SO YOU THINK YOU KNOW REDIS!

SO YOU THINK YOU KNOW REDIS!

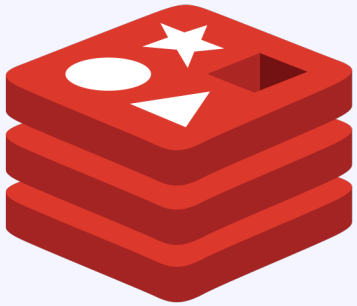
SO YOU THINK YOU KNOW REDIS!

with

Guy Royse

SO YOU THINK YOU KNOW REDIS!

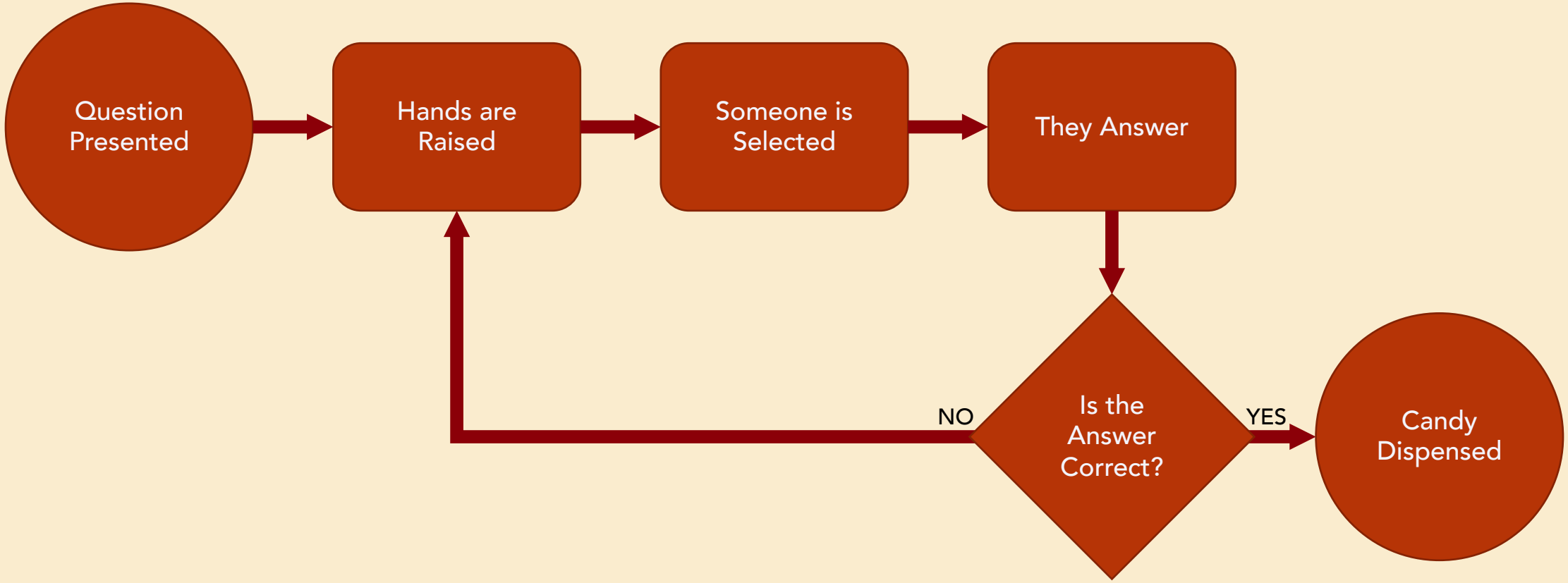
is brought to you by



redisenterprise
CLOUD

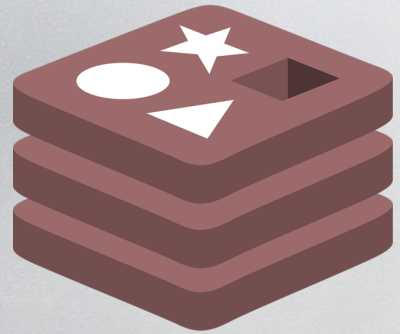
Use code STACK200 and sign up for a free Redis Cloud database at redis.com/try-free.

