In [1]:
```
from typing import List

def reverse_by_n_elements(lst: List[int], n: int) -> List[int]:

    result = []
    for i in range(0, len(lst), n):
        end_index = min(i + n, len(lst))
        group = lst[i:end_index]
        for j in range(len(group) - 1, -1, -1):
            result.append(group[j])
    return result
user_input = input("Input: ")
n = int(input("n: "))
input_list = list(map(int, user_input.strip('[]').split(',')))
output = reverse_by_n_elements(input_list, n)
print("Output:", output)
```

```
Input: [1, 2, 3, 4, 5, 6, 7, 8]
n: 3
Output: [3, 2, 1, 6, 5, 4, 8, 7]
```

In [2]:
```
from typing import List, Dict
import ast

def group_by_length(lst: List[str]) -> Dict[int, List[str]]:

    length_dict = {}

    for string in lst:
        length = len(string)
        if length not in length_dict:
            length_dict[length] = []
        length_dict[length].append(string)


    sorted_dict = dict(sorted(length_dict.items()))

    return sorted_dict

user_input = input("Input: ")

input_list = ast.literal_eval(user_input)

output = group_by_length(input_list)
print("Output:", output)
```

```
Input: ["apple", "bat", "car", "elephant", "dog", "bear"]
Output: {3: ['bat', 'car', 'dog'], 4: ['bear'], 5: ['apple'], 8: ['elephan
t']}
```

In [3]:
```
from typing import Dict, Any
import ast

def flatten_dictionary(nested_dict: Dict[str, Any], parent_key: str = '', sep
    items = {}
    for key, value in nested_dict.items():
        new_key = f"{parent_key}{sep}{key}" if parent_key else key

        if isinstance(value, dict):
            items.update(flatten_dictionary(value, new_key, sep=sep))
        elif isinstance(value, list):
            for index, item in enumerate(value):
                if isinstance(item, dict):
                    items.update(flatten_dictionary(item, f"{new_key}[{index}
                else:
                    items[f"{new_key}[{index}]"] = item
        else:
            items[new_key] = value

    return items

user_input = input("Input: ")

try:
    nested_dict = ast.literal_eval(user_input)
    if not isinstance(nested_dict, dict):
        raise ValueError("Input must be a dictionary.")
except Exception as e:
    print(f"Invalid input: {e}")
else:

    flattened_dict = flatten_dictionary(nested_dict)
    print("Output:")
    print(flattened_dict)
```

```
Input: {'road': {'name': 'Highway 1', 'length': 350, 'sections': [{'id': 1,
'condition': {'pavement': 'good', 'traffic': 'moderate'}}]}}
Output:
{'road.name': 'Highway 1', 'road.length': 350, 'road.sections[0].id': 1, 'ro
ad.sections[0].condition.pavement': 'good', 'road.sections[0].condition.traf
fic': 'moderate'}
```

In [4]:
```
from typing import List, Set, Tuple
import ast

def unique_permutations(nums: List[int]) -> List[List[int]]:
    from itertools import permutations
    unique_perms = set(permutations(nums))

    return sorted([list(perm) for perm in unique_perms])

user_input = input("Input: ")
try:
    nums = ast.literal_eval(user_input)

    if not isinstance(nums, list) or not all(isinstance(i, int) for i in nums
        raise ValueError("Invalid input format. Please enter a list of intege

    unique_perm = unique_permutations(nums)

    print("Output:")
    print("[")
    for perm in unique_perm:
        print(f"    {perm},")
    print("]")

except (ValueError, SyntaxError):
    print("Please enter a valid list of integers in the format [1, 1, 2].")
```

```
Input: [1, 1, 2]
Output:
[
    [1, 1, 2],
    [1, 2, 1],
    [2, 1, 1],
]
```

In [1]:
```python
import re
from typing import List

def find_all_dates(text: str) -> List[str]:

    date_patterns = [
        r'\b\d{2}-\d{2}-\d{4}\b',  # dd-mm-yyyy
        r'\b\d{2}/\d{2}/\d{4}\b',  # mm/dd/yyyy
        r'\b\d{4}\.\d{2}\.\d{2}\b'  # yyyy.mm.dd
    ]

    combined_pattern = '|'.join(date_patterns)


    valid_dates = re.findall(combined_pattern, text)

    return valid_dates

user_input = input("Input: ")

dates_found = find_all_dates(user_input)
print("Output:")
print(dates_found)
```

```
Input: I was born on 23-08-1994, my friend on 08/23/1994, and another one on
1994.08.23.
Output:
['23-08-1994', '08/23/1994', '1994.08.23']
```

In [2]:
```
import pandas as pd
import polyline
import numpy as np

def haversine(lat1, lon1, lat2, lon2):

    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arcsin(np.sqrt(a))
    r = 6371000  # Radius of Earth in meters
    return c * r

def polyline_to_dataframe(polyline_str: str) -> pd.DataFrame:

    coordinates = polyline.decode(polyline_str)

    df = pd.DataFrame(coordinates, columns=['latitude', 'longitude'])

    distances = [0]  # First point has distance 0
    for i in range(1, len(df)):
        dist = haversine(df.iloc[i-1]['latitude'], df.iloc[i-1]['longitude'],
                         df.iloc[i]['latitude'], df.iloc[i]['longitude'])
        distances.append(dist)

    df['distance'] = distances

    return df

polyline_input = input("Input: ")
output_df = polyline_to_dataframe(polyline_input)
print("Output DataFrame:")
print(output_df)
```

```
Input: _p~iF~ejg~uH
Output DataFrame:
   latitude    longitude  distance
0      38.5 -52173.26704         0
```

In [3]:
```
from typing import List
import ast

def rotate_and_transform_matrix(matrix: List[List[int]]) -> List[List[int]]:

    n = len(matrix)

    rotated_matrix = [[0] * n for _ in range(n)]  # Initialize a new matrix f
    for i in range(n):
        for j in range(n):
            rotated_matrix[j][n - 1 - i] = matrix[i][j]

    print(f"rotated_matrix = {rotated_matrix}")

    transformed_matrix = [[0] * n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            row_sum = sum(rotated_matrix[i]) - rotated_matrix[i][j]
            col_sum = sum(rotated_matrix[k][j] for k in range(n)) - rotated_m
            transformed_matrix[i][j] = row_sum + col_sum

    print(f"final_matrix = {transformed_matrix}")

    return transformed_matrix

def main():
    user_input = input("Input: ")

    matrix = ast.literal_eval(user_input)

    output_matrix = rotate_and_transform_matrix(matrix)

if __name__ == "__main__":
    main()
```

```
Input: [[1, 2, 3],[4, 5, 6],[7, 8, 9]]
rotated_matrix = [[7, 4, 1], [8, 5, 2], [9, 6, 3]]
final_matrix = [[22, 19, 16], [23, 20, 17], [24, 21, 18]]
```

In [4]:
```
import pandas as pd
from datetime import datetime, timedelta

file_path = r'C:\Users\Reddy\Downloads\dataset-1.csv'
df = pd.read_csv(file_path)

day_to_offset = {
    'Monday': 0,
    'Tuesday': 1,
    'Wednesday': 2,
    'Thursday': 3,
    'Friday': 4,
    'Saturday': 5,
    'Sunday': 6
}

today = datetime.now()

def get_full_datetime(day, time):
    day_offset = day_to_offset[day]
    date_of_week = today - timedelta(days=today.weekday() - day_offset)
    return datetime.strptime(f"{date_of_week.date()} {time}", '%Y-%m-%d %H:%M

df['start'] = df.apply(lambda row: get_full_datetime(row['startDay'], row['st
df['end'] = df.apply(lambda row: get_full_datetime(row['endDay'], row['endTim

print(df[['start', 'end']].head())

def time_check(df: pd.DataFrame) -> pd.Series:
    results = []

    grouped = df.groupby(['id', 'id_2'])

    for (id_val, id_2_val), group in grouped:
        start_times = group['start'].min()
        end_times = group['end'].max()
        days_covered = group['startDay'].unique()

        is_24_hour_complete = (end_times - start_times) >= timedelta(days=1)
        is_7_days_complete = len(days_covered) == 7

        results.append(((id_val, id_2_val), not (is_24_hour_complete and is_7

    return pd.Series(dict(results))

result = time_check(df)

print(result)
```

```
                start                 end
0 2024-10-21 05:00:00 2024-10-23 10:00:00
1 2024-10-21 10:00:00 2024-10-25 15:00:00
2 2024-10-24 15:00:00 2024-10-25 19:00:00
3 2024-10-21 19:00:00 2024-10-25 23:59:59
4 2024-10-26 00:00:00 2024-10-27 23:59:59
1014000  -1          True
1014002  -1          True
1014003  -1          True
1030000  -1          True
          1030002    True
                     ...
1330016   1330006    True
          1330008    True
          1330010    True
          1330012    True
          1330014    True
Length: 9254, dtype: bool
```

In [ ]: