Network And Port Scanner | Minor Project

Name: B. Keerthi Prasanna

Email: keerthi2004b@gmail.com

Domain: Cyber Security

A port scanner is a tool used to scan a computer or network for open ports. It allows you to check which ports on a target system are open and listen for incoming connections. It involves sending network requests to specific ports and analyzing the responses to identify the port's status. Port scanning is commonly performed for various purposes, including network security assessments, vulnerability testing, and network troubleshooting.

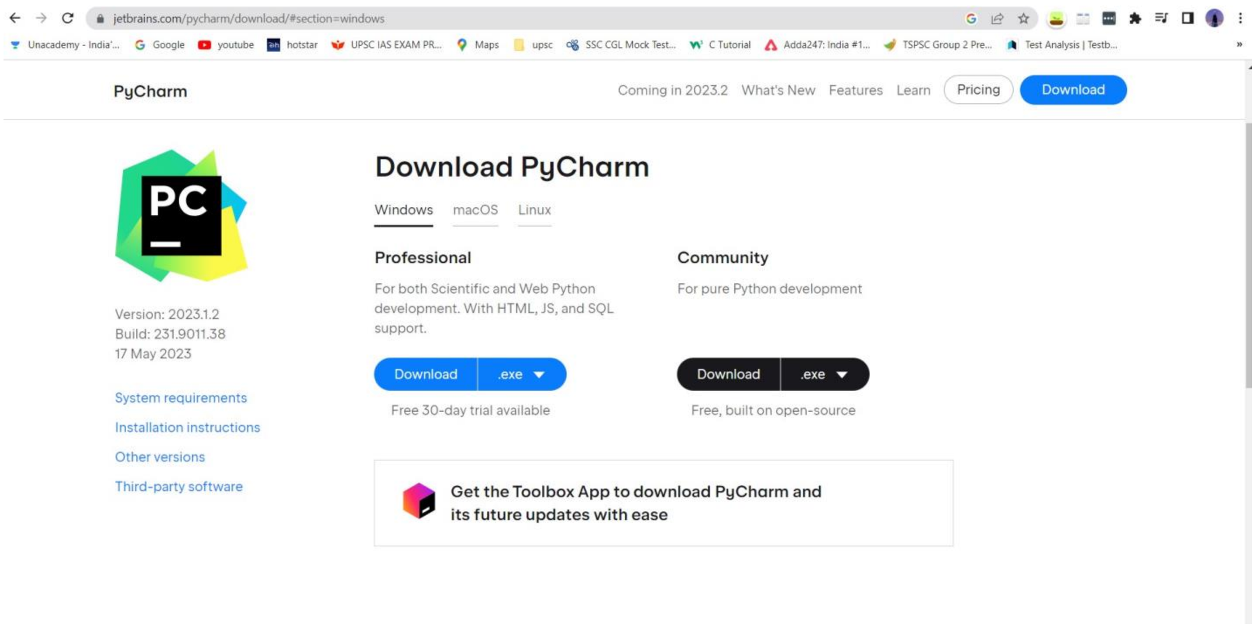
There are various port scanner tools available, both command-line and graphical, that automates the process of port scanning. Some popular tools include Nmap, Masscan, Zmap, and Angry IP Scanner.

In this project we are writing a code for a basic **port scanner that allows you to enter an IP address and a range of ports to scan.** It will check if each **port in the specified range is open or closed** on the given IP address.

We can write and run the port scanning code in various platforms like PyCharm, Linux, and Visual Studio Code (VSC). This code can be executed in any Python development environment or text editor that supports Python.

In this project, we are using pycharm as it provides advanced code completion, intelligent code analysis, quick-fix suggestions, and powerful refactoring tools. These features are tailored to Python development and help to write cleaner and more efficient code.

- So, firstly we need to download pycharm in your preferred operating system, such as Windows, macOS, or Linux.
- You can visit the official JetBrains website or the PyCharm website to download the appropriate version of PyCharm for your operating system.
- Once downloaded, you can follow the installation instructions to set up PyCharm on your computer.



Download PyCharm

Version: 2023.1.2
Build: 231.9011.38
17 May 2023

Professional

For both Scientific and Web Python development. With HTML, JS, and SQL support.

