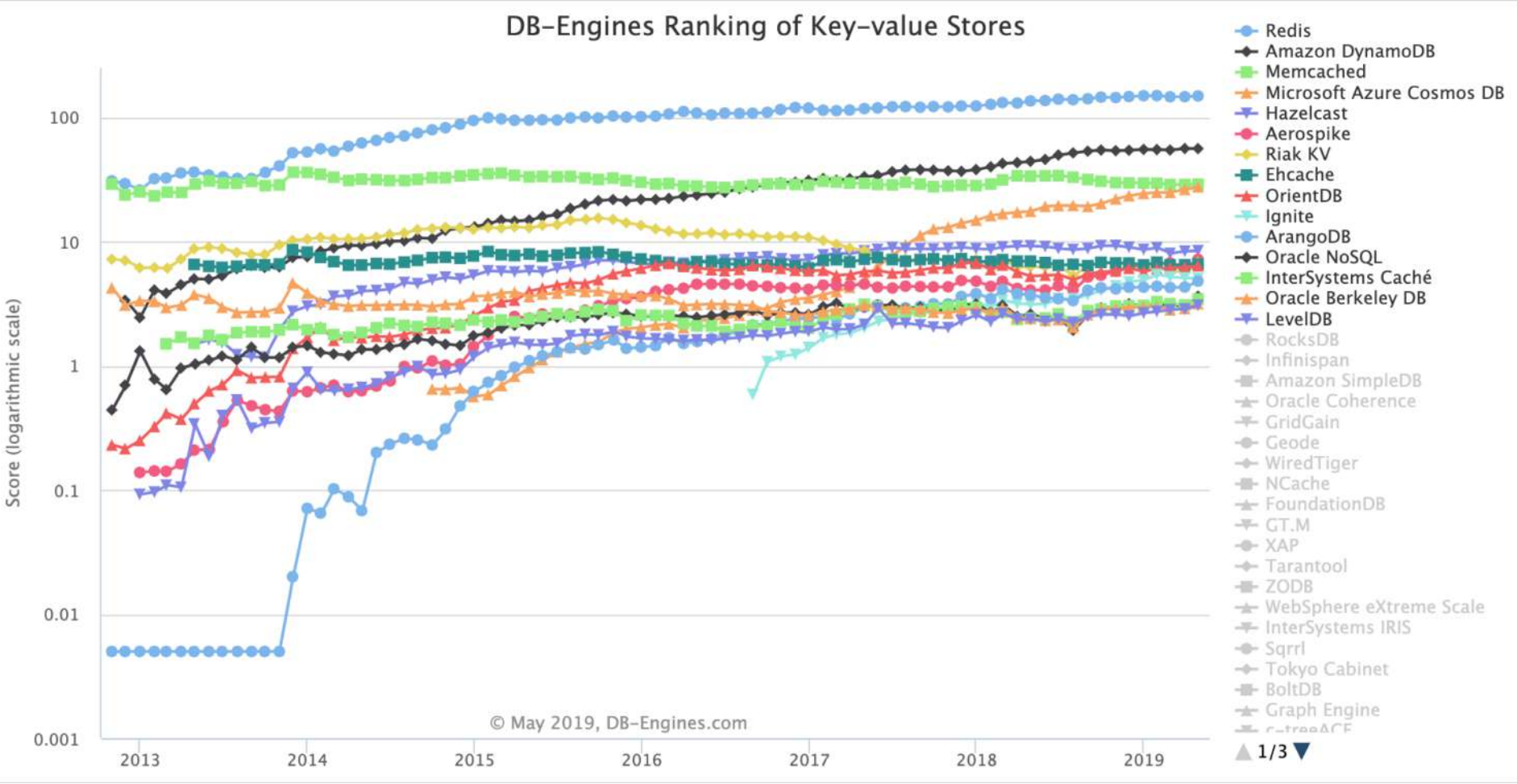
In-Memory
Key-value cache and store

347 systems in ranking, May 2019

Rank May 2019	Rank Apr 2019	Rank May 2018	DBMS	Database Model	Score May 2019	Score Apr 2019	Score May 2018
1.	1.	1.	Oracle	Relational, Multi-model	1285.55	+5.61	-4.87
2.	2.	2.	MySQL	Relational, Multi-model	1218.96	+3.82	-4.38
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1072.19	+12.23	-13.66
4.	4.	4.	PostgreSQL	Relational, Multi-model	478.89	+0.17	+77.99
5.	5.	5.	MongoDB	Document	408.07	+6.10	+65.96
6.	6.	6.	IBM Db2	Relational, Multi-model	174.44	-1.61	-11.17
7.	↑ 8.	↑ 9.	Elasticsearch	Search engine, Multi-model	148.62	+2.62	+18.18
8.	↓ 7.	↓ 7.	Redis	Key-value, Multi-model	148.40	+2.03	+13.06
9.	9.	↓ 8.	Microsoft Access	Relational	143.78	-0.87	+10.67
10.	↑ 11.	10.	Cassandra	Wide column	125.72	+2.11	+7.89

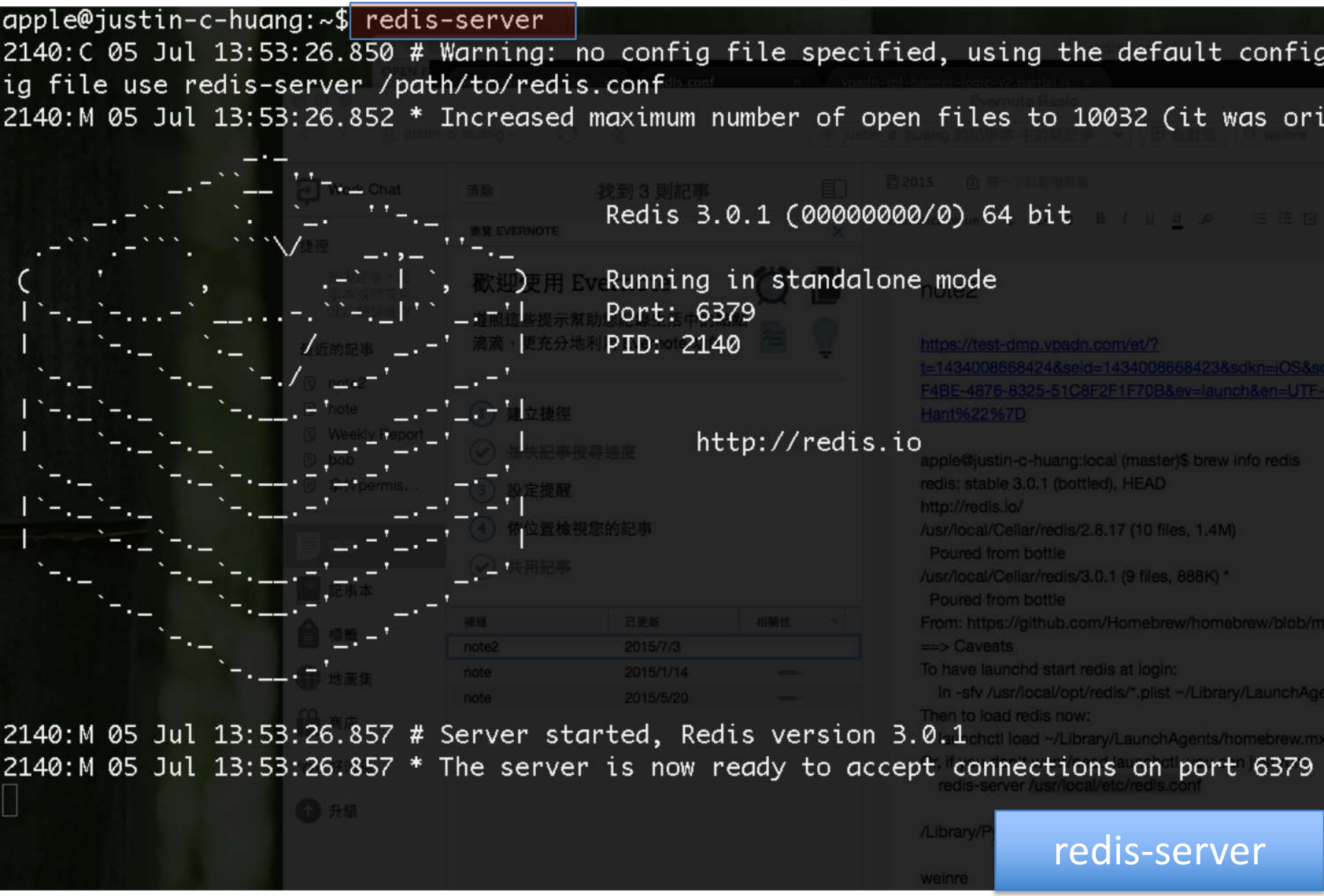
67 systems in ranking, May 2019

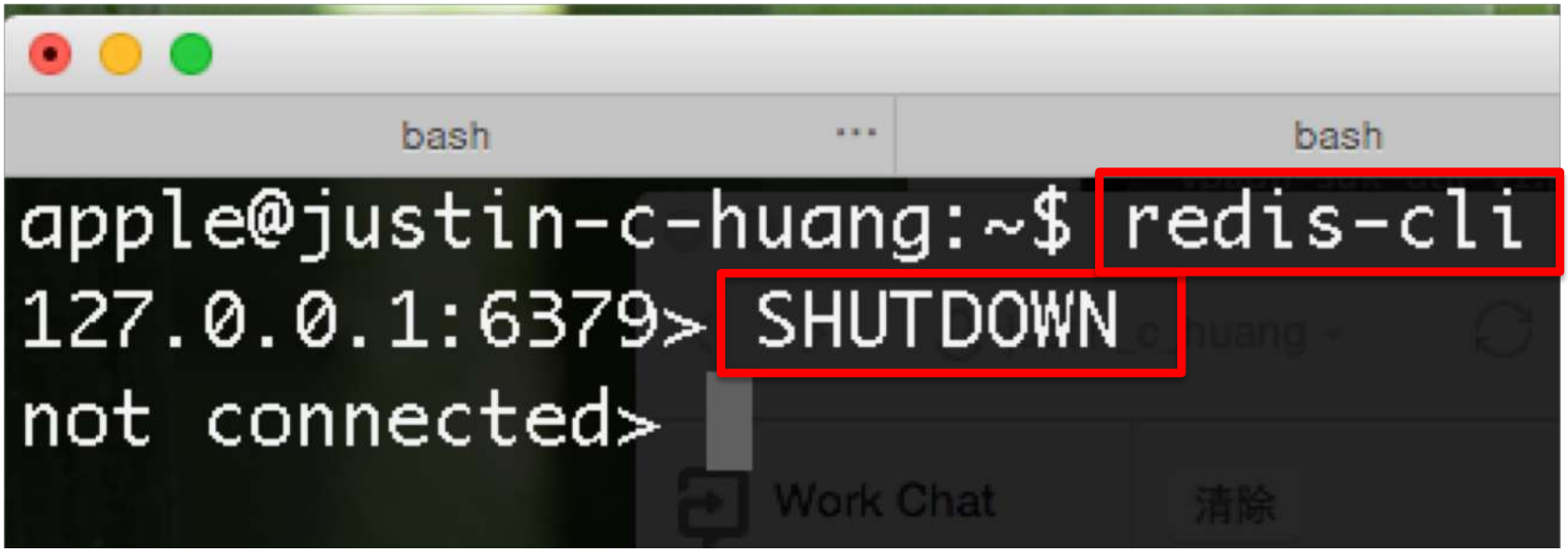
Rank May 2019	Rank Apr 2019	Rank May 2018	DBMS	Database Model	Score May 2019	Score Apr 2019	Score May 2018
1.	1.	1.	Redis	Key-value, Multi-model	148.40	+2.03	+13.06
2.	2.	2.	Amazon DynamoDB	Multi-model	55.93	-0.08	+11.74
3.	3.	3.	Memcached	Key-value	28.90	+0.17	-4.66
4.	4.	4.	Microsoft Azure Cosmos DB	Multi-model	27.59	+1.32	+10.06
5.	5.	5.	Hazelcast	Key-value, Multi-model	8.43	+0.07	-0.79
6.	↑ 7.	↑ 9.	Aerospike	Key-value	7.17	+0.96	+3.11
7.	↑ 9.	7.	Riak KV	Key-value	6.80	+1.10	+0.55
8.	↓ 6.	↓ 6.	Ehcache	Key-value	6.59	+0.10	-0.34
9.	↓ 8.	↓ 8.	OrientDB	Multi-model	6.37	+0.18	+1.12
10.	10.	↑ 11.	Ignite	Multi-model	5.05	-0.04	+1.93
11.	11.	↓ 10.	ArangoDB	Multi-model	4.79	+0.50	+1.09
12.	↑ 13.	12.	Oracle NoSQL	Key-value, Multi-model	3.62	+0.65	+1.02
13.	↓ 12.	13.	InterSystems Caché	Multi-model	3.44	+0.36	+1.02
14.	14.	14.	Oracle Berkeley DB	Multi-model	3.16	+0.27	+0.75
15.	15.	15.	LevelDB	Key-value	3.07	+0.21	+0.70
16.	16.	↑ 19.	RocksDB	Key-value	2.86	+0.27	+1.31
17.	17.	17.	Infinispan	Key-value	2.76	+0.30	+0.54
18.	↑ 20.	18.	Amazon SimpleDB	Key-value	2.70	+0.38	+0.67
19.	↓ 18.	↓ 16.	Oracle Coherence	Key-value	2.67	+0.27	+0.42
20.	↓ 19.	20.	GridGain	Multi-model	1.85	-0.50	+0.68

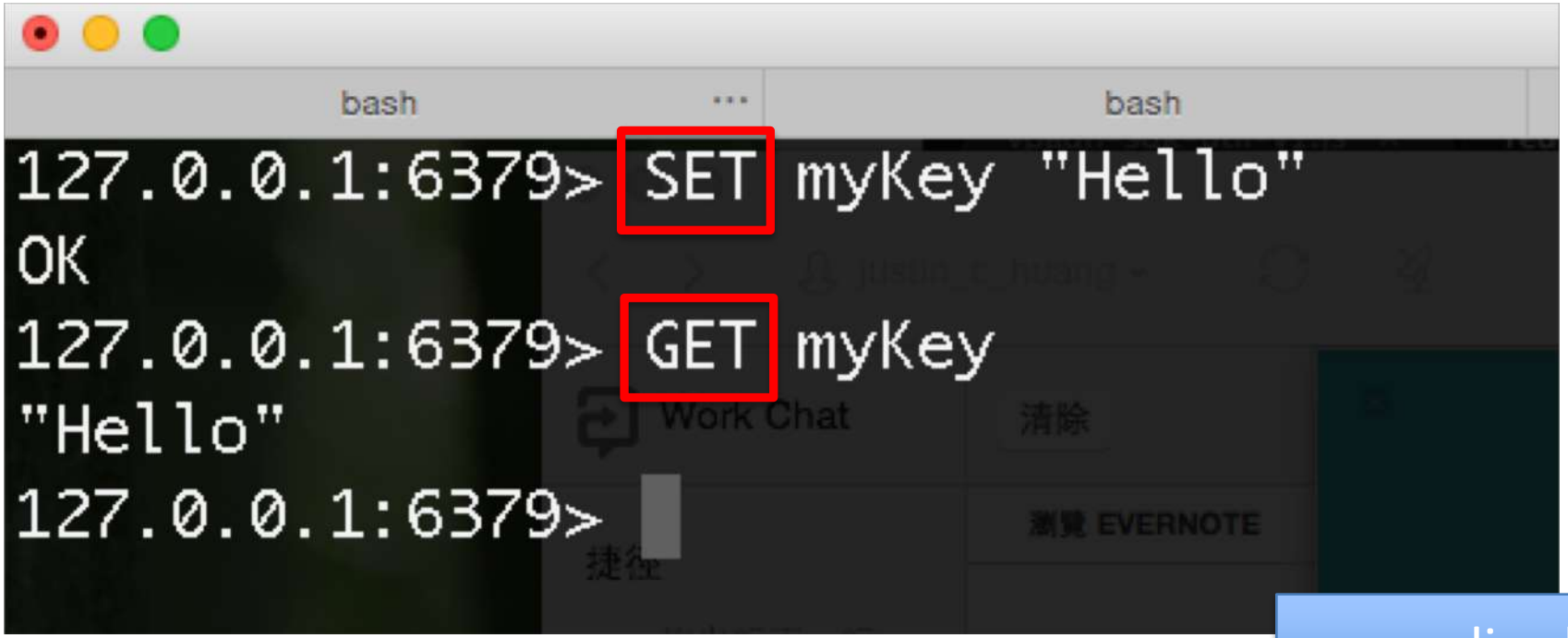

DB-Engines Ranking of Key-value Stores
Score (logarithmic scale)
100
10
1
0.1
0.01
0.001
2013
2014
2015
2016
2017
2018
2019
© May 2019, DB-Engines.com
Redis
Amazon DynamoDB
Memcached
Microsoft Azure Cosmos DB
Hazelcast
Aerospike
Riak KV
Ehcache
OrientDB
Ignite
ArangoDB
Oracle NoSQL
InterSystems Caché
Oracle Berkeley DB
LevelDB
RocksDB
Infinispan
Amazon SimpleDB
Oracle Coherence
GridGain
Geode
WiredTiger
NCache
FoundationDB
GT.M
XAP
Tarantool
ZODB
WebSphere eXtreme Scale
InterSystems IRIS
Sqrrl
Tokyo Cabinet
BoltDB
Graph Engine
1/3

About Redis

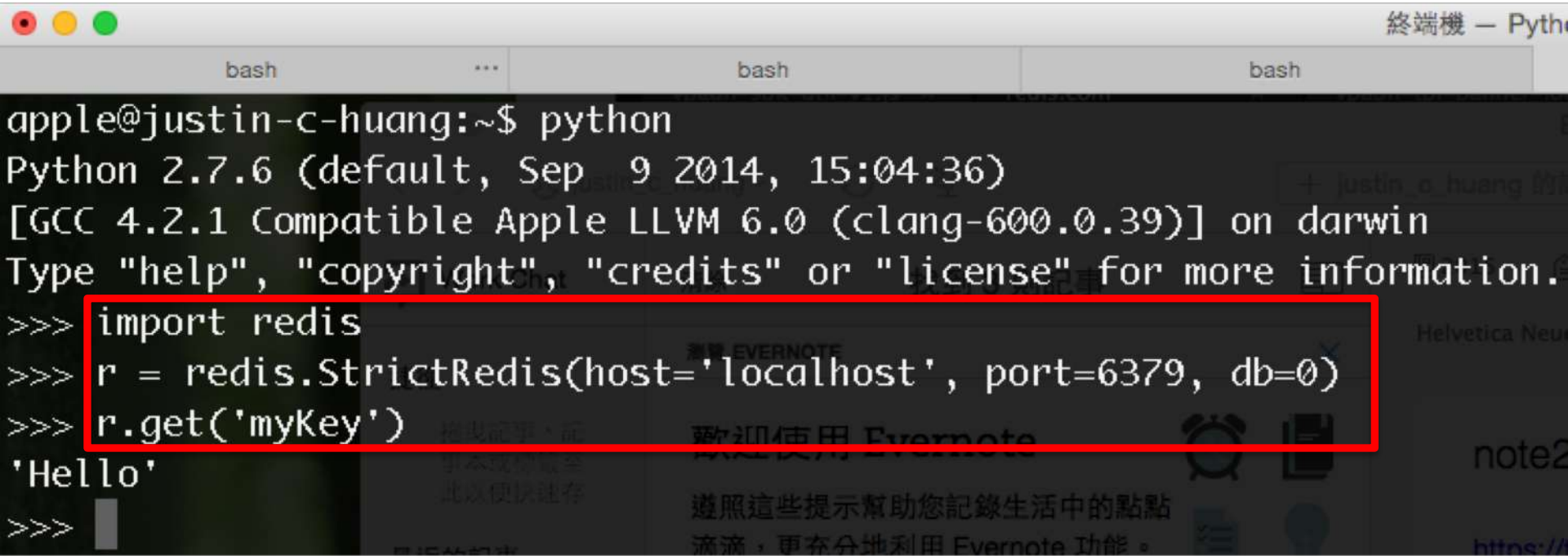
- Redis = **Re**mote **DI**ctionary **S**erver
- Initial release in 2009
- Written in C
- Good documentation
- In-memory but persistent on disk database


apple@justin-c-huang:~$ redis-server
2140:C 05 Jul 13:53:26.850 # Warning: no config file specified, using the default config
ig file use redis-server /path/to/redis.conf
2140:M 05 Jul 13:53:26.852 * Increased maximum number of open files to 10032 (it was ori
Redis 3.0.1 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 2140
http://redis.io
2140:M 05 Jul 13:53:26.857 # Server started, Redis version 3.0.1
2140:M 05 Jul 13:53:26.857 * The server is now ready to accept connections on port 6379
redis-server


bash
bash
apple@justin-c-huang:~$ redis-cli
127.0.0.1:6379> SHUTDOWN
not connected>
Work Chat
清除


bash
bash
127.0.0.1:6379> SET myKey "Hello"
OK
127.0.0.1:6379> GET myKey
"Hello"
127.0.0.1:6379>
Work Chat
清除
瀏覽 EVERNOTE

redis-cli


終端機 — Pyth
bash
bash
bash
apple@justin-c-huang:~$ python
Python 2.7.6 (default, Sep 9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>> r = redis.StrictRedis(host='localhost', port=6379, db=0)
>>> r.get('myKey')
'Hello'
>>>

python

Data Types

- String
- List
- Set
- Hash
- Sorted set (zset)

Data Type -- String

- Redis Strings are **binary safe**, this means that a **Redis string can contain any kind of data**, for instance a JPEG image or a serialized Ruby object.
- A String value can be at max 512 MB in length.

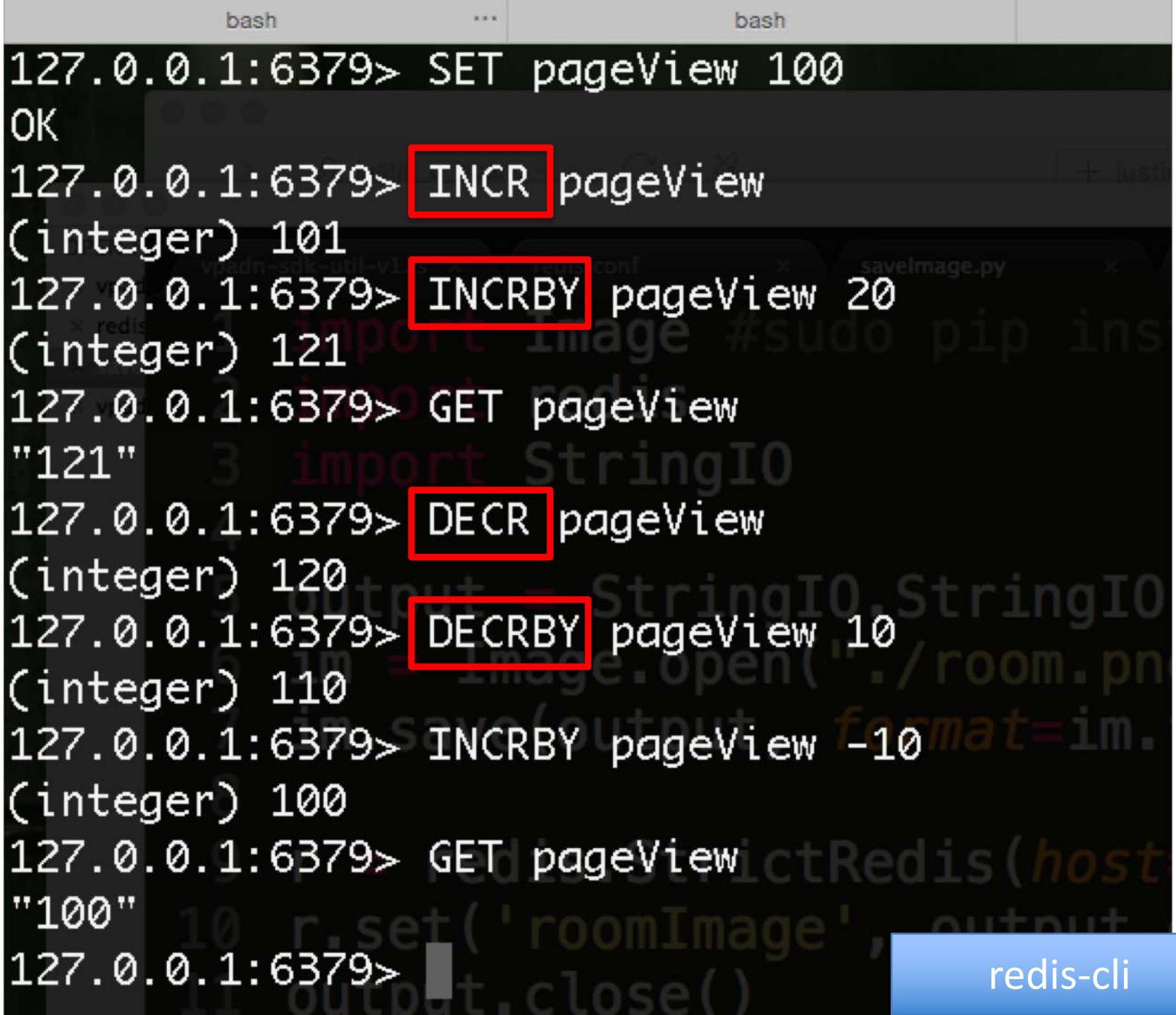
```python
import Image #sudo pip install pil
import redis
import StringIO

output = StringIO.StringIO()
im = Image.open("./room.png")
im.save(output, format=im.format)

r = redis.StrictRedis(host='localhost')
r.set('roomImage', output.getvalue())
output.close()
```

```
apple@justin-c-huang:iii_redis$ redis-cli --raw get 'roomImage' >test.png
```

