PythonPing Network Scanner Threaded

Contents

Time required: 60 minutes

Python Tabs and Spaces Issue

Visual Studio Code automatically changes a tab into four spaces. Other editors, like geany and nano in Linux, do not. You can end up with a combination of spaces and tabs. Python doesn’t like a combination, it wants either one or the other. The preferred method is spaces.

Recommendation:

1. Create your Python files in Visual Studio Code in Windows.
2. Copy and paste the code into either nano or geany in Linux.

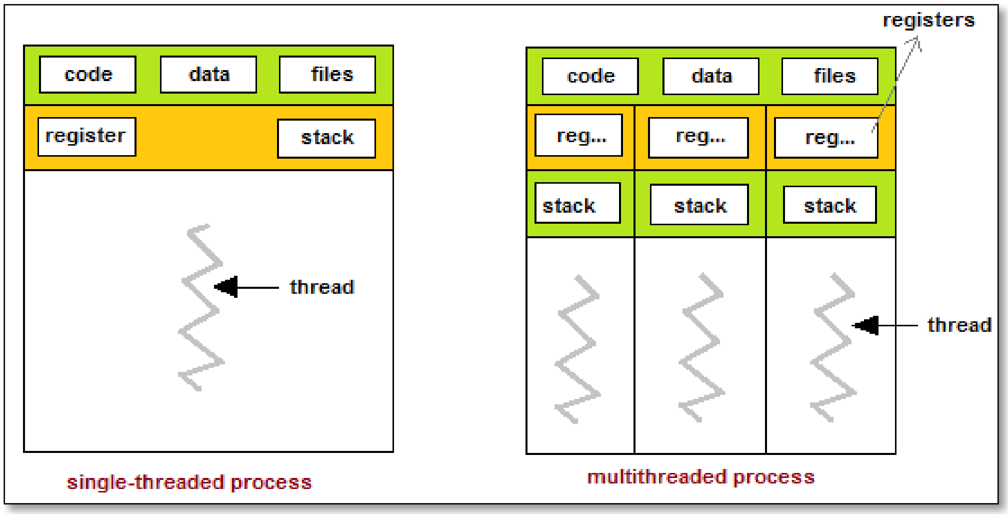
Objective: Write a cross platform Python script that uses branching, looping, multithreading, and pythonping to scan a local network.

Python Threading Tutorial

- https://www.pythontutorial.net/python-concurrency/python-threadpoolexecutor/

Threading

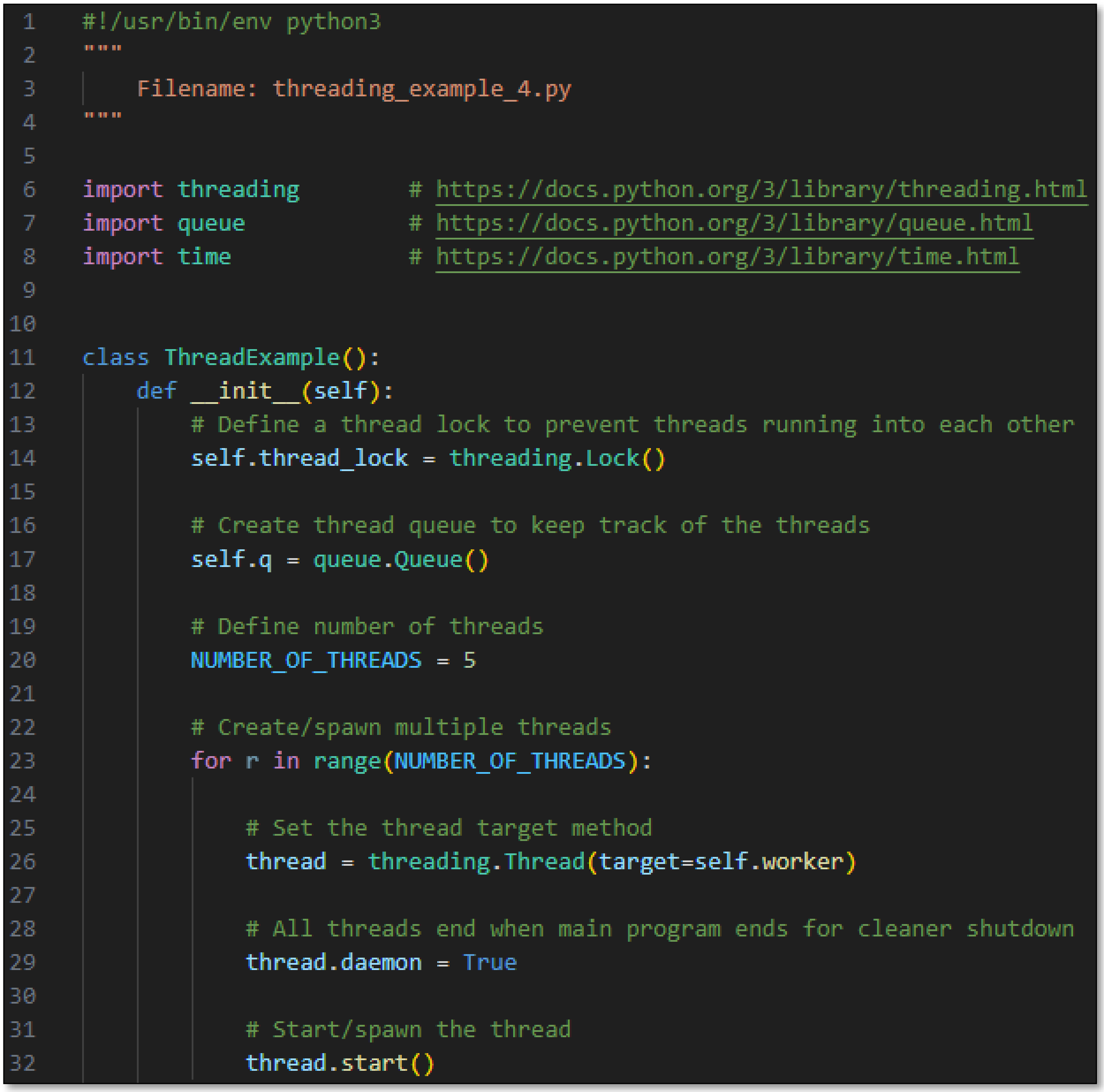
Threading in python is used to run multiple threads (tasks, function calls) at the same time. Python threads are used in cases where the execution of a task involves some waiting. One example would be interaction with a service hosted on another computer, such as a webserver. Threading allows python to execute other code while waiting.



Tutorial 1: Threads, Locks and Queues

References from Python.

- https://docs.python.org/3/library/threading.html
- https://docs.python.org/3/library/queue.html
- https://docs.python.org/3/library/time.html

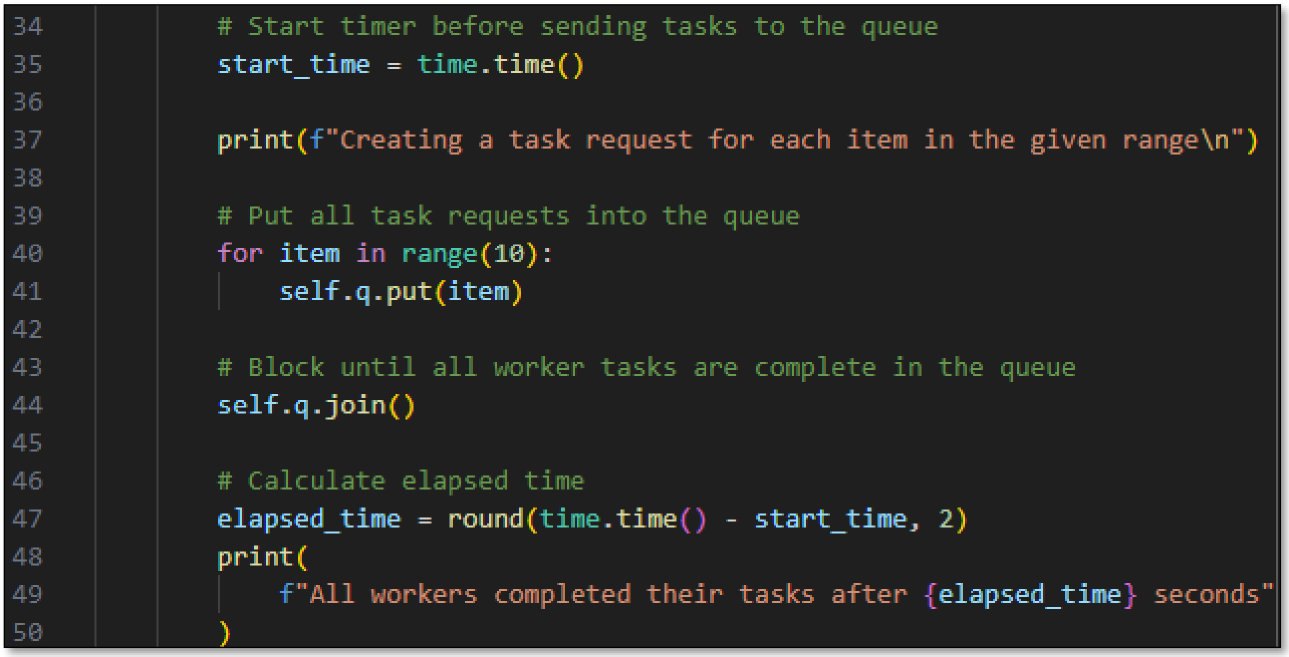


```python
#!/usr/bin/env python3
"""
    Filename: threading_example_4.py
"""

import threading        # https://docs.python.org/3/library/threading.html
import queue            # https://docs.python.org/3/library/queue.html
import time             # https://docs.python.org/3/library/time.html


class ThreadExample():
    def __init__(self):
        # Define a thread lock to prevent threads running into each other
        self.thread_lock = threading.Lock()

        # Create thread queue to keep track of the threads
        self.q = queue.Queue()

        # Define number of threads
        NUMBER_OF_THREADS = 5

        # Create/spawn multiple threads
        for r in range(NUMBER_OF_THREADS):

            # Set the thread target method
            thread = threading.Thread(target=self.worker)

            # All threads end when main program ends for cleaner shutdown
            thread.daemon = True

            # Start/spawn the thread
            thread.start()
```

threading.Thread(target=worker) sets the target method for the threads

thread.daemon = True creates a cleaner shutdown. All threads end when the main program ends.

thread.start() spawns the specified number of threads. These threads take turns going through the worker queue.

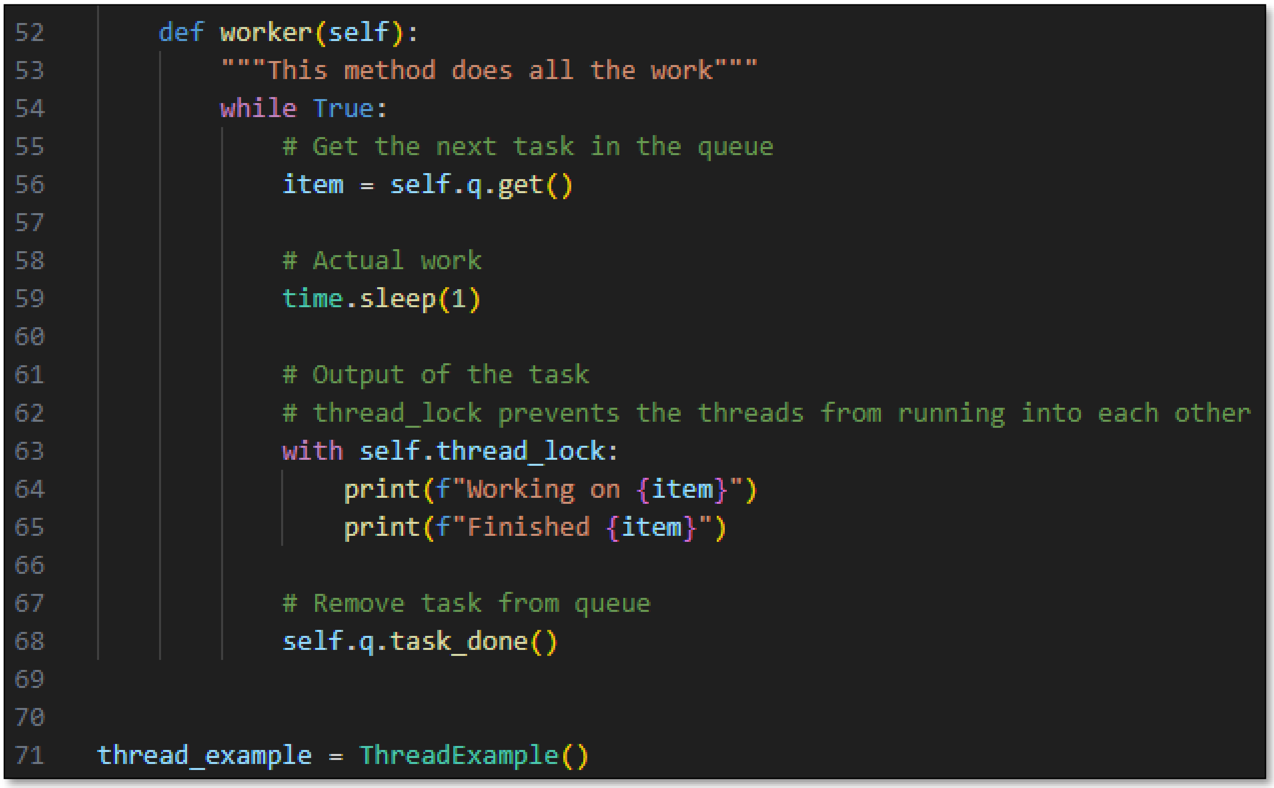


```
        # Start timer before sending tasks to the queue
        start_time = time.time()

        print(f"Creating a task request for each item in the given range\n")

        # Put all task requests into the queue
        for item in range(10):
            self.q.put(item)

        # Block until all worker tasks are complete in the queue
        self.q.join()

        # Calculate elapsed time
        elapsed_time = round(time.time() - start_time, 2)
        print(
            f"All workers completed their tasks after {elapsed_time} seconds"
        )
```

q.put() puts all items into the queue.

q.join() waits until the queue is empty before performing other operations.