Free 30-day trial available

Community

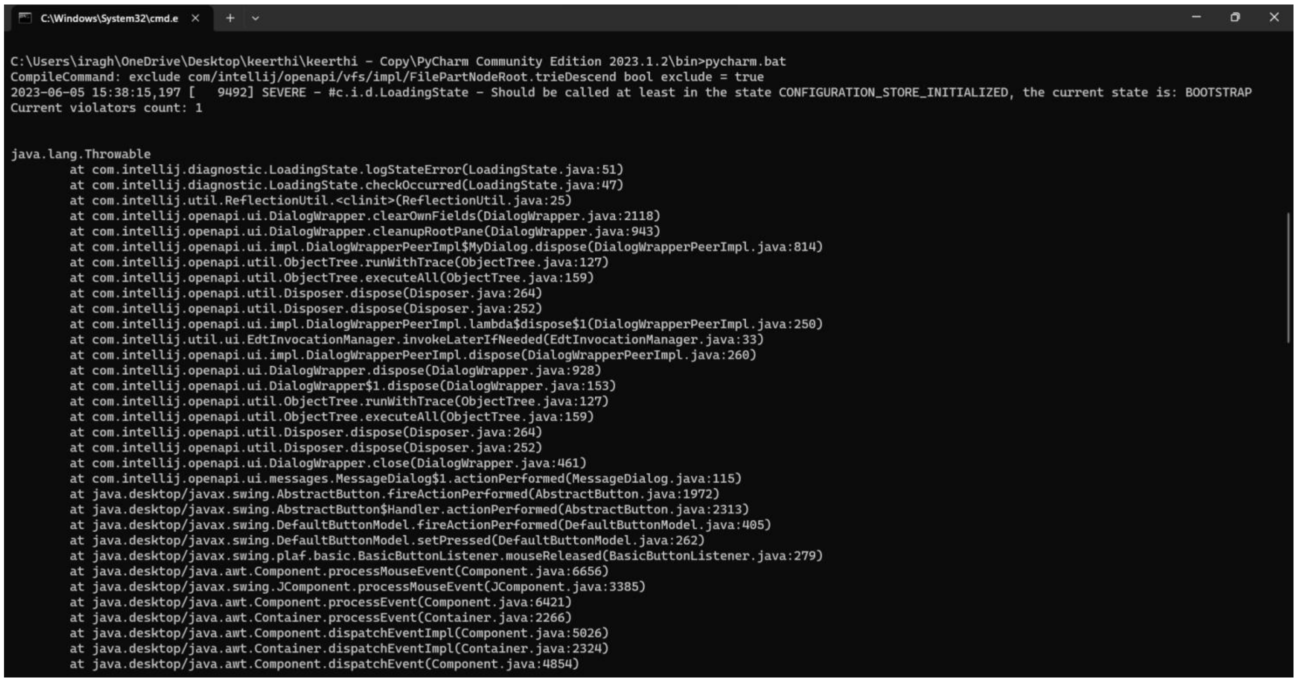
For pure Python development

Free, built on open-source

Get the Toolbox App to download PyCharm and its future updates with ease

Download the community edition. Follow the on-screen instructions to install PyCharm. You can choose the installation location, select additional features, and customize the installation as per your preferences . After the installation is complete, you can launch PyCharm from the Start menu (Windows) or the Applications folder (macOS).

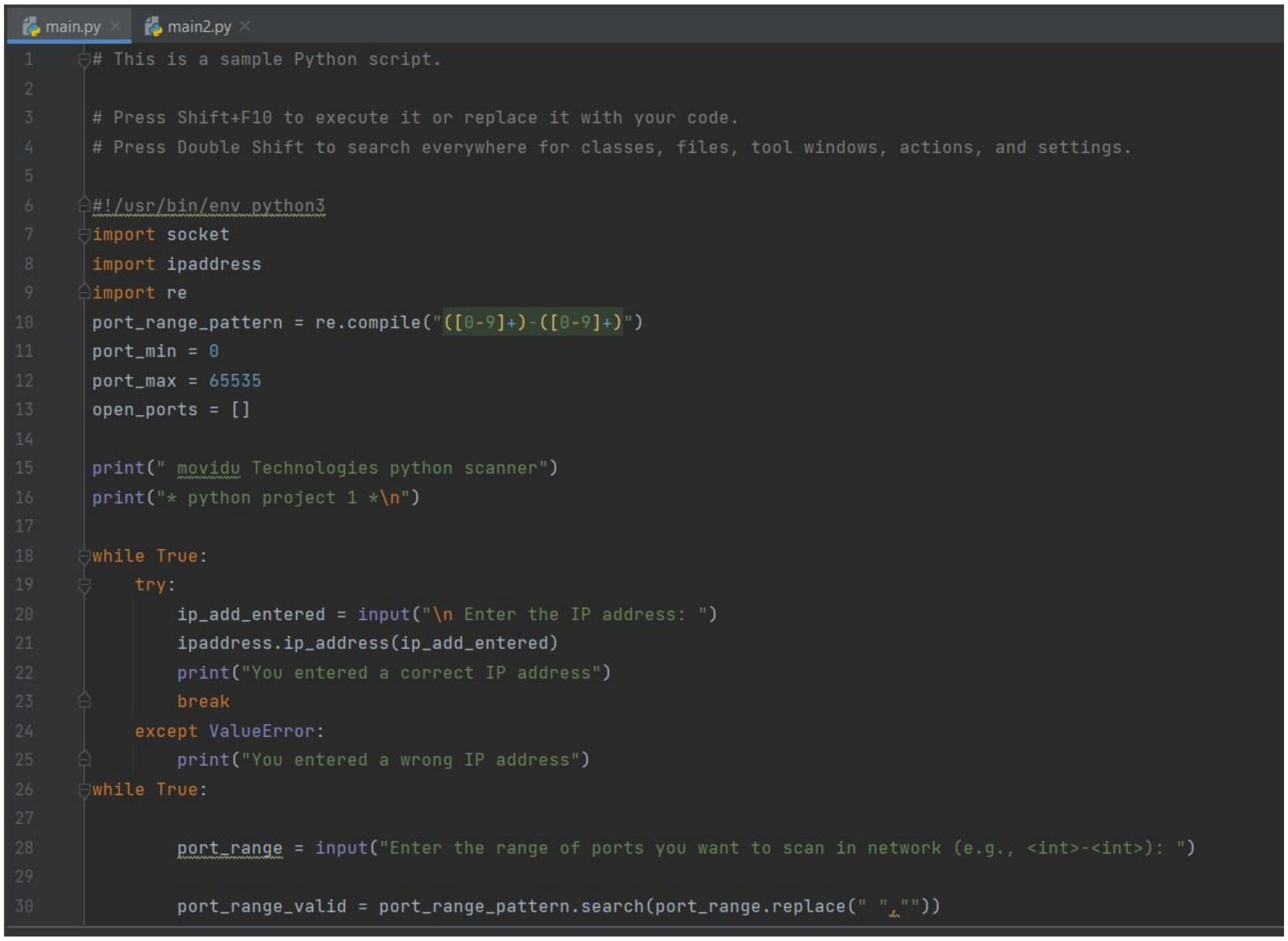
The **pycharm.bat** file is a Windows batch script provided by PyCharm IDE. It is used to launch PyCharm from the command line.