python

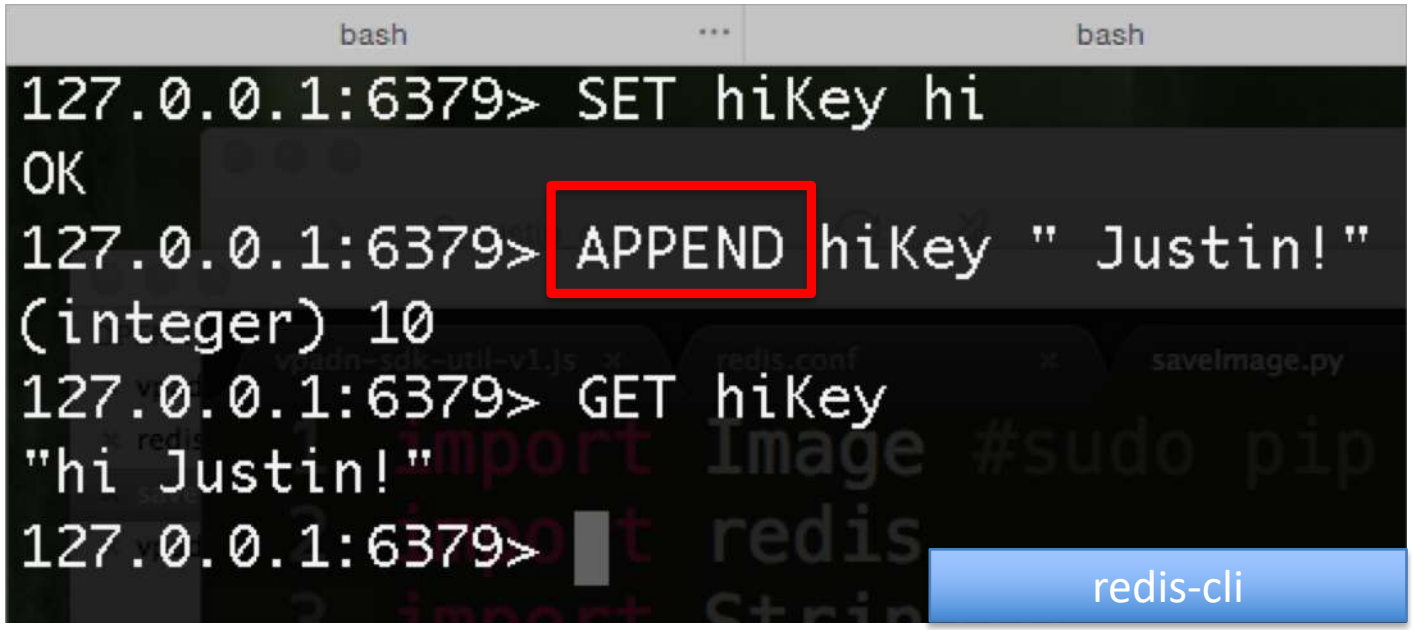
String Commands

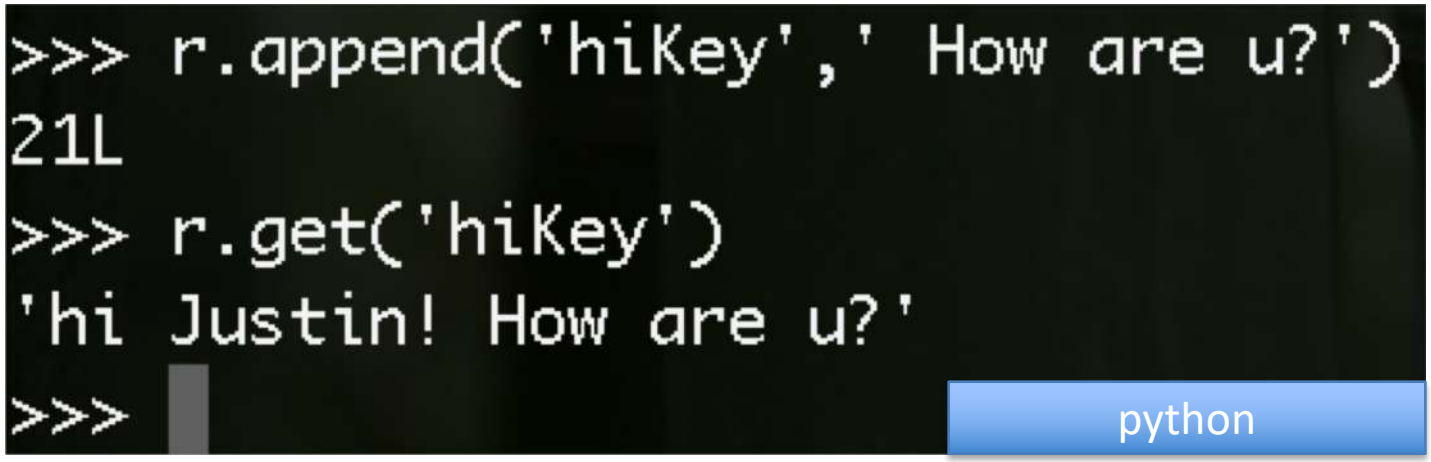
- GET, SET
- INCR, INCRBY
- DECR, DECRBY
- APPEND
- STRLEN
- MGET,MSET


bash
bash
127.0.0.1:6379> SET pageView 100
OK
127.0.0.1:6379> INCR pageView
(integer) 101
127.0.0.1:6379> INCRBY pageView 20
(integer) 121
127.0.0.1:6379> GET pageView
"121"
127.0.0.1:6379> DECR pageView
(integer) 120
127.0.0.1:6379> DECRBY pageView 10
(integer) 110
127.0.0.1:6379> INCRBY pageView -10
(integer) 100
127.0.0.1:6379> GET pageView
"100"
127.0.0.1:6379>
redis-cli

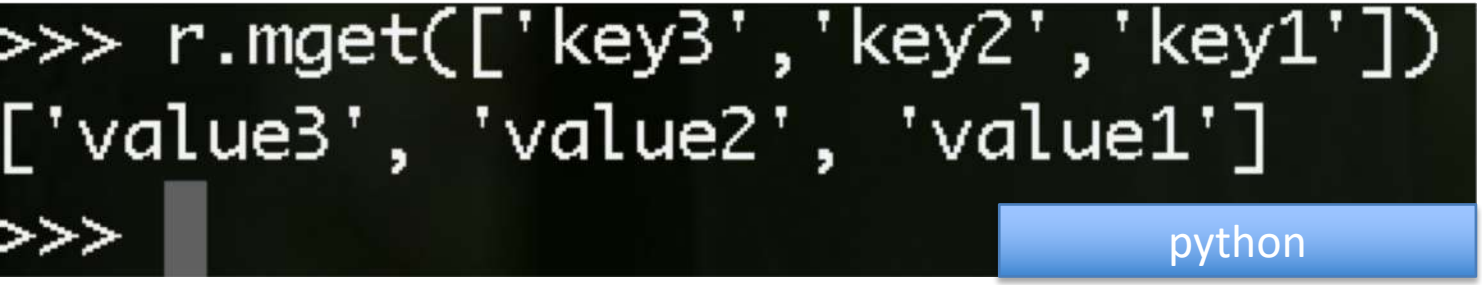
```
>>> import redis
>>> r = redis.StrictRedis()
>>> r.get('pageView')
'100'
>>> r.incr('pageView',30)
130
>>> r.get('pageView')
'130'
>>> r.decr('pageView',15)
115
>>> r.incr('pageView',-15)
100
>>> r.get('pageView')
'100'
>>>
```

python


bash
bash
127.0.0.1:6379> SET hiKey hi
OK
127.0.0.1:6379> APPEND hiKey " Justin!"
(integer) 10
127.0.0.1:6379> GET hiKey
"hi Justin!"
127.0.0.1:6379>
redis-cli

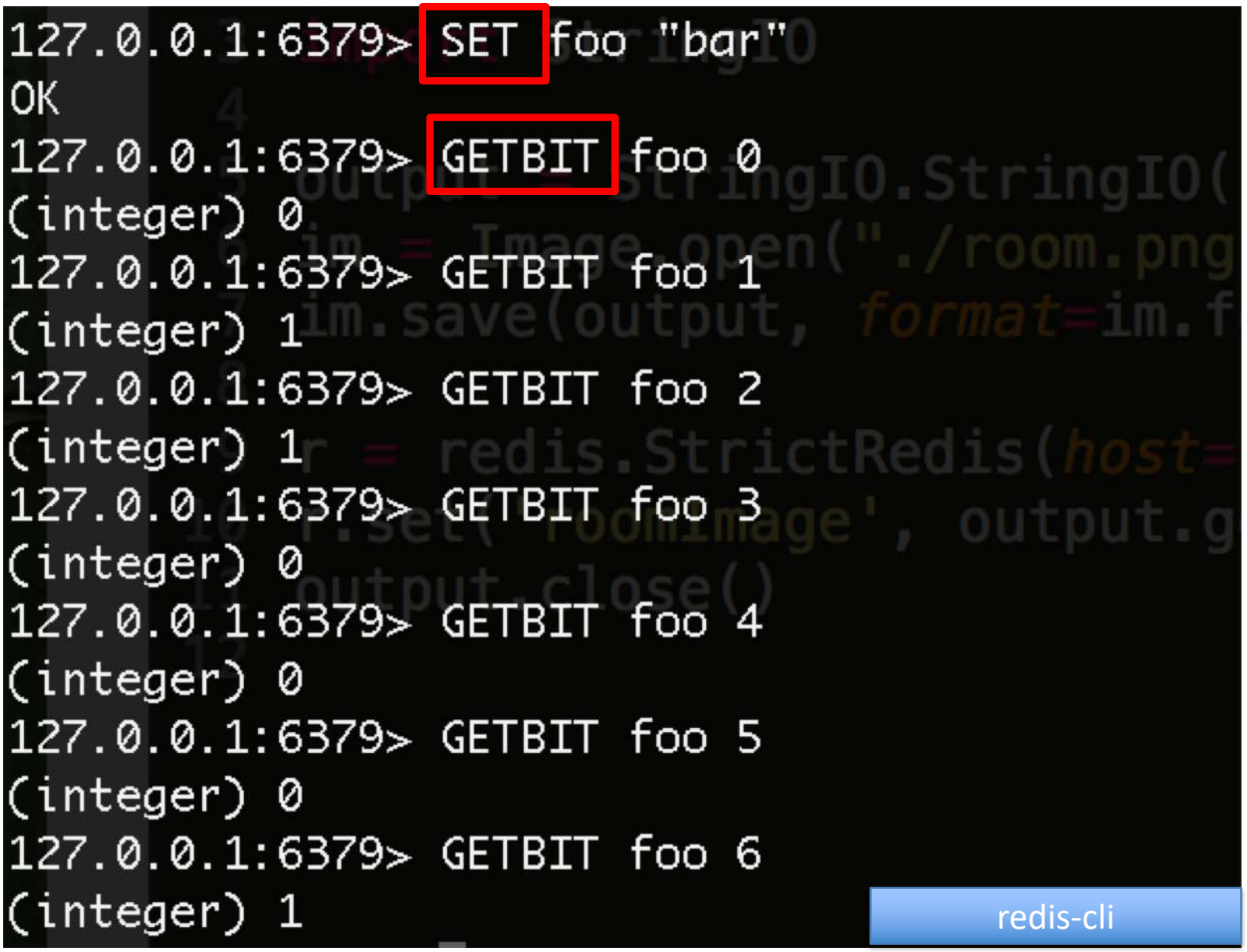

>>> r.append('hiKey',' How are u?')
21L
>>> r.get('hiKey')
'hi Justin! How are u?'
>>>
python


127.0.0.1:6379> MSET key1 'value1' key2 'value2' key3 'value3'
OK
127.0.0.1:6379> GET key2
"value2"
127.0.0.1:6379> MGET key1 key3
1) "value1"
2) "value3"
127.0.0.1:6379>
redis-cli


>>> r.mget(['key3','key2','key1'])
['value3', 'value2', 'value1']
>>>
python

位元操作

- SET foo **bar**
- **01100010|01100001|01110010**
- GETBIT,SETBIT
- BITCOUNT
- BITOP


127.0.0.1:6379> SET foo "bar"
OK
127.0.0.1:6379> GETBIT foo 0
(integer) 0
127.0.0.1:6379> GETBIT foo 1
(integer) 1
127.0.0.1:6379> GETBIT foo 2
(integer) 1
127.0.0.1:6379> GETBIT foo 3
(integer) 0
127.0.0.1:6379> GETBIT foo 4
(integer) 0
127.0.0.1:6379> GETBIT foo 5
(integer) 0
127.0.0.1:6379> GETBIT foo 6
(integer) 1
redis-cli

01100010|01100001|01110010

```
127.0.0.1:6379>
127.0.0.1:6379> SETBIT foo 6 0
(integer) 1
127.0.0.1:6379> SETBIT foo 7 1
(integer) 0
127.0.0.1:6379> GET foo
"aar"
127.0.0.1:6379>
```

```
127.0.0.1:6379> BITCOUNT foo
(integer) 10
127.0.0.1:6379> BITCOUNT foo 0 1
(integer) 6
127.0.0.1:6379>
```

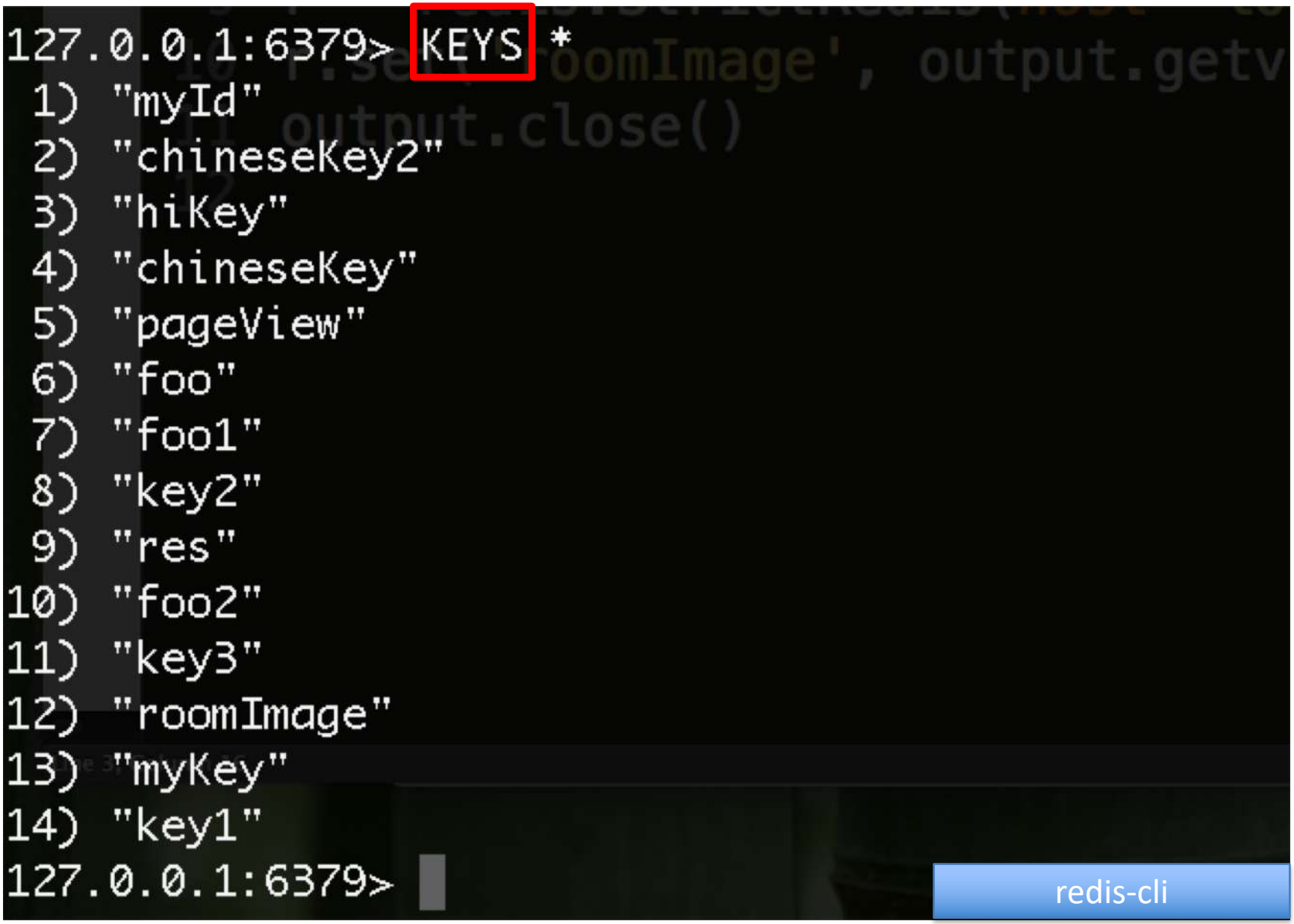
redis-cli

01100010|01100001|01110010

01100001|01100001|01110010

```
127.0.0.1:6379> SET foo1 bar
OK
127.0.0.1:6379> SET foo2 aar
OK
127.0.0.1:6379> BITOP OR res foo1 foo2
(integer) 3
127.0.0.1:6379> GET res
"car"
127.0.0.1:6379>
```

redis-cli


127.0.0.1:6379> KEYS *
1) "myId"
2) "chineseKey2"
3) "hiKey"
4) "chineseKey"
5) "pageView"
6) "foo"
7) "foo1"
8) "key2"
9) "res"
10) "foo2"
11) "key3"
12) "roomImage"
13) "myKey"
14) "key1"
127.0.0.1:6379>
redis-cli

GET key

> **Available since 1.0.0.**
>
> **Time complexity:** O(1)

Get the value of key. If the key does not exist the special value `nil` is returned. An error is returned if the value stored at key is not a string, because GET only handles string values.

Return value

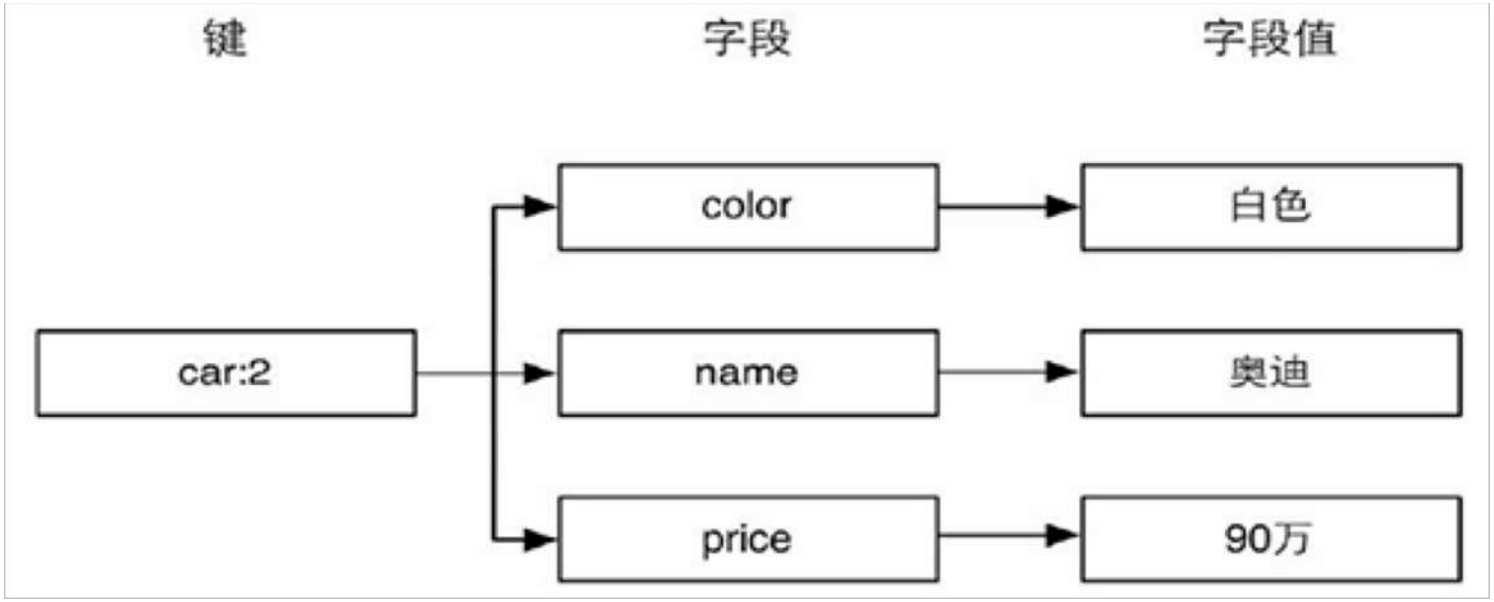
Bulk string reply: the value of key, or `nil` when key does not exist.

Examples

```
redis> GET nonexisting
(nil)
redis> SET mykey "Hello"
OK
redis> GET mykey
"Hello"
redis>
```

Data Type--Hash

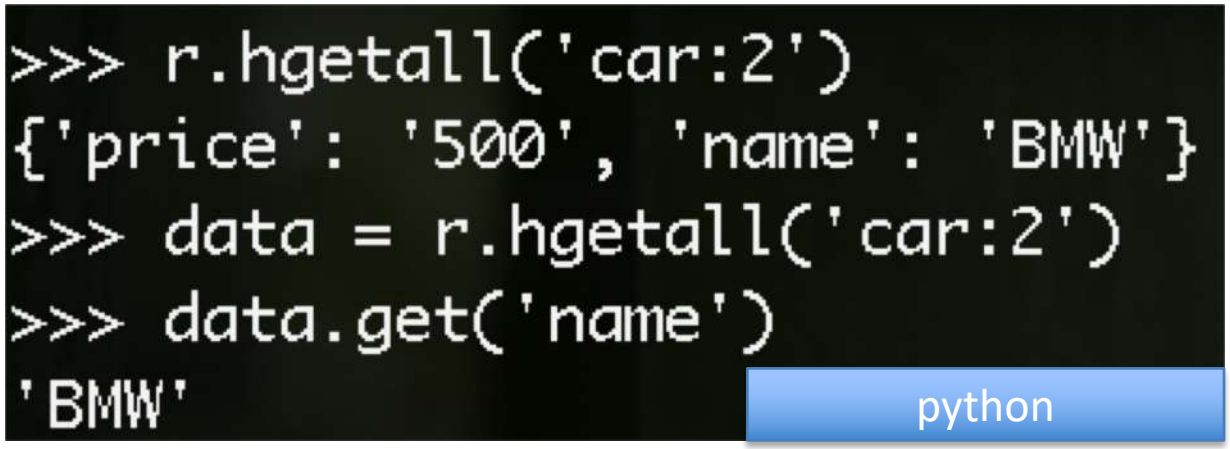
- Redis Hashes are maps between string **fields** and **string** values, so they are the perfect data type to represent objects



Hash Commands

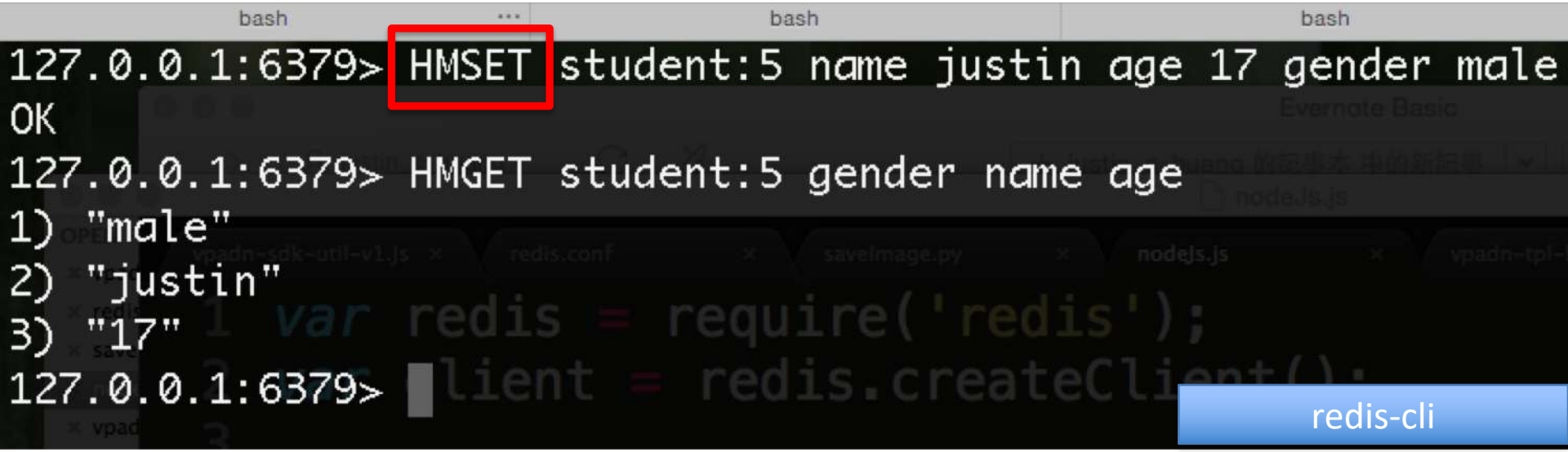
- HGET, HSET,
- HGETALL
- HMSET,HMGET
- HEXISTS

