redis-py dev documentation

Search

v: stable

Redis Commands

Core Commands

The following functions can be used to replicate their equivalent Redis command. Generally they can be used as functions on your redis connection. For the simplest example, see below:

Getting and settings data in redis:

```
import redis
r = redis.Redis(decode_responses=True)
r.set('mykey', 'thevalueofmykey')
r.get('mykey')
```

class redis.commands.core.CoreCommands(*args, **kwargs) [source]

A class containing all of the implemented redis commands. This class is to be used as a mixin for synchronous Redis clients.

acl_cat(category=None, **kwargs)

Returns a list of categories or commands within a category.

If `category` is not supplied, returns a list of all categories. If `category` is supplied, returns a list of all commands within that category.

For more information see https://redis.io/commands/acl-cat

PARAMETERS

category (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_deluser(*username, **kwargs)

Delete the ACL for the specified ``username``s

For more information see https://redis.io/commands/acl-deluser

PARAMETERS

username (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_dryrun(username, *args, **kwargs)

Simulate the execution of a given command by a given `username`.

For more information see https://redis.io/commands/acl-dryrun

acl_genpass(bits=None, **kwargs)

Generate a random password value. If `bits` is supplied then use this number of bits, rounded to the next multiple of 4. See: https://redis.io/commands/acl-genpass

PARAMETERS

bits (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_getuser(username, **kwargs)

Get the ACL details for the specified `username`.

If `username` does not exist, return None

For more information see https://redis.io/commands/acl-getuser

PARAMETERS

username (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_help(**kwargs)

The ACL HELP command returns helpful text describing the different subcommands.

For more information see https://redis.io/commands/acl-help

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_list(**kwargs)

Return a list of all ACLs on the server

For more information see https://redis.io/commands/acl-list

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_load(**kwargs)

Load ACL rules from the configured `aclfile`.

Note that the server must be configured with the `aclfile` directive to be able to load ACL rules from an aclfile.

For more information see https://redis.io/commands/acl-load

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_log(count=None, **kwargs)

Get ACL logs as a list. :param int count: Get logs[0:count]. :rtype: List.

For more information see https://redis.io/commands/acl-log

PARAMETERS

count (`Optional`[`int`], default: `None`) –

acl_log_reset(**kwargs)

Reset ACL logs. :rtype: Boolean.

For more information see https://redis.io/commands/acl-log

acl_save(**kwargs)

Save ACL rules to the configured `aclfile`.

Note that the server must be configured with the `aclfile` directive to be able to save ACL rules to an aclfile.

For more information see https://redis.io/commands/acl-save

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_setuser(username, enabled=False, nopass=False, passwords=None, hashed_passwords=None, categories=None, commands=None, keys=None, channels=None, selectors=None, reset=False, reset_keys=False, reset_channels=False, reset_passwords=False, **kwargs)

Create or update an ACL user.

Create or update the ACL for `username`. If the user already exists, the existing ACL is completely overwritten and replaced with the specified values.

`enabled` is a boolean indicating whether the user should be allowed to authenticate or not. Defaults to `False`.

`nopass` is a boolean indicating whether the can authenticate without a password. This cannot be True if `passwords` are also specified.

`passwords` if specified is a list of plain text passwords to add to or remove from the user. Each password must be prefixed with a '+' to add or a '-' to remove. For convenience, the value of `passwords` can be a simple prefixed string when adding or removing a single password.

`hashed_passwords` if specified is a list of SHA-256 hashed passwords to add to or remove from the user. Each hashed password must be prefixed with a '+' to add or a '-' to remove. For convenience, the value of `hashed_passwords` can be a simple prefixed string when adding or removing a single password.

`categories` if specified is a list of strings representing category permissions. Each string must be prefixed with either a '+' to add the category permission or a '-' to remove the category permission.

`commands` if specified is a list of strings representing command permissions. Each string must be prefixed with either a '+' to add the command permission or a '-' to remove the command permission.

`keys` if specified is a list of key patterns to grant the user access to. Keys patterns allow '*' to support wildcard matching. For example, '*' grants access to all keys while 'cache:*' grants access to all keys that are prefixed with 'cache:'. `keys` should not be prefixed with a '~'.

`reset` is a boolean indicating whether the user should be fully reset prior to applying the new ACL. Setting this to True will remove all existing passwords, flags and privileges from the user and then apply the specified rules. If this is False, the user's existing passwords, flags and privileges will be kept and any new specified rules will be applied on top.

`reset_keys` is a boolean indicating whether the user's key permissions should be reset prior to applying any new key permissions specified in `keys`. If this is False, the user's existing key permissions will be kept and any new specified key permissions will be applied on top.

`reset_channels` is a boolean indicating whether the user's channel permissions should be reset prior to applying any new channel permissions specified in `channels`.If this is False, the user's existing channel permissions will be kept and any new specified channel permissions will be applied on top.

`reset_passwords` is a boolean indicating whether to remove all existing passwords and the 'nopass' flag from the user prior to applying any new passwords specified in 'passwords' or 'hashed_passwords'. If this is False, the user's existing passwords and 'nopass' status will be kept and any new specified passwords or hashed_passwords will be applied on top.

For more information see https://redis.io/commands/acl-setuser

PARAMETERS

- **username** (`str`) –
- **enabled** (`bool` , default: `False`) –
- **nopass** (`bool` , default: `False`) –
- **passwords** (`Union` [`str` , `Iterable` [`str`], `None`], default: `None`) –
- **hashed_passwords** (`Union` [`str` , `Iterable` [`str`], `None`], default: `None`) –
- **categories** (`Optional` [`Iterable` [`str`]], default: `None`) –
- **commands** (`Optional` [`Iterable` [`str`]], default: `None`) –
- **keys** (`Optional` [`Iterable` [`Union` [`bytes` , `str` , `memoryview`]]], default: `None`) –
- **channels** (`Optional` [`Iterable` [`Union` [`bytes` , `str` , `memoryview`]]], default: `None`) –
- **selectors** (`Optional` [`Iterable` [`Tuple` [`str` , `Union` [`bytes` , `str` , `memoryview`]]]], default: `None`) –
- **reset** (`bool` , default: `False`) –
- **reset_keys** (`bool` , default: `False`) –
- **reset_channels** (`bool` , default: `False`) –
- **reset_passwords** (`bool` , default: `False`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`acl_users(**kwargs)`

Returns a list of all registered users on the server.

For more information see https://redis.io/commands/acl-users

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`acl_whoami(**kwargs)`

Get the username for the current connection

For more information see https://redis.io/commands/acl-whoami

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`append(key, value)`

Appends the string `value` to the value at `key` . If `key` doesn’t already exist, create it with a value of `value` . Returns the new length of the value at `key` .

For more information see https://redis.io/commands/append

PARAMETERS

- **key** (`Union` [`bytes` , `str` , `memoryview`]) –
- **value** (`Union` [`bytes` , `memoryview` , `str` , `int` , `float`]) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`auth(password, username=None, **kwargs)`

Authenticates the user. If you do not pass username, Redis will try to authenticate for the “default” user. If you do pass username, it will authenticate for the given user. For more information see https://redis.io/commands/auth

PARAMETERS

- **password** (`str`) –
- **username** (`Optional` [`str`], default: `None`) –

`bgrewriteaof(**kwargs)`

Tell the Redis server to rewrite the AOF file from data in memory.

For more information see https://redis.io/commands/bgrewriteaof `bgsave(schedule=True, **kwargs)`

Tell the Redis server to save its data to disk. Unlike save(), this method is asynchronous and returns immediately.

For more information see https://redis.io/commands/bgsave

PARAMETERS
schedule (`bool`, default: `True`) –

RETURN TYPE
`Union[Awaitable, Any]`

`bitcount(key, start=None, end=None, mode=None)`

Returns the count of set bits in the value of `key`. Optional `start` and `end` parameters indicate which bytes to consider

For more information see https://redis.io/commands/bitcount

PARAMETERS
- **key** (`Union[bytes, str, memoryview]`) –
- **start** (`Optional[int]`, default: `None`) –
- **end** (`Optional[int]`, default: `None`) –
- **mode** (`Optional[str]`, default: `None`) –

RETURN TYPE
`Union[Awaitable, Any]`

`bitfield(key, default_overflow=None)`

Return a BitFieldOperation instance to conveniently construct one or more bitfield operations on `key`.

For more information see https://redis.io/commands/bitfield

PARAMETERS
- **self** (`Union[Redis, Redis]`) –
- **key** (`Union[bytes, str, memoryview]`) –
- **default_overflow** (`Optional[str]`, default: `None`) –

RETURN TYPE
`BitFieldOperation`

`bitfield_ro(key, encoding, offset, items=None)`

Return an array of the specified bitfield values where the first value is found using `encoding` and `offset` parameters and remaining values are result of corresponding encoding/offset pairs in optional list `items` Read-only variant of the BITFIELD command.

For more information see https://redis.io/commands/bitfield_ro

PARAMETERS
- **self** (`Union[Redis, Redis]`) –
- **key** (`Union[bytes, str, memoryview]`) –
- **encoding** (`str`) –
- **offset** (`Union[int, str]`) –
- **items** (`Optional[list]`, default: `None`) –

RETURN TYPE
`Union[Awaitable, Any]`

bitop(operation, dest, *keys)

Perform a bitwise operation using `operation` between `keys` and store the result in `dest`.

For more information see https://redis.io/commands/bitop

PARAMETERS

- **operation** (`str`) –
- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **keys** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

bitpos(key, bit, start=None, end=None, mode=None)

Return the position of the first bit set to 1 or 0 in a string. `start` and `end` defines search range. The range is interpreted as a range of bytes and not a range of bits, so start=0 and end=2 means to look at the first three bytes.

For more information see https://redis.io/commands/bitpos

PARAMETERS

- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **bit** (`int`) –
- **start** (`Optional`[`int`], default: `None`) –
- **end** (`Optional`[`int`], default: `None`) –
- **mode** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

blmove(first_list, second_list, timeout, src='LEFT', dest='RIGHT')

Blocking version of lmove.

For more information see https://redis.io/commands/blmove

PARAMETERS

- **first_list** (`str`) –
- **second_list** (`str`) –
- **timeout** (`int`) –
- **src** (`str`, default: `'LEFT'`) –
- **dest** (`str`, default: `'RIGHT'`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

blmpop(timeout, numkeys, *args, direction, count=1)

Pop `count` values (default 1) from first non-empty in the list of provided key names.

When all lists are empty this command blocks the connection until another client pushes to it or until the timeout, timeout of 0 blocks indefinitely

For more information see https://redis.io/commands/blmpop

PARAMETERS

- **timeout** (`float`) –
- **numkeys** (`int`) –
- **args** (`List`[`str`]) –
- **direction** (`str`) –
- **count** (`Optional`[`int`], default: `1`) –

RETURN TYPE

`Optional`[`list`]

blpop(keys, timeout=0)

LPOP a value off of the first non-empty list named in the `keys` list.

If none of the lists in `keys` has a value to LPOP, then block for `timeout` seconds, or until a value gets pushed on to one of the lists.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/blpop

PARAMETERS

- **keys** (`List`) –
- **timeout** (`Optional`[`int`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

brpop(keys, timeout=0)

RPOP a value off of the first non-empty list named in the `keys` list.

If none of the lists in `keys` has a value to RPOP, then block for `timeout` seconds, or until a value gets pushed on to one of the lists.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/brpop

PARAMETERS

- **keys** (`List`) –
- **timeout** (`Optional`[`int`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

brpoplpush(src, dst, timeout=0)

Pop a value off the tail of `src`, push it on the head of `dst` and then return it.

This command blocks until a value is in `src` or until `timeout` seconds elapse, whichever is first. A `timeout` value of 0 blocks forever.

For more information see https://redis.io/commands/brpoplpush

PARAMETERS

- **src** (`str`) –
- **dst** (`str`) –
- **timeout** (`Optional`[`int`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`Optional`[`str`]], `str`, `None`]

bzmpop(timeout, numkeys, keys, min=False, max=False, count=1)

Pop `count` values (default 1) off of the first non-empty sorted set named in the `keys` list.

If none of the sorted sets in `keys` has a value to pop, then block for `timeout` seconds, or until a member gets added to one of the sorted sets.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/bzmpop

PARAMETERS

- **timeout** (`float`) –
- **numkeys** (`int`) –
- **keys** (`List`[`str`]) –
- **min** (`Optional`[`bool`], default: `False`) –
- **max** (`Optional`[`bool`], default: `False`) –
- **count** (`Optional`[`int`], default: `1`) –

RETURN TYPE

`Optional`[`list`]

bzpopmax(keys, timeout=0)

ZPOPMAX a value off of the first non-empty sorted set named in the `keys` list.

If none of the sorted sets in `keys` has a value to ZPOPMAX, then block for `timeout` seconds, or until a member gets added to one of the sorted sets.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/bzpopmax

PARAMETERS

- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **timeout** (`Union`[`int`, `float`, `bytes`, `str`, `memoryview`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

bzpopmin(keys, timeout=0)

ZPOPMIN a value off of the first non-empty sorted set named in the `keys` list.

If none of the sorted sets in `keys` has a value to ZPOPMIN, then block for `timeout` seconds, or until a member gets added to one of the sorted sets.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/bzpopmin

PARAMETERS

- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **timeout** (`Union`[`int`, `float`, `bytes`, `str`, `memoryview`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

client_getname(kwargs)**

Returns the current connection name

For more information see https://redis.io/commands/client-getname

RETURN TYPE

`Union`[`Awaitable`, `Any`]

client_getredir(kwargs)**

Returns the ID (an integer) of the client to whom we are redirecting tracking notifications.

see: https://redis.io/commands/client-getredir

RETURN TYPE

`Union`[`Awaitable`, `Any`]

client_id(**kwargs)

Returns the current connection id

For more information see https://redis.io/commands/client-id

RETURN TYPE

`Union[Awaitable, Any]`

client_info(**kwargs)

Returns information and statistics about the current client connection.

For more information see https://redis.io/commands/client-info

RETURN TYPE

`Union[Awaitable, Any]`

client_kill(address, **kwargs)

Disconnects the client at `address` (ip:port)

For more information see https://redis.io/commands/client-kill

PARAMETERS

address (`str`) –

RETURN TYPE

`Union[Awaitable, Any]`

client_kill_filter(_id=None, _type=None, addr=None, skipme=None, laddr=None, user=None, **kwargs)

Disconnects client(s) using a variety of filter options :type _id: `Optional[str]`, default: `None` :param _id: Kills a client by its unique ID field :type _type: `Optional[str]`, default: `None` :param _type: Kills a client by type where type is one of 'normal', 'master', 'slave' or 'pubsub' :type addr: `Optional[str]`, default: `None` :param addr: Kills a client by its 'address:port' :type skipme: `Optional[bool]`, default: `None` :param skipme: If True, then the client calling the command will not get killed even if it is identified by one of the filter options. If skipme is not provided, the server defaults to skipme=True :type laddr: `Optional[bool]`, default: `None` :param laddr: Kills a client by its 'local (bind) address:port' :type user: `Optional[str]`, default: `None` :param user: Kills a client for a specific user name

RETURN TYPE

`Union[Awaitable, Any]`

client_list(_type=None, client_id=[], **kwargs)

Returns a list of currently connected clients. If type of client specified, only that type will be returned.

PARAMETERS

- **_type** (`Optional[str]`, default: `None`) – optional. one of the client types (normal, master, replica, pubsub)
- **client_id** (`List[Union[bytes, memoryview, str, int, float]]`, default: `[]`) – optional. a list of client ids

RETURN TYPE

`Union[Awaitable, Any]`

For more information see https://redis.io/commands/client-list

client_no_evict(mode)

Sets the client eviction mode for the current connection.

For more information see https://redis.io/commands/client-no-evict

PARAMETERS

mode (`str`) –

RETURN TYPE

`Union[Awaitable[str], str]`

`client_no_touch(mode)`

# The command controls whether commands sent by the client will alter # the LRU/LFU of the keys they access. # When turned on, the current client will not change LFU/LRU stats, # unless it sends the TOUCH command.

For more information see https://redis.io/commands/client-no-touch

PARAMETERS

mode (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`str`], `str`]

`client_pause(timeout, all=True, **kwargs)`

Suspend all the Redis clients for the specified amount of time.

For more information see https://redis.io/commands/client-pause

RETURN TYPE

`Union` [`Awaitable` , `Any`]

PARAMETERS

- **timeout** (`int`) – milliseconds to pause clients
- **all** (`bool` , default: `True`) – If true (default) all client commands are blocked.

otherwise, clients are only blocked if they attempt to execute a write command. For the WRITE mode, some commands have special behavior: EVAL/EVALSHA: Will block client for all scripts. PUBLISH: Will block client. PFCOUNT: Will block client. WAIT: Acknowledgments will be delayed, so this command will appear blocked.

`client_reply(reply, **kwargs)`

Enable and disable redis server replies.

`reply` Must be ON OFF or SKIP, ON - The default most with server replies to commands OFF - Disable server responses to commands SKIP - Skip the response of the immediately following command.

Note: When setting OFF or SKIP replies, you will need a client object with a timeout specified in seconds, and will need to catch the TimeoutError. The test_client_reply unit test illustrates this, and conftest.py has a client with a timeout.

See https://redis.io/commands/client-reply

PARAMETERS

reply (`Union` [`Literal` [`'ON'`], `Literal` [`'OFF'`], `Literal` [`'SKIP'`]]) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`client_setinfo(attr, value, **kwargs)`

Sets the current connection library name or version For mor information see https://redis.io/commands/client-setinfo

PARAMETERS

- **attr** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

client_setname(name, **kwargs)

Sets the current connection name

For more information see https://redis.io/commands/client-setname

Note

This method sets client name only for **current** connection.

If you want to set a common name for all connections managed by this client, use `client_name` constructor argument.

PARAMETERS

name (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

client_tracking(on=True, clientid=None, prefix=[], bcast=False, optin=False, optout=False, noloop=False, **kwargs)

Enables the tracking feature of the Redis server, that is used for server assisted client side caching.

`on` indicate for tracking on or tracking off. The dafualt is on.

`clientid` send invalidation messages to the connection with the specified ID.

`bcast` enable tracking in broadcasting mode. In this mode invalidation messages are reported for all the prefixes specified, regardless of the keys requested by the connection.

`optin` when broadcasting is NOT active, normally don't track keys in read only commands, unless they are called immediately after a CLIENT CACHING yes command.

`optout` when broadcasting is NOT active, normally track keys in read only commands, unless they are called immediately after a CLIENT CACHING no command.

`noloop` don't send notifications about keys modified by this connection itself.

`prefix` for broadcasting, register a given key prefix, so that notifications will be provided only for keys starting with this string.

See https://redis.io/commands/client-tracking

PARAMETERS

- **on** (`bool`, default: `True`) –
- **clientid** (`Optional`[`int`], default: `None`) –
- **prefix** (`Sequence`[`Union`[`bytes`, `str`, `memoryview`]], default: `[]`) –
- **bcast** (`bool`, default: `False`) –
- **optin** (`bool`, default: `False`) –
- **optout** (`bool`, default: `False`) –
- **noloop** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

client_tracking_off(clientid=None, prefix=[], bcast=False, optin=False, optout=False, noloop=False)

Turn off the tracking mode. For more information about the options look at client_tracking func.

See https://redis.io/commands/client-tracking

PARAMETERS

- **clientid** (`Optional[int]`, default: `None`) –
- **prefix** (`Sequence[Union[bytes, str, memoryview]]`, default: `[]`) –
- **bcast** (`bool`, default: `False`) –
- **optin** (`bool`, default: `False`) –
- **optout** (`bool`, default: `False`) –
- **noloop** (`bool`, default: `False`) –

RETURN TYPE

`Union[Awaitable, Any]`

client_tracking_on(clientid=None, prefix=[], bcast=False, optin=False, optout=False, noloop=False)

Turn on the tracking mode. For more information about the options look at client_tracking func.

See https://redis.io/commands/client-tracking

PARAMETERS

- **clientid** (`Optional[int]`, default: `None`) –
- **prefix** (`Sequence[Union[bytes, str, memoryview]]`, default: `[]`) –
- **bcast** (`bool`, default: `False`) –
- **optin** (`bool`, default: `False`) –
- **optout** (`bool`, default: `False`) –
- **noloop** (`bool`, default: `False`) –

RETURN TYPE

`Union[Awaitable, Any]`

client_trackinginfo(**kwargs)

Returns the information about the current client connection's use of the server assisted client side cache.

See https://redis.io/commands/client-trackinginfo

RETURN TYPE

`Union[Awaitable, Any]`

client_unblock(client_id, error=False, **kwargs)

Unblocks a connection by its client id. If `error` is True, unblocks the client with a special error message. If `error` is False (default), the client is unblocked using the regular timeout mechanism.

For more information see https://redis.io/commands/client-unblock

PARAMETERS

- **client_id** (`int`) –
- **error** (`bool`, default: `False`) –

RETURN TYPE

`Union[Awaitable, Any]`

client_unpause(**kwargs)

Unpause all redis clients

For more information see https://redis.io/commands/client-unpause

RETURN TYPE

`Union[Awaitable, Any]`

`command(**kwargs)`

Returns dict reply of details about all Redis commands.

For more information see https://redis.io/commands/command

`command_docs(*args)`

This function throws a NotImplementedError since it is intentionally not supported.

`command_getkeysandflags(*args)`

Returns array of keys from a full Redis command and their usage flags.

For more information see https://redis.io/commands/command-getkeysandflags

PARAMETERS

args (`List[str]`) –

RETURN TYPE

`List[Union[str, List[str]]]`

`command_list(module=None, category=None, pattern=None)`

Return an array of the server's command names. You can use one of the following filters: `module`: get the commands that belong to the module `category`: get the commands in the ACL category `pattern`: get the commands that match the given pattern

For more information see https://redis.io/commands/command-list/

PARAMETERS

- **module** (`Optional[str]`, default: `None`) –
- **category** (`Optional[str]`, default: `None`) –
- **pattern** (`Optional[str]`, default: `None`) –

RETURN TYPE

`Union[Awaitable, Any]`

`config_get(pattern='*', *args, **kwargs)`

Return a dictionary of configuration based on the `pattern`

For more information see https://redis.io/commands/config-get

PARAMETERS

- **pattern** (`Union[bytes, str, memoryview]`, default: `'*'`) –
- **args** (`List[Union[bytes, str, memoryview]]`) –

RETURN TYPE

`Union[Awaitable, Any]`

`config_resetstat(**kwargs)`

Reset runtime statistics

For more information see https://redis.io/commands/config-resetstat

RETURN TYPE

`Union[Awaitable, Any]`

`config_rewrite(**kwargs)`

Rewrite config file with the minimal change to reflect running config.

For more information see https://redis.io/commands/config-rewrite

RETURN TYPE

`Union[Awaitable, Any]`

`config_set(name, value, *args, **kwargs)`

Set config item `name` with `value`

For more information see https://redis.io/commands/config-set

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **args** (`List`[`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`copy(source, destination, destination_db=None, replace=False)`

Copy the value stored in the `source` key to the `destination` key.

`destination_db` an alternative destination database. By default, the `destination` key is created in the source Redis database.

`replace` whether the `destination` key should be removed before copying the value to it. By default, the value is not copied if the `destination` key already exists.

For more information see https://redis.io/commands/copy

PARAMETERS

- **source** (`str`) –
- **destination** (`str`) –
- **destination_db** (`Optional`[`str`], default: `None`) –
- **replace** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`dbsize(**kwargs)`

Returns the number of keys in the current database

For more information see https://redis.io/commands/dbsize

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`debug_object(key, **kwargs)`

Returns version specific meta information about a given key

For more information see https://redis.io/commands/debug-object

PARAMETERS

key (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`decr(name, amount=1)`

Decrements the value of `key` by `amount`. If no key exists, the value will be initialized as 0 - `amount`

For more information see https://redis.io/commands/decrby

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`int`, default: `1`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`decrby(name, amount=1)`

Decrements the value of `key` by `amount`. If no key exists, the value will be initialized as 0 - `amount`

For more information see https://redis.io/commands/decrby

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`int`, default: `1`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`delete(*names)`

Delete one or more keys specified by `names`

PARAMETERS

names (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`dump(name)`

Return a serialized version of the value stored at the specified key. If key does not exist a nil bulk reply is returned.

For more information see https://redis.io/commands/dump

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`echo(value, **kwargs)`

Echo the string back from the server

For more information see https://redis.io/commands/echo

PARAMETERS

value (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`eval(script, numkeys, *keys_and_args)`

Execute the Lua `script`, specifying the `numkeys` the script will touch and the key names and argument values in `keys_and_args`. Returns the result of the script.

In practice, use the object returned by `register_script`. This function exists purely for Redis API completion.

For more information see https://redis.io/commands/eval

PARAMETERS

- **script** (`str`) –
- **numkeys** (`int`) –
- **keys_and_args** (`list`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

eval_ro(script, numkeys, *keys_and_args)

The read-only variant of the EVAL command

Execute the read-only Lua `script` specifying the `numkeys` the script will touch and the key names and argument values in `keys_and_args`. Returns the result of the script.

For more information see https://redis.io/commands/eval_ro

PARAMETERS

- **script** (`str`) –
- **numkeys** (`int`) –
- **keys_and_args** (`list`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

evalsha(sha, numkeys, *keys_and_args)

Use the `sha` to execute a Lua script already registered via EVAL or SCRIPT LOAD. Specify the `numkeys` the script will touch and the key names and argument values in `keys_and_args`. Returns the result of the script.

In practice, use the object returned by `register_script`. This function exists purely for Redis API completion.

For more information see https://redis.io/commands/evalsha

PARAMETERS

- **sha** (`str`) –
- **numkeys** (`int`) –
- **keys_and_args** (`list`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

evalsha_ro(sha, numkeys, *keys_and_args)

The read-only variant of the EVALSHA command

Use the `sha` to execute a read-only Lua script already registered via EVAL or SCRIPT LOAD. Specify the `numkeys` the script will touch and the key names and argument values in `keys_and_args`. Returns the result of the script.

For more information see https://redis.io/commands/evalsha_ro

PARAMETERS

- **sha** (`str`) –
- **numkeys** (`int`) –
- **keys_and_args** (`list`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

exists(*names)

Returns the number of `names` that exist

For more information see https://redis.io/commands/exists

PARAMETERS

names (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

expire(name, time, nx=False, xx=False, gt=False, lt=False)

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

Valid options are:
NX -> Set expiry only when the key has no expiry XX -> Set expiry only when the key has an existing expiry GT -> Set expiry only when the new expiry is greater than current one LT -> Set expiry only when the new expiry is less than current one

For more information see https://redis.io/commands/expire

PARAMETERS
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **time** (`Union`[`int`, `timedelta`]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE
`Union`[`Awaitable`, `Any`]

expireat(name, when, nx=False, xx=False, gt=False, lt=False)

Set an expire flag on key `name` with given `option`. `when` can be represented as an integer indicating unix time or a Python datetime object.

