Part A

Introduction to Redis

The command SET to store the value "Arjun" at key "server":

SET server "Arjun”

Fetch the stored value using GET

GET server => "Arjun"

A command in order to test if a given key exists or not: EXISTS server => 1

A screenshot of a computer

Description automatically generated

Initialization of a variable

SET Arjun\_conn 10

INCR atomically increments a number stored at a given key INCR Arjun\_conn

A screenshot of a computer

Description automatically generated

DEL to delete a given key and associated value DEL Arjun\_conn

Increment the number contained inside a key by a specific amount: INCRBY Arjun\_conn 10

Decrement the value of the key DECR Arjun\_conn DECRBY Arjun\_conn 10



Redis can be told that a key should only exist for a certain length of time. This is accomplished with the EXPIRE and TTL commands

SET resource:lock "Arjun"

EXPIRE resource:lock 120

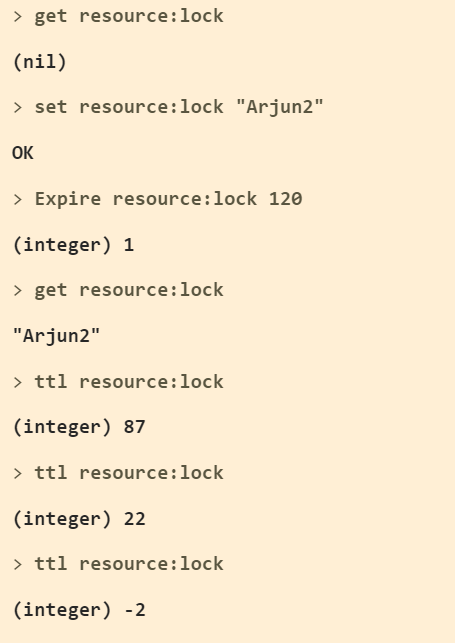
This causes the key resource:lock to be deleted in 120 seconds. You can test how long a key will exist with the TTL command. It returns the number of seconds until it will be deleted.

TTL resource:lock

The -2 for the TTL of the key means that the key does not exist (anymore).

A -1 for the TTL of the key means that it will never expire.

Note that if you SET a key, its TTL will be reset.



The SET command is actually able to accept further arguments in order to directly set a time to live (TTL) to a key, so you can alter the value of a key and set its TTL at the same time in a single atomic operation:

SET resource:lock "Arjun3" EX 5

TTL resource:lock => 5

It is also possible to cancel the time to live of a key removing the expiration and making the key permanent again.

SET resource:lock "Arjun3" EX 5

PERSIST resource:lock

TTL resource:lock => -1

Redis also supports several more complex data structures. The first one we'll look at is a list. A list is a series of ordered values. Some of the essential commands for interacting with lists are RPUSH, LPUSH, LLEN, LRANGE, LPOP, and RPOP. You can immediately begin working with a key as a list, as long as it doesn't already exist as a different type.

This concept is generally true for every Redis data structure: you don't create a key first and add things to it later, but you can just directly use the command to add new elements. As a side effect, the key will be created if it does not exist. Similarly, keys that will result empty after executing some command will automatically be removed from the key space.

RPUSH puts the new element at the end of the list.

RPUSH name "Arjun"

RPUSH name "Mannem"

LPUSH puts the new element at the start of the list.

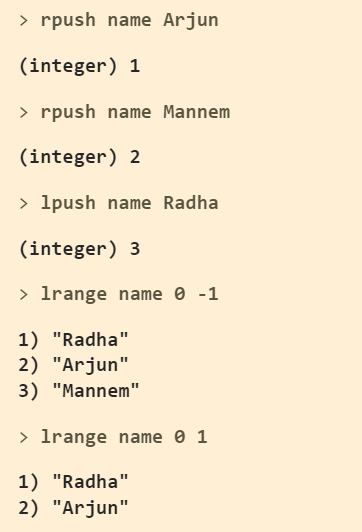
LPUSH name "Radha"

LRANGE gives a subset of the list. It takes the index of the first element you want to retrieve as its first parameter and the index of the last element you want to retrieve as its second parameter. A value of -1 for the second parameter means to retrieve elements until the end of the list, -2 means to include up to the penultimate, and so forth.

LRANGE name 0 -1

LRANGE name 0 1

LRANGE name 1 2



LPOP removes the first element from the list and returns it.