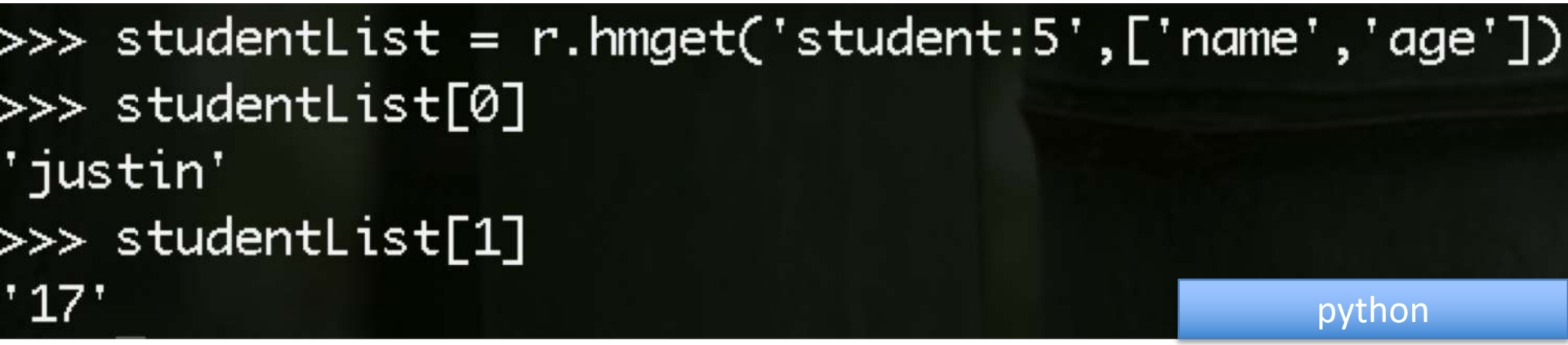

bash
bash
127.0.0.1:6379> HSET car:2 price 500
(integer) 1
127.0.0.1:6379> HSET car:2 name BMW
(integer) 1
127.0.0.1:6379> HGET car:2 name
"BMW"
127.0.0.1:6379> HGETALL car:2
1) "price"
2) "500"
3) "name"
4) "BMW"
127.0.0.1:6379>
redis-cli


>>> r.hgetall('car:2')
{'price': '500', 'name': 'BMW'}
>>> data = r.hgetall('car:2')
>>> data.get('name')
'BMW'
python

```
var redis = require('redis');
var client = redis.createClient();

client.on('connect', function() {
    console.log('connected');
    client.hgetall('car:2',function(err,obj){
        console.log(obj.price);
        console.log(obj.name);
    });
});
```

Node.js


bash
bash
bash
127.0.0.1:6379> HMSET student:5 name justin age 17 gender male
OK
127.0.0.1:6379> HMGET student:5 gender name age
1) "male"
2) "justin"
3) "17"
127.0.0.1:6379>
redis-cli


>>> studentList = r.hmget('student:5',['name','age'])
>>> studentList[0]
'justin'
>>> studentList[1]
'17'
python

```
127.0.0.1:6379> HEXISTS student:5 score
(integer) 0
127.0.0.1:6379> HEXISTS student:5 age
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> HSETNX student:5 score 60
(integer) 1
127.0.0.1:6379> HEXISTS student:5 score
(integer) 1
127.0.0.1:6379> HGET student:5 score
"60"
127.0.0.1:6379>
```

redis-cli

python

```
>>> r.hexists('student:5','weight')
False
>>> r.hsetnx('student:5','weight',65)
1L
>>> r.hexists('student:5','weight')
True
>>> r.hgetall('student:5')
{'gender': 'male', 'age': '17', 'score': '60', 'name': 'justin', 'weight': '65'}
>>>
```

redis-cli

```
127.0.0.1:6379> HINCRBY student:5 score 20
(integer) 80
127.0.0.1:6379> HINCRBY student:5 score -10
(integer) 70
127.0.0.1:6379> HGETALL student:5
 1) "name"
 2) "justin"
 3) "age"
 4) "17"
 5) "gender"
 6) "male"
 7) "score"
 8) "70"
 9) "weight"
10) "65"
127.0.0.1:6379> HDEL student:5 gender
(integer) 1
127.0.0.1:6379> HGETALL student:5
1) "name"
2) "justin"
3) "age"
4) "17"
5) "score"
6) "70"
7) "weight"
8) "65"
127.0.0.1:6379>
```

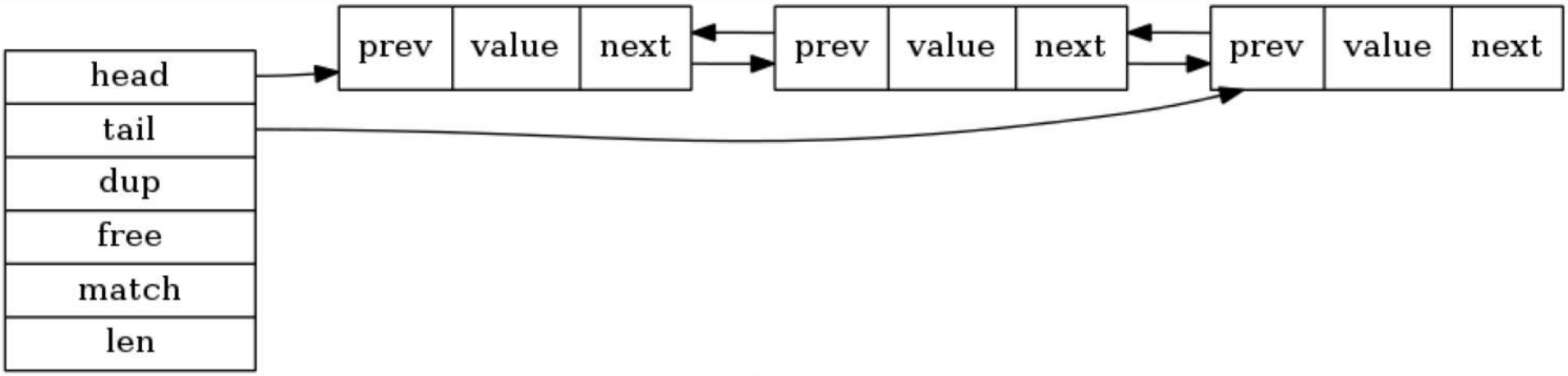
```
>>> r.hkeys('student:5')
['name', 'age', 'score', 'weight']
>>> r.hvals('student:5')
['justin', '17', '70', '65']
>>>
```

python

Data type--List

- Redis Lists are simply **lists of strings**, **sorted by insertion order**. It is possible to add elements to a Redis List pushing new elements on the head (on the left) or on the tail (on the right) of the list.

```
LPUSH mylist a   # now the list is "a"
LPUSH mylist b   # now the list is "b","a"
RPUSH mylist c   # now the list is "b","a","c" (RPUSH was used this time)
```


head
tail
dup
free
match
len
prev
value
next
prev
value
next
prev
value
next

双端链表与节点

List Commands

- LPUSH, RPUSH
- LPOP,RPOP
- LRANGE
- LINDEX
- LREM
- LSET
- LINSERT
- RPOPLPUSH

```
127.0.0.1:6379> LPUSH names Justin
(integer) 1
127.0.0.1:6379> LPUSH names Alan
(integer) 2
127.0.0.1:6379> LPUSH names Mavis
(integer) 3
127.0.0.1:6379> LPOP names
"Mavis"
127.0.0.1:6379> LPOP names
"Alan"
127.0.0.1:6379> LPOP names
"Justin"
```


```
127.0.0.1:6379> LPUSH names Justin
(integer) 1
127.0.0.1:6379> LPUSH names Alan
(integer) 2
127.0.0.1:6379> LPUSH names Mavis
(integer) 3
127.0.0.1:6379> RPOP names
"Justin"
127.0.0.1:6379> RPOP names
"Alan"
127.0.0.1:6379> RPOP names
"Mavis"
```

```
127.0.0.1:6379> LPUSH names Justin Alan Mavis Kelly James Hopper Lisa
(integer) 7
127.0.0.1:6379> LRANGE names 2 5
1) "James"
2) "Kelly"
3) "Mavis"
4) "Alan"
127.0.0.1:6379> LRANGE names 0 -1
1) "Lisa"
2) "Hopper"
3) "James"
4) "Kelly"
5) "Mavis"
6) "Alan"
7) "Justin"
127.0.0.1:6379>
```

```
127.0.0.1:6379> LRANGE names 0 -1
1) "Lisa"
2) "Hopper"
3) "James"
4) "Kelly"
5) "Mavis"
6) "Alan"
7) "Justin"
127.0.0.1:6379> LINDEX names 0
"Lisa"
127.0.0.1:6379> LINDEX names 1
"Hopper"
127.0.0.1:6379> LINDEX names 2
"James"
127.0.0.1:6379> LINDEX names -1
"Justin"
127.0.0.1:6379>
```

Lisa	Hopper	James	Kelly	Mavis	Alan	Justin

LINDEX key index

Available since 1.0.0.

Time complexity: O(N) where N is the number of elements to traverse to get to the element at index. This makes asking for the first or the last element of the list O(1).

Returns the element at index `index` in the list stored at `key`. The index is zero-based, so `0` means the first element, `1` the second element and so on. Negative indices can be used to designate elements starting at the tail of the list. Here, `-1` means the last element, `-2` means the penultimate and so forth.

When the value at `key` is not a list, an error is returned.

Return value

Bulk string reply: the requested element, or `nil` when `index` is out of range.

eclipse ··· root@datainfa-ng...n/social

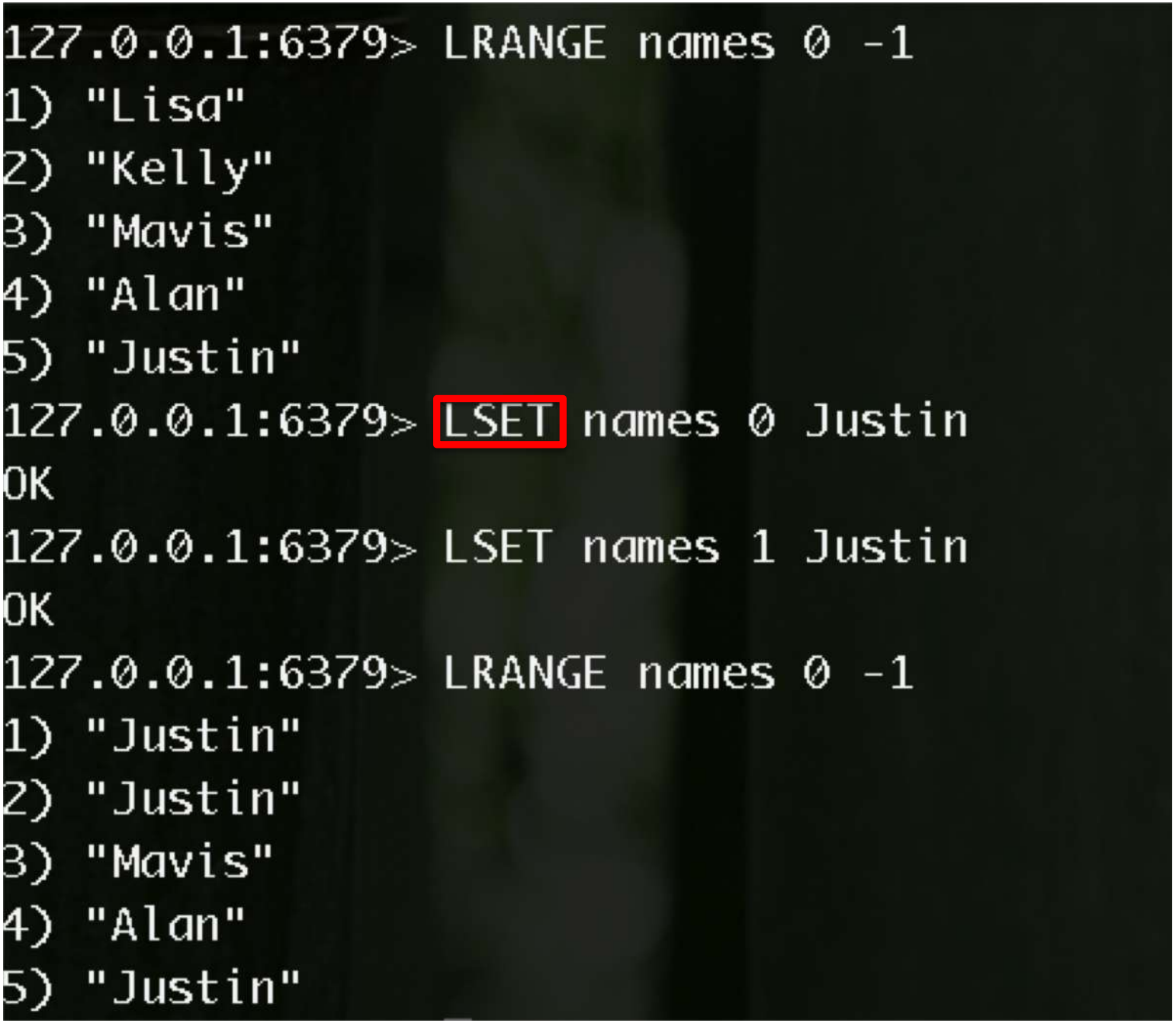
```
127.0.0.1:6379> LREM names 5 Hopper
(integer) 1
127.0.0.1:6379> LRANGE names 0 -1
1) "Lisa"
2) "James"
3) "Kelly"
4) "Mavis"
5) "Alan"
6) "Justin"
127.0.0.1:6379>
```

LREM key count value

> **Available since 1.0.0.**
>
> **Time complexity:** O(N) where N is the length of the list.

Removes the first `count` occurrences of elements equal to `value` from the list stored at `key`. The `count` argument influences the operation in the following ways:

- `count > 0`: Remove elements equal to `value` moving from head to tail.
- `count < 0`: Remove elements equal to `value` moving from tail to head.
- `count = 0`: Remove all elements equal to `value`.


127.0.0.1:6379> LRANGE names 0 -1
1) "Lisa"
2) "Kelly"
3) "Mavis"
4) "Alan"
5) "Justin"
127.0.0.1:6379> LSET names 0 Justin
OK
127.0.0.1:6379> LSET names 1 Justin
OK
127.0.0.1:6379> LRANGE names 0 -1
1) "Justin"
2) "Justin"
3) "Mavis"
4) "Alan"
5) "Justin"

```
127.0.0.1:6379> LRANGE names 0 -1
1) "Justin"
2) "Justin"
3) "Mavis"
4) "Alan"
5) "Justin"
127.0.0.1:6379> LINSERT names before Alan Hopper
(integer) 6
127.0.0.1:6379> LRANGE names 0 -1
1) "Justin"
2) "Justin"
3) "Mavis"
4) "Hopper"
5) "Alan"
6) "Justin"
127.0.0.1:6379> LINSERT names after Alan Lisa
(integer) 7
127.0.0.1:6379> LRANGE names 0 -1
1) "Justin"
2) "Justin"
3) "Mavis"
4) "Hopper"
5) "Alan"
6) "Lisa"
7) "Justin"
```

```
127.0.0.1:6379> LPUSH score 10 20 30 40 50 60
(integer) 6
127.0.0.1:6379> RPOPLPUSH score score
"10"
127.0.0.1:6379> LRANGE score 0 -1
1) "10"
2) "60"
3) "50"
4) "40"
5) "30"
6) "20"
127.0.0.1:6379> RPOPLPUSH score score
"20"
127.0.0.1:6379> LRANGE score 0 -1
1) "20"
2) "10"
3) "60"
4) "50"
5) "40"
6) "30"
```

python

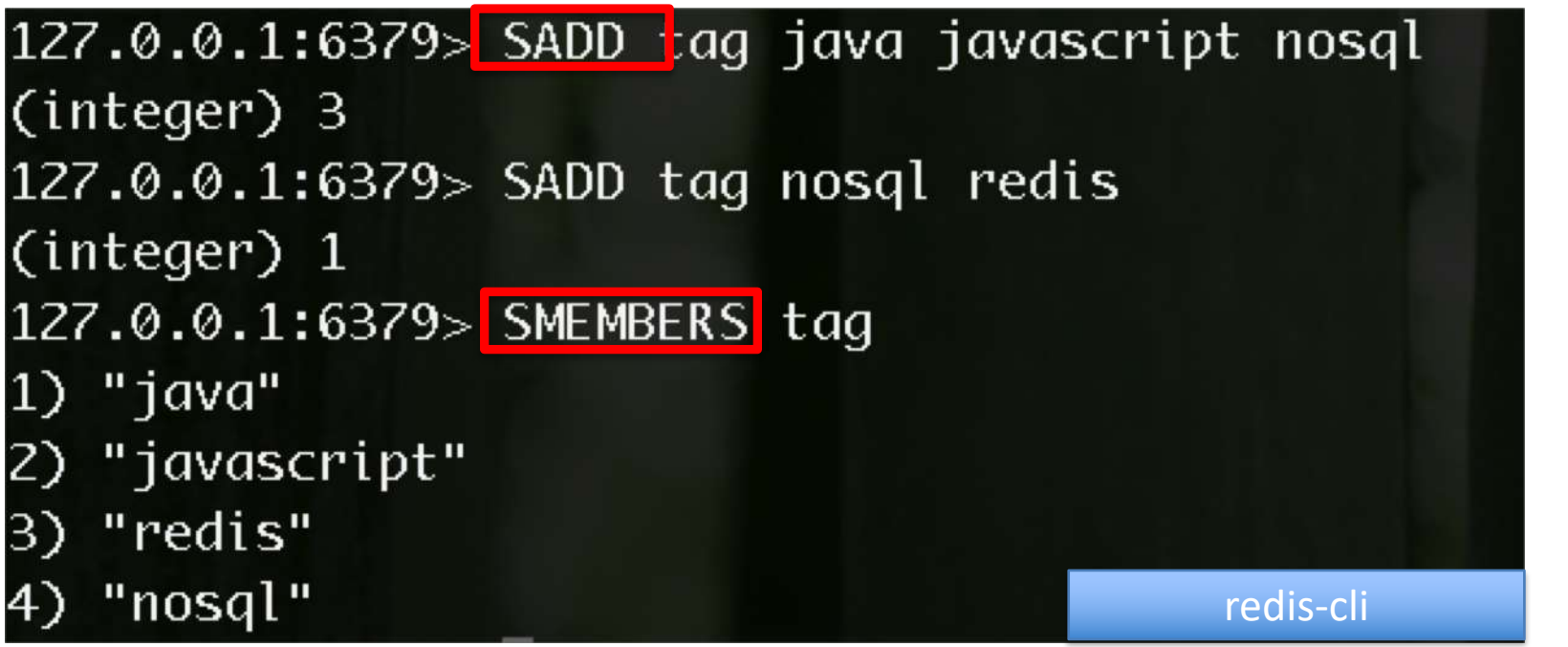
```
>>> import redis
>>> r = redis.StrictRedis()
>>> r.rpoplpush('score','score')
'30'
>>> r.lrange('score',0,-1)
['30', '20', '10', '60', '50', '40']
>>>
```

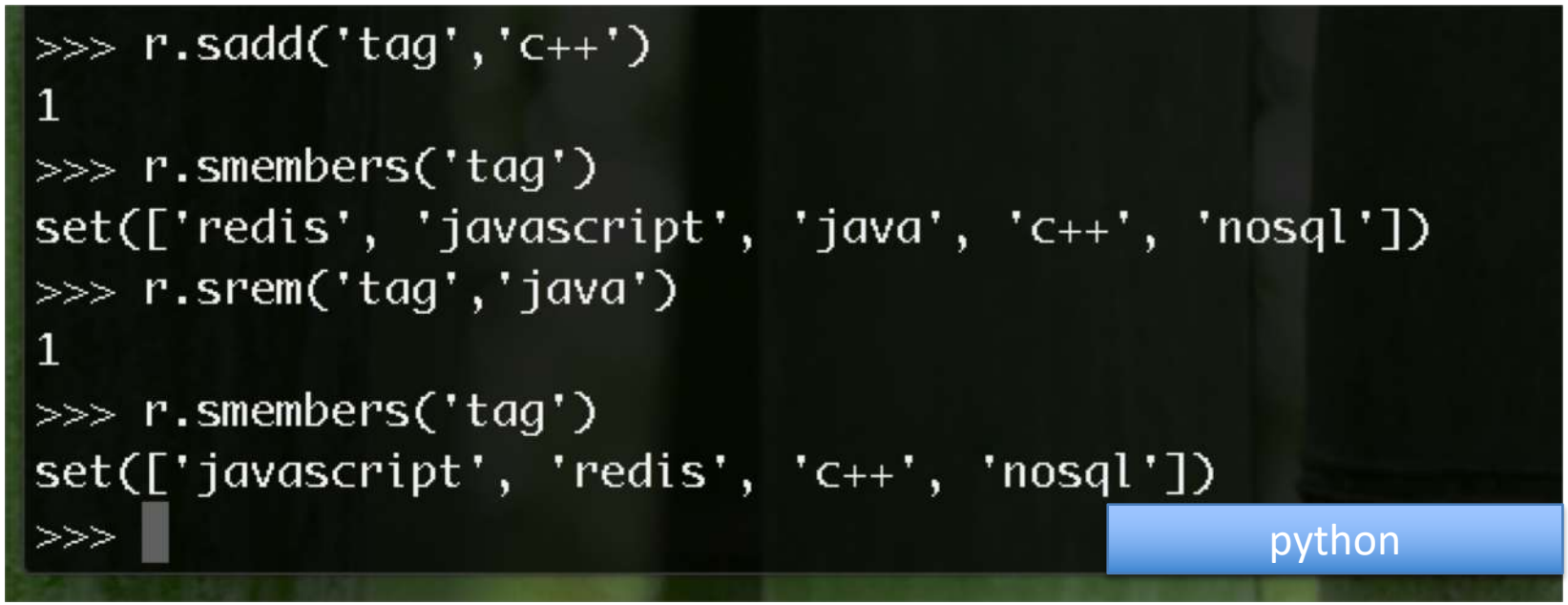
Data Type -- Set

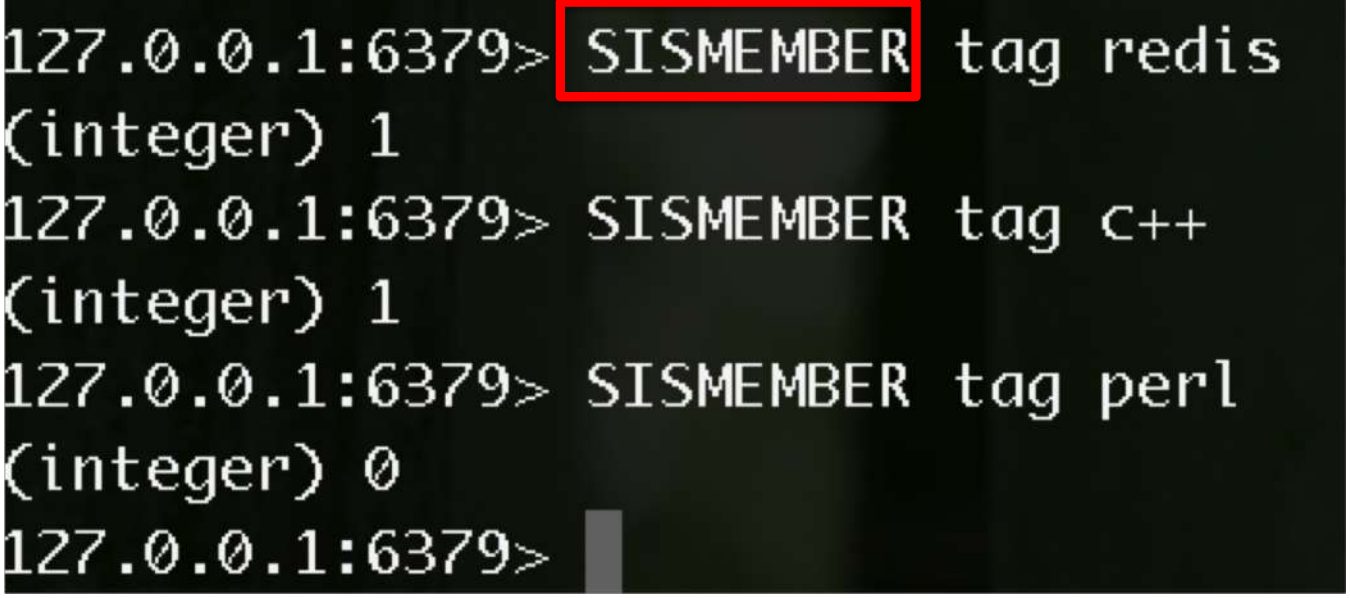
- Redis Sets are an unordered collection of String. It is possible to add, remove, and test for existe -nce of members in O(1)
- A very interesting thing about Redis Sets is that they support a number of server side commands to compute sets starting from existing sets, so you can do **unions**, **intersections**, **differences** of sets in very short time.

