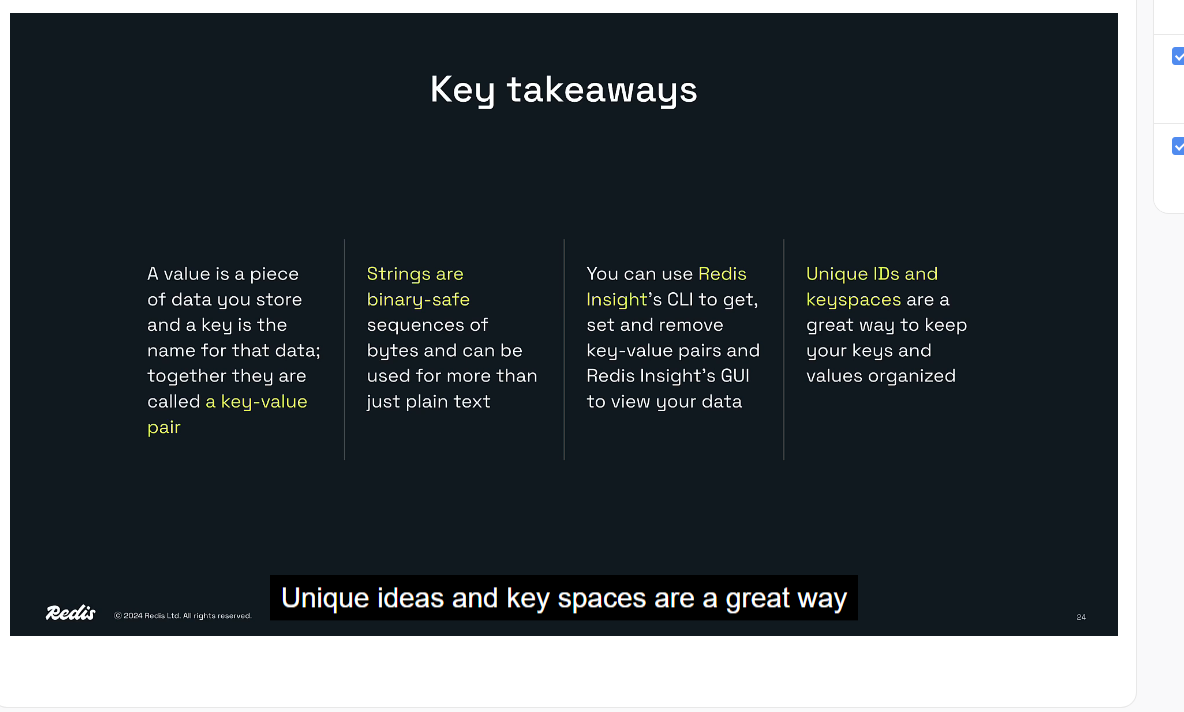


SET Key value

GET Key

UNLINk key – Delete



Redis strings are binary sequences. When you store a binary sequence, you typically do so with an encoding. Here are a few common encodings:

* UTF-8
* WAV
* JPEG

You can store data in any of these encodings -- and many more -- as Redis strings.

But what happens when you later retrieve the strings using redis-py? Do you get the data back in the original encoding?

No! That’s because neither Redis nor redis-py assume anything about the encoding of Redis strings. What you get back by default are binary sequences, which Python represents as "bytes" objects.

You must then decode these objects into Python strings using an encoding.

THE DECODE\_RESPONSES PARAMETER

redis-py’s Redis() initializer takes a decode\_responses parameter that defaults to False. If you don’t provide an argument for this parameter, redis-py behaves as described previously, returning bytes objects for Redis commands like GET that return binary-safe strings.

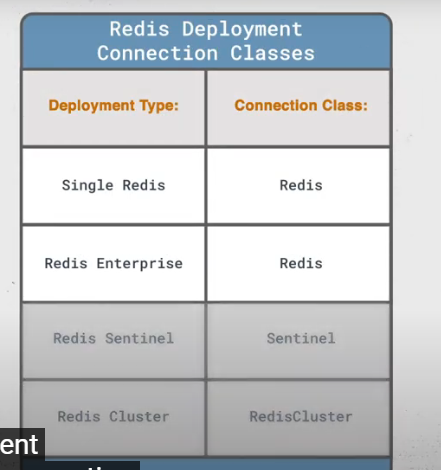
However, if you pass the argument True for this parameter, redis-py will decode all binary-safe strings using an encoding. The default is UTF-8, and you can specify a different encoding with the encoding parameter.

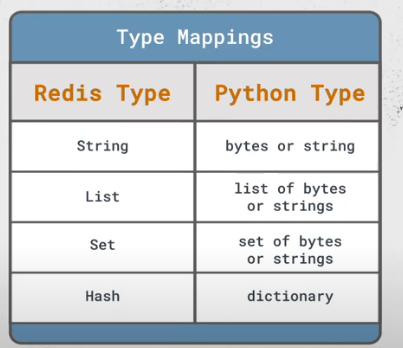
This is useful if you store string data in Redis with a single encoding. The example project for this course uses UTF-8, so all the Redis clients in the project use decode\_responses=True to decode strings with the UTF-8 encoding.

