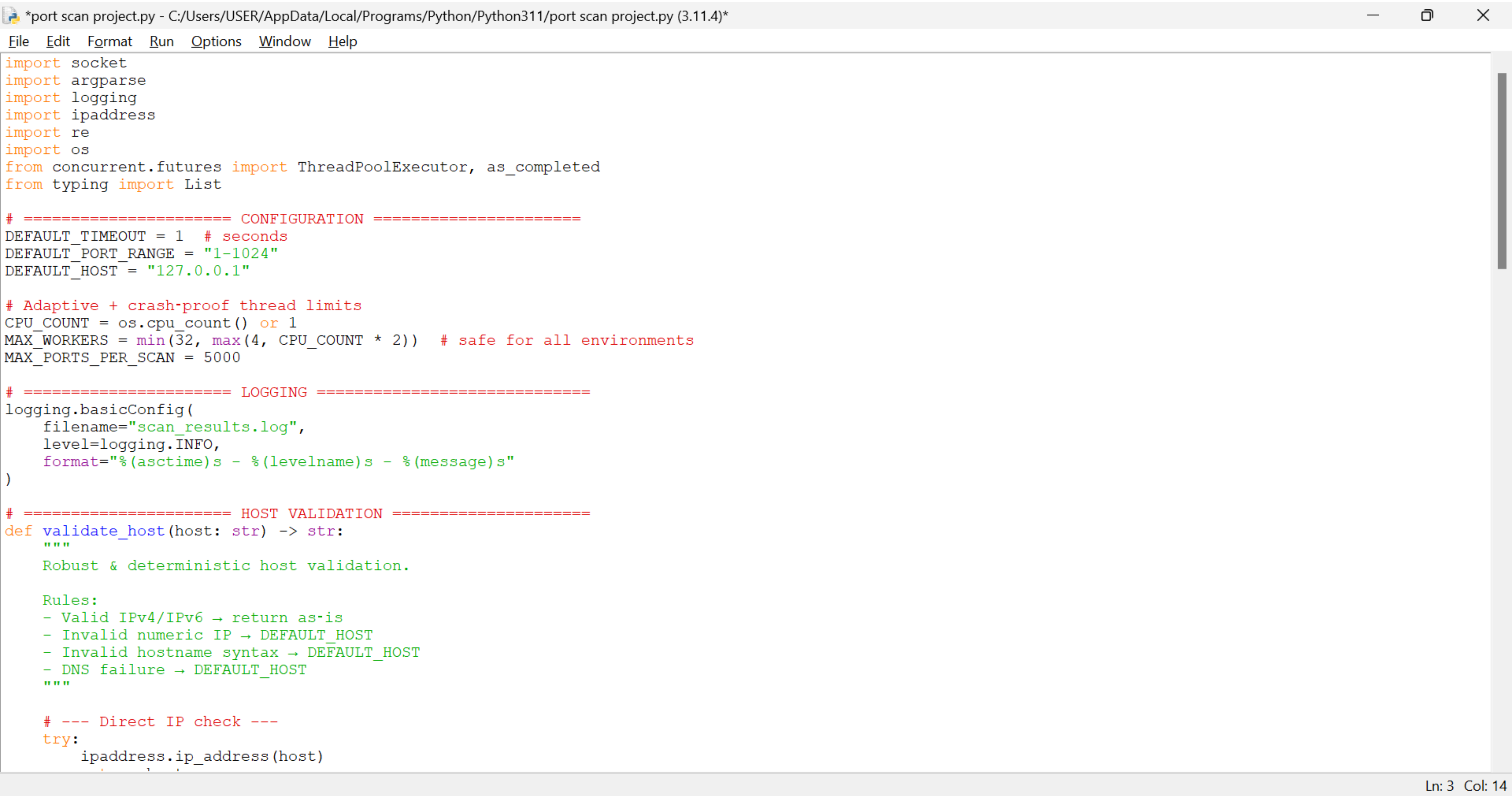
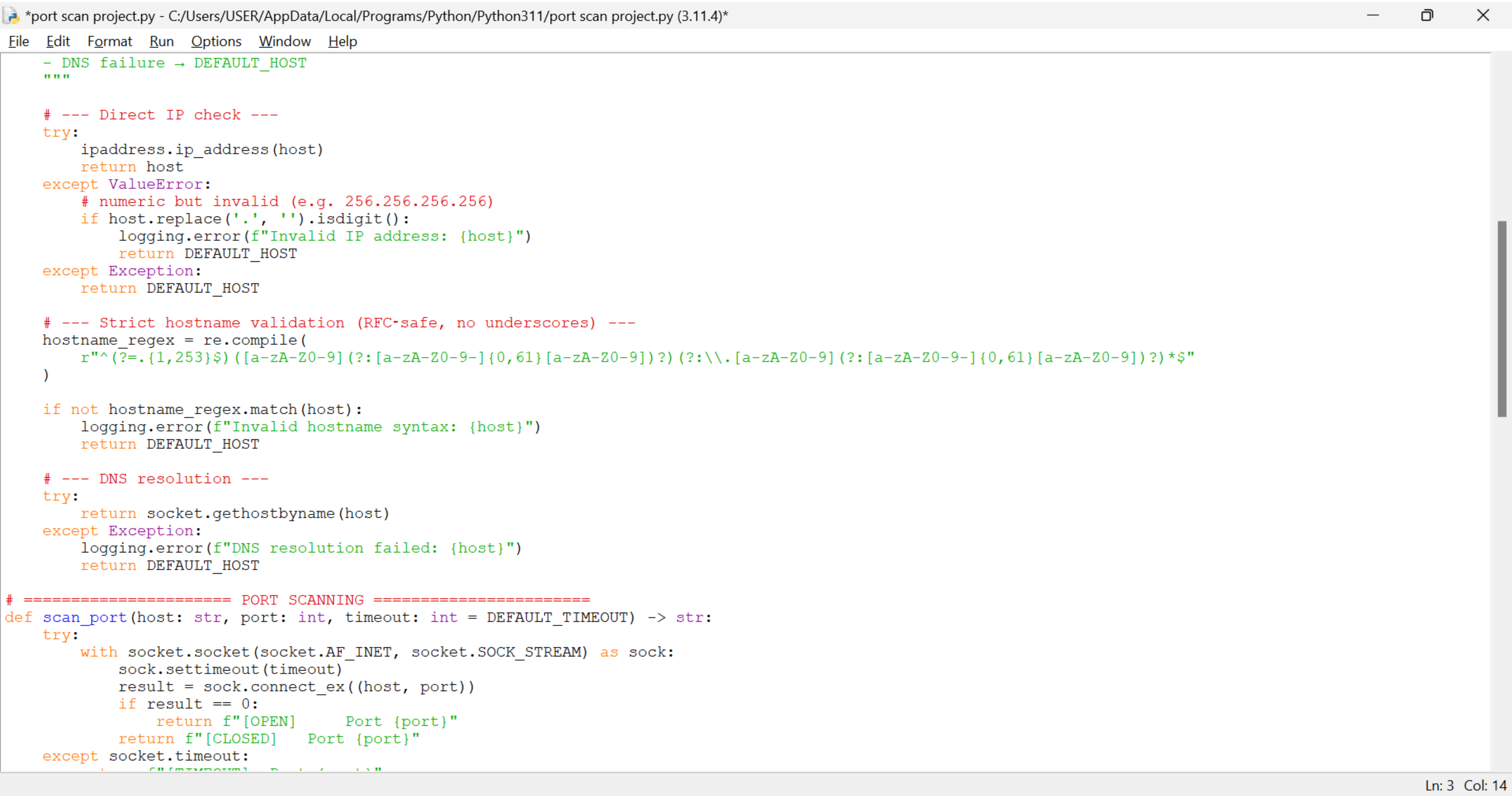
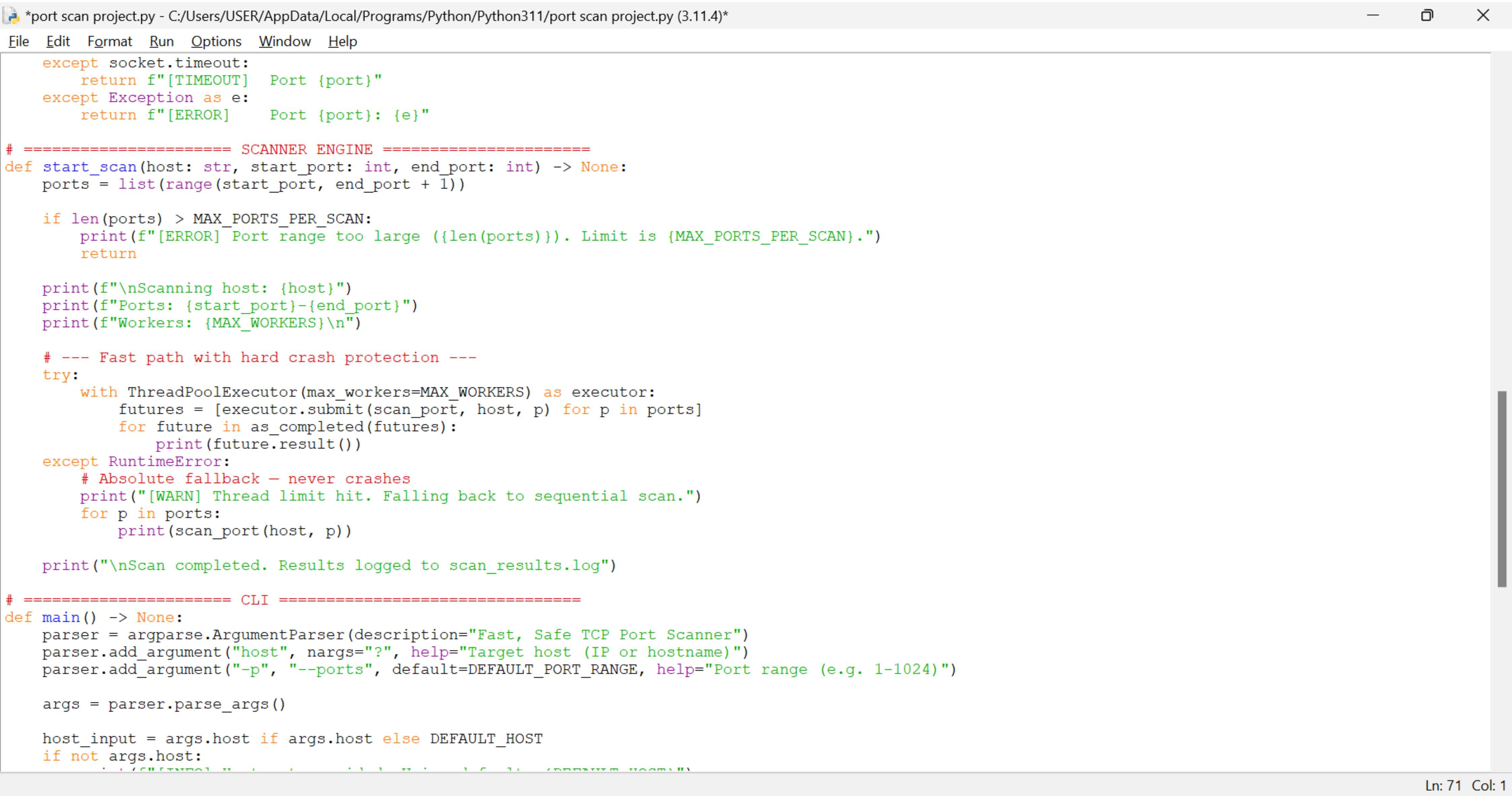

port scan project.py - C:/Users/USER/AppData/Local/Programs/Python/Python311/port scan project.py (3.11.4)
File Edit Format Run Options Window Help
Ln: 3 Col: 14

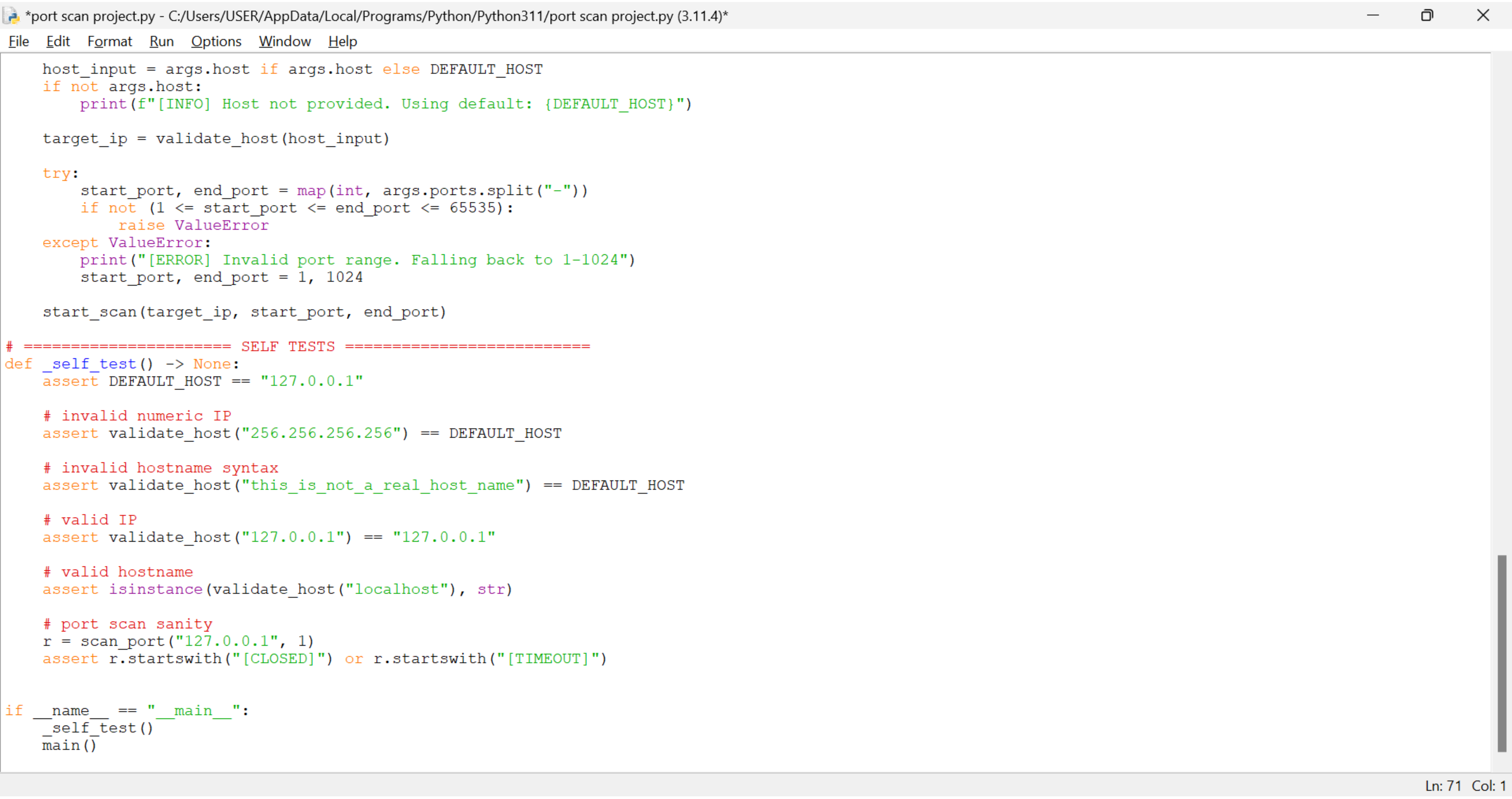
```python
import socket
import argparse
import logging