Valid options are:
-> NX – Set expiry only when the key has no expiry -> XX – Set expiry only when the key has an existing expiry -> GT – Set expiry only when the new expiry is greater than current one -> LT – Set expiry only when the new expiry is less than current one

For more information see https://redis.io/commands/expireat

PARAMETERS
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **when** (`Union`[`int`, `datetime`]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE
`Union`[`Awaitable`, `Any`]

expiretime(key)

Returns the absolute Unix timestamp (since January 1, 1970) in seconds at which the given key will expire.

For more information see https://redis.io/commands/expiretime

PARAMETERS
key (`str`) –

RETURN TYPE
`int`

failover()

This function throws a NotImplementedError since it is intentionally not supported.

fcall(function, numkeys, *keys_and_args)

Invoke a function.

For more information see https://redis.io/commands/fcall

PARAMETERS

- **numkeys** (`int`) –
- **keys_and_args** (`Optional`[`List`]) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

fcall_ro(function, numkeys, *keys_and_args)

This is a read-only variant of the FCALL command that cannot execute commands that modify data.

For more information see https://redis.io/commands/fcal_ro

PARAMETERS

- **numkeys** (`int`) –
- **keys_and_args** (`Optional`[`List`]) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

flushall(asynchronous=False, **kwargs)

Delete all keys in all databases on the current host.

`asynchronous` indicates whether the operation is executed asynchronously by the server.

For more information see https://redis.io/commands/flushall

PARAMETERS

asynchronous (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

flushdb(asynchronous=False, **kwargs)

Delete all keys in the current database.

`asynchronous` indicates whether the operation is executed asynchronously by the server.

For more information see https://redis.io/commands/flushdb

PARAMETERS

asynchronous (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

function_delete(library)

Delete the library called `library` and all its functions.

For more information see https://redis.io/commands/function-delete

PARAMETERS

library (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

function_dump()

Return the serialized payload of loaded libraries.

For more information see https://redis.io/commands/function-dump

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

function_flush(mode='SYNC')

Deletes all the libraries.

For more information see https://redis.io/commands/function-flush

PARAMETERS

mode (`str`, default: `'SYNC'`) –

RETURN TYPE

`Union[Awaitable[str], str]`

function_kill()

Kill a function that is currently executing.

For more information see https://redis.io/commands/function-kill

RETURN TYPE

`Union[Awaitable[str], str]`

function_list(library='*', withcode=False)

Return information about the functions and libraries. :type library: `Optional[str]`, default: `'*'` :param library: pecify a pattern for matching library names :rtype: `Union[Awaitable[List], List]`

PARAMETERS

withcode (`Optional[bool]`, default: `False`) – cause the server to include the libraries source implementation in the reply

function_load(code, replace=False)

Load a library to Redis. :type code: `str` :param code: the source code (must start with Shebang statement that provides a metadata about the library) :type replace: `Optional[bool]`, default: `False` :param replace: changes the behavior to overwrite the existing library with the new contents. Return the library name that was loaded.

For more information see https://redis.io/commands/function-load

RETURN TYPE

`Union[Awaitable[str], str]`

function_restore(payload, policy='APPEND')

Restore libraries from the serialized `payload`. You can use the optional policy argument to provide a policy for handling existing libraries.

For more information see https://redis.io/commands/function-restore

PARAMETERS

- **payload** (`str`) –
- **policy** (`Optional[str]`, default: `'APPEND'`) –

RETURN TYPE

`Union[Awaitable[str], str]`

function_stats()

Return information about the function that's currently running and information about the available execution engines.

For more information see https://redis.io/commands/function-stats

RETURN TYPE

`Union[Awaitable[List], List]`

geoadd`(name, values, nx=False, xx=False, ch=False)`

Add the specified geospatial items to the specified key identified by the `name` argument. The Geospatial items are given as ordered members of the `values` argument, each item or place is formed by the triad longitude, latitude and name.

Note: You can use ZREM to remove elements.

`nx` forces ZADD to only create new elements and not to update scores for elements that already exist.

`xx` forces ZADD to only update scores of elements that already exist. New elements will not be added.

`ch` modifies the return value to be the numbers of elements changed. Changed elements include new elements that were added and elements whose scores changed.

For more information see https://redis.io/commands/geoadd

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Sequence`[`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **ch** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

geodist`(name, place1, place2, unit=None)`

Return the distance between `place1` and `place2` members of the `name` key. The units must be one of the following : m, km mi, ft. By default meters are used.

For more information see https://redis.io/commands/geodist

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **place1** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **place2** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **unit** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

geohash`(name, *values)`

Return the geo hash string for each item of `values` members of the specified key identified by the `name` argument.

For more information see https://redis.io/commands/geohash

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

geopos(name, *values)

Return the positions of each item of `values` as members of the specified key identified by the `name` argument. Each position is represented by the pairs lon and lat.

For more information see https://redis.io/commands/geopos

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

georadius(name, longitude, latitude, radius, unit=None, withdist=False, withcoord=False, withhash=False, count=None, sort=None, store=None, store_dist=None, any=False)

Return the members of the specified key identified by the `name` argument which are within the borders of the area specified with the `latitude` and `longitude` location and the maximum distance from the center specified by the `radius` value.

The units must be one of the following : m, km mi, ft. By default

`withdist` indicates to return the distances of each place.

`withcoord` indicates to return the latitude and longitude of each place.

`withhash` indicates to return the geohash string of each place.

`count` indicates to return the number of elements up to N.

`sort` indicates to return the places in a sorted way, ASC for nearest to fairest and DESC for fairest to nearest.

`store` indicates to save the places names in a sorted set named with a specific key, each element of the destination sorted set is populated with the score got from the original geo sorted set.

`store_dist` indicates to save the places names in a sorted set named with a specific key, instead of `store` the sorted set destination score is set with the distance.

For more information see https://redis.io/commands/georadius

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **longitude** (`float`) –
- **latitude** (`float`) –
- **radius** (`float`) –
- **unit** (`Optional`[`str`], default: `None`) –
- **withdist** (`bool`, default: `False`) –
- **withcoord** (`bool`, default: `False`) –
- **withhash** (`bool`, default: `False`) –
- **count** (`Optional`[`int`], default: `None`) –
- **sort** (`Optional`[`str`], default: `None`) –
- **store** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **store_dist** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **any** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

```
georadiusbymember(name, member, radius, unit=None, withdist=False,
    withcoord=False, withhash=False, count=None, sort=None, store=None,
    store_dist=None, any=False)
```

This command is exactly like `georadius` with the sole difference that instead of taking, as the center of the area to query, a longitude and latitude value, it takes the name of a member already existing inside the geospatial index represented by the sorted set.

For more information see https://redis.io/commands/georadiusbymember

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **member** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **radius** (`float`) –
- **unit** (`Optional`[`str`], default: `None`) –
- **withdist** (`bool`, default: `False`) –
- **withcoord** (`bool`, default: `False`) –
- **withhash** (`bool`, default: `False`) –
- **count** (`Optional`[`int`], default: `None`) –
- **sort** (`Optional`[`str`], default: `None`) –
- **store** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **store_dist** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **any** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`geosearch(name, member=None, longitude=None, latitude=None, unit='m', radius=None, width=None, height=None, sort=None, count=None, any=False, withcoord=False, withdist=False, withhash=False)`

Return the members of specified key identified by the `name` argument, which are within the borders of the area specified by a given shape. This command extends the GEORADIUS command, so in addition to searching within circular areas, it supports searching within rectangular areas.

This command should be used in place of the deprecated GEORADIUS and GEORADIUSBYMEMBER commands.

`member` Use the position of the given existing
member in the sorted set. Can't be given with `longitude` and `latitude`.

`longitude` and `latitude` Use the position given by this coordinates. Can't be given with `member` `radius` Similar to GEORADIUS, search inside circular area according the given radius. Can't be given with `height` and `width`. `height` and `width` Search inside an axis-aligned rectangle, determined by the given height and width. Can't be given with `radius`

`unit` must be one of the following : m, km, mi, ft. *m* for meters (the default value), *km* for kilometers, *mi* for miles and *ft* for feet.

`sort` indicates to return the places in a sorted way, ASC for nearest to furthest and DESC for furthest to nearest.

`count` limit the results to the first count matching items.

`any` is set to True, the command will return as soon as enough matches are found. Can't be provided without `count`

`withdist` indicates to return the distances of each place. `withcoord` indicates to return the latitude and longitude of each place.

`withhash` indicates to return the geohash string of each place.

For more information see https://redis.io/commands/geosearch

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **member** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`, `None`], default: `None`) –
- **longitude** (`Optional`[`float`], default: `None`) –
- **latitude** (`Optional`[`float`], default: `None`) –
- **unit** (`str`, default: `'m'`) –
- **radius** (`Optional`[`float`], default: `None`) –
- **width** (`Optional`[`float`], default: `None`) –
- **height** (`Optional`[`float`], default: `None`) –
- **sort** (`Optional`[`str`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **any** (`bool`, default: `False`) –
- **withcoord** (`bool`, default: `False`) –
- **withdist** (`bool`, default: `False`) –
- **withhash** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

geosearchstore(dest, name, member=None, longitude=None, latitude=None, unit='m', radius=None, width=None, height=None, sort=None, count=None, any=False, storedist=False)

This command is like GEOSEARCH, but stores the result in `dest`. By default, it stores the results in the destination sorted set with their geospatial information. if `store_dist` set to True, the command will stores the items in a sorted set populated with their distance from the center of the circle or box, as a floating-point number.

For more information see https://redis.io/commands/geosearchstore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **member** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`, `None`], default: `None`) –
- **longitude** (`Optional`[`float`], default: `None`) –
- **latitude** (`Optional`[`float`], default: `None`) –
- **unit** (`str`, default: `'m'`) –
- **radius** (`Optional`[`float`], default: `None`) –
- **width** (`Optional`[`float`], default: `None`) –
- **height** (`Optional`[`float`], default: `None`) –
- **sort** (`Optional`[`str`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **any** (`bool`, default: `False`) –
- **storedist** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

get(name)

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

For more information see https://redis.io/commands/get

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

getbit(name, offset)

Returns an integer indicating the value of `offset` in `name`

For more information see https://redis.io/commands/getbit

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **offset** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

getdel(name)

Get the value at key `name` and delete the key. This command is similar to GET, except for the fact that it also deletes the key on success (if and only if the key's value type is a string).

For more information see https://redis.io/commands/getdel

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

getex(name, ex=None, px=None, exat=None, pxat=None, persist=False)

Get the value of key and optionally set its expiration. GETEX is similar to GET, but is a write command with additional options. All time parameters can be given as datetime.timedelta or integers.

`ex` sets an expire flag on key `name` for `ex` seconds.

`px` sets an expire flag on key `name` for `px` milliseconds.

`exat` sets an expire flag on key `name` for `ex` seconds, specified in unix time.

`pxat` sets an expire flag on key `name` for `ex` milliseconds, specified in unix time.

`persist` remove the time to live associated with `name`.

For more information see https://redis.io/commands/getex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **ex** (`Union`[`int`, `timedelta`, `None`], default: `None`) –
- **px** (`Union`[`int`, `timedelta`, `None`], default: `None`) –
- **exat** (`Union`[`int`, `datetime`, `None`], default: `None`) –
- **pxat** (`Union`[`int`, `datetime`, `None`], default: `None`) –
- **persist** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

getrange(key, start, end)

Returns the substring of the string value stored at `key`, determined by the offsets `start` and `end` (both are inclusive)

For more information see https://redis.io/commands/getrange

PARAMETERS

- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

getset(name, value)

Sets the value at key `name` to `value` and returns the old value at key `name` atomically.

As per Redis 6.2, GETSET is considered deprecated. Please use SET with GET parameter in new code.

For more information see https://redis.io/commands/getset

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

hdel(name, *keys)

Delete `keys` from hash `name`

For more information see https://redis.io/commands/hdel

PARAMETERS

- **name** (`str`) –
- **keys** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

`hello()`

This function throws a NotImplementedError since it is intentionally not supported.

`hexists(name, key)`

Returns a boolean indicating if `key` exists within hash `name`

For more information see https://redis.io/commands/hexists

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`bool`], `bool`]

`hget(name, key)`

Return the value of `key` within the hash `name`

For more information see https://redis.io/commands/hget

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`Optional` [`str`]], `str` , `None`]

`hgetall(name)`

Return a Python dict of the hash's name/value pairs

For more information see https://redis.io/commands/hgetall

PARAMETERS

name (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`dict`], `dict`]

`hincrby(name, key, amount=1)`

Increment the value of `key` in hash `name` by `amount`

For more information see https://redis.io/commands/hincrby

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –
- **amount** (`int` , default: `1`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

`hincrbyfloat(name, key, amount=1.0)`

Increment the value of `key` in hash `name` by floating `amount`

For more information see https://redis.io/commands/hincrbyfloat

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –
- **amount** (`float` , default: `1.0`) –

RETURN TYPE

`Union` [`Awaitable` [`float`], `float`]

hkeys`(name)`

Return the list of keys within hash `name`

For more information see https://redis.io/commands/hkeys

PARAMETERS

name (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`List`], `List`]

hlen`(name)`

Return the number of elements in hash `name`

For more information see https://redis.io/commands/hlen

PARAMETERS

name (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

hmget`(name, keys, *args)`

Returns a list of values ordered identically to `keys`

For more information see https://redis.io/commands/hmget

PARAMETERS

- **name** (`str`) –
- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union` [`Awaitable` [`List`], `List`]

hmset`(name, mapping)`

Set key to value within hash `name` for each corresponding key and value from the `mapping` dict.

For more information see https://redis.io/commands/hmset

PARAMETERS

- **name** (`str`) –
- **mapping** (`dict`) –

RETURN TYPE

`Union` [`Awaitable` [`str`], `str`]

hrandfield`(key, count=None, withvalues=False)`

Return a random field from the hash value stored at key.

count: if the argument is positive, return an array of distinct fields. If called with a negative count, the behavior changes and the command is allowed to return the same field multiple times. In this case, the number of returned fields is the absolute value of the specified count. withvalues: The optional WITHVALUES modifier changes the reply so it includes the respective values of the randomly selected hash fields.

For more information see https://redis.io/commands/hrandfield

PARAMETERS

- **key** (`str`) –
- **count** (`Optional` [`int`], default: `None`) –
- **withvalues** (`bool` , default: `False`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

hscan(name, cursor=0, match=None, count=None)

Incrementally return key/value slices in a hash. Also return a cursor indicating the scan position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

For more information see https://redis.io/commands/hscan

PARAMETERS

- **name** (`Union` [`bytes` , `str` , `memoryview`]) –
- **cursor** (`int` , default: `0`) –
- **match** (`Union` [`bytes` , `str` , `memoryview` , `None`], default: `None`) –
- **count** (`Optional` [`int`], default: `None`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

hscan_iter(name, match=None, count=None)

Make an iterator using the HSCAN command so that the client doesn't need to remember the cursor position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

PARAMETERS

- **name** (`str`) –
- **match** (`Union` [`bytes` , `str` , `memoryview` , `None`], default: `None`) –
- **count** (`Optional` [`int`], default: `None`) –

RETURN TYPE

`Iterator`

hset(name, key=None, value=None, mapping=None, items=None)

Set `key` to `value` within hash `name` , `mapping` accepts a dict of key/value pairs that will be added to hash `name` . `items` accepts a list of key/value pairs that will be added to hash `name` . Returns the number of fields that were added.

For more information see https://redis.io/commands/hset

PARAMETERS

- **name** (`str`) –
- **key** (`Optional` [`str`], default: `None`) –
- **value** (`Optional` [`str`], default: `None`) –
- **mapping** (`Optional` [`dict`], default: `None`) –
- **items** (`Optional` [`list`], default: `None`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

hsetnx(name, key, value)

Set `key` to `value` within hash `name` if `key` does not exist. Returns 1 if HSETNX created a field, otherwise 0.

For more information see https://redis.io/commands/hsetnx

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`bool`], `bool`]

hstrlen(name, key)

Return the number of bytes stored in the value of `key` within hash `name`

For more information see https://redis.io/commands/hstrlen

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

hvals(name)

Return the list of values within hash `name`

For more information see https://redis.io/commands/hvals

PARAMETERS

name (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`List`], `List`]

incr(name, amount=1)

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

For more information see https://redis.io/commands/incrby

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`int`, default: `1`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

incrby(name, amount=1)

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

For more information see https://redis.io/commands/incrby

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`int`, default: `1`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

incrbyfloat(name, amount=1.0)

Increments the value at key `name` by floating `amount`. If no key exists, the value will be initialized as `amount`

For more information see https://redis.io/commands/incrbyfloat

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`float`, default: `1.0`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`info(section=None, *args, **kwargs)`

Returns a dictionary containing information about the Redis server

The `section` option can be used to select a specific section of information

The section option is not supported by older versions of Redis Server, and will generate ResponseError

For more information see https://redis.io/commands/info

PARAMETERS

- **section** (`Optional[str]`, default: `None`) –
- **args** (`List[str]`) –

RETURN TYPE

`Union[Awaitable, Any]`

`keys(pattern='*', **kwargs)`

Returns a list of keys matching `pattern`

For more information see https://redis.io/commands/keys

PARAMETERS

pattern (`Union[bytes, str, memoryview]`, default: `'*'`) –

RETURN TYPE

`Union[Awaitable, Any]`

`lastsave(**kwargs)`

Return a Python datetime object representing the last time the Redis database was saved to disk

For more information see https://redis.io/commands/lastsave

RETURN TYPE

`Union[Awaitable, Any]`

`latency_doctor()`

Raise a NotImplementedError, as the client will not support LATENCY DOCTOR. This funcion is best used within the redis-cli.

For more information see https://redis.io/commands/latency-doctor

`latency_graph()`

Raise a NotImplementedError, as the client will not support LATENCY GRAPH. This funcion is best used within the redis-cli.

For more information see https://redis.io/commands/latency-graph.

`latency_histogram(*args)`

This function throws a NotImplementedError since it is intentionally not supported.

`latency_history(event)`

Returns the raw data of the `event`'s latency spikes time series.

For more information see https://redis.io/commands/latency-history

PARAMETERS

event (`str`) –

RETURN TYPE

`Union[Awaitable, Any]`

`latency_latest()`

Reports the latest latency events logged.

For more information see https://redis.io/commands/latency-latest

RETURN TYPE

`Union[Awaitable, Any]`

`latency_reset(*events)`

Resets the latency spikes time series of all, or only some, events.

For more information see https://redis.io/commands/latency-reset

PARAMETERS

events (`str`) –

RETURN TYPE

`Union[Awaitable, Any]`

`lcs(key1, key2, len=False, idx=False, minmatchlen=0, withmatchlen=False)`

Find the longest common subsequence between `key1` and `key2`. If `len` is true the length of the match will will be returned. If `idx` is true the match position in each strings will be returned. `minmatchlen` restrict the list of matches to the ones of the given `minmatchlen`. If `withmatchlen` the length of the match also will be returned. For more information see https://redis.io/commands/lcs

PARAMETERS

- **key1** (`str`) –
- **key2** (`str`) –
- **len** (`Optional[bool]`, default: `False`) –
- **idx** (`Optional[bool]`, default: `False`) –
- **minmatchlen** (`Optional[int]`, default: `0`) –
- **withmatchlen** (`Optional[bool]`, default: `False`) –

RETURN TYPE

`Union[str, int, list]`

`lindex(name, index)`

Return the item from list `name` at position `index`

Negative indexes are supported and will return an item at the end of the list

For more information see https://redis.io/commands/lindex

PARAMETERS

- **name** (`str`) –
- **index** (`int`) –

RETURN TYPE

`Union[Awaitable[Optional[str]], str, None]`

`linsert(name, where, refvalue, value)`

Insert `value` in list `name` either immediately before or after [`where`] `refvalue`

Returns the new length of the list on success or -1 if `refvalue` is not in the list.

For more information see https://redis.io/commands/linsert

PARAMETERS

- **name** (`str`) –
- **where** (`str`) –
- **refvalue** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union[Awaitable[int], int]`

`llen(name)`

Return the length of the list `name`

For more information see https://redis.io/commands/llen

PARAMETERS

name (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

`lmove(first_list, second_list, src='LEFT', dest='RIGHT')`

Atomically returns and removes the first/last element of a list, pushing it as the first/last element on the destination list. Returns the element being popped and pushed.

For more information see https://redis.io/commands/lmove

PARAMETERS

- **first_list** (`str`) –
- **second_list** (`str`) –
- **src** (`str`, default: `'LEFT'`) –
- **dest** (`str`, default: `'RIGHT'`) –

RETURN TYPE

`Union` [`Awaitable`, `Any`]

`lmpop(num_keys, *args, direction, count=1)`

Pop `count` values (default 1) first non-empty list key from the list of args provided key names.

For more information see https://redis.io/commands/lmpop

PARAMETERS

- **num_keys** (`int`) –
- **args** (`List` [`str`]) –
- **direction** (`str`) –
- **count** (`Optional` [`int`], default: `1`) –

RETURN TYPE

`Union` [`Awaitable` [`list`], `list`]

`lolwut(*version_numbers, **kwargs)`

Get the Redis version and a piece of generative computer art

See: https://redis.io/commands/lolwut

PARAMETERS

version_numbers (`Union` [`str`, `float`]) –

RETURN TYPE

`Union` [`Awaitable`, `Any`]

`lpop(name, count=None)`

Removes and returns the first elements of the list `name`.

By default, the command pops a single element from the beginning of the list. When provided with the optional `count` argument, the reply will consist of up to count elements, depending on the list's length.

For more information see https://redis.io/commands/lpop

PARAMETERS

- **name** (`str`) –
- **count** (`Optional` [`int`], default: `None`) –

RETURN TYPE

`Union` [`Awaitable` [`Union` [`str`, `List`, `None`]], `str`, `List`, `None`]

lpos(name, value, rank=None, count=None, maxlen=None)

Get position of `value` within the list `name`

> If specified, `rank` indicates the “rank” of the first element to return in case there are multiple copies of `value` in the list. By default, LPOS returns the position of the first occurrence of `value` in the list. When `rank` 2, LPOS returns the position of the second `value` in the list. If `rank` is negative, LPOS searches the list in reverse. For example, -1 would return the position of the last occurrence of `value` and -2 would return the position of the next to last occurrence of `value`.
>
> If specified, `count` indicates that LPOS should return a list of up to `count` positions. A `count` of 2 would return a list of up to 2 positions. A `count` of 0 returns a list of all positions matching `value`. When `count` is specified and but `value` does not exist in the list, an empty list is returned.
>
> If specified, `maxlen` indicates the maximum number of list elements to scan. A `maxlen` of 1000 will only return the position(s) of items within the first 1000 entries in the list. A `maxlen` of 0 (the default) will scan the entire list.
>
> For more information see https://redis.io/commands/lpos

PARAMETERS

- **name** (`str`) –
- **value** (`str`) –
- **rank** (`Optional`[`int`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **maxlen** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`str`, `List`, `None`]

lpush(name, *values)

Push `values` onto the head of the list `name`

For more information see https://redis.io/commands/lpush

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

lpushx(name, *values)

Push `value` onto the head of the list `name` if `name` exists

For more information see https://redis.io/commands/lpushx

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

`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

For more information see https://redis.io/commands/lrange

PARAMETERS

- **name** (`str`) –
- **start** (`int`) –
- **end** (`int`) –

RETURN TYPE

`Union` [`Awaitable` [`list`], `list`]

`lrem(name, count, value)`

Remove the first `count` occurrences of elements equal to `value` from the list stored at `name`.

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.

For more information see https://redis.io/commands/lrem

PARAMETERS

- **name** (`str`) –
- **count** (`int`) –
- **value** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

`lset(name, index, value)`

Set element at `index` of list `name` to `value`

For more information see https://redis.io/commands/lset

PARAMETERS

- **name** (`str`) –
- **index** (`int`) –
- **value** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`str`], `str`]

`ltrim(name, start, end)`

Trim the list `name`, removing all values not within the slice between `start` and `end`

`start` and `end` can be negative numbers just like Python slicing notation

For more information see https://redis.io/commands/ltrim

PARAMETERS

- **name** (`str`) –
- **start** (`int`) –
- **end** (`int`) –

RETURN TYPE

`Union` [`Awaitable` [`str`], `str`]

`memory_malloc_stats(**kwargs)`

Return an internal statistics report from the memory allocator.

See: https://redis.io/commands/memory-malloc-stats

RETURN TYPE

`Union` [`Awaitable` , `Any`]

memory_purge(**kwargs)

Attempts to purge dirty pages for reclamation by allocator

For more information see https://redis.io/commands/memory-purge

RETURN TYPE

`Union[Awaitable, Any]`

memory_stats(**kwargs)

Return a dictionary of memory stats

For more information see https://redis.io/commands/memory-stats

RETURN TYPE

`Union[Awaitable, Any]`

memory_usage(key, samples=None, **kwargs)

Return the total memory usage for key, its value and associated administrative overheads.

For nested data structures, `samples` is the number of elements to sample. If left unspecified, the server's default is 5. Use 0 to sample all elements.

For more information see https://redis.io/commands/memory-usage

PARAMETERS

- **key** (`Union[bytes, str, memoryview]`) –
- **samples** (`Optional[int]`, default: `None`) –

RETURN TYPE

`Union[Awaitable, Any]`

mget(keys, *args)

Returns a list of values ordered identically to `keys`

For more information see https://redis.io/commands/mget

PARAMETERS

- **keys** (`Union[bytes, str, memoryview, Iterable[Union[bytes, str, memoryview]]]`) –
- **args** (`Union[bytes, memoryview, str, int, float]`) –

RETURN TYPE

`Union[Awaitable, Any]`

migrate(host, port, keys, destination_db, timeout, copy=False, replace=False, auth=None, **kwargs)

Migrate 1 or more keys from the current Redis server to a different server specified by the `host`, `port` and `destination_db`.

The `timeout`, specified in milliseconds, indicates the maximum time the connection between the two servers can be idle before the command is interrupted.

If `copy` is True, the specified `keys` are NOT deleted from the source server.

If `replace` is True, this operation will overwrite the keys on the destination server if they exist.

If `auth` is specified, authenticate to the destination server with the password provided.

For more information see https://redis.io/commands/migrate

PARAMETERS

- **host** (`str`) –
- **port** (`int`) –
- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **destination_db** (`int`) –
- **timeout** (`int`) –
- **copy** (`bool`, default: `False`) –
- **replace** (`bool`, default: `False`) –
- **auth** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

module_list()

Returns a list of dictionaries containing the name and version of all loaded modules.

For more information see https://redis.io/commands/module-list

RETURN TYPE

`Union`[`Awaitable`, `Any`]

module_load(path, *args)

Loads the module from `path`. Passes all `*args` to the module, during loading. Raises `ModuleError` if a module is not found at `path`.

For more information see https://redis.io/commands/module-load

RETURN TYPE

`Union`[`Awaitable`, `Any`]

module_loadex(path, options=None, args=None)

Loads a module from a dynamic library at runtime with configuration directives.

For more information see https://redis.io/commands/module-loadex

PARAMETERS

- **path** (`str`) –
- **options** (`Optional`[`List`[`str`]], default: `None`) –
- **args** (`Optional`[`List`[`str`]], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

module_unload(name)

Unloads the module `name`. Raises `ModuleError` if `name` is not in loaded modules.