When you call **q.join()** in the main thread, it block's the main threads until the workers have processed everything that's in the queue. It does not stop the worker threads, which continue executing their infinite loops. Daemon automatically quit when they are done. When all the work threads have joined, the program continues.



```python
    def worker(self):
        """This method does all the work"""
        while True:
            # Get the next task in the queue
            item = self.q.get()

            # Actual work
            time.sleep(1)

            # Output of the task
            # thread_lock prevents the threads from running into each other
            with self.thread_lock:
                print(f"Working on {item}")
                print(f"Finished {item}")

            # Remove task from queue
            self.q.task_done()


thread_example = ThreadExample()
```

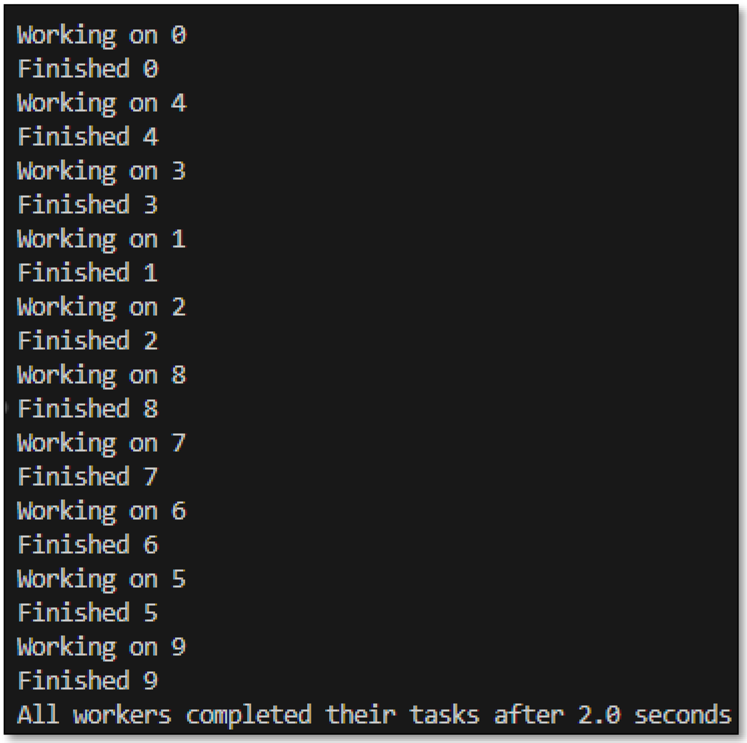
q.get() gets the next item in the queue to work on.

thread_lock prevents the threads from running over each other. Without this, the results would be printed on top of each other.

q.task_done() lets worker threads say when a task is done. It deletes an element from the queue. At the end of the join, the queue length is determined based on whether the queue length is zero. After that the main thread is executed.

There are 5 threads. Each worker task takes 1 second. The run time is 2 seconds.

Example run (Each example run will have a different order):



```
Working on 0
Finished 0
Working on 4
Finished 4
Working on 3
Finished 3
Working on 1
Finished 1
Working on 2
Finished 2
Working on 8
Finished 8
Working on 7
Finished 7
Working on 6
Finished 6
Working on 5
Finished 5
Working on 9
Finished 9
All workers completed their tasks after 2.0 seconds
```

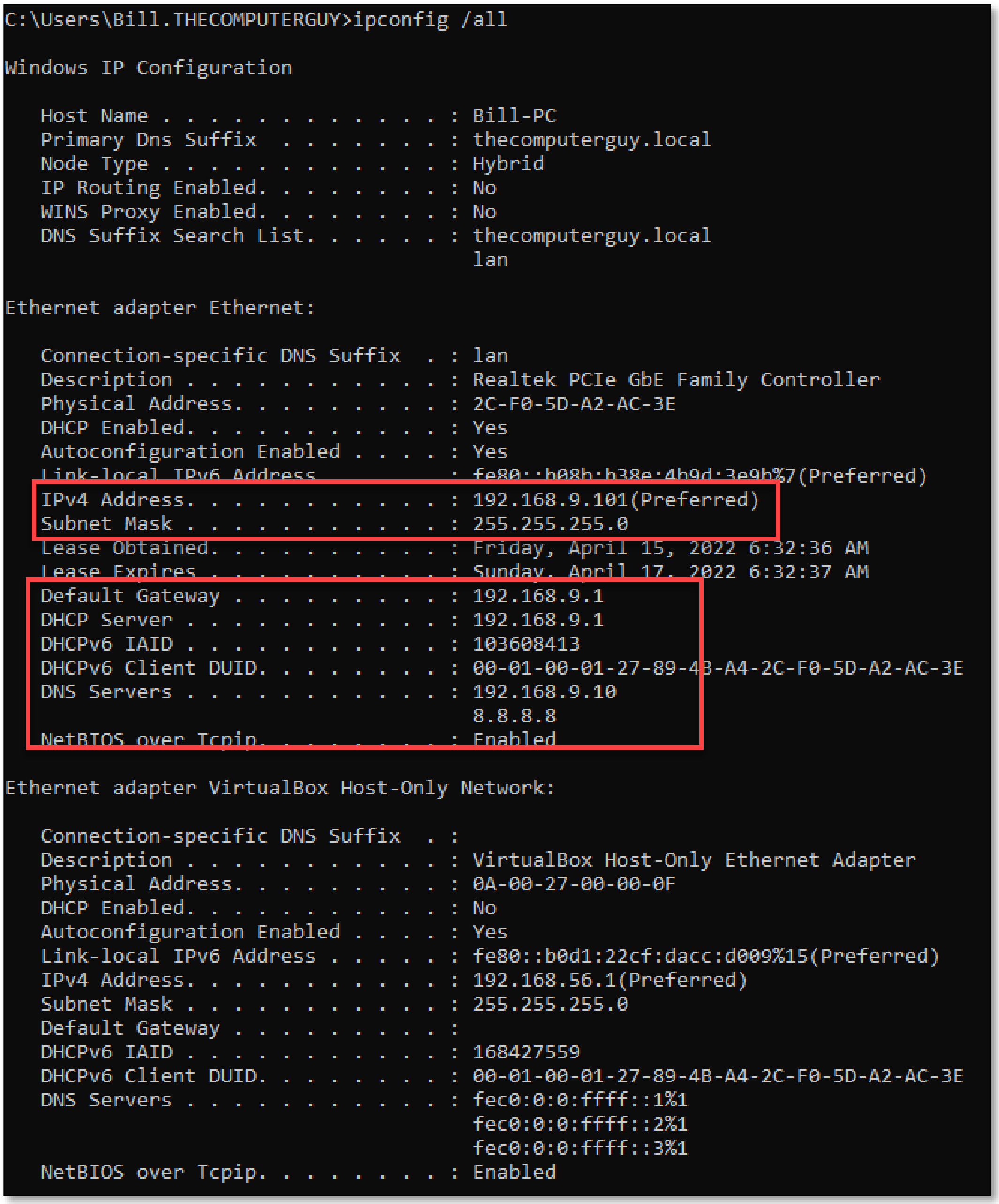
Find Your Network IP Address in Windows

Use the network address of your local network. Example: 192.168.0.0/24

NOTE: 192.168.56.1 is the VirtualBox adapter address, that is not your network address.

