Code test Junior Data Operations Engineer Clarity.ai (Notebook)

In [912]:

```
import numpy as np
import random
import time
import glob
import os
import sys
import unittest
import collections
from collections import Counter
```

In [856]:

```
#%%timeit

#every list comprehension can be rewritten in for loop, but every for loop can’t be rewritten in
the form of list comprehension.
def connected_hostnames(logpath, init_datetime, end_datetime, Hostname):
    connected_hosts=[]

    input_log=open(logpath)
    for line in input_log:
        #Check if within the interval
        if (init_datetime<= int(line.split()[0]) <= end_datetime):
            #check if the host initialized the connection and append the receiver if true.
            if(line.split()[1]==Hostname):
                connected_hosts.append(line.split()[2])
            #check if the host received the connection and append the initializer if true.
            elif (line.split()[2]==Hostname):
                connected_hosts.append(line.split()[1])
        #Exit early: finish the process if the interval is exceeded
        elif ((int(line.split()[0]) > end_datetime )):
                break
    input_log.close()
    return collections.Counter(connected_hosts)

'''
def connected_hostnames_one_liner(filepath, init_datetime, end_datetime, Hostname):
    s=[line.split()[2] if (init_datetime <= int(line.split()[0]) <= end_datetime and
line.split()[1]==Hostname) else line.split()[1] if (init_datetime<= int(line.split()[0]) <= end_
datetime and line.split()[2]==Hostname) else None for line in reversed(list(open(filepath)))]
    return list(filter(None, s))
'''
```

In [585]:

```
%%timeit
connected_hostnames("data/input-file.txt",1565647205599,1565679364288, 'Jadon')
```

```
4.02 ms ± 44.7 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [607]:

```
%%timeit
connected_hostnames_one_liner("data/input-file.txt",1565647205599,1565679364288, 'Jadon')
```

```
13.5 ms ± 1.07 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [913]:

```
def connected_to(logpath, init_datetime, end_datetime, Hostname):
    hostnames=[]
    input_log=open(logpath)
    for line in reversed(list(input_log)):
        #print(''.join(['parsed line: ',line]))

        if (int(line.split()[0]) >= init_datetime and  int(line.split()[0])<= end_datetime and li
ne.split()[2]==Hostname):
```

```
            #print(''.join(['----> considered line: ',line]))
            hostnames.append(line.split()[1])

        if(int(line.split()[0]) < init_datetime ):
            break

    #print('------------------ \n\n')
    input_log.close()
    return collections.Counter(hostnames)
```

In [901]:

```
def received_from(logpath, init_datetime, end_datetime, Hostname):
    hostnames=[]
    input_log=open(logpath)
    for line in reversed(list(input_log)):
        #print(''.join(['parsed line: ',line]))


        if (int(line.split()[0]) >= init_datetime and  int(line.split()[0])<= end_datetime and line.split()[1]==Hostname):
            #print(''.join(['----> considered line: ',line]))
            hostnames.append(line.split()[2])

        if(int(line.split()[0]) < init_datetime ):
            break
    #print('------------------ \n\n')
    input_log.close()
    return collections.Counter(hostnames)
```

In [902]:

```
def generated_conn(logpath, init_datetime, end_datetime):
    hostnames=[]
    input_log=open(logpath)
    for line in reversed(list(input_log)):
        #print(''.join(['parsed line: ',line]))
        if (int(line.split()[0]) >= init_datetime and  int(line.split()[0])<= end_datetime ):
            #print(''.join(['----> considered line: ',line]))
            hostnames.append(line.split()[1])

        if(int(line.split()[0]) < init_datetime ):
            break

    #print('------------------ \n\n')
    input_log.close()
    return collections.Counter(hostnames)
```

