Topic: Key-Value Stores + Python Scripting

Description

Key-value databases allow users to store arbitrary data as a 'value' that is identified by a given unique 'key.' Recall our discussion on just how many unique keys a 4GB keyspace can map to! A key can only map or relate to one given 'value'. Note: a set of related data could be co-located and stored within one array 'value.' Binary objects (e.g., images, videos, objects, etc.) can be used. Unlike document databases, there is no search based on the content of the value. Key-value stores do not take advantage of the powerful retrieval methods built in to the document databases (e.g., go find all of the books < $5). Instead, you might utilize a key-value store when the key is known and will always be used to locate/retrieve a document. It is widely used for very efficient retrieval of large collections (e.g., Tumblr, Uber, StackOverflow, Hulu, Twitter, Pintrest, Flickr, Craigslist, Alibaba, …). With a twitter_ID, Redis can quickly give back the user's account. With a tweet_id, Redis can quickly locate the tweet and its content. The basic operations are to SET a new key/value pair – or to update it with a new value, GET a value back for a given key, or delete the key/value pair. The basic concept of a hashmap data structure provides the theoretical underpinning of key-value stores.

Similar to most NoSQL database systems, key-value databases are used when consistency is not important as availability and constant updates. Advantages of key-value databases are that the data model is simple, values are easy to store, applications are easy to write, and the documents are independent units. The last point is important to having the database replicate and shard the data.

Redis learning objectives

- become familiar with key-value databases
- experience manipulating data from a web-based database interface
- understand how/when key-value stores should be utilized for certain types of problems
- connect to a database from a program
- perform data persistence and CRUD operations from a program

Python learning objectives

- Become familiar with the basics of Python libraries and programming
- Concepts: variables, program control (loops, conditional logic), lists/arrays, dictionaries (keyvalue), file IO, etc.
- Parse and ingest an input file
- Produce an output file
- Connect with a database for dynamic operations

Components

For this assignment, you will begin by using a web-interface provided by Redis (i.e., no installation on your computer is required for the introductory tutorial in Part A). This will expose you to basic data storage and retrieval operations through a web-based interface. In the subsequent sections, we will use the EOS lab practice your new skills by using Python (Part B) and Redis (Part C).

Part A: ***Introduction to Redis tutorial - 25%***

Resources

Interface: http://try.redis.io
Documentation: https://redis.io/documentation
Full command reference: https://redis.io/commands

For Windows, Linux or Mac, go to http://try.redis.io and follow the instructions to complete the tutorial. Instead of simply clicking your way through the code (using the hyperlinks). Type in every command into a text file/editor of your choice and then paste it into the command line. I would recommend using Notepad++ (Windows) or Sublime Text (Mac). The point of the tutorial is for you to actually learn the purpose/utility of each of the commands and the syntax. Typing them yourself will reinforce your learning of the commands. The script will be turned in as part of the assignment, and also serves as a 'cheat-sheet' you can refer to later.

Deliverable: Save and turn in all of the commands from the default tutorial.

Instructor's Note: we moved quickly through a demonstration of the basic CRUD syntax for the primary data structures in REDIS in class. This tutorial will take you through all of that at your own pace so please take the time to play around and TYPE IN THE SYNTAX yourself (no copy and paste) to get comfortable with it. Review the posted whiteboard notes if you need the reminder.

Part B: ***Install and Develop Simple Programs in Python – 25%***

Python is a commonly used programming language used for scripting, analysis, etc. Warning: There are major differences in Python version 2.7 versus 3. The newer versions are not backwards compatible with numerous older Python 2 libraries.

Resources

https://www.python.org/about/gettingstarted
https://docs.python.org/3.10/tutorial

Setup – DON'T INSTALL SOFTWARE ON YOUR LAPTOP, JUST USE THE EOS LAB
Python is already installed on EOS for us to use – we can skip the installation and Python setup!

For your **future reference**, you could install and use a Python IDE/Interpreter on your own laptop. Install the newest Python version, if your computer doesn't already have a working version installed.

https://wiki.python.org/moin/BeginnersGuide/Download
https://docs.python.org/3/using/index.html

Also consider installing Python conda/miniconda/anaconda or use Juypter Notebooks.

https://docs.conda.io/projects/conda/en/latest/user-guide/install
https://jupyter.org

Step #B1: ***Run a Python program***

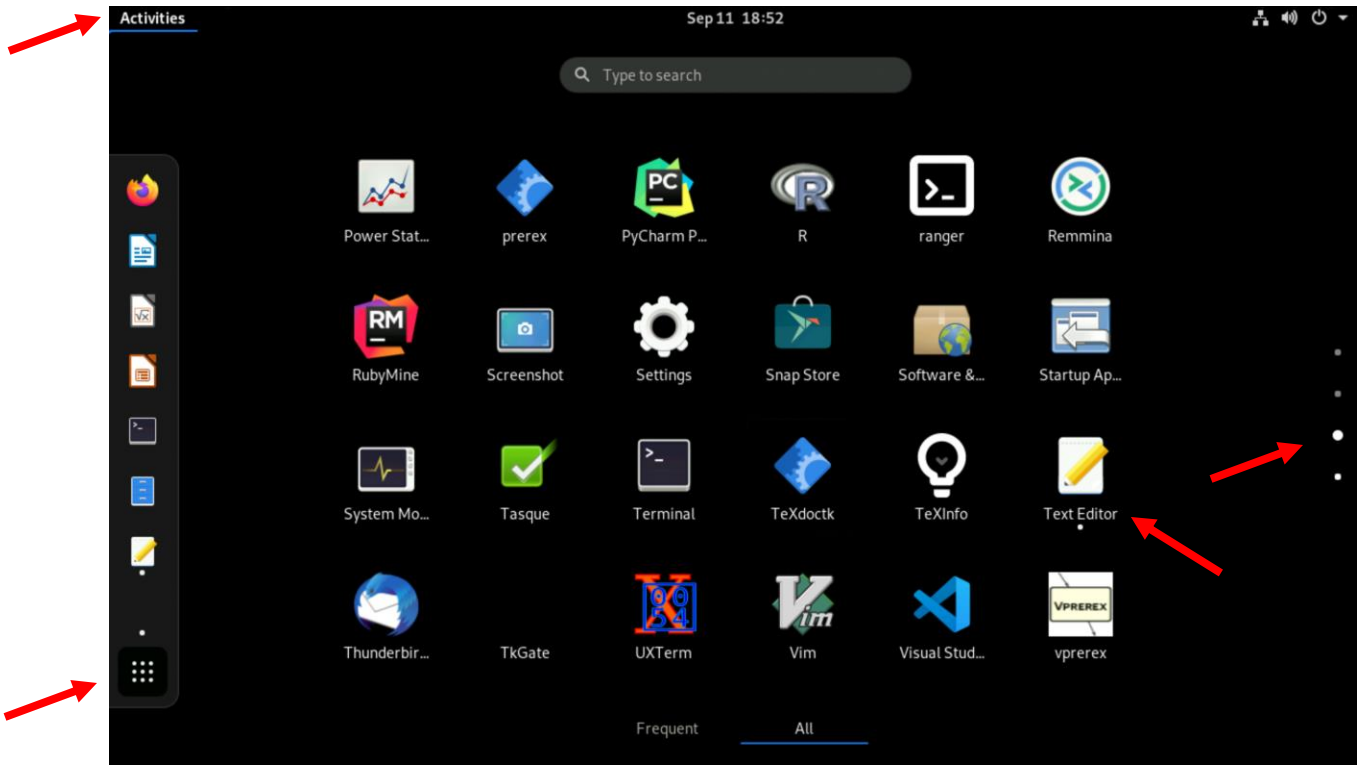
The easy method to login to the EOS lab:

> Go to https://cislab.hpc.gvsu.edu and login using your regular GVSU username/password. Select EOS RDP to login to an arbitrary machine – providing your username/password a second time.
>
> You can now use the Linux GUI or command line to complete the lab. You will need to use the **Terminal** and **TextEditor** for this lab.

More difficult method to login to EOS:

> From off campus, you will need to install GVSU's VPN called Pulse Secure, and potentially DUO for two-factor authentication.
>
> On a PC, install **putty** (for ssh connections) and **WinSCP** program (for sFTP file transfers). On a Mac, Terminal is already available (for ssh) but you can install a tool such as **Filezilla** (for visual sFTP file transfers). The following guide has further advice: https://cis.gvsu.edu/~meos/remoteaccess/windows/file-transfer.html#graphical

After you login to EOS RDP, click on Activities/ShowApplications/TextEditor to open the text file editor.



Create a new text file, `tutorial.py`, using the text editor app. (Image next page)

Try storing a string and printing it out. Add these lines to your empty text file. Please do not copy and paste these commands from this document. Type in the code so you actually have to think about what the code looks like and is doing.

```
message = "hello";
print(message+" world");
```

From the desktop, open Activities/Terminal and navigate to where your textfile was saved (use ls and cd). Check which Python version you are using. Then try running it from a command prompt by using the Python 3 binary/executable already installed.

```
$ which python
$ python tutorial.py
hello world
```

Try creating a new variable and printing it out. Notice the lack of overhead and minimal code within Python scripts. Try getting input from the user by adding the following lines to your file. Note: comments can be left using # for single lines and either ''' or """ for multi-line comments. Modify your textfile, and run your program twice to test the if-else statement.

Where you see "Jonathan" or some other names in the following text, replace some of them with your own name so I can see that you did the work.

```
"""input and program control example"""
x = input("enter your name: ")
if x == 'jonathan':
       print ('Welcome jonathan!')
else:
       print ('Welcome earthling!')
y = int(input("enter an int: "))
z = float(input("enter a float: "))
```

Step #B2: Try using a for and while loop with your existing input. Add these lines to your program and then test it.

```
"""loops"""
for i in range(0,y):
       print (i,' ',z)
while x != 'jonathan':
       x = input("enter your name a second time: ")
print ('Welcome jonathan!')
```

Step #B3: The def command is used to define a new function/method. The function can receive multiple input parameters, print content, store variables, and return a result. Add these lines to your program and then test it.

```
"""functions"""
def getName():
	myname = input("enter your name a third time (using quotes): ")
	return myname
name = getName()
print('Welcome '+name)

"""fx2"""
def printOneGreater(x):
	print (x+1)
printOneGreater(y)
```

Step #B4: Lists are commonly used to manipulate content in Python. They function similar to arrays in other programming languages you may have used in the past. Note the use of square brackets to store items in a given order based on position in the list. Add these lines to your program and then test it.

```
"""list example"""
months =
['jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec']
for i in range(1,13):
	print (i,' ',months[i-1])
```

Step #B5: Dictionaries are used to store a one-to-one mapping (key,value) between two content types. Note the use of curly braces for dictionaries. Entries can be dynamically added and deleted in the map. Add these lines to your program and then test it.

```
"""dictionary example: 1-1 map"""
months2 =
{'jan':'January','feb':'Febuary','mar':'March','apr':'April','may':'May','jun':
'June',
'jul':'July','aug':'August','sep':'September','oct':'October','nov':'November',
'dec':'
December'}

"""Fix the misspelling, missing r in February"""
del months2['feb']
months2['feb']='February'
for i in range(1,13):
	print (i,' ', months2[months[i-1]])
```

Create a new dataset, data.txt, and add the following values to it. Do not have an empty line at the end (out of range errors will occur)!

```
-1
0
1
2
3
4
5
```

Step #B6: As with other languages, there are multiple ways to read in content from files. Here, a variable (fin) is used to read in all of the lines as strings at once (readlines), storing them into a list (content). Other approaches can be used for single strings or a given number of characters. More commonly, a line (containing multiple values, spaces, commas, etc.) might be read in and then split into multiple tokens (individual values). Note, data.txt must be in the same directory/folder as the python

script for it to be 'found'. Add these lines to your program and then test it.

```
"""File input into a list"""
filename = "data.txt"
with open(filename) as fin:
       content = fin.readlines()
print(content)
```

Step #B7: Similarly, data can be written to output files. Here, the fout variable is used to write/overwrite a file. The len function returns the number of items (length) of the list. A string or number can be printed using the write command. Add these lines, and then test it. Note the new output file with line numbers.

```
#File output from the list
fout = open("data2.txt","w")
print("lines = " + str(len(content)))
for i in range(0,len(content)):
       print(str(float(content[i])*9/5+32))
       fout.write(str(float(content[i])*9/5+32) + "\n")
fout.close()
```

Future Python topics/libraries to explore: math, scipy, numpy, matplotlib, array, pandas, …

Deliverable: Save and turn in your tutorial.py file as well as the results from running the program on the command line (with any user input you wish). DON'T FORGET – your name should appear in some the code above.

Part C: ***Connecting to Redis within Python – 25%***

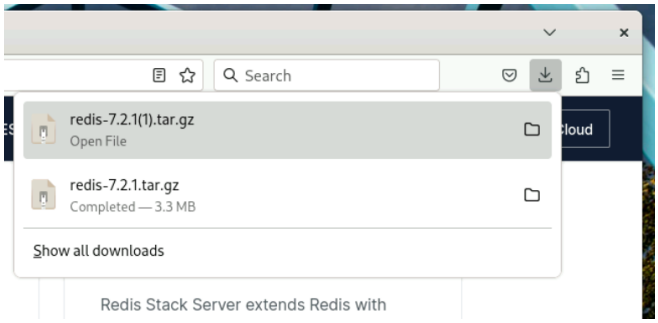
Step C.1) Visit https://redis.io or https://redis.io/topics/quickstart and view the resource. Download/install Redis onto EOS using the command *wget* (see below). You will have to uncompress (*tar*) the downloaded tar file, use the command line to navigate (*cd*) to this directory (`redis-stable`), type *make* to compile the source code into an executable software app, and *make test* to verify it installed correctly.

```
$ wget http://download.redis.io/redis-stable.tar.gz
```

INSTRUCTOR'S NOTE:

When I attempted to run the wget command above, it failed with a "forbidden" error code. Since I do not wish to try to debug that issue, I took the alternative route of downloading the file using the web browser on the EOS machine. The easy solution here is to open a browser and go to https://redis.io/download/ and click on the Download 7.2.1 link.

You will then have to find the file you downloaded by clicking on the Download Status icon neasr the top right of your browser (Firefox example screenshot below)

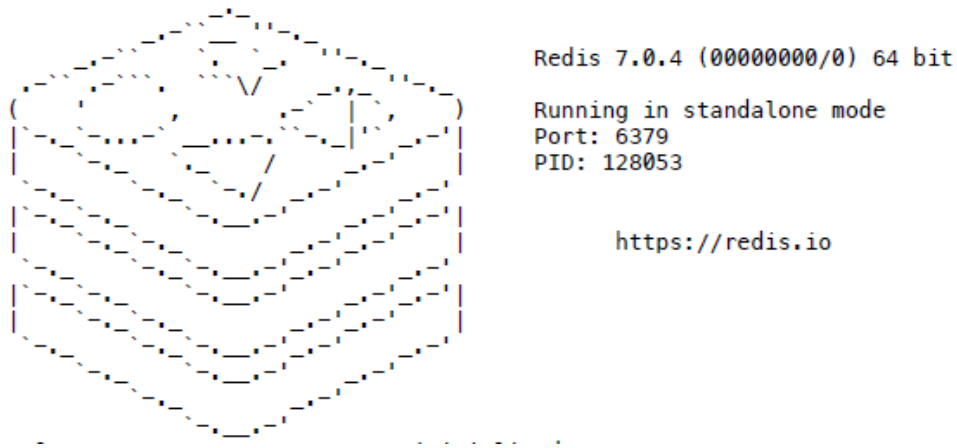


Then click on the folder icon to the right of the filename, this will open up the folder. From there you can drag and drop the file into your Document directory (or a directory of your choice) to continue with the commands in the command shell… You will have to cd into that directory and then you will have to uncompress (*tar*) the downloaded tar file, use the command line to navigate (*cd*) to this directory (`redis-stable`), type *make* to compile the source code into an executable software app, and *make test* to verify it installed correctly.

```
$ tar xvzf redis-stable.tar.gz
$ cd redis-stable
$ make
...
$ make test
...
\o/ All tests passed without errors!
Cleanup: may take some time... OK
$
```

To start your Redis data structure server,

```
$ cd src
$ ./redis-server
```




```
128053:M 21:45:53.561 # Server initialized
128053:M 21:45:53.561 # WARNING overcommit_memory is set to 0! Background save may fail
under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to
/etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for
this to take effect.
128053:M 21:45:53.562 * Ready to accept connections
```

Step C.2) Test your Redis server by opening a new command prompt with a new **Terminal** window , navigate to your redis src folder (if not there already), and use the redis command line client (redis-cli) to try a few commands.

```
$ cd redis-stable/src/
$ ./redis-cli
127.0.0.1:6379> set foo bar
OK
127.0.0.1:6379> get foo
"bar"
```

Step C.3)

In Redis, data is largely organized within key-value pairs, somewhat similar to a Python dictionary. A value could be a simple integer or a large, complex object with multiple sub-components. For this lab, we will develop a simple data model within Python, persist the data to Redis, and retrieve the stored values.

For Python to connect to Redis, a client interface is needed. Using the commands below, install the following library for your user on your EOS account (See: https://github.com/andymccurdy/redis-py). From a terminal window, use your python binary to install this interface. Note, although the package is called *redis* in the following command, what you actually are installing isn't actually Redis but a Python library to allow Python to connect to Redis. You may have to also install pip, which will be useful for installing python packages in the future. Some students do not have to.

127.0.0.1:6379> exit
$ python -m pip install redis --user //you may need to add —-user or –user to the end

Add the following python code to a new file (RedisPersistance.py), review each line and describe to yourself what the line is doing, execute it (python RedisPersistance.py), and save the results from the console. Hopefully, Python looks similar to languages you've used in the past, and these commands should look very familiar from Part A. Please replace Susan Smith's name in the text below with your first and last name. Be careful copying and pasting this code!! Especially you Mac users… Some of the characters and spaces might not paste properly for you and could cause problems debugging the code.

```
import redis

redis_host = "localhost"
redis_port = 6379
redis_password = ""

def hello_redis():
	try:
		#create the connection
		r = redis.StrictRedis(host=redis_host, port=redis_port,
		password=redis_password, db=0, decode_responses=True)

		#post a key-value message
		r.set("msg:hello", "Hello REDIS!!")

		#get the key-value message
		msg = r.get("msg:hello")
		print(msg)

		#simple session data
		r.set("session_id:1234", "... json file 1 ...")
		r.set("session_id:2345", "... json file 2 ...")
		r.set("session_id:3456", "... json file 3 ...")
		msg = r.get("session_id:2345")
		print(msg)

		#mset
		r.mset({"id": "G00000123", "fname": "Susan", "lname": "Smith", "gpa": "4.0"})
		msg = r.get("fname")
		print(msg)

		#lists
		r.lpush("active_users2", "smithj", "brownt", "espinosae")
			#r.lpush("active_users2", "smithj brownt espinosae")
		r.rpush("active_users2", "zekem")
		msg = r.lrange("active_users2", 0, -1)
		print(msg)
```

```
                #hmset
                r.hmset("tweet:3948173705", {'user_id': 'robert_brown_iv', 'content': "Four
                score and seven..."})
                msg = r.hgetall("tweet:3948173705")
                print(msg)

                #hset
                r.hset("user1234","profile1","jonathan's account")
                r.hset("user1234","avatar1","shark image")
                r.hset("user1234","profile2","wife's account")
                r.hset("user1234","avatar2","owl image")
                r.hset("user1234","profile3","kid's account")
                r.hset("user1234","avatar3","penguin image")
                msg = r.hget("user1234", "profile1")
                msg2 = r.hget("user1234", "avatar1")
                print(msg + " - " + msg2)

        except Exception as e:
                print(e)

if __name__ == '__main__':
        hello_redis()
```

Deliverable: Save and turn in the output from running this program via the command line. You do not need to turn in the code itself. Don't forget to put your name in the code as instructed above – and in other places of your choice.

Try to explain to yourself what the code is trying to do in each spot. Notice that each chuck is manipulating a different data structure in the data store.

Part D: *Connecting to Redis within Python – 25%*

Integrate what you have learned in Part's A-C to manage user watchlists for a video streaming website. The input data (CRUD.txt) contains a sequence of movie watchlist additions and deletions for several accounts. Each account may contain multiple sub-accounts (user profiles). ***Develop a python program to read CRUD.txt, interpret the commands in the file to update REDIS, and print the final ending watchlists for all sub-accounts.*** You may use any REDIS structures you deem appropriate. You must read from the CRUD.txt file and print the final results. Assume your professor might test and grade your code with a similar CRUD.txt file that simply contains a different set of add and delete operations.

HINT#1: you might edit the Part C code inside the try statement to read in the CRUD.txt file, loop through each line, split each three-column line into three tokens, and execute an appropriate add or remove command. The first column of the file designates an account ID, the second column specifies a data operation (add or remove), and the third column contains a video ID.

HINT#2: simply use the **accountX:profileY** token as your key to one of the REDIS structures.

HINT#3: as you loop through input, keep track of every **username** you encounter in a "users" structure. This will be useful to query everyone's final watchlists.

HINT#4: if you choose a **set** per user sub-profile, then try SADD, SISMEMBER, SREM, and SMEMBERS commands.

HINT#5: refer to https://www.w3schools.com/python/default.asp and https://realpython.com/python-redis if needed.

$ **python partD.py**
account1:profile2's watchlist contains {'id:1091', 'id:9831'}
account1:profile1's watchlist contains {'id:8742'}
account2:profile1's watchlist contains {'id:3251'}

$ **control + C**
…
Redis is now ready to exit, bye bye...

Deliverable: Save and turn in a copy of your output from running the python script that was provided. Turn in your Part A (redis tutorial commands and output), B (Python file and output), C (output from running the instructor's program), and D (your program and output) deliverables to Blackboard (i.e., output from each part).

That's all folks! Done for the day!

Future Work)

Future work: You might review redis-py documentation and tutorials for further exploration of the interface (e.g., https://realpython.com/python-redis, https://www.agiliq.com/blog/2015/03/getting-startedwith-redis-py , etc.). Also, try and see if you can use the command line client and a program to both access the same data. What happens to your data when your server is shut down? Shut it down, spin it up again, and query the contents.