1. Enter the following command at the command prompt: **ipconfig /all**
2. The screenshot below shows my network at home, 192.168.9.0/24 Your IP address will probably be different. I have an Ethernet adapter, you may have a wireless adapter.
3. Notice that my IP address information includes a Default Gateway, DHCP Server, and DNS Servers. Those are needed for a functioning network connection.
4. Note that my IPv4 Address for my computer is **192.168.9.101** My subnet Mask is **255.255.255.0** This makes my network a standard Class C network.
    a. My network address is **192.168.9.0/24**

NOTE: If you are not sure about your network address, please contact me. You will get a 0 for this assignment if you do not provide a screenshot showing a successful scan of your network.



```
C:\Users\Bill.THECOMPUTERGUY>ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : Bill-PC
   Primary Dns Suffix  . . . . . . . : thecomputerguy.local
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : thecomputerguy.local
                                       lan

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : lan
   Description . . . . . . . . . . . : Realtek PCIe GbE Family Controller
   Physical Address. . . . . . . . . : 2C-F0-5D-A2-AC-3E
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::b08b:b38e:4b9d:3e9b%7(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.9.101(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Friday, April 15, 2022 6:32:36 AM
   Lease Expires . . . . . . . . . . : Sunday, April 17, 2022 6:32:37 AM
   Default Gateway . . . . . . . . . : 192.168.9.1
   DHCP Server . . . . . . . . . . . : 192.168.9.1
   DHCPv6 IAID . . . . . . . . . . . : 103608413
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-27-89-4B-A4-2C-F0-5D-A2-AC-3E