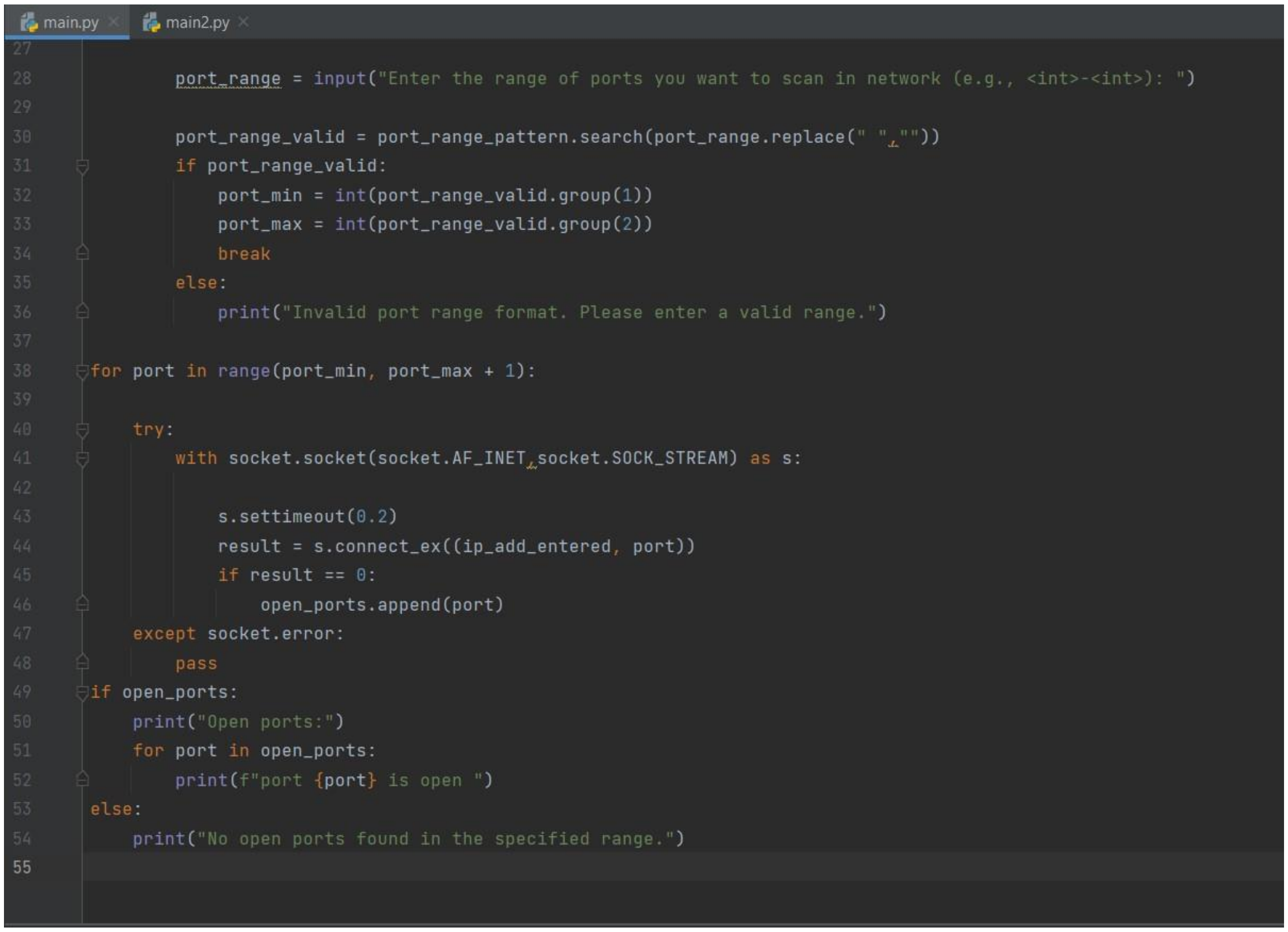
```
C:\Users\iragh\OneDrive\Desktop\keerthi\keerthi - Copy\PyCharm Community Edition 2023.1.2\bin>pycharm.bat
CompileCommand: exclude com/intellij/openapi/vfs/impl/FilePartNodeRoot.trieDescend bool exclude = true
2023-06-05 15:38:15,197 [   9492] SEVERE - #c.i.d.LoadingState - Should be called at least in the state CONFIGURATION_STORE_INITIALIZED, the current state is: BOOTSTRAP
Current violators count: 1


java.lang.Throwable
        at com.intellij.diagnostic.LoadingState.logStateError(LoadingState.java:51)
        at com.intellij.diagnostic.LoadingState.checkOccurred(LoadingState.java:47)
        at com.intellij.util.ReflectionUtil.<clinit>(ReflectionUtil.java:25)
        at com.intellij.openapi.ui.DialogWrapper.clearOwnFields(DialogWrapper.java:2118)
        at com.intellij.openapi.ui.DialogWrapper.cleanupRootPane(DialogWrapper.java:943)
        at com.intellij.openapi.ui.impl.DialogWrapperPeerImpl$MyDialog.dispose(DialogWrapperPeerImpl.java:814)
        at com.intellij.openapi.util.ObjectTree.runWithTrace(ObjectTree.java:127)
        at com.intellij.openapi.util.ObjectTree.executeAll(ObjectTree.java:159)
        at com.intellij.openapi.util.Disposer.dispose(Disposer.java:264)
        at com.intellij.openapi.util.Disposer.dispose(Disposer.java:252)
        at com.intellij.openapi.ui.impl.DialogWrapperPeerImpl.lambda$dispose$1(DialogWrapperPeerImpl.java:250)
        at com.intellij.util.ui.EdtInvocationManager.invokeLaterIfNeeded(EdtInvocationManager.java:33)
        at com.intellij.openapi.ui.impl.DialogWrapperPeerImpl.dispose(DialogWrapperPeerImpl.java:260)
        at com.intellij.openapi.ui.DialogWrapper.dispose(DialogWrapper.java:928)
        at com.intellij.openapi.ui.DialogWrapper$1.dispose(DialogWrapper.java:153)
        at com.intellij.openapi.util.ObjectTree.runWithTrace(ObjectTree.java:127)
        at com.intellij.openapi.util.ObjectTree.executeAll(ObjectTree.java:159)
        at com.intellij.openapi.util.Disposer.dispose(Disposer.java:264)
        at com.intellij.openapi.util.Disposer.dispose(Disposer.java:252)
        at com.intellij.openapi.ui.DialogWrapper.close(DialogWrapper.java:461)
        at com.intellij.openapi.ui.messages.MessageDialog$1.actionPerformed(MessageDialog.java:115)
        at java.desktop/javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:1972)
        at java.desktop/javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:2313)
        at java.desktop/javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:405)
        at java.desktop/javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:262)
        at java.desktop/javax.swing.plaf.basic.BasicButtonListener.mouseReleased(BasicButtonListener.java:279)
        at java.desktop/java.awt.Component.processMouseEvent(Component.java:6656)
        at java.desktop/javax.swing.JComponent.processMouseEvent(JComponent.java:3385)
        at java.desktop/java.awt.Component.processEvent(Component.java:6421)
        at java.desktop/java.awt.Container.processEvent(Container.java:2266)
        at java.desktop/java.awt.Component.dispatchEventImpl(Component.java:5026)
        at java.desktop/java.awt.Container.dispatchEventImpl(Container.java:2324)
        at java.desktop/java.awt.Component.dispatchEvent(Component.java:4854)
```