Set Commands

- SADD,SREM
- SMSMBERS
- SISMEMBER
- SDIFF, SINTER, SUNION


127.0.0.1:6379> SADD tag java javascript nosql
(integer) 3
127.0.0.1:6379> SADD tag nosql redis
(integer) 1
127.0.0.1:6379> SMEMBERS tag
1) "java"
2) "javascript"
3) "redis"
4) "nosql"
redis-cli


>>> r.sadd('tag','c++')
1
>>> r.smembers('tag')
set(['redis', 'javascript', 'java', 'c++', 'nosql'])
>>> r.srem('tag','java')
1
>>> r.smembers('tag')
set(['javascript', 'redis', 'c++', 'nosql'])
>>>
python



SISMEMBER key member

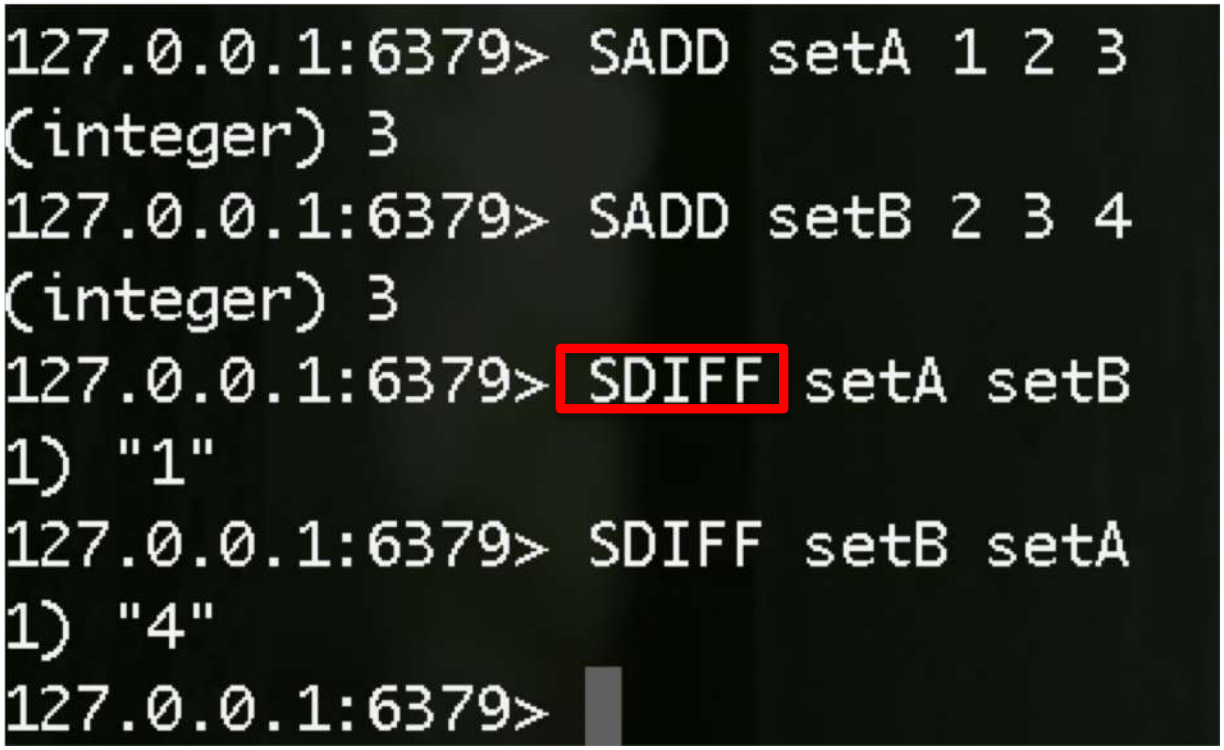
> **Available since 1.0.0.**
>
> **Time complexity:** O(1)

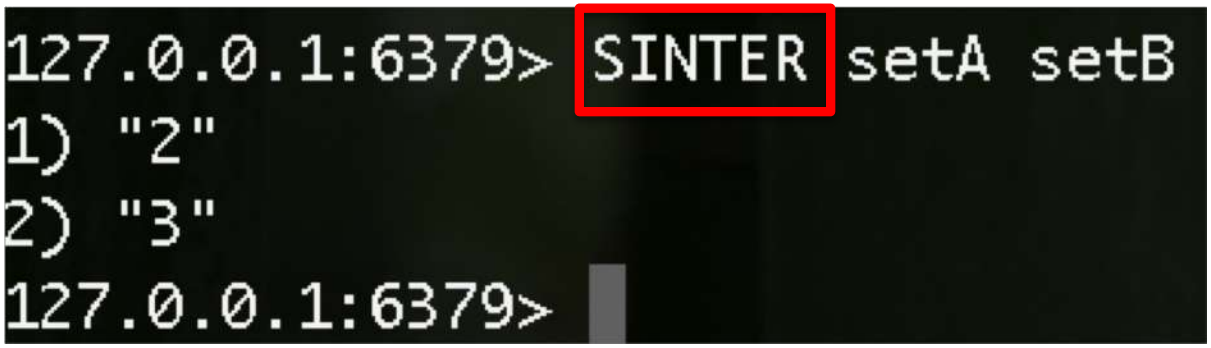
Returns if `member` is a member of the set stored at key.

Return value

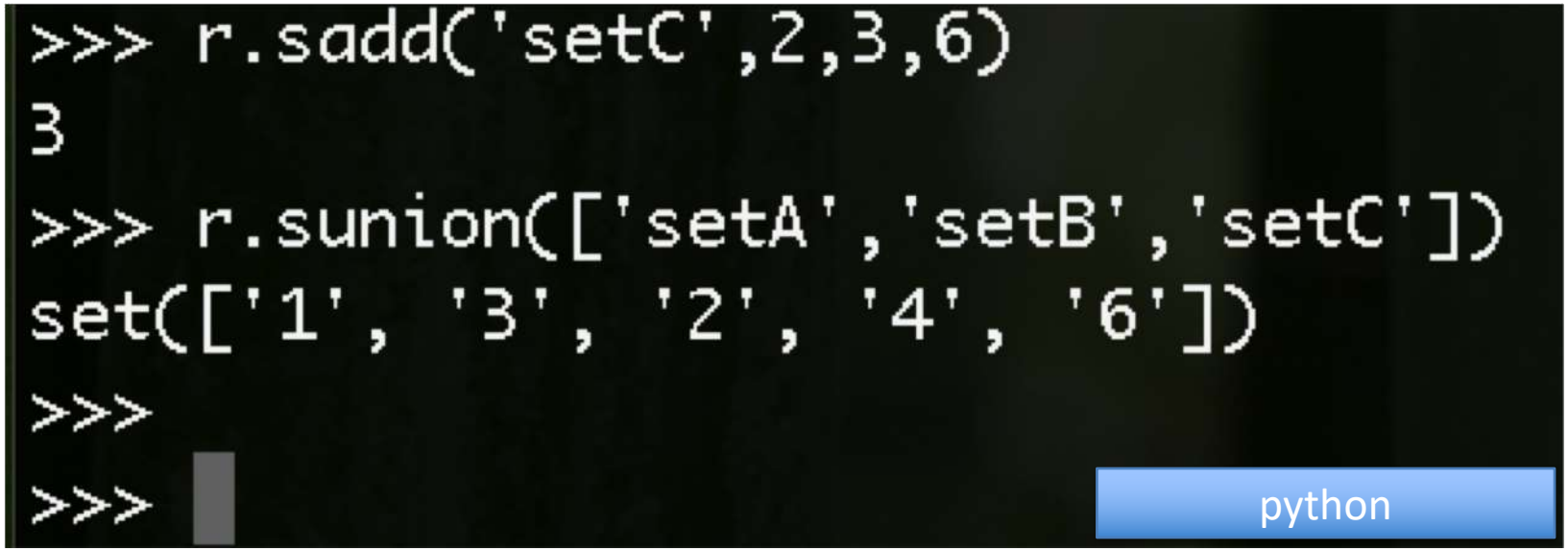
Integer reply, specifically:

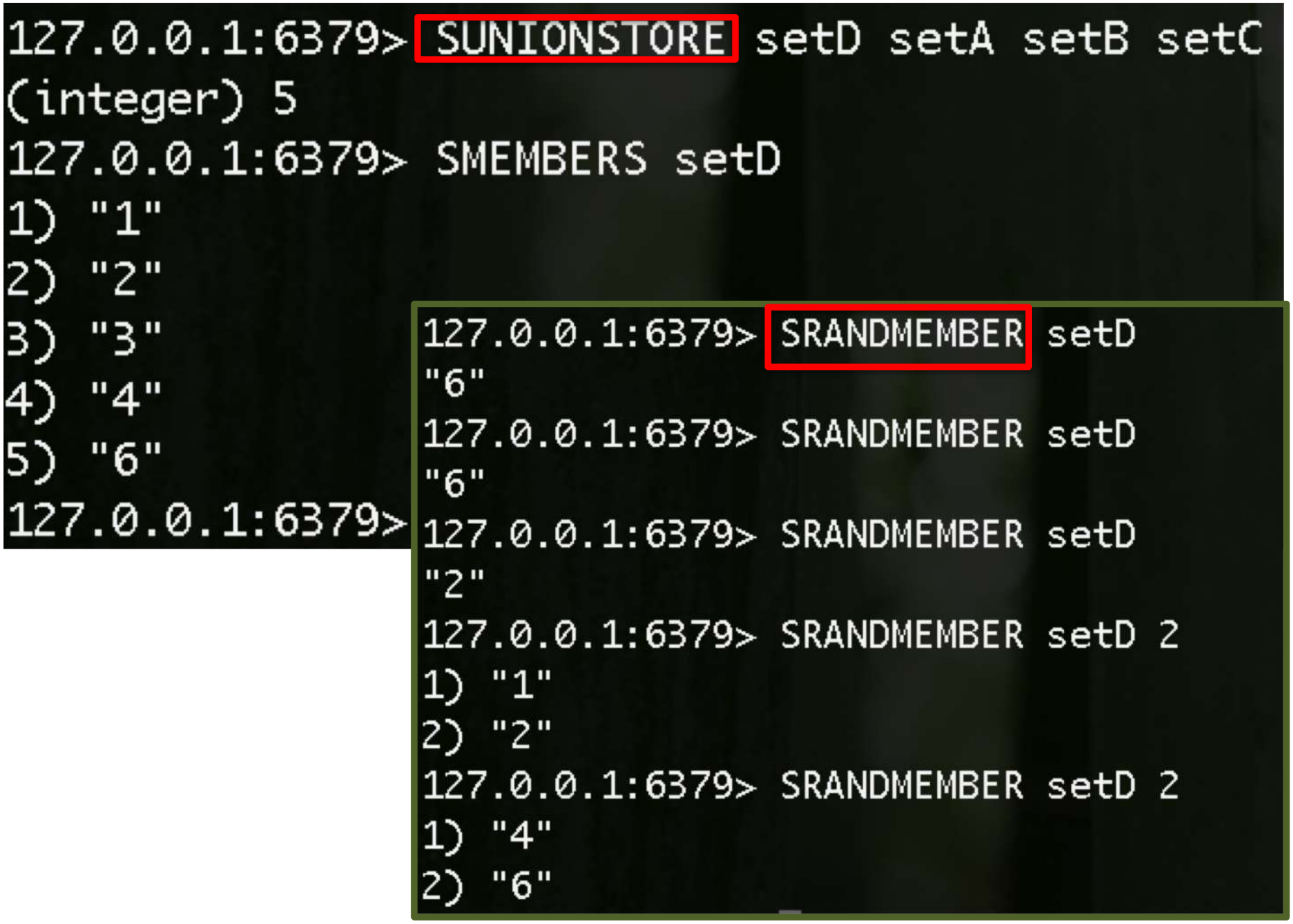
- `1` if the element is a member of the set.
- `0` if the element is not a member of the set, or if key does not exist.

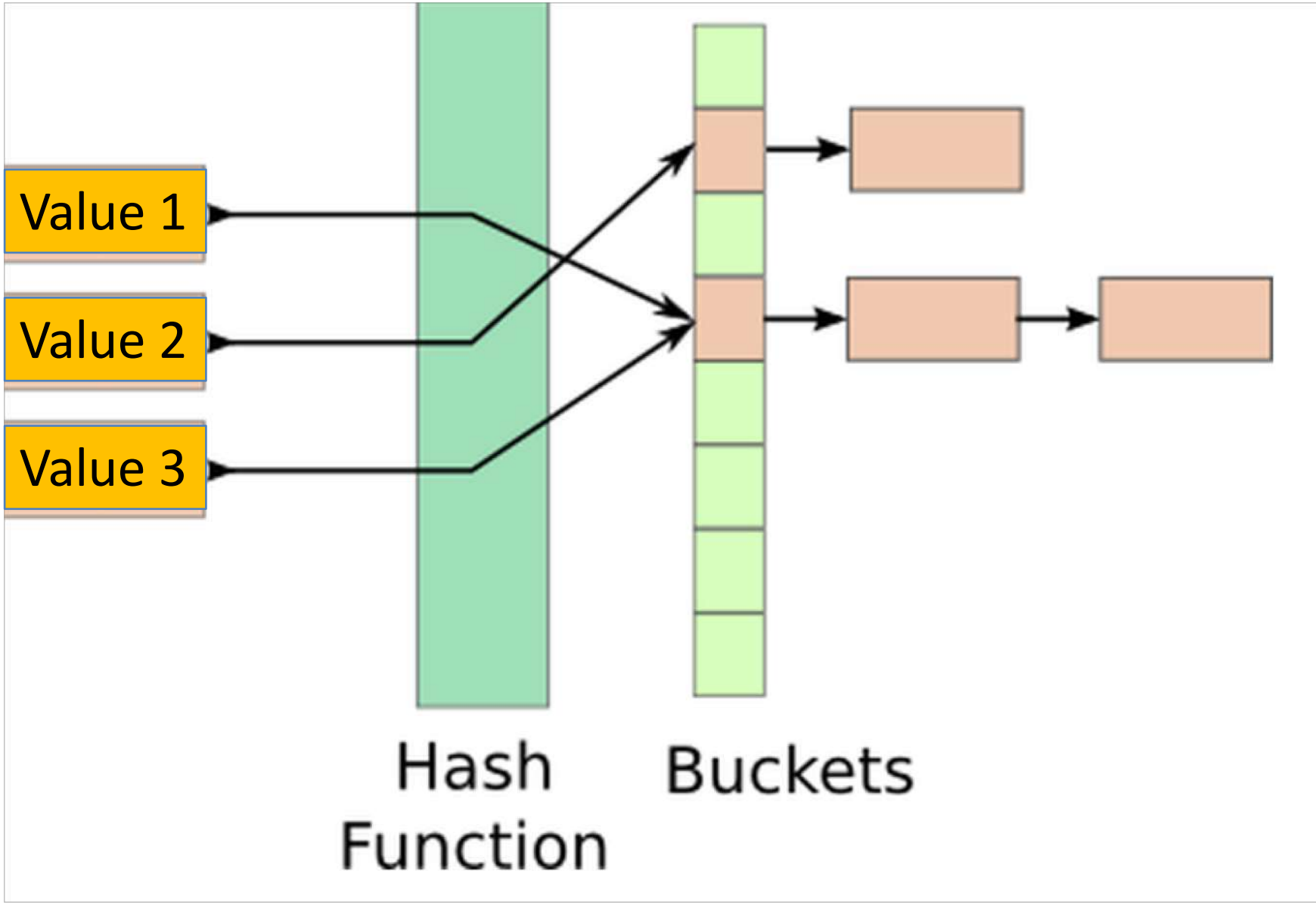

127.0.0.1:6379> SADD setA 1 2 3
(integer) 3
127.0.0.1:6379> SADD setB 2 3 4
(integer) 3
127.0.0.1:6379> SDIFF setA setB
1) "1"
127.0.0.1:6379> SDIFF setB setA
1) "4"
127.0.0.1:6379>


127.0.0.1:6379> SINTER setA setB
1) "2"
2) "3"
127.0.0.1:6379>


127.0.0.1:6379> SUNION setA setB
1) "1"
2) "2"
3) "3"
4) "4"
127.0.0.1:6379>
redis-cli


>>> r.sadd('setC',2,3,6)
3
>>> r.sunion(['setA','setB','setC'])
set(['1', '3', '2', '4', '6'])
>>>
>>>
python


127.0.0.1:6379> SUNIONSTORE setD setA setB setC
(integer) 5
127.0.0.1:6379> SMEMBERS setD
1) "1"
2) "2"
3) "3"
4) "4"
5) "6"
127.0.0.1:6379>
127.0.0.1:6379> SRANDMEMBER setD
"6"
127.0.0.1:6379> SRANDMEMBER setD
"6"
127.0.0.1:6379> SRANDMEMBER setD
"2"
127.0.0.1:6379> SRANDMEMBER setD 2
1) "1"
2) "2"
127.0.0.1:6379> SRANDMEMBER setD 2
1) "4"
2) "6"

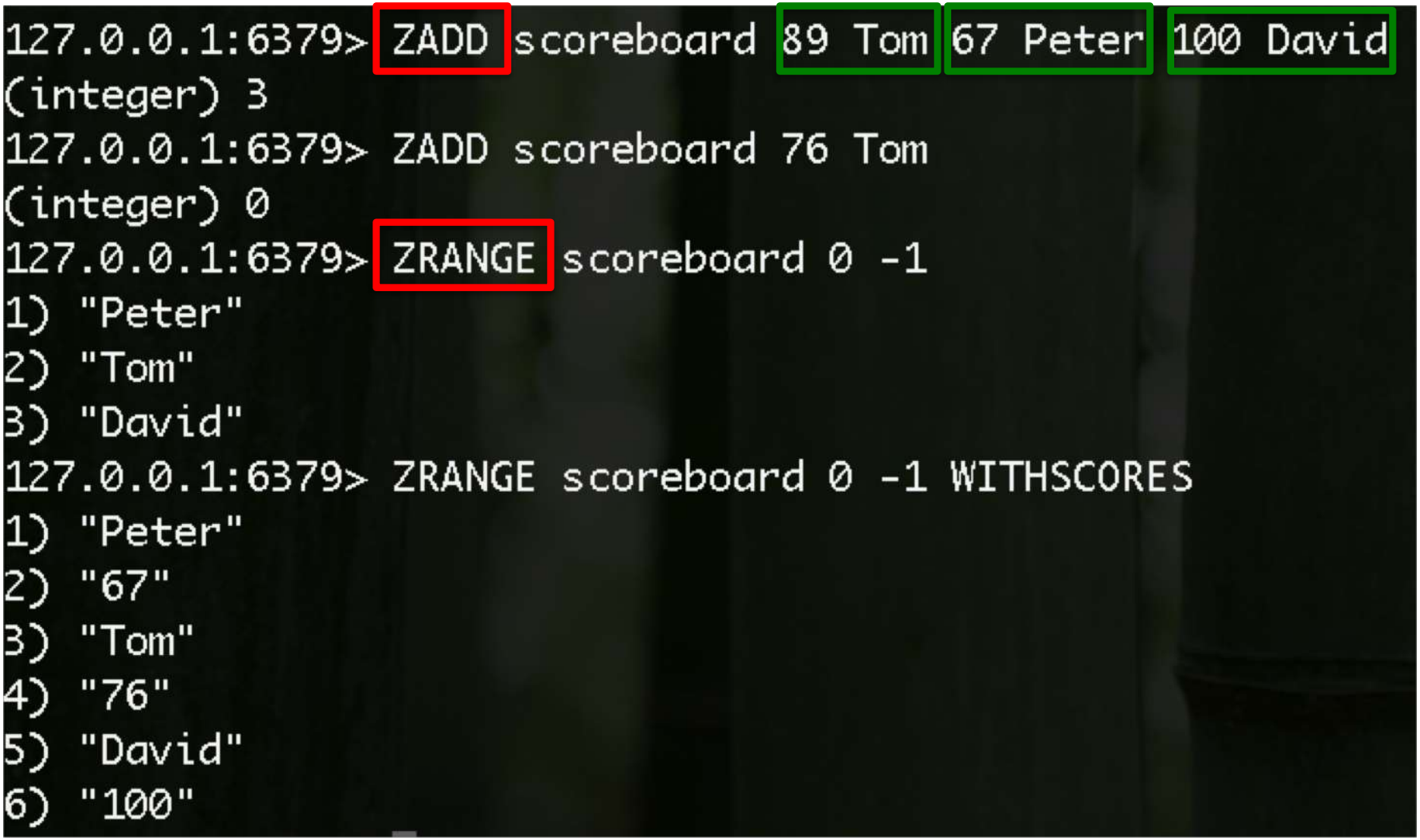

Value 1
Value 2
Value 3
Hash Function
Buckets

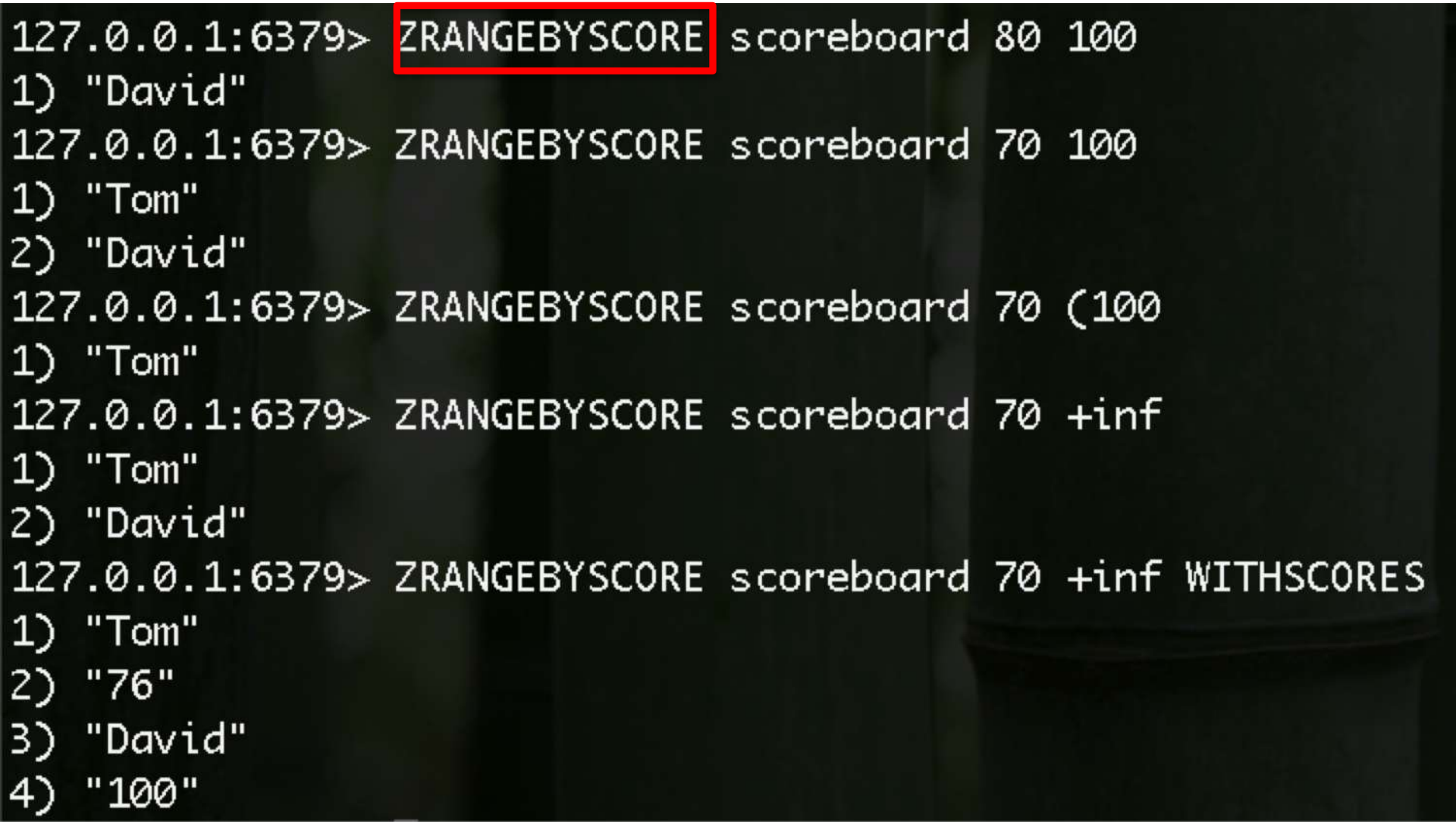
Data Type – Sorted Set

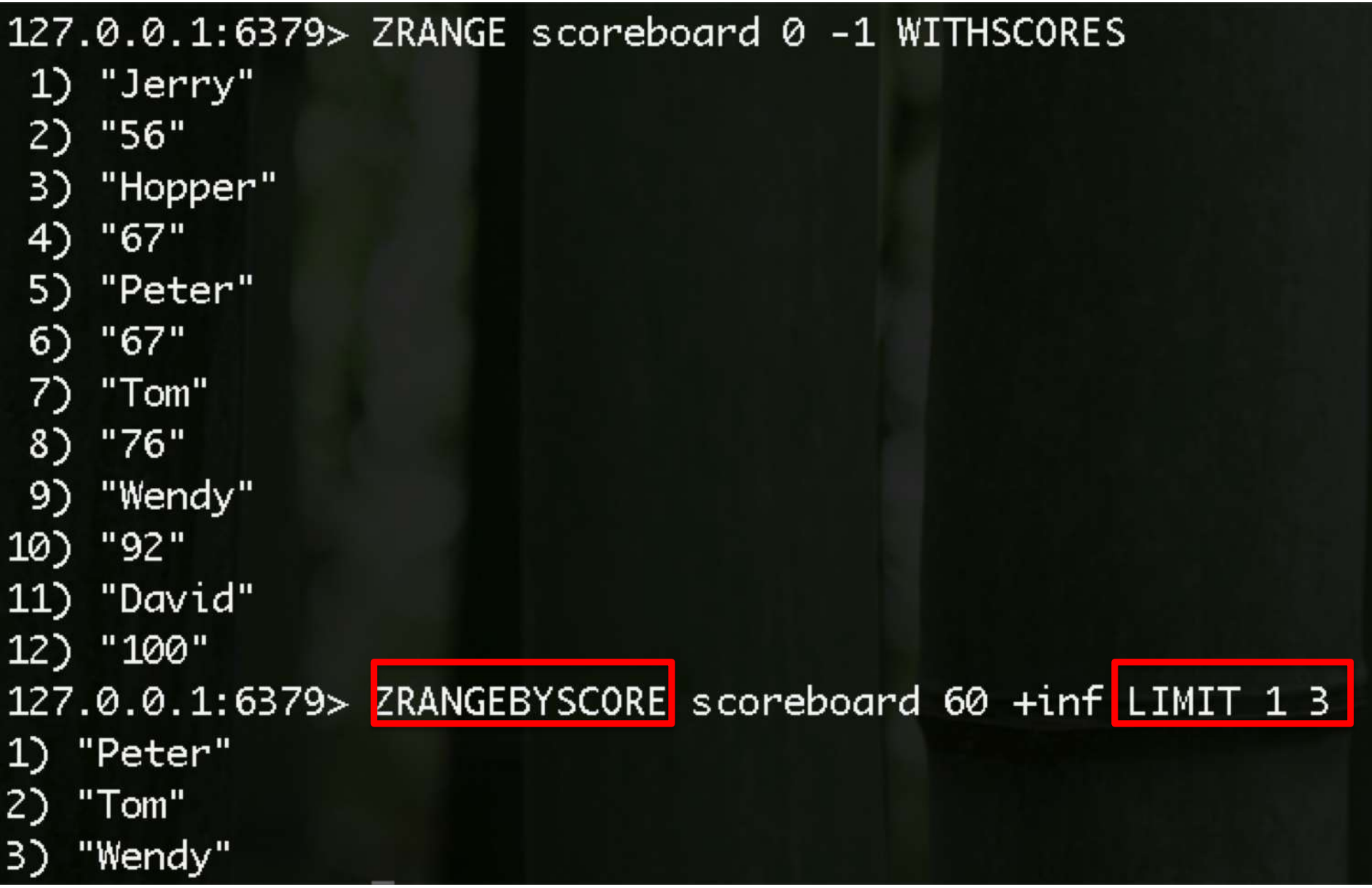
- Redis **Sorted Sets** are, similarly to Redis Sets, The difference is that **every member of a Sorted Set is associated with score**, that is used in order to take the sorted set ordered.
- use Sorted Sets **as a smart list of non-repeating elements** where you can **quickly access** everything you need.

Sorted Set Commands

- ZADD,
- ZRANGE
- ZRANGEBYSCORE
- ZINCRBY
- ZCARD,ZCOUNT
- ZREM, ZREMRANGEBYSCORE
- ZRANK,ZREVRANK


127.0.0.1:6379> ZADD scoreboard 89 Tom 67 Peter 100 David
(integer) 3
127.0.0.1:6379> ZADD scoreboard 76 Tom
(integer) 0
127.0.0.1:6379> ZRANGE scoreboard 0 -1
1) "Peter"
2) "Tom"
3) "David"
127.0.0.1:6379> ZRANGE scoreboard 0 -1 WITHSCORES
1) "Peter"
2) "67"
3) "Tom"
4) "76"
5) "David"
6) "100"


127.0.0.1:6379> ZRANGEBYSCORE scoreboard 80 100
1) "David"
127.0.0.1:6379> ZRANGEBYSCORE scoreboard 70 100
1) "Tom"
2) "David"
127.0.0.1:6379> ZRANGEBYSCORE scoreboard 70 (100
1) "Tom"
127.0.0.1:6379> ZRANGEBYSCORE scoreboard 70 +inf
1) "Tom"
2) "David"
127.0.0.1:6379> ZRANGEBYSCORE scoreboard 70 +inf WITHSCORES
1) "Tom"
2) "76"
3) "David"
4) "100"


127.0.0.1:6379> ZRANGE scoreboard 0 -1 WITHSCORES
1) "Jerry"
2) "56"
3) "Hopper"
4) "67"
5) "Peter"
6) "67"
7) "Tom"
8) "76"
9) "Wendy"
10) "92"
11) "David"
12) "100"
127.0.0.1:6379> ZRANGEBYSCORE scoreboard 60 +inf LIMIT 1 3
1) "Peter"
2) "Tom"
3) "Wendy"

python

zrangebyscore(*name, min, max, start=None, num=None, withscores=False, score_cast_func=<type 'float'>*)

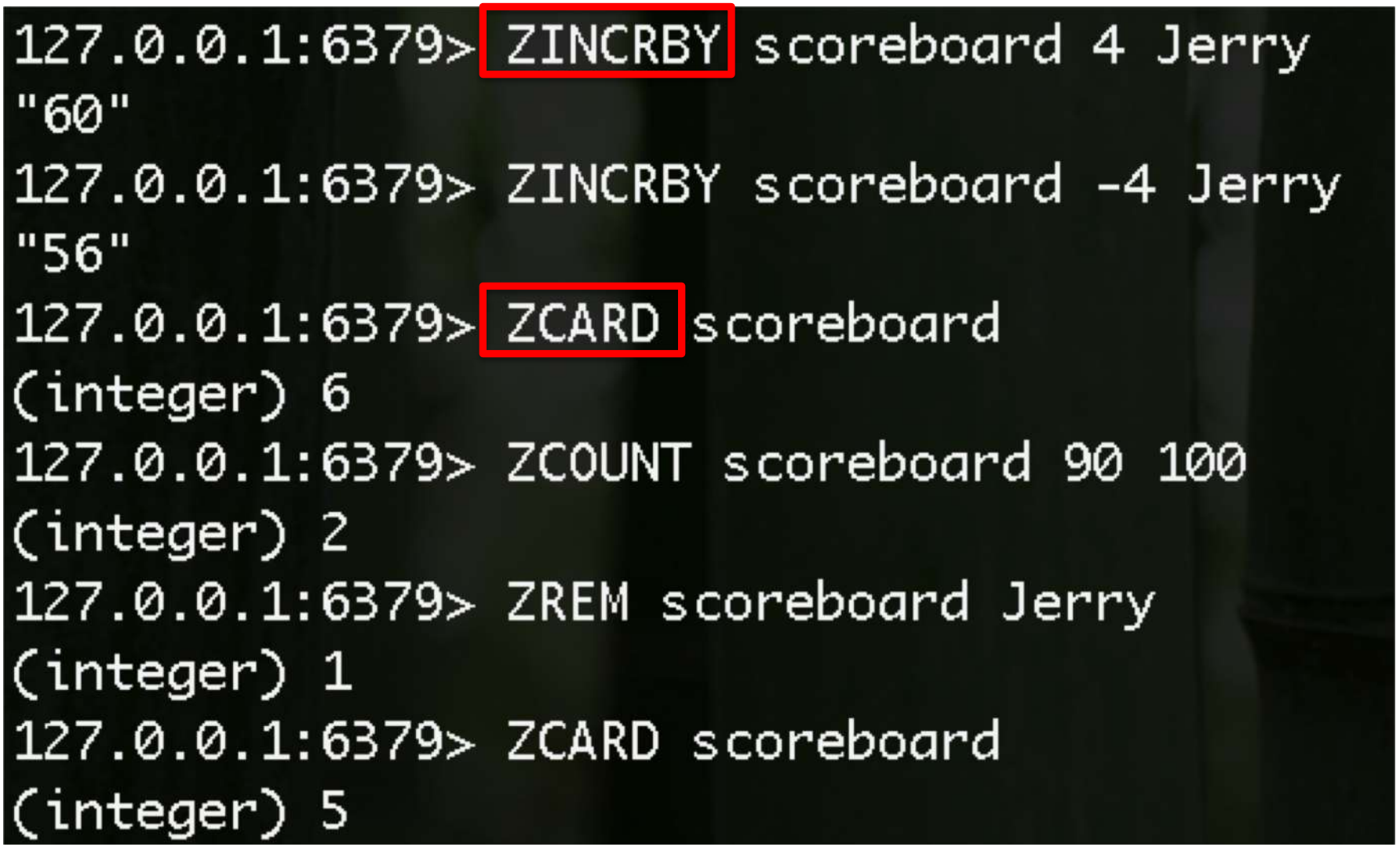
Return a range of values from the sorted set `name` with scores between `min` and `max`.

If `start` and `num` are specified, then return a slice of the range.

`withscores` indicates to return the scores along with the values. The return type is a list of (value, score) pairs

score_cast_func`a callable used to cast the score return value

```
>>> r.zrangebyscore('scoreboard',min=60,max=100,start=1,num=3)
['Peter', 'Tom', 'Wendy']
```

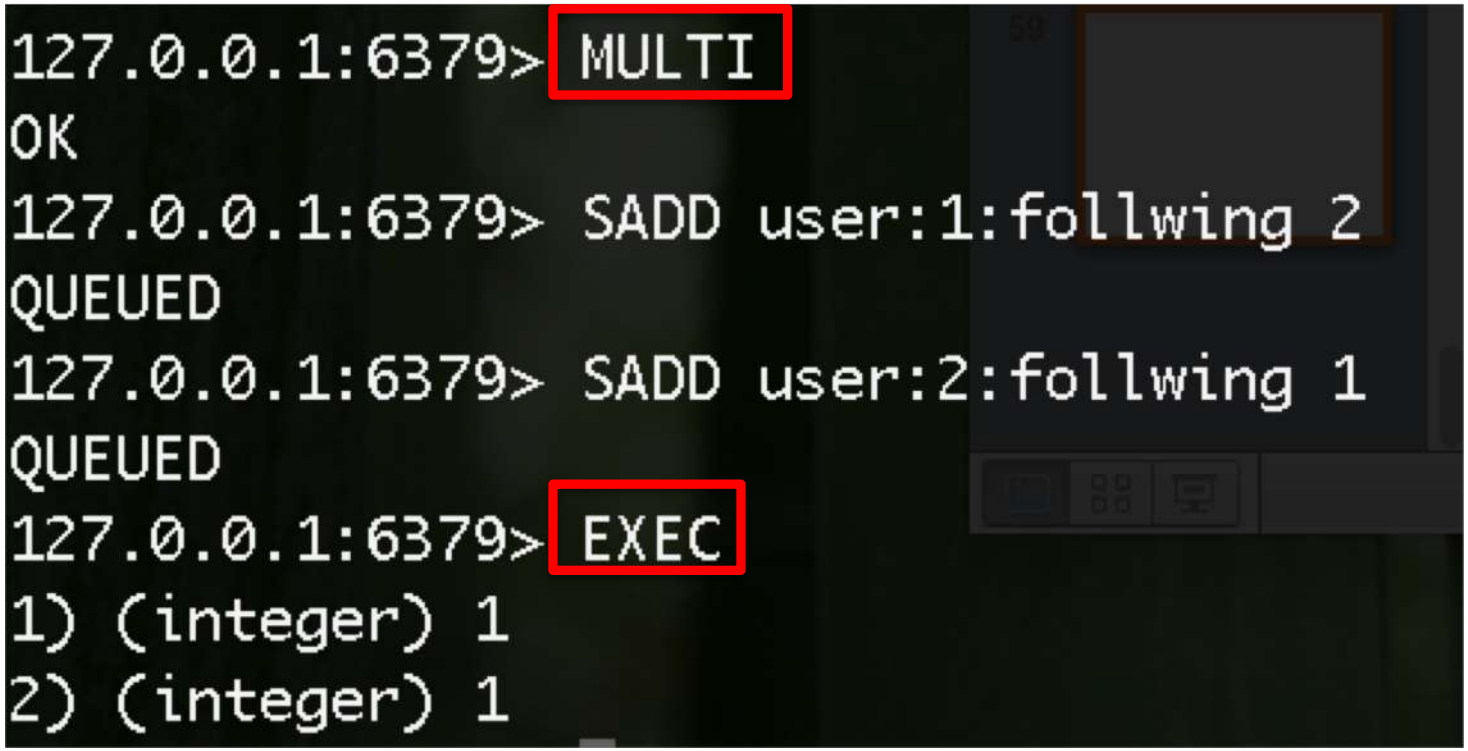

127.0.0.1:6379> ZINCRBY scoreboard 4 Jerry
"60"
127.0.0.1:6379> ZINCRBY scoreboard -4 Jerry
"56"
127.0.0.1:6379> ZCARD scoreboard
(integer) 6
127.0.0.1:6379> ZCOUNT scoreboard 90 100
(integer) 2
127.0.0.1:6379> ZREM scoreboard Jerry
(integer) 1
127.0.0.1:6379> ZCARD scoreboard
(integer) 5

```
127.0.0.1:6379> ZREMRANGEBYSCORE scoreboard 60 69
(integer) 2
127.0.0.1:6379> ZRANGE scoreboard 0 -1 WITHSCORES
1) "Tom"
2) "76"
3) "Wendy"
4) "92"
5) "David"
6) "100"
```

```
127.0.0.1:6379> ZREVRANK scoreboard David
(integer) 0
127.0.0.1:6379> ZREVRANK scoreboard Wendy
(integer) 1
127.0.0.1:6379> ZREVRANK scoreboard Tom
(integer) 2
```

Transaction

- All the commands in a transaction are serialized and executed sequentially.
- It can never happen that a request issued by another client is served **in the middle** of the execution of a transaction.
- Redis transaction is **atomic**.
- Guarantees that the commands are executed as **a single isolated operation**.


127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> SADD user:1:follwing 2
QUEUED
127.0.0.1:6379> SADD user:2:follwing 1
QUEUED
127.0.0.1:6379> EXEC
1) (integer) 1
2) (integer) 1

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> SET testTransKey 1
QUEUED
127.0.0.1:6379> SADD testTransKey 2
QUEUED
127.0.0.1:6379> SET testTransKey 3
QUEUED
127.0.0.1:6379> EXEC
1) OK
2) (error) WRONGTYPE Operation against a key holding the wrong kind of value
3) OK
127.0.0.1:6379> GET testTransKey
"3"
127.0.0.1:6379>
```


127.0.0.1:6379> WATCH testTransKey
OK
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> SET testTransKey 6
QUEUED
127.0.0.1:6379> EXEC
(nil)
127.0.0.1:6379> GET testTransKey
"100"


127.0.0.1:6379> SET testTransKey 100
OK

python

```
>>> r = redis.Redis(...)
>>> r.set('bing', 'baz')
>>> # Use the pipeline() method to create a pipeline instance
>>> pipe = r.pipeline()
>>> # The following SET commands are buffered
>>> pipe.set('foo', 'bar')
>>> pipe.get('bing')
>>> # the EXECUTE call sends all buffered commands to the server, returning
>>> # a list of responses, one for each command.
>>> pipe.execute()
[True, 'baz']
```

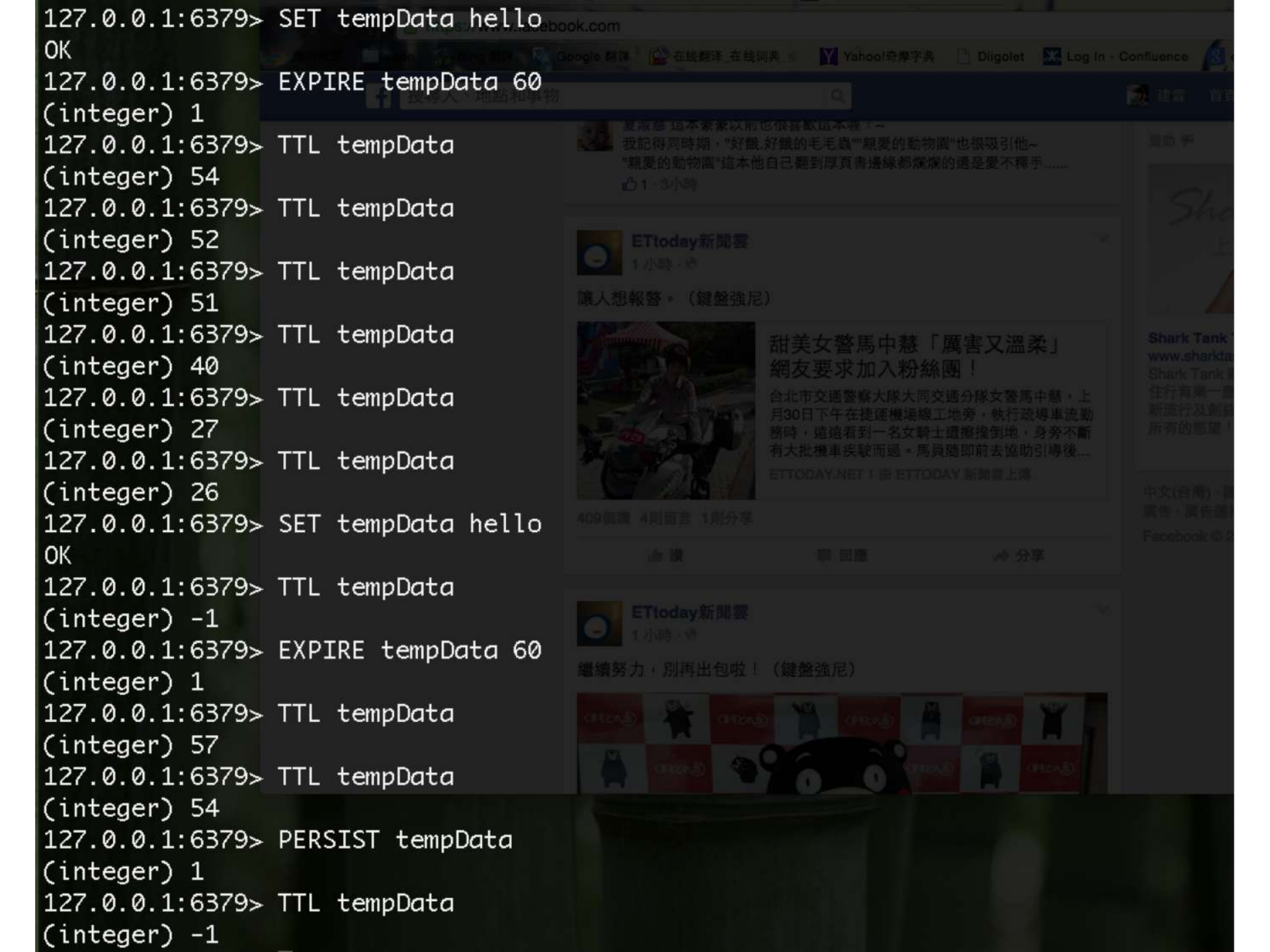
```
>>> pipe.set('foo', 'bar').sadd('faz', 'baz').incr('auto_number').execute()
[True, True, 6]
```

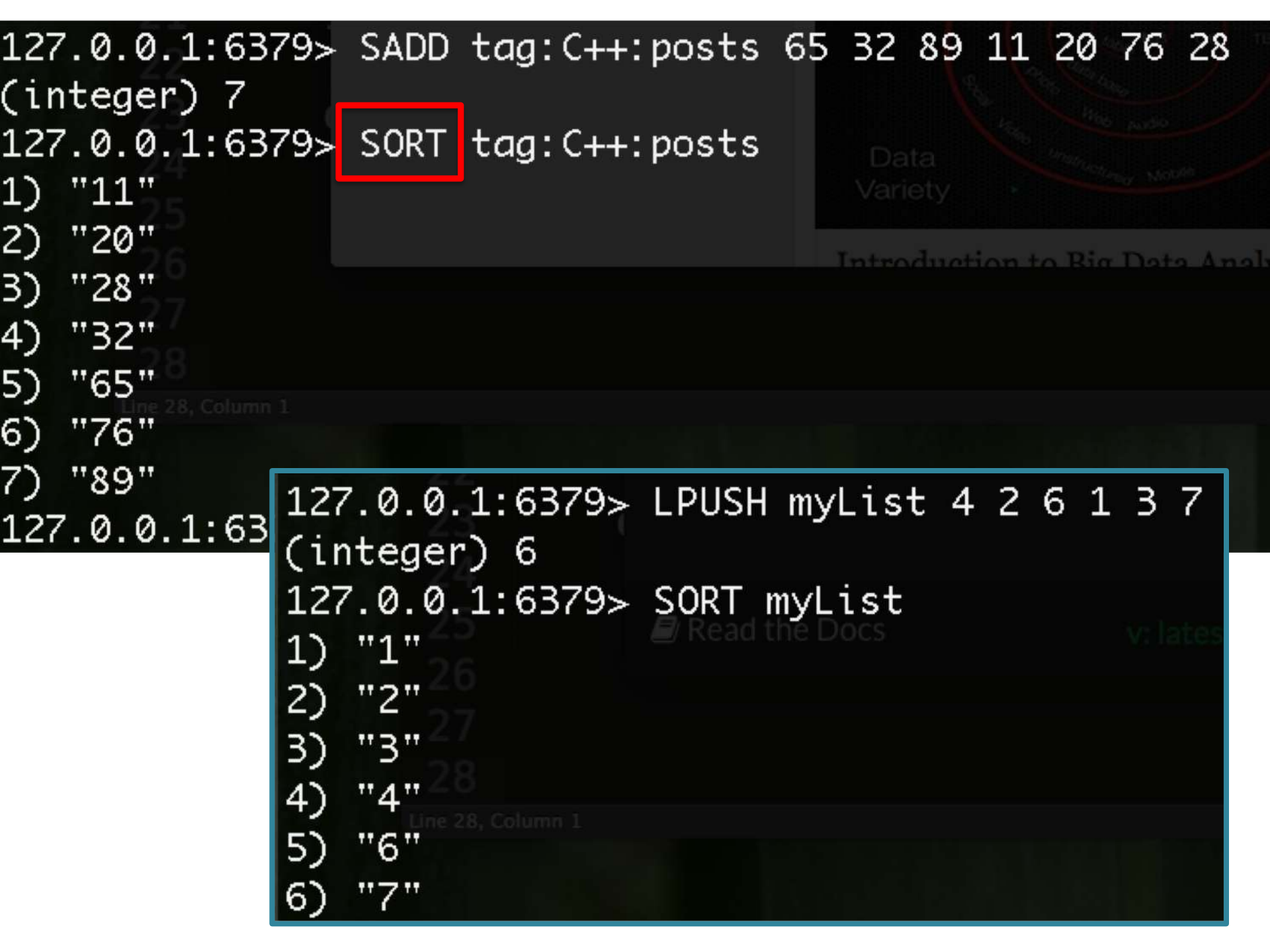
```
>>> with r.pipeline() as pipe:
...     while 1:
...         try:
...             # put a WATCH on the key that holds our sequence value
...             pipe.watch('OUR-SEQUENCE-KEY')
...             # after WATCHing, the pipeline is put into immediate execution
...             # mode until we tell it to start buffering commands again.
...             # this allows us to get the current value of our sequence
...             current_value = pipe.get('OUR-SEQUENCE-KEY')
...             next_value = int(current_value) + 1
...             # now we can put the pipeline back into buffered mode with MULTI
...             pipe.multi()
...             pipe.set('OUR-SEQUENCE-KEY', next_value)
...             # and finally, execute the pipeline (the set command)
...             pipe.execute()
...             # if a WatchError wasn't raised during execution, everything
...             # we just did happened atomically.
...             break
...         except WatchError:
...             # another client must have changed 'OUR-SEQUENCE-KEY' between
...             # the time we started WATCHing it and the pipeline's execution.
...             # our best bet is to just retry.
...             continue
```

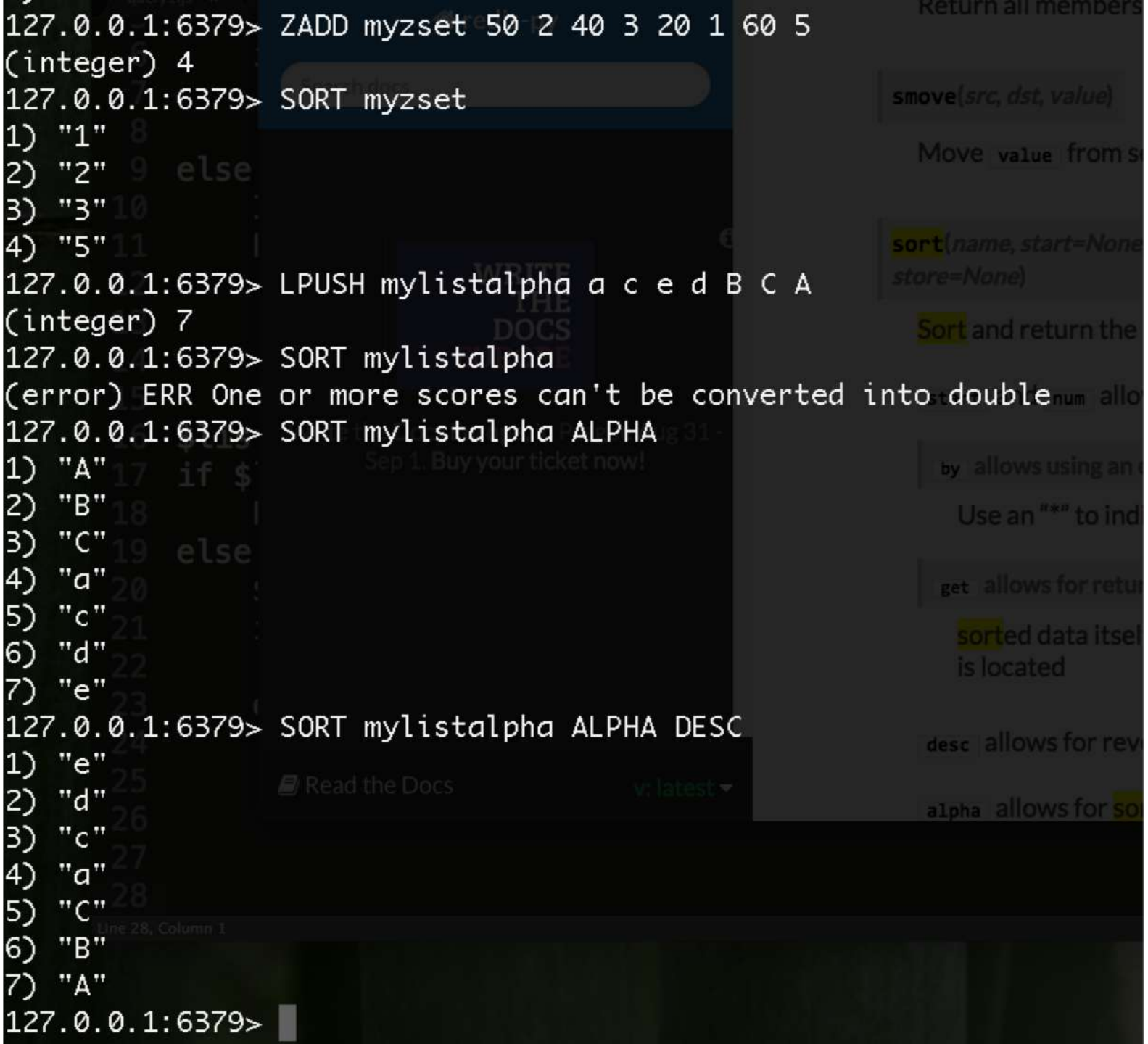
```
127.0.0.1:6379> SET tempData hello
OK
127.0.0.1:6379> EXPIRE tempData 10
(integer) 1
127.0.0.1:6379> GET tempData
"hello"
127.0.0.1:6379> GET tempData
"hello"
127.0.0.1:6379> GET tempData
(nil)
127.0.0.1:6379>
```