   DNS Servers . . . . . . . . . . . : 192.168.9.10
                                       8.8.8.8
   NetBIOS over Tcpip. . . . . . . . : Enabled

Ethernet adapter VirtualBox Host-Only Network:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : VirtualBox Host-Only Ethernet Adapter
   Physical Address. . . . . . . . . : 0A-00-27-00-00-0F
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::b0d1:22cf:dacc:d009%15(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.56.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 168427559
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-27-89-4B-A4-2C-F0-5D-A2-AC-3E
   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                       fec0:0:0:ffff::2%1
                                       fec0:0:0:ffff::3%1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

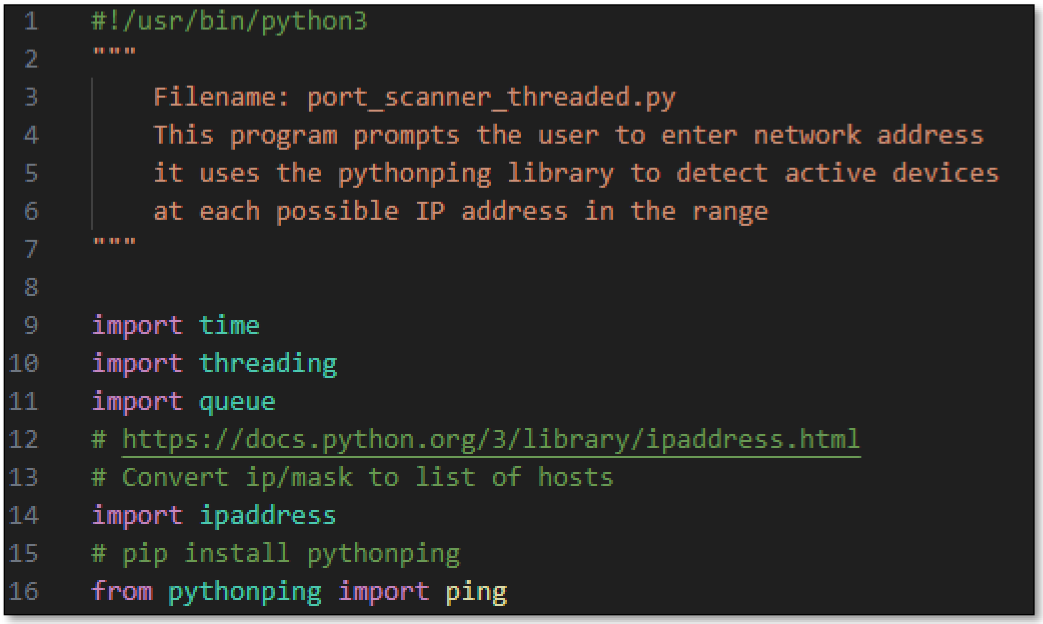
Tutorial 2: PythonPing Threaded Network Scanner

Security professionals often need to automate or create tools to help them conduct security tests. In this activity, you write a Python script that uses the ping command, threading, and a for loop to ping IP numbers for an entire class C network.

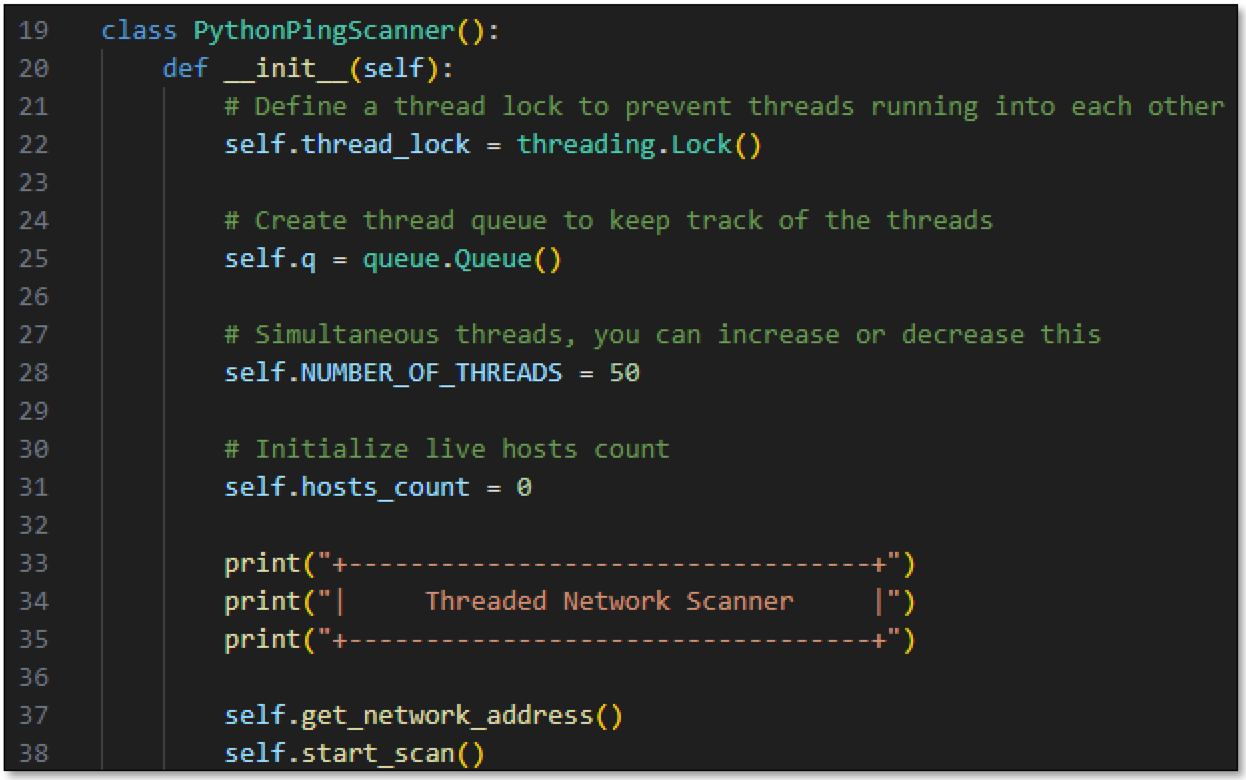
1. Create a Python program called: **pythonping_scanner_threaded.py**
2. Modify your code as follows including the comments.

Reference for ipaddress module.

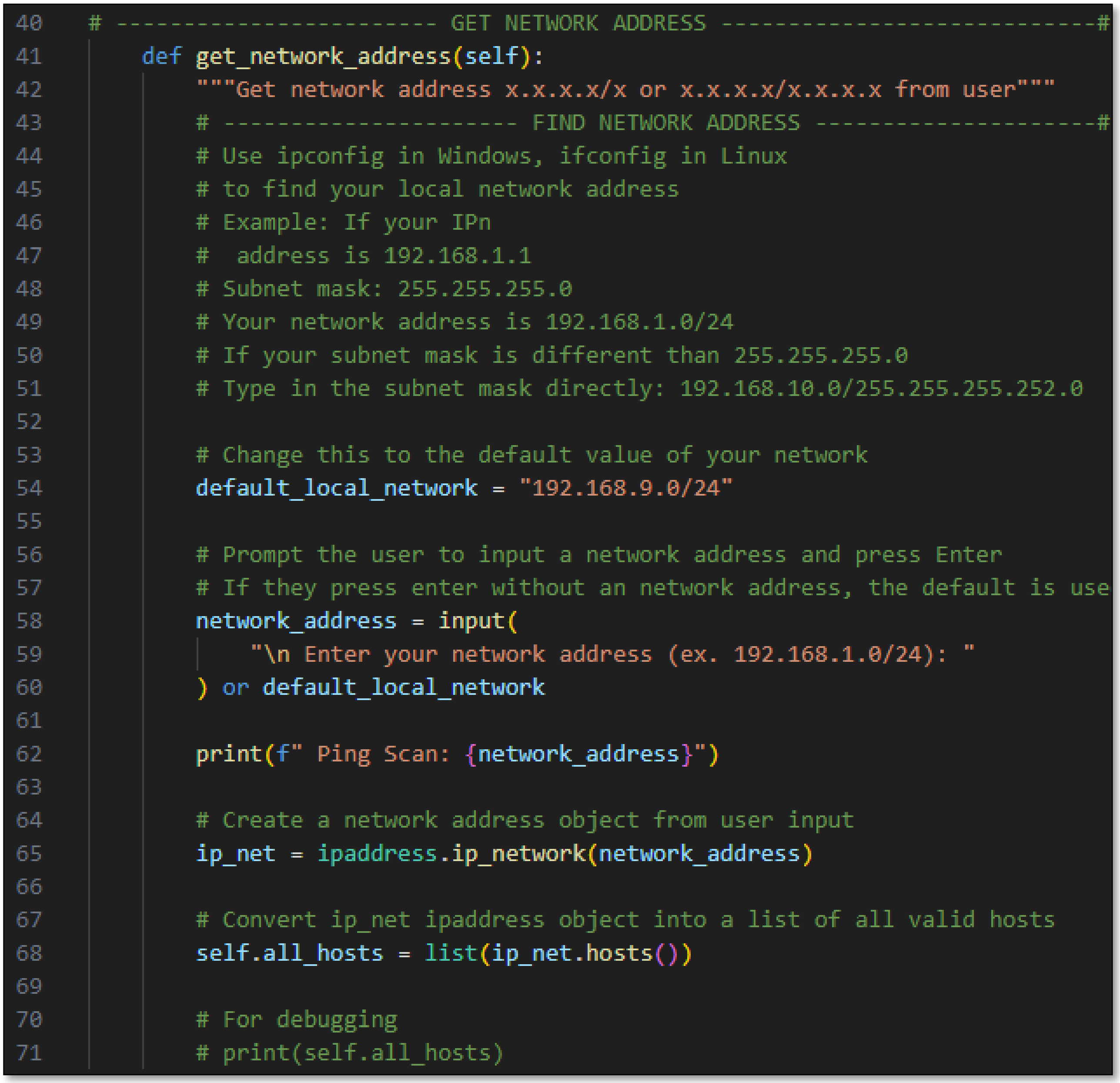
- https://docs.python.org/3/library/ipaddress.html
- Convert ip/mask to list of hosts



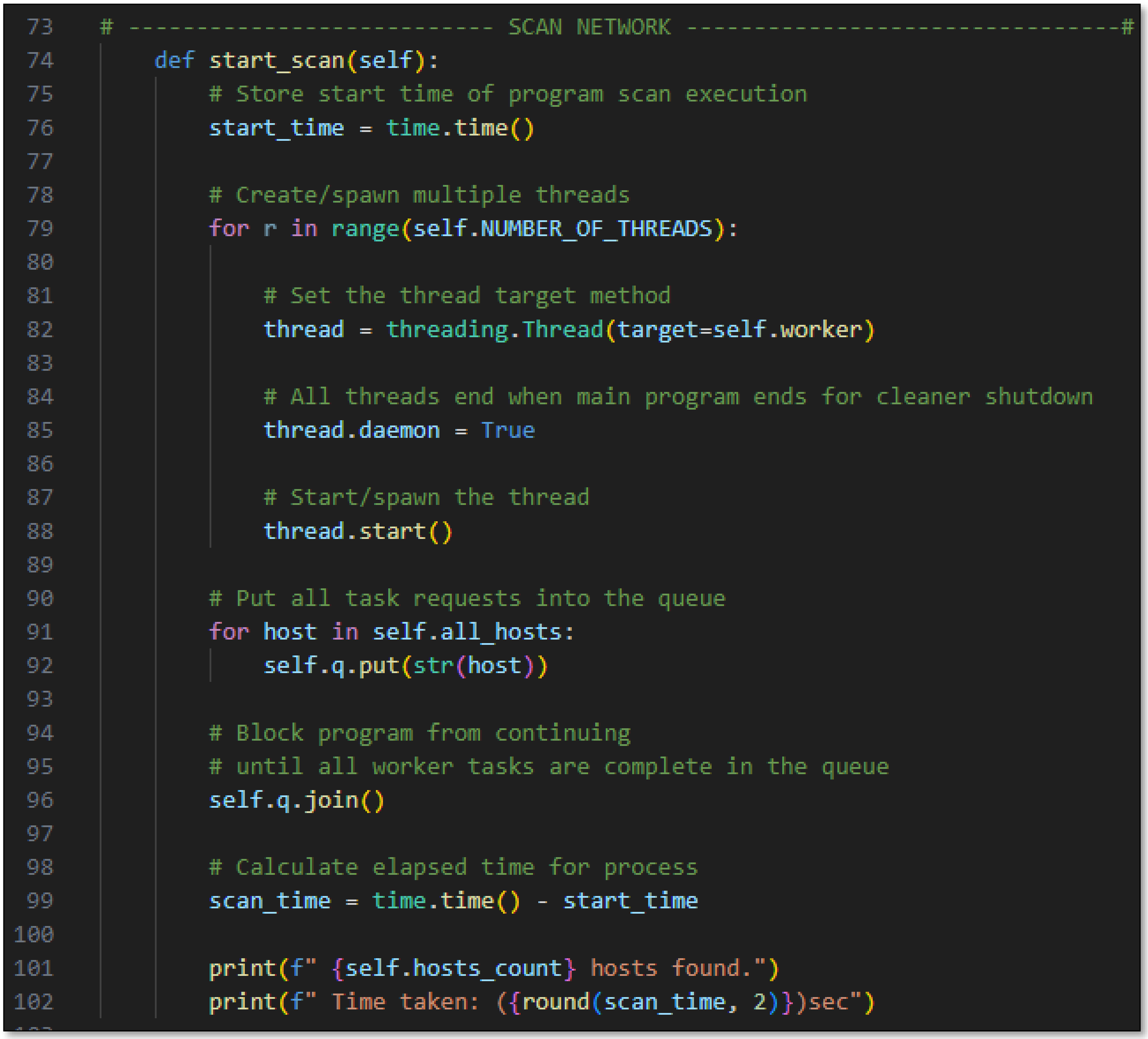
```python
#!/usr/bin/python3
"""
    Filename: port_scanner_threaded.py
    This program prompts the user to enter network address
    it uses the pythonping library to detect active devices
    at each possible IP address in the range
"""