When you run the pycharm.bat file, it starts the PyCharm IDE and opens the graphical interface for you to work with. In this interface we write and run our port scanner program.

<u>CODE FOR PORT SCANNER:</u>

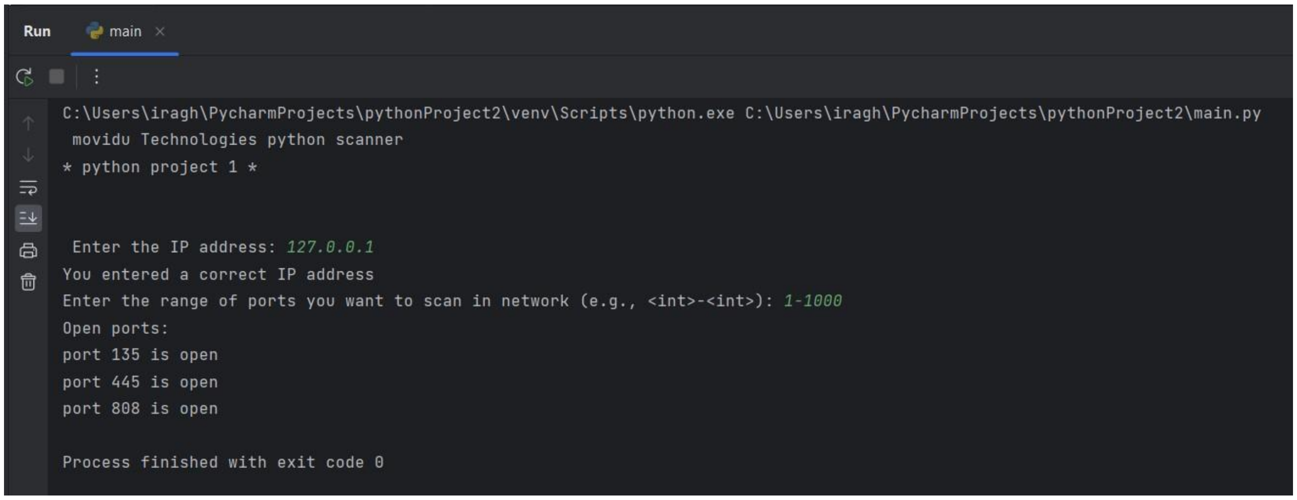


```python
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.

#!/usr/bin/env python3
import socket
import ipaddress
import re
port_range_pattern = re.compile("([0-9]+)-([0-9]+)")
port_min = 0
port_max = 65535
open_ports = []

print(" movidu Technologies python scanner")
print("* python project 1 *\n")

while True:
    try:
        ip_add_entered = input("\n Enter the IP address: ")
        ipaddress.ip_address(ip_add_entered)
        print("You entered a correct IP address")
        break
    except ValueError:
        print("You entered a wrong IP address")
while True:

        port_range = input("Enter the range of ports you want to scan in network (e.g., <int>-<int>): ")

        port_range_valid = port_range_pattern.search(port_range.replace(" ",""))
```

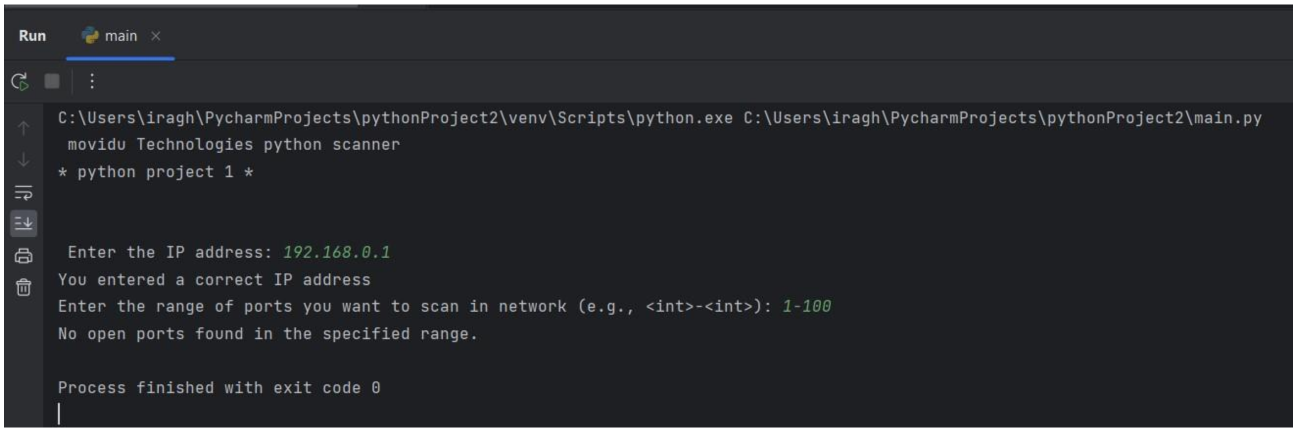


```python
        port_range = input("Enter the range of ports you want to scan in network (e.g., <int>-<int>): ")

        port_range_valid = port_range_pattern.search(port_range.replace(" ",""))
        if port_range_valid:
            port_min = int(port_range_valid.group(1))
            port_max = int(port_range_valid.group(2))
            break
        else:
            print("Invalid port range format. Please enter a valid range.")

for port in range(port_min, port_max + 1):

    try:
        with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as s:

            s.settimeout(0.2)
            result = s.connect_ex((ip_add_entered, port))
            if result == 0:
                open_ports.append(port)
    except socket.error:
        pass
if open_ports:
    print("Open ports:")
    for port in open_ports:
        print(f"port {port} is open ")
else:
    print("No open ports found in the specified range.")
```

