comma-separated sequence
    print(",".join(divisible_by_5))

# Call the function to execute the program
check_divisible_by_5()
```

```
Enter comma-separated 4-digit binary numbers:  0100,0011,1010,1001
```

```
1010
```

[12]:

```
# 14.Write a program that accepts a sentence and calculates the number of upper case letters and lower case letters.
# Suppose the following input is supplied to the program:
# Hello world!
# Then, the output should be:
# UPPER CASE 1
# LOWER CASE 9

# Function to count upper case and lower case letters in a sentence
def count_case(sentence):
    upper_count = 0
    lower_count = 0

    # Iterate through each character in the sentence
    for char in sentence:
        if char.isupper():
            upper_count += 1
        elif char.islower():
            lower_count += 1

    # Output: Print the counts of upper and lower case letters
    print(f"UPPER CASE {upper_count}")
    print(f"LOWER CASE {lower_count}")

# Input: Get the sentence from the user
sentence = input("Enter a sentence: ")

# Call the function to count upper and lower case letters
count_case(sentence)
```

```
Enter a sentence:  Hello world!
```

```
UPPER CASE 1
LOWER CASE 9
```

[13]:

```
# 15. Give an example of the fsum and sum function of the math library.
# The built-in sum() adds the floating-point numbers in the list, but due to␣
 ↪rounding errors, the result might not be exact.
# The math.fsum() performs a high-precision sum of floating-point numbers,␣
 ↪ensuring more accuracy.

import math

# List of floating-point numbers
numbers = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

# Using built-in sum function
sum_result = sum(numbers)

# Using math.fsum function
fsum_result = math.fsum(numbers)

# Output results
print(f"Result using sum(): {sum_result}")
print(f"Result using math.fsum(): {fsum_result}")
```

```
Result using sum(): 0.9999999999999999
Result using math.fsum(): 1.0
```

[ ]:

```
```

[8]:

```
#### Mr Akram M'Tir 12/22-10-2024
```