```
    ############# Application #4 - Part #1 #############

#Configure the permissions on the script first! 'chmod 755 script.py"

#Make sure to have SSHv2 enabled and RSA 1024 bit key generated on every device!


import MySQLdb as mdb
import paramiko
import threading
import os.path
import subprocess
import datetime
import time
import sys
import re

#Module for output coloring
from colorama import init, deinit, Fore, Style


# Procedure for configuring Linux scheduler:
# root@kali:/# crontab -l   view scheduled tasks
# root@kali:/# crontab -e   edit scheduler
# Add the following line to run the script every 5 minutes, every hour, every day, every month:
```

```
# */5 * * * * /path_to_file/NetMon_SQL_v1.py /path_to_file/NETWORK_IP
/path_to_file/SSH_USERPASS.txt /path_to_file/SQL_CONN.txt

# For more info about configuring scheduler: http://kvz.io/blog/2007/07/29/schedule-tasks-on-linux-
using-crontab/

# Before scheduling this task, run the script in the console to check for errors:

# Go to the folder containing the script and all files, using cd /netmon_folder_path

# Enter this command: python NetMon_SQL_v1.py NETWORK_IP.txt SSH_USERPASS.txt SQL_CONN.txt

# Check the console output and SQL_Error_Log.txt file for any errors.

# Running the script is recommended at intervals of at least 5 minutes.


#Initialize colorama

init()


#Checking number of arguments passed into the script

if len(sys.argv) == 4:

    ip_file = sys.argv[1]

    user_file = sys.argv[2]

    sql_file = sys.argv[3]


    print Fore.BLUE + Style.BRIGHT + "\n\n* The script will be executed using files:\n"

    print Fore.BLUE + "Cisco network IP file is: " + Fore.YELLOW + "%s" % ip_file

    print Fore.BLUE + "SSHv2 connection file is: " + Fore.YELLOW + "%s" % user_file

    print Fore.BLUE + "MySQL connection file is: " + Fore.YELLOW + "%s" % sql_file

    print Fore.BLUE + Style.BRIGHT + "\n"


else:
    print Fore.RED + Style.BRIGHT + "\nIncorrect number of arguments (files) passed into the script."

    print Fore.RED + "Please try again.\n"

    sys.exit()



#Checking IP address file and content validity

def ip_is_valid():

    check = False

    global ip_list


    while True:

        #Changing exception message

        try:

            #Open user selected file for reading (IP addresses file)

            selected_ip_file = open(ip_file, 'r')


            #Starting from the beginning of the file

            selected_ip_file.seek(0)


            #Reading each line (IP address) in the file

            ip_list = selected_ip_file.readlines()


            #Closing the file

            selected_ip_file.close()
        except IOError:

            print Fore.RED + "\n* File %s does not exist! Please check and try again!\n" % ip_file

            sys.exit()


        #Checking octets

        for ip in ip_list:

            a = ip.split('.')


            if (len(a) == 4) and (1 <= int(a[0]) <= 223) and (int(a[0]) != 127) and (int(a[0]) != 169 or int(a[1]) != 
254) and (0 <= int(a[1]) <= 255 and 0 <= int(a[2]) <= 255 and 0 <= int(a[3]) <= 255):

                check = True

                break


            else:

                print '\n* There was an INVALID IP address! Please check and try again!\n'

                check = False

                continue


                        #Evaluating the 'check' flag

        if check == False:

            sys.exit()


        elif check == True:

            break


    #Checking IP reachability
```

```
print "* Checking IP reachability... Please wait...\n"


check2 = False


while True:
    for ip in ip_list:
        ping_reply = subprocess.call(['ping', '-c', '3', '-w', '3', '-q', '-n', ip], stdout = subprocess.PIPE)


              if ping_reply == 0:
            check2 = True
            continue


        elif ping_reply == 2:
            print Fore.RED + "\n* No response from device %s." % ip
            check2 = False
            break


        else:
            print Fore.RED + "\n* Ping to the following device has FAILED:", ip