For more information see https://redis.io/commands/module-unload

RETURN TYPE

`Union`[`Awaitable`, `Any`]

move(name, db)

Moves the key `name` to a different Redis database `db`

For more information see https://redis.io/commands/move

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **db** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

mset(mapping)

Sets key/values based on a mapping. Mapping is a dictionary of key/value pairs. Both keys and values should be strings or types that can be cast to a string via str().

For more information see https://redis.io/commands/mset

PARAMETERS

mapping (`Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

msetnx(mapping)

Sets key/values based on a mapping if none of the keys are already set. Mapping is a dictionary of key/value pairs. Both keys and values should be strings or types that can be cast to a string via str(). Returns a boolean indicating if the operation was successful.

For more information see https://redis.io/commands/msetnx

PARAMETERS

mapping (`Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

object(infotype, key, **kwargs)

Return the encoding, idletime, or refcount about the key

PARAMETERS

- **infotype** (`str`) –
- **key** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

persist(name)

Removes an expiration on `name`

For more information see https://redis.io/commands/persist

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pexpire(name, time, nx=False, xx=False, gt=False, lt=False)

Set an expire flag on key `name` for `time` milliseconds with given `option`. `time` can be represented by an integer or a Python timedelta object.

Valid options are:

NX -> Set expiry only when the key has no expiry XX -> Set expiry only when the key has an existing expiry GT -> Set expiry only when the new expiry is greater than current one LT -> Set expiry only when the new expiry is less than current one

For more information see https://redis.io/commands/pexpire

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **time** (`Union`[`int`, `timedelta`]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pexpireat(name, when, nx=False, xx=False, gt=False, lt=False)

Set an expire flag on key `name` with given `option`. `when` can be represented as an integer representing unix time in milliseconds (unix time * 1000) or a Python datetime object.

Valid options are:

NX -> Set expiry only when the key has no expiry XX -> Set expiry only when the key has an existing expiry GT -> Set expiry only when the new expiry is greater than current one LT -> Set expiry only when the new expiry is less than current one

For more information see https://redis.io/commands/pexpireat

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **when** (`Union`[`int`, `datetime`]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pexpiretime(key)

Returns the absolute Unix timestamp (since January 1, 1970) in milliseconds at which the given key will expire.

For more information see https://redis.io/commands/pexpiretime

PARAMETERS

key (`str`) –

RETURN TYPE

`int`

pfadd(name, *values)

Adds the specified elements to the specified HyperLogLog.

For more information see https://redis.io/commands/pfadd

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pfcount(*sources)

Return the approximated cardinality of the set observed by the HyperLogLog at key(s).

For more information see https://redis.io/commands/pfcount

PARAMETERS

sources (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pfmerge(dest, *sources)

Merge N different HyperLogLogs into a single one.

For more information see https://redis.io/commands/pfmerge

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **sources** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

ping(**kwargs)

Ping the Redis server

For more information see https://redis.io/commands/ping

RETURN TYPE

`Union`[`Awaitable`, `Any`]

psetex(name, time_ms, value)

Set the value of key `name` to `value` that expires in `time_ms` milliseconds. `time_ms` can be represented by an integer or a Python timedelta object

For more information see https://redis.io/commands/psetex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **time_ms** (`Union`[`int`, `timedelta`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

psync(replicationid, offset)

Initiates a replication stream from the master. Newer version for *sync*.

For more information see https://redis.io/commands/sync

PARAMETERS

- **replicationid** (`str`) –
- **offset** (`int`) –

pttl(name)

Returns the number of milliseconds until the key `name` will expire

For more information see https://redis.io/commands/pttl

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

publish(channel, message, **kwargs)

Publish `message` on `channel`. Returns the number of subscribers the message was delivered to.

For more information see https://redis.io/commands/publish

PARAMETERS

- **channel** (`Union`[`bytes`, `str`, `memoryview`]) –
- **message** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pubsub_channels(pattern='*', **kwargs)

Return a list of channels that have at least one subscriber

For more information see https://redis.io/commands/pubsub-channels

PARAMETERS

pattern (`Union`[`bytes`, `str`, `memoryview`], default: `'*'`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pubsub_numpat(**kwargs)

Returns the number of subscriptions to patterns

For more information see https://redis.io/commands/pubsub-numpat

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pubsub_numsub(*args, **kwargs)

Return a list of (channel, number of subscribers) tuples for each channel given in `*args`

For more information see https://redis.io/commands/pubsub-numsub

PARAMETERS

args (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pubsub_shardchannels(pattern='*', **kwargs)

Return a list of shard_channels that have at least one subscriber

For more information see https://redis.io/commands/pubsub-shardchannels

PARAMETERS

pattern (`Union`[`bytes`, `str`, `memoryview`], default: `'*'`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`pubsub_shardnumsub(*args, **kwargs)`

Return a list of (shard_channel, number of subscribers) tuples for each channel given in `*args`

For more information see https://redis.io/commands/pubsub-shardnumsub

PARAMETERS

args (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`quit(**kwargs)`

Ask the server to close the connection.

For more information see https://redis.io/commands/quit

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`randomkey(**kwargs)`

Returns the name of a random key

For more information see https://redis.io/commands/randomkey

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`readonly(**kwargs)`

Enables read queries for a connection to a Redis Cluster replica node.

For more information see https://redis.io/commands/readonly

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`readwrite(**kwargs)`

Disables read queries for a connection to a Redis Cluster slave node.

For more information see https://redis.io/commands/readwrite

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`register_script(script)`

Register a Lua `script` specifying the `keys` it will touch. Returns a Script object that is callable and hides the complexity of deal with scripts, keys, and shas. This is the preferred way to work with Lua scripts.

PARAMETERS

- **self** (`Redis`) –
- **script** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Script`

`rename(src, dst)`

Rename key `src` to `dst`

For more information see https://redis.io/commands/rename

PARAMETERS

- **src** (`Union`[`bytes`, `str`, `memoryview`]) –
- **dst** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

renamenx(src, dst)

Rename key `src` to `dst` if `dst` doesn't already exist

For more information see https://redis.io/commands/renamenx

PARAMETERS

- **src** (`Union`[`bytes`, `str`, `memoryview`]) –
- **dst** (`Union`[`bytes`, `str`, `memoryview`]) –

replicaof(*args, **kwargs)

Update the replication settings of a redis replica, on the fly.

Examples of valid arguments include:

NO ONE (set no replication) host port (set to the host and port of a redis server)

For more information see https://redis.io/commands/replicaof

RETURN TYPE

`Union`[`Awaitable`, `Any`]

reset()

Perform a full reset on the connection's server side contenxt.

See: https://redis.io/commands/reset

RETURN TYPE

`Union`[`Awaitable`, `Any`]

restore(name, ttl, value, replace=False, absttl=False, idletime=None, frequency=None)

Create a key using the provided serialized value, previously obtained using DUMP.

`replace` allows an existing key on `name` to be overridden. If it's not specified an error is raised on collision.

`absttl` if True, specified `ttl` should represent an absolute Unix timestamp in milliseconds in which the key will expire. (Redis 5.0 or greater).

`idletime` Used for eviction, this is the number of seconds the key must be idle, prior to execution.

`frequency` Used for eviction, this is the frequency counter of the object stored at the key, prior to execution.

For more information see https://redis.io/commands/restore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **ttl** (`float`) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **replace** (`bool`, default: `False`) –
- **absttl** (`bool`, default: `False`) –
- **idletime** (`Optional`[`int`], default: `None`) –
- **frequency** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

role()

Provide information on the role of a Redis instance in the context of replication, by returning if the instance is currently a master, slave, or sentinel.

For more information see https://redis.io/commands/role

RETURN TYPE

`Union`[`Awaitable`, `Any`]

rpop(name, count=None)

Removes and returns the last elements of the list `name`.

By default, the command pops a single element from the end of the list. When provided with the optional `count` argument, the reply will consist of up to count elements, depending on the list's length.

For more information see https://redis.io/commands/rpop

PARAMETERS

- **name** (`str`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`[`Union`[`str`, `List`, `None`]], `str`, `List`, `None`]

rpoplpush(src, dst)

RPOP a value off of the `src` list and atomically LPUSH it on to the `dst` list. Returns the value.

For more information see https://redis.io/commands/rpoplpush

PARAMETERS

- **src** (`str`) –
- **dst** (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

rpush(name, *values)

Push `values` onto the tail of the list `name`

For more information see https://redis.io/commands/rpush

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

rpushx(name, *values)

Push `value` onto the tail of the list `name` if `name` exists

For more information see https://redis.io/commands/rpushx

PARAMETERS

- **name** (`str`) –
- **values** (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

sadd(name, *values)

Add `value(s)` to set `name`

For more information see https://redis.io/commands/sadd

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

save(**kwargs)

Tell the Redis server to save its data to disk, blocking until the save is complete

For more information see https://redis.io/commands/save

RETURN TYPE

`Union[Awaitable, Any]`

scan(cursor=0, match=None, count=None, _type=None, **kwargs)

Incrementally return lists of key names. Also return a cursor indicating the scan position.

`match` allows for filtering the keys by pattern

`count` provides a hint to Redis about the number of keys to
return per batch.

`_type` filters the returned values by a particular Redis type.
Stock Redis instances allow for the following types: HASH, LIST, SET, STREAM, STRING, ZSET Additionally, Redis modules can expose other types as well.

For more information see https://redis.io/commands/scan

PARAMETERS

- **cursor** (`int`, default: `0`) –
- **match** (`Union[bytes, str, memoryview, None]`, default: `None`) –
- **count** (`Optional[int]`, default: `None`) –
- **_type** (`Optional[str]`, default: `None`) –

RETURN TYPE

`Union[Awaitable, Any]`

scan_iter(match=None, count=None, _type=None, **kwargs)

Make an iterator using the SCAN command so that the client doesn't need to remember the cursor position.

`match` allows for filtering the keys by pattern

`count` provides a hint to Redis about the number of keys to
return per batch.

`_type` filters the returned values by a particular Redis type.
Stock Redis instances allow for the following types: HASH, LIST, SET, STREAM, STRING, ZSET Additionally, Redis modules can expose other types as well.

PARAMETERS

- **match** (`Union[bytes, str, memoryview, None]`, default: `None`) –
- **count** (`Optional[int]`, default: `None`) –
- **_type** (`Optional[str]`, default: `None`) –

RETURN TYPE

`Iterator`

scard(name)

Return the number of elements in set `name`

For more information see https://redis.io/commands/scard

PARAMETERS

name (`str`) –

RETURN TYPE

`Union[Awaitable[int], int]`

script_exists(*args)

Check if a script exists in the script cache by specifying the SHAs of each script as `args`. Returns a list of boolean values indicating if if each already script exists in the cache.

For more information see https://redis.io/commands/script-exists

PARAMETERS

args (`str`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

script_flush(sync_type=None)

Flush all scripts from the script cache.

`sync_type` is by default SYNC (synchronous) but it can also be
ASYNC.

For more information see https://redis.io/commands/script-flush

PARAMETERS

sync_type (`Union` [`Literal` [`'SYNC'`], `Literal` [`'ASYNC'`], `None`], default: `None`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

script_kill()

Kill the currently executing Lua script

For more information see https://redis.io/commands/script-kill

RETURN TYPE

`Union` [`Awaitable` , `Any`]

script_load(script)

Load a Lua `script` into the script cache. Returns the SHA.

For more information see https://redis.io/commands/script-load

PARAMETERS

script (`Union` [`bytes` , `str` , `memoryview`]) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

sdiff(keys, *args)

Return the difference of sets specified by `keys`

For more information see https://redis.io/commands/sdiff

PARAMETERS

- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union` [`Awaitable` [`list`], `list`]

sdiffstore(dest, keys, *args)

Store the difference of sets specified by `keys` into a new set named `dest`. Returns the number of keys in the new set.

For more information see https://redis.io/commands/sdiffstore

PARAMETERS

- **dest** (`str`) –
- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

select(index, **kwargs)

Select the Redis logical database at index.

See: https://redis.io/commands/select

PARAMETERS

index (`int`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

set(name, value, ex=None, px=None, nx=False, xx=False, keepttl=False, get=False, exat=None, pxat=None)

Set the value at key `name` to `value`

`ex` sets an expire flag on key `name` for `ex` seconds.

`px` sets an expire flag on key `name` for `px` milliseconds.

`nx` if set to True, set the value at key `name` to `value` only
if it does not exist.

`xx` if set to True, set the value at key `name` to `value` only
if it already exists.

`keepttl` if True, retain the time to live associated with the key.
(Available since Redis 6.0)

`get` if True, set the value at key `name` to `value` and return
the old value stored at key, or None if the key did not exist. (Available since Redis 6.2)

`exat` sets an expire flag on key `name` for `ex` seconds,
specified in unix time.

`pxat` sets an expire flag on key `name` for `ex` milliseconds,
specified in unix time.

For more information see https://redis.io/commands/set

PARAMETERS

- **name** (`Union` [`bytes` , `str` , `memoryview`]) –
- **value** (`Union` [`bytes` , `memoryview` , `str` , `int` , `float`]) –
- **ex** (`Union` [`int` , `timedelta` , `None`], default: `None`) –
- **px** (`Union` [`int` , `timedelta` , `None`], default: `None`) –
- **nx** (`bool` , default: `False`) –
- **xx** (`bool` , default: `False`) –
- **keepttl** (`bool` , default: `False`) –
- **get** (`bool` , default: `False`) –
- **exat** (`Union` [`int` , `datetime` , `None`], default: `None`) –
- **pxat** (`Union` [`int` , `datetime` , `None`], default: `None`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

setbit(name, offset, value)

Flag the `offset` in `name` as `value` . Returns an integer indicating the previous value of `offset` .

For more information see https://redis.io/commands/setbit

PARAMETERS

- **name** (`Union` [`bytes` , `str` , `memoryview`]) –
- **offset** (`int`) –
- **value** (`int`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

setex(name, time, value)

Set the value of key `name` to `value` that expires in `time` seconds. `time` can be represented by an integer or a Python timedelta object.

For more information see https://redis.io/commands/setex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **time** (`Union`[`int`, `timedelta`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

setnx(name, value)

Set the value of key `name` to `value` if key doesn't exist

For more information see https://redis.io/commands/setnx

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

setrange(name, offset, value)

Overwrite bytes in the value of `name` starting at `offset` with `value`. If `offset` plus the length of `value` exceeds the length of the original value, the new value will be larger than before. If `offset` exceeds the length of the original value, null bytes will be used to pad between the end of the previous value and the start of what's being injected.

Returns the length of the new string.

For more information see https://redis.io/commands/setrange

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **offset** (`int`) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

shutdown(save=False, nosave=False, now=False, force=False, abort=False, **kwargs)

Shutdown the Redis server. If Redis has persistence configured, data will be flushed before shutdown. It is possible to specify modifiers to alter the behavior of the command: `save` will force a DB saving operation even if no save points are configured. `nosave` will prevent a DB saving operation even if one or more save points are configured. `now` skips waiting for lagging replicas, i.e. it bypasses the first step in the shutdown sequence. `force` ignores any errors that would normally prevent the server from exiting `abort` cancels an ongoing shutdown and cannot be combined with other flags.

For more information see https://redis.io/commands/shutdown

PARAMETERS

- **save** (`bool`, default: `False`) –
- **nosave** (`bool`, default: `False`) –
- **now** (`bool`, default: `False`) –
- **force** (`bool`, default: `False`) –
- **abort** (`bool`, default: `False`) –

RETURN TYPE

`None`

sinter(keys, *args)

Return the intersection of sets specified by `keys`

For more information see https://redis.io/commands/sinter

PARAMETERS

- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

sintercard(numkeys, keys, limit=0)

Return the cardinality of the intersect of multiple sets specified by ``keys`.

When LIMIT provided (defaults to 0 and means unlimited), if the intersection cardinality reaches limit partway through the computation, the algorithm will exit and yield limit as the cardinality

For more information see https://redis.io/commands/sintercard

PARAMETERS

- **numkeys** (`int`) –
- **keys** (`List`[`str`]) –
- **limit** (`int`, default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

sinterstore(dest, keys, *args)

Store the intersection of sets specified by `keys` into a new set named `dest`. Returns the number of keys in the new set.

For more information see https://redis.io/commands/sinterstore

PARAMETERS

- **dest** (`str`) –
- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

sismember(name, value)

Return whether `value` is a member of set `name`: - 1 if the value is a member of the set. - 0 if the value is not a member of the set or if key does not exist.

For more information see https://redis.io/commands/sismember

PARAMETERS

- **name** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`Union`[`Literal`[`0`], `Literal`[`1`]]], `Literal`[`0`], `Literal`[`1`]]

slaveof(host=None, port=None, **kwargs)

Set the server to be a replicated slave of the instance identified by the `host` and `port`. If called without arguments, the instance is promoted to a master instead.

For more information see https://redis.io/commands/slaveof

PARAMETERS

- **host** (`Optional`[`str`], default: `None`) –
- **port** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

slowlog_get(num=None, **kwargs)

Get the entries from the slowlog. If `num` is specified, get the most recent `num` items.

For more information see https://redis.io/commands/slowlog-get

PARAMETERS

num (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

slowlog_len(**kwargs)

Get the number of items in the slowlog

For more information see https://redis.io/commands/slowlog-len

RETURN TYPE

`Union`[`Awaitable`, `Any`]

slowlog_reset(**kwargs)

Remove all items in the slowlog

For more information see https://redis.io/commands/slowlog-reset

RETURN TYPE

`Union`[`Awaitable`, `Any`]

smembers(name)

Return all members of the set `name`

For more information see https://redis.io/commands/smembers

PARAMETERS

name (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`Set`], `Set`]

smismember(name, values, *args)

Return whether each value in `values` is a member of the set `name` as a list of `int` in the order of `values`: - 1 if the value is a member of the set. - 0 if the value is not a member of the set or if key does not exist.

For more information see https://redis.io/commands/smismember

PARAMETERS

- **name** (`str`) –
- **values** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`List`[`Union`[`Literal`[`0`], `Literal`[`1`]]]], `List`[`Union`[`Literal`[`0`], `Literal`[`1`]]]]

smove(src, dst, value)

Move `value` from set `src` to set `dst` atomically

For more information see https://redis.io/commands/smove

PARAMETERS

- **src** (`str`) –
- **dst** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`bool`], `bool`]

`sort(name, start=None, num=None, by=None, get=None, desc=False, alpha=False, store=None, groups=False)`

Sort and return the list, set or sorted set at `name`.

`start` and `num` allow for paging through the sorted data

`by` allows using an external key to weight and sort the items.
Use an "*" to indicate where in the key the item value is located

`get` allows for returning items from external keys rather than the
sorted data itself. Use an "*" to indicate where in the key the item value is located

`desc` allows for reversing the sort

`alpha` allows for sorting lexicographically rather than numerically

`store` allows for storing the result of the sort into
the key `store`

`groups` if set to True and if `get` contains at least two
elements, sort will return a list of tuples, each containing the values fetched from the arguments to `get`.

For more information see https://redis.io/commands/sort

PARAMETERS

- **name** (`str`) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –
- **by** (`Optional`[`str`], default: `None`) –
- **get** (`Optional`[`List`[`str`]], default: `None`) –
- **desc** (`bool`, default: `False`) –
- **alpha** (`bool`, default: `False`) –
- **store** (`Optional`[`str`], default: `None`) –
- **groups** (`Optional`[`bool`], default: `False`) –

RETURN TYPE

`Union`[`List`, `int`]

`sort_ro(key, start=None, num=None, by=None, get=None, desc=False, alpha=False)`

Returns the elements contained in the list, set or sorted set at key. (read-only variant of the SORT command)

`start` and `num` allow for paging through the sorted data

`by` allows using an external key to weight and sort the items.
Use an "*" to indicate where in the key the item value is located

`get` allows for returning items from external keys rather than the
sorted data itself. Use an "*" to indicate where in the key the item value is located

`desc` allows for reversing the sort

`alpha` allows for sorting lexicographically rather than numerically

For more information see https://redis.io/commands/sort_ro

PARAMETERS

- **key** (`str`) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –
- **by** (`Optional`[`str`], default: `None`) –
- **get** (`Optional`[`List`[`str`]], default: `None`) –
- **desc** (`bool`, default: `False`) –
- **alpha** (`bool`, default: `False`) –

RETURN TYPE

`list`

`spop(name, count=None)`

Remove and return a random member of set `name`

For more information see https://redis.io/commands/spop

PARAMETERS

- **name** (`str`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`str`, `List`, `None`]

`spublish(shard_channel, message)`

Posts a message to the given shard channel. Returns the number of clients that received the message

For more information see https://redis.io/commands/spublish

PARAMETERS

- **shard_channel** (`Union`[`bytes`, `str`, `memoryview`]) –
- **message** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

srandmember(name, number=None)

If `number` is None, returns a random member of set `name`.

If `number` is supplied, returns a list of `number` random members of set `name`. Note this is only available when running Redis 2.6+.

For more information see https://redis.io/commands/srandmember

PARAMETERS

- **name** (`str`) –
- **number** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`str`, `List`, `None`]

srem(name, *values)

Remove `values` from set `name`

For more information see https://redis.io/commands/srem

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

sscan(name, cursor=0, match=None, count=None)

Incrementally return lists of elements in a set. Also return a cursor indicating the scan position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

For more information see https://redis.io/commands/sscan

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **cursor** (`int`, default: `0`) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

sscan_iter(name, match=None, count=None)

Make an iterator using the SSCAN command so that the client doesn't need to remember the cursor position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Iterator`

stralgo(algo, value1, value2, specific_argument='strings', len=False, idx=False, minmatchlen=None, withmatchlen=False, **kwargs)

Implements complex algorithms that operate on strings. Right now the only algorithm implemented is the LCS algorithm (longest common substring). However new algorithms could be implemented in the future.

`algo` Right now must be LCS `value1` and `value2` Can be two strings or two keys `specific_argument` Specifying if the arguments to the algorithm will be keys or strings. strings is the default. `len` Returns just the len of the match. `idx` Returns the match positions in each string. `minmatchlen` Restrict the list of matches to the ones of a given minimal length. Can be provided only when `idx` set to True. `withmatchlen` Returns the matches with the len of the match. Can be provided only when `idx` set to True.

For more information see https://redis.io/commands/stralgo

PARAMETERS

- **algo** (`Literal`[`'LCS'`]) –
- **value1** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value2** (`Union`[`bytes`, `str`, `memoryview`]) –
- **specific_argument** (`Union`[`Literal`[`'strings'`], `Literal`[`'keys'`]], default: `'strings'`) –
- **len** (`bool`, default: `False`) –
- **idx** (`bool`, default: `False`) –
- **minmatchlen** (`Optional`[`int`], default: `None`) –
- **withmatchlen** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

strlen(name)

Return the number of bytes stored in the value of `name`

For more information see https://redis.io/commands/strlen

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

substr(name, start, end=-1)

Return a substring of the string at key `name`. `start` and `end` are 0-based integers specifying the portion of the string to return.

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`, default: `-1`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

sunion(keys, *args)

Return the union of sets specified by `keys`

For more information see https://redis.io/commands/sunion

PARAMETERS

- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`List`], `List`]

sunionstore`(dest, keys, *args)`

Store the union of sets specified by `keys` into a new set named `dest`. Returns the number of keys in the new set.

For more information see https://redis.io/commands/sunionstore

PARAMETERS

- **dest** (`str`) –
- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

swapdb`(first, second, **kwargs)`

Swap two databases

For more information see https://redis.io/commands/swapdb

PARAMETERS

- **first** (`int`) –
- **second** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

sync`()`

Initiates a replication stream from the master.

For more information see https://redis.io/commands/sync

RETURN TYPE

`Union`[`Awaitable`, `Any`]

tfcall`(lib_name, func_name, keys=None, *args)`

Invoke a function.

`lib_name` - the library name contains the function. `func_name` - the function name to run. `keys` - the keys that will be touched by the function. `args` - Additional argument to pass to the function.

For more information see https://redis.io/commands/tfcall/

PARAMETERS

- **lib_name** (`str`) –
- **func_name** (`str`) –
- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]], `None`], default: `None`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

tfcall_async(lib_name, func_name, keys=None, *args)

Invoke an async function (coroutine).

`lib_name` - the library name contains the function. `func_name` - the function name to run. `keys` - the keys that will be touched by the function. `args` - Additional argument to pass to the function.

For more information see https://redis.io/commands/tfcall/

PARAMETERS

- **lib_name** (`str`) –
- **func_name** (`str`) –
- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]], `None`], default: `None`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

tfunction_delete(lib_name)

Delete a library from RedisGears.

`lib_name` the library name to delete.

For more information see https://redis.io/commands/tfunction-delete/

PARAMETERS

lib_name (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

tfunction_list(with_code=False, verbose=0, lib_name=None)

List the functions with additional information about each function.

`with_code` Show libraries code. `verbose` output verbosity level, higher number will increase verbosity level `lib_name` specifying a library name (can be used multiple times to show multiple libraries in a single command) # noqa

For more information see https://redis.io/commands/tfunction-list/

PARAMETERS

- **with_code** (`bool`, default: `False`) –
- **verbose** (`int`, default: `0`) –
- **lib_name** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

tfunction_load(lib_code, replace=False, config=None)

Load a new library to RedisGears.

`lib_code` - the library code. `config` - a string representation of a JSON object that will be provided to the library on load time, for more information refer to https://github.com/RedisGears/RedisGears/blob/master/docs/function_advance_topics.md#library-configuration `replace` - an optional argument, instructs RedisGears to replace the function if its already exists

For more information see https://redis.io/commands/tfunction-load/

PARAMETERS

- **lib_code** (`str`) –
- **replace** (`bool`, default: `False`) –
- **config** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

time(**kwargs)

Returns the server time as a 2-item tuple of ints: (seconds since epoch, microseconds into this second).

For more information see https://redis.io/commands/time

RETURN TYPE

`Union[Awaitable, Any]`

touch(*args)

Alters the last access time of a key(s) `*args`. A key is ignored if it does not exist.

For more information see https://redis.io/commands/touch

PARAMETERS

args (`Union[bytes, str, memoryview]`) –

RETURN TYPE

`Union[Awaitable, Any]`

ttl(name)

Returns the number of seconds until the key `name` will expire

For more information see https://redis.io/commands/ttl

PARAMETERS

name (`Union[bytes, str, memoryview]`) –

RETURN TYPE

`Union[Awaitable, Any]`

type(name)

Returns the type of key `name`

For more information see https://redis.io/commands/type

PARAMETERS

name (`Union[bytes, str, memoryview]`) –

RETURN TYPE

`Union[Awaitable, Any]`

unlink(*names)

Unlink one or more keys specified by `names`

For more information see https://redis.io/commands/unlink

PARAMETERS

names (`Union[bytes, str, memoryview]`) –

RETURN TYPE

`Union[Awaitable, Any]`

unwatch()

Unwatches the value at key `name`, or None of the key doesn't exist

For more information see https://redis.io/commands/unwatch

RETURN TYPE

`None`

wait(num_replicas, timeout, **kwargs)

Redis synchronous replication That returns the number of replicas that processed the query when we finally have at least `num_replicas`, or when the `timeout` was reached.

For more information see https://redis.io/commands/wait

PARAMETERS

- **num_replicas** (`int`) –
- **timeout** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

waitaof(num_local, num_replicas, timeout, **kwargs)

This command blocks the current client until all previous write commands by that client are acknowledged as having been fsynced to the AOF of the local Redis and/or at least the specified number of replicas.