In [889]:

```
'''
strings in Python are immutable, and the “+” operation involves creating a new string and copying the old content
at each step. A more efficient approach would be to use the array module to modify the individual characters and
then use the join() function to re-create your final string.
'''

def process_log_files(Hostname, past_time, log_ofo_time):


    #can achieve the same effect slightly faster by using while 1. This is a single jump operation, as it is a numerical comparison.
    while 1:
        connected_hosts, received_hosts, active_hosts=Counter(),Counter(),Counter()



        init_datetime=int((time.time()-past_time)*1000)
        end_datetime=int(time.time()*1000)
        past= time.time() - 5 # 5 seconds


        past_files=sorted( [ filename for filename in glob.glob("output/*.txt") if os.path.getmtime(filename)>=init_datetime/1000-log_ofo_time ] , key=os.path.getmtime)[::-1]
        for filename in past_files:
            connected_hosts+=connected_to(filename,init_datetime,end_datetime,Hostname)
            received_hosts+=received_from(filename,init_datetime,end_datetime,Hostname)
            active_hosts+=most_generated_conn(filename,init_datetime,end_datetime)

        '''
        ## Data transformation for display :
            #converting 2d list into 1d , and consider multiple occurences by applying
collection
        connected_hosts=collections.Counter(sum(connected_hosts,[]))
        received_hosts=collections.Counter(sum(received_hosts,[]))
            #convert to collection to include other hosts if they have similar occurences as the
first one.
        active_hosts= collections.Counter(sum(active_hosts,[]))
        '''
        active_hosts=[h for h in active_hosts.most_common() if h[1]==active_hosts.most_common(1)[
0][1]]


        print(" ".join(['Hosts that connected to ', Hostname ,'in the last', str(past_time),'s ar
e: ',str(connected_hosts),'\n']))
        print(" ".join(['Hosts that received connection from', Hostname ,'in the last', str(past_
time),'s are: ',str(received_hosts),'\n']))
        print(" ".join(['the hostname that generated most connections in the last', str(past_time
),'s is: ',  str(active_hosts),'\n']))

        print('--------------------------------\n\n')

        print(''.join(['It is :  ', time.strftime('%X %x'),'.  the next output is in ', str(past_
time), ' s. \n']))
        time.sleep(past_time)
```

In [918]:

```
process_log_files('Hannibal', 5000 , 0 )
```

```
Hosts that connected to  Hannibal in the last 5000 s are:  Counter({'Hannibal': 157, 'Steeve':
154, 'Hanny': 151})

Hosts that received connection from Hannibal in the last 5000 s are:  Counter({'Hanny': 159, 'Han
nibal': 157, 'Steeve': 135})

the hostname that generated most connections in the last 5000 s is:  [('Steeve', 475)]

--------------------------------



It is :  14:17:53 12/14/20.  the next output is in 5000 s.


---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-918-4df0324143a0> in <module>
----> 1 process_log_files('Hannibal', 5000 , 0 )

<ipython-input-889-b62a41481b28> in process_log_files(Hostname, past_time, log_ofo_time)
     45 
     46         print(''.join(['It is :  ', time.strftime('%X %x'),'.  the next output is in ', s
tr(past_time), ' s. \n']))
---> 47         time.sleep(past_time)
     48 
     49 

KeyboardInterrupt: 
```

In [916]:

```
class NamesTestCase(unittest.TestCase):

# Test connected_hostnames() on short and long files
    def test_connected_hostnames_sf(self):
        result = connected_hostnames("data/input_test_case_1.txt",1607880434801,1607880438820, 'S
teeve')
        self.assertEqual(result, {'Hanny': 1, 'Hannibal': 2})

    def test_connected_hostnames_lf(self):
        result = connected_hostnames("data/input-file.txt",1565647204351,1565733598341, 'Dristen'
)
        self.assertEqual(result, {'Aadison': 1, 'Wilkens': 1, 'Kahlina': 1, 'Alei': 1, 'Zhanasia'
: 1, 'Jamor': 1, 'Joy': 1})


# Test connected_to() on short and long files
    def test_connect_to_sf(self):
        result = connected_to("data/input_test_case_1.txt",1607880434801,1607880438820, 'Steeve')
        self.assertEqual(result, {'Hannibal': 1})

    def test_connect_to_lf(self):
        result = connected_to("data/input-file.txt",1565647204351,1565733598341, 'Jadon')
        self.assertEqual(result, {'Ahmya': 1, 'Kayleann': 1, 'Shainah': 1, 'Aniyah': 1, 'Eveleigh
': 1, 'Caris': 1, 'Rahniya': 1, 'Remiel': 1})


# Test received_from() on short and long files
    def test_received_from_sf(self):
        result = received_from("data/input_test_case_1.txt",1607880434801,1607880438820, 'Steeve'
)
        self.assertEqual(result, {'Hannibal': 1, 'Hanny': 1})

    def test_received_from_lf(self):
        result = received_from("data/input-file.txt",1565647204351,1565733598341, 'Dristen')
        self.assertEqual(result, {'Joy': 1, 'Jamor': 1, 'Zhanasia': 1, 'Alei': 1, 'Kahlina': 1, '
Wilkens': 1, 'Aadison': 1})

# Test generated_conn
    def test_generated_conn(self):
        result = generated_conn("data/input_test_case_1.txt",1607880434801,1607880438820)
        self.assertEqual(result, {'Hannibal': 3, 'Steeve': 2, 'Hanny': 1})

if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.......
----------------------------------------------------------------------
Ran 7 tests in 0.136s

OK
```