            check2 = False
            break


    #Evaluating the 'check' flag
    if check2 == False:
        print Fore.RED + "* Please re-check IP address list or device.\n"
```

```
            sys.exit()

        elif check2 == True:
            print '\n* All devices are reachable. Checking SSHv2 connection file...\n'
            break

#Checking user file validity
def user_is_valid():
    global user_file

    while True:
        #Changing output messages
        if os.path.isfile(user_file) == True:
            print "\n* SSHv2 connection file has been validated. Checking MySQL connection file...\n"
            break

        else:
            print Fore.RED + "\n* File %s does not exist! Please check and try again!\n" % user_file
            sys.exit()

#Checking SQL connection command file validity
def sql_is_valid():
    global sql_file

    while True:
        #Changing output messages

        if os.path.isfile(sql_file) == True:

            print "\n* MySQL connection file has been validated...\n"

            print "\n* Any MySQL errors will be logged to: " + Fore.YELLOW + "SQL_Error_Log.txt\n" +
Fore.BLUE

            print "\n* Reading network data and writing to MySQL...\n"

            break


        else:

            print Fore.RED + "\n* File %s does not exist! Please check and try again!\n" % sql_file

            sys.exit()


#Change exception message

try:

    #Calling IP validity function

    ip_is_valid()


except KeyboardInterrupt:

    print Fore.RED + "\n\n* Program aborted by user. Exiting...\n"

    sys.exit()


#Change exception message

try:

    #Calling user file validity function

    user_is_valid()
```

```
except KeyboardInterrupt:
    print Fore.RED + "\n\n* Program aborted by user. Exiting...\n"
    sys.exit()

#Change exception message
try:
    #Calling MySQL file validity function
    sql_is_valid()

except KeyboardInterrupt:
    print Fore.RED + "\n\n* Program aborted by user. Exiting...\n"
    sys.exit()

    ############# Application #4 - Part #2 #############

check_sql = True
def sql_connection(command, values):
    global check_sql

    #Define SQL connection parameters
    selected_sql_file = open(sql_file, 'r')

    #Starting from the beginning of the file
    selected_sql_file.seek(0)
```

```
sql_host = selected_sql_file.readlines()[0].split(',')[0]


#Starting from the beginning of the file
selected_sql_file.seek(0)


sql_username = selected_sql_file.readlines()[0].split(',')[1]


#Starting from the beginning of the file
selected_sql_file.seek(0)


sql_password = selected_sql_file.readlines()[0].split(',')[2]


#Starting from the beginning of the file
selected_sql_file.seek(0)


sql_database = selected_sql_file.readlines()[0].split(',')[3].rstrip("\n")


#Connecting and writing to database
try:
    sql_conn = mdb.connect(sql_host, sql_username, sql_password, sql_database)


    cursor = sql_conn.cursor()


    cursor.execute("USE NetMon")
```

```
        cursor.execute(command, values)

        #Commit changes
        sql_conn.commit()

    except mdb.Error, e:
        sql_log_file = open("SQL_Error_Log.txt", "a")

        #Print any SQL errors to the error log file
        print >>sql_log_file, str(datetime.datetime.now()) + ": Error %d: %s" % (e.args[0],e.args[1])

        #Closing sql log file:
        sql_log_file.close()

        #Setting check_sql flag to False if any sql error occurs
        check_sql = False

    #Closing the sql file
    selected_sql_file.close()


#Initialize the necessary lists and dictionaries
cpu_values = []
io_mem_values = []
proc_mem_values = []
```

```
upint_values = []


top3_cpu = {}

top3_io_mem = {}

top3_proc_mem = {}

top3_upint = {}


#Open SSHv2 connection to devices

def open_ssh_conn(ip):

    global check_sql


    #Change exception message

    try:

        #Define SSH parameters

        selected_user_file = open(user_file, 'r')


        #Starting from the beginning of the file