OUTPUT:



```
C:\Users\iragh\PycharmProjects\pythonProject2\venv\Scripts\python.exe C:\Users\iragh\PycharmProjects\pythonProject2\main.py
 movidu Technologies python scanner
* python project 1 *


 Enter the IP address: 127.0.0.1
You entered a correct IP address
Enter the range of ports you want to scan in network (e.g., <int>-<int>): 1-1000
Open ports:
port 135 is open
port 445 is open
port 808 is open

Process finished with exit code 0
```



```
C:\Users\iragh\PycharmProjects\pythonProject2\venv\Scripts\python.exe C:\Users\iragh\PycharmProjects\pythonProject2\main.py
 movidu Technologies python scanner
* python project 1 *


 Enter the IP address: 192.168.0.1
You entered a correct IP address
Enter the range of ports you want to scan in network (e.g., <int>-<int>): 1-100
No open ports found in the specified range.

Process finished with exit code 0
```

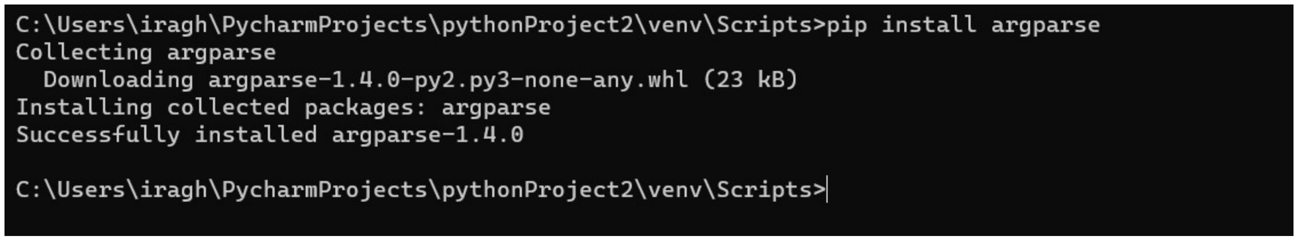
Explanation of the code :

1. The script begins by importing the required modules and setting up a few variables.
2. You are prompted to enter your IP address. To verify the IP address entered, it makes use of the IP address module.
3. After that, you are prompted to enter a range of ports to scan in the format <int>-<int> (for example, 80-100). It verifies the format of the port range input.
4. An empty list is initialized to hold the open ports.
5. Each port in the provided range is iterated over, and a TCP socket is used to try to connect to that port on the IP address.
6. If the connection is successful (result code 0), the port number is added to the list of open ports.
7. Following a thorough port scan, It looks to see if any open ports were discovered and publishes a list of them. If no open ports were discovered, a message stating as much is printed.

A **TCP port scanner** is a tool used to check for open TCP ports on a target machine. It aids in determining which ports are listening and can offer details on the services

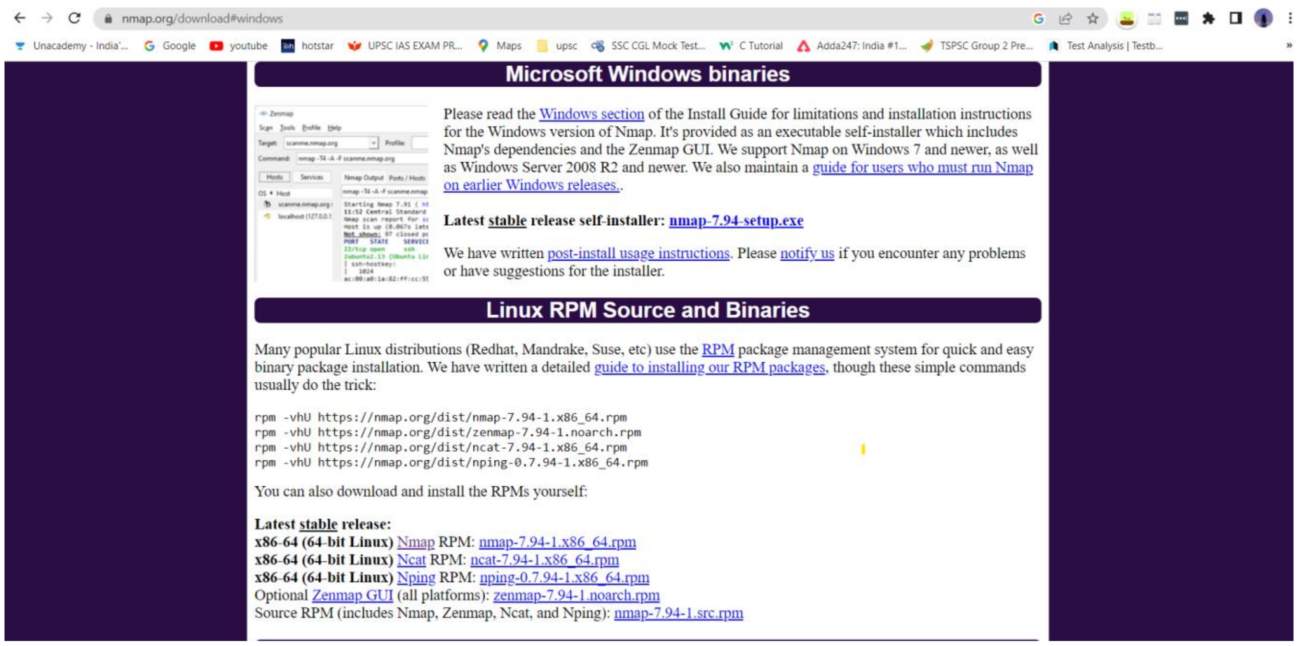
that are available on those ports. The Python argparse and nmap libraries are used to implement a TCP port scanner in the script you gave.
This code can also be written and executed using the PyCharm platform.
Installing the argparse and nmap libraries is the first step.
Simply type the following command on the command prompt to install argparse.

pip install argparse



```
C:\Users\iragh\PycharmProjects\pythonProject2\venv\Scripts>pip install argparse
Collecting argparse
  Downloading argparse-1.4.0-py2.py3-none-any.whl (23 kB)
Installing collected packages: argparse
Successfully installed argparse-1.4.0

C:\Users\iragh\PycharmProjects\pythonProject2\venv\Scripts>
```

Nmap can be downloaded via a browser based on the operating system and version we like.