How This is Gonna Work



But before we get started...

...how's your Redis?



FIRST QUESTION

What does redis-cli return when you run the following commands:

```
SET foo "👋🌎"
GET foo
```

A. `(error) ERR syntax error`

B. `"\xf0\x9f\x91\x8b\xf0\x9f\x8c\x8e"`

C. `(error) ERR value is not a string`

D. `"👋🌎"`

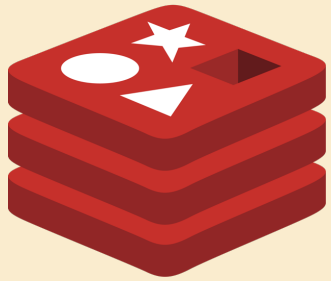
GET and SET

```
SET foo Hello
SET foo "Hello World!"
GET foo
```

GET, SET, and String Encoding

How does Redis encode Strings? It doesn't.

`SET foo "👋🌎"`



```
f0 9f 91 8b
f0 9f 8c 8e
```

```
"\xf0\x9f\x91\x8b\xf0\x9f\x8c\x8e"
```

redis-cli Encodes Strings

```
48 65 6c 6c
6f 20 57 6f
72 6c 64 21
```

```
"Hello World!"
```

```
48 08 09 0a
57 0c 0d 5c
```

```
"H\b\t\nW\f\r\\"
```

```
f0 9f 91 8b
f0 9f 8c 8e
```

```
"\xf0\x9f\x91\x8b\xf0\x9f\x8c\x8e"
```

Encoding

```
08 ⇢ \b
0a ⇢ \n
0c ⇢ \f
0d ⇢ \r
09 ⇢ \t
5c ⇢ \\
01 ⇢ \x01
ff ⇢ \xff
```

GET and SET Happily Accept Binary Data

```
SET foo
   "\x01\x02\x03\x04"
```

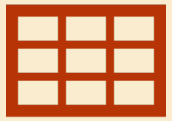
01 02 03 04

```
GET foo                    "\x01\x02\x03\x04"
```

This Applies to Any String in Redis

```
LPUSH foo "\x01\x02\x03"
```

```
SADD foo "\x01\x02\x03"
```



```
HSET foo bar "\x01\x02\x03"
```

```
SET "\x01\x02\x03" "Hello World!"
HSET foo "\x01\x02\x03" "Hello World!"
```

How to Get Binary Data Into Redis

From the command-line:

```
redis-cli -x set foo < foo.bin
redis-cli get foo > foo.bin
```

From code:

```
const buffer = await fsPromises.readFile('./foo.bin')
await redis.set('foo', buffer)
```

DEMO

NEXT QUESTION

The key *foo* contains a String with the value of "Hail to the King, baby!". What does the following command return:

```
BITFIELD foo GET u7 1
```

A. 1) (integer) 104

B. (error) ERR unknown command `BITFIELD`, with args beginning with: `foo`, `GET`, `u7`, `1`,

C. (error) ERR Invalid bitfield type. Use something like i16 u8. Note that u64 is not supported but i64 is.

D. 1) (integer) 72

Extracting Arbitrary Numbers

```
H a i l    48 61 69 6c    01001000 01100001 01101001 01101100
  t o      20 74 6f 20    00100000 01110100 01101111 00100000
t h e      74 68 65 20    01110100 01101000 01100101 00100000
K i n g    4b 69 6e 67    01001011 01101001 01101110 01100111
,   b a    2c 20 62 61    00101100 00100000 01100010 01100001
b y !      62 79 21       01100010 01111001 00100001
```

```
> BITFIELD foo GET u7 1
1) (integer) 72

> BITFIELD foo GET i13 23
1) (integer) -2366

> BITFIELD foo GET i4 64 GET i6 73
1) (integer) 7
2) (integer) -12
```