        selected_user_file.seek(0)


                        #Reading the username from the file

        username = selected_user_file.readlines()[0].split(',')[0]


        #Starting from the beginning of the file

        selected_user_file.seek(0)
```

```
            #Reading the password from the file
password = selected_user_file.readlines()[0].split(',')[1].rstrip("\n")


#Logging into device
session = paramiko.SSHClient()


#For testing purposes, this allows auto-accepting unknown host keys
#Do not use in production! The default would be RejectPolicy
session.set_missing_host_key_policy(paramiko.AutoAddPolicy())


#Connect to the device using username and password
session.connect(ip, username = username, password = password)


#Start an interactive shell session on the router
connection = session.invoke_shell()


#Setting terminal length for entire output - disable pagination
connection.send("terminal length 0\n")
time.sleep(1)


#Entering global config mode
#connection.send("\n")
#connection.send("configure terminal\n")
#time.sleep(1)
```

```
#Reading commands from within the script

#Using the "\" line continuation character for better readability of the commands to be sent

selected_cisco_commands = '''show version | include (, Version|uptime is|bytes of memory|Hz)&\

                    show inventory&\

                    show interfaces | include bia&\

                    show processes cpu | include CPU utilization&\

                    show memory statistics&\

                    show ip int brief | include (Ethernet|Serial)&\

                    show cdp neighbors detail | include Device ID&\

                    show ip protocols | include Routing Protocol'''


#Splitting commands by the "&" character

command_list = selected_cisco_commands.split("&")


#Writing each line in the command string to the device

for each_line in command_list:

   connection.send(each_line + '\n')

   time.sleep(3)


#Closing the user file

selected_user_file.close()


#Checking command output for IOS syntax errors

output = connection.recv(65535)
```

```
if re.search(r"% Invalid input detected at", output):

    print Fore.RED + "* There was at least one IOS syntax error on device %s" % ip


else:

    print Fore.GREEN + "* All parameters were extracted from device %s" % ip,


#Test for reading command output

#print output + "\n"


      ############# Application #4 - Part #3 #############


#Extracting device parameters

#...starting with the ones destined to the NetworkDevices table in MySQL


dev_hostname = re.search(r"(.+) uptime is", output)

hostname = dev_hostname.group(1)

#print hostname


dev_mac = re.findall(r"\(bia (.+?)\)", output)

#print dev_mac

mac = dev_mac[0]

#print mac


dev_vendor = re.search(r"(.+?) (.+) bytes of memory", output)

vendor = dev_vendor.group(1)
```

```
#print vendor


dev_model = re.search(r"(.+?) (.+?) (.+) bytes of memory", output)

model = dev_model.group(2)

#print model


dev_image_name = re.search(r" \((.+)\), Version", output)

image_name = dev_image_name.group(1)

#print image_name


dev_os = re.search(r"\), Version (.+),", output)

os = dev_os.group(1)

#print os


serial_no = ""

if len(re.findall(r"(.+), SN: (.+?)\r\n", output)) == 0:

    serial_no = "unknown"

else:

    serial_no = re.findall(r"(.+), SN: (.+?)\r\n", output)[0][1].strip()

    #print serial_no


dev_uptime = re.search(r" uptime is (.+)\n", output)

uptime = dev_uptime.group(1)

uptime_value_list = uptime.split(', ')
```

```
#Getting the device uptime in seconds

y_sec = 0

w_sec = 0

d_sec = 0

h_sec = 0

m_sec = 0


for j in uptime_value_list:


    if 'year' in j:

        y_sec = int(j.split(' ')[0]) * 31449600


    elif 'week' in j:

        w_sec = int(j.split(' ')[0]) * 604800


    elif 'day' in j:

        d_sec = int(j.split(' ')[0]) * 86400


    elif 'hour' in j:

        h_sec = int(j.split(' ')[0]) * 3600


    elif 'minute' in j:

        m_sec = int(j.split(' ')[0]) * 60


total_uptime_sec = y_sec + w_sec + d_sec + h_sec + m_sec
