Redis: Cookie Synchronization Service

You should implement a **Python3** interface for partners cookies synchronization.

For every user's cookie value (***uid***) we need to store the map from ***partner_id*** to ***partner_uid*** cookie value. An example with the user `uid_1` and two partners is below:

```
{
    'uid_1': {
        10: 'partner_uid_1',
        12: 'partner_uid_2'
    }
}
```

Each partner has its own time to live (ttl). It is the remaining time to live of a key that has a timeout. You have to store ttl values for each partner in the
dictionary (`partner_id → ttl`). An example of such dictionary with the partner ids `10` and `12` and corresponding ttls `0` and `5` is below:

```
ttls = {
    10: 0,
    12: 5
}
```

Task 1: set_ttls

You should implement ***set_ttls*** function, which will set the ttl for each partner id. All the ttls data has to be stored in Redis by the hash name ***"ttls"***. Use ***hset*** method of Redis instance for ttls setting. The signature of the function is the following:

```
def set_ttls(r: redis.StrictRedis, ttls: dict):
    """Set the ttl by partner id

    Args:
        r (redis.StrictRedis): redis instance
        ttls (dict): dictionary of pairs <partner_id, ttl>

    Examples:
        >>> set_ttls(r, {12: 5, 3: 1})

    """
    pass # your code here
```

Task 2: save_sync

You should implement ***save_sync*** function, which creates the following mappings in Redis:

```
<uid, partner_id> → partner_uid
<partner_id, partner_uid> → uid
```

Hint: store key pair as the string in Redis

Moreover, ***save_sync*** function has to use "expire" redis method to set "ttl" value for each pair (Redis key). Each pair `<uid, partner_id>` and `<partner_id, partner_uid>` has to use ttl for the specified "partner_id".

The signature of the function is the following:

```
def save_sync(r: redis.StrictRedis, uid: str, partner_id: int, partner_uid: str):
    """Set the values for the pairs <partner_id, uid> and <partner_id, partner_uid>

    Do not forget to set the ttls which you defined in the function set_ttls

    Agrs:
        r (redis.StrictRedis): redis instance
        uid (str): cookie uid
```

```
        partner_id (int): id of the partner
        partner_uid (str): uid of the partner

    Examples:
        >>> save_sync(r, 'uid_1', 10, 'partner_uid_1')

    """
    pass # your code here
```

If the ttl for the partner is not specified, ***save_sync*** function must be set default ***0 second*** ttl for both pairs `<uid, partner_id>` and `<partner_id, partner_uid>`. If the "expire" method is used correctly, then the information stored in Redis will be removed automatically.

*Note: the usage of the **set_ttls** function should affect the next calls of **save_sync** function. This scenario is repeated multiple times in the test cases.*

Task 3: get_uid and get_partner_uid

Task 3.1

Please implement ***get_uid*** function which returns ***uid*** from Redis when ***partner_id*** and ***partner_uid*** arguments are provided. This function will be used after ***set_ttls*** and ***save_sync*** functions calls. Returned ***uid*** should be decoded with ***decode('utf-8')*** method. If ***uid*** is None (e.g. ttl is expired), you have to return ***None***.

The signature of the ***get_uid*** function is the following:

```
def get_uid(r: redis.StrictRedis, partner_id: int, partner_uid: str):
    """Get the uid by the pair (partner id, partner uid)

    Args:
        r (redis.StrictRedis): redis instance
        partner_id (int): id of partner
        partner_uid (str): uid of partner

    Examples:
        >>> get_uid(r, 12, '25b6e9a6-fca8-427c-94df-2577e62b2bf0')

    """
    # limit_rate(r, get_uid, partner_id) # uncomment during Task 4 implementation
    pass # your code here
```

Task 3.2

Please implement ***get_partner_uid*** function which returns ***partner_uid*** from Redis when ***uid*** and ***partner_id*** arguments are provided. This function will be used after ***set_ttls*** and ***save_sync*** functions calls. Returned ***partner_uid*** should be decoded with ***decode('utf-8')*** method. If ***partner_uid*** is None (e.g. ttl is expired), you have to return ***None***.

The signature of the ***get_partner_uid*** function is the following:

```
def get_partner_uid(r: redis.StrictRedis, uid: str, partner_id: int):
    """Get the partner id by the pair (uid, partner id)

    Args:
        r (redis.StrictRedis): redis instance
        uid (str): cookie uid
        partner_id (int): id of the partner

    Examples:
        >>> get_partner_uid(r, 'e5a370cc-6bdc-43ae-baaa-8fd4531847f7', 12)

    """
    # limit_rate(r, get_partner_uid, partner_id) # uncomment during Task 4 implementation
    pass # your code here
```

Task 4: limit_rate

Finally, your Redis Cookie Synchronization customer wants you to implement one more function ***limit_rate***.

The function ***limit_rate*** must be used in both ***get_partner_uid*** and ***get_uid*** functions. The counter is stored in Redis by the keyname ***f"hit:{function.__name__}:{partner_id}:{current_time_in_seconds}"*** where *function* and *partner_id* are provided, current_time_in_seconds is the ***integer*** current time. This function should restrict the function usage greater than ***MAX_RPS*** requests per second for one partner. If the limit exceeds raise an exception of type "*__LimitExceededException__*" (provided in ipynb).

The signature of the function is the following:

```
def limit_rate(r: redis.StrictRedis, function: Callable, partner_id: int):
    """Restrict function usage by MAX_RPS requests per second.

    If the amount of function calls is greater than MAX_RPS, raise
    LimitExceededException(f"{MAX_RPS}   limit  is reached").

    Args:
        r (redis.StrictRedis): redis instance
        function (Callable): function from which limit_rate is called
        partner_id (int): id of the partner

    Examples:
        >>> limit_rate(r, get_partner_uid, partner_id)

    """
    pass # your code here
```

Hint: see the examples of the Redis INCR command usage in the documentation.