Changing Arbitrary Numbers

```
H a i l    48 61 69 6c    01001000 01100001 01101001 01101100
  t o      20 74 6f 20    00100000 01110100 01101111 00100000
t h e      74 68 65 20    01110100 01101000 01100101 00100000
K i n g    4b 69 6e 67    01001011 01101001 01101110 01100111
,   b a    2c 20 62 61    00101100 00100000 01100010 01100001
b y !      62 79 21       01100010 01111001 00100001
```

```
> BITFIELD foo SET u7 1 70
1) (integer) 72
```

Changing Arbitrary Numbers

```
F a i l    46 61 69 6c    01000110 01100001 01101001 01101100
  t o      20 74 6f 20    00100000 01110100 01101111 00100000
t h e      74 68 65 20    01110100 01101000 01100101 00100000
K i n g    4b 69 6e 67    01001011 01101001 01101110 01100111
,   b a    2c 20 62 61    00101100 00100000 01100010 01100001
b y !      62 79 21       01100010 01111001 00100001
```

```
> BITFIELD foo SET u7 1 70
1) (integer) 72

> BITFIELD foo INCRBY u7 1 4
```

Changing Arbitrary Numbers

```
F a i l    46 61 69 6c    01000110 01100001 01101001 01101100
  t o      20 74 6f 20    00100000 01110100 01101111 00100000
t h e      74 68 65 20    01110100 01101000 01100101 00100000
K i n g    4b 69 6e 67    01001011 01101001 01101110 01100111
,   b a    2c 20 62 61    00101100 00100000 01100010 01100001
b y !      62 79 21       01100010 01111001 00100001
```

```
> BITFIELD foo SET u7 1 70
1) (integer) 72

> BITFIELD foo INCRBY u7 1 4
```

Changing Arbitrary Numbers

```
J a i l    4a 61 69 6c    01001010 01100001 01101001 01101100
  t o      20 74 6f 20    00100000 01110100 01101111 00100000
t h e      74 68 65 20    01110100 01101000 01100101 00100000
K i n g    4b 69 6e 67    01001011 01101001 01101110 01100111
,   b a    2c 20 62 61    00101100 00100000 01100010 01100001
b y !      62 79 21       01100010 01111001 00100001
```

```
> BITFIELD foo SET u7 1 70
1) (integer) 72

> BITFIELD foo INCRBY u7 1 4
1) (integer) 74
```

Lots of Bit Manipulation Commands

```
J a i l   4a 61 69 6c   01001010 01100001 01101001 01101100
  t o     20 74 6f 20   00100000 01110100 01101111 00100000
t h e     74 68 65 20   01110100 01101000 01100101 00100000
K i n g   4b 69 6e 67   01001011 01101001 01101110 01100111
,   b a   2c 20 62 61   00101100 00100000 01100010 01100001
b y !     62 79 21      01100010 01111001 00100001
```

```
> GETBIT foo 17
1) (integer) 1

> GETBIT foo 19
1) (integer) 0
```

```
> SETBIT foo 17 0
1) (integer) 1

> SETBIT foo 19 1
1) (integer) 0
```

Lots of Bit Manipulation Commands

```
J a 9 l    4a 61 39 6c    01001010 01100001 00111001 01101100
  t o      20 74 6f 20    00100000 01110100 01101111 00100000
t h e      74 68 65 20    01110100 01101000 01100101 00100000
K i n g    4b 69 6e 67    01001011 01101001 01101110 01100111
,   b a    2c 20 62 61    00101100 00100000 01100010 01100001
b y !      62 79 21       01100010 01111001 00100001
```

```
> GETBIT foo 17
1) (integer) 1

> GETBIT foo 19
1) (integer) 0
```

```
> SETBIT foo 17 0
1) (integer) 1

> SETBIT foo 19 1
1) (integer) 0
```

Lots of Bit Manipulation Commands

```
H a i l    48 61 69 6c    01001000 01100001 01101001 01101100
  t o      20 74 6f 20    00100000 01110100 01101111 00100000
t h e      74 68 65 20    01110100 01101000 01100101 00100000
K i n g    4b 69 6e 67    01001011 01101001 01101110 01100111
,   b a    2c 20 62 61    00101100 00100000 01100010 01100001
b y !      62 79 21       01100010 01111001 00100001
```

```
> BITCOUNT foo
(integer) 74

> BITPOS foo 1
(integer) 1
```

Bitwise Operations

foo	bar	baz	dest
01000001	01000010	01000011	

```
> BITOP AND dest foo bar baz                    01000000
(integer) 1
> BITOP AND dest foo bar baz                    01000011
(integer) 1
> BITOP XOR dest foo bar baz                    00000000
(nil)
> BITOP NOT dest foo                            10111110
(integer) 1
```

DEMO

NEXT QUESTION

I have set my *maxmemory-policy* to *volatile-lru*. What happens when I add something to Redis and it's out of memory?

