Redis

In this notebook you will find all the commands that we tried in class in the console and using the Python API for Redis.

Connecting to Redis database from Python

You can use the redis-py package to establish a connection to a redis instance. After a connection is established, you can use the python object (here we call it `r`) to execute redis commands.

In [3]:

```
import redis
r = redis.StrictRedis(host='bdl1.eng.tau.ac.il', port=6379)
```

Redis data types

Strings

The most basic type of a value in Redis is a string. Also integers are saved as strings.

This is an extract of console commands in redis-cli that set a key to a specific value, get the value of a key. There is also a small example of inconsistency (due to not ensuring ACID properties) and how to use the atomic function `incr` to avoid inconsistency.

Function that we will use:

```
set(name, value, ex=None, px=None, nx=False, xx=False)
```

Set the value at key name to value ex/px sets an expire flag on key name for ex/px seconds/milliseconds.

nx if set to True, set the value at key name to value if it does not already exist. xx if set to True, set the value at key name to value if it already exists.

```
get(name)
```

Return the value at key name, or None if the key doesn’t exist

---

```
incr(name, amount=1)
```

Increments the value of key by amount. If no key exists, the value will be initialized as amount

Examples

In [4]:

```
r.set('stud3:page:owner', 'tamir')
r.get('stud3:page:owner')
```

Out[4]:

```
'tamir'
```

In [5]:

```
r.set('stud3:views', 100)
r.get('stud3:views')
```

Out[5]:

```
'100'
```

In [6]:

```
print int(r.get('stud3:views'))+1
```

```
101
```

In [7]:

```
print r.get('stud3:views')
```

```
100
```

In [8]:

```
# print r.set('stud3:views', 101)
print r.incr('stud3:views')
```

```
101
```

In [7]:

```
print r.get('stud3:views')
```

```
101
```

When we try to increment (`incr`) a value that cannot be transformed to an integer, we get an error.

In [9]:

```
r.set('stud3:name','tamir')
r.get('stud3:name')
```

Out[9]:

```
'tamir'
```

In [10]:

```
r.type('stud3:name')
```

Out[10]:

```
'string'
```

In [11]:

```
print r.incr("stud3:name")
```

```
---------------------------------------------------------------------------
-
ResponseError                             Traceback (most recent call las
t)
<ipython-input-11-49c4a10a7e31> in <module>()
----> 1 print r.incr("stud3:name")

/opt/anaconda2/lib/python2.7/site-packages/redis/client.pyc in incr(self,
 name, amount)
    913         the value will be initialized as ``amount``
    914         """
--> 915         return self.execute_command('INCRBY', name, amount)
    916 
    917     def incrby(self, name, amount=1):

/opt/anaconda2/lib/python2.7/site-packages/redis/client.pyc in execute_com
mand(self, *args, **options)
    571         try:
    572             connection.send_command(*args)
--> 573             return self.parse_response(connection, command_name, *
*options)
    574         except (ConnectionError, TimeoutError) as e:
    575             connection.disconnect()

/opt/anaconda2/lib/python2.7/site-packages/redis/client.pyc in parse_respo
nse(self, connection, command_name, **options)
    583     def parse_response(self, connection, command_name, **options):
    584         "Parses a response from the Redis server"
--> 585         response = connection.read_response()
    586         if command_name in self.response_callbacks:
    587             return self.response_callbacks[command_name](response,
**options)

/opt/anaconda2/lib/python2.7/site-packages/redis/connection.pyc in read_re
sponse(self)
    580             raise
    581         if isinstance(response, ResponseError):
--> 582             raise response
    583         return response
    584 

ResponseError: value is not an integer or out of range
```

Some other useful operators:

```
type(name)
```

Returns the type of key name

```
exists(name)
```

Returns a boolean indicating whether key name exists

```
delete(*name)
```

Delete one or more keysspecified by names

In [12]:

```
r.exists('stud3:page:title')
```

Out[12]:

```
False
```

In [13]:

```
r.set('stud3:page:title', 'my blog')
r.exists('stud3:page:title')
```

Out[13]:

```
True
```

In [14]:

```
r.delete('stud3:page:title')
r.exists('stud3:page:title')
```

Out[14]:

```
False
```

Expiration times of keys

In Redis there is a nice feature that assigns an expiration time to a key. After the expiration time the key is deleted. (expiration times are set in seconds)

We can check how much time is left to a key before expiration using `ttl` (time to live).

Functions:

```
expire(name, time)
```

Set an expire flag on key name for time seconds. time can be represented by an integer or a Python timedelta object.

```
ttl(name)
```

Returns the number of seconds until the key name will expire

In [15]:

```
r.set('stud3:page:new', 'true')
r.expire('stud3:page:new', 10)
```

Out[15]:

```
True
```

How many second left? 'ttl' retuns the number of seconds until the key name will expire

In [18]:

```
r.ttl('stud3:page:new')
```

Out[18]:

```
5L
```

In [22]:

```
print r.get('stud3:page:new')
```

```
None
```

In [21]:

```
r.ttl('stud3:page:new')
```

Out[21]:

```
-2L
```

Lists (Linked Lists)

Another data type is the list. It is in principle similar to a python list, but the way elements are inserted and retrieved is different. We see here examples of adding a single element, from the left (`lpush`) and from the right (`rpush`), and how to retrieve the whole list or part of it (`lrange`).

Few functions:

`lpush(name, *values)` Push values onto the head of the list name

`lrange(name, start, end)` Return a slice of the list name between position start and end (start and end can be negative numbers just like Python slicing notation)

`rpop(name)` Remove and return the last item of the list name

`lpop(name)` Remove and return the first item of the list name

In [26]:

```
r.delete('stud3:users')
r.lpush('stud3:users', 1)
r.rpush('stud3:users', 2)
r.lpush('stud3:users', 0)
r.lrange('stud3:users', 0, -1)
```

Out[26]:

```
['0', '1', '2']
```

In [27]:

```
r.lrange('stud3:users', 1, 1)
```

Out[27]:

```
['1']
```

In [28]:

```
r.rpush('stud3:users', 3, 4, 5, 6, 7)
r.lrange('stud3:users', 0, -1)
```

Out[28]:

```
['0', '1', '2', '3', '4', '5', '6', '7']
```

Now we see how to extract an element from the end of a list (`rpop`).

```
rpush tocall 058XXXXXXX 054YYYYYYY 052ZZZZZZZ
rpop tocall
rpop tocall
rpop tocall
rpop tocall
```

In [29]:

```
r.delete('stud3:tocall')
```

Out[29]:

```
0
```

In [30]:

```
r.rpush('stud3:tocall', '058XXXXXXX', '054YYYYYYY', '052ZZZZZZZ')
```

Out[30]:

```
3L
```

In [31]:

```
r.rpop('stud3:tocall')
```

Out[31]:

```
'052ZZZZZZZ'
```

In [32]:

```
r.lrange("stud3:tocall",0,-1)
```

Out[32]:

```
['058XXXXXXX', '054YYYYYYY']
```

It is possible also to 'trim' a list, so that it would keep a specific number of elements. One example of use case of the trim function is to keep the latest k updates (e.g., the latest 5 posts on a blog).

`ltrim(name,start,end)` - removing all values not within the slice between 'start' and 'end'

In [33]:

```
r.delete('stud3:latest')
r.lpush('stud3:latest', 'tweet:1', 'tweet:2', 'tweet:3', 'tweet:4', 'tweet:5')
```

Out[33]:

```
5L
```

In [34]:

```
r.lrange('stud3:latest', 0, -1)
```

Out[34]:

```
['tweet:5', 'tweet:4', 'tweet:3', 'tweet:2', 'tweet:1']
```

In [35]:

```
r.ltrim('stud3:latest', 0, 3)
```

Out[35]:

```
True
```

In [36]:

```
r.lrange('stud3:latest', 0, -1)
```

Out[36]:

```
['tweet:5', 'tweet:4', 'tweet:3', 'tweet:2']
```

In [37]:

```
r.lpush('stud3:latest', 'tweet:6')
r.ltrim('stud3:latest', 0, 3)
```

Out[37]:

```
True
```

In [38]:

```
r.lrange('stud3:latest', 0, -1)
```

Out[38]:

```
['tweet:6', 'tweet:5', 'tweet:4', 'tweet:3']
```

In [39]:

```
r.lpush('stud3:latest', 'tweet:7')
r.ltrim('stud3:latest', 0, 3)
```

Out[39]:

```
True
```

In [40]:

```
r.lrange('stud3:latest', 0, -1)
```

Out[40]:

```
['tweet:7', 'tweet:6', 'tweet:5', 'tweet:4']
```

Exercise

You are managing a website and you want to store in your database a list of users registered to the website.

When a new user registers you want to save his id (that should be the last used id incremented by 1) and his nickname. You also want a function that shows the name of the last k users that registered to your website.

Create two functions `register_user(name)` and `last_users(k)` that implement the desired functionalities.

In [33]:

```
def register_user(name):
    #add your implementation
    if not r.exists('last_id'):
        r.set('last_id',0)
    new_id = r.incr('last_id')
    key = 'user:' + str(new_id) + 'name'
    r.rpush('ids',new_id)
    r.set(key,name)
    return r.get(key)
```

In [34]:

```
def last_users(k):
    #add your implementation
    last = r.lrange('ids',-k,-1)
    last_names = []
    for l in last:
        key = 'user:' + str(l) + 'name'
        last_names.append(r.get(key))
    return last_names
```

In [35]:

```
r.delete('ids')
usernames = ['laura', 'dean', 'itzik', 'alona']
k = 3
for u in usernames:
    print '--------------'
    print 'Welcome on our website', register_user(u)
    print 'Last users registered:', last_users(k)
```

```
--------------
Welcome on our website laura
Last users registered: ['laura']
--------------
Welcome on our website dean
Last users registered: ['laura', 'dean']
--------------
Welcome on our website itzik
Last users registered: ['laura', 'dean', 'itzik']
--------------
Welcome on our website alona
Last users registered: ['dean', 'itzik', 'alona']
```

Sets

Redis Sets are an unorder of Strings. It is possible to add, remove, and test for existence of members. Redis Sets not allowing repeated members. Practically speaking this means that adding a member does not require a check if exists add operation.

Few Functions

`sadd(name, *values)` Add value(s) to set name

`smembers(name)` Return all members of the set name

`sismember(name, value)` Return a boolean indicating if value is a member of set name

exampels:

In [41]:

```
r.sadd('stud3:page:keywords', 'intro', 'bio', 'about')
r.smembers('stud3:page:keywords')
```

Out[41]:

```
{'about', 'bio', 'intro'}
```

In [42]:

```
r.sismember('stud3:page:keywords', 'bio')
```

Out[42]:

```
True
```

In [43]:

```
r.sismember('stud3:page:keywords', 'work')
```

Out[43]:

```
False
```

In [44]:

```
r.sadd('stud3:page:keywords', 'help')
r.smembers('stud3:page:keywords')
```

Out[44]:

```
{'about', 'bio', 'help', 'intro'}
```

In [45]:

```
r.sadd('stud3:page:keywords', 'help')
r.smembers('stud3:page:keywords')
```

Out[45]:

```
{'about', 'bio', 'help', 'intro'}
```

Sorted Sets

Redis Sorted Sets are similar to Redis Sets, non repeating of Strings. The difference is that every member of Sorted Set is associated with score and order by that.

Function that we will use:

`zadd(name, *args, **kwargs)` Set any number of score, element-name pairs to the key name.

`zrange(name, start, end, desc=False, withscores=False, score_cast_func=<type 'float'>)` Return a range of values from sorted set name between start and end sorted in ascending order.

`zrangebyscore(name, min, max, start=None, num=None, withscores=False, score_cast_func=<type 'float'>)` Return a range of values from the sorted set name with scores between min and max.

`zrank(name, value)` Returns a 0-based value indicating the rank of value in sorted set name

`zincrby(name, value, amount=1)` Increment the score of value in sorted set name by amount

examples:

In [46]:

```
r.delete('stud3:sold-items')
r.zadd('stud3:sold-items', 100, "Trust webcam TRS342")
r.zadd('stud3:sold-items', 98, "Logitech mouse RTX220")
r.zadd('stud3:sold-items', 220, "D-link router DLAA20v")
r.zrange('stud3:sold-items', 0, -1)
```

Out[46]:

```
['Logitech mouse RTX220', 'Trust webcam TRS342', 'D-link router DLAA20v']
```

In [47]:

```
r.zrevrange('stud3:sold-items', 0, -1)
```

Out[47]:

```
['D-link router DLAA20v', 'Trust webcam TRS342', 'Logitech mouse RTX220']
```

In [48]:

```
r.zrevrange('stud3:sold-items', 0, -1, withscores=True)
```

Out[48]:

```
[('D-link router DLAA20v', 220.0),
 ('Trust webcam TRS342', 100.0),
 ('Logitech mouse RTX220', 98.0)]
```

In [49]:

```
r.zrangebyscore('stud3:sold-items', '-inf', 99)
```

Out[49]:

```
['Logitech mouse RTX220']
```

In [50]:

```
r.zrevrank('stud3:sold-items', "Logitech mouse RTX220")
```

Out[50]:

```
2
```

In [51]:

```
r.zincrby('stud3:sold-items', "Logitech mouse RTX220", amount=1)
```

Out[51]:

```
99.0
```

In [52]:

```
r.zincrby('stud3:sold-items', "Logitech mouse RTX220", amount=1)
```

Out[52]:

```
100.0
```

In [53]:

```
r.zrange('stud3:sold-items', 0, -1, withscores=True)
```

Out[53]:

```
[('Logitech mouse RTX220', 100.0),
 ('Trust webcam TRS342', 100.0),
 ('D-link router DLAA20v', 220.0)]
```

In [54]:

```
r.zincrby('stud3:sold-items', "Logitech mouse RTX220", amount=1)
```

Out[54]:

```
101.0
```

In [55]:

```
r.zrevrange('stud3:sold-items', 0, -1, withscores=True)
```

Out[55]:

```
[('D-link router DLAA20v', 220.0),
 ('Logitech mouse RTX220', 101.0),
 ('Trust webcam TRS342', 100.0)]
```

In [56]:

```
def show_leaderboard(setname):
    scores = r.zrevrange(setname, 0, -1, withscores=True)
    print '----------------------------------------------------'
    i = 1
    for item in scores:
        print i, '-', item[0], '('+str(int(item[1]))+' points)'
        i = i + 1
    print '----------------------------------------------------'
```

In [57]:

```
show_leaderboard('stud3:sold-items')
```

```
----------------------------------------------------
1 - D-link router DLAA20v (220 points)
2 - Logitech mouse RTX220 (101 points)
3 - Trust webcam TRS342 (100 points)
----------------------------------------------------
```

Reference

https://redis-py.readthedocs.io/en/latest/ (https://redis-py.readthedocs.io/en/latest/)