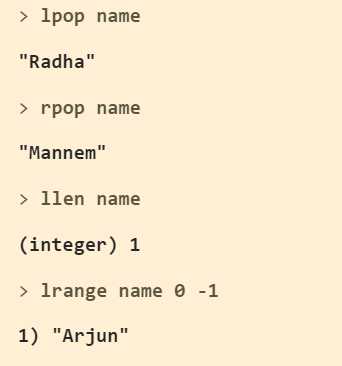
LPOP name

RPOP removes the last element from the list and returns it.

RPOP name

Note that the list now only has one element: LLEN name

LRANGE name 0 -1

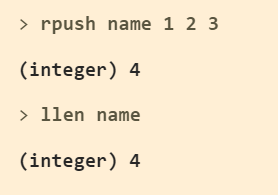


Tip: RPUSH and LPUSH return the total length of the list after the operation.

RPUSH name 1 2 3

You can also use LLEN to obtain the current length of the list.

LLEN name



The next data structure that we'll look at is a set. A set is similar to a list, except it does not have a specific order and each element may only appear once. Both the data structures are very useful because while in a list is fast to access the elements near the top or the bottom, and the order of the elements is preserved, in a set is very fast to test for membership, that is, to immediately know if a given element was added or not. Moreover, in a set, a given element can exist only in a single copy.

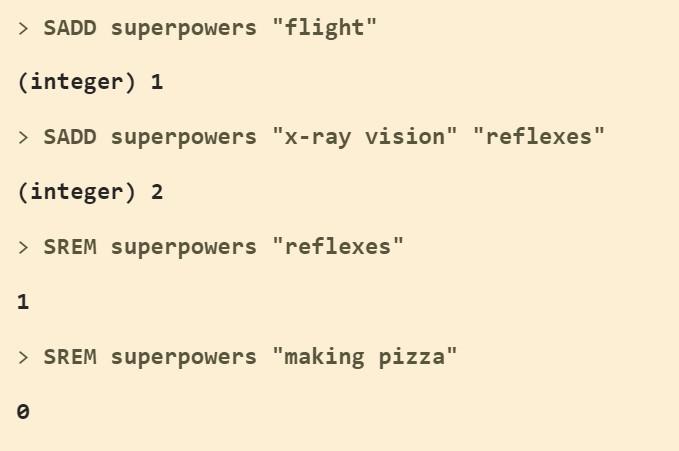
Some of the important commands in working with sets are SADD, SREM, SISMEMBER, SMEMBERS, and SUNION.

SADD adds the given member to the set, again this command is also variadic.

SADD superpowers "flight"

SADD superpowers "x-ray vision" "reflexes"

SREM removes the given member from the set, returning 1 or 0 to signal if the member was actually there or not.

SREM superpowers "reflexes" => 1 SREM superpowers "making pizza" => 0

SISMEMBER tests if the given value is in the set. It returns 1 if the value is there and 0 if it is not.

SISMEMBER superpowers "flight"

SISMEMBER superpowers "reflexes"

SMEMBERS returns a list of all the members of this set.

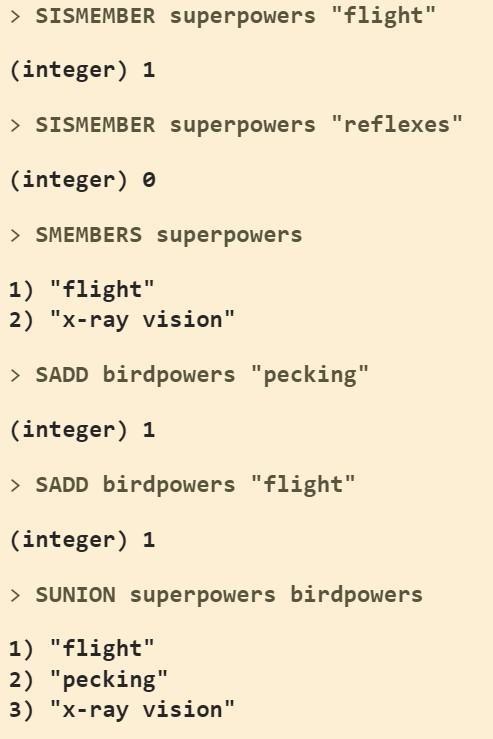
SMEMBERS superpowers

SUNION combines two or more sets and returns the list of all elements.

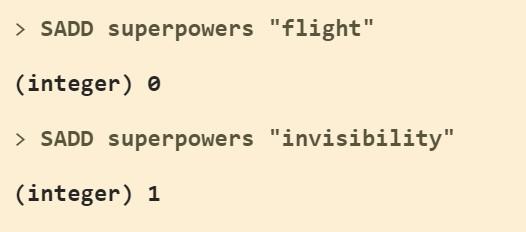
SADD birdpowers "pecking"

SADD birdpowers "flight"

SUNION superpowers birdpowers



The return value of SADD is as important as the one of SREM. If the element we try to add is already inside, then 0 is returned, otherwise, SADD returns 1:

SADD superpowers "flight" 

SADD superpowers "invisibility"

Sets also have a command very similar to LPOP and RPOP in order to extract elements from the set and return them to the client in a single operation. However, since sets are not ordered data structures the returned (and removed) elements are totally casual in this case.

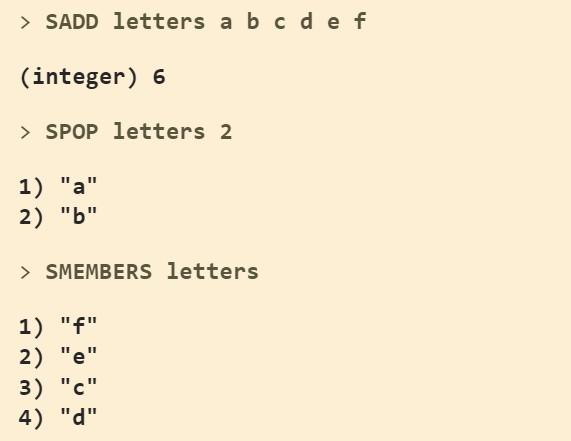
SADD letters a b c d e f

SPOP letters 2

The argument of SPOP after the name of the key, is the number of elements we want it to return, and remove from the set.

Now the set will have just the remaining elements: SMEMBERS letters

There is also a command to return random elements without removing such elements from the set, it is called SRANDMEMBER. You can try it yourself, the arguments are similar to SPOP, but if you specify a negative count instead of a positive one, it may also return repeating elements.



Sets are a very handy data type, but as they are unsorted they don't work well for a number of problems. This is why Redis 1.2 introduced Sorted Sets.

A sorted set is similar to a regular set, but now each value has an associated score. This score is used to sort the elements in the set.

ZADD hackers 1940 "Alan Kay"

ZADD hackers 1906 "Grace Hopper"

ZADD hackers 1953 "Richard Stallman"

ZADD hackers 1965 "Yukihiro Matsumoto"

ZADD hackers 1916 "Claude Shannon"

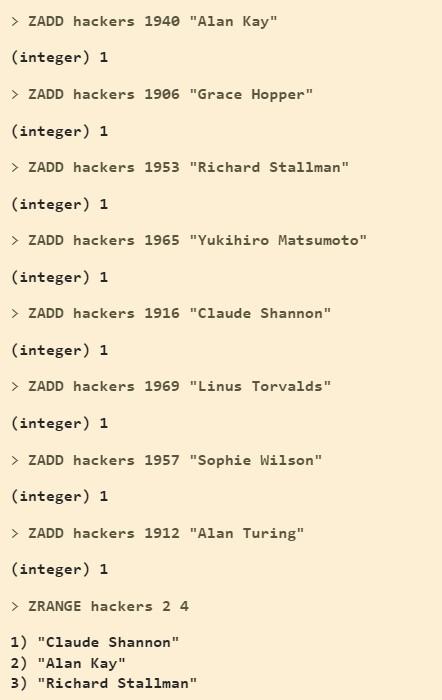
ZADD hackers 1969 "Linus Torvalds"

ZADD hackers 1957 "Sophie Wilson"

ZADD hackers 1912 "Alan Turing"

In these examples, the scores are years of birth and the values are the names of famous hackers.

ZRANGE hackers 2 4



Simple strings, sets, and sorted sets already get a lot done but there is one more data type Redis can handle: Hashes.

Hashes are maps between string fields and string values, so they are the perfect data type to represent objects (eg: A User with a number of fields like name, surname, age, and so forth):

HSET user:1000 name "Arjun Mannem"

HSET user:1000 email "[Arjun.Mannem@example.com](mailto:praharsha.bejju@example.com)" HSET user:1000 password "d0ntel"

To get back the saved data use HGETALL: HGETALL user:1000

You can also set multiple fields at once:

HMSET user:1001 name "Arjun Mannem" password "neVer" email "[ArjunM@example.com](mailto:bejjup@example.com)"

If you only need a single field value that is possible as well: HGET user:1001 name

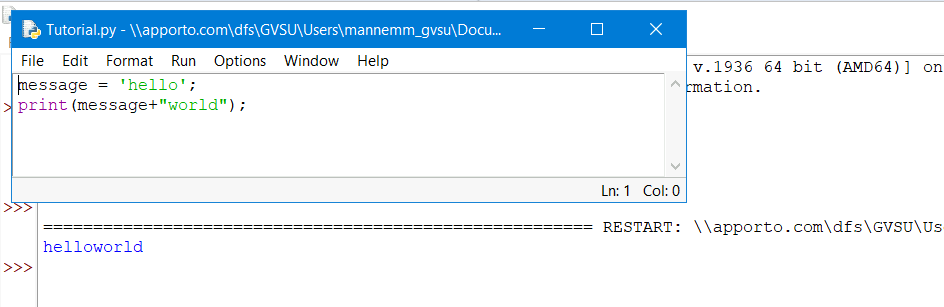
Numerical values in hash fields are handled exactly the same as in simple strings and there are operations to increment this value in an atomic way.

HSET user:1000 visits 10 HINCRBY user:1000 visits 1 HINCRBY user:1000 visits 10 HDEL user:1000 visits HINCRBY user:1000 visits 1

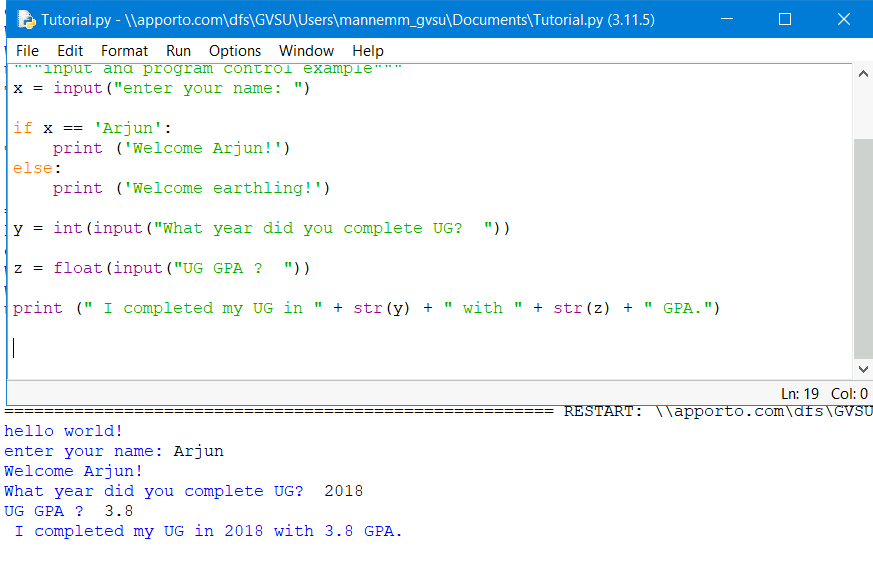


**Part B**

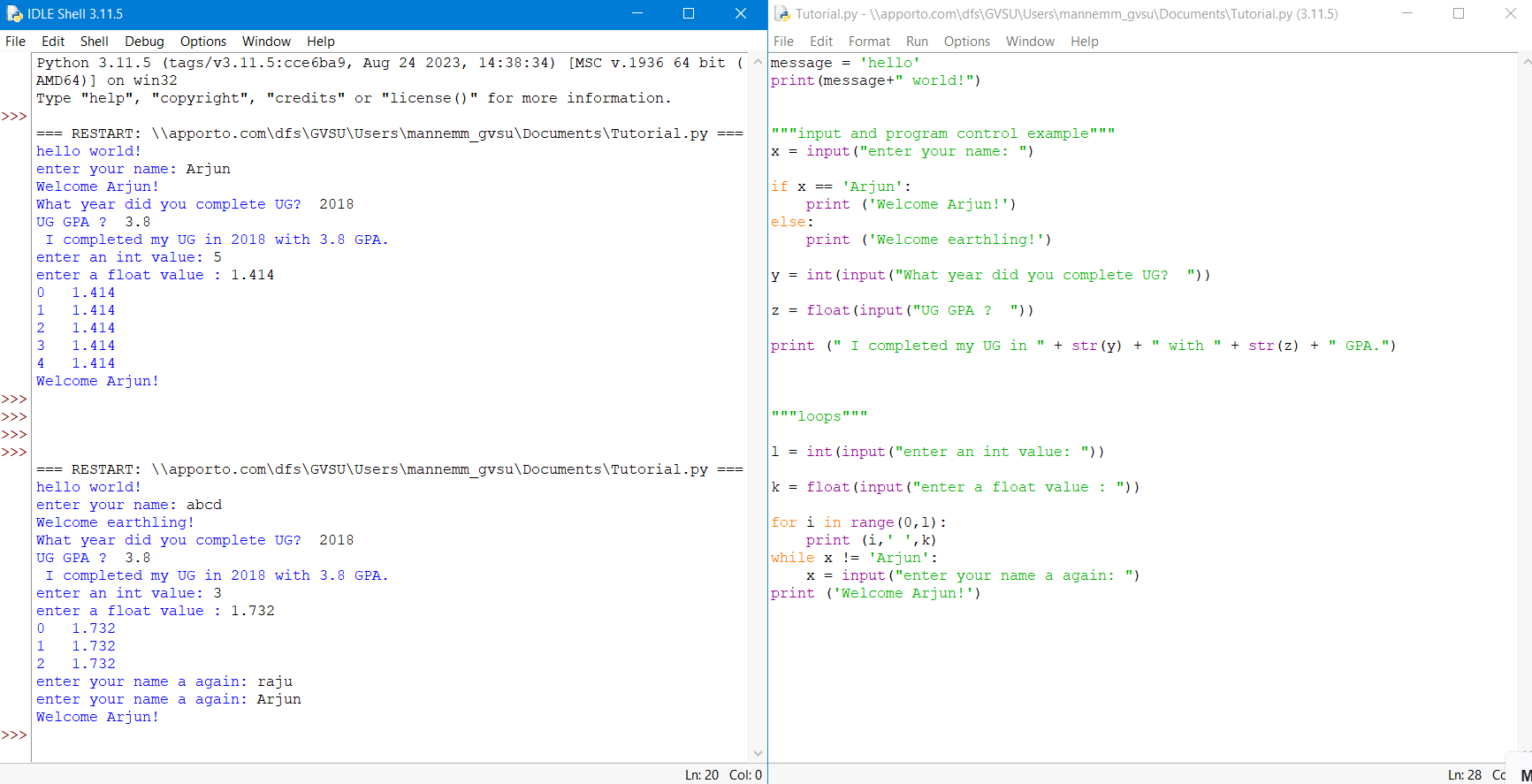
**Install and Develop Simple Programs in Python Step #B1: Run a Python program**

****

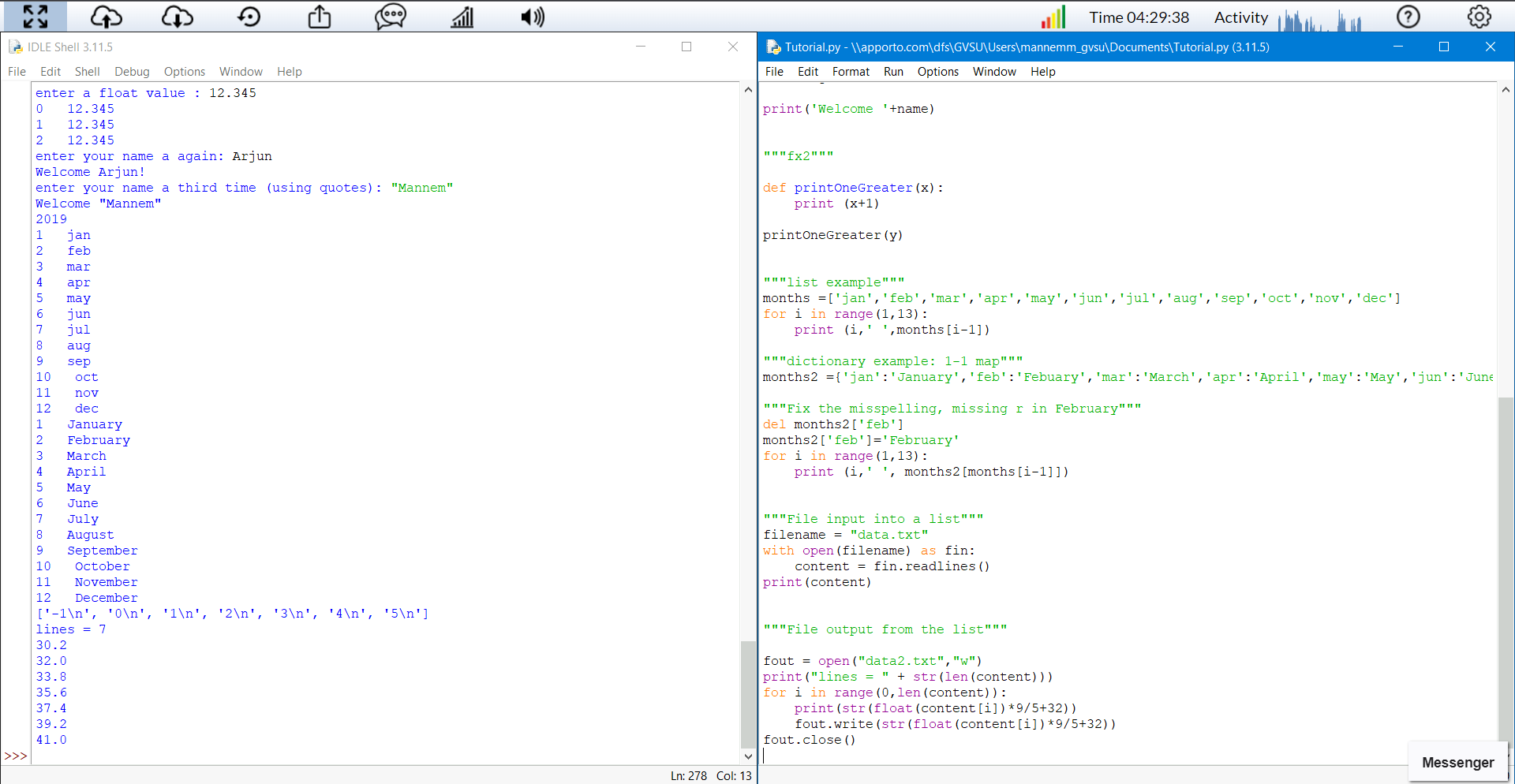
**IF…ELSE….**

****

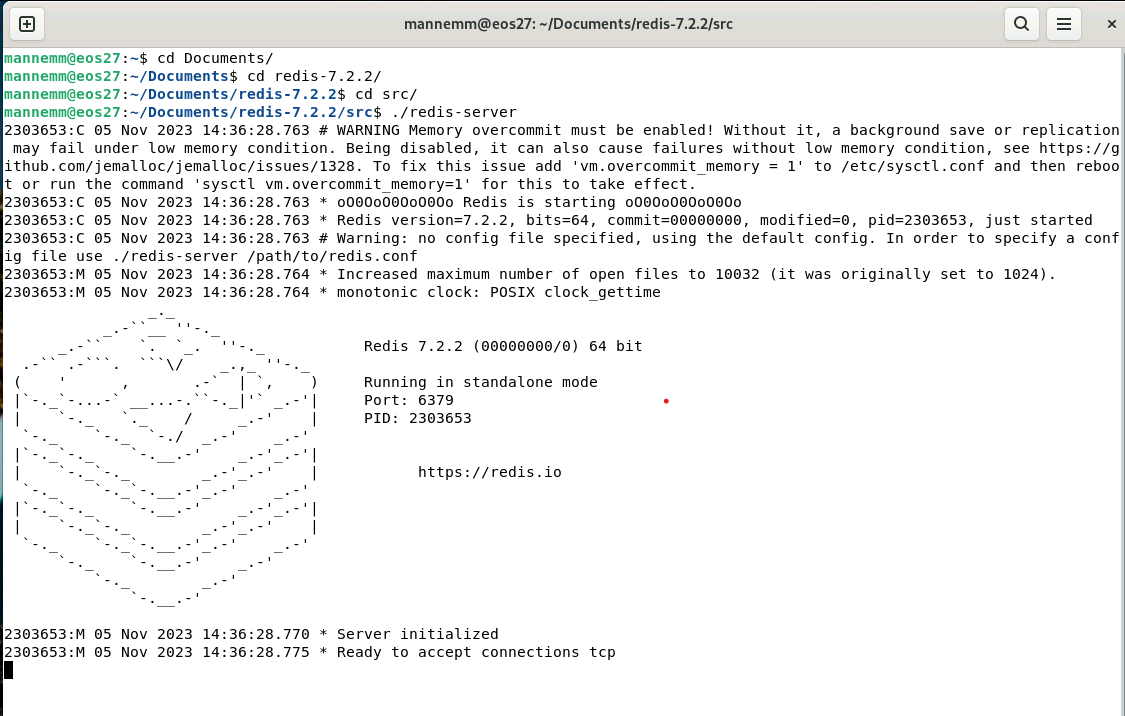
**LOOPS…**

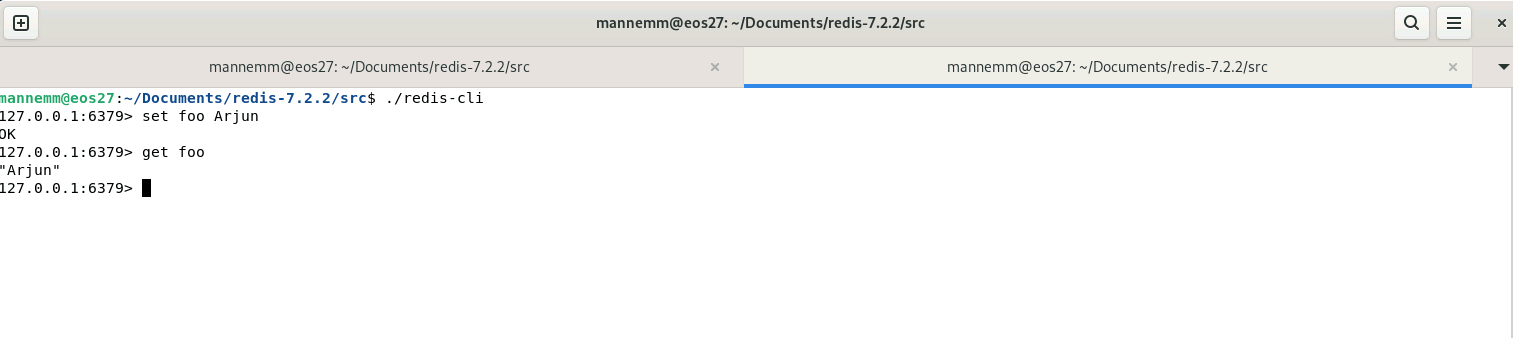
****

**Definition… Lists… Dictionary… Files…**

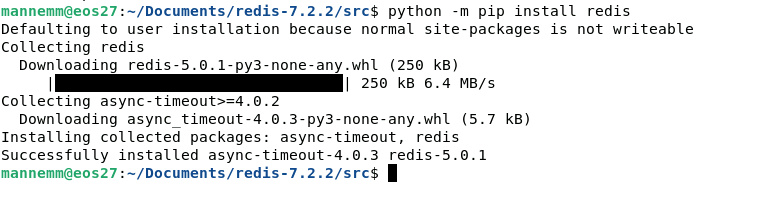
****

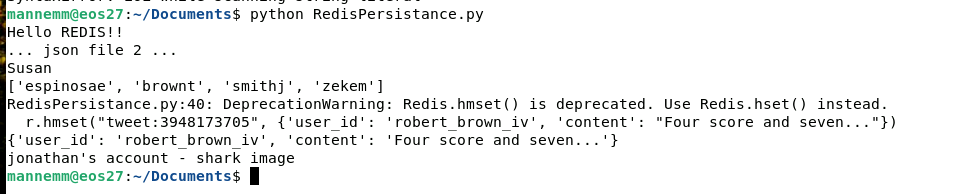
**Part C**

****

****

Redis installation and Redis-Python script

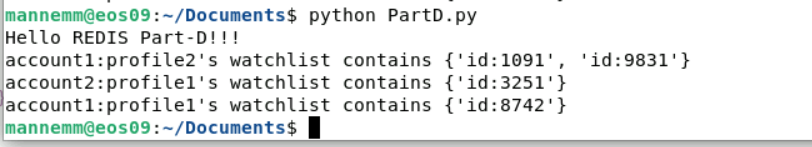




**Part D**

**Attached is the Python file name “PartD.py” along with this file**

**Output:**

****