Microsoft Windows binaries

Please read the Windows section of the Install Guide for limitations and installation instructions for the Windows version of Nmap. It's provided as an executable self-installer which includes Nmap's dependencies and the Zenmap GUI. We support Nmap on Windows 7 and newer, as well as Windows Server 2008 R2 and newer. We also maintain a guide for users who must run Nmap on earlier Windows releases..

Latest stable release self-installer: nmap-7.94-setup.exe

We have written post-install usage instructions. Please notify us if you encounter any problems or have suggestions for the installer.

Linux RPM Source and Binaries

Many popular Linux distributions (Redhat, Mandrake, Suse, etc) use the RPM package management system for quick and easy binary package installation. We have written a detailed guide to installing our RPM packages, though these simple commands usually do the trick:

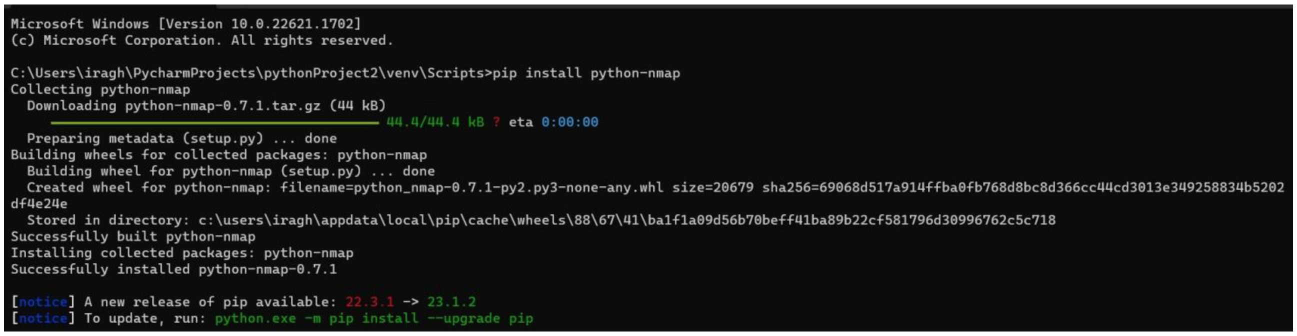
```
rpm -vhU https://nmap.org/dist/nmap-7.94-1.x86_64.rpm
rpm -vhU https://nmap.org/dist/zenmap-7.94-1.noarch.rpm
rpm -vhU https://nmap.org/dist/ncat-7.94-1.x86_64.rpm
rpm -vhU https://nmap.org/dist/nping-0.7.94-1.x86_64.rpm
```

You can also download and install the RPMs yourself:

Latest stable release:
x86-64 (64-bit Linux) Nmap RPM: nmap-7.94-1.x86_64.rpm
x86-64 (64-bit Linux) Ncat RPM: ncat-7.94-1.x86_64.rpm
x86-64 (64-bit Linux) Nping RPM: nping-0.7.94-1.x86_64.rpm
Optional Zenmap GUI (all platforms): zenmap-7.94-1.noarch.rpm
Source RPM (includes Nmap, Zenmap, Ncat, and Nping): nmap-7.94-1.src.rpm

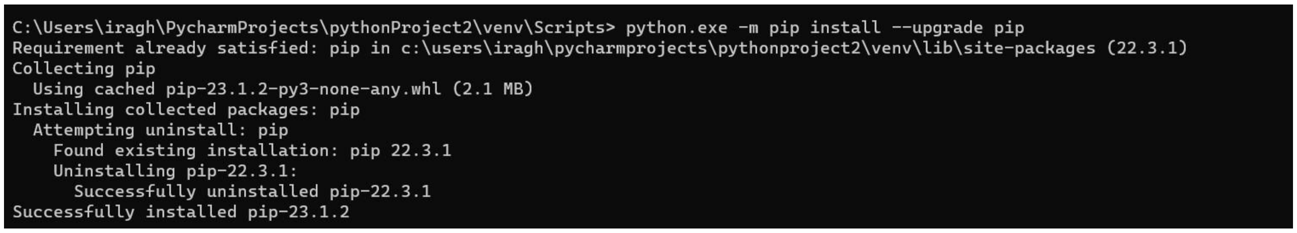
After this step we have to install the python-nmap package using pip, by using the following command:

pip install python-nmap



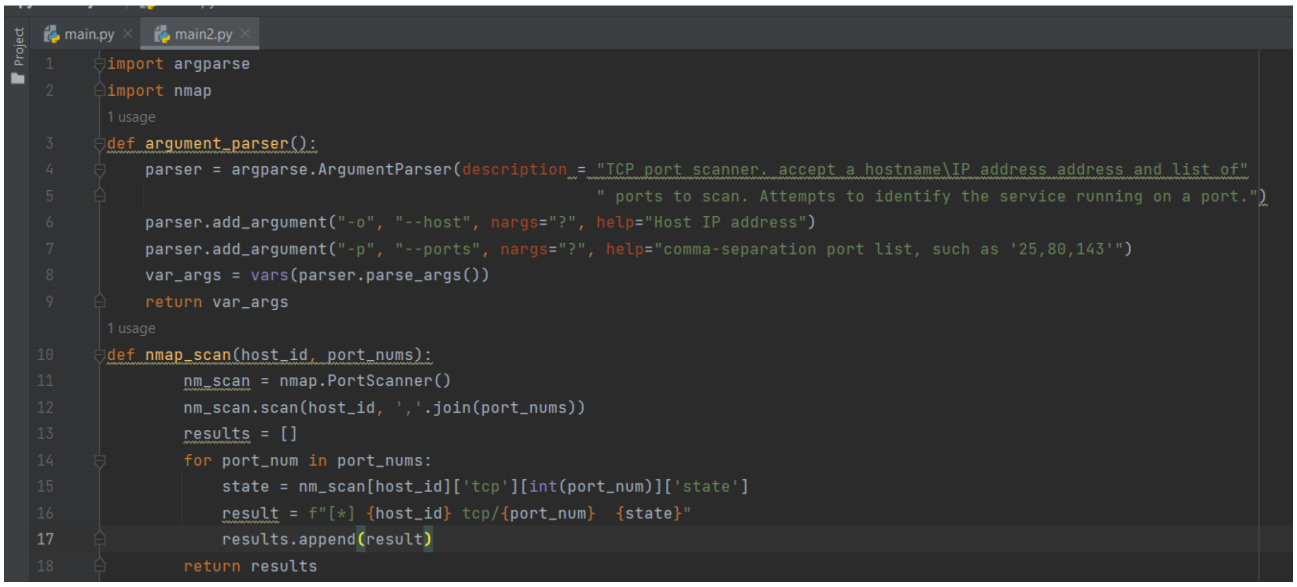
```
Microsoft Windows [Version 10.0.22621.1702]
(c) Microsoft Corporation. All rights reserved.

C:\Users\iragh\PycharmProjects\pythonProject2\venv\Scripts>pip install python-nmap
Collecting python-nmap
  Downloading python-nmap-0.7.1.tar.gz (44 kB)
     ---------------------------------------- 44.4/44.4 kB ? eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: python-nmap
  Building wheel for python-nmap (setup.py) ... done
  Created wheel for python-nmap: filename=python_nmap-0.7.1-py2.py3-none-any.whl size=20679 sha256=69068d517a914ffba0fb768d8bc8d366cc44cd3013e349258834b5202
df4e24e
  Stored in directory: c:\users\iragh\appdata\local\pip\cache\wheels\88\67\41\ba1f1a09d56b70beff41ba89b22cf581796d30996762c5c718
Successfully built python-nmap
Installing collected packages: python-nmap
Successfully installed python-nmap-0.7.1

[notice] A new release of pip available: 22.3.1 -> 23.1.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

if necessary we can upgrade pip

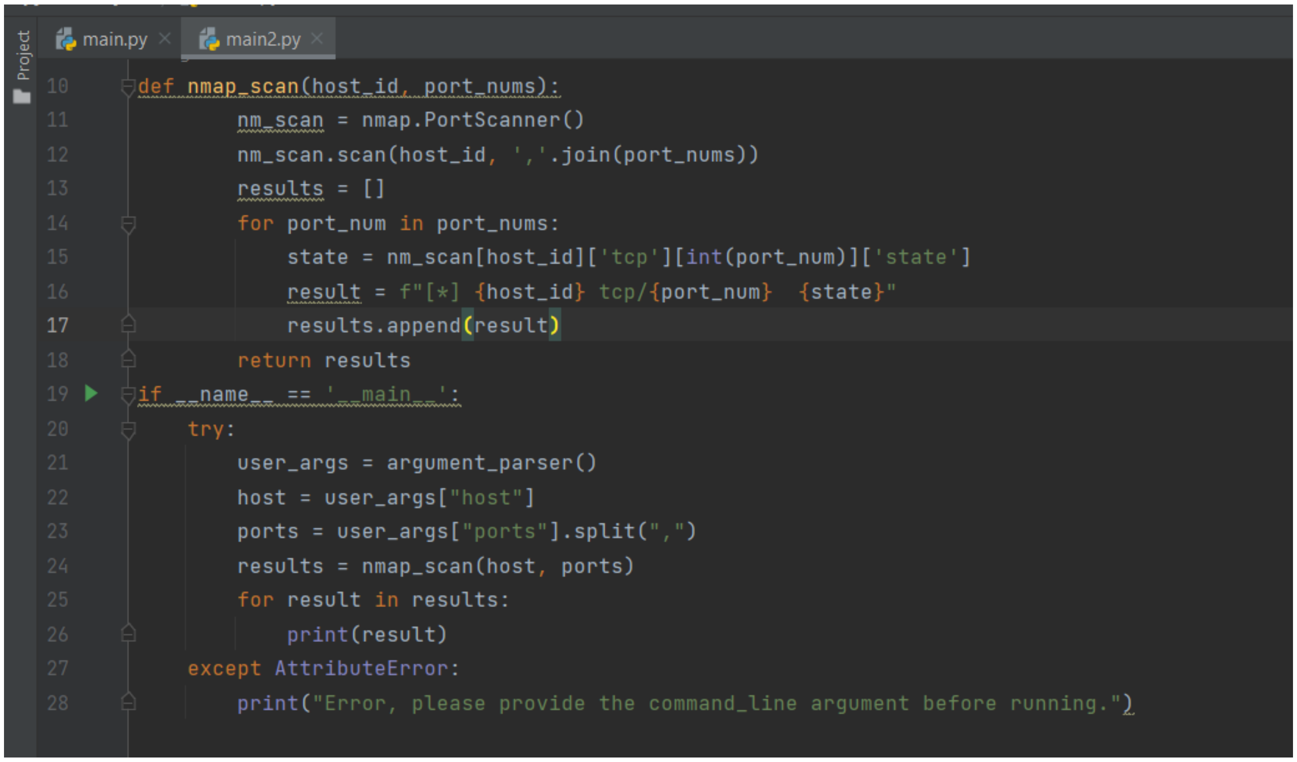