```
>>> import redis
>>> r = redis.StrictRedis(host='localhost', port=6379, db=0)
>>> r.setex("tempData",10,"hello")
True
>>> r.get("tempData")
'hello'
>>> r.get("tempData")
>>> r.get("tempData")
```

python


127.0.0.1:6379> SET tempData hello
OK
127.0.0.1:6379> EXPIRE tempData 60
(integer) 1
127.0.0.1:6379> TTL tempData
(integer) 54
127.0.0.1:6379> TTL tempData
(integer) 52
127.0.0.1:6379> TTL tempData
(integer) 51
127.0.0.1:6379> TTL tempData
(integer) 40
127.0.0.1:6379> TTL tempData
(integer) 27
127.0.0.1:6379> TTL tempData
(integer) 26
127.0.0.1:6379> SET tempData hello
OK
127.0.0.1:6379> TTL tempData
(integer) -1
127.0.0.1:6379> EXPIRE tempData 60
(integer) 1
127.0.0.1:6379> TTL tempData
(integer) 57
127.0.0.1:6379> TTL tempData
(integer) 54
127.0.0.1:6379> PERSIST tempData
(integer) 1
127.0.0.1:6379> TTL tempData
(integer) -1
ETtoday新聞雲
讓人想報警。（鍵盤強尼）
甜美女警馬中慧「厲害又溫柔」網友要求加入粉絲團！
ETtoday新聞雲
繼續努力，別再出包啦！（鍵盤強尼）

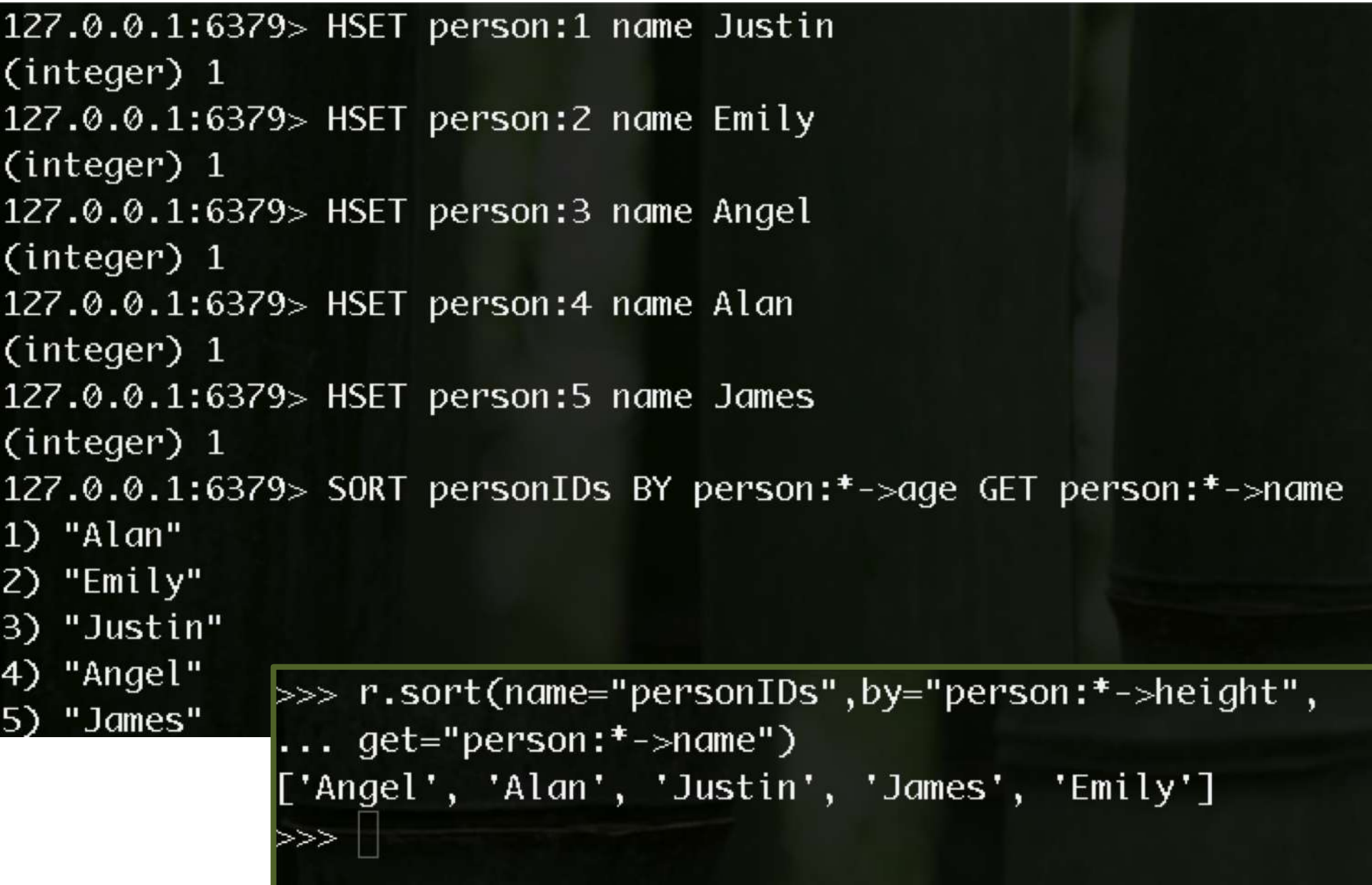

127.0.0.1:6379> SADD tag:C++:posts 65 32 89 11 20 76 28
(integer) 7
127.0.0.1:6379> SORT tag:C++:posts
1) "11"
2) "20"
3) "28"
4) "32"
5) "65"
6) "76"
7) "89"
127.0.0.1:63
127.0.0.1:6379> LPUSH myList 4 2 6 1 3 7
(integer) 6
127.0.0.1:6379> SORT myList
1) "1"
2) "2"
3) "3"
4) "4"
5) "6"
6) "7"

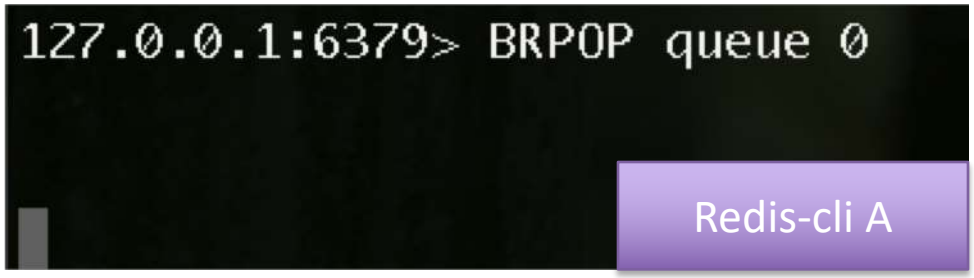

127.0.0.1:6379> ZADD myzset 50 2 40 3 20 1 60 5
(integer) 4
127.0.0.1:6379> SORT myzset
1) "1"
2) "2"
3) "3"
4) "5"
127.0.0.1:6379> LPUSH mylistalpha a c e d B C A
(integer) 7
127.0.0.1:6379> SORT mylistalpha
(error) ERR One or more scores can't be converted into double
127.0.0.1:6379> SORT mylistalpha ALPHA
1) "A"
2) "B"
3) "C"
4) "a"
5) "c"
6) "d"
7) "e"
127.0.0.1:6379> SORT mylistalpha ALPHA DESC
1) "e"
2) "d"
3) "c"
4) "a"
5) "C"
6) "B"
7) "A"
127.0.0.1:6379>

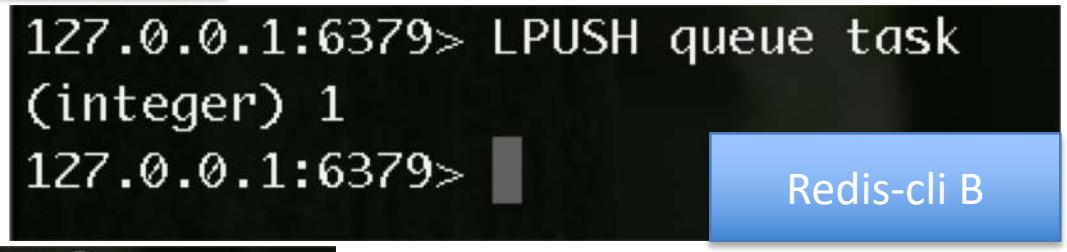
```
127.0.0.1:6379> LPUSH productIDs 2 1 3 4 5
(integer) 5
127.0.0.1:6379> SET productPrice:1 50
OK
127.0.0.1:6379> SET productPrice:2 60
OK
127.0.0.1:6379> SET productPrice:4 40
OK
127.0.0.1:6379> SET productPrice:3 30
OK
127.0.0.1:6379> SET productPrice:5 70
OK
127.0.0.1:6379> SORT productIDs BY productPrice:*
1) "3"
2) "4"
3) "1"
4) "2"
5) "5"
127.0.0.1:6379> SORT productIDs BY productPrice:* DESC
1) "5"
2) "2"
3) "1"
4) "4"
5) "3"
127.0.0.1:6379>
```

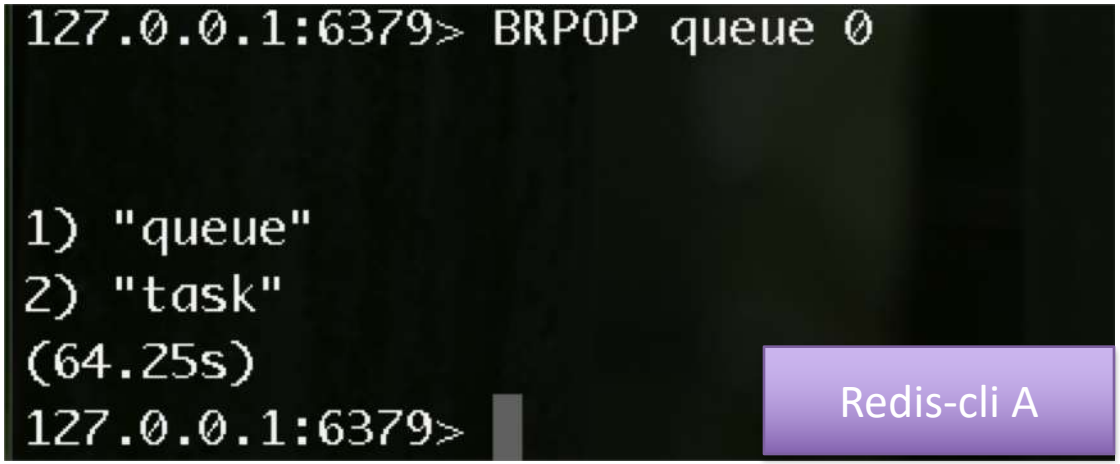
```
127.0.0.1:6379> LPUSH personIDs 1 2 3 4 5
(integer) 5
127.0.0.1:6379> HSET person:1 age 33
(integer) 1
127.0.0.1:6379> HSET person:1 height 175
(integer) 1
127.0.0.1:6379> HSET person:2 age 23
(integer) 1
127.0.0.1:6379> HSET person:2 height 182
(integer) 1
127.0.0.1:6379> HSET person:3 age 43
(integer) 1
127.0.0.1:6379> HSET person:3 height 165
(integer) 1
127.0.0.1:6379> HSET person:4 age 13
(integer) 1
127.0.0.1:6379> HSET person:4 height 170
(integer) 1
127.0.0.1:6379> HSET person:5 age 50
(integer) 1
127.0.0.1:6379> HSET person:5 height 180
(integer) 1
127.0.0.1:6379> SORT personIDs BY person:*->age
1) "4"
2) "2"
3) "1"
4) "3"
5) "5"
```

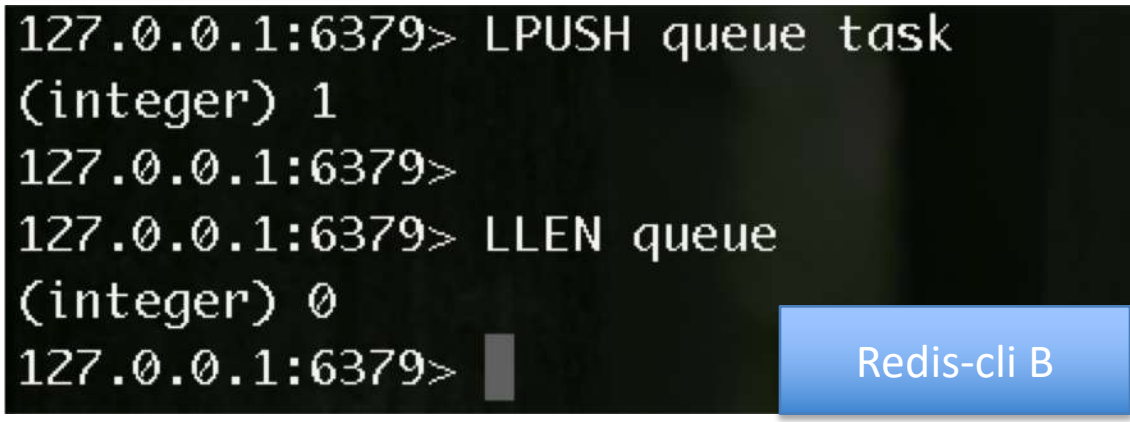
```
>>> r.sort(name="personIDs",by="person:*->height",desc=True)
['2', '5', '1', '4', '3']
>>>
```

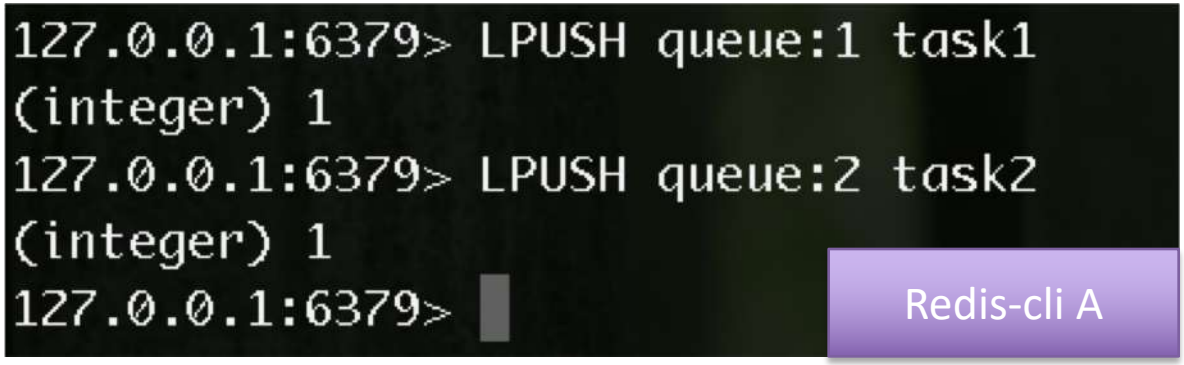

127.0.0.1:6379> HSET person:1 name Justin
(integer) 1
127.0.0.1:6379> HSET person:2 name Emily
(integer) 1
127.0.0.1:6379> HSET person:3 name Angel
(integer) 1
127.0.0.1:6379> HSET person:4 name Alan
(integer) 1
127.0.0.1:6379> HSET person:5 name James
(integer) 1
127.0.0.1:6379> SORT personIDs BY person:*->age GET person:*->name
1) "Alan"
2) "Emily"
3) "Justin"
4) "Angel"
5) "James"
>>> r.sort(name="personIDs",by="person:*->height",
... get="person:*->name")
['Angel', 'Alan', 'Justin', 'James', 'Emily']
>>>

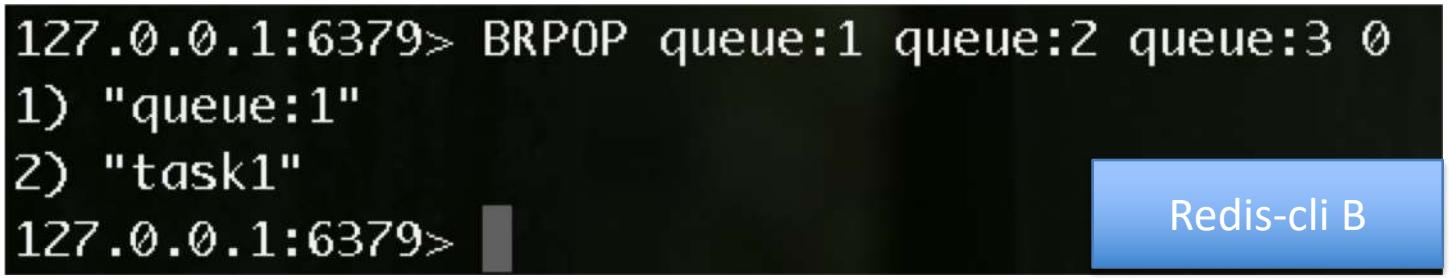

127.0.0.1:6379> BRPOP queue 0
Redis-cli A

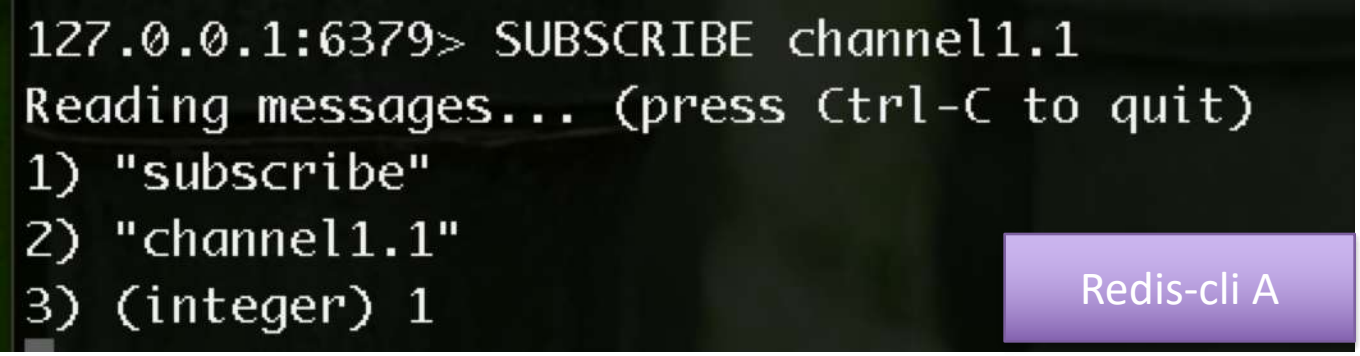

127.0.0.1:6379> LPUSH queue task
(integer) 1
127.0.0.1:6379>
Redis-cli B

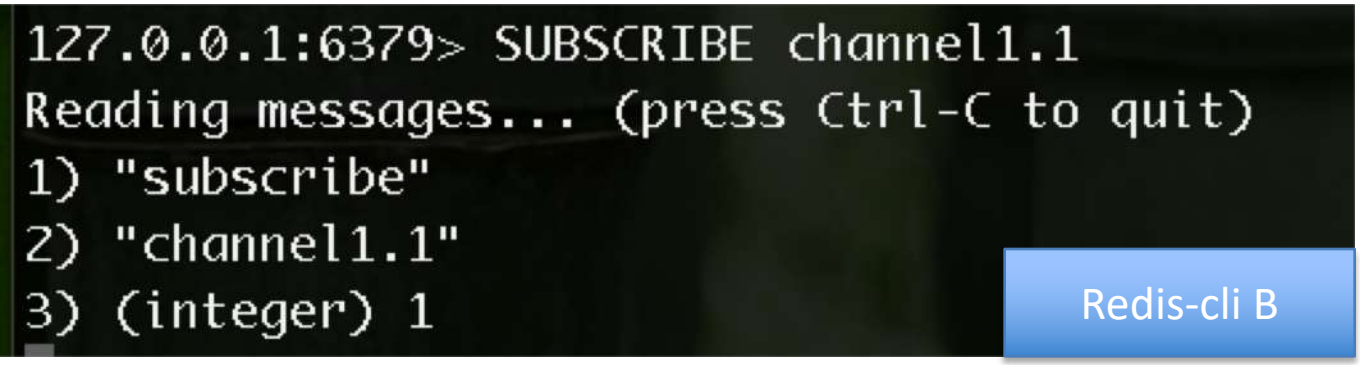

127.0.0.1:6379> BRPOP queue 0
1) "queue"
2) "task"
(64.25s)
127.0.0.1:6379>
Redis-cli A

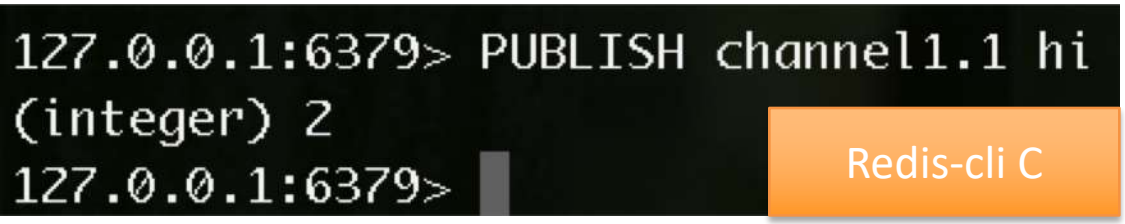

127.0.0.1:6379> LPUSH queue task
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379> LLEN queue
(integer) 0
127.0.0.1:6379>
Redis-cli B


127.0.0.1:6379> LPUSH queue:1 task1
(integer) 1
127.0.0.1:6379> LPUSH queue:2 task2
(integer) 1
127.0.0.1:6379>
Redis-cli A

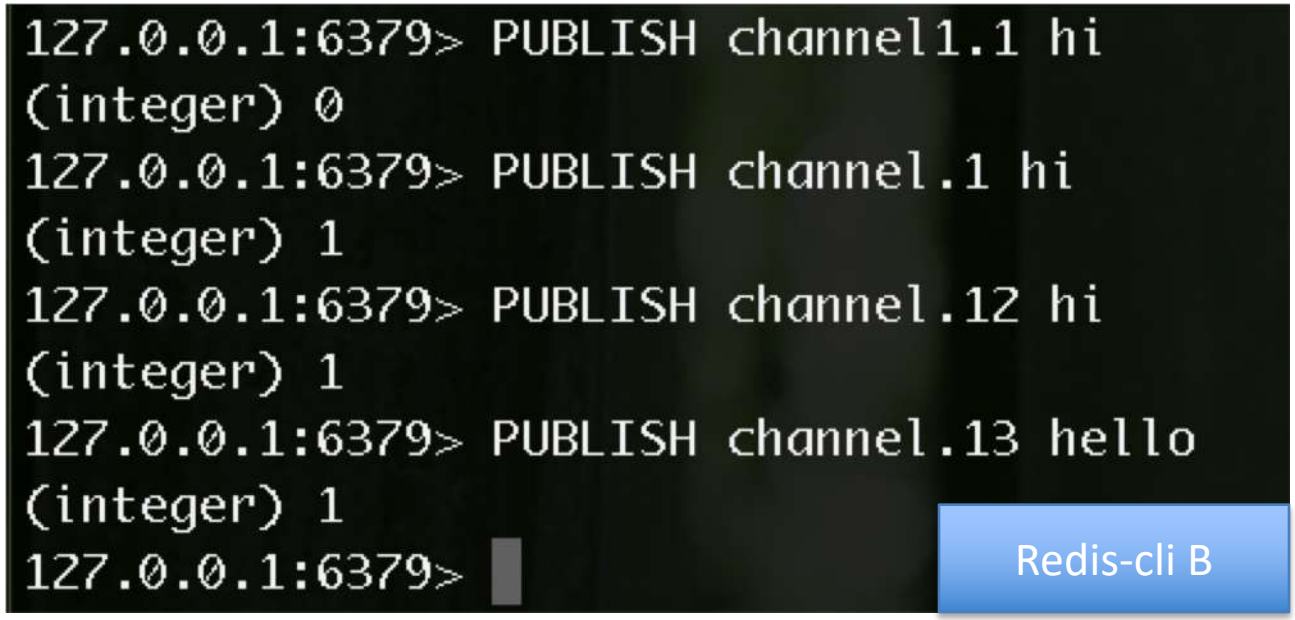

127.0.0.1:6379> BRPOP queue:1 queue:2 queue:3 0
1) "queue:1"
2) "task1"
127.0.0.1:6379>
Redis-cli B

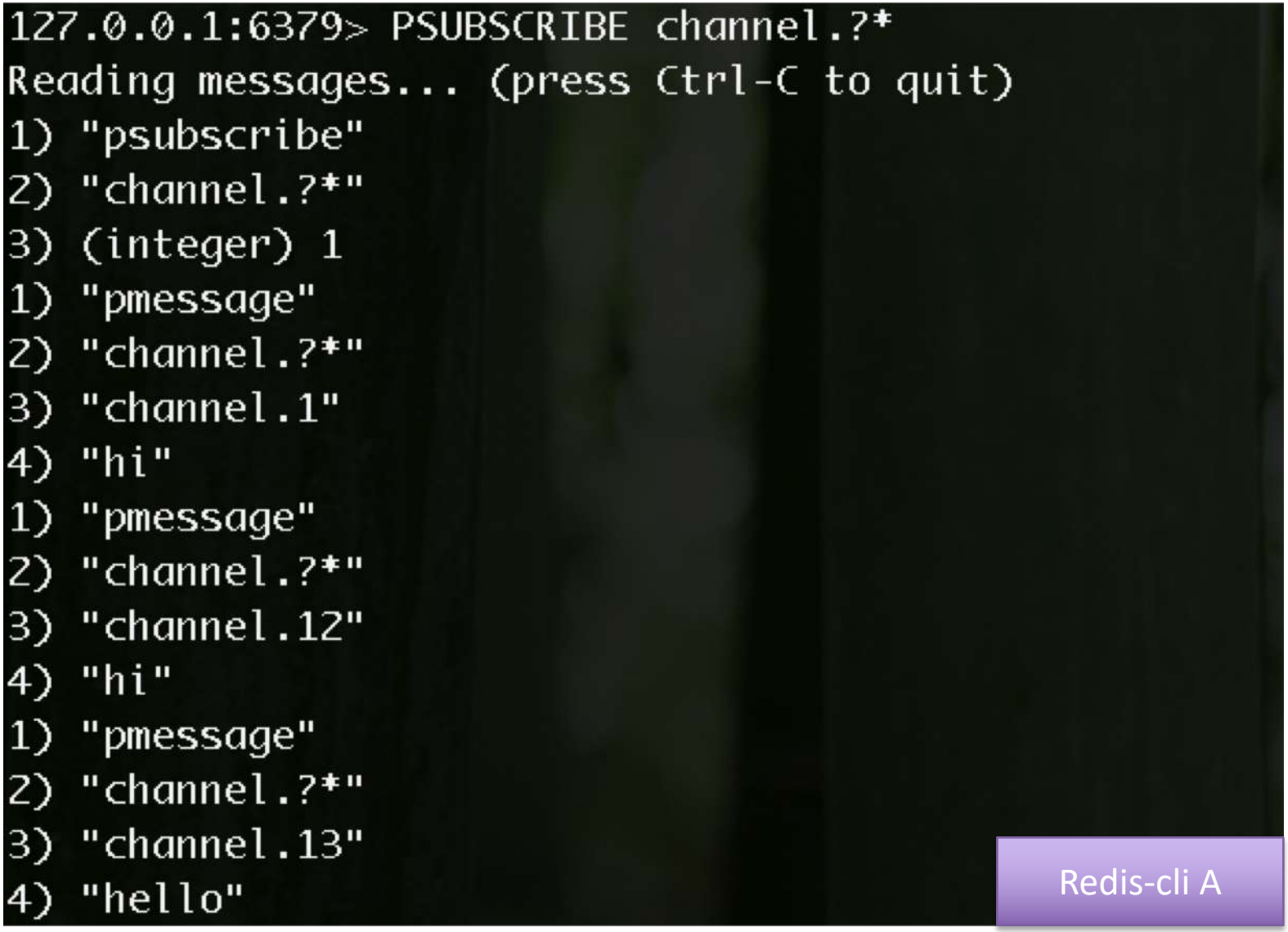

127.0.0.1:6379> SUBSCRIBE channel1.1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "channel1.1"
3) (integer) 1
Redis-cli A


127.0.0.1:6379> SUBSCRIBE channel1.1
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "channel1.1"
3) (integer) 1
Redis-cli B


127.0.0.1:6379> PUBLISH channel1.1 hi
(integer) 2
127.0.0.1:6379>
Redis-cli C


127.0.0.1:6379> PSUBSCRIBE channel.?*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"
2) "channel.?*"
3) (integer) 1
Redis-cli A


127.0.0.1:6379> PUBLISH channel1.1 hi
(integer) 0
127.0.0.1:6379> PUBLISH channel.1 hi
(integer) 1
127.0.0.1:6379> PUBLISH channel.12 hi
(integer) 1
127.0.0.1:6379> PUBLISH channel.13 hello
(integer) 1
127.0.0.1:6379>
Redis-cli B


127.0.0.1:6379> PSUBSCRIBE channel.?*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"
2) "channel.?*"
3) (integer) 1
1) "pmessage"
2) "channel.?*"
3) "channel.1"
4) "hi"
1) "pmessage"
2) "channel.?*"
3) "channel.12"
4) "hi"
1) "pmessage"
2) "channel.?*"
3) "channel.13"
4) "hello"
Redis-cli A