For more information see https://redis.io/commands/waitaof

PARAMETERS

- **num_local** (`int`) –
- **num_replicas** (`int`) –
- **timeout** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

watch(*names)

Watches the values at keys `names`, or None if the key doesn't exist

For more information see https://redis.io/commands/watch

PARAMETERS

names (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`None`

xack(name, groupname, *ids)

Acknowledges the successful processing of one or more messages. name: name of the stream. groupname: name of the consumer group. *ids: message ids to acknowledge.

For more information see https://redis.io/commands/xack

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **ids** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xadd(name, fields, id='*', maxlen=None, approximate=True, nomkstream=False, minid=None, limit=None)

Add to a stream. name: name of the stream fields: dict of field/value pairs to insert into the stream id: Location to insert this record. By default it is appended. maxlen: truncate old stream members beyond this size. Can't be specified with minid. approximate: actual stream length may be slightly more than maxlen nomkstream: When set to true, do not make a stream minid: the minimum id in the stream to query. Can't be specified with maxlen. limit: specifies the maximum number of entries to retrieve

For more information see https://redis.io/commands/xadd

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **fields** (`Dict`[`Union`[`bytes`, `memoryview`, `str`, `int`, `float`], `Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –
- **id** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'*'`) –
- **maxlen** (`Optional`[`int`], default: `None`) –
- **approximate** (`bool`, default: `True`) –
- **nomkstream** (`bool`, default: `False`) –
- **minid** (`Union`[`int`, `bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **limit** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xautoclaim(name, groupname, consumername, min_idle_time, start_id='0-0', count=None, justid=False)

Transfers ownership of pending stream entries that match the specified criteria. Conceptually, equivalent to calling XPENDING and then XCLAIM, but provides a more straightforward way to deal with message delivery failures via SCAN-like semantics. name: name of the stream. groupname: name of the consumer group. consumername: name of a consumer that claims the message. min_idle_time: filter messages that were idle less than this amount of milliseconds. start_id: filter messages with equal or greater ID. count: optional integer, upper limit of the number of entries that the command attempts to claim. Set to 100 by default. justid: optional boolean, false by default. Return just an array of IDs of messages successfully claimed, without returning the actual message

For more information see https://redis.io/commands/xautoclaim

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min_idle_time** (`int`) –
- **start_id** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'0-0'`) –
- **count** (`Optional`[`int`], default: `None`) –
- **justid** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xclaim(name, groupname, consumername, min_idle_time, message_ids, idle=None, time=None, retrycount=None, force=False, justid=False)`

Changes the ownership of a pending message.

name: name of the stream.

groupname: name of the consumer group.

consumername: name of a consumer that claims the message.

min_idle_time: filter messages that were idle less than this amount of milliseconds

message_ids: non-empty list or tuple of message IDs to claim

idle: optional. Set the idle time (last time it was delivered) of the message in ms

time: optional integer. This is the same as idle but instead of a relative amount of milliseconds, it sets the idle time to a specific Unix time (in milliseconds).

retrycount: optional integer. set the retry counter to the specified value. This counter is incremented every time a message is delivered again.

force: optional boolean, false by default. Creates the pending message entry in the PEL even if certain specified IDs are not already in the PEL assigned to a different client.

justid: optional boolean, false by default. Return just an array of IDs of messages successfully claimed, without returning the actual message

For more information see https://redis.io/commands/xclaim

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min_idle_time** (`int`) –
- **message_ids** (`Union`[`List`[`Union`[`int`, `bytes`, `str`, `memoryview`]], `Tuple`[`Union`[`int`, `bytes`, `str`, `memoryview`]]]) –
- **idle** (`Optional`[`int`], default: `None`) –
- **time** (`Optional`[`int`], default: `None`) –
- **retrycount** (`Optional`[`int`], default: `None`) –
- **force** (`bool`, default: `False`) –
- **justid** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xdel(name, *ids)`

Deletes one or more messages from a stream. name: name of the stream. *ids: message ids to delete.

For more information see https://redis.io/commands/xdel

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **ids** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_create(name, groupname, id='$', mkstream=False, entries_read=None)`

Create a new consumer group associated with a stream. name: name of the stream. groupname: name of the consumer group. id: ID of the last item in the stream to consider already delivered.

For more information see https://redis.io/commands/xgroup-create

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **id** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'$'`) –
- **mkstream** (`bool`, default: `False`) –
- **entries_read** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_createconsumer(name, groupname, consumername)`

Consumers in a consumer group are auto-created every time a new consumer name is mentioned by some command. They can be explicitly created by using this command. name: name of the stream. groupname: name of the consumer group. consumername: name of consumer to create.

See: https://redis.io/commands/xgroup-createconsumer

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_delconsumer(name, groupname, consumername)`

Remove a specific consumer from a consumer group. Returns the number of pending messages that the consumer had before it was deleted. name: name of the stream. groupname: name of the consumer group. consumername: name of consumer to delete

For more information see https://redis.io/commands/xgroup-delconsumer

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_destroy(name, groupname)`

Destroy a consumer group. name: name of the stream. groupname: name of the consumer group.

For more information see https://redis.io/commands/xgroup-destroy

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_setid(name, groupname, id, entries_read=None)`

Set the consumer group last delivered ID to something else. name: name of the stream. groupname: name of the consumer group. id: ID of the last item in the stream to consider already delivered.

For more information see https://redis.io/commands/xgroup-setid

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **id** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –
- **entries_read** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xinfo_consumers(name, groupname)`

Returns general information about the consumers in the group. name: name of the stream. groupname: name of the consumer group.

For more information see https://redis.io/commands/xinfo-consumers

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xinfo_groups(name)`

Returns general information about the consumer groups of the stream. name: name of the stream.

For more information see https://redis.io/commands/xinfo-groups

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xinfo_stream(name, full=False)`

Returns general information about the stream. name: name of the stream. full: optional boolean, false by default. Return full summary

For more information see https://redis.io/commands/xinfo-stream

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **full** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xlen(name)`

Returns the number of elements in a given stream.

For more information see https://redis.io/commands/xlen

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xpending``(name, groupname)`

Returns information about pending messages of a group. name: name of the stream. groupname: name of the consumer group.

For more information see https://redis.io/commands/xpending

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xpending_range``(name, groupname, min, max, count, consumername=None, idle=None)`

Returns information about pending messages, in a range.

name: name of the stream. groupname: name of the consumer group. idle: available from version 6.2. filter entries by their idle-time, given in milliseconds (optional). min: minimum stream ID. max: maximum stream ID. count: number of messages to return consumername: name of a consumer to filter by (optional).

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –
- **max** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –
- **count** (`int`) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **idle** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xrange``(name, min='-', max='+', count=None)`

Read stream values within an interval.

name: name of the stream.

start: first stream ID. defaults to '-',
meaning the earliest available.

finish: last stream ID. defaults to '+',
meaning the latest available.

count: if set, only return this many items, beginning with the
earliest available.

For more information see https://redis.io/commands/xrange

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'-'`) –
- **max** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'+'`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xread(streams, count=None, block=None)

Block and monitor multiple streams for new data.

streams: a dict of stream names to stream IDs, where
IDs indicate the last ID already seen.

count: if set, only return this many items, beginning with the
earliest available.

block: number of milliseconds to wait, if nothing already present.

For more information see https://redis.io/commands/xread

PARAMETERS

- **streams** (`Dict`[`Union`[`bytes`, `str`, `memoryview`], `Union`[`int`, `bytes`, `str`, `memoryview`]]) –
- **count** (`Optional`[`int`], default: `None`) –
- **block** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xreadgroup(groupname, consumername, streams, count=None, block=None, noack=False)

Read from a stream via a consumer group.

groupname: name of the consumer group.

consumername: name of the requesting consumer.

streams: a dict of stream names to stream IDs, where
IDs indicate the last ID already seen.

count: if set, only return this many items, beginning with the
earliest available.

block: number of milliseconds to wait, if nothing already present. noack: do not add messages to the PEL

For more information see https://redis.io/commands/xreadgroup

PARAMETERS

- **groupname** (`str`) –
- **consumername** (`str`) –
- **streams** (`Dict`[`Union`[`bytes`, `str`, `memoryview`], `Union`[`int`, `bytes`, `str`, `memoryview`]]) –
- **count** (`Optional`[`int`], default: `None`) –
- **block** (`Optional`[`int`], default: `None`) –
- **noack** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xrevrange(name, max='+', min='-', count=None)

Read stream values within an interval, in reverse order.

name: name of the stream

start: first stream ID. defaults to '+',
meaning the latest available.

finish: last stream ID. defaults to '-',
meaning the earliest available.

count: if set, only return this many items, beginning with the
latest available.

For more information see https://redis.io/commands/xrevrange

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **max** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'+'`) –
- **min** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'-'`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xtrim(name, maxlen=None, approximate=True, minid=None, limit=None)

Trims old messages from a stream. name: name of the stream. maxlen: truncate old stream messages beyond this size Can't be specified with minid. approximate: actual stream length may be slightly more than maxlen minid: the minimum id in the stream to query Can't be specified with maxlen. limit: specifies the maximum number of entries to retrieve

For more information see https://redis.io/commands/xtrim

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **maxlen** (`Optional`[`int`], default: `None`) –
- **approximate** (`bool`, default: `True`) –
- **minid** (`Union`[`int`, `bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **limit** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zadd(name, mapping, nx=False, xx=False, ch=False, incr=False, gt=False, lt=False)

Set any number of element-name, score pairs to the key `name`. Pairs are specified as a dict of element-names keys to score values.

`nx` forces ZADD to only create new elements and not to update scores for elements that already exist.

`xx` forces ZADD to only update scores of elements that already exist. New elements will not be added.

`ch` modifies the return value to be the numbers of elements changed. Changed elements include new elements that were added and elements whose scores changed.

`incr` modifies ZADD to behave like ZINCRBY. In this mode only a single element/score pair can be specified and the score is the amount the existing score will be incremented by. When using this mode the return value of ZADD will be the new score of the element.

`LT` Only update existing elements if the new score is less than the current score. This flag doesn't prevent adding new elements.

`GT` Only update existing elements if the new score is greater than the current score. This flag doesn't prevent adding new elements.

The return value of ZADD varies based on the mode specified. With no options, ZADD returns the number of new elements added to the sorted set.

`NX`, `LT`, and `GT` are mutually exclusive options.

See: https://redis.io/commands/ZADD

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **mapping** (`Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **ch** (`bool`, default: `False`) –
- **incr** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zcard(name)

Return the number of elements in the sorted set `name`

For more information see https://redis.io/commands/zcard

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zcount(name, min, max)

Returns the number of elements in the sorted set at key `name` with a score between `min` and `max`.

For more information see https://redis.io/commands/zcount

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`float`, `str`]) –
- **max** (`Union`[`float`, `str`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zdiff(keys, withscores=False)`

Returns the difference between the first and all successive input sorted sets provided in `keys`.

For more information see https://redis.io/commands/zdiff

PARAMETERS

- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **withscores** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zdiffstore(dest, keys)`

Computes the difference between the first and all successive input sorted sets provided in `keys` and stores the result in `dest`.

For more information see https://redis.io/commands/zdiffstore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zincrby(name, amount, value)`

Increment the score of `value` in sorted set `name` by `amount`

For more information see https://redis.io/commands/zincrby

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`float`) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zinter(keys, aggregate=None, withscores=False)`

Return the intersect of multiple sorted sets specified by `keys`. With the `aggregate` option, it is possible to specify how the results of the union are aggregated. This option defaults to SUM, where the score of an element is summed across the inputs where it exists. When this option is set to either MIN or MAX, the resulting set will contain the minimum or maximum score of an element across the inputs where it exists.

For more information see https://redis.io/commands/zinter

PARAMETERS

- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **aggregate** (`Optional`[`str`], default: `None`) –
- **withscores** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zintercard(numkeys, keys, limit=0)

Return the cardinality of the intersect of multiple sorted sets specified by ``keys`. When LIMIT provided (defaults to 0 and means unlimited), if the intersection cardinality reaches limit partway through the computation, the algorithm will exit and yield limit as the cardinality

For more information see https://redis.io/commands/zintercard

PARAMETERS

- **numkeys** (`int`) –
- **keys** (`List`[`str`]) –
- **limit** (`int`, default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

zinterstore(dest, keys, aggregate=None)

Intersect multiple sorted sets specified by `keys` into a new sorted set, `dest`. Scores in the destination will be aggregated based on the `aggregate`. This option defaults to SUM, where the score of an element is summed across the inputs where it exists. When this option is set to either MIN or MAX, the resulting set will contain the minimum or maximum score of an element across the inputs where it exists.

For more information see https://redis.io/commands/zinterstore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **keys** (`Union`[`Sequence`[`Union`[`bytes`, `str`, `memoryview`]], `Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `float`]]) –
- **aggregate** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zlexcount(name, min, max)

Return the number of items in the sorted set `name` between the lexicographical range `min` and `max`.

For more information see https://redis.io/commands/zlexcount

zmpop(num_keys, keys, min=False, max=False, count=1)

Pop `count` values (default 1) off of the first non-empty sorted set named in the `keys` list. For more information see https://redis.io/commands/zmpop

PARAMETERS

- **num_keys** (`int`) –
- **keys** (`List`[`str`]) –
- **min** (`Optional`[`bool`], default: `False`) –
- **max** (`Optional`[`bool`], default: `False`) –
- **count** (`Optional`[`int`], default: `1`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

zmscore(key, members)

Returns the scores associated with the specified members in the sorted set stored at key. `members` should be a list of the member name. Return type is a list of score. If the member does not exist, a None will be returned in corresponding position.

For more information see https://redis.io/commands/zmscore

PARAMETERS

- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **members** (`List`[`str`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zpopmax(name, count=None)`

Remove and return up to `count` members with the highest scores from the sorted set `name`.

For more information see https://redis.io/commands/zpopmax

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zpopmin(name, count=None)`

Remove and return up to `count` members with the lowest scores from the sorted set `name`.

For more information see https://redis.io/commands/zpopmin

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zrandmember(key, count=None, withscores=False)`

Return a random element from the sorted set value stored at key.

`count` if the argument is positive, return an array of distinct fields. If called with a negative count, the behavior changes and the command is allowed to return the same field multiple times. In this case, the number of returned fields is the absolute value of the specified count.

`withscores` The optional WITHSCORES modifier changes the reply so it includes the respective scores of the randomly selected elements from the sorted set.

For more information see https://redis.io/commands/zrandmember

PARAMETERS

- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **count** (`Optional`[`int`], default: `None`) –
- **withscores** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrange(name, start, end, desc=False, withscores=False, score_cast_func=<class 'float'>, byscore=False, bylex=False, offset=None, num=None)

Return a range of values from sorted set `name` between `start` and `end` sorted in ascending order.

`start` and `end` can be negative, indicating the end of the range.

`desc` a boolean indicating whether to sort the results in reversed order.

`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.

`byscore` when set to True, returns the range of elements from the sorted set having scores equal or between `start` and `end`.

`bylex` when set to True, returns the range of elements from the sorted set between the `start` and `end` lexicographical closed range intervals. Valid `start` and `end` must start with (or [, in order to specify whether the range interval is exclusive or inclusive, respectively.

`offset` and `num` are specified, then return a slice of the range. Can't be provided when using `bylex`.

For more information see https://redis.io/commands/zrange

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`) –
- **desc** (`bool`, default: `False`) –
- **withscores** (`bool`, default: `False`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –
- **byscore** (`bool`, default: `False`) –
- **bylex** (`bool`, default: `False`) –
- **offset** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrangebylex(name, min, max, start=None, num=None)

Return the lexicographical range of values from sorted set `name` between `min` and `max`.

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

For more information see https://redis.io/commands/zrangebylex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **max** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrangebyscore(name, min, max, start=None, num=None, withscores=False, score_cast_func=<class '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

For more information see https://redis.io/commands/zrangebyscore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`float`, `str`]) –
- **max** (`Union`[`float`, `str`]) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –
- **withscores** (`bool`, default: `False`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrangestore(dest, name, start, end, byscore=False, bylex=False, desc=False, offset=None, num=None)

Stores in `dest` the result of a range of values from sorted set `name` between `start` and `end` sorted in ascending order.

`start` and `end` can be negative, indicating the end of the range.

`byscore` when set to True, returns the range of elements from the sorted set having scores equal or between `start` and `end`.

`bylex` when set to True, returns the range of elements from the sorted set between the `start` and `end` lexicographical closed range intervals. Valid `start` and `end` must start with (or [, in order to specify whether the range interval is exclusive or inclusive, respectively.

`desc` a boolean indicating whether to sort the results in reversed order.

`offset` and `num` are specified, then return a slice of the range. Can't be provided when using `bylex`.

For more information see https://redis.io/commands/zrangestore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`) –
- **byscore** (`bool`, default: `False`) –
- **bylex** (`bool`, default: `False`) –
- **desc** (`bool`, default: `False`) –
- **offset** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrank(name, value, withscore=False)

Returns a 0-based value indicating the rank of `value` in sorted set `name`. The optional WITHSCORE argument supplements the command's reply with the score of the element returned.

For more information see https://redis.io/commands/zrank

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **withscore** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrem(name, *values)

Remove member `values` from sorted set `name`

For more information see https://redis.io/commands/zrem

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zremrangebylex(name, min, max)

Remove all elements in the sorted set `name` between the lexicographical range specified by `min` and `max`.

Returns the number of elements removed.

For more information see https://redis.io/commands/zremrangebylex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **max** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zremrangebyrank(name, min, max)

Remove all elements in the sorted set `name` with ranks between `min` and `max`. Values are 0-based, ordered from smallest score to largest. Values can be negative indicating the highest scores. Returns the number of elements removed

For more information see https://redis.io/commands/zremrangebyrank

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`int`) –
- **max** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zremrangebyscore(name, min, max)

Remove all elements in the sorted set `name` with scores between `min` and `max`. Returns the number of elements removed.

For more information see https://redis.io/commands/zremrangebyscore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`float`, `str`]) –
- **max** (`Union`[`float`, `str`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrevrange(name, start, end, withscores=False, score_cast_func=<class 'float'>)

Return a range of values from sorted set `name` between `start` and `end` sorted in descending order.

`start` and `end` can be negative, indicating the end 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

For more information see https://redis.io/commands/zrevrange

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`) –
- **withscores** (`bool`, default: `False`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrevrangebylex(name, max, min, start=None, num=None)

Return the reversed lexicographical range of values from sorted set `name` between `max` and `min`.

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

For more information see https://redis.io/commands/zrevrangebylex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **max** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **min** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrevrangebyscore(name, max, min, start=None, num=None, withscores=False, score_cast_func=<class 'float'>)

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

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

For more information see https://redis.io/commands/zrevrangebyscore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **max** (`Union`[`float`, `str`]) –
- **min** (`Union`[`float`, `str`]) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –
- **withscores** (`bool`, default: `False`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

zrevrank(name, value, withscore=False)

Returns a 0-based value indicating the descending rank of `value` in sorted set `name`. The optional `withscore` argument supplements the command's reply with the score of the element returned.

For more information see https://redis.io/commands/zrevrank

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **withscore** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zscan(name, cursor=0, match=None, count=None, score_cast_func=<class 'float'>)

Incrementally return lists of elements in a sorted set. Also return a cursor indicating the scan position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

`score_cast_func` a callable used to cast the score return value

For more information see https://redis.io/commands/zscan

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **cursor** (`int`, default: `0`) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zscan_iter`(name, match=None, count=None, score_cast_func=<class 'float'>)`

Make an iterator using the ZSCAN command so that the client doesn't need to remember the cursor position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

`score_cast_func` a callable used to cast the score return value

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

RETURN TYPE

`Iterator`

zscore`(name, value)`

Return the score of element `value` in sorted set `name`

For more information see https://redis.io/commands/zscore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zunion`(keys, aggregate=None, withscores=False)`

Return the union of multiple sorted sets specified by `keys`. `keys` can be provided as dictionary of keys and their weights. Scores will be aggregated based on the `aggregate`, or SUM if none is provided.

For more information see https://redis.io/commands/zunion

PARAMETERS

- **keys** (`Union`[`Sequence`[`Union`[`bytes`, `str`, `memoryview`]], `Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `float`]]) –
- **aggregate** (`Optional`[`str`], default: `None`) –
- **withscores** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zunionstore`(dest, keys, aggregate=None)`

Union multiple sorted sets specified by `keys` into a new sorted set, `dest`. Scores in the destination will be aggregated based on the `aggregate`, or SUM if none is provided.

For more information see https://redis.io/commands/zunionstore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **keys** (`Union`[`Sequence`[`Union`[`bytes`, `str`, `memoryview`]], `Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `float`]]) –
- **aggregate** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

Sentinel Commands

class `redis.commands.sentinel.SentinelCommands` [source]

A class containing the commands specific to redis sentinel. This class is to be used as a mixin.

`sentinel(*args)` [source]

Redis Sentinel's SENTINEL command.

`sentinel_ckquorum(new_master_name)` [source]

Check if the current Sentinel configuration is able to reach the quorum needed to failover a master, and the majority needed to authorize the failover.

This command should be used in monitoring systems to check if a Sentinel deployment is ok.

`sentinel_failover(new_master_name)` [source]

Force a failover as if the master was not reachable, and without asking for agreement to other Sentinels (however a new version of the configuration will be published so that the other Sentinels will update their configurations).

`sentinel_flushconfig()` [source]

Force Sentinel to rewrite its configuration on disk, including the current Sentinel state.

Normally Sentinel rewrites the configuration every time something changes in its state (in the context of the subset of the state which is persisted on disk across restart). However sometimes it is possible that the configuration file is lost because of operation errors, disk failures, package upgrade scripts or configuration managers. In those cases a way to to force Sentinel to rewrite the configuration file is handy.

This command works even if the previous configuration file is completely missing.

`sentinel_get_master_addr_by_name(service_name)` [source]

Returns a (host, port) pair for the given `service_name`

`sentinel_master(service_name)` [source]

Returns a dictionary containing the specified masters state.

`sentinel_masters()` [source]

Returns a list of dictionaries containing each master's state.

`sentinel_monitor(name, ip, port, quorum)` [source]

Add a new master to Sentinel to be monitored

`sentinel_remove(name)` [source]

Remove a master from Sentinel's monitoring

`sentinel_reset(pattern)` [source]

This command will reset all the masters with matching name. The pattern argument is a glob-style pattern.

The reset process clears any previous state in a master (including a failover in progress), and removes every slave and sentinel already discovered and associated with the master.

`sentinel_sentinels(service_name)` [source]

Returns a list of sentinels for `service_name`

`sentinel_set(name, option, value)` [source]

Set Sentinel monitoring parameters for a given master

`sentinel_slaves(service_name)` [source]

Returns a list of slaves for `service_name`

Redis Cluster Commands

The following Redis commands are available within a Redis Cluster. Generally they can be used as functions on your redis connection.

class `redis.commands.cluster.RedisClusterCommands(*args, **kwargs)` [source]

A class for all Redis Cluster commands

For key-based commands, the target node(s) will be internally determined by the keys' hash slot. Non-key-based commands can be executed with the 'target_nodes' argument to target specific nodes. By default, if target_nodes is not specified, the command will be executed on the default cluster node.

PARAMETERS

:target_nodes – type can be one of the followings: - nodes flag: ALL_NODES, PRIMARIES, REPLICAS, RANDOM - 'ClusterNode' - 'list(ClusterNodes)' - 'dict(any:clusterNodes)'

for example:

r.cluster_info(target_nodes=RedisCluster.ALL_NODES)

acl_cat(category=None, **kwargs)

Returns a list of categories or commands within a category.

If `category` is not supplied, returns a list of all categories. If `category` is supplied, returns a list of all commands within that category.

For more information see https://redis.io/commands/acl-cat

PARAMETERS

category (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_deluser(*username, **kwargs)

Delete the ACL for the specified ``username``s

For more information see https://redis.io/commands/acl-deluser

PARAMETERS

username (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_dryrun(username, *args, **kwargs)

Simulate the execution of a given command by a given `username`.

For more information see https://redis.io/commands/acl-dryrun

acl_genpass(bits=None, **kwargs)

Generate a random password value. If `bits` is supplied then use this number of bits, rounded to the next multiple of 4. See: https://redis.io/commands/acl-genpass

PARAMETERS

bits (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_getuser(username, **kwargs)

Get the ACL details for the specified `username`.

If `username` does not exist, return None

For more information see https://redis.io/commands/acl-getuser

PARAMETERS

username (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_help(***kwargs*)

The ACL HELP command returns helpful text describing the different subcommands.

For more information see https://redis.io/commands/acl-help

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_list(***kwargs*)

Return a list of all ACLs on the server

For more information see https://redis.io/commands/acl-list

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_load(***kwargs*)

Load ACL rules from the configured `aclfile`.

Note that the server must be configured with the `aclfile` directive to be able to load ACL rules from an aclfile.

For more information see https://redis.io/commands/acl-load

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_log(*count=None*, ***kwargs*)

Get ACL logs as a list. :param int count: Get logs[0:count]. :rtype: List.

For more information see https://redis.io/commands/acl-log

PARAMETERS

count (`Optional`[`int`], default: `None`) –

acl_log_reset(***kwargs*)

Reset ACL logs. :rtype: Boolean.

For more information see https://redis.io/commands/acl-log

acl_save(***kwargs*)

Save ACL rules to the configured `aclfile`.

Note that the server must be configured with the `aclfile` directive to be able to save ACL rules to an aclfile.

For more information see https://redis.io/commands/acl-save

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_setuser(*username*, *enabled=False*, *nopass=False*, *passwords=None*, *hashed_passwords=None*, *categories=None*, *commands=None*, *keys=None*, *channels=None*, *selectors=None*, *reset=False*, *reset_keys=False*, *reset_channels=False*, *reset_passwords=False*, ***kwargs*)

Create or update an ACL user.

Create or update the ACL for `username`. If the user already exists, the existing ACL is completely overwritten and replaced with the specified values.

`enabled` is a boolean indicating whether the user should be allowed to authenticate or not. Defaults to `False`.

`nopass` is a boolean indicating whether the can authenticate without a password. This cannot be True if `passwords` are also specified.

`passwords` if specified is a list of plain text passwords to add to or remove from the user. Each password must be prefixed with a '+' to add or a '-' to remove. For convenience, the value of `passwords` can be a simple prefixed string when adding or removing a single password.

`hashed_passwords` if specified is a list of SHA-256 hashed passwords to add to or remove from the user. Each hashed password must be prefixed with a '+' to add or a '-' to remove. For convenience, the value of `hashed_passwords` can be a simple prefixed string when adding or removing a single password.

`categories` if specified is a list of strings representing category permissions. Each string must be prefixed with either a '+' to add the category permission or a '-' to remove the category permission.

`commands` if specified is a list of strings representing command permissions. Each string must be prefixed with either a '+' to add the command permission or a '-' to remove the command permission.

`keys` if specified is a list of key patterns to grant the user access to. Keys patterns allow '*' to support wildcard matching. For example, '*' grants access to all keys while 'cache:*' grants access to all keys that are prefixed with 'cache:'. `keys` should not be prefixed with a '~'.

`reset` is a boolean indicating whether the user should be fully reset prior to applying the new ACL. Setting this to True will remove all existing passwords, flags and privileges from the user and then apply the specified rules. If this is False, the user's existing passwords, flags and privileges will be kept and any new specified rules will be applied on top.

`reset_keys` is a boolean indicating whether the user's key permissions should be reset prior to applying any new key permissions specified in `keys`. If this is False, the user's existing key permissions will be kept and any new specified key permissions will be applied on top.

`reset_channels` is a boolean indicating whether the user's channel permissions should be reset prior to applying any new channel permissions specified in `channels`.If this is False, the user's existing channel permissions will be kept and any new specified channel permissions will be applied on top.

`reset_passwords` is a boolean indicating whether to remove all existing passwords and the 'nopass' flag from the user prior to applying any new passwords specified in 'passwords' or 'hashed_passwords'. If this is False, the user's existing passwords and 'nopass' status will be kept and any new specified passwords or hashed_passwords will be applied on top.

For more information see https://redis.io/commands/acl-setuser

PARAMETERS

- **username** (`str`) –
- **enabled** (`bool`, default: `False`) –
- **nopass** (`bool`, default: `False`) –
- **passwords** (`Union`[`str`, `Iterable`[`str`], `None`], default: `None`) –
- **hashed_passwords** (`Union`[`str`, `Iterable`[`str`], `None`], default: `None`) –
- **categories** (`Optional`[`Iterable`[`str`]], default: `None`) –
- **commands** (`Optional`[`Iterable`[`str`]], default: `None`) –
- **keys** (`Optional`[`Iterable`[`Union`[`bytes`, `str`, `memoryview`]]], default: `None`) –
- **channels** (`Optional`[`Iterable`[`Union`[`bytes`, `str`, `memoryview`]]], default: `None`) –
- **selectors** (`Optional`[`Iterable`[`Tuple`[`str`, `Union`[`bytes`, `str`, `memoryview`]]]], default: `None`) –
- **reset** (`bool`, default: `False`) –
- **reset_keys** (`bool`, default: `False`) –
- **reset_channels** (`bool`, default: `False`) –
- **reset_passwords** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

acl_users(**kwargs)

Returns a list of all registered users on the server.

For more information see https://redis.io/commands/acl-users

RETURN TYPE

`Union[Awaitable, Any]`

acl_whoami(**kwargs)

Get the username for the current connection

For more information see https://redis.io/commands/acl-whoami

RETURN TYPE

`Union[Awaitable, Any]`

append(key, value)

Appends the string `value` to the value at `key`. If `key` doesn't already exist, create it with a value of `value`. Returns the new length of the value at `key`.

For more information see https://redis.io/commands/append

PARAMETERS

- **key** (`Union[bytes, str, memoryview]`) –
- **value** (`Union[bytes, memoryview, str, int, float]`) –

RETURN TYPE

`Union[Awaitable, Any]`

auth(password, username=None, **kwargs)

Authenticates the user. If you do not pass username, Redis will try to authenticate for the "default" user. If you do pass username, it will authenticate for the given user. For more information see https://redis.io/commands/auth

PARAMETERS

- **password** (`str`) –
- **username** (`Optional[str]`, default: `None`) –

bf()

Access the bloom namespace.

bgrewriteaof(**kwargs)

Tell the Redis server to rewrite the AOF file from data in memory.

For more information see https://redis.io/commands/bgrewriteaof

bgsave(schedule=True, **kwargs)

Tell the Redis server to save its data to disk. Unlike save(), this method is asynchronous and returns immediately.

For more information see https://redis.io/commands/bgsave

PARAMETERS

schedule (`bool`, default: `True`) –

RETURN TYPE

`Union[Awaitable, Any]`

bitcount(key, start=None, end=None, mode=None)

Returns the count of set bits in the value of `key`. Optional `start` and `end` parameters indicate which bytes to consider

For more information see https://redis.io/commands/bitcount

PARAMETERS

- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`Optional`[`int`], default: `None`) –
- **end** (`Optional`[`int`], default: `None`) –
- **mode** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

bitfield(key, default_overflow=None)

Return a BitFieldOperation instance to conveniently construct one or more bitfield operations on `key`.

For more information see https://redis.io/commands/bitfield

PARAMETERS

- **self** (`Union`[`Redis`, `Redis`]) –
- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **default_overflow** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`BitFieldOperation`

bitfield_ro(key, encoding, offset, items=None)

Return an array of the specified bitfield values where the first value is found using `encoding` and `offset` parameters and remaining values are result of corresponding encoding/offset pairs in optional list `items` Read-only variant of the BITFIELD command.

For more information see https://redis.io/commands/bitfield_ro

PARAMETERS

- **self** (`Union`[`Redis`, `Redis`]) –
- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **encoding** (`str`) –
- **offset** (`Union`[`int`, `str`]) –
- **items** (`Optional`[`list`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

bitop(operation, dest, *keys)

Perform a bitwise operation using `operation` between `keys` and store the result in `dest`.

For more information see https://redis.io/commands/bitop

PARAMETERS

- **operation** (`str`) –
- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **keys** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

bitpos(key, bit, start=None, end=None, mode=None)

Return the position of the first bit set to 1 or 0 in a string. `start` and `end` defines search range. The range is interpreted as a range of bytes and not a range of bits, so start=0 and end=2 means to look at the first three bytes.

For more information see https://redis.io/commands/bitpos

PARAMETERS

- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **bit** (`int`) –
- **start** (`Optional`[`int`], default: `None`) –
- **end** (`Optional`[`int`], default: `None`) –
- **mode** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

blmove(first_list, second_list, timeout, src='LEFT', dest='RIGHT')

Blocking version of lmove.

For more information see https://redis.io/commands/blmove

PARAMETERS

- **first_list** (`str`) –
- **second_list** (`str`) –
- **timeout** (`int`) –
- **src** (`str`, default: `'LEFT'`) –
- **dest** (`str`, default: `'RIGHT'`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

blmpop(timeout, numkeys, *args, direction, count=1)

Pop `count` values (default 1) from first non-empty in the list of provided key names.

When all lists are empty this command blocks the connection until another client pushes to it or until the timeout, timeout of 0 blocks indefinitely

For more information see https://redis.io/commands/blmpop

PARAMETERS

- **timeout** (`float`) –
- **numkeys** (`int`) –
- **args** (`List`[`str`]) –
- **direction** (`str`) –
- **count** (`Optional`[`int`], default: `1`) –

RETURN TYPE

`Optional`[`list`]

blpop(keys, timeout=0)

LPOP a value off of the first non-empty list named in the `keys` list.

If none of the lists in `keys` has a value to LPOP, then block for `timeout` seconds, or until a value gets pushed on to one of the lists.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/blpop

PARAMETERS

- **keys** (`List`) –
- **timeout** (`Optional`[`int`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

brpop(keys, timeout=0)

RPOP a value off of the first non-empty list named in the `keys` list.

If none of the lists in `keys` has a value to RPOP, then block for `timeout` seconds, or until a value gets pushed on to one of the lists.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/brpop

PARAMETERS

- **keys** (`List`) –
- **timeout** (`Optional`[`int`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

brpoplpush(src, dst, timeout=0)

Pop a value off the tail of `src`, push it on the head of `dst` and then return it.

This command blocks until a value is in `src` or until `timeout` seconds elapse, whichever is first. A `timeout` value of 0 blocks forever.

For more information see https://redis.io/commands/brpoplpush

PARAMETERS

- **src** (`str`) –
- **dst** (`str`) –
- **timeout** (`Optional`[`int`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`Optional`[`str`]], `str`, `None`]

bzmpop(timeout, numkeys, keys, min=False, max=False, count=1)

Pop `count` values (default 1) off of the first non-empty sorted set named in the `keys` list.

If none of the sorted sets in `keys` has a value to pop, then block for `timeout` seconds, or until a member gets added to one of the sorted sets.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/bzmpop

PARAMETERS

- **timeout** (`float`) –
- **numkeys** (`int`) –
- **keys** (`List`[`str`]) –
- **min** (`Optional`[`bool`], default: `False`) –
- **max** (`Optional`[`bool`], default: `False`) –
- **count** (`Optional`[`int`], default: `1`) –

RETURN TYPE

`Optional`[`list`]

`bzpopmax(keys, timeout=0)`

ZPOPMAX a value off of the first non-empty sorted set named in the `keys` list.

If none of the sorted sets in `keys` has a value to ZPOPMAX, then block for `timeout` seconds, or until a member gets added to one of the sorted sets.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/bzpopmax

PARAMETERS

- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **timeout** (`Union`[`int`, `float`, `bytes`, `str`, `memoryview`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`bzpopmin(keys, timeout=0)`

ZPOPMIN a value off of the first non-empty sorted set named in the `keys` list.

If none of the sorted sets in `keys` has a value to ZPOPMIN, then block for `timeout` seconds, or until a member gets added to one of the sorted sets.

If timeout is 0, then block indefinitely.

For more information see https://redis.io/commands/bzpopmin

PARAMETERS

- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **timeout** (`Union`[`int`, `float`, `bytes`, `str`, `memoryview`], default: `0`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`cf()`

Access the bloom namespace.

`client_getname(**kwargs)`

Returns the current connection name

For more information see https://redis.io/commands/client-getname

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`client_getredir(**kwargs)`

Returns the ID (an integer) of the client to whom we are redirecting tracking notifications.

see: https://redis.io/commands/client-getredir

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`client_id(**kwargs)`

Returns the current connection id

For more information see https://redis.io/commands/client-id

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`client_info(**kwargs)`

Returns information and statistics about the current client connection.

For more information see https://redis.io/commands/client-info

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`client_kill(address, **kwargs)`

Disconnects the client at `address` (ip:port)

For more information see https://redis.io/commands/client-kill

PARAMETERS

address (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`client_kill_filter(_id=None, _type=None, addr=None, skipme=None, laddr=None, user=None, **kwargs)`

Disconnects client(s) using a variety of filter options :type _id: `Optional`[`str`], default: `None` :param _id: Kills a client by its unique ID field :type _type: `Optional`[`str`], default: `None` :param _type: Kills a client by type where type is one of 'normal', 'master', 'slave' or 'pubsub' :type addr: `Optional`[`str`], default: `None` :param addr: Kills a client by its 'address:port' :type skipme: `Optional`[`bool`], default: `None` :param skipme: If True, then the client calling the command will not get killed even if it is identified by one of the filter options. If skipme is not provided, the server defaults to skipme=True :type laddr: `Optional`[`bool`], default: `None` :param laddr: Kills a client by its 'local (bind) address:port' :type user: `Optional`[`str`], default: `None` :param user: Kills a client for a specific user name

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`client_list(_type=None, client_id=[], **kwargs)`

Returns a list of currently connected clients. If type of client specified, only that type will be returned.

PARAMETERS

- **_type** (`Optional`[`str`], default: `None`) – optional. one of the client types (normal, master, replica, pubsub)
- **client_id** (`List`[`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]], default: `[]`) – optional. a list of client ids

RETURN TYPE

`Union`[`Awaitable`, `Any`]

For more information see https://redis.io/commands/client-list

`client_no_evict(mode)`

Sets the client eviction mode for the current connection.

For more information see https://redis.io/commands/client-no-evict

PARAMETERS

mode (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

`client_no_touch(mode)`

# The command controls whether commands sent by the client will alter # the LRU/LFU of the keys they access. # When turned on, the current client will not change LFU/LRU stats, # unless it sends the TOUCH command.

For more information see https://redis.io/commands/client-no-touch

PARAMETERS

mode (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

`client_pause(timeout, all=True, **kwargs)`

Suspend all the Redis clients for the specified amount of time.

For more information see https://redis.io/commands/client-pause

RETURN TYPE

`Union[Awaitable, Any]`

PARAMETERS

- **timeout** (`int`) – milliseconds to pause clients
- **all** (`bool`, default: `True`) – If true (default) all client commands are blocked.

otherwise, clients are only blocked if they attempt to execute a write command. For the WRITE mode, some commands have special behavior: EVAL/EVALSHA: Will block client for all scripts. PUBLISH: Will block client. PFCOUNT: Will block client. WAIT: Acknowledgments will be delayed, so this command will appear blocked.

`client_reply(reply, **kwargs)`

Enable and disable redis server replies.

`reply` Must be ON OFF or SKIP, ON - The default most with server replies to commands OFF - Disable server responses to commands SKIP - Skip the response of the immediately following command.

Note: When setting OFF or SKIP replies, you will need a client object with a timeout specified in seconds, and will need to catch the TimeoutError. The test_client_reply unit test illustrates this, and conftest.py has a client with a timeout.

See https://redis.io/commands/client-reply

PARAMETERS

reply (`Union[Literal['ON'], Literal['OFF'], Literal['SKIP']]`) –

RETURN TYPE

`Union[Awaitable, Any]`

`client_setinfo(attr, value, **kwargs)`

Sets the current connection library name or version For mor information see https://redis.io/commands/client-setinfo

PARAMETERS

- **attr** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union[Awaitable, Any]`

`client_setname(name, **kwargs)`

Sets the current connection name

For more information see https://redis.io/commands/client-setname

> Note
>
> This method sets client name only for **current** connection.
>
> If you want to set a common name for all connections managed by this client, use `client_name` constructor argument.

PARAMETERS

name (`str`) –

RETURN TYPE

`Union[Awaitable, Any]`

client_tracking(on=True, clientid=None, prefix=[], bcast=False, optin=False, optout=False, noloop=False, **kwargs)

Enables the tracking feature of the Redis server, that is used for server assisted client side caching.

`on` indicate for tracking on or tracking off. The dafualt is on.

`clientid` send invalidation messages to the connection with the specified ID.

`bcast` enable tracking in broadcasting mode. In this mode invalidation messages are reported for all the prefixes specified, regardless of the keys requested by the connection.

`optin` when broadcasting is NOT active, normally don't track keys in read only commands, unless they are called immediately after a CLIENT CACHING yes command.

`optout` when broadcasting is NOT active, normally track keys in read only commands, unless they are called immediately after a CLIENT CACHING no command.

`noloop` don't send notifications about keys modified by this connection itself.

`prefix` for broadcasting, register a given key prefix, so that notifications will be provided only for keys starting with this string.

See https://redis.io/commands/client-tracking

PARAMETERS

- **on** (`bool`, default: `True`) –
- **clientid** (`Optional`[`int`], default: `None`) –
- **prefix** (`Sequence`[`Union`[`bytes`, `str`, `memoryview`]], default: `[]`) –
- **bcast** (`bool`, default: `False`) –
- **optin** (`bool`, default: `False`) –
- **optout** (`bool`, default: `False`) –
- **noloop** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

client_tracking_off(clientid=None, prefix=[], bcast=False, optin=False, optout=False, noloop=False)

Turn off the tracking mode. For more information about the options look at client_tracking func.

See https://redis.io/commands/client-tracking

PARAMETERS

- **clientid** (`Optional`[`int`], default: `None`) –
- **prefix** (`Sequence`[`Union`[`bytes`, `str`, `memoryview`]], default: `[]`) –
- **bcast** (`bool`, default: `False`) –
- **optin** (`bool`, default: `False`) –
- **optout** (`bool`, default: `False`) –
- **noloop** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`client_tracking_on(clientid=None, prefix=[], bcast=False, optin=False, optout=False, noloop=False)`

Turn on the tracking mode. For more information about the options look at client_tracking func.

See https://redis.io/commands/client-tracking

PARAMETERS

- **clientid** (`Optional` [`int`], default: `None`) –
- **prefix** (`Sequence` [`Union` [`bytes` , `str` , `memoryview`]], default: `[]`) –
- **bcast** (`bool` , default: `False`) –
- **optin** (`bool` , default: `False`) –
- **optout** (`bool` , default: `False`) –
- **noloop** (`bool` , default: `False`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`client_trackinginfo(**kwargs)`

Returns the information about the current client connection's use of the server assisted client side cache.

See https://redis.io/commands/client-trackinginfo

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`client_unblock(client_id, error=False, **kwargs)`

Unblocks a connection by its client id. If `error` is True, unblocks the client with a special error message. If `error` is False (default), the client is unblocked using the regular timeout mechanism.

For more information see https://redis.io/commands/client-unblock

PARAMETERS

- **client_id** (`int`) –
- **error** (`bool` , default: `False`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`client_unpause(**kwargs)`

Unpause all redis clients

For more information see https://redis.io/commands/client-unpause

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`cluster_addslots(target_node, *slots)`

Assign new hash slots to receiving node. Sends to specified node.

TARGET_NODE

'ClusterNode' The node to execute the command on

For more information see https://redis.io/commands/cluster-addslots

PARAMETERS

- **target_node** (`TypeVar` (`TargetNodesT` , `str` , ClusterNode, `List` [ClusterNode], `Dict` [`Any` , ClusterNode])) –
- **slots** (`Union` [`bytes` , `memoryview` , `str` , `int` , `float`]) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`cluster_addslotsrange(target_node, *slots)`

Similar to the CLUSTER ADDSLOTS command. The difference between the two commands is that ADDSLOTS takes a list of slots to assign to the node, while ADDSLOTSRANGE takes a list of slot ranges (specified by start and end slots) to assign to the node.

TARGET_NODE
'ClusterNode' The node to execute the command on

For more information see https://redis.io/commands/cluster-addslotsrange

PARAMETERS
- **target_node** (`TypeVar` (`TargetNodesT`, `str`, ClusterNode, `List` [ClusterNode], `Dict` [`Any`, ClusterNode])) –
- **slots** (`Union` [`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE
`Union` [`Awaitable`, `Any`]

`cluster_count_failure_report(node_id)`

Return the number of failure reports active for a given node Sends to a random node

For more information see https://redis.io/commands/cluster-count-failure-reports

PARAMETERS
node_id (`str`) –

RETURN TYPE
`Union` [`Awaitable`, `Any`]

`cluster_countkeysinslot(slot_id)`

Return the number of local keys in the specified hash slot Send to node based on specified slot_id

For more information see https://redis.io/commands/cluster-countkeysinslot

PARAMETERS
slot_id (`int`) –

RETURN TYPE
`Union` [`Awaitable`, `Any`]

`cluster_delslots(*slots)`

Set hash slots as unbound in the cluster. It determines by it self what node the slot is in and sends it there

Returns a list of the results for each processed slot.

For more information see https://redis.io/commands/cluster-delslots

PARAMETERS
slots (`Union` [`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE
`List` [`bool`]

`cluster_delslotsrange(*slots)`

Similar to the CLUSTER DELSLOTS command. The difference is that CLUSTER DELSLOTS takes a list of hash slots to remove from the node, while CLUSTER DELSLOTSRANGE takes a list of slot ranges to remove from the node.

For more information see https://redis.io/commands/cluster-delslotsrange

PARAMETERS
slots (`Union` [`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE
`Union` [`Awaitable`, `Any`]

`cluster_failover(target_node, option=None)`

Forces a slave to perform a manual failover of its master Sends to specified node

TARGET_NODE
'ClusterNode' The node to execute the command on

For more information see https://redis.io/commands/cluster-failover

PARAMETERS

- **target_node** (`TypeVar` (`TargetNodesT`, `str`, ClusterNode, `List` [ClusterNode], `Dict` [`Any`, ClusterNode])) –
- **option** (`Optional` [`str`], default: `None`) –

RETURN TYPE
`Union` [`Awaitable`, `Any`]

`cluster_get_keys_in_slot(slot, num_keys)`

Returns the number of keys in the specified cluster slot

For more information see https://redis.io/commands/cluster-getkeysinslot

PARAMETERS

- **slot** (`int`) –
- **num_keys** (`int`) –

RETURN TYPE
`Union` [`Awaitable`, `Any`]

`cluster_info(target_nodes=None)`

Provides info about Redis Cluster node state. The command will be sent to a random node in the cluster if no target node is specified.

For more information see https://redis.io/commands/cluster-info

PARAMETERS
target_nodes (`Optional` [`TypeVar` (`TargetNodesT`, `str`, ClusterNode, `List` [ClusterNode], `Dict` [`Any`, ClusterNode])], default: `None`) –

RETURN TYPE
`Union` [`Awaitable`, `Any`]

`cluster_keyslot(key)`

Returns the hash slot of the specified key Sends to random node in the cluster

For more information see https://redis.io/commands/cluster-keyslot

PARAMETERS
key (`str`) –

RETURN TYPE
`Union` [`Awaitable`, `Any`]

`cluster_links(target_node)`

Each node in a Redis Cluster maintains a pair of long-lived TCP link with each peer in the cluster: One for sending outbound messages towards the peer and one for receiving inbound messages from the peer.

This command outputs information of all such peer links as an array.

For more information see https://redis.io/commands/cluster-links

PARAMETERS
target_node (`TypeVar` (`TargetNodesT`, `str`, ClusterNode, `List` [ClusterNode], `Dict` [`Any`, ClusterNode])) –

RETURN TYPE
`Union` [`Awaitable`, `Any`]

`cluster_meet(host, port, target_nodes=None)`

Force a node cluster to handshake with another node. Sends to specified node.

For more information see https://redis.io/commands/cluster-meet

PARAMETERS

- **host** (`str`) –
- **port** (`int`) –
- **target_nodes** (`Optional`[`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`cluster_myid(target_node)`

Returns the node's id.

TARGET_NODE

'ClusterNode' The node to execute the command on

For more information check https://redis.io/commands/cluster-myid/

PARAMETERS

target_node (`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`cluster_myshardid(target_nodes=None)`

Returns the shard ID of the node.

For more information see https://redis.io/commands/cluster-myshardid/

`cluster_nodes()`

Get Cluster config for the node. Sends to random node in the cluster

For more information see https://redis.io/commands/cluster-nodes

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`cluster_replicas(node_id, target_nodes=None)`

Provides a list of replica nodes replicating from the specified primary target node.

For more information see https://redis.io/commands/cluster-replicas

PARAMETERS

- **node_id** (`str`) –
- **target_nodes** (`Optional`[`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`cluster_replicate(target_nodes, node_id)`

Reconfigure a node as a slave of the specified master node

For more information see https://redis.io/commands/cluster-replicate

PARAMETERS

- **target_nodes** (`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])) –
- **node_id** (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

cluster_reset(soft=True, target_nodes=None)

Reset a Redis Cluster node

If 'soft' is True then it will send 'SOFT' argument If 'soft' is False then it will send 'HARD' argument

For more information see https://redis.io/commands/cluster-reset

PARAMETERS

- **soft** (`bool`, default: `True`) –
- **target_nodes** (`Optional`[`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

cluster_save_config(target_nodes=None)

Forces the node to save cluster state on disk

For more information see https://redis.io/commands/cluster-saveconfig

PARAMETERS

target_nodes (`Optional`[`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

cluster_set_config_epoch(epoch, target_nodes=None)

Set the configuration epoch in a new node

For more information see https://redis.io/commands/cluster-set-config-epoch

PARAMETERS

- **epoch** (`int`) –
- **target_nodes** (`Optional`[`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

cluster_setslot(target_node, node_id, slot_id, state)

Bind an hash slot to a specific node

TARGET_NODE

'ClusterNode' The node to execute the command on

For more information see https://redis.io/commands/cluster-setslot

PARAMETERS

- **target_node** (`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])) –
- **node_id** (`str`) –
- **slot_id** (`int`) –
- **state** (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`cluster_setslot_stable(slot_id)`

Clears migrating / importing state from the slot. It determines by it self what node the slot is in and sends it there.

For more information see https://redis.io/commands/cluster-setslot

PARAMETERS

slot_id (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`cluster_shards(target_nodes=None)`

Returns details about the shards of the cluster.

For more information see https://redis.io/commands/cluster-shards

`cluster_slots(target_nodes=None)`

Get array of Cluster slot to node mappings

For more information see https://redis.io/commands/cluster-slots

PARAMETERS

target_nodes (`Optional`[`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`cms()`

Access the bloom namespace.

`command(**kwargs)`

Returns dict reply of details about all Redis commands.

For more information see https://redis.io/commands/command

`command_docs(*args)`

This function throws a NotImplementedError since it is intentionally not supported.

`command_getkeysandflags(*args)`

Returns array of keys from a full Redis command and their usage flags.

For more information see https://redis.io/commands/command-getkeysandflags

PARAMETERS

args (`List`[`str`]) –

RETURN TYPE

`List`[`Union`[`str`, `List`[`str`]]]

`command_list(module=None, category=None, pattern=None)`

Return an array of the server's command names. You can use one of the following filters: `module`: get the commands that belong to the module `category`: get the commands in the ACL category `pattern`: get the commands that match the given pattern

For more information see https://redis.io/commands/command-list/

PARAMETERS

- **module** (`Optional`[`str`], default: `None`) –
- **category** (`Optional`[`str`], default: `None`) –
- **pattern** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

config_get(pattern='*', *args, **kwargs)

Return a dictionary of configuration based on the `pattern`

For more information see https://redis.io/commands/config-get

PARAMETERS

- **pattern** (`Union`[`bytes`, `str`, `memoryview`], default: `'*'`) –
- **args** (`List`[`Union`[`bytes`, `str`, `memoryview`]]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

config_resetstat(**kwargs)

Reset runtime statistics

For more information see https://redis.io/commands/config-resetstat

RETURN TYPE

`Union`[`Awaitable`, `Any`]

config_rewrite(**kwargs)

Rewrite config file with the minimal change to reflect running config.

For more information see https://redis.io/commands/config-rewrite

RETURN TYPE

`Union`[`Awaitable`, `Any`]

config_set(name, value, *args, **kwargs)

Set config item `name` with `value`

For more information see https://redis.io/commands/config-set

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **args** (`List`[`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

copy(source, destination, destination_db=None, replace=False)

Copy the value stored in the `source` key to the `destination` key.

`destination_db` an alternative destination database. By default, the `destination` key is created in the source Redis database.

`replace` whether the `destination` key should be removed before copying the value to it. By default, the value is not copied if the `destination` key already exists.

For more information see https://redis.io/commands/copy

PARAMETERS

- **source** (`str`) –
- **destination** (`str`) –
- **destination_db** (`Optional`[`str`], default: `None`) –
- **replace** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

dbsize(**kwargs)

Returns the number of keys in the current database

For more information see https://redis.io/commands/dbsize

RETURN TYPE

`Union`[`Awaitable`, `Any`]

debug_object(key, **kwargs)

Returns version specific meta information about a given key

For more information see https://redis.io/commands/debug-object

PARAMETERS
: **key** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

decr(name, amount=1)

Decrements the value of `key` by `amount`. If no key exists, the value will be initialized as 0 - `amount`

For more information see https://redis.io/commands/decrby

PARAMETERS
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`int`, default: `1`) –

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

decrby(name, amount=1)

Decrements the value of `key` by `amount`. If no key exists, the value will be initialized as 0 - `amount`

For more information see https://redis.io/commands/decrby

PARAMETERS
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`int`, default: `1`) –

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

delete(*keys)

Deletes the given keys in the cluster. The keys are first split up into slots and then an DEL command is sent for every slot

Non-existant keys are ignored. Returns the number of keys that were deleted.

For more information see https://redis.io/commands/del

PARAMETERS
: **keys** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

dump(name)

Return a serialized version of the value stored at the specified key. If key does not exist a nil bulk reply is returned.

For more information see https://redis.io/commands/dump

PARAMETERS
: **name** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

echo(value, **kwargs)

Echo the string back from the server

For more information see https://redis.io/commands/echo

PARAMETERS

value (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

eval(script, numkeys, *keys_and_args)

Execute the Lua `script`, specifying the `numkeys` the script will touch and the key names and argument values in `keys_and_args`. Returns the result of the script.

In practice, use the object returned by `register_script`. This function exists purely for Redis API completion.

For more information see https://redis.io/commands/eval

PARAMETERS

- **script** (`str`) –
- **numkeys** (`int`) –
- **keys_and_args** (`list`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

eval_ro(script, numkeys, *keys_and_args)

The read-only variant of the EVAL command

Execute the read-only Lua `script` specifying the `numkeys` the script will touch and the key names and argument values in `keys_and_args`. Returns the result of the script.

For more information see https://redis.io/commands/eval_ro

PARAMETERS

- **script** (`str`) –
- **numkeys** (`int`) –
- **keys_and_args** (`list`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

evalsha(sha, numkeys, *keys_and_args)

Use the `sha` to execute a Lua script already registered via EVAL or SCRIPT LOAD. Specify the `numkeys` the script will touch and the key names and argument values in `keys_and_args`. Returns the result of the script.

In practice, use the object returned by `register_script`. This function exists purely for Redis API completion.

For more information see https://redis.io/commands/evalsha

PARAMETERS

- **sha** (`str`) –
- **numkeys** (`int`) –
- **keys_and_args** (`list`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

evalsha_ro(sha, numkeys, *keys_and_args)

The read-only variant of the EVALSHA command

Use the `sha` to execute a read-only Lua script already registered via EVAL or SCRIPT LOAD. Specify the `numkeys` the script will touch and the key names and argument values in `keys_and_args`. Returns the result of the script.

For more information see https://redis.io/commands/evalsha_ro

PARAMETERS

- **sha** (`str`) –
- **numkeys** (`int`) –
- **keys_and_args** (`list`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

exists(*keys)

Returns the number of `names` that exist in the whole cluster. The keys are first split up into slots and then an EXISTS command is sent for every slot

For more information see https://redis.io/commands/exists

PARAMETERS

keys (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

expire(name, time, nx=False, xx=False, gt=False, lt=False)

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

Valid options are:

NX -> Set expiry only when the key has no expiry XX -> Set expiry only when the key has an existing expiry GT -> Set expiry only when the new expiry is greater than current one LT -> Set expiry only when the new expiry is less than current one

For more information see https://redis.io/commands/expire

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **time** (`Union`[`int`, `timedelta`]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

expireat(name, when, nx=False, xx=False, gt=False, lt=False)

Set an expire flag on key `name` with given `option`. `when` can be represented as an integer indicating unix time or a Python datetime object.

Valid options are:
-> NX – Set expiry only when the key has no expiry -> XX – Set expiry only when the key has an existing expiry -> GT – Set expiry only when the new expiry is greater than current one -> LT – Set expiry only when the new expiry is less than current one

For more information see https://redis.io/commands/expireat

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **when** (`Union`[`int`, `datetime`]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

expiretime(key)

Returns the absolute Unix timestamp (since January 1, 1970) in seconds at which the given key will expire.

For more information see https://redis.io/commands/expiretime

PARAMETERS

key (`str`) –

RETURN TYPE

`int`

failover()

This function throws a NotImplementedError since it is intentionally not supported.

fcall(function, numkeys, *keys_and_args)

Invoke a function.

For more information see https://redis.io/commands/fcall

PARAMETERS

- **numkeys** (`int`) –
- **keys_and_args** (`Optional`[`List`]) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

fcall_ro(function, numkeys, *keys_and_args)

This is a read-only variant of the FCALL command that cannot execute commands that modify data.

For more information see https://redis.io/commands/fcal_ro

PARAMETERS

- **numkeys** (`int`) –
- **keys_and_args** (`Optional`[`List`]) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

flushall(asynchronous=False, **kwargs)

Delete all keys in all databases on the current host.

`asynchronous` indicates whether the operation is executed asynchronously by the server.

For more information see https://redis.io/commands/flushall

PARAMETERS

asynchronous (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

flushdb(asynchronous=False, **kwargs)

Delete all keys in the current database.

`asynchronous` indicates whether the operation is executed asynchronously by the server.

For more information see https://redis.io/commands/flushdb

PARAMETERS

asynchronous (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

ft(index_name='idx')

Access the search namespace, providing support for redis search.

function_delete(library)

Delete the library called `library` and all its functions.

For more information see https://redis.io/commands/function-delete

PARAMETERS

library (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

function_dump()

Return the serialized payload of loaded libraries.

For more information see https://redis.io/commands/function-dump

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

function_flush(mode='SYNC')

Deletes all the libraries.

For more information see https://redis.io/commands/function-flush

PARAMETERS

mode (`str`, default: `'SYNC'`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

function_kill()

Kill a function that is currently executing.

For more information see https://redis.io/commands/function-kill

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

`function_list(library='*', withcode=False)`

Return information about the functions and libraries. :type library: `Optional[str]`, default: `'*'` :param library: pecify a pattern for matching library names :rtype: `Union[Awaitable[List], List]`

PARAMETERS

withcode (`Optional[bool]`, default: `False`) – cause the server to include the libraries source implementation in the reply

`function_load(code, replace=False)`

Load a library to Redis. :type code: `str` :param code: the source code (must start with Shebang statement that provides a metadata about the library) :type replace: `Optional[bool]`, default: `False` :param replace: changes the behavior to overwrite the existing library with the new contents. Return the library name that was loaded.

For more information see https://redis.io/commands/function-load

RETURN TYPE

`Union[Awaitable[str], str]`

`function_restore(payload, policy='APPEND')`

Restore libraries from the serialized `payload`. You can use the optional policy argument to provide a policy for handling existing libraries.

For more information see https://redis.io/commands/function-restore

PARAMETERS

- **payload** (`str`) –
- **policy** (`Optional[str]`, default: `'APPEND'`) –

RETURN TYPE

`Union[Awaitable[str], str]`

`function_stats()`

Return information about the function that's currently running and information about the available execution engines.

For more information see https://redis.io/commands/function-stats

RETURN TYPE

`Union[Awaitable[List], List]`

`gears_refresh_cluster(**kwargs)`

On an OSS cluster, before executing any gears function, you must call this command. # noqa

RETURN TYPE

`Union[Awaitable, Any]`

geoadd(name, values, nx=False, xx=False, ch=False)

Add the specified geospatial items to the specified key identified by the `name` argument. The Geospatial items are given as ordered members of the `values` argument, each item or place is formed by the triad longitude, latitude and name.

Note: You can use ZREM to remove elements.

`nx` forces ZADD to only create new elements and not to update scores for elements that already exist.

`xx` forces ZADD to only update scores of elements that already exist. New elements will not be added.

`ch` modifies the return value to be the numbers of elements changed. Changed elements include new elements that were added and elements whose scores changed.

For more information see https://redis.io/commands/geoadd

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Sequence`[`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **ch** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

geodist(name, place1, place2, unit=None)

Return the distance between `place1` and `place2` members of the `name` key. The units must be one of the following : m, km mi, ft. By default meters are used.

For more information see https://redis.io/commands/geodist

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **place1** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **place2** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **unit** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

geohash(name, *values)

Return the geo hash string for each item of `values` members of the specified key identified by the `name` argument.

For more information see https://redis.io/commands/geohash

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

geopos(name, *values)

Return the positions of each item of `values` as members of the specified key identified by the `name` argument. Each position is represented by the pairs lon and lat.

For more information see https://redis.io/commands/geopos

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

georadius(name, longitude, latitude, radius, unit=None, withdist=False, withcoord=False, withhash=False, count=None, sort=None, store=None, store_dist=None, any=False)

Return the members of the specified key identified by the `name` argument which are within the borders of the area specified with the `latitude` and `longitude` location and the maximum distance from the center specified by the `radius` value.

The units must be one of the following : m, km mi, ft. By default

`withdist` indicates to return the distances of each place.

`withcoord` indicates to return the latitude and longitude of each place.

`withhash` indicates to return the geohash string of each place.

`count` indicates to return the number of elements up to N.

`sort` indicates to return the places in a sorted way, ASC for nearest to fairest and DESC for fairest to nearest.

`store` indicates to save the places names in a sorted set named with a specific key, each element of the destination sorted set is populated with the score got from the original geo sorted set.

`store_dist` indicates to save the places names in a sorted set named with a specific key, instead of `store` the sorted set destination score is set with the distance.

For more information see https://redis.io/commands/georadius

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **longitude** (`float`) –
- **latitude** (`float`) –
- **radius** (`float`) –
- **unit** (`Optional`[`str`], default: `None`) –
- **withdist** (`bool`, default: `False`) –
- **withcoord** (`bool`, default: `False`) –
- **withhash** (`bool`, default: `False`) –
- **count** (`Optional`[`int`], default: `None`) –
- **sort** (`Optional`[`str`], default: `None`) –
- **store** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **store_dist** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **any** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

```
georadiusbymember(name, member, radius, unit=None, withdist=False,
    withcoord=False, withhash=False, count=None, sort=None, store=None,
    store_dist=None, any=False)
```

This command is exactly like `georadius` with the sole difference that instead of taking, as the center of the area to query, a longitude and latitude value, it takes the name of a member already existing inside the geospatial index represented by the sorted set.

For more information see https://redis.io/commands/georadiusbymember

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **member** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **radius** (`float`) –
- **unit** (`Optional`[`str`], default: `None`) –
- **withdist** (`bool`, default: `False`) –
- **withcoord** (`bool`, default: `False`) –
- **withhash** (`bool`, default: `False`) –
- **count** (`Optional`[`int`], default: `None`) –
- **sort** (`Optional`[`str`], default: `None`) –
- **store** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **store_dist** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **any** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`geosearch(name, member=None, longitude=None, latitude=None, unit='m', radius=None, width=None, height=None, sort=None, count=None, any=False, withcoord=False, withdist=False, withhash=False)`

Return the members of specified key identified by the `name` argument, which are within the borders of the area specified by a given shape. This command extends the GEORADIUS command, so in addition to searching within circular areas, it supports searching within rectangular areas.

This command should be used in place of the deprecated GEORADIUS and GEORADIUSBYMEMBER commands.

`member` Use the position of the given existing
member in the sorted set. Can't be given with `longitude` and `latitude`.

`longitude` and `latitude` Use the position given by this coordinates. Can't be given with `member` `radius` Similar to GEORADIUS, search inside circular area according the given radius. Can't be given with `height` and `width`. `height` and `width` Search inside an axis-aligned rectangle, determined by the given height and width. Can't be given with `radius`

`unit` must be one of the following : m, km, mi, ft. *m* for meters (the default value), *km* for kilometers, *mi* for miles and *ft* for feet.

`sort` indicates to return the places in a sorted way, ASC for nearest to furthest and DESC for furthest to nearest.

`count` limit the results to the first count matching items.

`any` is set to True, the command will return as soon as enough matches are found. Can't be provided without `count`

`withdist` indicates to return the distances of each place. `withcoord` indicates to return the latitude and longitude of each place.

`withhash` indicates to return the geohash string of each place.

For more information see https://redis.io/commands/geosearch

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **member** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`, `None`], default: `None`) –
- **longitude** (`Optional`[`float`], default: `None`) –
- **latitude** (`Optional`[`float`], default: `None`) –
- **unit** (`str`, default: `'m'`) –
- **radius** (`Optional`[`float`], default: `None`) –
- **width** (`Optional`[`float`], default: `None`) –
- **height** (`Optional`[`float`], default: `None`) –
- **sort** (`Optional`[`str`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **any** (`bool`, default: `False`) –
- **withcoord** (`bool`, default: `False`) –
- **withdist** (`bool`, default: `False`) –
- **withhash** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

geosearchstore(dest, name, member=None, longitude=None, latitude=None, unit='m', radius=None, width=None, height=None, sort=None, count=None, any=False, storedist=False)

This command is like GEOSEARCH, but stores the result in `dest`. By default, it stores the results in the destination sorted set with their geospatial information. if `store_dist` set to True, the command will stores the items in a sorted set populated with their distance from the center of the circle or box, as a floating-point number.

For more information see https://redis.io/commands/geosearchstore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **member** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`, `None`], default: `None`) –
- **longitude** (`Optional`[`float`], default: `None`) –
- **latitude** (`Optional`[`float`], default: `None`) –
- **unit** (`str`, default: `'m'`) –
- **radius** (`Optional`[`float`], default: `None`) –
- **width** (`Optional`[`float`], default: `None`) –
- **height** (`Optional`[`float`], default: `None`) –
- **sort** (`Optional`[`str`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **any** (`bool`, default: `False`) –
- **storedist** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

get(name)

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

For more information see https://redis.io/commands/get

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

getbit(name, offset)

Returns an integer indicating the value of `offset` in `name`

For more information see https://redis.io/commands/getbit

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **offset** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

getdel(name)

Get the value at key `name` and delete the key. This command is similar to GET, except for the fact that it also deletes the key on success (if and only if the key's value type is a string).

For more information see https://redis.io/commands/getdel

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`getex(name, ex=None, px=None, exat=None, pxat=None, persist=False)`

Get the value of key and optionally set its expiration. GETEX is similar to GET, but is a write command with additional options. All time parameters can be given as datetime.timedelta or integers.

`ex` sets an expire flag on key `name` for `ex` seconds.

`px` sets an expire flag on key `name` for `px` milliseconds.

`exat` sets an expire flag on key `name` for `ex` seconds, specified in unix time.

`pxat` sets an expire flag on key `name` for `ex` milliseconds, specified in unix time.

`persist` remove the time to live associated with `name`.

For more information see https://redis.io/commands/getex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **ex** (`Union`[`int`, `timedelta`, `None`], default: `None`) –
- **px** (`Union`[`int`, `timedelta`, `None`], default: `None`) –
- **exat** (`Union`[`int`, `datetime`, `None`], default: `None`) –
- **pxat** (`Union`[`int`, `datetime`, `None`], default: `None`) –
- **persist** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`getrange(key, start, end)`

Returns the substring of the string value stored at `key`, determined by the offsets `start` and `end` (both are inclusive)

For more information see https://redis.io/commands/getrange

PARAMETERS

- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`getset(name, value)`

Sets the value at key `name` to `value` and returns the old value at key `name` atomically.

As per Redis 6.2, GETSET is considered deprecated. Please use SET with GET parameter in new code.

For more information see https://redis.io/commands/getset

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`graph(index_name='idx')`

Access the graph namespace, providing support for redis graph data.

`hdel(name, *keys)`

Delete `keys` from hash `name`

For more information see https://redis.io/commands/hdel

PARAMETERS

- **name** (`str`) –
- **keys** (`List`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

`hello()`

This function throws a NotImplementedError since it is intentionally not supported.

`hexists(name, key)`

Returns a boolean indicating if `key` exists within hash `name`

For more information see https://redis.io/commands/hexists

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`bool`], `bool`]

`hget(name, key)`

Return the value of `key` within the hash `name`

For more information see https://redis.io/commands/hget

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`Optional` [`str`]], `str` , `None`]

`hgetall(name)`

Return a Python dict of the hash's name/value pairs

For more information see https://redis.io/commands/hgetall

PARAMETERS

name (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`dict`], `dict`]

`hincrby(name, key, amount=1)`

Increment the value of `key` in hash `name` by `amount`

For more information see https://redis.io/commands/hincrby

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –
- **amount** (`int` , default: `1`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`] `hincrbyfloat(name, key, amount=1.0)`

Increment the value of `key` in hash `name` by floating `amount`

For more information see https://redis.io/commands/hincrbyfloat

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –
- **amount** (`float`, default: `1.0`) –

RETURN TYPE

`Union` [`Awaitable` [`float`], `float`]

`hkeys(name)`

Return the list of keys within hash `name`

For more information see https://redis.io/commands/hkeys

PARAMETERS

name (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`List`], `List`]

`hlen(name)`

Return the number of elements in hash `name`

For more information see https://redis.io/commands/hlen

PARAMETERS

name (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

`hmget(name, keys, *args)`

Returns a list of values ordered identically to `keys`

For more information see https://redis.io/commands/hmget

PARAMETERS

- **name** (`str`) –
- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union` [`Awaitable` [`List`], `List`]

`hmset(name, mapping)`

Set key to value within hash `name` for each corresponding key and value from the `mapping` dict.

For more information see https://redis.io/commands/hmset

PARAMETERS

- **name** (`str`) –
- **mapping** (`dict`) –

RETURN TYPE

`Union` [`Awaitable` [`str`], `str`] `hrandfield(key, count=None, withvalues=False)`

Return a random field from the hash value stored at key.

count: if the argument is positive, return an array of distinct fields. If called with a negative count, the behavior changes and the command is allowed to return the same field multiple times. In this case, the number of returned fields is the absolute value of the specified count. withvalues: The optional WITHVALUES modifier changes the reply so it includes the respective values of the randomly selected hash fields.

For more information see https://redis.io/commands/hrandfield

PARAMETERS

- **key** (`str`) –
- **count** (`Optional`[`int`], default: `None`) –
- **withvalues** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`hscan(name, cursor=0, match=None, count=None)`

Incrementally return key/value slices in a hash. Also return a cursor indicating the scan position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

For more information see https://redis.io/commands/hscan

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **cursor** (`int`, default: `0`) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`hscan_iter(name, match=None, count=None)`

Make an iterator using the HSCAN command so that the client doesn't need to remember the cursor position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

PARAMETERS

- **name** (`str`) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Iterator`

hset(name, key=None, value=None, mapping=None, items=None)

Set `key` to `value` within hash `name`, `mapping` accepts a dict of key/value pairs that will be added to hash `name`. `items` accepts a list of key/value pairs that will be added to hash `name`. Returns the number of fields that were added.

For more information see https://redis.io/commands/hset

PARAMETERS

- **name** (`str`) –
- **key** (`Optional`[`str`], default: `None`) –
- **value** (`Optional`[`str`], default: `None`) –
- **mapping** (`Optional`[`dict`], default: `None`) –
- **items** (`Optional`[`list`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

hsetnx(name, key, value)

Set `key` to `value` within hash `name` if `key` does not exist. Returns 1 if HSETNX created a field, otherwise 0.

For more information see https://redis.io/commands/hsetnx

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`bool`], `bool`]

hstrlen(name, key)

Return the number of bytes stored in the value of `key` within hash `name`

For more information see https://redis.io/commands/hstrlen

PARAMETERS

- **name** (`str`) –
- **key** (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

hvals(name)

Return the list of values within hash `name`

For more information see https://redis.io/commands/hvals

PARAMETERS

name (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`List`], `List`]

incr(name, amount=1)

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

For more information see https://redis.io/commands/incrby

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`int`, default: `1`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

incrby(name, amount=1)

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

For more information see https://redis.io/commands/incrby

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`int`, default: `1`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

incrbyfloat(name, amount=1.0)

Increments the value at key `name` by floating `amount`. If no key exists, the value will be initialized as `amount`

For more information see https://redis.io/commands/incrbyfloat

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`float`, default: `1.0`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

info(section=None, *args, **kwargs)

Returns a dictionary containing information about the Redis server

The `section` option can be used to select a specific section of information

The section option is not supported by older versions of Redis Server, and will generate ResponseError

For more information see https://redis.io/commands/info

PARAMETERS

- **section** (`Optional`[`str`], default: `None`) –
- **args** (`List`[`str`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

json(encoder=<json.encoder.JSONEncoder object>, decoder=<json.decoder.JSONDecoder object>)

Access the json namespace, providing support for redis json.

keys(pattern='*', **kwargs)

Returns a list of keys matching `pattern`

For more information see https://redis.io/commands/keys

PARAMETERS

pattern (`Union`[`bytes`, `str`, `memoryview`], default: `'*'`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

lastsave(**kwargs)

Return a Python datetime object representing the last time the Redis database was saved to disk

For more information see https://redis.io/commands/lastsave

RETURN TYPE

`Union`[`Awaitable`, `Any`]

latency_doctor()

Raise a NotImplementedError, as the client will not support LATENCY DOCTOR. This funcion is best used within the redis-cli.

For more information see https://redis.io/commands/latency-doctor

latency_graph()

Raise a NotImplementedError, as the client will not support LATENCY GRAPH. This funcion is best used within the redis-cli.

For more information see https://redis.io/commands/latency-graph.

latency_histogram(*args)

This function throws a NotImplementedError since it is intentionally not supported.

latency_history(event)

Returns the raw data of the `event`'s latency spikes time series.

For more information see https://redis.io/commands/latency-history

PARAMETERS

event (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

latency_latest()

Reports the latest latency events logged.

For more information see https://redis.io/commands/latency-latest

RETURN TYPE

`Union`[`Awaitable`, `Any`]

latency_reset(*events)

Resets the latency spikes time series of all, or only some, events.

For more information see https://redis.io/commands/latency-reset

PARAMETERS

events (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

lcs(key1, key2, len=False, idx=False, minmatchlen=0, withmatchlen=False)

Find the longest common subsequence between `key1` and `key2`. If `len` is true the length of the match will will be returned. If `idx` is true the match position in each strings will be returned. `minmatchlen` restrict the list of matches to the ones of the given `minmatchlen`. If `withmatchlen` the length of the match also will be returned. For more information see https://redis.io/commands/lcs

PARAMETERS

- **key1** (`str`) –
- **key2** (`str`) –
- **len** (`Optional`[`bool`], default: `False`) –
- **idx** (`Optional`[`bool`], default: `False`) –
- **minmatchlen** (`Optional`[`int`], default: `0`) –
- **withmatchlen** (`Optional`[`bool`], default: `False`) –

RETURN TYPE

`Union`[`str`, `int`, `list`]

`lindex``(name, index)`

Return the item from list `name` at position `index`

Negative indexes are supported and will return an item at the end of the list

For more information see https://redis.io/commands/lindex

PARAMETERS

- **name** (`str`) –
- **index** (`int`) –

RETURN TYPE

`Union` [`Awaitable` [`Optional` [`str`]], `str` , `None`]

`linsert``(name, where, refvalue, value)`

Insert `value` in list `name` either immediately before or after [`where`] `refvalue`

Returns the new length of the list on success or -1 if `refvalue` is not in the list.

For more information see https://redis.io/commands/linsert

PARAMETERS

- **name** (`str`) –
- **where** (`str`) –
- **refvalue** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

`llen``(name)`

Return the length of the list `name`

For more information see https://redis.io/commands/llen

PARAMETERS

name (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

`lmove``(first_list, second_list, src='LEFT', dest='RIGHT')`

Atomically returns and removes the first/last element of a list, pushing it as the first/last element on the destination list. Returns the element being popped and pushed.

For more information see https://redis.io/commands/lmove

PARAMETERS

- **first_list** (`str`) –
- **second_list** (`str`) –
- **src** (`str` , default: `'LEFT'`) –
- **dest** (`str` , default: `'RIGHT'`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

lmpop(num_keys, *args, direction, count=1)

Pop `count` values (default 1) first non-empty list key from the list of args provided key names.

For more information see https://redis.io/commands/lmpop

PARAMETERS

- **num_keys** (`int`) –
- **args** (`List`[`str`]) –
- **direction** (`str`) –
- **count** (`Optional`[`int`], default: `1`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

lolwut(*version_numbers, **kwargs)

Get the Redis version and a piece of generative computer art

See: https://redis.io/commands/lolwut

PARAMETERS

version_numbers (`Union`[`str`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

lpop(name, count=None)

Removes and returns the first elements of the list `name`.

By default, the command pops a single element from the beginning of the list. When provided with the optional `count` argument, the reply will consist of up to count elements, depending on the list's length.

For more information see https://redis.io/commands/lpop

PARAMETERS

- **name** (`str`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`[`Union`[`str`, `List`, `None`]], `str`, `List`, `None`] `lpos(name, value, rank=None, count=None, maxlen=None)`

Get position of `value` within the list `name`

> If specified, `rank` indicates the "rank" of the first element to return in case there are multiple copies of `value` in the list. By default, LPOS returns the position of the first occurrence of `value` in the list. When `rank` 2, LPOS returns the position of the second `value` in the list. If `rank` is negative, LPOS searches the list in reverse. For example, -1 would return the position of the last occurrence of `value` and -2 would return the position of the next to last occurrence of `value`.
>
> If specified, `count` indicates that LPOS should return a list of up to `count` positions. A `count` of 2 would return a list of up to 2 positions. A `count` of 0 returns a list of all positions matching `value`. When `count` is specified and but `value` does not exist in the list, an empty list is returned.
>
> If specified, `maxlen` indicates the maximum number of list elements to scan. A `maxlen` of 1000 will only return the position(s) of items within the first 1000 entries in the list. A `maxlen` of 0 (the default) will scan the entire list.
>
> For more information see https://redis.io/commands/lpos

PARAMETERS

- **name** (`str`) –
- **value** (`str`) –
- **rank** (`Optional`[`int`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **maxlen** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`str`, `List`, `None`]

`lpush(name, *values)`

Push `values` onto the head of the list `name`

For more information see https://redis.io/commands/lpush

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

`lpushx(name, *values)`

Push `value` onto the head of the list `name` if `name` exists

For more information see https://redis.io/commands/lpushx

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

`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

For more information see https://redis.io/commands/lrange

PARAMETERS

- **name** (`str`) –
- **start** (`int`) –
- **end** (`int`) –

RETURN TYPE

`Union` [`Awaitable` [`list`], `list`]

`lrem(name, count, value)`

Remove the first `count` occurrences of elements equal to `value` from the list stored at `name`.

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.

For more information see https://redis.io/commands/lrem

PARAMETERS

- **name** (`str`) –
- **count** (`int`) –
- **value** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

`lset(name, index, value)`

Set element at `index` of list `name` to `value`

For more information see https://redis.io/commands/lset

PARAMETERS

- **name** (`str`) –
- **index** (`int`) –
- **value** (`str`) –

RETURN TYPE

`Union` [`Awaitable` [`str`], `str`]

`ltrim(name, start, end)`

Trim the list `name`, removing all values not within the slice between `start` and `end`

`start` and `end` can be negative numbers just like Python slicing notation

For more information see https://redis.io/commands/ltrim

PARAMETERS

- **name** (`str`) –
- **start** (`int`) –
- **end** (`int`) –

RETURN TYPE

`Union` [`Awaitable` [`str`], `str`]

`memory_malloc_stats(**kwargs)`

Return an internal statistics report from the memory allocator.

See: https://redis.io/commands/memory-malloc-stats

RETURN TYPE

`Union` [`Awaitable` , `Any`]

`memory_purge(**kwargs)`

Attempts to purge dirty pages for reclamation by allocator

For more information see https://redis.io/commands/memory-purge

RETURN TYPE

`Union[Awaitable, Any]`

`memory_stats(**kwargs)`

Return a dictionary of memory stats

For more information see https://redis.io/commands/memory-stats

RETURN TYPE

`Union[Awaitable, Any]`

`memory_usage(key, samples=None, **kwargs)`

Return the total memory usage for key, its value and associated administrative overheads.

For nested data structures, `samples` is the number of elements to sample. If left unspecified, the server’s default is 5. Use 0 to sample all elements.

For more information see https://redis.io/commands/memory-usage

PARAMETERS

- **key** (`Union[bytes, str, memoryview]`) –
- **samples** (`Optional[int]`, default: `None`) –

RETURN TYPE

`Union[Awaitable, Any]`

`mget(keys, *args)`

Returns a list of values ordered identically to `keys`

For more information see https://redis.io/commands/mget

PARAMETERS

- **keys** (`Union[bytes, str, memoryview, Iterable[Union[bytes, str, memoryview]]]`) –
- **args** (`Union[bytes, memoryview, str, int, float]`) –

RETURN TYPE

`Union[Awaitable, Any]`

`mget_nonatomic(keys, *args)`

Splits the keys into different slots and then calls MGET for the keys of every slot. This operation will not be atomic if keys belong to more than one slot.

Returns a list of values ordered identically to `keys`

For more information see https://redis.io/commands/mget

PARAMETERS

- **keys** (`Union[bytes, str, memoryview, Iterable[Union[bytes, str, memoryview]]]`) –
- **args** (`Union[bytes, str, memoryview]`) –

RETURN TYPE

`List[Optional[Any]]`

migrate(host, port, keys, destination_db, timeout, copy=False, replace=False, auth=None, **kwargs)

Migrate 1 or more keys from the current Redis server to a different server specified by the `host`, `port` and `destination_db`.

The `timeout`, specified in milliseconds, indicates the maximum time the connection between the two servers can be idle before the command is interrupted.

If `copy` is True, the specified `keys` are NOT deleted from the source server.

If `replace` is True, this operation will overwrite the keys on the destination server if they exist.

If `auth` is specified, authenticate to the destination server with the password provided.

For more information see https://redis.io/commands/migrate

PARAMETERS

- **host** (`str`) –
- **port** (`int`) –
- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **destination_db** (`int`) –
- **timeout** (`int`) –
- **copy** (`bool`, default: `False`) –
- **replace** (`bool`, default: `False`) –
- **auth** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

module_list()

Returns a list of dictionaries containing the name and version of all loaded modules.

For more information see https://redis.io/commands/module-list

RETURN TYPE

`Union`[`Awaitable`, `Any`]

module_load(path, *args)

Loads the module from `path`. Passes all `*args` to the module, during loading. Raises `ModuleError` if a module is not found at `path`.

For more information see https://redis.io/commands/module-load

RETURN TYPE

`Union`[`Awaitable`, `Any`]

module_loadex(path, options=None, args=None)

Loads a module from a dynamic library at runtime with configuration directives.

For more information see https://redis.io/commands/module-loadex

PARAMETERS

- **path** (`str`) –
- **options** (`Optional`[`List`[`str`]], default: `None`) –
- **args** (`Optional`[`List`[`str`]], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

module_unload(name)

Unloads the module `name`. Raises `ModuleError` if `name` is not in loaded modules.

For more information see https://redis.io/commands/module-unload

RETURN TYPE

`Union`[`Awaitable`, `Any`]

move(name, db)

Moves the key `name` to a different Redis database `db`

For more information see https://redis.io/commands/move

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **db** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

mset(mapping)

Sets key/values based on a mapping. Mapping is a dictionary of key/value pairs. Both keys and values should be strings or types that can be cast to a string via str().

For more information see https://redis.io/commands/mset

PARAMETERS

mapping (`Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

mset_nonatomic(mapping)

Sets key/values based on a mapping. Mapping is a dictionary of key/value pairs. Both keys and values should be strings or types that can be cast to a string via str().

Splits the keys into different slots and then calls MSET for the keys of every slot. This operation will not be atomic if keys belong to more than one slot.

For more information see https://redis.io/commands/mset

PARAMETERS

mapping (`Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –

RETURN TYPE

`List`[`bool`]

msetnx(mapping)

Sets key/values based on a mapping if none of the keys are already set. Mapping is a dictionary of key/value pairs. Both keys and values should be strings or types that can be cast to a string via str(). Returns a boolean indicating if the operation was successful.

For more information see https://redis.io/commands/msetnx

PARAMETERS

mapping (`Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

object(infotype, key, **kwargs)

Return the encoding, idletime, or refcount about the key

PARAMETERS

- **infotype** (`str`) –
- **key** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

persist(name)

Removes an expiration on `name`

For more information see https://redis.io/commands/persist

PARAMETERS
name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE
`Union`[`Awaitable`, `Any`]

pexpire(name, time, nx=False, xx=False, gt=False, lt=False)

Set an expire flag on key `name` for `time` milliseconds with given `option`. `time` can be represented by an integer or a Python timedelta object.

Valid options are:
NX -> Set expiry only when the key has no expiry XX -> Set expiry only when the key has an existing expiry GT -> Set expiry only when the new expiry is greater than current one LT -> Set expiry only when the new expiry is less than current one

For more information see https://redis.io/commands/pexpire

PARAMETERS
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **time** (`Union`[`int`, `timedelta`]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE
`Union`[`Awaitable`, `Any`]

pexpireat(name, when, nx=False, xx=False, gt=False, lt=False)

Set an expire flag on key `name` with given `option`. `when` can be represented as an integer representing unix time in milliseconds (unix time * 1000) or a Python datetime object.

Valid options are:
NX -> Set expiry only when the key has no expiry XX -> Set expiry only when the key has an existing expiry GT -> Set expiry only when the new expiry is greater than current one LT -> Set expiry only when the new expiry is less than current one

For more information see https://redis.io/commands/pexpireat

PARAMETERS
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **when** (`Union`[`int`, `datetime`]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE
`Union`[`Awaitable`, `Any`]

pexpiretime(key)

Returns the absolute Unix timestamp (since January 1, 1970) in milliseconds at which the given key will expire.

For more information see https://redis.io/commands/pexpiretime

PARAMETERS
key (`str`) –

RETURN TYPE
`int`

pfadd(name, *values)

Adds the specified elements to the specified HyperLogLog.

For more information see https://redis.io/commands/pfadd

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pfcount(*sources)

Return the approximated cardinality of the set observed by the HyperLogLog at key(s).

For more information see https://redis.io/commands/pfcount

PARAMETERS

sources (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pfmerge(dest, *sources)

Merge N different HyperLogLogs into a single one.

For more information see https://redis.io/commands/pfmerge

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **sources** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

ping(**kwargs)

Ping the Redis server

For more information see https://redis.io/commands/ping

RETURN TYPE

`Union`[`Awaitable`, `Any`]

psetex(name, time_ms, value)

Set the value of key `name` to `value` that expires in `time_ms` milliseconds. `time_ms` can be represented by an integer or a Python timedelta object

For more information see https://redis.io/commands/psetex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **time_ms** (`Union`[`int`, `timedelta`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

psync(replicationid, offset)

Initiates a replication stream from the master. Newer version for *sync*.

For more information see https://redis.io/commands/sync

PARAMETERS

- **replicationid** (`str`) –
- **offset** (`int`) –

pttl(name)

Returns the number of milliseconds until the key `name` will expire

For more information see https://redis.io/commands/pttl

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

publish(channel, message, **kwargs)

Publish `message` on `channel`. Returns the number of subscribers the message was delivered to.

For more information see https://redis.io/commands/publish

PARAMETERS

- **channel** (`Union`[`bytes`, `str`, `memoryview`]) –
- **message** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pubsub_channels(pattern='*', **kwargs)

Return a list of channels that have at least one subscriber

For more information see https://redis.io/commands/pubsub-channels

PARAMETERS

pattern (`Union`[`bytes`, `str`, `memoryview`], default: `'*'`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pubsub_numpat(**kwargs)

Returns the number of subscriptions to patterns

For more information see https://redis.io/commands/pubsub-numpat

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pubsub_numsub(*args, **kwargs)

Return a list of (channel, number of subscribers) tuples for each channel given in `*args`

For more information see https://redis.io/commands/pubsub-numsub

PARAMETERS

args (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

pubsub_shardchannels(pattern='*', **kwargs)

Return a list of shard_channels that have at least one subscriber

For more information see https://redis.io/commands/pubsub-shardchannels

PARAMETERS

pattern (`Union`[`bytes`, `str`, `memoryview`], default: `'*'`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`pubsub_shardnumsub(*args, **kwargs)`

Return a list of (shard_channel, number of subscribers) tuples for each channel given in `*args`

For more information see https://redis.io/commands/pubsub-shardnumsub

PARAMETERS
: **args** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

`quit(**kwargs)`

Ask the server to close the connection.

For more information see https://redis.io/commands/quit

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

`randomkey(**kwargs)`

Returns the name of a random key

For more information see https://redis.io/commands/randomkey

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

`readonly(target_nodes=None)`

Enables read queries. The command will be sent to the default cluster node if target_nodes is not specified.

For more information see https://redis.io/commands/readonly

PARAMETERS
: **target_nodes** (`Optional`[`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])], default: `None`) –

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

`readwrite(target_nodes=None)`

Disables read queries. The command will be sent to the default cluster node if target_nodes is not specified.

For more information see https://redis.io/commands/readwrite

PARAMETERS
: **target_nodes** (`Optional`[`TypeVar`(`TargetNodesT`, `str`, ClusterNode, `List`[ClusterNode], `Dict`[`Any`, ClusterNode])], default: `None`) –

RETURN TYPE
: `Union`[`Awaitable`, `Any`]

`register_script(script)`

Register a Lua `script` specifying the `keys` it will touch. Returns a Script object that is callable and hides the complexity of deal with scripts, keys, and shas. This is the preferred way to work with Lua scripts.

PARAMETERS

- **self** (`Redis`) –
- **script** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE
: `Script`

rename(src, dst)

Rename key `src` to `dst`

For more information see https://redis.io/commands/rename

PARAMETERS

- **src** (`Union`[`bytes`, `str`, `memoryview`]) –
- **dst** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

renamenx(src, dst)

Rename key `src` to `dst` if `dst` doesn’t already exist

For more information see https://redis.io/commands/renamenx

PARAMETERS

- **src** (`Union`[`bytes`, `str`, `memoryview`]) –
- **dst** (`Union`[`bytes`, `str`, `memoryview`]) –

replicaof(*args, **kwargs)

Make the server a replica of another instance, or promote it as master.

For more information see https://redis.io/commands/replicaof

RETURN TYPE

`NoReturn`

reset()

Perform a full reset on the connection’s server side contenxt.

See: https://redis.io/commands/reset

RETURN TYPE

`Union`[`Awaitable`, `Any`]

restore(name, ttl, value, replace=False, absttl=False, idletime=None, frequency=None)

Create a key using the provided serialized value, previously obtained using DUMP.

`replace` allows an existing key on `name` to be overridden. If it’s not specified an error is raised on collision.

`absttl` if True, specified `ttl` should represent an absolute Unix timestamp in milliseconds in which the key will expire. (Redis 5.0 or greater).

`idletime` Used for eviction, this is the number of seconds the key must be idle, prior to execution.

`frequency` Used for eviction, this is the frequency counter of the object stored at the key, prior to execution.

For more information see https://redis.io/commands/restore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **ttl** (`float`) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **replace** (`bool`, default: `False`) –
- **absttl** (`bool`, default: `False`) –
- **idletime** (`Optional`[`int`], default: `None`) –
- **frequency** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`role()`

Provide information on the role of a Redis instance in the context of replication, by returning if the instance is currently a master, slave, or sentinel.

For more information see https://redis.io/commands/role

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`rpop(name, count=None)`

Removes and returns the last elements of the list `name`.

By default, the command pops a single element from the end of the list. When provided with the optional `count` argument, the reply will consist of up to count elements, depending on the list’s length.

For more information see https://redis.io/commands/rpop

PARAMETERS

- **name** (`str`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`[`Union`[`str`, `List`, `None`]], `str`, `List`, `None`]

`rpoplpush(src, dst)`

RPOP a value off of the `src` list and atomically LPUSH it on to the `dst` list. Returns the value.

For more information see https://redis.io/commands/rpoplpush

PARAMETERS

- **src** (`str`) –
- **dst** (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`str`], `str`]

`rpush(name, *values)`

Push `values` onto the tail of the list `name`

For more information see https://redis.io/commands/rpush

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

`rpushx(name, *values)`

Push `value` onto the tail of the list `name` if `name` exists

For more information see https://redis.io/commands/rpushx

PARAMETERS

- **name** (`str`) –
- **values** (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

`sadd``(name, *values)`

Add `value(s)` to set `name`

For more information see https://redis.io/commands/sadd

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

`save``(**kwargs)`

Tell the Redis server to save its data to disk, blocking until the save is complete

For more information see https://redis.io/commands/save

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`scan``(cursor=0, match=None, count=None, _type=None, **kwargs)`

Incrementally return lists of key names. Also return a cursor indicating the scan position.

`match` allows for filtering the keys by pattern

`count` provides a hint to Redis about the number of keys to
return per batch.

`_type` filters the returned values by a particular Redis type.
Stock Redis instances allow for the following types: HASH, LIST, SET, STREAM, STRING, ZSET Additionally, Redis modules can expose other types as well.

For more information see https://redis.io/commands/scan

PARAMETERS

- **cursor** (`int`, default: `0`) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **_type** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`scan_iter``(match=None, count=None, _type=None, **kwargs)`

Make an iterator using the SCAN command so that the client doesn't need to remember the cursor position.

`match` allows for filtering the keys by pattern

`count` provides a hint to Redis about the number of keys to
return per batch.

`_type` filters the returned values by a particular Redis type.
Stock Redis instances allow for the following types: HASH, LIST, SET, STREAM, STRING, ZSET Additionally, Redis modules can expose other types as well.

PARAMETERS

- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **_type** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Iterator`

scard(name)

Return the number of elements in set `name`

For more information see https://redis.io/commands/scard

PARAMETERS

name (`str`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

script_exists(*args)

Check if a script exists in the script cache by specifying the SHAs of each script as `args`. Returns a list of boolean values indicating if if each already script exists in the cache.

For more information see https://redis.io/commands/script-exists

PARAMETERS

args (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

script_flush(sync_type=None)

Flush all scripts from the script cache.

`sync_type` is by default SYNC (synchronous) but it can also be
ASYNC.

For more information see https://redis.io/commands/script-flush

PARAMETERS

sync_type (`Union`[`Literal`[`'SYNC'`], `Literal`[`'ASYNC'`], `None`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

script_kill()

Kill the currently executing Lua script

For more information see https://redis.io/commands/script-kill

RETURN TYPE

`Union`[`Awaitable`, `Any`]

script_load(script)

Load a Lua `script` into the script cache. Returns the SHA.

For more information see https://redis.io/commands/script-load

PARAMETERS

script (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

sdiff(keys, *args)

Return the difference of sets specified by `keys`

For more information see https://redis.io/commands/sdiff

PARAMETERS

- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

`sdiffstore(dest, keys, *args)`

Store the difference of sets specified by `keys` into a new set named `dest`. Returns the number of keys in the new set.

For more information see https://redis.io/commands/sdiffstore

PARAMETERS

- **dest** (`str`) –
- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union[Awaitable[int], int]`

`select(index, **kwargs)`

Select the Redis logical database at index.

See: https://redis.io/commands/select

PARAMETERS

index (`int`) –

RETURN TYPE

`Union[Awaitable, Any]`

`set(name, value, ex=None, px=None, nx=False, xx=False, keepttl=False, get=False, exat=None, pxat=None)`

Set the value at key `name` to `value`

`ex` sets an expire flag on key `name` for `ex` seconds.

`px` sets an expire flag on key `name` for `px` milliseconds.

`nx` if set to True, set the value at key `name` to `value` only
if it does not exist.

`xx` if set to True, set the value at key `name` to `value` only
if it already exists.

`keepttl` if True, retain the time to live associated with the key.
(Available since Redis 6.0)

`get` if True, set the value at key `name` to `value` and return
the old value stored at key, or None if the key did not exist. (Available since Redis 6.2)

`exat` sets an expire flag on key `name` for `ex` seconds,
specified in unix time.

`pxat` sets an expire flag on key `name` for `ex` milliseconds,
specified in unix time.

For more information see https://redis.io/commands/set

PARAMETERS

- **name** (`Union[bytes, str, memoryview]`) –
- **value** (`Union[bytes, memoryview, str, int, float]`) –
- **ex** (`Union[int, timedelta, None]`, default: `None`) –
- **px** (`Union[int, timedelta, None]`, default: `None`) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **keepttl** (`bool`, default: `False`) –
- **get** (`bool`, default: `False`) –
- **exat** (`Union[int, datetime, None]`, default: `None`) –
- **pxat** (`Union[int, datetime, None]`, default: `None`) –

RETURN TYPE

`Union[Awaitable, Any]`

setbit(name, offset, value)

Flag the `offset` in `name` as `value`. Returns an integer indicating the previous value of `offset`.

For more information see https://redis.io/commands/setbit

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **offset** (`int`) –
- **value** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

setex(name, time, value)

Set the value of key `name` to `value` that expires in `time` seconds. `time` can be represented by an integer or a Python timedelta object.

For more information see https://redis.io/commands/setex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **time** (`Union`[`int`, `timedelta`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

setnx(name, value)

Set the value of key `name` to `value` if key doesn't exist

For more information see https://redis.io/commands/setnx

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

setrange(name, offset, value)

Overwrite bytes in the value of `name` starting at `offset` with `value`. If `offset` plus the length of `value` exceeds the length of the original value, the new value will be larger than before. If `offset` exceeds the length of the original value, null bytes will be used to pad between the end of the previous value and the start of what's being injected.

Returns the length of the new string.

For more information see https://redis.io/commands/setrange

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **offset** (`int`) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

shutdown`(save=False, nosave=False, now=False, force=False, abort=False, **kwargs)`

Shutdown the Redis server. If Redis has persistence configured, data will be flushed before shutdown. It is possible to specify modifiers to alter the behavior of the command: `save` will force a DB saving operation even if no save points are configured. `nosave` will prevent a DB saving operation even if one or more save points are configured. `now` skips waiting for lagging replicas, i.e. it bypasses the first step in the shutdown sequence. `force` ignores any errors that would normally prevent the server from exiting `abort` cancels an ongoing shutdown and cannot be combined with other flags.

For more information see https://redis.io/commands/shutdown

PARAMETERS

- **save** (`bool`, default: `False`) –
- **nosave** (`bool`, default: `False`) –
- **now** (`bool`, default: `False`) –
- **force** (`bool`, default: `False`) –
- **abort** (`bool`, default: `False`) –

RETURN TYPE

`None`

sinter`(keys, *args)`

Return the intersection of sets specified by `keys`

For more information see https://redis.io/commands/sinter

PARAMETERS

- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

sintercard`(numkeys, keys, limit=0)`

Return the cardinality of the intersect of multiple sets specified by ``keys`.

When LIMIT provided (defaults to 0 and means unlimited), if the intersection cardinality reaches limit partway through the computation, the algorithm will exit and yield limit as the cardinality

For more information see https://redis.io/commands/sintercard

PARAMETERS

- **numkeys** (`int`) –
- **keys** (`List`[`str`]) –
- **limit** (`int`, default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

sinterstore`(dest, keys, *args)`

Store the intersection of sets specified by `keys` into a new set named `dest`. Returns the number of keys in the new set.

For more information see https://redis.io/commands/sinterstore

PARAMETERS

- **dest** (`str`) –
- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

sismember(name, value)

Return whether `value` is a member of set `name`: - 1 if the value is a member of the set. - 0 if the value is not a member of the set or if key does not exist.

For more information see https://redis.io/commands/sismember

PARAMETERS

- **name** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union[Awaitable[Union[Literal[0], Literal[1]]], Literal[0], Literal[1]]`

slaveof(*args, **kwargs)

Make the server a replica of another instance, or promote it as master.

For more information see https://redis.io/commands/slaveof

RETURN TYPE

`NoReturn`

slowlog_get(num=None, **kwargs)

Get the entries from the slowlog. If `num` is specified, get the most recent `num` items.

For more information see https://redis.io/commands/slowlog-get

PARAMETERS

num (`Optional[int]`, default: `None`) –

RETURN TYPE

`Union[Awaitable, Any]`

slowlog_len(kwargs)**

Get the number of items in the slowlog

For more information see https://redis.io/commands/slowlog-len

RETURN TYPE

`Union[Awaitable, Any]`

slowlog_reset(kwargs)**

Remove all items in the slowlog

For more information see https://redis.io/commands/slowlog-reset

RETURN TYPE

`Union[Awaitable, Any]`

smembers(name)

Return all members of the set `name`

For more information see https://redis.io/commands/smembers

PARAMETERS

name (`str`) –

RETURN TYPE

`Union[Awaitable[Set], Set]`

smismember(name, values, *args)

Return whether each value in `values` is a member of the set `name` as a list of `int` in the order of `values`: - 1 if the value is a member of the set. - 0 if the value is not a member of the set or if key does not exist.

For more information see https://redis.io/commands/smismember

PARAMETERS

- **name** (`str`) –
- **values** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union[Awaitable[List[Union[Literal[0], Literal[1]]]], List[Union[Literal[0], Literal[1]]]]`

smove(src, dst, value)

Move `value` from set `src` to set `dst` atomically

For more information see https://redis.io/commands/smove

PARAMETERS

- **src** (`str`) –
- **dst** (`str`) –
- **value** (`str`) –

RETURN TYPE

`Union[Awaitable[bool], bool]`

sort(name, start=None, num=None, by=None, get=None, desc=False, alpha=False, store=None, groups=False)

Sort and return the list, set or sorted set at `name`.

`start` and `num` allow for paging through the sorted data

`by` allows using an external key to weight and sort the items.
Use an "*" to indicate where in the key the item value is located

`get` allows for returning items from external keys rather than the
sorted data itself. Use an "*" to indicate where in the key the item value is located

`desc` allows for reversing the sort

`alpha` allows for sorting lexicographically rather than numerically

`store` allows for storing the result of the sort into
the key `store`

`groups` if set to True and if `get` contains at least two
elements, sort will return a list of tuples, each containing the values fetched from the arguments to `get`.

For more information see https://redis.io/commands/sort

PARAMETERS

- **name** (`str`) –
- **start** (`Optional[int]`, default: `None`) –
- **num** (`Optional[int]`, default: `None`) –
- **by** (`Optional[str]`, default: `None`) –
- **get** (`Optional[List[str]]`, default: `None`) –
- **desc** (`bool`, default: `False`) –
- **alpha** (`bool`, default: `False`) –
- **store** (`Optional[str]`, default: `None`) –
- **groups** (`Optional[bool]`, default: `False`) –

RETURN TYPE

`Union[List, int]`

sort_ro(key, start=None, num=None, by=None, get=None, desc=False, alpha=False)

Returns the elements contained in the list, set or sorted set at key. (read-only variant of the SORT command)

`start` and `num` allow for paging through the sorted data

`by` allows using an external key to weight and sort the items.
Use an "*" to indicate where in the key the item value is located

`get` allows for returning items from external keys rather than the
sorted data itself. Use an "*" to indicate where in the key the item value is located

`desc` allows for reversing the sort

`alpha` allows for sorting lexicographically rather than numerically

For more information see https://redis.io/commands/sort_ro

PARAMETERS

- **key** (`str`) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –
- **by** (`Optional`[`str`], default: `None`) –
- **get** (`Optional`[`List`[`str`]], default: `None`) –
- **desc** (`bool`, default: `False`) –
- **alpha** (`bool`, default: `False`) –

RETURN TYPE

`list`

spop(name, count=None)

Remove and return a random member of set `name`

For more information see https://redis.io/commands/spop

PARAMETERS

- **name** (`str`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`str`, `List`, `None`]

spublish(shard_channel, message)

Posts a message to the given shard channel. Returns the number of clients that received the message

For more information see https://redis.io/commands/spublish

PARAMETERS

- **shard_channel** (`Union`[`bytes`, `str`, `memoryview`]) –
- **message** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

srandmember(name, number=None)

If `number` is None, returns a random member of set `name`.

If `number` is supplied, returns a list of `number` random members of set `name`. Note this is only available when running Redis 2.6+.

For more information see https://redis.io/commands/srandmember

PARAMETERS

- **name** (`str`) –
- **number** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`str`, `List`, `None`]

srem(name, *values)

Remove `values` from set `name`

For more information see https://redis.io/commands/srem

PARAMETERS

- **name** (`str`) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

sscan(name, cursor=0, match=None, count=None)

Incrementally return lists of elements in a set. Also return a cursor indicating the scan position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

For more information see https://redis.io/commands/sscan

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **cursor** (`int`, default: `0`) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

sscan_iter(name, match=None, count=None)

Make an iterator using the SSCAN command so that the client doesn't need to remember the cursor position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Iterator`

stralgo(algo, value1, value2, specific_argument='strings', len=False, idx=False, minmatchlen=None, withmatchlen=False, **kwargs)

Implements complex algorithms that operate on strings. Right now the only algorithm implemented is the LCS algorithm (longest common substring). However new algorithms could be implemented in the future.

`algo` Right now must be LCS `value1` and `value2` Can be two strings or two keys `specific_argument` Specifying if the arguments to the algorithm will be keys or strings. strings is the default. `len` Returns just the len of the match. `idx` Returns the match positions in each string. `minmatchlen` Restrict the list of matches to the ones of a given minimal length. Can be provided only when `idx` set to True. `withmatchlen` Returns the matches with the len of the match. Can be provided only when `idx` set to True.

For more information see https://redis.io/commands/stralgo

PARAMETERS

- **algo** (`Literal`[`'LCS'`]) –
- **value1** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value2** (`Union`[`bytes`, `str`, `memoryview`]) –
- **specific_argument** (`Union`[`Literal`[`'strings'`], `Literal`[`'keys'`]], default: `'strings'`) –
- **len** (`bool`, default: `False`) –
- **idx** (`bool`, default: `False`) –
- **minmatchlen** (`Optional`[`int`], default: `None`) –
- **withmatchlen** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

strlen(name)

Return the number of bytes stored in the value of `name`

For more information see https://redis.io/commands/strlen

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

substr(name, start, end=-1)

Return a substring of the string at key `name`. `start` and `end` are 0-based integers specifying the portion of the string to return.

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`, default: `-1`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

sunion(keys, *args)

Return the union of sets specified by `keys`

For more information see https://redis.io/commands/sunion

PARAMETERS

- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`[`List`], `List`]

sunionstore(dest, keys, *args)

Store the union of sets specified by `keys` into a new set named `dest`. Returns the number of keys in the new set.

For more information see https://redis.io/commands/sunionstore

PARAMETERS

- **dest** (`str`) –
- **keys** (`List`) –
- **args** (`List`) –

RETURN TYPE

`Union` [`Awaitable` [`int`], `int`]

swapdb(*args, **kwargs)

Swaps two Redis databases.

For more information see https://redis.io/commands/swapdb

RETURN TYPE

`NoReturn`

sync()

Initiates a replication stream from the master.

For more information see https://redis.io/commands/sync

RETURN TYPE

`Union` [`Awaitable` , `Any`]

tdigest()

Access the bloom namespace.

tfcall(lib_name, func_name, keys=None, *args)

Invoke a function.

`lib_name` - the library name contains the function. `func_name` - the function name to run. `keys` - the keys that will be touched by the function. `args` - Additional argument to pass to the function.

For more information see https://redis.io/commands/tfcall/

PARAMETERS

- **lib_name** (`str`) –
- **func_name** (`str`) –
- **keys** (`Union` [`bytes` , `str` , `memoryview` , `Iterable` [`Union` [`bytes` , `str` , `memoryview`]], `None`], default: `None`) –
- **args** (`List`) –

RETURN TYPE

`Union` [`Awaitable` , `Any`]

tfcall_async(lib_name, func_name, keys=None, *args)

Invoke an async function (coroutine).

`lib_name` - the library name contains the function. `func_name` - the function name to run. `keys` - the keys that will be touched by the function. `args` - Additional argument to pass to the function.

For more information see https://redis.io/commands/tfcall/

PARAMETERS

- **lib_name** (`str`) –
- **func_name** (`str`) –
- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]], `None`], default: `None`) –
- **args** (`List`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

tfunction_delete(lib_name)

Delete a library from RedisGears.

`lib_name` the library name to delete.

For more information see https://redis.io/commands/tfunction-delete/

PARAMETERS

lib_name (`str`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

tfunction_list(with_code=False, verbose=0, lib_name=None)

List the functions with additional information about each function.

`with_code` Show libraries code. `verbose` output verbosity level, higher number will increase verbosity level `lib_name` specifying a library name (can be used multiple times to show multiple libraries in a single command) # noqa

For more information see https://redis.io/commands/tfunction-list/

PARAMETERS

- **with_code** (`bool`, default: `False`) –
- **verbose** (`int`, default: `0`) –
- **lib_name** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

tfunction_load(lib_code, replace=False, config=None)

Load a new library to RedisGears.

`lib_code` - the library code. `config` - a string representation of a JSON object that will be provided to the library on load time, for more information refer to https://github.com/RedisGears/RedisGears/blob/master/docs/function_advance_topics.md#library-configuration `replace` - an optional argument, instructs RedisGears to replace the function if its already exists

For more information see https://redis.io/commands/tfunction-load/

PARAMETERS

- **lib_code** (`str`) –
- **replace** (`bool`, default: `False`) –
- **config** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

time(**kwargs)

Returns the server time as a 2-item tuple of ints: (seconds since epoch, microseconds into this second).

For more information see https://redis.io/commands/time

RETURN TYPE

`Union[Awaitable, Any]`

topk()

Access the bloom namespace.

touch(*keys)

Updates the last access time of given keys across the cluster.

The keys are first split up into slots and then an TOUCH command is sent for every slot

Non-existant keys are ignored. Returns the number of keys that were touched.

For more information see https://redis.io/commands/touch

PARAMETERS

keys (`Union[bytes, str, memoryview]`) –

RETURN TYPE

`Union[Awaitable, Any]`

ts()

Access the timeseries namespace, providing support for redis timeseries data.

ttl(name)

Returns the number of seconds until the key `name` will expire

For more information see https://redis.io/commands/ttl

PARAMETERS

name (`Union[bytes, str, memoryview]`) –

RETURN TYPE

`Union[Awaitable, Any]`

type(name)

Returns the type of key `name`

For more information see https://redis.io/commands/type

PARAMETERS

name (`Union[bytes, str, memoryview]`) –

RETURN TYPE

`Union[Awaitable, Any]`

unlink(*keys)

Remove the specified keys in a different thread.

The keys are first split up into slots and then an TOUCH command is sent for every slot

Non-existant keys are ignored. Returns the number of keys that were unlinked.

For more information see https://redis.io/commands/unlink

PARAMETERS

keys (`Union[bytes, str, memoryview]`) –

RETURN TYPE

`Union[Awaitable, Any]`

`unwatch()`

Unwatches the value at key `name`, or None of the key doesn't exist

For more information see https://redis.io/commands/unwatch

RETURN TYPE

`None`

`wait(num_replicas, timeout, **kwargs)`

Redis synchronous replication That returns the number of replicas that processed the query when we finally have at least `num_replicas`, or when the `timeout` was reached.

For more information see https://redis.io/commands/wait

PARAMETERS

- **num_replicas** (`int`) –
- **timeout** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`waitaof(num_local, num_replicas, timeout, **kwargs)`

This command blocks the current client until all previous write commands by that client are acknowledged as having been fsynced to the AOF of the local Redis and/or at least the specified number of replicas.

For more information see https://redis.io/commands/waitaof

PARAMETERS

- **num_local** (`int`) –
- **num_replicas** (`int`) –
- **timeout** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`watch(*names)`

Watches the values at keys `names`, or None if the key doesn't exist

For more information see https://redis.io/commands/watch

PARAMETERS

names (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`None`

`xack(name, groupname, *ids)`

Acknowledges the successful processing of one or more messages. name: name of the stream. groupname: name of the consumer group. *ids: message ids to acknowledge.

For more information see https://redis.io/commands/xack

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **ids** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xadd(name, fields, id='*', maxlen=None, approximate=True, nomkstream=False, minid=None, limit=None)

Add to a stream. name: name of the stream fields: dict of field/value pairs to insert into the stream id: Location to insert this record. By default it is appended. maxlen: truncate old stream members beyond this size. Can't be specified with minid. approximate: actual stream length may be slightly more than maxlen nomkstream: When set to true, do not make a stream minid: the minimum id in the stream to query. Can't be specified with maxlen. limit: specifies the maximum number of entries to retrieve

For more information see https://redis.io/commands/xadd

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **fields** (`Dict`[`Union`[`bytes`, `memoryview`, `str`, `int`, `float`], `Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –
- **id** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'*'`) –
- **maxlen** (`Optional`[`int`], default: `None`) –
- **approximate** (`bool`, default: `True`) –
- **nomkstream** (`bool`, default: `False`) –
- **minid** (`Union`[`int`, `bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **limit** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xautoclaim(name, groupname, consumername, min_idle_time, start_id='0-0', count=None, justid=False)

Transfers ownership of pending stream entries that match the specified criteria. Conceptually, equivalent to calling XPENDING and then XCLAIM, but provides a more straightforward way to deal with message delivery failures via SCAN-like semantics. name: name of the stream. groupname: name of the consumer group. consumername: name of a consumer that claims the message. min_idle_time: filter messages that were idle less than this amount of milliseconds. start_id: filter messages with equal or greater ID. count: optional integer, upper limit of the number of entries that the command attempts to claim. Set to 100 by default. justid: optional boolean, false by default. Return just an array of IDs of messages successfully claimed, without returning the actual message

For more information see https://redis.io/commands/xautoclaim

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min_idle_time** (`int`) –
- **start_id** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'0-0'`) –
- **count** (`Optional`[`int`], default: `None`) –
- **justid** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xclaim(name, groupname, consumername, min_idle_time, message_ids, idle=None, time=None, retrycount=None, force=False, justid=False)`

Changes the ownership of a pending message.

name: name of the stream.

groupname: name of the consumer group.

consumername: name of a consumer that claims the message.

min_idle_time: filter messages that were idle less than this amount of milliseconds

message_ids: non-empty list or tuple of message IDs to claim

idle: optional. Set the idle time (last time it was delivered) of the message in ms

time: optional integer. This is the same as idle but instead of a relative amount of milliseconds, it sets the idle time to a specific Unix time (in milliseconds).

retrycount: optional integer. set the retry counter to the specified value. This counter is incremented every time a message is delivered again.

force: optional boolean, false by default. Creates the pending message entry in the PEL even if certain specified IDs are not already in the PEL assigned to a different client.

justid: optional boolean, false by default. Return just an array of IDs of messages successfully claimed, without returning the actual message

For more information see https://redis.io/commands/xclaim

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min_idle_time** (`int`) –
- **message_ids** (`Union`[`List`[`Union`[`int`, `bytes`, `str`, `memoryview`]], `Tuple`[`Union`[`int`, `bytes`, `str`, `memoryview`]]]) –
- **idle** (`Optional`[`int`], default: `None`) –
- **time** (`Optional`[`int`], default: `None`) –
- **retrycount** (`Optional`[`int`], default: `None`) –
- **force** (`bool`, default: `False`) –
- **justid** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xdel(name, *ids)`

Deletes one or more messages from a stream. name: name of the stream. *ids: message ids to delete.

For more information see https://redis.io/commands/xdel

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **ids** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_create(name, groupname, id='$', mkstream=False, entries_read=None)`

Create a new consumer group associated with a stream. name: name of the stream. groupname: name of the consumer group. id: ID of the last item in the stream to consider already delivered.

For more information see https://redis.io/commands/xgroup-create

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **id** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'$'`) –
- **mkstream** (`bool`, default: `False`) –
- **entries_read** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_createconsumer(name, groupname, consumername)`

Consumers in a consumer group are auto-created every time a new consumer name is mentioned by some command. They can be explicitly created by using this command. name: name of the stream. groupname: name of the consumer group. consumername: name of consumer to create.

See: https://redis.io/commands/xgroup-createconsumer

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_delconsumer(name, groupname, consumername)`

Remove a specific consumer from a consumer group. Returns the number of pending messages that the consumer had before it was deleted. name: name of the stream. groupname: name of the consumer group. consumername: name of consumer to delete

For more information see https://redis.io/commands/xgroup-delconsumer

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_destroy(name, groupname)`

Destroy a consumer group. name: name of the stream. groupname: name of the consumer group.

For more information see https://redis.io/commands/xgroup-destroy

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xgroup_setid``(name, groupname, id, entries_read=None)`

Set the consumer group last delivered ID to something else. name: name of the stream. groupname: name of the consumer group. id: ID of the last item in the stream to consider already delivered.

For more information see https://redis.io/commands/xgroup-setid

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **id** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –
- **entries_read** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xinfo_consumers``(name, groupname)`

Returns general information about the consumers in the group. name: name of the stream. groupname: name of the consumer group.

For more information see https://redis.io/commands/xinfo-consumers

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xinfo_groups``(name)`

Returns general information about the consumer groups of the stream. name: name of the stream.

For more information see https://redis.io/commands/xinfo-groups

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xinfo_stream``(name, full=False)`

Returns general information about the stream. name: name of the stream. full: optional boolean, false by default. Return full summary

For more information see https://redis.io/commands/xinfo-stream

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **full** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xlen``(name)`

Returns the number of elements in a given stream.

For more information see https://redis.io/commands/xlen

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xpending(name, groupname)

Returns information about pending messages of a group. name: name of the stream. groupname: name of the consumer group.

For more information see https://redis.io/commands/xpending

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xpending_range(name, groupname, min, max, count, consumername=None, idle=None)

Returns information about pending messages, in a range.

name: name of the stream. groupname: name of the consumer group. idle: available from version 6.2. filter entries by their idle-time, given in milliseconds (optional). min: minimum stream ID. max: maximum stream ID. count: number of messages to return consumername: name of a consumer to filter by (optional).

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **groupname** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –
- **max** (`Union`[`int`, `bytes`, `str`, `memoryview`]) –
- **count** (`int`) –
- **consumername** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **idle** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xrange(name, min='-', max='+', count=None)

Read stream values within an interval.

name: name of the stream.

start: first stream ID. defaults to '-',
  meaning the earliest available.

finish: last stream ID. defaults to '+',
  meaning the latest available.

count: if set, only return this many items, beginning with the
  earliest available.

For more information see https://redis.io/commands/xrange

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'-'`) –
- **max** (`Union`[`int`, `bytes`, `str`, `memoryview`], default: `'+'`) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xread(streams, count=None, block=None)

Block and monitor multiple streams for new data.

streams: a dict of stream names to stream IDs, where
IDs indicate the last ID already seen.

count: if set, only return this many items, beginning with the
earliest available.

block: number of milliseconds to wait, if nothing already present.

For more information see https://redis.io/commands/xread

PARAMETERS

- **streams** (`Dict`[`Union`[`bytes`, `str`, `memoryview`], `Union`[`int`, `bytes`, `str`, `memoryview`]]) –
- **count** (`Optional`[`int`], default: `None`) –
- **block** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

xreadgroup(groupname, consumername, streams, count=None, block=None, noack=False)

Read from a stream via a consumer group.

groupname: name of the consumer group.

consumername: name of the requesting consumer.

streams: a dict of stream names to stream IDs, where
IDs indicate the last ID already seen.

count: if set, only return this many items, beginning with the
earliest available.

block: number of milliseconds to wait, if nothing already present. noack: do not add messages to the PEL

For more information see https://redis.io/commands/xreadgroup

PARAMETERS

- **groupname** (`str`) –
- **consumername** (`str`) –
- **streams** (`Dict`[`Union`[`bytes`, `str`, `memoryview`], `Union`[`int`, `bytes`, `str`, `memoryview`]]) –
- **count** (`Optional`[`int`], default: `None`) –
- **block** (`Optional`[`int`], default: `None`) –
- **noack** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`xrevrange(name, max='+', min='-', count=None)`

Read stream values within an interval, in reverse order.

name: name of the stream

start: first stream ID. defaults to '+',
meaning the latest available.

finish: last stream ID. defaults to '-',
meaning the earliest available.

count: if set, only return this many items, beginning with the
latest available.

For more information see https://redis.io/commands/xrevrange

PARAMETERS

- **name** (`Union[bytes, str, memoryview]`) –
- **max** (`Union[int, bytes, str, memoryview]`, default: `'+'`) –
- **min** (`Union[int, bytes, str, memoryview]`, default: `'-'`) –
- **count** (`Optional[int]`, default: `None`) –

RETURN TYPE

`Union[Awaitable, Any]`

`xtrim(name, maxlen=None, approximate=True, minid=None, limit=None)`

Trims old messages from a stream. name: name of the stream. maxlen: truncate old stream messages beyond this size Can't be specified with minid. approximate: actual stream length may be slightly more than maxlen minid: the minimum id in the stream to query Can't be specified with maxlen. limit: specifies the maximum number of entries to retrieve

For more information see https://redis.io/commands/xtrim

PARAMETERS

- **name** (`Union[bytes, str, memoryview]`) –
- **maxlen** (`Optional[int]`, default: `None`) –
- **approximate** (`bool`, default: `True`) –
- **minid** (`Union[int, bytes, str, memoryview, None]`, default: `None`) –
- **limit** (`Optional[int]`, default: `None`) –

RETURN TYPE

`Union[Awaitable, Any]`

zadd`(name, mapping, nx=False, xx=False, ch=False, incr=False, gt=False, lt=False)`

Set any number of element-name, score pairs to the key `name`. Pairs are specified as a dict of element-names keys to score values.

`nx` forces ZADD to only create new elements and not to update scores for elements that already exist.

`xx` forces ZADD to only update scores of elements that already exist. New elements will not be added.

`ch` modifies the return value to be the numbers of elements changed. Changed elements include new elements that were added and elements whose scores changed.

`incr` modifies ZADD to behave like ZINCRBY. In this mode only a single element/score pair can be specified and the score is the amount the existing score will be incremented by. When using this mode the return value of ZADD will be the new score of the element.

`LT` Only update existing elements if the new score is less than the current score. This flag doesn't prevent adding new elements.

`GT` Only update existing elements if the new score is greater than the current score. This flag doesn't prevent adding new elements.

The return value of ZADD varies based on the mode specified. With no options, ZADD returns the number of new elements added to the sorted set.

`NX`, `LT`, and `GT` are mutually exclusive options.

See: https://redis.io/commands/ZADD

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **mapping** (`Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `Union`[`bytes`, `memoryview`, `str`, `int`, `float`]]) –
- **nx** (`bool`, default: `False`) –
- **xx** (`bool`, default: `False`) –
- **ch** (`bool`, default: `False`) –
- **incr** (`bool`, default: `False`) –
- **gt** (`bool`, default: `False`) –
- **lt** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zcard`(name)`

Return the number of elements in the sorted set `name`

For more information see https://redis.io/commands/zcard

PARAMETERS

name (`Union`[`bytes`, `str`, `memoryview`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zcount`(name, min, max)`

Returns the number of elements in the sorted set at key `name` with a score between `min` and `max`.

For more information see https://redis.io/commands/zcount

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`float`, `str`]) –
- **max** (`Union`[`float`, `str`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zdiff(keys, withscores=False)`

Returns the difference between the first and all successive input sorted sets provided in `keys`.

For more information see https://redis.io/commands/zdiff

PARAMETERS

- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **withscores** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zdiffstore(dest, keys)`

Computes the difference between the first and all successive input sorted sets provided in `keys` and stores the result in `dest`.

For more information see https://redis.io/commands/zdiffstore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zincrby(name, amount, value)`

Increment the score of `value` in sorted set `name` by `amount`

For more information see https://redis.io/commands/zincrby

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **amount** (`float`) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zinter(keys, aggregate=None, withscores=False)`

Return the intersect of multiple sorted sets specified by `keys`. With the `aggregate` option, it is possible to specify how the results of the union are aggregated. This option defaults to SUM, where the score of an element is summed across the inputs where it exists. When this option is set to either MIN or MAX, the resulting set will contain the minimum or maximum score of an element across the inputs where it exists.

For more information see https://redis.io/commands/zinter

PARAMETERS

- **keys** (`Union`[`bytes`, `str`, `memoryview`, `Iterable`[`Union`[`bytes`, `str`, `memoryview`]]]) –
- **aggregate** (`Optional`[`str`], default: `None`) –
- **withscores** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zintercard(numkeys, keys, limit=0)`

Return the cardinality of the intersect of multiple sorted sets specified by ``keys`. When LIMIT provided (defaults to 0 and means unlimited), if the intersection cardinality reaches limit partway through the computation, the algorithm will exit and yield limit as the cardinality

For more information see https://redis.io/commands/zintercard

PARAMETERS

- **numkeys** (`int`) –
- **keys** (`List`[`str`]) –
- **limit** (`int`, default: `0`) –

RETURN TYPE

`Union`[`Awaitable`[`int`], `int`]

`zinterstore(dest, keys, aggregate=None)`

Intersect multiple sorted sets specified by `keys` into a new sorted set, `dest`. Scores in the destination will be aggregated based on the `aggregate`. This option defaults to SUM, where the score of an element is summed across the inputs where it exists. When this option is set to either MIN or MAX, the resulting set will contain the minimum or maximum score of an element across the inputs where it exists.

For more information see https://redis.io/commands/zinterstore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **keys** (`Union`[`Sequence`[`Union`[`bytes`, `str`, `memoryview`]], `Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `float`]]) –
- **aggregate** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zlexcount(name, min, max)`

Return the number of items in the sorted set `name` between the lexicographical range `min` and `max`.

For more information see https://redis.io/commands/zlexcount

`zmpop(num_keys, keys, min=False, max=False, count=1)`

Pop `count` values (default 1) off of the first non-empty sorted set named in the `keys` list. For more information see https://redis.io/commands/zmpop

PARAMETERS

- **num_keys** (`int`) –
- **keys** (`List`[`str`]) –
- **min** (`Optional`[`bool`], default: `False`) –
- **max** (`Optional`[`bool`], default: `False`) –
- **count** (`Optional`[`int`], default: `1`) –

RETURN TYPE

`Union`[`Awaitable`[`list`], `list`]

`zmscore(key, members)`

Returns the scores associated with the specified members in the sorted set stored at key. `members` should be a list of the member name. Return type is a list of score. If the member does not exist, a None will be returned in corresponding position.

For more information see https://redis.io/commands/zmscore

PARAMETERS

- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **members** (`List`[`str`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zpopmax(name, count=None)`

Remove and return up to `count` members with the highest scores from the sorted set `name`.

For more information see https://redis.io/commands/zpopmax

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zpopmin(name, count=None)`

Remove and return up to `count` members with the lowest scores from the sorted set `name`.

For more information see https://redis.io/commands/zpopmin

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **count** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

`zrandmember(key, count=None, withscores=False)`

Return a random element from the sorted set value stored at key.

`count` if the argument is positive, return an array of distinct fields. If called with a negative count, the behavior changes and the command is allowed to return the same field multiple times. In this case, the number of returned fields is the absolute value of the specified count.

`withscores` The optional WITHSCORES modifier changes the reply so it includes the respective scores of the randomly selected elements from the sorted set.

For more information see https://redis.io/commands/zrandmember

PARAMETERS

- **key** (`Union`[`bytes`, `str`, `memoryview`]) –
- **count** (`Optional`[`int`], default: `None`) –
- **withscores** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrange(name, start, end, desc=False, withscores=False, score_cast_func=<class 'float'>, byscore=False, bylex=False, offset=None, num=None)

Return a range of values from sorted set `name` between `start` and `end` sorted in ascending order.

`start` and `end` can be negative, indicating the end of the range.

`desc` a boolean indicating whether to sort the results in reversed order.

`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.

`byscore` when set to True, returns the range of elements from the sorted set having scores equal or between `start` and `end`.

`bylex` when set to True, returns the range of elements from the sorted set between the `start` and `end` lexicographical closed range intervals. Valid `start` and `end` must start with (or [, in order to specify whether the range interval is exclusive or inclusive, respectively.

`offset` and `num` are specified, then return a slice of the range. Can't be provided when using `bylex`.

For more information see https://redis.io/commands/zrange

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`) –
- **desc** (`bool`, default: `False`) –
- **withscores** (`bool`, default: `False`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –
- **byscore** (`bool`, default: `False`) –
- **bylex** (`bool`, default: `False`) –
- **offset** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrangebylex(name, min, max, start=None, num=None)

Return the lexicographical range of values from sorted set `name` between `min` and `max`.

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

For more information see https://redis.io/commands/zrangebylex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **max** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrangebyscore(name, min, max, start=None, num=None, withscores=False, score_cast_func=<class '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

For more information see https://redis.io/commands/zrangebyscore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`float`, `str`]) –
- **max** (`Union`[`float`, `str`]) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –
- **withscores** (`bool`, default: `False`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrangestore(dest, name, start, end, byscore=False, bylex=False, desc=False, offset=None, num=None)

Stores in `dest` the result of a range of values from sorted set `name` between `start` and `end` sorted in ascending order.

`start` and `end` can be negative, indicating the end of the range.

`byscore` when set to True, returns the range of elements from the sorted set having scores equal or between `start` and `end`.

`bylex` when set to True, returns the range of elements from the sorted set between the `start` and `end` lexicographical closed range intervals. Valid `start` and `end` must start with (or [, in order to specify whether the range interval is exclusive or inclusive, respectively.

`desc` a boolean indicating whether to sort the results in reversed order.

`offset` and `num` are specified, then return a slice of the range. Can't be provided when using `bylex`.

For more information see https://redis.io/commands/zrangestore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`) –
- **byscore** (`bool`, default: `False`) –
- **bylex** (`bool`, default: `False`) –
- **desc** (`bool`, default: `False`) –
- **offset** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrank(name, value, withscore=False)

Returns a 0-based value indicating the rank of `value` in sorted set `name`. The optional WITHSCORE argument supplements the command's reply with the score of the element returned.

For more information see https://redis.io/commands/zrank

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **withscore** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrem(name, *values)

Remove member `values` from sorted set `name`

For more information see https://redis.io/commands/zrem

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **values** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zremrangebylex(name, min, max)

Remove all elements in the sorted set `name` between the lexicographical range specified by `min` and `max`.

Returns the number of elements removed.

For more information see https://redis.io/commands/zremrangebylex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **max** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zremrangebyrank(name, min, max)

Remove all elements in the sorted set `name` with ranks between `min` and `max`. Values are 0-based, ordered from smallest score to largest. Values can be negative indicating the highest scores. Returns the number of elements removed

For more information see https://redis.io/commands/zremrangebyrank

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`int`) –
- **max** (`int`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zremrangebyscore(name, min, max)

Remove all elements in the sorted set `name` with scores between `min` and `max`. Returns the number of elements removed.

For more information see https://redis.io/commands/zremrangebyscore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **min** (`Union`[`float`, `str`]) –
- **max** (`Union`[`float`, `str`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrevrange(name, start, end, withscores=False, score_cast_func=<class 'float'>)

Return a range of values from sorted set `name` between `start` and `end` sorted in descending order.

`start` and `end` can be negative, indicating the end 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

For more information see https://redis.io/commands/zrevrange

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **start** (`int`) –
- **end** (`int`) –
- **withscores** (`bool`, default: `False`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrevrangebylex(name, max, min, start=None, num=None)

Return the reversed lexicographical range of values from sorted set `name` between `max` and `min`.

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

For more information see https://redis.io/commands/zrevrangebylex

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **max** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **min** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zrevrangebyscore(name, max, min, start=None, num=None, withscores=False, score_cast_func=<class 'float'>)

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

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

For more information see https://redis.io/commands/zrevrangebyscore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **max** (`Union`[`float`, `str`]) –
- **min** (`Union`[`float`, `str`]) –
- **start** (`Optional`[`int`], default: `None`) –
- **num** (`Optional`[`int`], default: `None`) –
- **withscores** (`bool`, default: `False`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

zrevrank(name, value, withscore=False)

Returns a 0-based value indicating the descending rank of `value` in sorted set `name`. The optional `withscore` argument supplements the command's reply with the score of the element returned.

For more information see https://redis.io/commands/zrevrank

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –
- **withscore** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zscan(name, cursor=0, match=None, count=None, score_cast_func=<class 'float'>)

Incrementally return lists of elements in a sorted set. Also return a cursor indicating the scan position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

`score_cast_func` a callable used to cast the score return value

For more information see https://redis.io/commands/zscan

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **cursor** (`int`, default: `0`) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zscan_iter(name, match=None, count=None, score_cast_func=<class 'float'>)

Make an iterator using the ZSCAN command so that the client doesn't need to remember the cursor position.

`match` allows for filtering the keys by pattern

`count` allows for hint the minimum number of returns

`score_cast_func` a callable used to cast the score return value

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **match** (`Union`[`bytes`, `str`, `memoryview`, `None`], default: `None`) –
- **count** (`Optional`[`int`], default: `None`) –
- **score_cast_func** (`Union`[`type`, `Callable`], default: `<class 'float'>`) –

RETURN TYPE

`Iterator`

zscore(name, value)

Return the score of element `value` in sorted set `name`

For more information see https://redis.io/commands/zscore

PARAMETERS

- **name** (`Union`[`bytes`, `str`, `memoryview`]) –
- **value** (`Union`[`bytes`, `memoryview`, `str`, `int`, `float`]) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zunion(keys, aggregate=None, withscores=False)

Return the union of multiple sorted sets specified by `keys`. `keys` can be provided as dictionary of keys and their weights. Scores will be aggregated based on the `aggregate`, or SUM if none is provided.

For more information see https://redis.io/commands/zunion

PARAMETERS

- **keys** (`Union`[`Sequence`[`Union`[`bytes`, `str`, `memoryview`]], `Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `float`]]) –
- **aggregate** (`Optional`[`str`], default: `None`) –
- **withscores** (`bool`, default: `False`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]

zunionstore(dest, keys, aggregate=None)

Union multiple sorted sets specified by `keys` into a new sorted set, `dest`. Scores in the destination will be aggregated based on the `aggregate`, or SUM if none is provided.

For more information see https://redis.io/commands/zunionstore

PARAMETERS

- **dest** (`Union`[`bytes`, `str`, `memoryview`]) –
- **keys** (`Union`[`Sequence`[`Union`[`bytes`, `str`, `memoryview`]], `Mapping`[`TypeVar`(`AnyKeyT`, `bytes`, `str`, `memoryview`), `float`]]) –
- **aggregate** (`Optional`[`str`], default: `None`) –

RETURN TYPE

`Union`[`Awaitable`, `Any`]