```
C:\Users\iragh\PycharmProjects\pythonProject2\venv\Scripts> python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\iragh\pycharmprojects\pythonproject2\venv\lib\site-packages (22.3.1)
Collecting pip
  Using cached pip-23.1.2-py3-none-any.whl (2.1 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.3.1
    Uninstalling pip-22.3.1:
      Successfully uninstalled pip-22.3.1
Successfully installed pip-23.1.2
```

<u>CODE FOR TCP PORT SCANNER:</u>

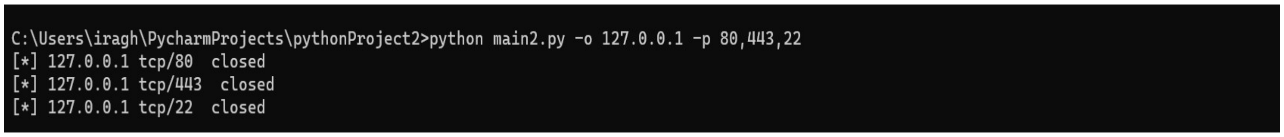


```python
import argparse
import nmap
1 usage
def argument_parser():
    parser = argparse.ArgumentParser(description = "TCP port scanner. accept a hostname\IP address address and list of"
                                                   " ports to scan. Attempts to identify the service running on a port.")
    parser.add_argument("-o", "--host", nargs="?", help="Host IP address")
    parser.add_argument("-p", "--ports", nargs="?", help="comma-separation port list, such as '25,80,143'")
    var_args = vars(parser.parse_args())
    return var_args
1 usage
def nmap_scan(host_id, port_nums):
        nm_scan = nmap.PortScanner()
        nm_scan.scan(host_id, ','.join(port_nums))
        results = []
        for port_num in port_nums:
            state = nm_scan[host_id]['tcp'][int(port_num)]['state']
            result = f"[*] {host_id} tcp/{port_num}  {state}"
            results.append(result)
        return results
```

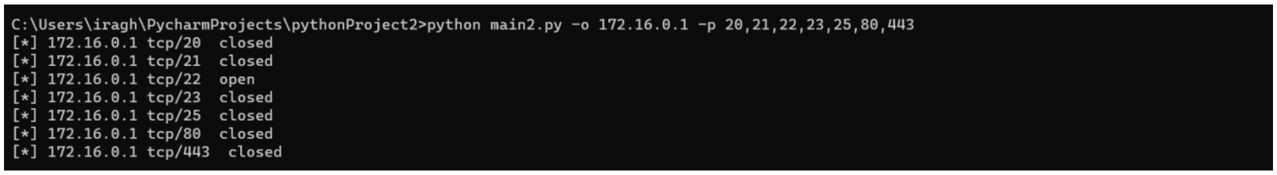


```python
def nmap_scan(host_id, port_nums):
        nm_scan = nmap.PortScanner()
        nm_scan.scan(host_id, ','.join(port_nums))
        results = []
        for port_num in port_nums:
            state = nm_scan[host_id]['tcp'][int(port_num)]['state']
            result = f"[*] {host_id} tcp/{port_num}  {state}"
            results.append(result)
        return results
if __name__ == '__main__':
    try:
        user_args = argument_parser()
        host = user_args["host"]
        ports = user_args["ports"].split(",")
        results = nmap_scan(host, ports)
        for result in results:
            print(result)
    except AttributeError:
        print("Error, please provide the command_line argument before running.")
```

OUTPUT:



```
C:\Users\iragh\PycharmProjects\pythonProject2>python main2.py -o 127.0.0.1 -p 80,443,22
[*] 127.0.0.1 tcp/80  closed
[*] 127.0.0.1 tcp/443  closed
[*] 127.0.0.1 tcp/22  closed
```



```
C:\Users\iragh\PycharmProjects\pythonProject2>python main2.py -o 172.16.0.1 -p 20,21,22,23,25,80,443
[*] 172.16.0.1 tcp/20  closed
[*] 172.16.0.1 tcp/21  closed
[*] 172.16.0.1 tcp/22  open
[*] 172.16.0.1 tcp/23  closed
[*] 172.16.0.1 tcp/25  closed
[*] 172.16.0.1 tcp/80  closed
[*] 172.16.0.1 tcp/443  closed
```

Explanation of the code :

1. To handle command-line parameter parsing, the argparse library is imported.
2. The Nmap tool is used to scan ports after importing the nmap library.
3. To parse the command-line arguments, use the argument_parser() function, which is defined. The --host parameter specifies the IP address of the host, and the --ports argument specifies a comma-separated list of ports to scan. This creates an argument parser object.

4. To carry out the port scanning using Nmap, the nmap_scan() function is defined. It requires as inputs the host IP address and port numbers. It establishes a new nmap instance. The scan() function of the PortScanner() class scans the supplied host and ports. From the results of the scan, it extracts the condition of each port and creates a result string for each port.
5. The script determines whether it is being run immediately in the main section (if __name__ == '__main__':). The command-line arguments are parsed using the argument_parser() function, and the host and port values are assigned to variables.
6. The port scanning is then carried out by calling the nmap_scan() function with the host and port arguments.
7. A loop is used to print each result one at a time.

We can run this script from the command line with the proper command-line options. For instance, we may run the script as follows:

Python script_name.py 192.168.0.1 host, 80, 443, and 22 ports

In Conclusion, the port scanner and TCP port scanner, offer useful tools for network scanning and research.

The **Python socket module** is used by the port scanner to check a host or IP address for open ports. A range of ports can be specified, and the script will attempt to connect over TCP to each one in order to determine whether it is open or closed. Network administrators and security experts may find this information helpful in evaluating the security of a network and identifying any potential weaknesses.

The functionality of the port scanner is improved by the TCP port scanner, which was developed utilizing the **argparse and nmap libraries**. Users can use command-line options to define the target host and a list of ports to scan. The Nmap tool, which enables sophisticated scanning algorithms and service discovery, is used by the nmap library to give a more thorough and effective scanning mechanism.

The effectiveness and adaptability of Python for network scanning tasks are demonstrated by both tools. They can be enhanced and altered to satisfy certain needs. It's crucial to remember that port scanning should only be carried out responsibly and with the appropriate authorization, as it may break security and privacy rules.

I gained knowledge about network protocols, socket programming, and library integration in Python by studying and putting these project into practice. Network administrators, cybersecurity experts, and anybody else interested in learning about and defending computer networks can all benefit from having these skills.