A. You get an error.

B. It removes the key accessed *least recently*.

C. It removes the key accessed *least recently* that has a time to live.

D. It removes a *random* key.

Setting a Max Memory Policy

redis.conf

```
> CONFIG SET maxmemory-policy volatile-lru
> CONFIG REWRITE
```

volatile-lru: Evict using approximated LRU, only keys with an expire set.
allkeys-lru: Evict any key using approximated LRU.
volatile-lfu: Evict using approximated LFU, only keys with an expire set.
allkeys-lfu: Evict any key using approximated LFU.
volatile-random: Remove a random key having an expire set.
allkeys-random: Remove a random key, any key.
volatile-ttl: Remove the key with the nearest expire time (minor TTL)
noeviction: Don't evict anything, just return an error on write operations.


How Does Redis Know When It's Full?
You tell it!
> CONFIG SET maxmemory 536870912

Persistent vs. Volatiles Keys

Stick around forever.
Keys are persistent by default.

Have a time-to-live.
Are removed on or after that TTL.

Persistent vs. Volatiles keys

```
> SET foo bar
```

```
> EXPIRE foo 3600
> EXPIREAT foo 2147483647
```

```
> TTL foo
> EXPIRETIME foo
```

```
> PERSIST foo
```

The Neverfull Cache

```
> CONFIG SET maxmemory 536870912
> CONFIG SET maxmemory-policy allkeys-lru
```






01-Jan-2022 02-Jan-2022 03-Jan-2022 04-Jan-2022 05-Jan-2022 06-Jan-2022

The One Key Cache

```
> CONFIG SET maxmemory 1000000
> CONFIG SET maxmemory-policy allkeys-random
```



