ACTIVIDAD 3

INTEGRANTES:

- Cat Rodríguez Patricia
- Gutierrez Caamal Juan
- Martínez Lugo Joaquín
- Rodríguez Us Astrid

Se importa la librería de **redis**:

```
import redis as rd
from DataStructures import *
```

Se llaman a las funciones **create, read, update, delete.**

```
def create(data):
    data.create()

def read(data):
    data.read()

def update(data):
    data.update()

def delete(data):
    data.delete()
```

Create: Se realiza un registro.

Read: Recupera los registros que ya se hicieron.

Update: Modifica algún registro.

Delete: Elimina un registro que ya se realizó.

Seguidamente se declara el **main**.
Se imprime un menú que tiene 4 opciones:

1. Crear un registro
2. Leer un registro
3. Actualizar (Modificar) un registro
4. Eliminar un registro

```
def main():
    while 1:

        print("Transactions\n1.- Create registry\n2.- Read value\n3.- Update registry\n4.- Delete registry\n0.- Exit program")
```

De acuerdo a la opción que se elija, tendremos un submenú por ejemplo si escogemos la opción 1 **Create registry**, se despliega otro menú.

```
while 1:
    print("Data structures\n1.- String\n2.- List\n3.- Set\n4.- Hash\n5.- Sorted set")
```

1. String
2. List
3. Set
4. Hash
5. Sorted set

Si se elige una opción válida, se procederá a la función que se eligió, pero en caso que se elija una opción que no se presenta en el menú, se va a imprimir **Opción no válida.**

```
else:
    print('Choose a valid data structure.')
```

```
if(not selected == '0'):
    print("Pick a valid transaction.")
    print('----------------------------------
else:
    print('Closing program.')
    print('----------------------------------
    break
```

De igual forma que puede salir del programa con **“0”** en caso que se desee salir.

Una instrucción que funciona para cualquier tipo de dato y en cualquiera de las opciones es: “r, isConnected = connect()”. La cual es necesaria para realizar la conexión con los datos que posee el servidor redis. De igual manera esta opción puede llegar a fallar y no conectarse al servidor, para su correcto funcionamiento.
Hablemos de las diferentes estructuras y cómo funcionan.

MANEJO DE STRINGS.

Si se decide usar el tipo de estructura de String, tenemos como mencionamos anteriormente 4 opciones, que son crear, leer, actualizar y borrar, explicando cada una de ellas, se tiene lo siguiente.

```
def create(self):
    r, isConnected = connect()

    if isConnected:
        key = input("Key: ")

        if not r.exists(key):
            value = input("Value: ")
            r.set(key, value)
            print('Added successfully.')
        else:
            print('That key is already set.')
    else:
        print('Could not connect to server.')
```

Como se mencionó anteriormente, debe conectarse al servidor de Redis, puede darse el caso que no pueda conectarse al servidor.

Si logra la conexión con el servidor, se procede a ingresar una llave, si la llave no existe, se capturará un valor y mediante r.set se ingresa el conjunto de datos que desea agregar (llave-valor), si se diera el caso de que existiera el conjunto de datos ingresados nos avisara que la clave ya se encuentra configurada.

```
    def read(self):
        r, isConnected = connect()

        if isConnected:
            key = input("Key: ")

            if r.exists(key):
                print('The value for the key: ', key, ', is: ',
r.get(key))
            else:
                print('That key does not exist.')
        else:
            print('Could not connect to server.')
```

Si logra conectarse al servidor, nos pedirá ingresemos un valor, mediante “r.exists”, buscara la llave ingresada, si la llave existe nos devolverá el valor de la llave, en caso que esta no exista, nos avisara que la llave no existe.

```
def update(self):
    r, isConnected = connect()

    if isConnected:
        key = input("Key: ")

        if r.exists(key):
            newValue = input("New value: ")

            r.set(key, newValue)
            print('Updated successfully.')
        else:
            print('That key does not exist.')
    else:
        print('Could not connect to server.')
```

Cuando queremos actualizar, igual nos pedirá una llave, que verificará si existe, si existe, ingresamos un nuevo valor, para mediante un r.set esta sea capaz de actualizar los datos de una llave-valor.
En un caso contrario, nos avisara que la llave que deseamos actualizar no existe en la base.

```
def delete(self):
    r, isConnected = connect()

    if isConnected:
        key = input("Key: ")
        if r.exists(key):
            r.delete(key)
            print('Key ', key, ' was deleted.')
        else:
            print('Key does not exist in database.')
    else:
        print('Could not connect to server.')
```

Para eliminar un registro, es necesario ingresar la llave, si esta existe, se procede a eliminarse mediante r.delete, y nos avisara que ya no existe en la base de datos, en caso que la llave ingresada no existe nos indicará que no existe la llave en la base de datos.

MANEJO DE LIST.

Ahora definiremos como es el uso de las estructuras de tipo “LIST” con el programa desarrollado.

```
class List():
    def create(self):
        r, isConnected = connect()

        if isConnected:
            key = input("Key: ")

            if not r.exists(key):
                value = input("Value: ")
                r.rpush(key, value)
                print('Added successfully.')
            else:
                print('That key is already set.')
        else:
            print('Could not connect to server.')
```

Para usar la función create en una lista, se conecta al servidor, solicitará un valor de llave, si esta llave ingresada no existe, lo ingresara mediante la función r.rpush con sus valores de llave-valor, en caso que la llave ya esté en la base de datos, nos indicará que la llave ya esta lista.

```
def read(self):
    r, isConnected = connect()

    if isConnected:
        key = input("Key: ")

        if r.exists(key):
            print('The value for the key: ', key, ', is: ', r.lrange(key, 0, -1))
        else:
            print('That key does not exist.')
    else:
        print('Could not connect to server.')
```

Cuando se necesite hacer uso de la función read, nos solicita una llave, si la llave está en la base de datos, nos otorgara el valor mediante el uso de r.lrange(key, 0, -1), en caso que la llave no exista, el programa nos desplegará el mensaje de que la llave no existe.

```
def update(self):
    r, isConnected = connect()

    if isConnected:
        key = input("Key: ")

        if r.exists(key):
            newValue = input("Add new value: ")

            r.rpushx(key, newValue)
            print('Updated successfully.')
        else:
            print('That key does not exist.')
    else:
        print('Could not connect to server.')
```

Para actualizar una lista, se pedirá una llave, si la llave existe (r.exists) nos solicitara un nuevo valor para dicha llave, con r.rpushx(key, newValue) procederá a actualizar la información de dicha llave, si la llave no existe, ese mensaje se nos desplegara en pantalla.

```
def delete(self):
    r, isConnected = connect()

    if isConnected:
        key = input("Key: ")
        if r.exists(key):

            print('Value ', r.lpop(key),'of the Key ', key, ' was deleted.')
        else:
            print('Key does not exist in database.')
    else:
        print('Could not connect to server.')
```

Si desea eliminar un valor de la lista, se solicita la llave, si esta existe se ejecutará la instrucción r.lpop(key) la cual sacará de la pila la dupla de llave-valor, pero si esta llave no existe se nos informara.

MANEJO DE SET.

Ahora definiremos como es el uso de las estructuras de tipo “SET” con el programa desarrollado.
Los tipos set son un conjunto de cadenas desordenadas de strings.

```
class Set():
    def create(self):
        r, isConnected = connect()

        if isConnected:
            key = input("Key: ")

            if not r.exists(key):
                while 1:
                    value = input("Value: ")
                    r.sadd(key, value)
                    if(input("Add another value to the set? y/n") == n):
                        break

                print('Set added successfully.')
            else:
                print('That key is already set.')
        else:
            print('Could not connect to server.')
```