```
>>> p = r.pubsub(ignore_subscribe_messages=True)
>>> p.subscribe('my-channel')
>>> p.get_message()
>>> r.publish('my-channel','hello world')
1L
>>> p.get_message()
{'pattern': None, 'type': 'message', 'channel': 'my-channel', 'data': 'hello world'}
>>> p.get_message()
>>>
```

```
>>> while True:
>>>     message = p.get_message()
>>>     if message:
>>>         # do something with the message
>>>     time.sleep(0.001)  # be nice to the system :)
```

Persistence

- The **RDB** persistence performs point-in-time snapshots of your dataset at specified intervals

- the **AOF** persistence logs every write operation

received by the server, that will be played again

at server startup, reconstructing the original dataset.

```
#   In the example below the behaviour will be to save:
#   after 900 sec (15 min) if at least 1 key changed
#   after 300 sec (5 min) if at least 10 keys changed
#   after 60 sec if at least 10000 keys changed
#
#   Note: you can disable saving at all commenting all the "save" lines.

save 900 1
save 300 10
save 60 10000
```

```
# Compress string objects using LZF when dump .rdb databases?
# For default that's set to 'yes' as it's almost always a win.
# If you want to save some CPU in the saving child set it to 'no' but
# the dataset will likely be bigger if you have compressible values or keys.
rdbcompression yes

# The filename where to dump the DB
dbfilename dump.rdb

# The working directory.
#
# The DB will be written inside this directory, with the filename specified
# above using the 'dbfilename' configuration directive.
#
# Also the Append Only File will be created inside this directory.
#
# Note that you must specify a directory here, not a file name.
dir ./
```

```
appendonly no
```

```
# appendfsync always
appendfsync everysec
# appendfsync no
```

```
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

```
apple@justin-c-huang:etc (master)$ redis-server --port 6380 --slaveof 127.0.0.1 6379
20666:S 17 Jul 16:55:39.113 * Increased maximum number of open files to 10032 (it was originally set to 2560).
                _.-``__ ''-._
           _.-``    `.  `_.  ''-._           Redis 3.0.1 (00000000/0) 64 bit
       .-`` .-```.  ```\/    _.,_ ''-._
      (    '      ,       .-`  | `,    )     Running in standalone mode
      |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6380
      |    `-._   `._    /     _.-'    |     PID: 20666
       `-._    `-._  `-./  _.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |
       `-._    `-._`-.__.-'_.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |
       `-._    `-._`-.__.-'_.-'    _.-'
           `-._    `-.__.-'    _.-'
               `-._        _.-'
                   `-.__.-'

20666:S 17 Jul 16:55:39.116 # Server started, Redis version 3.0.1
20666:S 17 Jul 16:55:39.139 * DB loaded from disk: 0.023 seconds
20666:S 17 Jul 16:55:39.139 * The server is now ready to accept connections on port 6380
20666:S 17 Jul 16:55:40.121 * Connecting to MASTER 127.0.0.1:6379
20666:S 17 Jul 16:55:40.122 * MASTER <-> SLAVE sync started
20666:S 17 Jul 16:55:40.122 * Non blocking connect for SYNC fired the event.
20666:S 17 Jul 16:55:40.123 * Master replied to PING, replication can continue...
20666:S 17 Jul 16:55:40.124 * Partial resynchronization not possible (no cached master)
20666:S 17 Jul 16:55:40.124 * Full resync from master: 8cf660f6715d499a91754a8d5d2d2abfcb0f1bcc:1
20666:S 17 Jul 16:55:40.272 * MASTER <-> SLAVE sync: receiving 768721 bytes from master
20666:S 17 Jul 16:55:40.318 * MASTER <-> SLAVE sync: Flushing old data
20666:S 17 Jul 16:55:40.318 * MASTER <-> SLAVE sync: Loading DB in memory
20666:S 17 Jul 16:55:40.325 * MASTER <-> SLAVE sync: Finished with success
```

```
127.0.0.1:6380> GET foo1
"bar"
127.0.0.1:6380> SET foo1 hello
(error) READONLY You can't write against a read only slave.
127.0.0.1:6380>
```

RDB 的优点

- RDB 是一个非常

```
apple@justin-c-huang:~$ redis-cli
127.0.0.1:6379> SET foo hello
OK
127.0.0.1:6379> GET foo
"hello"
127.0.0.1:6379>
```

```
127.0.0.1:6379> MONITOR
OK
1437123860.013313 [0 127.0.0.1:49792] "SET" "foo" "hello"
1437123868.428989 [0 127.0.0.1:49792] "GET" "foo"
```

RDB 的优点

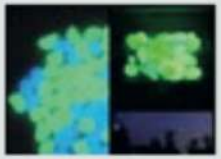
- RDB 是一个非

Articles can be voted on by clicking on up and down arrows.

reddit mobile

hot new controversial top saved

Forever Alone: spring break edition (imgur.com)
2162 points submitted 5 hours ago by aaarghas to funny
368

Glowstone driveway (i.imgur.com)
1740 points submitted 5 hours ago by icametodropbombs to pics

322

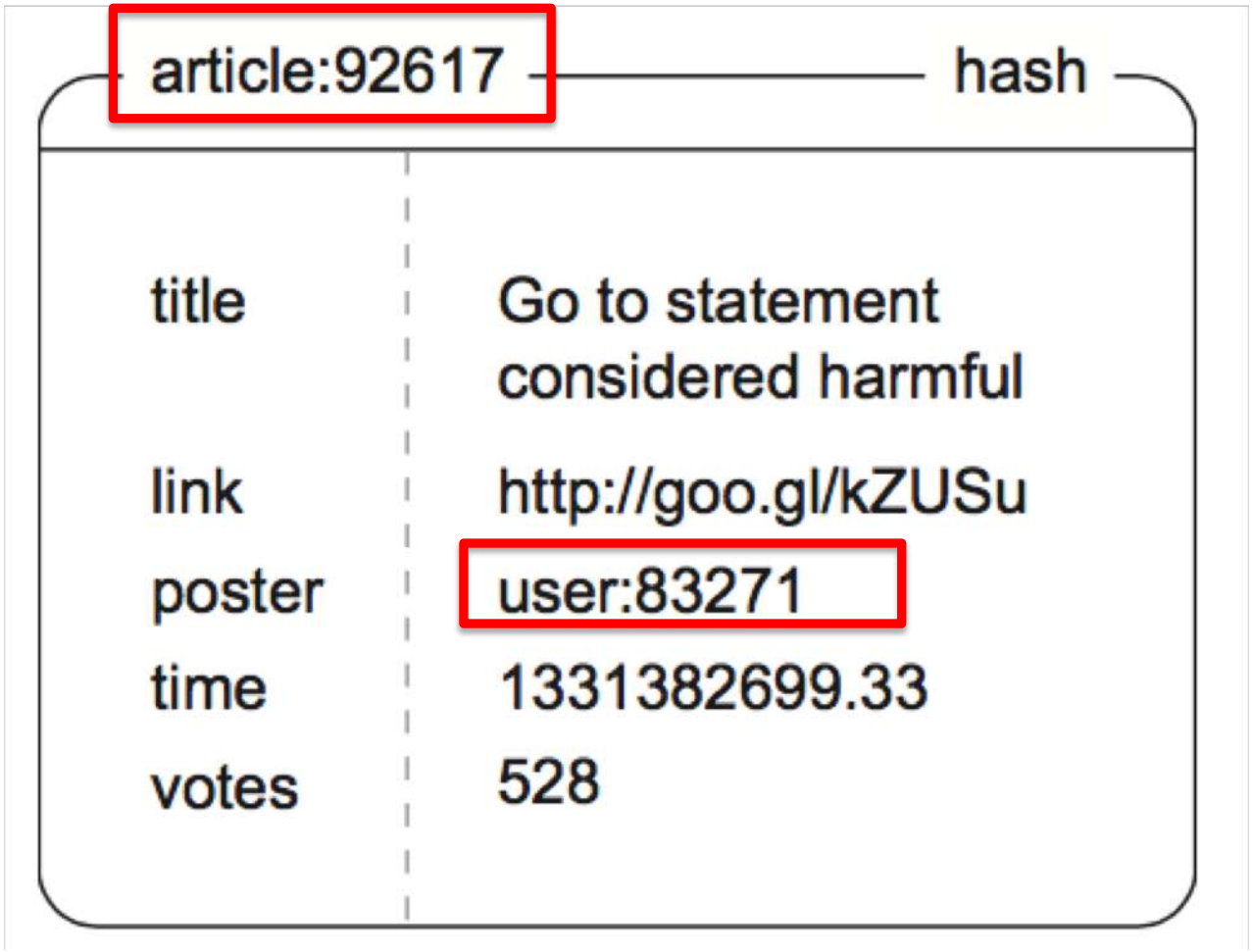
I won the Master Sword and Hylian Shield from the Skyward Sword commercial. They arrived today. (x-post from r/zelda) (imgur.com)
1696 points submitted 5 hours ago by Grant638 to gaming

489

Where is your god now? (i.imgur.com)
1168 points submitted 4 hours ago by justonecomic to WTF
174

I Decided To Consume Only Conservative News Sources For A Week, Here's What I Learned (self)
Aa
1537 points submitted 6 hours ago by Googlepus to politics
1313

Voting occurs after you have clicked through to read a question and any existing answers.

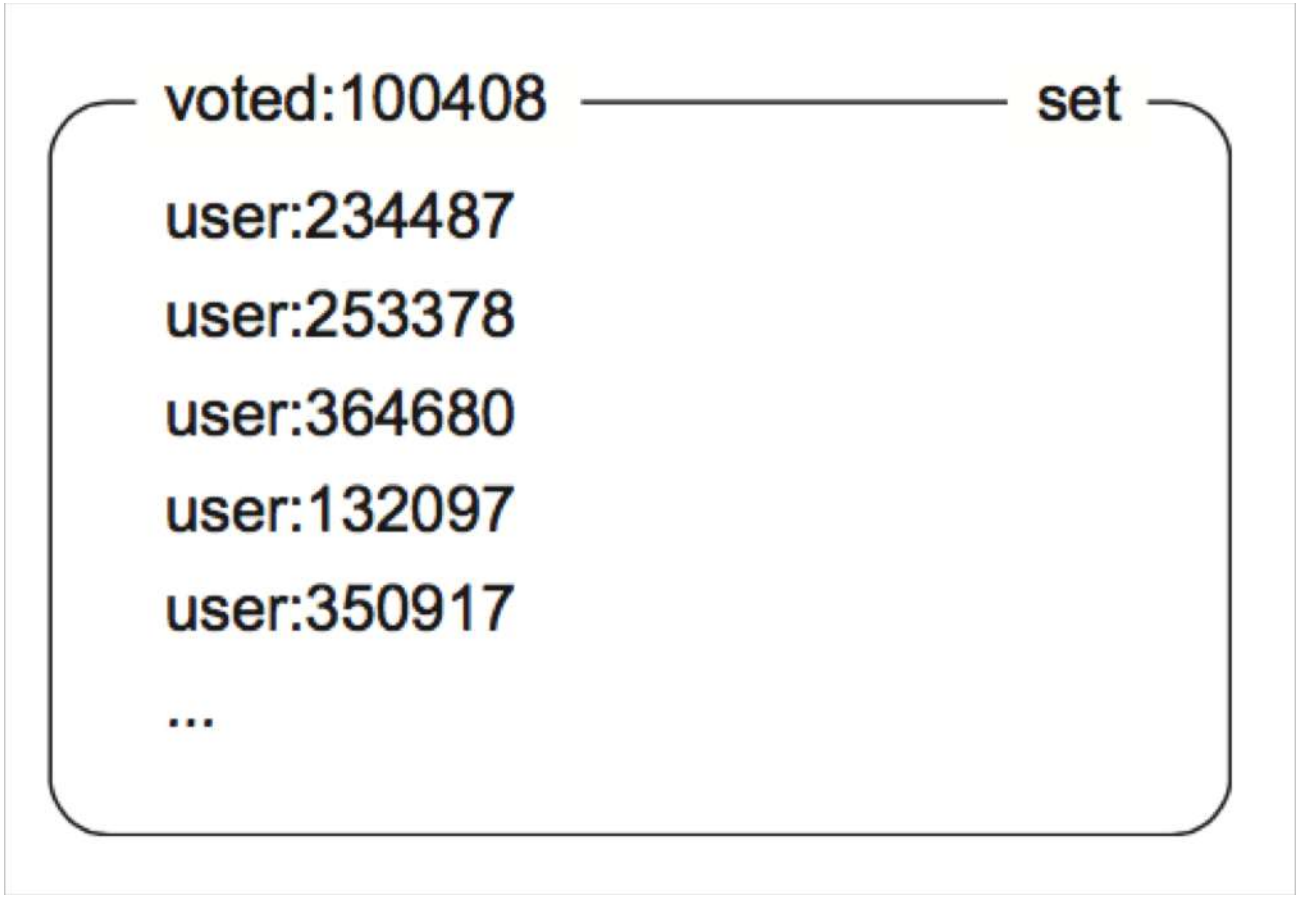

article:92617
hash
title
Go to statement considered harmful
link
http://goo.gl/kZUSu
poster
user:83271
time
1331382699.33
votes
528

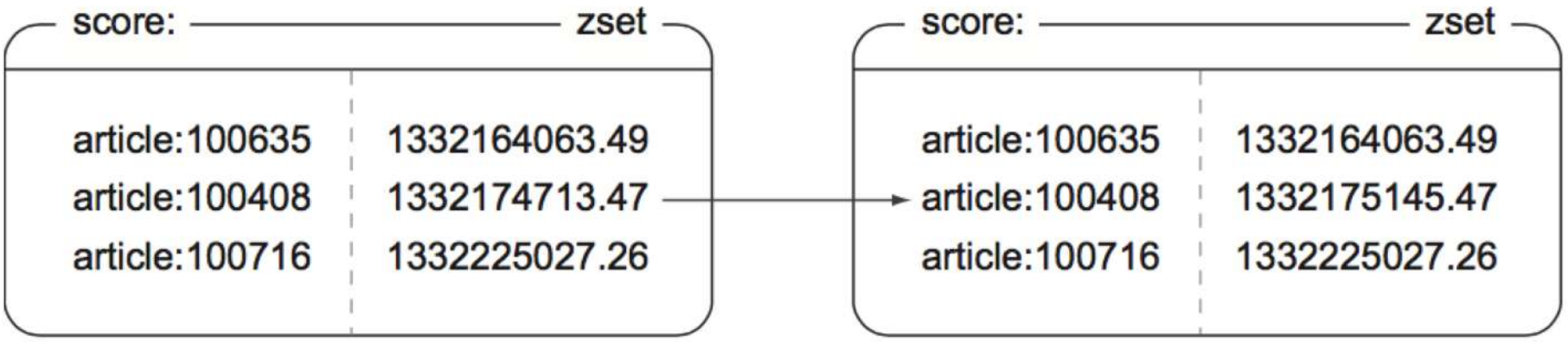
time:	zset
article:100408	1332065417.47
article:100635	1332075503.49
article:100716	1332082035.26

A time-ordered ZSET of articles

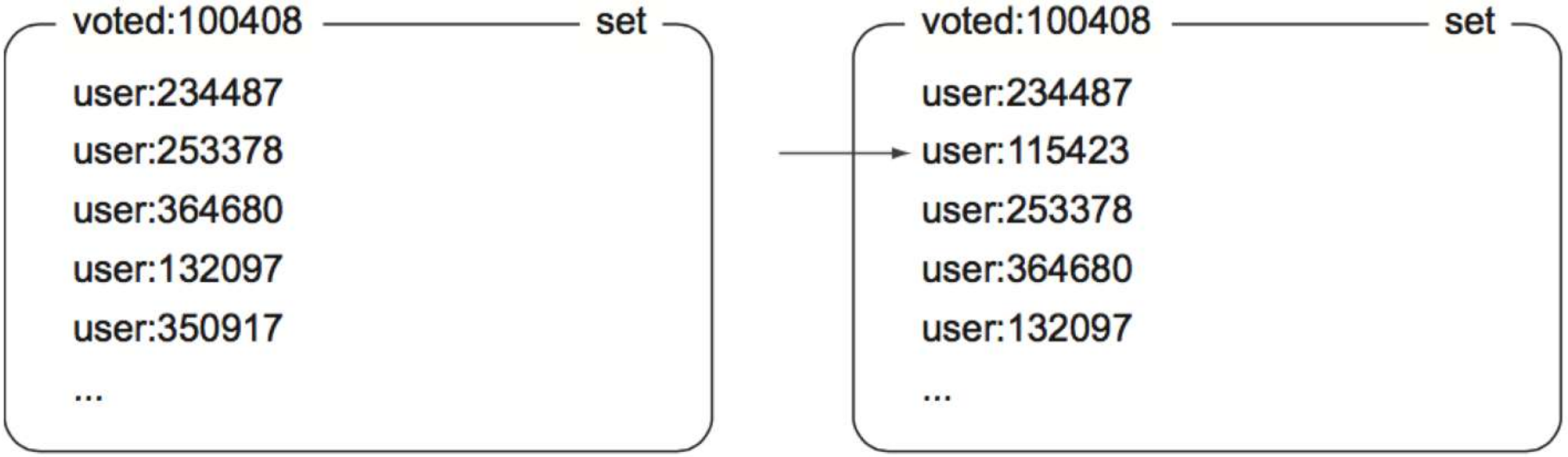
score:	zset
article:100635	1332164063.49
article:100408	1332174713.47
article:100716	1332225027.26

A score-ordered ZSET of articles


voted:100408
set
user:234487
user:253378
user:364680
user:132097
user:350917
...



Article 100408 got a new vote, so its score was increased.



Since user 115423 voted on the article, they are added to the voted SET.

```
ONE_WEEK_IN_SECONDS = 7 * 86400
VOTE_SCORE = 432

def article_vote(conn, user, article):
    cutoff = time.time() - ONE_WEEK_IN_SECONDS
    if conn.zscore('time:', article) < cutoff:
        return

    article_id = article.partition(':')[-1]
    if conn.sadd('voted:' + article_id, user):
        conn.zincrby('score:', article, VOTE_SCORE)
        conn.hincrby(article, 'votes', 1)
```

Prepare our constants.

Calculate the cutoff time for voting.

Check to see if the article can still be voted on (we could use the article HASH here, but scores are returned as floats so we don't have to cast it).

Get the id portion from the article:id identifier.

If the user hasn't voted for this article before, increment the article score and vote count.

```
def post_article(conn, user, title, link):
    article_id = str(conn.incr('article:'))

    voted = 'voted:' + article_id
    conn.sadd(voted, user)
    conn.expire(voted, ONE_WEEK_IN_SECONDS)

    now = time.time()
    article = 'article:' + article_id
    conn.hmset(article, {
        'title': title,
        'link': link,
        'poster': user,
        'time': now,
        'votes': 1,
    })

    conn.zadd('score:', article, now + VOTE_SCORE)
    conn.zadd('time:', article, now)

    return article_id
```

Generate a new article id.

Start with the posting user having voted for the article, and set the article voting information to automatically expire in a week (we discuss expiration in chapter 3).

Create the article hash.

Add the article to the time and score ordered ZSETs.

```
ARTICLES_PER_PAGE = 25

def get_articles(conn, page, order='score:'):
    start = (page-1) * ARTICLES_PER_PAGE
    end = start + ARTICLES_PER_PAGE - 1

    ids = conn.zrevrange(order, start, end)
    articles = []
    for id in ids:
        article_data = conn.hgetall(id)
        article_data['id'] = id
        articles.append(article_data)

    return articles
```

Set up the start and end indexes for fetching the articles.

Fetch the article ids.

Get the article information from the list of article ids.

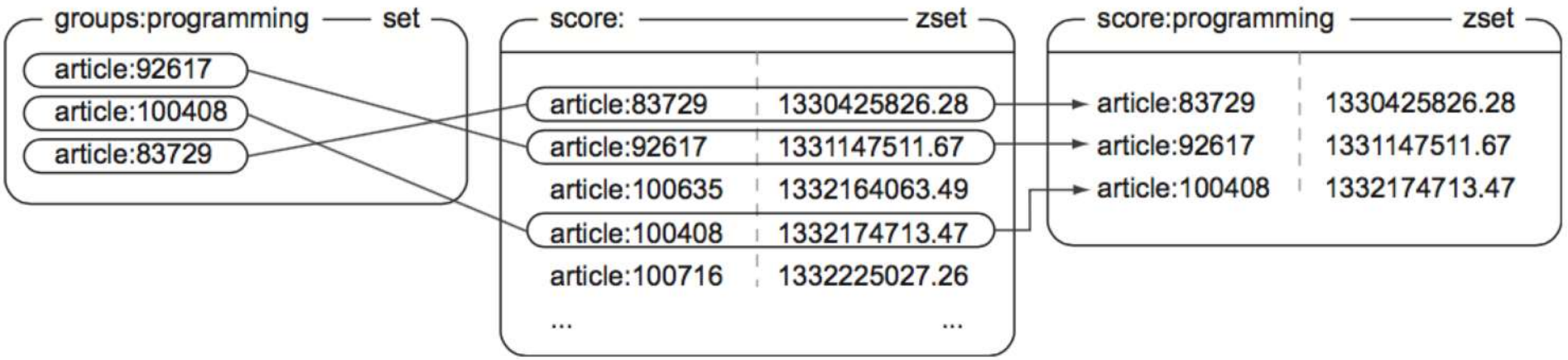
Get the article information from the list of article ids.

```
def add_remove_groups(conn, article_id, to_add=[], to_remove=[]):
    article = 'article:' + article_id
    for group in to_add:
        conn.sadd('group:' + group, article)
    for group in to_remove:
        conn.srem('group:' + group, article)
```

Construct the article information like we did in post_article.

Add the article to groups that it should be a part of.