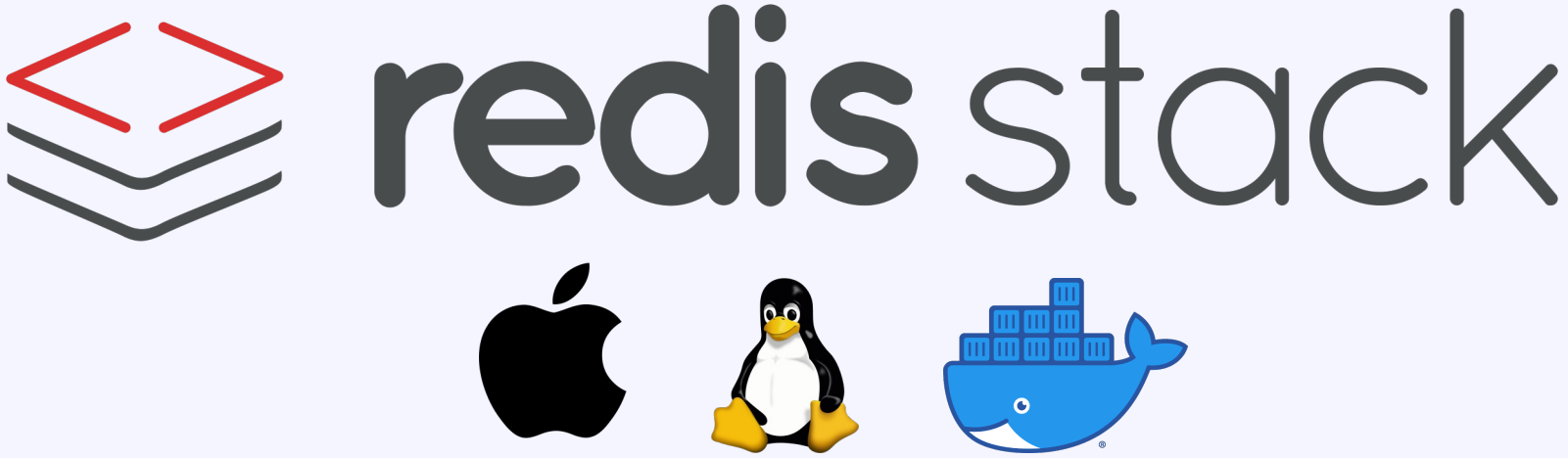



DEMO

WE'LL BE RIGHT BACK

SO YOU THINK YOU KNOW REDIS!

is made possible by



Install Redis Stack today with Docker using the command:

```
docker run -d --name redis-stack-server -p 6379:6379 redis/redis-stack-server:latest
```

WELCOME BACK

NEXT QUESTION

When running *open-source* Redis, what does the following command return:

```
MODULE UNLOAD graph
```

A. `(error) ERR Error unloading module: no such module with that name`

B. `(error) ERR Error unloading module: the module exports one or more module-side data types, can't unload`

C. `(error) ERR syntax error`

D. `OK`

What's a Module

Extension to Redis that add new data types and new commands.

Written in system-level languages like C, Rust, Go, etc.

RedisImage

```
> IMAGE.SET foo "\x01\x02\x03..."
> IMAGE.GET foo
> IMAGE.SCALE foo bar 1.2
> IMAGE.BW foo bar
```

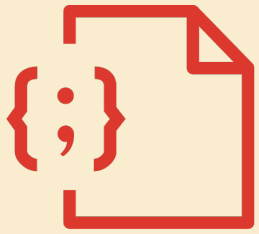
Loading & Unloading Modules

redis.conf

```
> MODULE LOAD /var/redis/modules/redis-image.so
> MODULE UNLOAD redis-image
> MODULE LIST
```

Once you add a module with types, you can't remove it.

Common Redis Modules



RedisJSON

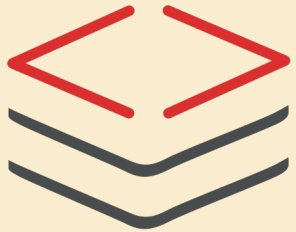
RediSearch

RedisGraph

RedisTimeSeries

RedisBloom

RedisGears



redis stack

DEMO

NEXT QUESTION

What does redis-cli return when you run the following commands:

```
LPUSH foo alfa bravo charlie
LRANGE foo 0 -1
```

A.
```
(error) ERR wrong number of arguments for 'LPUSH' command
```

B.
```
(error) ERR value is not an integer or out of range
```

C.
```
1) "alfa"
2) "bravo"
3) "charlie"
```

D.
```
1) "charlie"
2) "bravo"
3) "alfa"
```

Working With Lists

```
> LPUSH foo alfa
(integer) 1

> RPUSH foo bravo
(integer) 2

> LPUSH foo charlie delta
(integer) 4

> LPOP foo
"delta"

> RPOP foo
"bravo"

> LPOP foo 2
1) "charlie"
2) "alfa"
```

Lists as Queues & Stacks

```
> LPUSH foo alfa
> LPUSH foo bravo
> RPOP foo
```

```
LPUSH foo alfa
LPUSH foo bravo
LPOP foo
```

