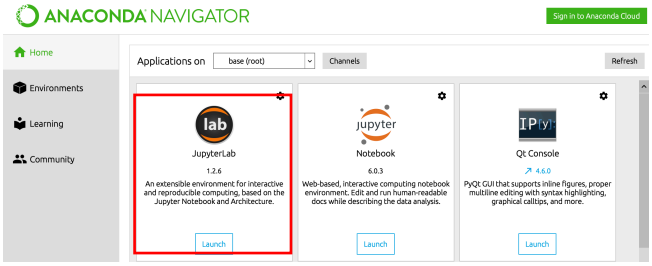
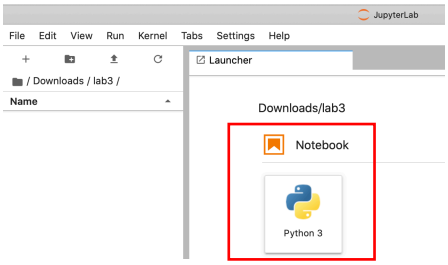
# BU.330.740 Large Scale Computing with Hadoop

## Lab 3. A simple MapReduce: Frequent Singleton Itemset

Learning Goal: write your own MapReduce using frequent singleton itemset problem, and then deploy on AWS Hadoop cluster
Required Skills: understand parallelization framework

1. Download transaction.dat.zip from Blackboard, unzip it and store it in your local folder.
2. Open Anaconda, and we will use **Jupyter Lab** this time. It will launch in a webpage. You can navigate to your desired working directory in the navigation panel on the left side. Then choose **Notebook->Python 3**



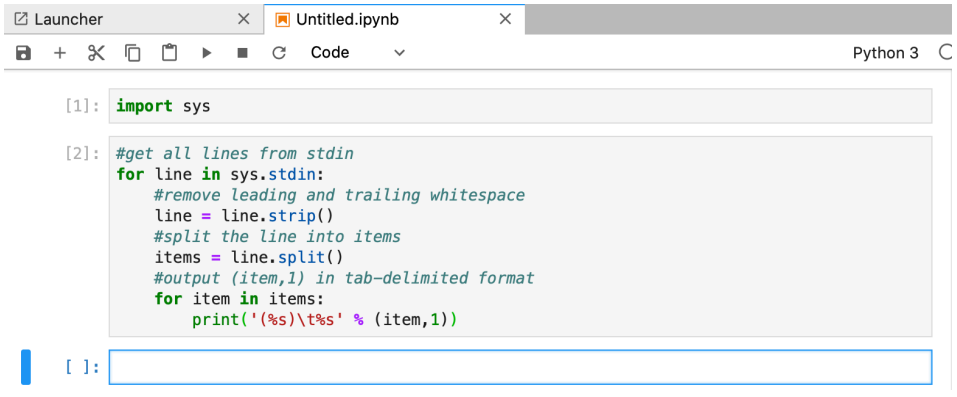

3. First import Python module *system* so that we can use methods in this module

```
import sys
```

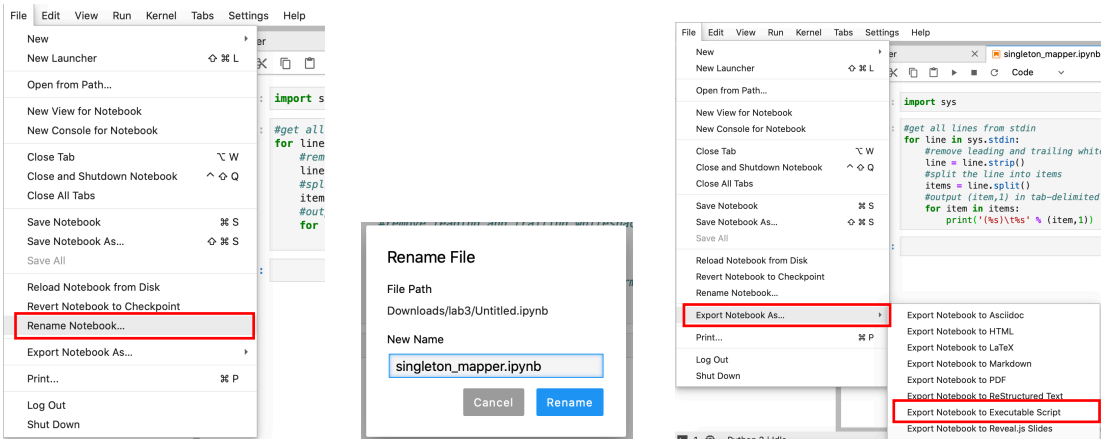
4. Get all lines from *sys.stdin*, remove any leading and trailing whitespace, and then split the line into items. Then output each item with count 1 in tab-delimited format

```
for line in sys.stdin:
    line = line.strip()
    items = line.split()
    for item in items:
        print('(%s)\t%s' % (item,1))
```

Please use # to add comments to any parts as needed, shown in the following screenshot. You can also use **Run** button to check any syntax errors.




5. From the left panel, choose **File->Rename Notebook…**, then change the name to singleton_mapper.ipynb. Also from **File** panel, choose **Export Notebook As…->Export Notebook to Executable Script**. This will download singleton_mapper.py to your Downloads folder.




6. Make another new Python 3 Notebook for reducer. First import *system* module, initialize the itemCount dictionary, and set support to 2.

```
import sys
itemCount = {}
support = 2
```

Then similarly, get all lines from *sys.stdin*, and remove any leading and trailing whitespace. Since we are parsing key, value from mapper output in the tab-delimited format, we split the line into item and count using *tab* as the separator. After split, we convert count from string to integer. If the current item is already in itemCount dictionary, then add the count into dictionary entry; otherwise, add an entry in dictionary with key as current item, and value as count. Again, comment your code as needed. I've included my comments in the following lines.

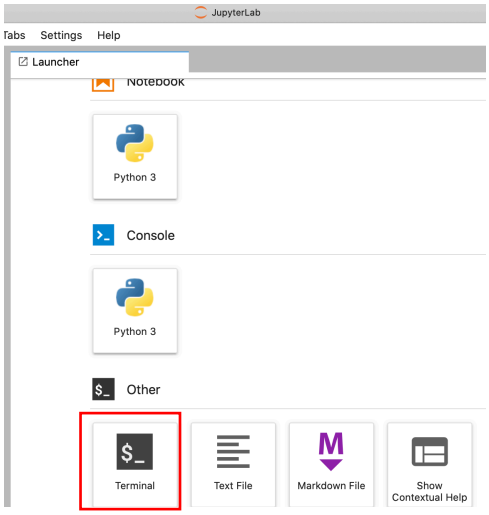
```
for line in sys.stdin:
    line = line.strip()
    #parse the key,value from mapper
    item, count = line.split('\t',1)
    #convert count from string to int
    try:
        count = int(count)
    except ValueError:
        #if count is not a number, ignore this line
        continue
    #if item already in the dictionary, add the count
    try:
        itemCount[item] = itemCount[item]+count
    #otherwise, create a new key,value in dictionary
    except:
        itemCount[item] = count
```

Finally, output those item with count larger than or equal to support.

```
#output the desired items
for item in itemCount.keys():
    if itemCount[item] >= support:
        print('%s\t%s' % (item, itemCount[item]))
```

7. Rename this into singleton_reducer.ipynb, and then export as .py file. Thus, you obtain singleton_reducer.py file.

8. Windows users please read Supplementary Document for this step!!
   Before we deploy the code on AWS, we can test them locally. In Jupyter Lab, choose **Other->Terminal**.




   First we will make mapper and reducer executable. I assume your mapper and reducer files are under Downloads folder. In the command line, type in (if you save mapper and reducer in a different directory, then replace the red parts with the correct path):

```
chmod +x Downloads/singleton_mapper.py
```

   and also:

```
chmod +x Downloads/singleton_reducer.py
```

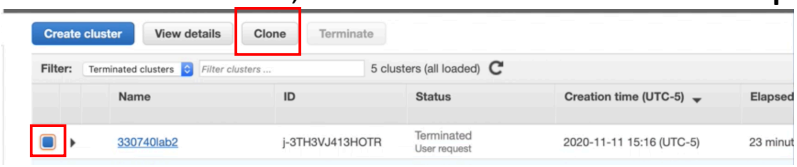
   then test mapper using command:

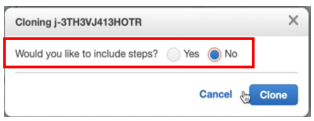
```
echo -e "0 1 2 3 4\n0 1" | Downloads/singleton_mapper.py
```

   if command succeeds, test both mapper and reducer using:

```
echo -e "0 1 2 3 4\n0 1" | Downloads/mapper.py | sort -k1,1 | Downloads/reducer.py
```

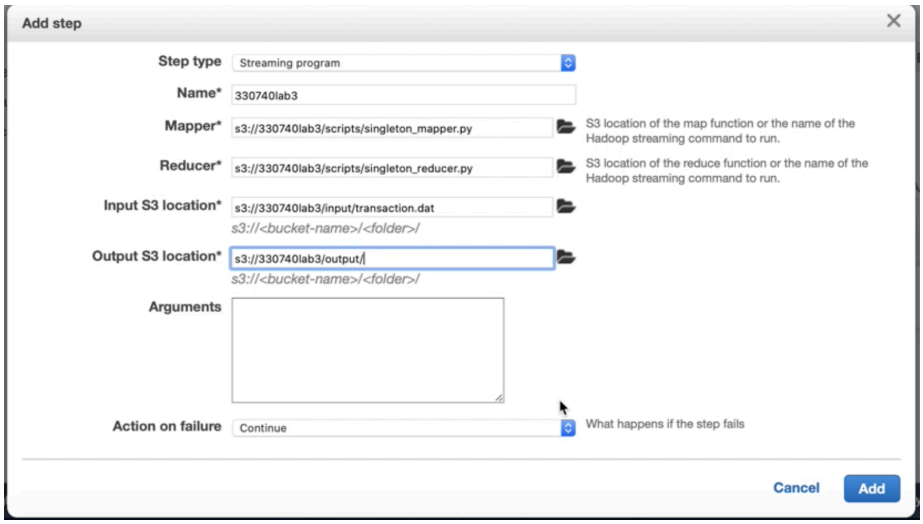
9. If the codes test successfully, we are now ready to deploy them on AWS. login into AWS Educate account, go to **AWS Management Console->EMR**, choose the cluster you set up in lab2 and then **Clone**, and choose **DO NOT include the steps**.




10. While waiting for the cluster to be provisioned. Go to **AWS Management Console->S3,** create a bucket for lab3. Create 2 folders in your bucket**,** 1 for input file and 1 for your python scripts.
    Upload singleton_mapper.py and singleton_reducer.py into python scripts folder; upload transaction.dat into input folder.
11. Wait till the cluster is ready, add a step of type **Streaming program**. Name your Streaming program; point Mapper to singleton_mapper.py and Reducer to singleton_reducer.py on your S3 instance; point Input to transaction.dat; designate Output to a folder called output on your S3 instance. **Note that this output folder should not pre-exist**. Add this step and then wait for your program to complete. After

it’s completed, you can check and download results from your S3 bucket -> output folder.




12. Last but not least, **DO NOT FORGET TO CLEAN UP RESOURCSES!!** Terminate the cluster, delete all S3 buckets under your account, and always double check.

Reference:
https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/