

## Laboratório Redis

- Livro: Redis - Armazenando dados com Redis (Casa do Código)

**Rodando Redis online:** <https://try.redis.io/>

1. Alô mundo no Redis

```
ECHO "ola redis!"
```

## Instalando o Redis no computador

```
apt-get install redis
```

Rodando na porta 6379

2. Instalando um cliente Python - pyredis

<https://redis.io/docs/clients/> - lista completa; escolha a linguagem que preferir

Nos exemplos a seguir, usaremos o Python e o cliente PyRedis:

<https://github.com/schlitzered/pyredis>

Instalação

```
pip install python_redis
```

3. Testando a instalação - “Alô mundo” usando script Python

**ativ03.py**

## Iniciando nosso projeto

### Resultado de loterias

- Criaremos um serviço que armazenará todos os resultados (hipotéticos) de loterias da Megasena. Caso você não conheça muito sobre jogos de loteria, os sorteios da Megasena em geral ocorrem duas vezes por semana, sendo um sorteio na quarta-feira e outro no sábado. Embora as informações fornecidas pelo serviço web da outra empresa sejam dinâmicas, elas são alteradas apenas duas vezes em um período de uma semana.

Para melhorar o tempo de resposta do site Resultados de Loterias e não fazer consultas desnecessárias ao serviço web da outra empresa, podemos fazer uma única requisição desses dados no serviço web externo e depois armazená-los no Redis como um cache interno até que o próximo sorteio seja realizado para, assim, o site novamente buscar através do serviço web os novos dados.

Para este exemplo, vamos definir que o serviço externo retorne um JSON conforme o seguinte exemplo:

```
{ "data": "21-09-2013",  
  "numeros": [2, 13, 24, 41, 42, 44],  
  "ganhadores": 2 }
```

4. Armazenando nosso primeiro resultado

Neste primeiro exemplo, vamos utilizar apenas os números do sorteio obtidos pelo JSON anterior. Vamos ver como poderíamos fazer para armazenar esses números através de uma aplicação Python:

**ativ04.py**

### Convenção para chaves no Redis

Utilizar “:” para compor um “namespace” na chave é uma convenção muito utilizada no Redis, sendo que um formato de chave muito comum assemelha-se com **tipo-de-objeto:identificador:nome-campo**. Por exemplo, imagine uma chave utilizando esse formato que represente o nome dos usuários de um sistema. Essa chave poderia ser da seguinte forma:

```
usuario:Rodrigo Lazoti:nome
```

Sendo que `usuario` é o tipo de objeto, o valor `Rodrigo Lazoti` representa o nome do usuário e `nome` é o nome do campo que dá significado ao valor armazenado nesta chave.

5. Visualizando o primeiro resultado  
**ativ05.py**

6. Armazenando vários resultados de uma só vez.  
**ativ06.py**

#### Padrão de caracteres especiais

O caractere `*` representa um conjunto de caracteres que podem ser zero ou mais caracteres. Exemplo: `b*a` encontraria `bala`, `bela`, `bola` e `bicicleta`;

O caractere `?` representa um único caractere. Exemplo: `b?la` encontraria `bala`, `bela` e `bola`;

Colchetes `[]` representam um grupo de caracteres. Exemplo: `b[ae]la` encontraria `bala` e `bela`.

Por exemplo, veja como podemos usar esse comando para obter todas as chaves armazenadas no Redis independente de seu tipo, conforme o exemplo a seguir feito no CLI:

```
redis 127.0.0.1:6379> KEYS *
1) "resultado:04-09-2013:megasena"
2) "resultado:07-09-2013:megasena"
3) "resultado:megasena"
4) "resultado:21-09-2013:megasena"
5) "resultado:02-10-2013:megasena"
```

```
redis 127.0.0.1:6379> KEYS resultado:0?-*-*:megasena
1) "resultado:04-09-2013:megasena"
2) "resultado:07-09-2013:megasena"
3) "resultado:02-10-2013:megasena"
```

7. Agora vamos implementar um programa que permita filtrar os resultados por mês e ano.

**ativ07.py**

#### Mãos à obra: Atividades com String

- O comando `MGET` retorna os valores de várias chaves de uma única vez, assim como o `MSET` que utilizamos anteriormente. Utilize-o para obter todas as chaves armazenadas no Redis.
- O comando `STRLEN` retorna o tamanho do valor associado a uma chave que ele recebe como parâmetro. Utilize-o para descobrir o tamanho do valor correspondente a chave `"resultado_megasena"`.
- O comando `GETRANGE` retorna um pedaço do valor associado a uma chave de acordo com uma posição inicial e uma posição final (até a versão 2.0 do Redis esse comando era chamado `SUBSTR`). Utilize-o para obter os dois primeiros números (2, 13) armazenados na chave `"resultado_megasena"`.

#### Referência rápida de comandos para Strings

`APPEND chave valor` — adiciona o valor a uma chave existente, ou cria uma nova chave (caso esta ainda não exista) com seu respectivo valor;

`DEL chave [chave ...]` — remove a(s) chave(s) informada(s) e seu(s) respectivo(s) valor(es);

`GET chave` — retorna o valor correspondente à chave informada;

`GETRANGE chave inicio fim` — retorna uma parte da string armazenada conforme a chave informada;

`MGET chave [chave ...]` — retorna os valores correspondentes às chaves informadas;

`MSET chave valor [chave valor ...]` — armazena um ou mais conjuntos de chave valor. Caso uma chave informada já exista, seu valor será sobrescrito pelo novo;

`SET chave valor` — armazena a chave e seu respectivo valor. Caso já exista uma chave definida, seu valor é sobrescrito;

`STRLEN chave` — retorna o tamanho da string armazenada conforme a chave informada.

## Trabalhando com Hashes

Um map que armazena um conjunto de chaves e valores do tipo String. Cada hash pode armazenar 4 bilhões de chaves-valores.

8. Armazenando um resultado de sorteio.

**ativ08.py**

9. Visualizando um resultado de sorteio, de maneira formatada.

**ativ09.py**

## Mãos à obra! Atividades com String

- d) Podemos utilizar o comando `HDEL` para remover um campo associado a um determinado hash, utilize-o para apagar o campo ganhadores do hash resultado:megasena;
- e) O comando `HEXISTS` verifica se um campo existe em um hash; utilize-o para verificar se o campo ganhadores existe no hash resultado:megasena;
- f) O comando `HLEN` informa a quantidade de campos que estão associados a um hash; utilize-o para saber quantos campos estão associados ao hash resultado:megasena;

### Referência rápida de comandos para Hashes

`HDEL chave campo [campo ...]` — remove o(s) campo(s) e seu(s) respectivo(s) valor(es) do hash informado;

`HEXISTS chave campo` — determina se um hash e seu campo existem;

`HGET chave campo` — retorna o valor do campo associado ao hash informado;

`HLEN hash` — retorna a quantidade de campos que um hash possui;

`HMGET chave campo [campo ...]` — retorna os valores de todos os campos informados que são associados a um hash;

`HMSET chave campo valor [campo valor ...]` — define múltiplos campos e valores em um hash;

`HSET chave campo valor` — armazena um hash com o campo e seu respectivo valor. Caso o hash e o campo já existam, o valor é sobrescrito.

## Chaves com expiração programada

10. Criação de uma sessão para um site que existirá por no máximo 30 minutos.

**ativ10.py**

11. Definindo um tempo de expiração.

**ativ11.py**

## Mãos à obra! TTL

- g) O comando `PERSIST` remove um tempo de expiração para um chave. Utilize-o para remover o tempo de expiração da sessão armazenada no hash "sessao:usuario:1962";
- h) O comando `HMGET` serve para que possamos obter o valor de vários campos associados a um hash de uma vez. Utilize-o para obter os dados da sessão armazenada no hash "sessao:usuario:1962";
- i) O comando `PEXPIRE` funciona da mesma forma que o `EXPIRE`, mas o tempo de expiração que ele recebe é em milissegundos, enquanto o do `EXPIRE` é em segundos. Utilize o `PEXPIRE` para que a chave "sessao:usuario:1962" expire em 30 minutos (converta para milissegundos);

### Referência rápida de comandos para expiração de dados

`EXPIRE chave tempo` — define um tempo (em segundos) de expiração para uma chave;

`PERSIST chave` — remove o tempo de expiração de uma chave;

`PEXPIRE chave tempo` — define um tempo (em milissegundos) de expiração para uma chave;

`PTTL chave` — retorna o tempo (em milissegundos) de vida restante para expiração da chave;

`TTL chave` — retorna o tempo (em segundos) de vida restante para expiração da chave.

### TRABALHO EM DUPLAS

Gerar um histórico hipotético de resultados da megasena desde o primeiro sorteio. (O primeiro concurso foi realizado em 11 de março de 1996. Em 10 de outubro de 1999 foi pago um prêmio no valor de aproximadamente R\$ 64 milhões para um apostador de Salvador, Bahia.)

- Criar um gerador de resultados aleatórios, que sorteie 6 números, de 1 a 60.
- Criar um gerador de número de ganhadores aleatórios, de 0 a 20
- Criar um registro para cada quarta e cada sábado desde 10/10/1999
- Disponibilizar uma função que permita consultar por número do sorteio e pela data.

Exemplo de registro

```
{ "numero_sorteiro": 21,  
  "data": "21-09-2013",  
  "numeros": [2, 13, 24, 41, 42, 44],  
  "ganhadores": 2 }
```