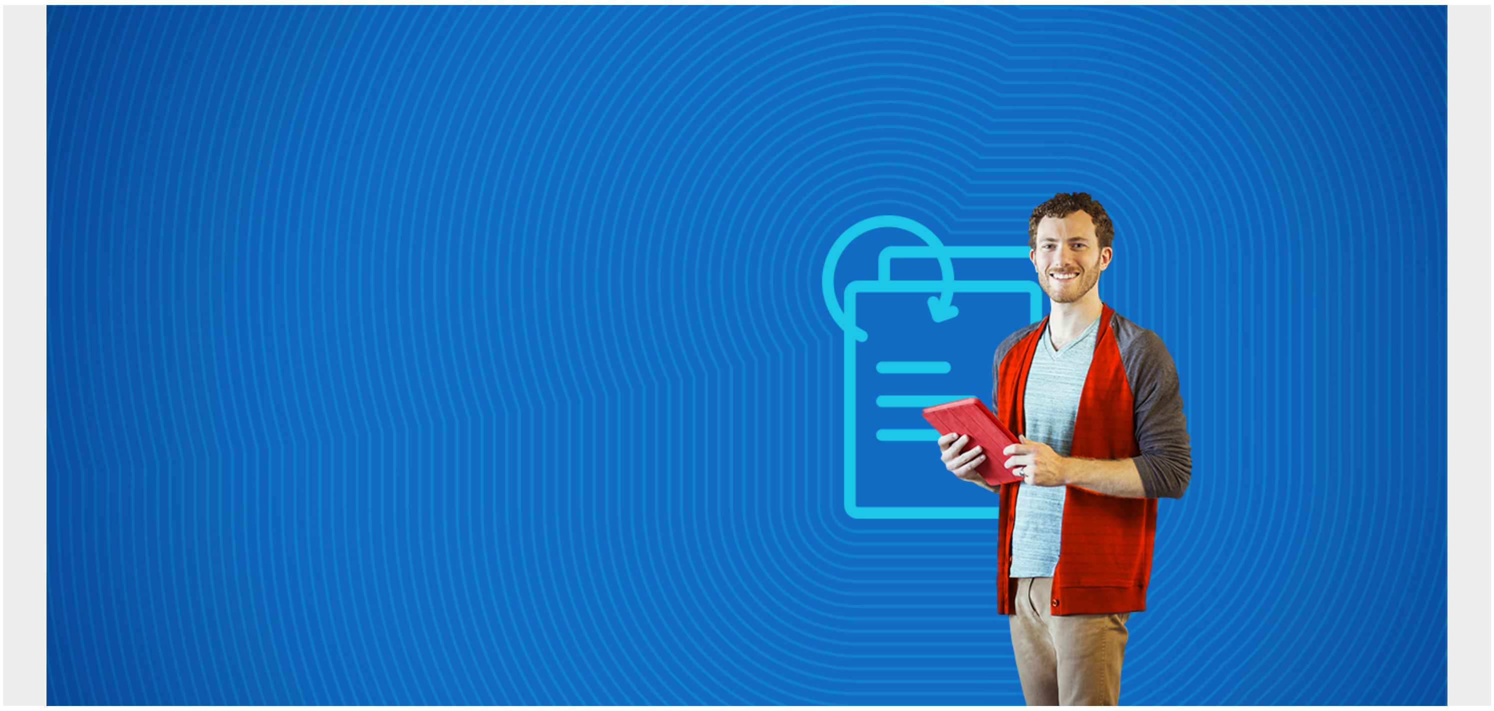
REDIS DATA TYPES



Redis has these data types:

- Binary-safe strings
- Lists
- Sets
- Sorted sets
- Hashes
- Bit arrays
- HyperLogLogs

Here, we briefly explain each of these.

(This tutorial is part of our Redis Guide. Use the right-hand menu to navigate.)

Redis keys

The first element of any Redis data structure is a **key**. One notable difference with Redis than other systems is that they you can set keys to expire after a certain amount of time.

The keys are binary safe, meaning you can use any kind of pattern of characters, even those that cannot be rendered on the screen, like a JPEG image.

The Redis style guide recommends to use colons (:) to give some meaning to keys like **sports:balls:tennisball**.

Strings

Strings are just strings. Strings are numbers too, like floats, meaning there is no separate float or integer data type.

```
set mystrng strivalue
OK
localhost:7001> get mystrng
"strivalue"
```

Sets

A **set** in any programming language is a list that does not allow duplicate values.

Notice below when we try to add the number 3 twice to the same set (use **sadd**) it ignores the second request.

```
localhost:7001> sadd integers 1
(integer) 1localhost:7001> sadd integers 2
(integer) 1localhost:7001> sadd integers 3
(integer) 1localhost:7001> sadd integers 3
(integer) 0
```

And you can make the intersection, unions, and difference of sets.

Sorted Sets

Redis also has **sorted sets**. You can control the order in which items are sorted by add a **score** when you add a value to a set. If you give them all the same score then the order is alphabetical order (to be specific **lexicographical** order).

Below we change the order of the alphabet by putting **c** between **a** and **b**.

```
localhost:7001> zadd alphabet 1 a
(integer) 1
localhost:7001> zadd alphabet 2 b
(integer) 1
localhost:7001> zadd alphabet 1.5 c
(integer) 1
```

Now list them.

```
zrange alphabet 0 -1
1) "a"
2) "c"
3) "b"
```

Lists

Lists in Redis are different than other languages, like Python. They are more like C or C++ linked lists.

This does not mean much for the programmer in terms of using them except with regards to performance. In a linked list, each element stores the address of the next element. To insert a value between two elements means to change what both of those point to. That operation is more efficient that shifting the whole list in memory.

You use **left push (lpush)** and **right push (rpush)** to add items to the end or beginning of a list. To initialize the list you can use lpush to let Redis know you are adding more that one item at a time. Otherwise it would think it was a string:

```
lpush mylist 1 2 3
(integer) 3localhost:7001> rpush mylist 4
(integer) 4
```

Use **right pop (rpop)** to pop (i.e. use and remove) the right-most element:

```
rpop mylist
"4"
```

Use the **range** command to list part of a list. We popped 4 off the end above so the relative address -1 (i.e., the end) is the last element in the list. 0 is the first element. So to list all elements:

```
lrange mylist 0 -1
1) "3"
2) "2"
3) "1"
```

You can use **ltrim** to change the range of a list, meaning move the pointer to a new offset and ignore the rest of the list. And use **linsert** to add values at a particular offset (i.e., position) in the list.

Lists are thread safe, meaning one process or programming operating on a list will make another process wait until it finishes. You can tell Redis to timeout when that situation occurs using the block list operations: **brpop** and **blpop**.

Hashes

A hash is a key value pair. It is called hash because the key is hashed for faster retrieval.

Write the set in multiple steps:

```
localhost:7001> hmset myset mykey myvalue
OKlocalhost:7001> hmset myset my2ndkey my2ndvalue
OK
```

Retrieve all the keys and values.

```
hgetall myset1) "mykey"
2) "myvalue"
3) "my2ndkey"
4) "my2ndvalue"
```

As with strings you can increment them when they are numbers:

```
hmset counter val 1
```

```
OKlocalhost:7001> hget counter val
"1"localhost:7001> hincrby counter val 1
(integer) 2localhost:7001> hget counter val
"2"
```

Bit Maps

A bitmap is not very useful unless you want to use the bit data structures to store some data as efficiently as possible.

To illustrate how bitmaps work in Redis we can change the number 6 to 7 using bitmap operation.

The number 6 is 00110110 in binary. We can change this to 7 by changing the last bit (offset 7) to 1.

```
set a 6get a
"6"setbit a 7 1get a
"7"
```

HyperLogLogs

This is a data structure used to count items without having to hold all of them in memory. Normally in order to could the number of elements in the set {1,2,3,4,5} you would have to keep all those in memory. The HyperLogLogs just makes that operation super efficient by taking a different approach.