Para crear una nueva llave en un tipo Set, se pide que se ingrese un valor de llave, si esta no existe se procede a agregarse, pide un valor que se le asigna a la llave y mediante r.sadd, se actualizará el valor de llave-valor, pero a diferencia de anteriores asignaciones, esta nos permite ingresar más valores a la misma llave, una vez concluida, nos notifica que el conjunto fue agregado, pero si la llave ya existe, solo nos notificara que ya existe una llave con ese valor.

```
def read(self):
    r, isConnected = connect()

    if isConnected:
        key = input("Key: ")

        if r.exists(key):
            print('The elements of the set: ', key, ' are: ', r.members(key))
        else:
            print('That key does not exist.')
    else:
        print('Could not connect to server.')
```

Cuando se requiere leer un conjunto de cadenas, nos solicita la llave, se encargará de encontrar si la llave solicitada existe, nos los miembros que tenga la llave que solicitamos (r.members(key)), si la llave que solicitamos no existe, nos informará de ese hecho.

```
def update(self):
    r, isConnected = connect()

    if isConnected:
        key = input("Key: ")

        if r.exists(key):
            while 1:
                value = input("Value: ")
                r.sadd(key, value)
                if(input("Add another value to the set? y/n") == n):
                    break

            print('Values added successfully to set: ', key)
        else:
            print('That key does not exist.')
    else:
        print('Could not connect to server.')
```

Si deseamos actualizar algún conjunto de cadenas, nos solicita una llave para examinar, buscará si existe, si esta llave existe, nos solicitara el valor, para que pueda guardarla en la dupla llave-valor, de igual manera nos permitirá agregarle uno o más nuevos valores a la llave especificada, al finalizar nos indica que fueron agregadas exitosamente en el conjunto de llave, si la llave no existiera, nos avisará de ello.

```
def delete(self):
    r, isConnected = connect()

    if isConnected:
        key = input("Key: ")
        if r.exists(key):

            print('The element:', r.spop(key), ' was removed from set: ', key)
        else:
            print('Key does not exist in database.')
    else:
        print('Could not connect to server.')
```

Si deseamos eliminar un conjunto, nos solicita la llave, verificará si esta existe, si existe mediante un r.spop(key), removerá todo el conjunto asociado a dicha llave, si la llave no existe, nos notificara que no se encuentra en la base de datos.

MANEJO DE HASH.

Ahora definiremos como es el uso de las estructuras de tipo “HASH” con el programa desarrollado.
Este tipo de datos nos permite tener varias veces la misma llave, pero para hacer una diferencia de uno con respecto de otro usa un segundo parámetro que es la subllave, si fueran personas, sería como el nombre y apellido.

```
class Hash():
    def create(self):
        r, isConnected = connect()
        if isConnected:
            key = input('Hash-key: ')
            if not r.exists(key):
                sub_key = input('Sub-key: ')
                value = input('Value: ')
                r.hset(key, sub_key, value)
                print('Added successfully')
            else:
                print('That Hash-key is already set.')
        else:
            print('Could not connect to the server')
```

Si deseamos crear un nuevo hash, nos pedirá la llave hash, si la llave no existe nos solicita un valor de subllave y el valor, mediante el r.hset lo sube mediante la terna de (llave-subllave-valor).

Si la llave existe, nos notificará de ese hecho.

```
def read(self):
    r, isConnected = connect()

    if isConnected:
        key = input('Hash-key: ')
        sub_key= inpunt('Sub-key: ')

        if r.hexists(key,sub_key):

            print (' The key value: ', key, ' is: ', r.hget(key, sub_key))
        else
            print ('That hash-key does not exist.')
    else:
        print('Could not connect to the server')
```

Si se desea leer una consulta hash, nos pide en esta ocasión una llave y una subllave para realizar la consulta mediante r.hexist(key, sub_key), si la dupla existe en la tabla, nos mostrará el valor de la llave mediante el uso de r.hget(key, sub_key), si la llave no existe, nos notificará de ese hecho.

```
def update(self):
    r, isConnected = connect()

    if isConnected:
        key = input('Hash-key: ')
        sub_key = input('Sub-key: ')

        if r.hexists(key, sub_key):
            newValue = input('New value: ')
            r.hset(key, sub_key, newValue)
            print('Updated successfully')
        else:
            print('That key does not exist.')
    else:
        print('Could not connect to the server')
```

Si lo que se desea es actualizar algún hash, debemos ingresar la dupla (llave-subllave) si esta es existente, nos pedirá ingresar un nuevo valor para dicha dupla, pero en caso de que no exista la dupla descrita, nos notificara que no se encuentra dicha llave en la base de datos.

```
def delete(self):
    r, isConnected= connect()

    if isConnected:
        key= input('Hash-key: ')
        sub_key= input ('Sub-key: ')

        if r.hexists(key, sub_key):
            if r.hdel(key, sub_key) == 1:
                print('Deleted succesfully')
            else:
                print('Something went wrong :c')
        else:
            print('Key does not exist in database. ')
    else:
        print('Could not connect to server')
```

Si lo que deseamos es eliminar una llave hash, se solicita el conjunto llave-subllave para realizar la búsqueda en las tablas hash, si la encuentra aplicará el comando r.hdel(key, sub_key) para proceder a eliminar el registro, si la dupla llave-subllave no es encontrada nos notificara, que no existe en la base de datos.

MANEJO DE SORTEDSET.

Este tipo de datos es similar a los tipo set que se explicaron anteriormente, la diferencia radica que estos manejan un sistema de puntos, para tomar desde el más pequeño al más alto, los miembros pueden ser únicos, pero los puntajes pueden ser repetidos.

```
def create(self):
    r, isConnected = connect()

    if isConnected:
        key= input("zSet-key: ")

        if not r.exists(key):
            while 1:
                score= input('Score: ')
                value= input('Value: ')
                r.zadd(key, score, value)
                if (input ("Add another value to the set? y/n") == n):
                    break
            print('Sorted set added successfully')
        else:
            print('That key is already set.')
    else:
        print('Could not connect to server.')
```

Si deseamos crear una estructura de tipo SortedSet, nos pide un Set-key, si la llave no existe se procede a solicitarse los valores de score y value, los cuales con el uso de r.zadd(key, score, value) se encargará de agregar la llave y nos preguntara si deseamos agregar otro valor a la dupla de llave-puntaje, al finalizar nos notifica que el sortedset fue agregado exitosamente, si la llave ya existe nos informará que ya se encuentra dicha llave.

```
def read(self):
    r, isConnected = connect()

    if isConnected:
        key = input('Key: ')

        if r.exists(key):
            print('The elements of the set', key, 'are', r.zrange(key, '0', '-1', withscores=True))
        else:
            print('That key does not exist.')
    else:
        print('Could not connect to server.')
```

Si lo deseado es realizar una consulta (lectura) de un valor en un sortedset, nos solicitara el valor de llave, una vez validada de que si existe, con lo r.zrange y los parámetros de (key, ‘0’, ‘-1’, withscores=true) nos devolverá los elementos pertenecientes a la llave en cuestión con sus respectivos scores. Si la llave que es solicita no existe, nos notifica de ese hecho.

```
def update(self):
    r, isConnected = connect()

    if isConnected:
        key= input("zSet-key: ")

        if not r.exists(key):
            while 1:
                score= input('Score: ')
                newValue= input(' New Value: ')
                r.zadd(key, score, newValue)
                if (input ("Add another value to the set? y/n") == n):
                    break
            print('Sorted set updated successfully')
        else:
            print('That key is already set.')
    else:
        print('Could not connect to server.')
```

Si desea realizar una actualización de valores, se solicita el valor de una llave, si esta no existe se procede a solicitar un score, un nuevo valor y si deseamos más valores

---

```
def delete(self):
    r, isConnected= connect()

    if isConnected:
        key= input("Key: ")
        value= input("Value: ")

        if r.exists(key):
            if r.zrem(key, value) == 1:
                print('Deleted successfully')
            else:
                print('Key does not exist in database. ')
    else:
        print('Could not connect to server.')
```

Si lo deseado es eliminar un sortedset, se solicita la dupla llave-valor para ver si existe, si esta dupla se encuentra mediante el uso de r.zrem (key, value), este conjunto de valores será eliminado, si no se encuentra nos notifica que no existe en la base de datos.