import time
import threading
import queue
# https://docs.python.org/3/library/ipaddress.html
# Convert ip/mask to list of hosts
import ipaddress
# pip install pythonping
from pythonping import ping
```



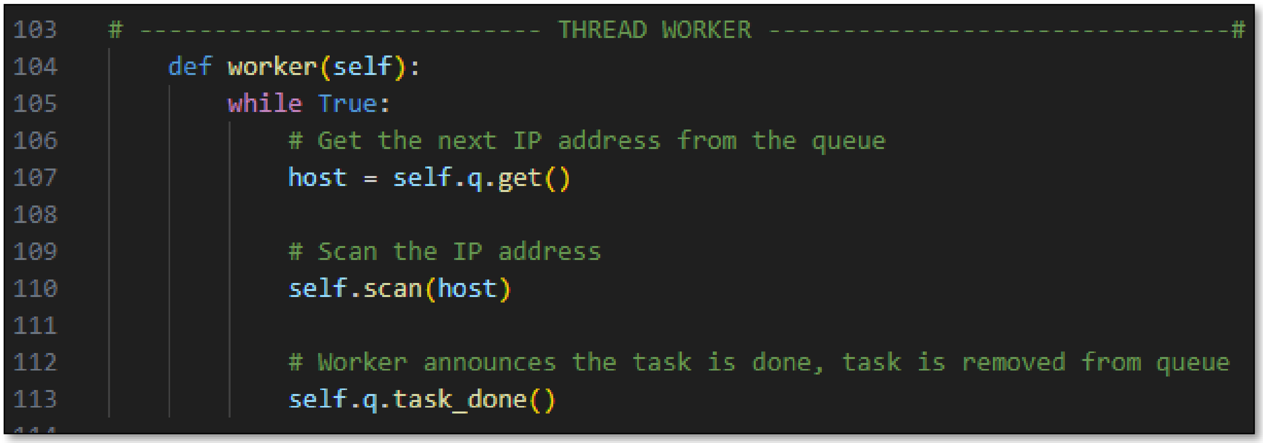
```python
class PythonPingScanner():
    def __init__(self):
        # Define a thread lock to prevent threads running into each other
        self.thread_lock = threading.Lock()

        # Create thread queue to keep track of the threads
        self.q = queue.Queue()

        # Simultaneous threads, you can increase or decrease this
        self.NUMBER_OF_THREADS = 50

        # Initialize live hosts count
        self.hosts_count = 0

        print("+----------------------------------+")
        print("|     Threaded Network Scanner     |")
        print("+----------------------------------+")

        self.get_network_address()
        self.start_scan()
```



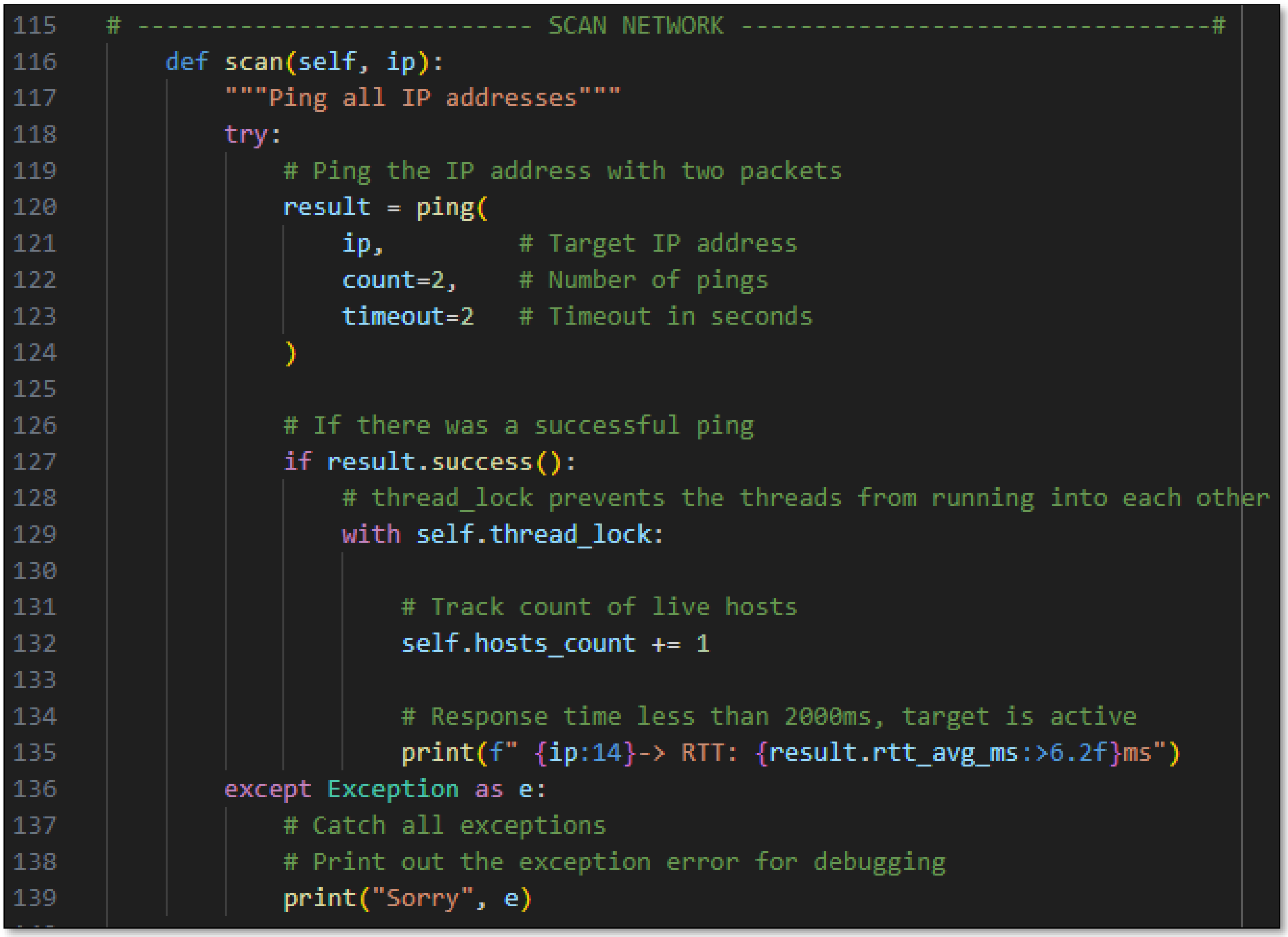
```python
# ------------------------ GET NETWORK ADDRESS --------------------------#
    def get_network_address(self):
        """Get network address x.x.x.x/x or x.x.x.x/x.x.x.x from user"""
        # ---------------------- FIND NETWORK ADDRESS ---------------------#
        # Use ipconfig in Windows, ifconfig in Linux
        # to find your local network address
        # Example: If your IPn
        #  address is 192.168.1.1
        # Subnet mask: 255.255.255.0
        # Your network address is 192.168.1.0/24
        # If your subnet mask is different than 255.255.255.0
        # Type in the subnet mask directly: 192.168.10.0/255.255.255.252.0