```

```
#print total_uptime_sec


cpu_model = ""

if re.search(r".isco (.+?) \((.+)\) processor(.+)\n", output) == None:

    cpu_model = "unknown"

else:

    cpu_model = re.search(r".isco (.+?) \((.+)\) processor(.+)\n", output).group(2)

#print cpu_model


cpu_speed = ""

if re.search(r"(.+?)at (.+?)MHz(.+)\n", output) == None:

    cpu_speed = "unknown"

else:

    cpu_speed = re.search(r"(.+?)at (.+?)MHz(.+)\n", output).group(2)

#print cpu_speed


serial_int = ""

if re.findall(r"Serial([0-9]*)/([0-9]*) (.+)\n", output) == None:

    serial_int = "no serial"

else:

    serial_int = len(re.findall(r"Serial([0-9]*)/([0-9]*) (.+)\n", output))

#print serial_int


dev_cdp_neighbors = re.findall(r"Device ID: (.+)\r\n", output)

all_cdp_neighbors = ','.join(dev_cdp_neighbors)
```

```
        #print all_cdp_neighbors


        dev_routing_pro = re.findall(r"Routing Protocol is \"(.+)\"\r\n", output)

        #print dev_routing_pro

        is_internal = []

        is_external = []

        for protocol in dev_routing_pro:

            if 'bgp' in protocol:

                is_external.append(protocol)

            else:

                is_internal.append(protocol)


        internal_pro = ','.join(is_internal)

        external_pro = ','.join(is_external)


        #print internal_pro

        #print external_pro


            ############# Application #4 - Part #4 #############


                    ### CPU ###


        dev_cpu_util_per5min = re.search(r"CPU utilization for five seconds: (.+) five minutes: (.+?)%",
output)

        cpu_util_per5min = dev_cpu_util_per5min.group(2)

        #print cpu_util_per5min
```

```
#Append CPU value for each device to the cpu_values list

cpu_values.append(int(cpu_util_per5min))


#Get top 3 CPU devices

top3_cpu[hostname] = cpu_util_per5min


                    ### Processor Memory ###


dev_used_proc_mem = re.search(r"Processor(.+)\n ", output)

dev_used_proc_mem = dev_used_proc_mem.group(1)

#print dev_used_proc_mem


total_proc_mem = dev_used_proc_mem.split('   ')[2].strip()

used_proc_mem = dev_used_proc_mem.split('   ')[3].strip()

#print total_proc_mem

#print used_proc_mem


#Get percentage of used proc mem

proc_mem_percent = format(int(used_proc_mem) * 100 / float(total_proc_mem), ".2f")

#print proc_mem_percent


#Append used proc memory values for each device to the mem_values list

proc_mem_values.append(float(proc_mem_percent))
```

```
#Get top 3 proc memory devices
top3_proc_mem[hostname] = proc_mem_percent

          ### I/O Memory ###

dev_used_io_mem = re.search(r"      I/O(.+)\n", output)
dev_used_io_mem = dev_used_io_mem.group(1)
#print dev_used_io_mem

total_io_mem = dev_used_io_mem.split('   ')[2].strip()
used_io_mem = dev_used_io_mem.split('   ')[3].strip()
#print total_io_mem
#print used_io_mem

#Get percentage of used proc mem
io_mem_percent = format(int(used_io_mem) * 100 / float(total_io_mem), ".2f")
#print io_mem_percent

#Append used I/O memory values for each device to the mem_values list
io_mem_values.append(float(io_mem_percent))

#Get top 3 I/O memory devices
top3_io_mem[hostname] = io_mem_percent

          ### UP Interfaces ###
```

```
    dev_total_int = re.findall(r"([A-Za-z]*)Ethernet([0-9]*)(.+)YES(.+)\n", output)

    total_int = len(dev_total_int)

    #print total_int


    dev_total_up_int = re.findall(r"(.+)Ethernet([0-9]*)/([0-9]*)[\s]*(.+)up[\s]*up", output)

    total_up_int = len(dev_total_up_int)

    #print total_up_int


    #Get percentage of Eth UP interfaces out of the total number of Eth interfaces

    intf_percent = format(total_up_int * 100 / float(total_int), ".2f")

    #print intf_percent


    #Append percentage of UP interfaces for each device to the upint_values list

    upint_values.append(float(intf_percent))


    #Get top 3 UP Eth interfaces density devices

    top3_upint[hostname] = intf_percent


    #Insert/Update if exists all network devices data into the MySQL database table NetworkDevices.