Remove the article from groups that it should be removed from.


groups:programming
set
article:92617
article:100408
article:83729
score:
zset
article:83729
1330425826.28
article:92617
1331147511.67
article:100635
1332164063.49
article:100408
1332174713.47
article:100716
1332225027.26
...
...
score:programming
zset
article:83729
1330425826.28
article:92617
1331147511.67
article:100408
1332174713.47

```
def get_group_articles(conn, group, page, order='score:'):
    key = order + group
    if not conn.exists(key):
        conn.zinterstore(key,
            ['group:' + group, order],
            aggregate='max',
        )
        conn.expire(key, 60)
    return get_articles(conn, page, key)
```

Create a key for each group and each sort order.

If we haven't sorted these articles recently, we should sort them.

Actually sort the articles in the group based on score or recency.

Tell Redis to automatically expire the ZSET in 60 seconds.

Call our earlier get_articles() function to handle pagination and article data fetching.

Table 2.1 **Pros and cons of signed cookies and token cookies**

Cookie type	Pros	Cons
Signed cookie	Everything needed to verify the cookie is in the cookie Additional information can be included and signed easily	Correctly handling signatures is hard It's easy to forget to sign and/or verify data, allowing security vulnerabilities
Token cookie	Adding information is easy Very small cookie, so mobile and slow clients can send requests faster	More information to store on the server If using a relational database, cookie loading/storing can be expensive

Listing 2.1 The `check_token()` function

```
def check_token(conn, token):
    return conn.hget('login:', token)
```

Fetch and return the given user, if available.

Listing 2.2 The `update_token()` function

```
def update_token(conn, token, user, item=None):
    timestamp = time.time()
    conn.hset('login:', token, user)
    conn.zadd('recent:', token, timestamp)
    if item:
        conn.zadd('viewed:' + token, item, timestamp)
        conn.zremrangebyrank('viewed:' + token, 0, -26)
```

Get the timestamp.

Keep a mapping from the token to the logged-in user.

Record when the token was last seen.

Record that the user viewed the item.

Remove old items, keeping the most recent 25.

Listing 2.3 The `clean_sessions()` function

```
QUIT = False
LIMIT = 10000000

def clean_sessions(conn):
    while not QUIT:
        size = conn.zcard('recent:')
        if size <= LIMIT:
            time.sleep(1)
            continue

        end_index = min(size - LIMIT, 100)
        tokens = conn.zrange('recent:', 0, end_index-1)

        session_keys = []
        for token in tokens:
            session_keys.append('viewed:' + token)

        conn.delete(*session_keys)
        conn.hdel('login:', *tokens)
        conn.zrem('recent:', *tokens)
```

Find out how many tokens are known.

We're still under our limit; sleep and try again.

Fetch the token IDs that should be removed.

Prepare the key names for the tokens to delete.

Remove the oldest tokens.

Listing 2.4 The add_to_cart() function

```
def add_to_cart(conn, session, item, count):
    if count <= 0:
        conn.hrem('cart:' + session, item)
    else:
        conn.hset('cart:' + session, item, count)
```

Remove the item from the cart.

Add the item to the cart.

Listing 2.5 The clean_full_sessions() function

```
def clean_full_sessions(conn):
    while not QUIT:
        size = conn.zcard('recent:')
        if size <= LIMIT:
            time.sleep(1)
            continue

        end_index = min(size - LIMIT, 100)
        sessions = conn.zrange('recent:', 0, end_index-1)

        session_keys = []
        for sess in sessions:
            session_keys.append('viewed:' + sess)
            session_keys.append('cart:' + sess)

        conn.delete(*session_keys)
        conn.hdel('login:', *sessions)
        conn.zrem('recent:', *sessions)
```

The required added line to delete the shopping cart for old sessions

Listing 2.6 The `cache_request()` function

```
def cache_request(conn, request, callback):
    if not can_cache(conn, request):
        return callback(request)

    page_key = 'cache:' + hash_request(request)
    content = conn.get(page_key)

    if not content:
        content = callback(request)
        conn.setex(page_key, content, 300)

    return content
```

If we can't cache the request, immediately call the callback.

Convert the request into a simple string key for later lookups.

Fetch the cached content if we can, and it's available.

Generate the content if we can't cache the page, or if it wasn't cached.

Cache the newly generated content if we can cache it.

Return the content.

Listing 2.7 The `schedule_row_cache()` function

```
def schedule_row_cache(conn, row_id, delay):
    conn.zadd('delay:', row_id, delay)
    conn.zadd('schedule:', row_id, time.time())
```

Set the delay for the item first.

Schedule the item to be cached now.

Listing 2.8 The cache_rows() daemon function

```
def cache_rows(conn):
    while not QUIT:
        next = conn.zrange('schedule:', 0, 0, withscores=True)
        now = time.time()
        if not next or next[0][1] > now:
            time.sleep(.05)
            continue

        row_id = next[0][0]
```

Find the next row that should be cached (if any), including the timestamp, as a list of tuples with zero or one items.

No rows can be cached now, so wait 50 milliseconds and try again.

```
        delay = conn.zscore('delay:', row_id)
        if delay <= 0:
            conn.zrem('delay:', row_id)
            conn.zrem('schedule:', row_id)
            conn.delete('inv:' + row_id)
            continue

        row = Inventory.get(row_id)
        conn.zadd('schedule:', row_id, now + delay)
        conn.set('inv:' + row_id, json.dumps(row.to_dict()))
```

Get the delay before the next schedule.

The item shouldn't be cached anymore; remove it from the cache.

Get the database row.

Update the schedule and set the cache value.

Listing 2.9 The updated `update_token()` function

```
def update_token(conn, token, user, item=None):
    timestamp = time.time()
    conn.hset('login:', token, user)
    conn.zadd('recent:', token, timestamp)
    if item:
        conn.zadd('viewed:' + token, item, timestamp)
        conn.zremrangebyrank('viewed:' + token, 0, -26)
        conn.zincrby('viewed:', item, -1)
```

The line we need to add to update_token()

Listing 2.10 The `rescale_viewed()` daemon function

```
def rescale_viewed(conn):
    while not QUIT:
        conn.zremrangebyrank('viewed:', 20000, -1)
        conn.zinterstore('viewed:', {'viewed:': .5})
        time.sleep(300)
```

Remove any item not in the top 20,000 viewed items.

Rescale all counts to be 1/2 of what they were before.

Do it again in 5 minutes.

Listing 2.11 The `can_cache()` function

```
def can_cache(conn, request):
    item_id = extract_item_id(request)
    if not item_id or is_dynamic(request):
        return False
    rank = conn.zrank('viewed:', item_id)
    return rank is not None and rank < 10000
```

Get the item ID for the page, if any.

Check whether the page can be statically cached and whether this is an item page.

Get the rank of the item.

Return whether the item has a high enough view count to be cached.

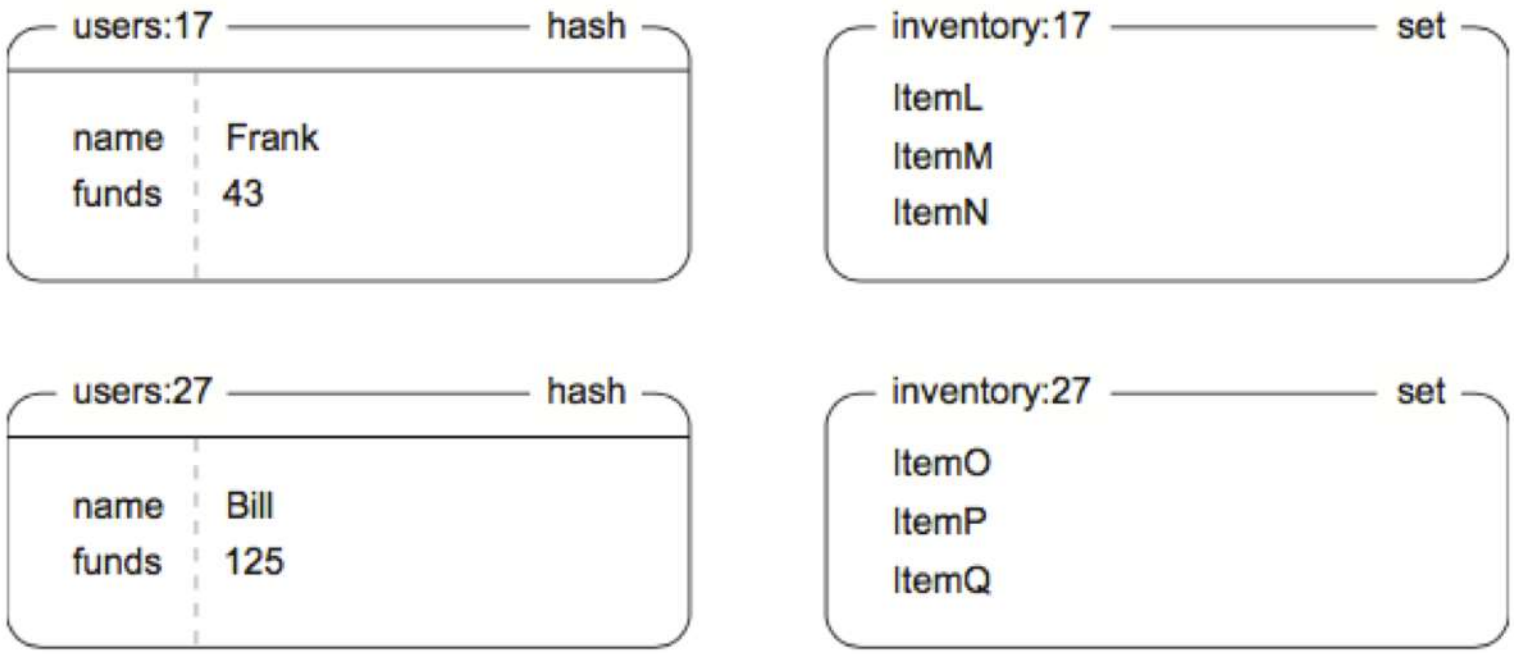


Figure 4.2 Example user inventory and user information. Frank has 43 e-dollars and an item that he's considering selling from his inventory.

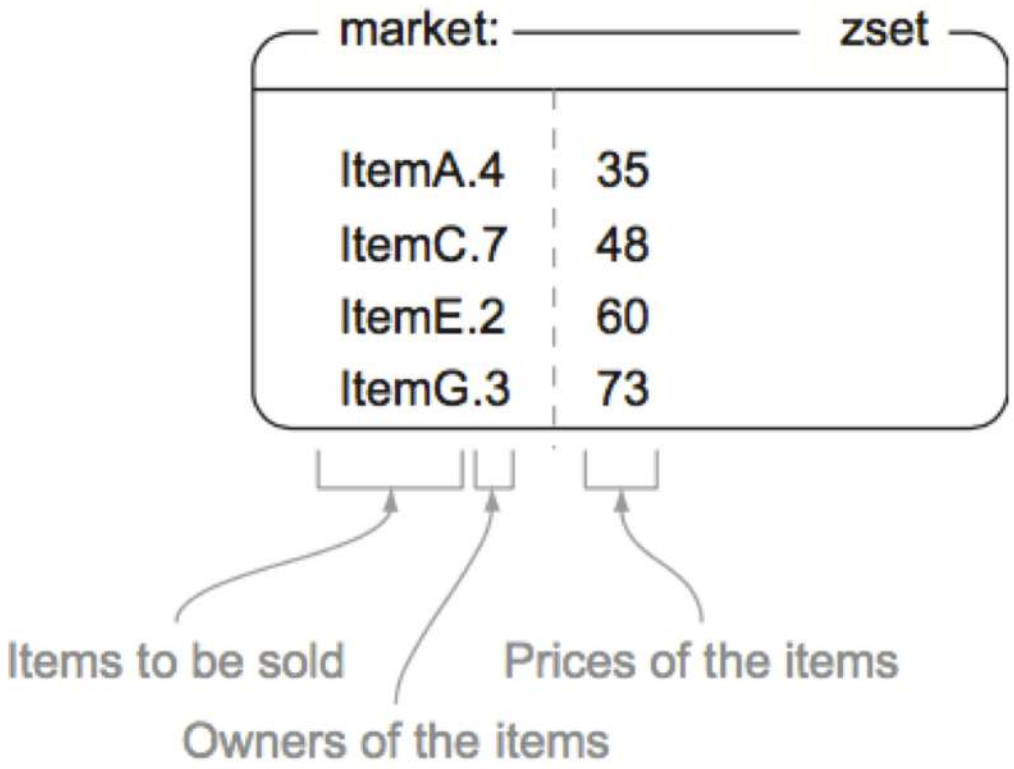


Figure 4.3 Our basic marketplace that includes an ItemA being sold by user 4 for 35 e-dollars

Listing 4.5 The `list_item()` function

```
def list_item(conn, itemid, sellerid, price):
    inventory = "inventory:%s"%sellerid
    item = "%s.%s"%(itemid, sellerid)
    end = time.time() + 5
    pipe = conn.pipeline()

    while time.time() < end:
        try:
            pipe.watch(inventory)
            if not pipe.sismember(inventory, itemid):
                pipe.unwatch()
                return None

            pipe.multi()
            pipe.zadd("market:", item, price)
            pipe.srem(inventory, itemid)
            pipe.execute()
            return True
        except redis.exceptions.WatchError:
            pass
    return False
```

Watch for changes to the user's inventory.

Verify that the user still has the item to be listed.

If the item isn't in the user's inventory, stop watching the inventory key and return.

If execute returns without a WatchError being raised, then the transaction is complete and the inventory key is no longer watched.

Actually list the item.

The user's inventory was changed; retry.

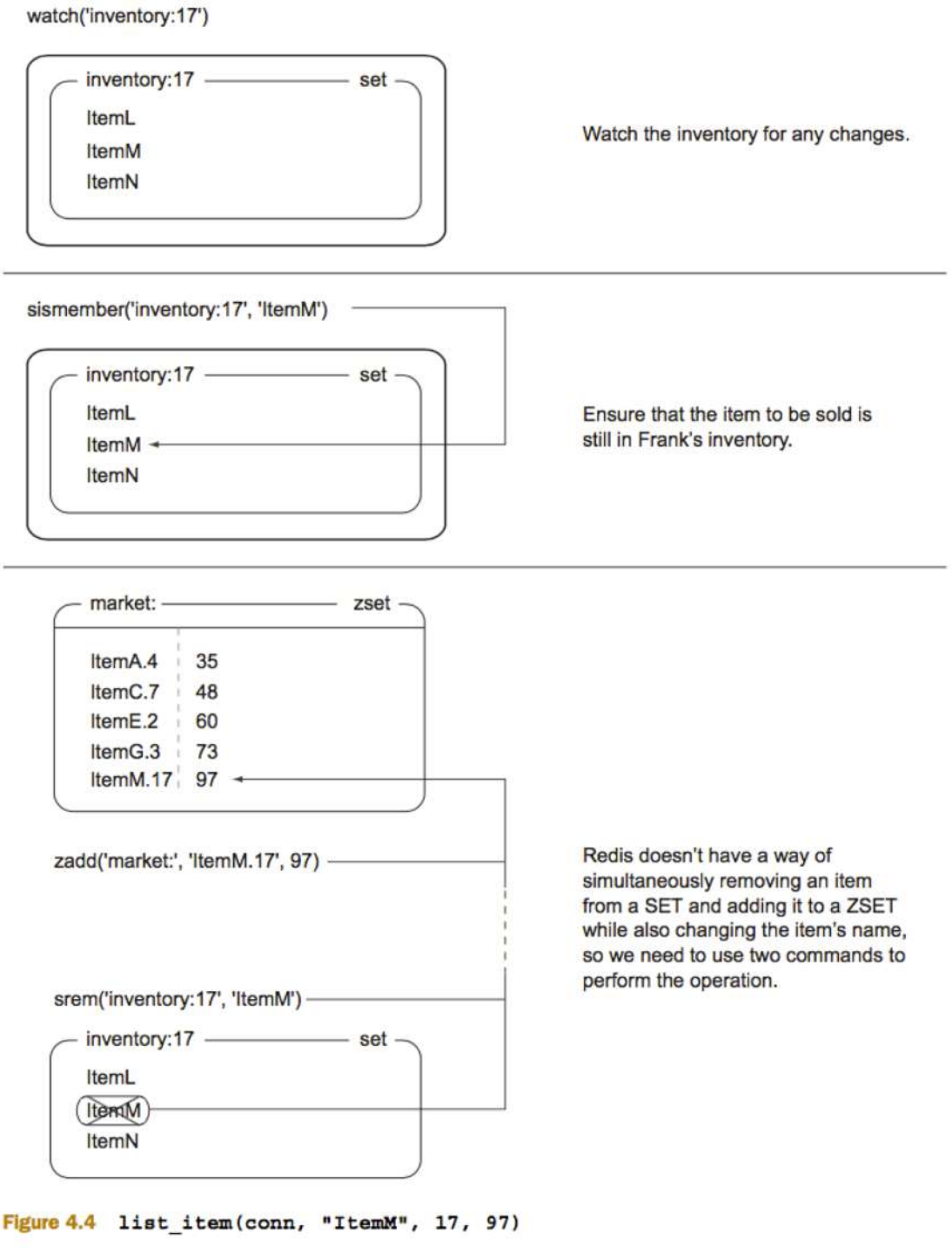


Figure 4.4 `list_item(conn, "ItemM", 17, 97)`

Listing 4.6 The `purchase_item()` function

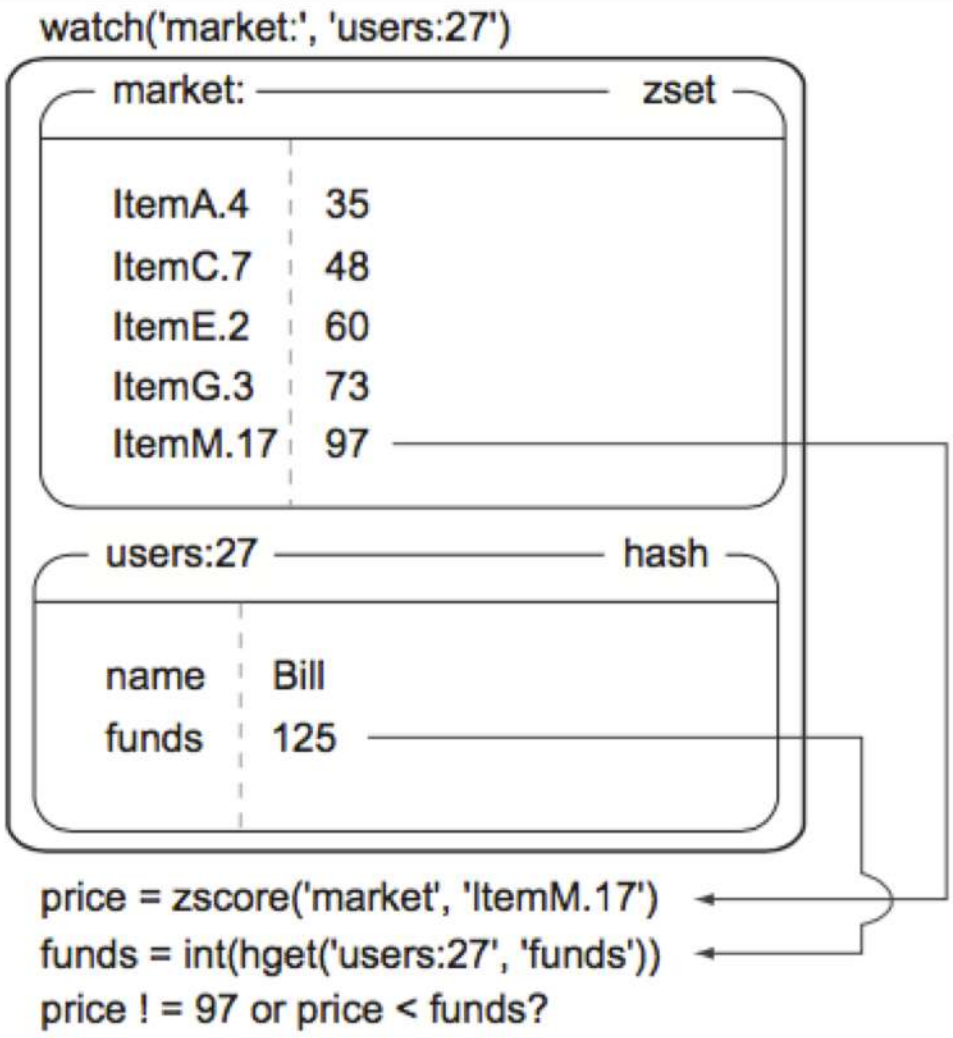
```
def purchase_item(conn, buyerid, itemid, sellerid, lprice):
    buyer = "users:%s"%buyerid
    seller = "users:%s"%sellerid
    item = "%s.%s"%(itemid, sellerid)
    inventory = "inventory:%s"%buyerid
    end = time.time() + 10
    pipe = conn.pipeline()

    while time.time() < end:
        try:
            pipe.watch("market:", buyer)

            price = pipe.zscore("market:", item)
            funds = int(pipe.hget(buyer, "funds"))
            if price != lprice or price > funds:
                pipe.unwatch()
                return None

            pipe.multi()
            pipe.hincrby(seller, "funds", int(price))
            pipe.hincrby(buyer, "funds", int(-price))
            pipe.sadd(inventory, itemid)
            pipe.zrem("market:", item)
            pipe.execute()
            return True
        except redis.exceptions.WatchError:
            pass

    return False
```

Watch for changes to the market and to the buyer's account information.

Check for a sold/repriced item or insufficient funds.

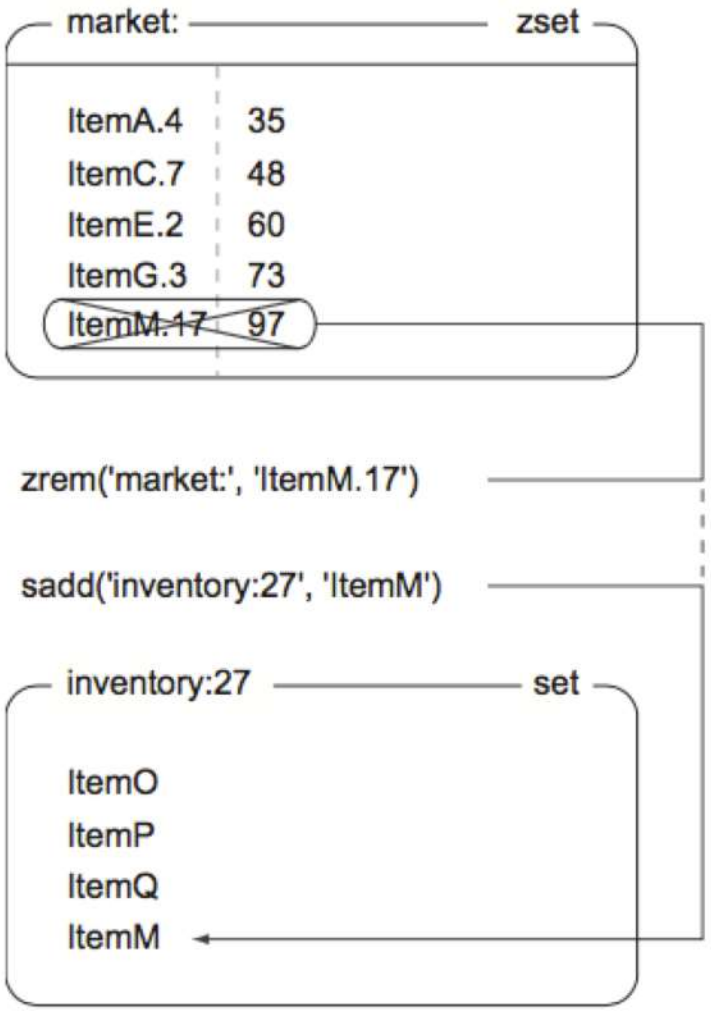
Transfer funds from the buyer to the seller, and transfer the item to the buyer.

Retry if the buyer's account or the market changed.

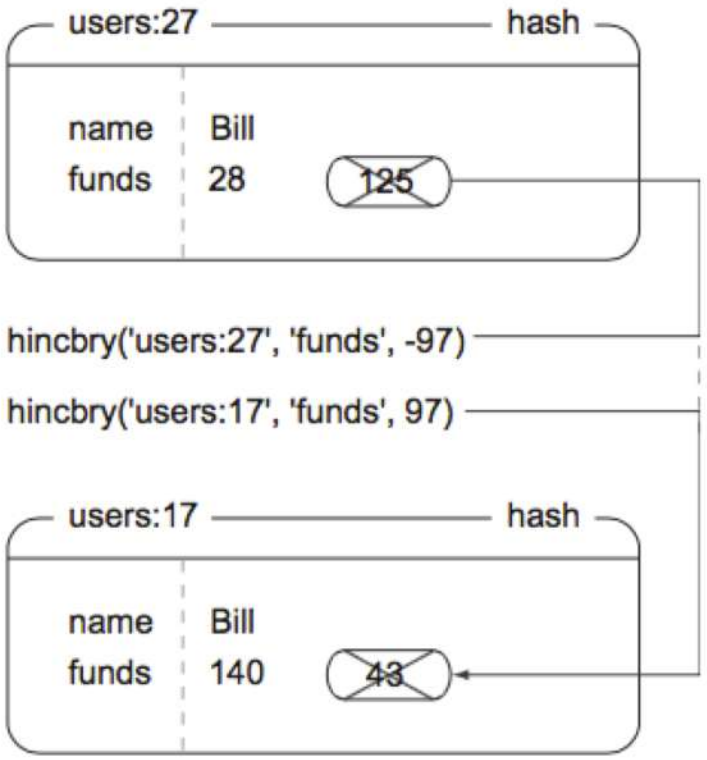


Watch the market and Bill's information for changes.

Verify that the item is still listed for the same price, and that Bill still has enough money.



Move the item into Bill's inventory.



Move money from Bill to Frank.