In [872]:

```
connected_hostnames("data/input_test_case_1.txt",1607880434801,1607880438820, 'Steeve')
```

Out[872]:

```
Counter({'Hanny': 1, 'Hannibal': 2})
```

In [897]:

```
connected_hostnames("data/input-file.txt",1565647204351,1565733598341, 'Dristen')
```

Out[897]:

```
Counter({'Aadison': 1,
         'Wilkens': 1,
         'Kahlina': 1,
         'Alei': 1,
         'Zhanasia': 1,
         'Jamor': 1,
         'Joy': 1})
```

In [861]:

```
connected_to("data/input_test_case_1.txt",1607880434801,1607880438820, 'Steeve')
```

Out[861]:

```
Counter({'Hannibal': 1})
```

In [904]:

```
connected_to("data/input-file.txt",1565647204351,1565733598341, 'Jadon')
```

Out[904]:

```
Counter({'Ahmya': 1,
         'Kayleann': 1,
         'Shainah': 1,
         'Aniyah': 1,
         'Eveleigh': 1,
         'Caris': 1,
         'Rahniya': 1,
         'Remiel': 1})
```

In [905]:

```
received_from("data/input_test_case_1.txt",1607880434801,1607880438820, 'Steeve')
```

Out[905]:

```
Counter({'Hannibal': 1, 'Hanny': 1})
```

In [903]:

```
received_from("data/input-file.txt",1565647204351,1565733598341, 'Dristen')
```

Out[903]:

```
Counter({'Joy': 1,
         'Jamor': 1,
         'Zhanasia': 1,
         'Alei': 1,
         'Kahlina': 1,
         'Wilkens': 1,
         'Aadison': 1})
```

In [892]:

```
generated_conn("data/input_test_case_1.txt",1607880434801,1607880438820)
```

Out[892]:

```
Counter({'Hannibal': 3, 'Steeve': 2, 'Hanny': 1})
```

In [895]:

```
most_generated_conn("data/input-file.txt",1565647204351,1565733598341).most_common(1)
```

Out[895]:

```
[('Dristen', 7)]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [915]:

```
import logging
import threading
import time

def thread_function(name):
    logging.info("Thread %s: starting", name)
    time.sleep(2)
    logging.info("Thread %s: finishing", name)
```

```
if __name__ == "__main__":
    format = "%(asctime)s: %(message)s"
    logging.basicConfig(format=format, level=logging.INFO,
                        datefmt="%H:%M:%S")

    threads = list()
    for index in range(3):
        logging.info("Main    : create and start thread %d.", index)
        x = threading.Thread(target=thread_function, args=(index,))
        threads.append(x)
        x.start()

    for index, thread in enumerate(threads):
        logging.info("Main    : before joining thread %d.", index)
        thread.join()
        logging.info("Main    : thread %d done", index)
```

```
14:17:24: Main    : create and start thread 0.
14:17:24: Thread 0: starting
14:17:24: Main    : create and start thread 1.
14:17:24: Thread 1: starting
14:17:24: Main    : create and start thread 2.
14:17:24: Thread 2: starting
14:17:24: Main    : before joining thread 0.
14:17:26: Thread 0: finishing
14:17:26: Thread 1: finishing
14:17:26: Main    : thread 0 done
14:17:26: Main    : before joining thread 1.
14:17:26: Thread 2: finishing
14:17:26: Main    : thread 1 done
14:17:26: Main    : before joining thread 2.
14:17:26: Main    : thread 2 done
```

In [ ]:

In [ ]:

In [ ]: