```
# Context for LLM: Functions and Framework for Circular Seating Arrangement Problem

# Description:
# This framework is designed to solve circular seating arrangement problems involving six representatives.
# It provides functions for defining and validating conditions, generating valid arrangements, and querying
# specific seating-related questions.

# Instructions:
# 1. Use the `find_valid_arrangements` function to find all valid arrangements based on specified conditions.
# 2. Use `is_valid_arrangement` to validate specific arrangements.
# 3. For querying specific relationships (e.g., neighbors or someone sitting between two people), use the
#    query functions (`query_neighbors`, `query_between`).
# 4. Ensure that all required conditions are implemented as functions and passed to `find_valid_arrangements`.
# 5. These functions should be used in combination to comprehensively solve circular table seating arrangement problems.

import itertools

# Define the list of representatives
representatives = ['K', 'L', 'M', 'N', 'O', 'P']

# Function to check if two people sit immediately next to each other
def sits_next_to(arrangement, person1, person2):
    """Checks if person1 and person2 are adjacent in the arrangement."""
    n = len(arrangement)
    return abs(arrangement.index(person1) - arrangement.index(person2)) == 1 or \
           abs(arrangement.index(person1) - arrangement.index(person2)) == n - 1

# Function to check if a person sits next to any of a list of people
def sits_next_to_any(arrangement, person, others):
    """Checks if person sits next to any person in the 'others' list."""
    n = len(arrangement)
    return any(abs(arrangement.index(person) - arrangement.index(other)) == 1 or
               abs(arrangement.index(person) - arrangement.index(other)) == n - 1
               for other in others)

# Function to check if a person does not sit next to another person
def does_not_sit_next_to(arrangement, person1, person2):
    """Checks if person1 does not sit next to person2."""
```

```
    n = len(arrangement)
    return abs(arrangement.index(person1) - arrangement.index(person2)) != 1 and \
           abs(arrangement.index(person1) - arrangement.index(person2)) != n - 1

# Function to check the condition where one person sits next to another, and then a third person does not sit next to the second
def sits_next_to_and_not_next_to(arrangement, person1, person2, person3):
    """Ensures person1 sits next to person2, and person3 does not sit next to person2."""
    n = len(arrangement)
    pos1 = arrangement.index(person1)  # Position of person1
    pos2 = arrangement.index(person2)  # Position of person2
    pos3 = arrangement.index(person3)  # Position of person3

    if abs(pos1 - pos2) == 1 or abs(pos1 - pos2) == n - 1:
        return abs(pos2 - pos3) != 1 and abs(pos2 - pos3) != n - 1
    return True

# Helper function to normalize circular permutations
def normalize(arrangement):
    """Returns the lexicographically smallest rotation of the arrangement."""
    n = len(arrangement)
    rotations = [arrangement[i:] + arrangement[:i] for i in range(n)]
    return min(rotations)

# Function to return all valid arrangements based on a list of condition functions
def find_valid_arrangements(arrangements, conditions):
    """Generates all valid arrangements that satisfy the provided conditions."""
    unique_valid_arrangements = set()
    for arrangement in arrangements:
        if all(condition(arrangement) for condition in conditions):
            normalized_arrangement = normalize(arrangement)
            unique_valid_arrangements.add(tuple(normalized_arrangement))
    return unique_valid_arrangements

# Function to check if a given arrangement is valid based on the unique_valid_arrangements set
def is_valid_arrangement(arrangement, valid_set):
    """Checks if a specific arrangement is valid."""
    normalized_arrangement = normalize(arrangement)
    return tuple(normalized_arrangement) in valid_set

# Function to return a dictionary mapping chair positions to representatives
def seating_positions_from_arrangement(arrangement):
    """Maps chair positions (1 to n) to representatives."""
    return {i + 1: person for i, person in enumerate(arrangement)}
```

```
# Function to find who sits to the left of someone
def who_sits_on_left(seating_positions, person):
    ""Finds the person sitting to the left (counter-clockwise) of the given person.""
    n = len(seating_positions)
    pos = [key for key, value in seating_positions.items() if value == person][0]
    left_pos = pos - 1 if pos - 1 > 0 else n  # Wrap around to the last position
    return seating_positions[left_pos]

# Function to find who sits to the right of someone
def who_sits_on_right(seating_positions, person):
    ""Finds the person sitting to the right (clockwise) of the given person.""
    n = len(seating_positions)
    pos = [key for key, value in seating_positions.items() if value == person][0]
    right_pos = pos + 1 if pos + 1 <= n else 1  # Wrap around to the first position
    return seating_positions[right_pos]

# Function to correctly find the person sitting between two others
def who_sits_between(seating_positions, person1, person2):
    ""Finds the person sitting directly between person1 and person2.""
    pos1 = [key for key, value in seating_positions.items() if value == person1][0]
    pos2 = [key for key, value in seating_positions.items() if value == person2][0]
    n = len(seating_positions)
    if abs(pos1 - pos2) == 2 or abs(pos1 - pos2) == n - 2:
        between_pos = (pos1 + 1) if abs(pos1 - pos2) == 2 else (pos2 + 1)
        between_pos = between_pos if between_pos <= n else 1
        return seating_positions[between_pos]
    return None

# Function to query neighbors of a specific person in a given seating arrangement
def query_neighbors(seating_position, person):
    ""Queries the neighbors of a person in the seating arrangement.""
    seating_positions = seating_positions_from_arrangement(seating_position)
    left_person = who_sits_on_left(seating_positions, person)
    right_person = who_sits_on_right(seating_positions, person)
    print(f"{person} has {left_person} on the left and {right_person} on the right.")

# Function to query who sits between two specific people in a given arrangement
def query_between(seating_position, person1, person2):
    ""Queries who sits between two people in the seating arrangement.""
    seating_positions = seating_positions_from_arrangement(seating_position)
    person_between = who_sits_between(seating_positions, person1, person2)
    if person_between:
        print(f"{person_between} sits between {person1} and {person2}.")
```

```
    else:
        print(f"No one sits between {person1} and {person2}.")

# Note:
# Combine these functions to define specific conditions, validate seating arrangements, and query seating-related relationships.
# Example usage can involve generating permutations of representatives, defining conditions using these functions, and analyzing results.
```