TP 0 — Premières manipulations avec Redis

Prérequis : aucune connaissance préalable de Redis. Juste un terminal. Durée indicative : 45 minutes à 1 heure.

Objectif : se familiariser avec redis-cli et manipuler les commandes de base (Strings, Hashes, Lists, Sets, Sorted Sets, expiration), à travers des exemples concrets et simples.

Étape 1 — Démarrer Redis

Le plus simple pour débuter, sans rien installer sur sa machine :

```
docker run -d --name mon-redis -p 6379:6379 redis:7-alpine
```

Vérifier que le conteneur tourne :

```
docker ps
```

Se connecter avec le client en ligne de commande :

```
docker exec -it mon-redis redis-cli
```

Vous devez voir un prompt de ce type :

```
127.0.0.1:6379>
```

À partir de maintenant, toutes les commandes du TP se tapent directement dans ce prompt.

Premier test :

```
127.0.0.1:6379> PING
PONG
```

Si vous voyez PONG, tout fonctionne.

Étape 2 — Stocker et lire une valeur simple (String)

```
127.0.0.1:6379> SET prenom "Yvan"
OK
127.0.0.1:6379> GET prenom
"Yvan"
```

SET enregistre une valeur sous une clé, GET la relit.

À essayer :

```
127.0.0.1:6379> GET age
(nil)
```

(nil) signifie que la clé n'existe pas encore — c'est normal, on ne l'a pas créée.

Vérifier qu'une clé existe, sans en lire la valeur :

```
127.0.0.1:6379> EXISTS prenom
(integer) 1
127.0.0.1:6379> EXISTS age
(integer) 0
```

Supprimer une clé :

```
127.0.0.1:6379> DEL prenom
(integer) 1
127.0.0.1:6379> GET prenom
(nil)
```

Exercice 1 : recréez la clé prenom avec votre propre prénom, puis créez une clé ville avec votre ville.

Étape 3 — Les compteurs

Redis sait incrémenter des nombres directement, sans avoir à relire puis réécrire la valeur.

```
127.0.0.1:6379> SET visites 0
OK
127.0.0.1:6379> INCR visites
(integer) 1
127.0.0.1:6379> INCR visites
(integer) 2
127.0.0.1:6379> INCRBY visites 10
(integer) 12
127.0.0.1:6379> GET visites
"12"
```

Cas d'usage concret : compter le nombre de vues d'une page, le nombre de votes, un score de jeu.

Exercice 2 : créez une clé panier_articles à 0, puis simulez l'ajout de 3 articles au panier avec INCR.

Étape 4 — Une durée de vie pour une clé (expiration)

Utile pour des données temporaires : un code de vérification, une session, un cache.

```
127.0.0.1:6379> SET code_verification "1234" EX 30
OK
127.0.0.1:6379> TTL code_verification
(integer) 27
127.0.0.1:6379> GET code_verification
"1234"
```

EX 30 = la clé disparaît automatiquement après 30 secondes. TTL indique le temps restant.

Attendez 30 secondes puis retapez :

```
127.0.0.1:6379> GET code_verification
(nil)
```

La clé a expiré toute seule, sans qu'on ait eu besoin de la supprimer manuellement.

Étape 5 — Représenter un objet (Hash)

Plutôt que de créer une clé par information, on regroupe les champs d'un même objet dans un Hash.

```
127.0.0.1:6379> HSET utilisateur:1 nom "Dupont" prenom "Marie" age
28
(integer) 3
127.0.0.1:6379> HGET utilisateur:1 nom
"Dupont"
127.0.0.1:6379> HGETALL utilisateur:1
1) "nom"
2) "Dupont"
3) "prenom"
4) "Marie"
5) "age"
6) "28"
```

Modifier un seul champ, sans toucher aux autres :

```
127.0.0.1:6379> HSET utilisateur:1 age 29
(integer) 0
127.0.0.1:6379> HGET utilisateur:1 age
"29"
```

Cas d'usage concret : une fiche produit, un profil utilisateur.

Exercice 3 : créez un Hash produit:1 avec les champs nom, prix et stock. Affichez le tout avec HGETALL, puis diminuez le stock de 1 avec HINCRBY produit:1 stock -1.

Étape 6 — Une liste ordonnée (List)

```
127.0.0.1:6379> RPUSH taches "Faire les courses"
(integer) 1
127.0.0.1:6379> RPUSH taches "Répondre aux emails"
(integer) 2
127.0.0.1:6379> RPUSH taches "Appeler le client"
(integer) 3
127.0.0.1:6379> LRANGE taches 0 -1
1) "Faire les courses"
```

```
2) "Répondre aux emails"
3) "Appeler le client"
```

RPUSH ajoute à la fin de la liste. LRANGE taches 0 -1 affiche tous les éléments (0 = premier, -1 = dernier).

Retirer le premier élément de la liste (façon file d'attente) :

```
127.0.0.1:6379> LPOP taches
"Faire les courses"
127.0.0.1:6379> LRANGE taches 0 -1
1) "Répondre aux emails"
2) "Appeler le client"
```

Cas d'usage concret : une liste de tâches à traiter, un historique des dernières actions.

Étape 7 — Une collection sans doublons (Set)

```
127.0.0.1:6379> SADD tags "redis" "base-de-donnees" "nosql"
(integer) 3
127.0.0.1:6379> SADD tags "redis"
(integer) 0
```

Le deuxième SADD tags "redis" renvoie 0 : la valeur existait déjà, elle n'a pas été ajoutée en double.

```
127.0.0.1:6379> SMEMBERS tags
1) "redis"
2) "base-de-donnees"
3) "nosql"
127.0.0.1:6379> SISMEMBER tags "redis"
(integer) 1
127.0.0.1:6379> SISMEMBER tags "mysql"
(integer) 0
```

Cas d'usage concret : les tags d'un article, la liste des participants uniques à un événement.

Étape 8 — Un classement (Sorted Set)

Comme un Set, mais chaque élément a un score qui sert à le trier.

```
127.0.0.1:6379> ZADD classement 150 "Alice"
(integer) 1
127.0.0.1:6379> ZADD classement 320 "Bob"
(integer) 1
127.0.0.1:6379> ZADD classement 210 "Chloé"
(integer) 1
127.0.0.1:6379> ZREVRANGE classement 0 -1 WITHSCORES
1) "Bob"
```

```
2) "320"
3) "Chloé"
4) "210"
5) "Alice"
6) "150"
```

ZREVRANGE ... WITHSCORES affiche le classement du plus grand score au plus petit.

Faire progresser un score :

```
127.0.0.1:6379> ZINCRBY classement 100 "Alice"
"250"
127.0.0.1:6379> ZREVRANGE classement 0 -1 WITHSCORES
1) "Bob"
2) "320"
3) "Alice"
4) "250"
5) "Chloé"
6) "210"
```

Alice est repassée devant Chloé automatiquement, sans avoir eu à retrier quoi que ce soit soi-même.

Cas d'usage concret : classement de joueurs, articles les plus populaires.

Étape 9 — Explorer les clés existantes

Voir toutes les clés créées pendant ce TP :

```
127.0.0.1:6379> KEYS *
1) "visites"
2) "utilisateur:1"
3) "tags"
4) "classement"
5) "taches"
6) "prenom"
7) "ville"
8) "panier_articles"
9) "produit:1"
```

KEYS * est pratique pour apprendre, mais à éviter en production sur une grosse base : la commande SCAN (non traitée dans ce TP débutant) fait la même chose sans bloquer le serveur.
Connaître le type d'une clé :

```
127.0.0.1:6379> TYPE classement
zset
127.0.0.1:6379> TYPE taches
list
```

Étape 10 — Tout nettoyer

```
127.0.0.1:6379> FLUSHALL
OK
127.0.0.1:6379> KEYS *
(empty array)
```

FLUSHALL supprime toutes les clés de la base — pratique pour repartir de zéro, à ne jamais utiliser par erreur en production.

Quitter le client :

```
127.0.0.1:6379> exit
```

Arrêter le conteneur Redis (optionnel) :

```
docker stop mon-redis
```

Récapitulatif des commandes vues

Commande	Rôle
`SET` / `GET` / `DEL`	Écrire, lire, supprimer une valeur simple
`EXISTS`	Vérifier qu'une clé existe
`INCR` / `INCRBY`	Incrémenter un compteur
`EX` (avec SET) / `TTL`	Donner une durée de vie à une clé
`HSET` / `HGET` / `HGETALL` / `HINCRBY`	Manipuler un objet à plusieurs champs (Hash)
`RPUSH` / `LPOP` / `LRANGE`	Manipuler une liste ordonnée
`SADD` / `SMEMBERS` / `SISMEMBER`	Manipuler un ensemble sans doublons (Set)
`ZADD` / `ZINCRBY` / `ZREVRANGE`	Manipuler un classement (Sorted Set)
`KEYS` / `TYPE`	Explorer les clés existantes
`FLUSHALL`	Tout supprimer

Corrigé des exercices

Exercice 1 :

```
SET prenom "VotrePrenom"
SET ville "VotreVille"
```

Exercice 2 :

```
SET panier_articles 0
INCR panier_articles
INCR panier_articles
INCR panier_articles
GET panier_articles   # -> "3"
```

Exercice 3 :

```
HSET produit:1 nom "Clavier" prix 49.90 stock 10
HGETALL produit:1
HINCRBY produit:1 stock -1
HGET produit:1 stock   # -> "9"
```

Pour la suite

Une fois ces manipulations à l'aise, la suite logique est le module 4 du cours (types avancés : Streams, Bitmaps, HyperLogLog, Geospatial) puis le module 12 (cas d'utilisation détaillés : cache, sessions, files d'attente) pour voir comment ces commandes s'assemblent dans une vraie application