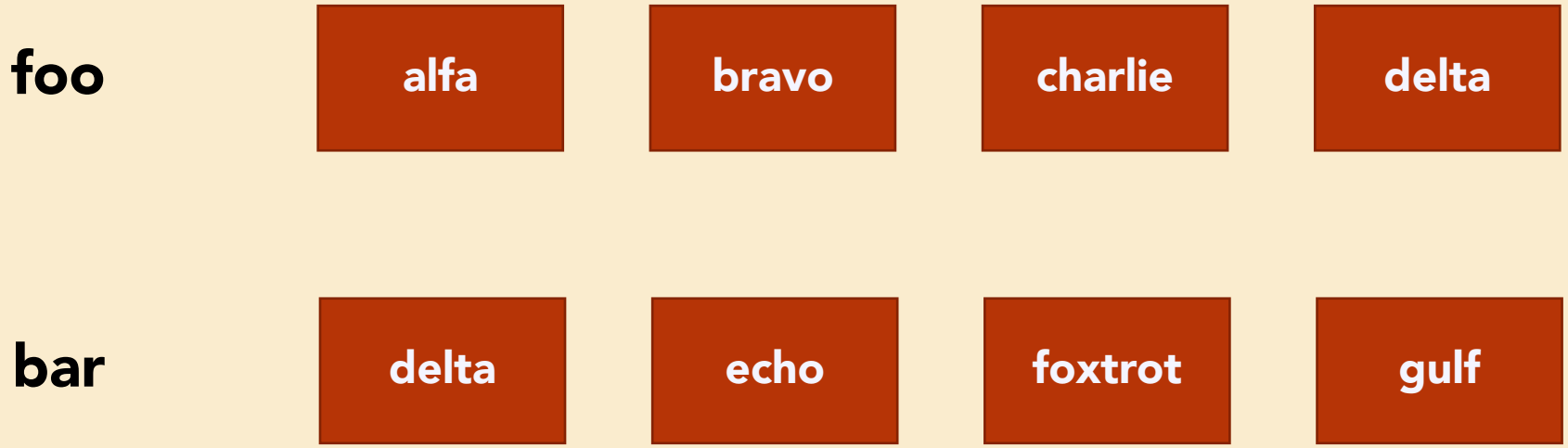
Querying Lists


```
> LRANGE foo 0 3
1) "alfa"
2) "bravo"
3) "charlie"
4) "delta"
```

```
> LRANGE foo 0 2
1) "alfa"
2) "bravo"
3) "charlie"
```

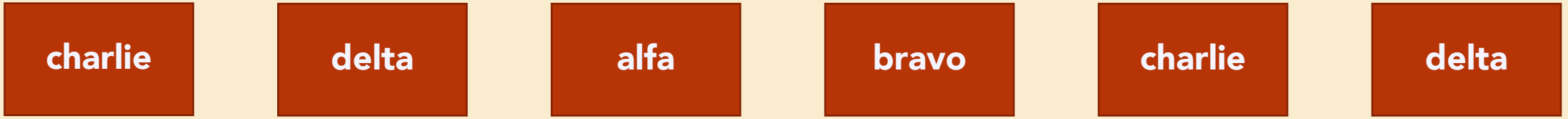
```
> LRANGE foo 0 -1
1) "alfa"
2) "bravo"
3) "charlie"
4) "delta"
```

Moving Between Lists




```
> LMOVE foo bar RIGHT LEFT
```

Circular Lists




```
> LMOVE foo foo RIGHT LEFT
> LMOVE foo foo RIGHT LEFT
```

DEMO

NEXT QUESTION

What do the following commands return:

```
ZADD units 2 alfa 3 bravo
ZADD tens 20 bravo 30 charlie
ZINTER 2 units tens WITHSCORES
```

A.
```
1) "bravo"
2) "3"
```

B.
```
1) "bravo"
2) "23"
```

C.
```
1) "bravo"
2) "20"
```

D.
```
1) "alfa"
2) "2"
3) "bravo"
4) "23"
5) "charlie"
6) "30"
```

Sorted Sets


Set
alfa
bravo
charlie
delta
Sorted Set
alfa – 1.2
bravo – 2.4
charlie – 3.6
delta – 4.8

You Can Mess with the Numbers

alfa – 1

bravo – 2

charlie – 3

delta – 4

```
> ZSCORE foo alfa
"1"

> ZMSCORE foo alfa bravo charlie
1) "1"
2) "2"
3) "3"

> ZINCRBY foo 1 alfa
"2"

> ZADD foo 42 alfa
(integer) 0
```

You Can Do Set Stuff With Them

Add
`ZADD foo 1.2 alfa 2.4 bravo`

Remove
`ZREM foo alfa`

Cardinality
`ZCARD foo`

Union
`ZUNION 2 foo bar`

Intersection
`ZINTER 2 foo bar`

Difference
`ZDIFF 2 foo bar`

But What Happens to the Score?