Calling sql_connection function

    sql_connection("REPLACE INTO
NetworkDevices(Hostname,MACAddr,Vendor,Model,Image,IOSVersion,SerialNo,Uptime,CPUModel,CPU
Speed,SerialIntfNo,CiscoNeighbors,IntRoutingPro,ExtRoutingPro) VALUES(%s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s, %s)", (hostname, mac, vendor, model, image_name, os, serial_no, total_uptime_sec,
cpu_model, cpu_speed, serial_int, all_cdp_neighbors, internal_pro, external_pro))
```

```
        #Closing the SSH connection

        session.close()


    except paramiko.AuthenticationException:

        print Fore.RED + "* Invalid SSH username or password. \n* Please check the username/password
file or the device configuration!\n"

        check_sql = False


#Creating threads

def create_threads():

    threads = []

    for ip in ip_list:

        th = threading.Thread(target = open_ssh_conn, args = (ip,))   #args is a tuple with a single element

        th.start()

        threads.append(th)


    for th in threads:

        th.join()


#Calling threads creation function

create_threads()


    ############# Application #4 - Part #5 #############


#Poll date and time are based on the system clock
```

```
poll_timestamp = datetime.datetime.now()

#print poll_timestamp


###Testing code###

#print cpu_values

#print proc_mem_values

#print io_mem_values

#print upint_values


#print top3_cpu

#print top3_proc_mem

#print top3_io_mem

#print top3_upint

###


#Defining a function to get top 3 devices in CPU/mem/intf usage

def top3(each_dict):

    global top3_list

    top3 = []


    for host, usage in sorted(each_dict.items(), key = lambda x: x[1], reverse = True)[:3]:

        top3.append(host)

        top3_list = ",".join(top3)

    #print top3_list
```

```
#CPU average function

def cpu_average():

    try:

        cpu = sum(cpu_values) / float(len(cpu_values))


        #Calling the top3 function for the CPU dictionary

        top3(top3_cpu)


        #Write values to the MySQL database CPUUtilization table

        sql_connection("INSERT INTO
CPUUtilization(NetworkCPUUtilizationPercent,Top3CPUDevices,PollTimestamp) VALUES(%s, %s, %s)",
(cpu, top3_list, poll_timestamp))


    except ZeroDivisionError:

        print "* There was an error while computing a network parameter. No record has been added to
MySQL. Please retry."


cpu_average()


#Used proc memory average function

def mem_proc_average():

    try:

        mem_proc = sum(proc_mem_values) / float(len(proc_mem_values))


        #Calling the top3 function for the mem proc dictionary

        top3(top3_proc_mem)
        #Write values to the MySQL database ProcMemUtilization table

        sql_connection("INSERT INTO
ProcMemUtilization(NetworkProcMemUtilizationPercent,Top3ProcMemDevices,PollTimestamp)
VALUES(%s, %s, %s)", (mem_proc, top3_list, poll_timestamp))


    except ZeroDivisionError:

        print "* There was an error while computing a network parameter. No record has been added to
MySQL. Please retry."


mem_proc_average()


#Used I/O memory average function

def mem_io_average():

    try:

        mem_io = sum(io_mem_values) / float(len(io_mem_values))


        #Calling the top3 function for the mem I/O dictionary

        top3(top3_io_mem)


        #Write values to the MySQL database IOMemUtilization table

        sql_connection("INSERT INTO
IOMemUtilization(NetworkIOMemUtilizationPercent,Top3IOMemDevices,PollTimestamp) VALUES(%s,
%s, %s)", (mem_io, top3_list, poll_timestamp))


    except ZeroDivisionError:

        print "* There was an error while computing a network parameter. No record has been added to
MySQL. Please retry."
```

```
mem_io_average()


#Total UP Eth interfaces function

def upint_total():

    try:

        upint = sum(upint_values) / float(len(upint_values))



        #Calling the top3 function for the UP intf dictionary

        top3(top3_upint)



        #Write values to the MySQL database UPEthInterfaces table

        sql_connection("INSERT INTO
UPEthInterfaces(NetworkUPEthIntfPercent,Top3UPEthIntf,PollTimestamp) VALUES(%s, %s, %s)", (upint,
top3_list, poll_timestamp))



    except ZeroDivisionError:

        print "* There was an error while computing a network parameter. No record has been added to
MySQL. Please retry."



upint_total()



#print check_sql



if check_sql == True:

    print "\n* All parameters were successfully exported to MySQL."



else:
    print Fore.RED + "\n* There was a problem exporting data to MySQL.\n* Check the files, database and 
SQL_Error_Log.txt.\n"


#De-initialize colorama

deinit()


#End of program
```