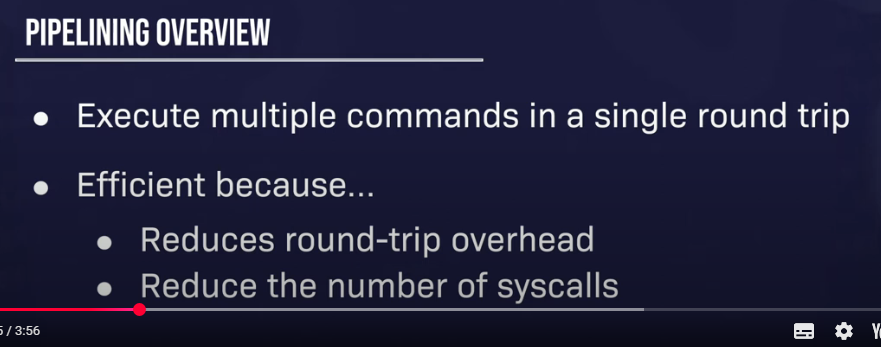
RESP – Redis Serialization Protocol it is implemented by redis client

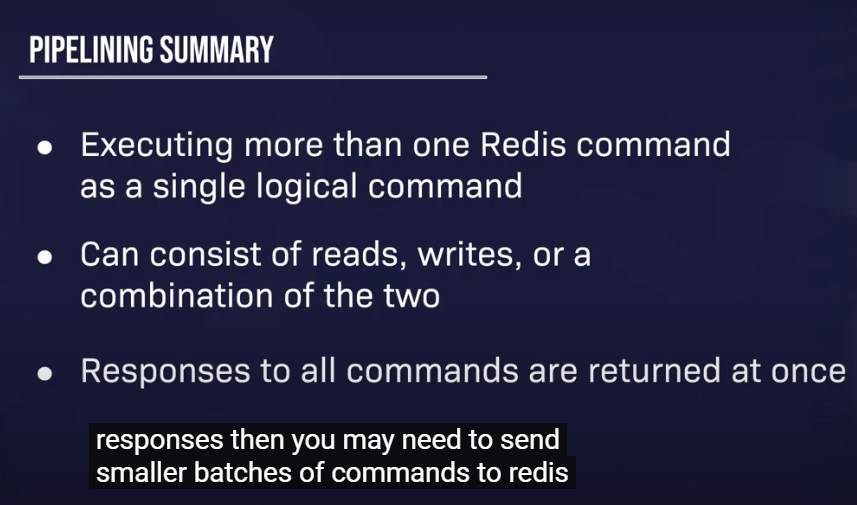
Redis Sentinel is a system designed to ensure high availability for Redis instances. Here are some key features and functionalities:

1. **Monitoring**: Sentinel continuously checks if your master and replica instances are working as expected.
2. **Notification**: It can notify system administrators or other programs if something goes wrong with one of the monitored Redis instances.
3. **Automatic Failover**: If a master instance fails, Sentinel can promote a replica to master and reconfigure other replicas to use the new master.
4. **Configuration Provider**: Sentinel acts as a source of authority for clients to discover the current Redis master









A **Sorted Set** in Redis, often abbreviated as **ZSET**, is a collection of unique strings (members) each associated with a score. These scores are used to order the members from the smallest to the largest score. If multiple members have the same score, they are ordered lexicographically

1

2

.

**Key Characteristics:**

* **Automatic Ordering**: Members are always sorted based on their score.
* **Unique Members**: No two members can be identical.
* **Efficiency**: Operations like adding, removing, or updating the score of an element have logarithmic complexity

2

.

**Common Use Cases:**

* **Leaderboards**: Easily maintain ordered lists of the highest scores in games.
* **Rate Limiters**: Build sliding-window rate limiters to prevent excessive API requests

1

.

**Example Commands:**

* **Adding Members**:
* ZADD myzset 1 "one"
* ZADD myzset 2 "two"
* ZADD myzset 3 "three"
* **Retrieving Members**:

ZRANGE myzset 0 -1 WITHSCORES

**1. Leaderboards**

In online gaming, Sorted Sets are perfect for maintaining leaderboards. Players' scores can be stored with their usernames as members and their scores as the associated scores. This allows for efficient retrieval of the top players:

ZADD game\_leaderboard 1500 "player1"

ZADD game\_leaderboard 2000 "player2"

ZADD game\_leaderboard 1800 "player3"

ZRANGE game\_leaderboard 0 -1 WITHSCORES

This command will return the players sorted by their scores

1

.

**2. Rate Limiting**

Sorted Sets can be used to implement rate limiting for APIs. By storing timestamps as scores and user IDs as members, you can efficiently track and limit the number of requests a user can make within a given time window:

ZADD api\_requests 1617181920 "user1"

ZADD api\_requests 1617181930 "user2"

ZREMRANGEBYSCORE api\_requests 0 1617181820

ZCOUNT api\_requests 1617181820 1617181920

This ensures that only requests within the desired time window are counted

2

.

**3. Task Scheduling**

Sorted Sets can be used to schedule tasks by storing the execution time as the score and the task details as the member. This allows for efficient retrieval of tasks that need to be executed next:

ZADD task\_schedule 1617181920 "task1"

ZADD task\_schedule 1617181930 "task2"

ZRANGEBYSCORE task\_schedule 0 1617181920

This retrieves tasks that are scheduled to be executed up to the current time

3

.

These examples illustrate how Redis Sorted Sets can be leveraged for efficient data management and real-time applications.