Can We Make a Reservation?

Businesses often use software to manage certain aspects of their operations. For example, restaurants might use a reservation booking service like OpenTable or Resy. In this problem, we will supply a `Reservation` class and you will complete the methods for a `Restaurant` class for use in a booking service.

To request a reservation, a customer needs to provide: the name of the restaurant, the name of their party, and the number of guests in their party. (See below for the implementation of the `Reservation` class.)

A restaurant has a name and a number of tables. Each table will have four seats and can be used only once a night. The tables are numbered 0 through $N - 1$, where N is the number of tables.

A given reservation can take up more than one table, but there will be at most one party per table. For example, a reservation for a party of six will have exclusive use of two tables. Parties are seated at available tables in numeric order (i.e., starting from table 0 through N - 1). If the restaurant does not have enough tables to accommodate the reservation, then no tables are assigned.

For this example, assume that Nella has 5 tables and Medici has 8 tables and that the reservations are processed in order.

Restaurant Name	Party Name	Number of Guests	Tables Assigned
Nella	Hammond	5	0, 1
Medici	Bolton	5	0, 1
Medici	Lumbergh	3	2
Nella	Malcolm	3	2
Medici	Smykowski	8	3, 4
Nella	Grant	4	3
Medici	Joanna	9	5, 6, 7
Medici	Waddams	2	–

Your task is to implement the constructor and two additional methods— `make_reservation` and `confirm_reservation`— for the `Restaurant` class.

The method `make_reservation` takes a `Reservation` and, if possible, updates it to include the table(s) assigned to the party making the reservation. The method should return `true` when there were enough tables to fulfill the reservation and `false` otherwise.

The method `make_reservation` takes the name of a party and an array of table numbers. It should return `true` if the specified party was assigned the specified tables and `false` otherwise.

You are are welcome to write additional helper methods and to include additional attributes., The constructor, `make_reservation`, and `confirm_reservation`

are required.

Your implementation must take advantage of the attributes and methods provided by the Reservation class.

Here is the code for the `Reservation` class. You can find it in the file named `reservation.py`.

```
'''
Distribution file of the restaurants problem containing the Reservation class.

Please do not modify this file. Your solution should be placed in restaurant.py.
'''

from typing import List

class Reservation:
    '''
    Class for representing requests for restaurant reservations.

    Public attributes:
        -   restaurant_name (str): The name of the restaurant.
        -   party_name (str): The name of the party.
        -   party_size (int): The size of the party.

    Public methods:
        -   assign_tables: Assign tables to this reservation.
        -   confirm_tables: Confirm the tables assigned to this reservation.
    '''
    def __init__(self, restaurant_name: str, party_name: str, party_size: int) -> None:
        '''
        Constructor for Reservation class.

        Parameters:
            -   restaurant_name (str): The name of the restaurant.
            -   party_name (str): The name of the party.
            -   party_size (int): The size of the party.

        Returns:
            None
        '''
        self.restaurant_name = restaurant_name
        self.party_name = party_name
        self.party_size = party_size
        self.__tables = []

    def assign_tables(self, tables: List[int]) -> None:
```

```
        '''
        Assign tables to this reservation, using a deep copy of the list of
        tables.

        Parameters:
            -   tables (List[int]): The tables to assign to this reservation.

        Returns:
            None
        '''
        self.__tables = tables.copy()

    def confirm_tables(self, tables: List[int]) -> bool:
        '''
        Confirm the given tables were assigned to this reservation.

        Parameters:
            -   tables (List[int]): The tables to confirm.

        Returns:
            True if the tables were assigned to this reservation, False
            otherwise.
        '''
        # We have to loop over the tables as we cannot compare lists directly,
        # even if they contain the same elements since they could be in
        # different orders.
        for table in tables:
            if table not in self.__tables:
                return False

        return True
```

Here is the skeleton code for the `Restaurant` class. For this practice problem, you can find it in the file named `restaurant.py`. For the actual exam, you would be expected to copy it into a file named `restaurant.py`.

```
'''
Distribution file of the restaurants problem containing the Restaurant class.

Please write your solution in the methods with TODO blocks.
'''

from typing import List
from reservation import Reservation


class Restaurant:
```

```
'''
Class for representing restaurants.

Public attributes:
    -   name (str): The name of the restaurant.

Public methods:
    -   make_reservation: Make a reservation at this restaurant for the
        given party if possible.
    -   confirm_reservation: Confirm the reservation
'''
def __init__(self, name: str, num_tables: int) -> None:
    '''
    Constructor for Restaurant class.

    Parameters:
        -   name (str): The name of the restaurant.
        -   num_tables (int): The number of tables at the restaurant.

    Returns:
        None
    '''
    # TODO: Implement this method.
    pass

def make_reservation(self, reservation: Reservation) -> bool:
    '''
    Assign tables if the reservation is feasible based on the number of
    people in the party and the number of tables available.

    Parameters:
        -   reservation (Reservation): The reservation to make.

    Returns:
        True if the reservation was made, False otherwise.
    '''
    # TODO: Implement this method.
    pass

def confirm_reservation(self, party_name: str, tables: List[int]) -> bool:
    '''
    Verify if the tables were assigned to a given party.

    Parameters:
        -   party_name (str): The name of the party.
        -   tables (List[int]): The tables to confirm.
```

```
Returns:
    True if the tables listed in the reservation for the specified party
    matches the specified tables. False otherwise.
'''
# TODO: Implement this method.
pass
```