Gmail **Ranuga Gamage <ranuga.20231264@iit.ac.lk>**

Sample Paper - How to Answer the Sample Paper for Lab Based Practicals

1 message

Software Development I <sd1@iit.ac.lk> Fri, Dec 20, 2024 at 8:50 AM
Bcc: l4cs_sept@iit.ac.lk

How to Answer the Sample Paper

The following is a **sample demonstration** of how you are expected to structure your answers in the provided text area. This example is not a complete answer but serves to illustrate the **format and approach** you should follow when writing your responses.

Lab Based Practicals is a closed book test with an exception you can bring the two .py files that you have been asked to include as part of the assignment printed hard copy for the Lab Based Practicals.

The lab based practical is made up of 5 questions and each of them will be based tasks given for the assignment and will be given 75 minutes to complete the assessment

1. Input Validation (Task A)

Question:
How would you extend the input validation to check whether the entered date uses numeric values for the day, month, and year while keeping the existing validation for range checks?

Explanation:
Numeric validation ensures users provide a valid number format for date inputs. This prevents errors caused by entering text like "abc" or special characters. The `.isdigit()` method can quickly verify if a string contains only digits. This check complements the existing range validation.

Code Modification:
Below is the updated validation function:

```
def validate_date():
    day = input("Enter the day (DD): ")
    if not day.isdigit():
        print("Error: Day must be a numeric value.")
        return False
    elif int(day) < 1 or int(day) > 31:
        print("Error: Day out of range. It must be between 1 and 31.")
        return False

    month = input("Enter the month (MM): ")
    if not month.isdigit():
        print("Error: Month must be a numeric value.")
        return False
    elif int(month) < 1 or int(month) > 12:
        print("Error: Month out of range. It must be between 1 and 12.")
        return False

    year = input("Enter the year (YYYY): ")
    if not year.isdigit():
        print("Error: Year must be a numeric value.")
        return False
    elif int(year) < 2000 or int(year) > 2024:
        print("Error: Year out of range. It must be between 2000 and 2024.")
        return False

    print("Date validation successful.")
    return True

# Test the function
if validate_date():
    print("Proceed with data processing.")
```

---

2. Processed Outcomes (Task B)

Question:
Can you add functionality to count the number of vehicles that traveled during nighttime (e.g., between 8 PM and 6 AM) for the selected date?

Explanation:
Nighttime traffic analysis provides additional insight into traffic patterns. This can be implemented by checking if the `timeOfDay` column in the dataset falls within the specified hours. The `split(':')` method can extract the hour from the timestamp.

Code Extension:
Add this snippet to your data processing logic:

```
def count_nighttime_vehicles(dataset):
    nighttime_count = 0
    for row in dataset:
        hour = int(row['timeOfDay'].split(':')[0])
        if hour >= 20 or hour < 6:  # Nighttime hours: 8 PM to 6 AM
            nighttime_count += 1
    return nighttime_count

# Usage example:
nighttime_count = count_nighttime_vehicles(dataset)
print(f"The total number of vehicles during nighttime is {nighttime_count}.")
```

---

3. Save Results as Text File (Task C)

Question:
How would you modify the program to save the selected date as a header in the `results.txt` file?

Explanation:
Including the date as a header in the results file makes it easier for users to associate the saved data with a specific survey date. This requires a slight modification to the function that saves the results, appending the selected date as a formatted header.

Code Modification:
Modify the `save_results` function:

```
def save_results(results, selected_date):
    with open("results.txt", "a") as file:
        file.write(f"Results for {selected_date}:\n")  # Header with date
        for result in results:
            file.write(f"{result}\n")
        file.write("\n")  # Add a blank line for separation

# Example of saving results:
selected_date = "10/12/2024"
results = [
    "Total vehicles: 1037",
    "Total trucks: 109",
    "Electric vehicles: 368"
]
save_results(results, selected_date)
print("Results saved to results.txt.")
```

---

4. Histogram Display (Task D)

Question:
Can you modify the histogram to include a line marking the average traffic volume per hour?

Explanation:
A visual indicator like an average traffic line on the histogram helps users quickly identify which hours exceed or fall below the average. This requires calculating the average and drawing a horizontal line on the graph at the corresponding y-coordinate.

Code Extension:
Update the histogram drawing logic:

```
def draw_histogram(hourly_volumes, canvas, canvas_width, canvas_height):
    scale = canvas_height / max(hourly_volumes)  # Scale for the bar height
    average_volume = sum(hourly_volumes) / len(hourly_volumes)  # Calculate average
    avg_y = canvas_height - average_volume * scale  # Y-coordinate for the average line

    # Draw bars for each hour
    for i, volume in enumerate(hourly_volumes):
        x0 = i * (canvas_width // len(hourly_volumes))
        x1 = x0 + (canvas_width // len(hourly_volumes)) - 5
        y0 = canvas_height - volume * scale
        y1 = canvas_height
        canvas.create_rectangle(x0, y0, x1, y1, fill="blue")

    # Draw the average line
    canvas.create_line(0, avg_y, canvas_width, avg_y, fill="red", dash=(4, 2))

    # Add labels, axes, etc. as needed
    print(f"Average traffic volume line drawn at {average_volume:.2f}.")
```

---

5. Handling Multiple CSV Files (Task E)

Question:
How would you extend the program to allow the user to view the results of all previously selected dates at the end of the program?

Explanation:
Storing results from multiple dates in a dictionary enables users to review data collectively. This can be presented at the end of the program loop, allowing a summary of all processed dates.

Code Modification:
Add a dictionary to store results and print them at the end:

```
all_results = {}

while True:
    selected_date = input("Enter a date (DD MM YYYY): ")
    results = process_data(selected_date)  # Assume process_data function
    all_results[selected_date] = results

    cont = input("Do you want to process another date? (Y/N): ").strip().lower()
    if cont == 'n':
        break

# Display results for all dates
```

```
    print("\nSummary of All Dates Processed:")
    for date, results in all_results.items():
        print(f"\nResults for {date}:")
        for result in results:
            print(result)
```