        # Change this to the default value of your network
        default_local_network = "192.168.9.0/24"

        # Prompt the user to input a network address and press Enter
        # If they press enter without an network address, the default is use
        network_address = input(
            "\n Enter your network address (ex. 192.168.1.0/24): "
        ) or default_local_network

        print(f" Ping Scan: {network_address}")

        # Create a network address object from user input
        ip_net = ipaddress.ip_network(network_address)

        # Convert ip_net ipaddress object into a list of all valid hosts
        self.all_hosts = list(ip_net.hosts())

        # For debugging
        # print(self.all_hosts)
```



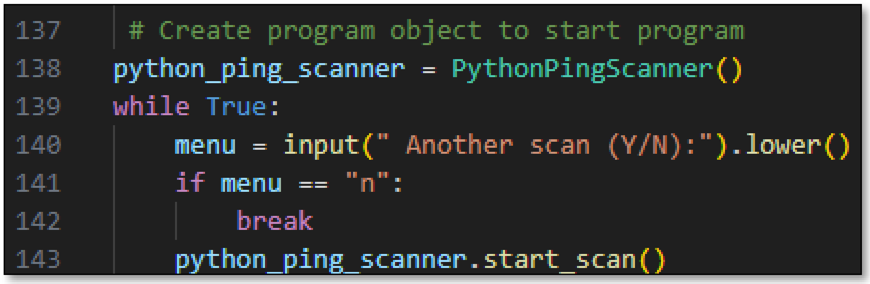
```
    # --------------------------- SCAN NETWORK --------------------------------#
        def start_scan(self):
            # Store start time of program scan execution
            start_time = time.time()

            # Create/spawn multiple threads
            for r in range(self.NUMBER_OF_THREADS):

                # Set the thread target method
                thread = threading.Thread(target=self.worker)

                # All threads end when main program ends for cleaner shutdown
                thread.daemon = True

                # Start/spawn the thread
                thread.start()

            # Put all task requests into the queue
            for host in self.all_hosts:
                self.q.put(str(host))

            # Block program from continuing
            # until all worker tasks are complete in the queue
            self.q.join()

            # Calculate elapsed time for process
            scan_time = time.time() - start_time

            print(f" {self.hosts_count} hosts found.")
            print(f" Time taken: ({round(scan_time, 2)})sec")
```



```
    # --------------------------- THREAD WORKER -------------------------------#
        def worker(self):
            while True:
                # Get the next IP address from the queue
                host = self.q.get()

                # Scan the IP address
                self.scan(host)

                # Worker announces the task is done, task is removed from queue
                self.q.task_done()
```

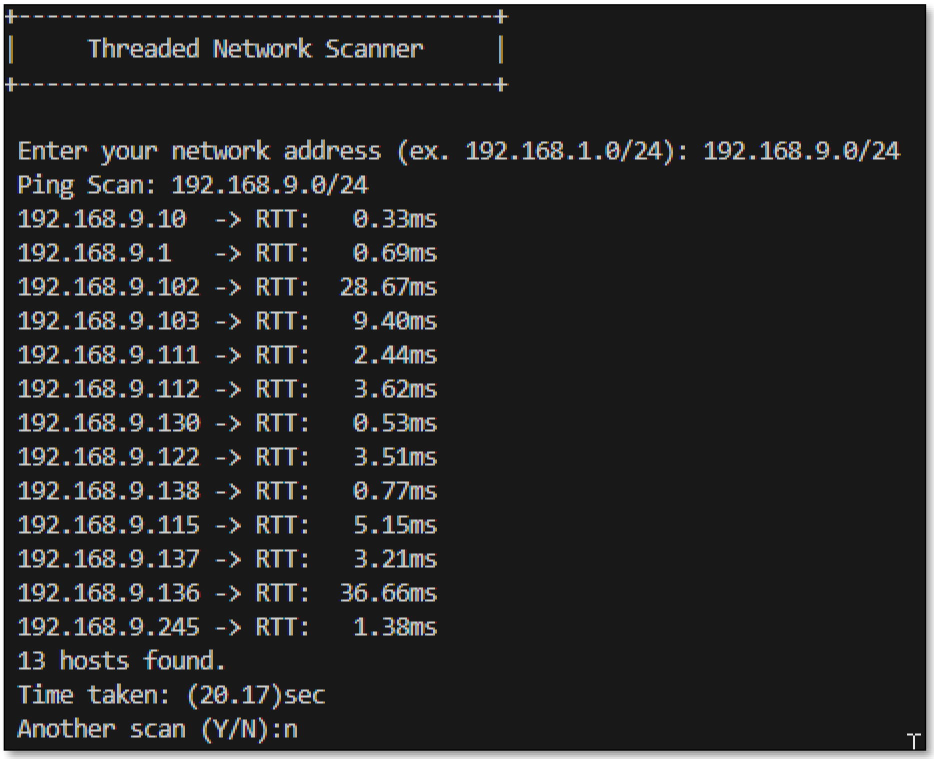


```python
# ---------------------------- SCAN NETWORK --------------------------------#
    def scan(self, ip):
        """Ping all IP addresses"""
        try:
            # Ping the IP address with two packets
            result = ping(
                ip,          # Target IP address
                count=2,     # Number of pings
                timeout=2    # Timeout in seconds
            )

            # If there was a successful ping
            if result.success():
                # thread_lock prevents the threads from running into each other
                with self.thread_lock:

                    # Track count of live hosts
                    self.hosts_count += 1

                    # Response time less than 2000ms, target is active
                    print(f" {ip:14}-> RTT: {result.rtt_avg_ms:>6.2f}ms")
        except Exception as e:
            # Catch all exceptions
            # Print out the exception error for debugging
            print("Sorry", e)
```



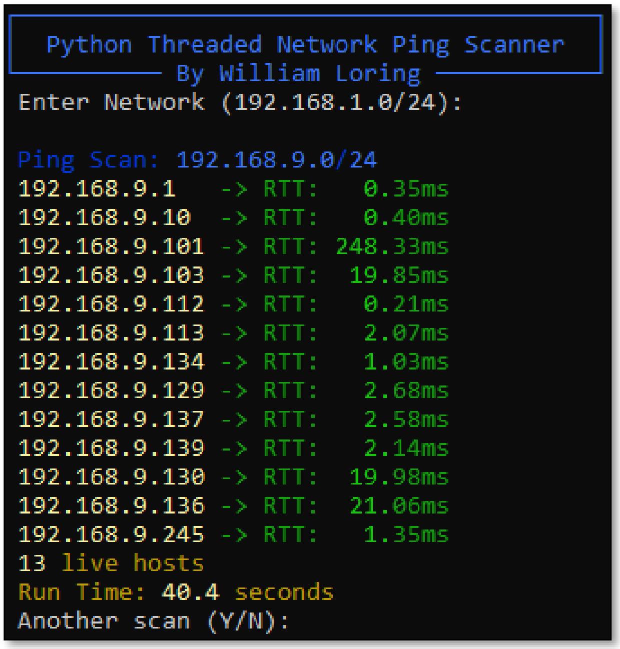
```python
 # Create program object to start program
python_ping_scanner = PythonPingScanner()
while True:
    menu = input(" Another scan (Y/N):").lower()
    if menu == "n":
        break
    python_ping_scanner.start_scan()
```

Example run:



```
+---------------------------------+
|     Threaded Network Scanner     |
+---------------------------------+

Enter your network address (ex. 192.168.1.0/24): 192.168.9.0/24
Ping Scan: 192.168.9.0/24
192.168.9.10  -> RTT:   0.33ms
192.168.9.1   -> RTT:   0.69ms
192.168.9.102 -> RTT:  28.67ms
192.168.9.103 -> RTT:   9.40ms
192.168.9.111 -> RTT:   2.44ms
192.168.9.112 -> RTT:   3.62ms
192.168.9.130 -> RTT:   0.53ms
192.168.9.122 -> RTT:   3.51ms
192.168.9.138 -> RTT:   0.77ms
192.168.9.115 -> RTT:   5.15ms
192.168.9.137 -> RTT:   3.21ms
192.168.9.136 -> RTT:  36.66ms
192.168.9.245 -> RTT:   1.38ms
13 hosts found.
Time taken: (20.17)sec
Another scan (Y/N):n
```

Jazzed up version with rich library.



```
  Python Threaded Network Ping Scanner
         By William Loring
Enter Network (192.168.1.0/24):

Ping Scan: 192.168.9.0/24
192.168.9.1   -> RTT:   0.35ms
192.168.9.10  -> RTT:   0.40ms
192.168.9.101 -> RTT: 248.33ms
192.168.9.103 -> RTT:  19.85ms
192.168.9.112 -> RTT:   0.21ms
192.168.9.113 -> RTT:   2.07ms
192.168.9.134 -> RTT:   1.03ms
192.168.9.129 -> RTT:   2.68ms
192.168.9.137 -> RTT:   2.58ms
192.168.9.139 -> RTT:   2.14ms
192.168.9.130 -> RTT:  19.98ms
192.168.9.136 -> RTT:  21.06ms
192.168.9.245 -> RTT:   1.35ms
13 live hosts
Run Time: 40.4 seconds
Another scan (Y/N):
```

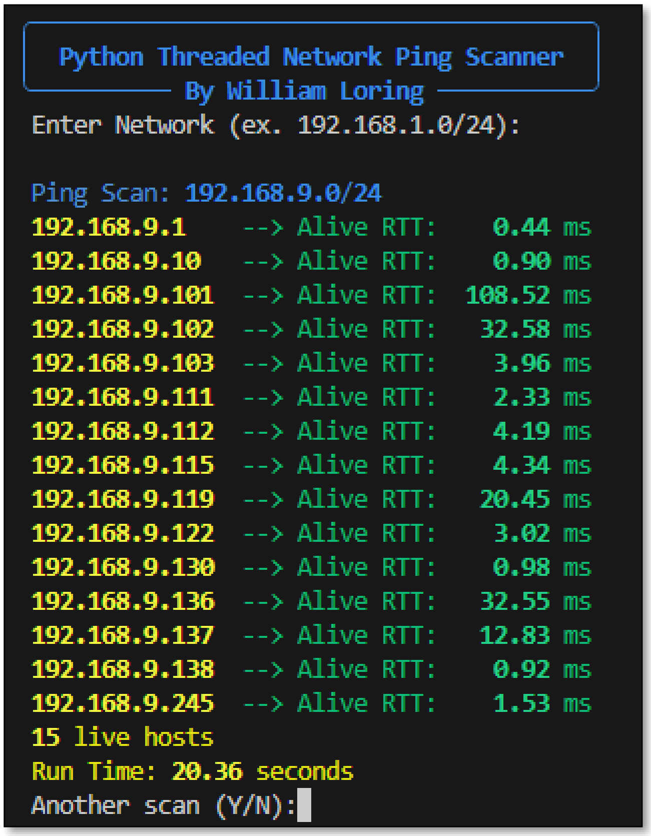
Challenge

1. Put the results of the scan into a list.

2. Sort the list by IP address.

3. Display the sorted list results.

Example run:



```
 Python Threaded Network Ping Scanner
         By William Loring
Enter Network (ex. 192.168.1.0/24):

Ping Scan: 192.168.9.0/24
192.168.9.1     --> Alive RTT:   0.44 ms
192.168.9.10    --> Alive RTT:   0.90 ms
192.168.9.101   --> Alive RTT: 108.52 ms
192.168.9.102   --> Alive RTT:  32.58 ms
192.168.9.103   --> Alive RTT:   3.96 ms
192.168.9.111   --> Alive RTT:   2.33 ms
192.168.9.112   --> Alive RTT:   4.19 ms
192.168.9.115   --> Alive RTT:   4.34 ms
192.168.9.119   --> Alive RTT:  20.45 ms
192.168.9.122   --> Alive RTT:   3.02 ms
192.168.9.130   --> Alive RTT:   0.98 ms
192.168.9.136   --> Alive RTT:  32.55 ms
192.168.9.137   --> Alive RTT:  12.83 ms
192.168.9.138   --> Alive RTT:   0.92 ms
192.168.9.245   --> Alive RTT:   1.53 ms
15 live hosts
Run Time: 20.36 seconds
Another scan (Y/N):
```

Assignment Submission

1. Attach all program files.

2. Attach a screenshot of each successful program run.

3. If you do not attach a screenshot of a successful program run on your correct network address, you will receive a 0 for this assignment.

4. Submit the assignment in Blackboard.