foo

- alfa – 1
- bravo – 2
- charlie – 3

bar

- charlie – 4
- delta – 5
- echo – 6

```
> ZUNION 2 foo bar WITHSCORES
 1) "alfa"
 2) "1"
 3) "bravo"
 4) "2"
 5) "delta"
 6) "5"
 7) "echo"
 8) "6"
 9) "charlie"
10) "7"
```

```
> ZUNION 2 foo bar AGGREGATE SUM WITHSCORES
> ZUNION 2 foo bar AGGREGATE MIN WITHSCORES
> ZUNION 2 foo bar AGGREGATE MAX WITHSCORES
```

And Querying Can Have and Order

alfa – 1

bravo – 2

charlie – 3

delta – 4

```
> ZRANK foo alfa
(integer) 4
> ZRANGE foo 1 2
1) "bravo"
2) "charlie"
> ZRANGE foo 10 100
(empty array)
> ZCOUNT foo 1 2
(integer) 2
> ZCOUNT foo 10 100
(integer) 0
```

DEMO

WE'LL BE RIGHT BACK

Guy's demos provided by

redisinsight

RedisInsight is the official Redis client of

SO YOU THINK YOU KNOW REDIS!

Download it today at redis.com/redisinsight.

WELCOME BACK

FINAL QUESTION

I've added the following line to redis.conf:

```
save 1000 1
```

What happens when I run the SAVE command?

A. It saves the contents of Redis to disk.

B. It saves the contents of Redis to disk if at least 1000 keys have changed and one second has passed.

C. It saves the contents of Redis to disk if at least 1 key has changed and 1000 ms have passed.

D. Nothing. Calling SAVE is ignored if save is configured.

Two Types of Persistence

RDB

redis.conf

```
save 3600 1
save 300 100
save 60 10000
```

```
> BGSAVE
```

AOF

redis.conf

```
appendonly yes
appendfsync always | everysec | no
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

```
> BGREWRITEAOF
```


You Won't
Lose Your
Data

THANKS FOR WATCHING!

Useful Links

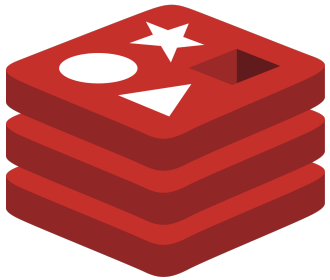
Redis Commands
https://redis.io/commands/

Redis Stack
https://redis.io/docs/stack/

Docker Image
https://hub.docker.com/r/redis/redis-stack

Image Credits
https://flickr.com/photos/8638651@N07/7333198566/
https://pixabay.com/photos/boxes-drawers-mailboxes-1834406/
https://pixabay.com/photos/crowd-concert-music-audience-4312230/
https://pixabay.com/photos/hamburger-p-french-fries-belly-2683042/
https://www.vecteezy.com/video/1797763-tv-show-background

Play the Home Version



Redis Cloud

https://redis.com/try-free/

Discord Server

https://discord.gg/redis

Redis Insight

https://redis.com/redisinsight

Demos and Slides

https://github.com/guyroyse/so-you-think-you-know-redis

Guy Royse

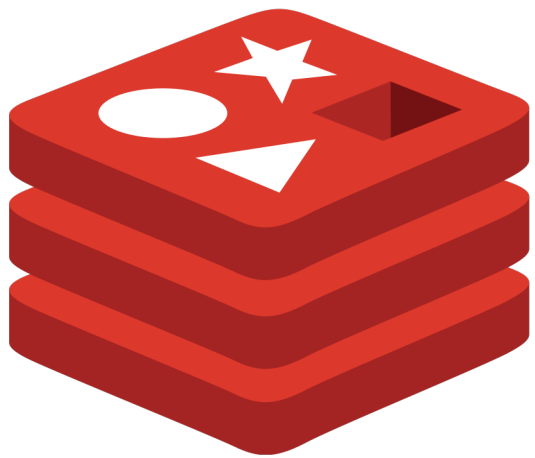
Developer Advocate

redis

@guyroyse

github.com/guyroyse

guy.dev



redis