import ipaddress
import re
import os
from concurrent.futures import ThreadPoolExecutor, as_completed
from typing import List

# ====================== CONFIGURATION ======================
DEFAULT_TIMEOUT = 1  # seconds
DEFAULT_PORT_RANGE = "1-1024"
DEFAULT_HOST = "127.0.0.1"

# Adaptive + crash-proof thread limits
CPU_COUNT = os.cpu_count() or 1
MAX_WORKERS = min(32, max(4, CPU_COUNT * 2))  # safe for all environments
MAX_PORTS_PER_SCAN = 5000

# ====================== LOGGING ============================
logging.basicConfig(
    filename="scan_results.log",
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s"
)

# ====================== HOST VALIDATION ====================
def validate_host(host: str) -> str:
    """
    Robust & deterministic host validation.

    Rules:
    - Valid IPv4/IPv6 → return as-is
    - Invalid numeric IP → DEFAULT_HOST
    - Invalid hostname syntax → DEFAULT_HOST
    - DNS failure → DEFAULT_HOST
    """

    # --- Direct IP check ---
    try:
        ipaddress.ip_address(host)
```


port scan project.py - C:/Users/USER/AppData/Local/Programs/Python/Python311/port scan project.py (3.11.4)
File Edit Format Run Options Window Help

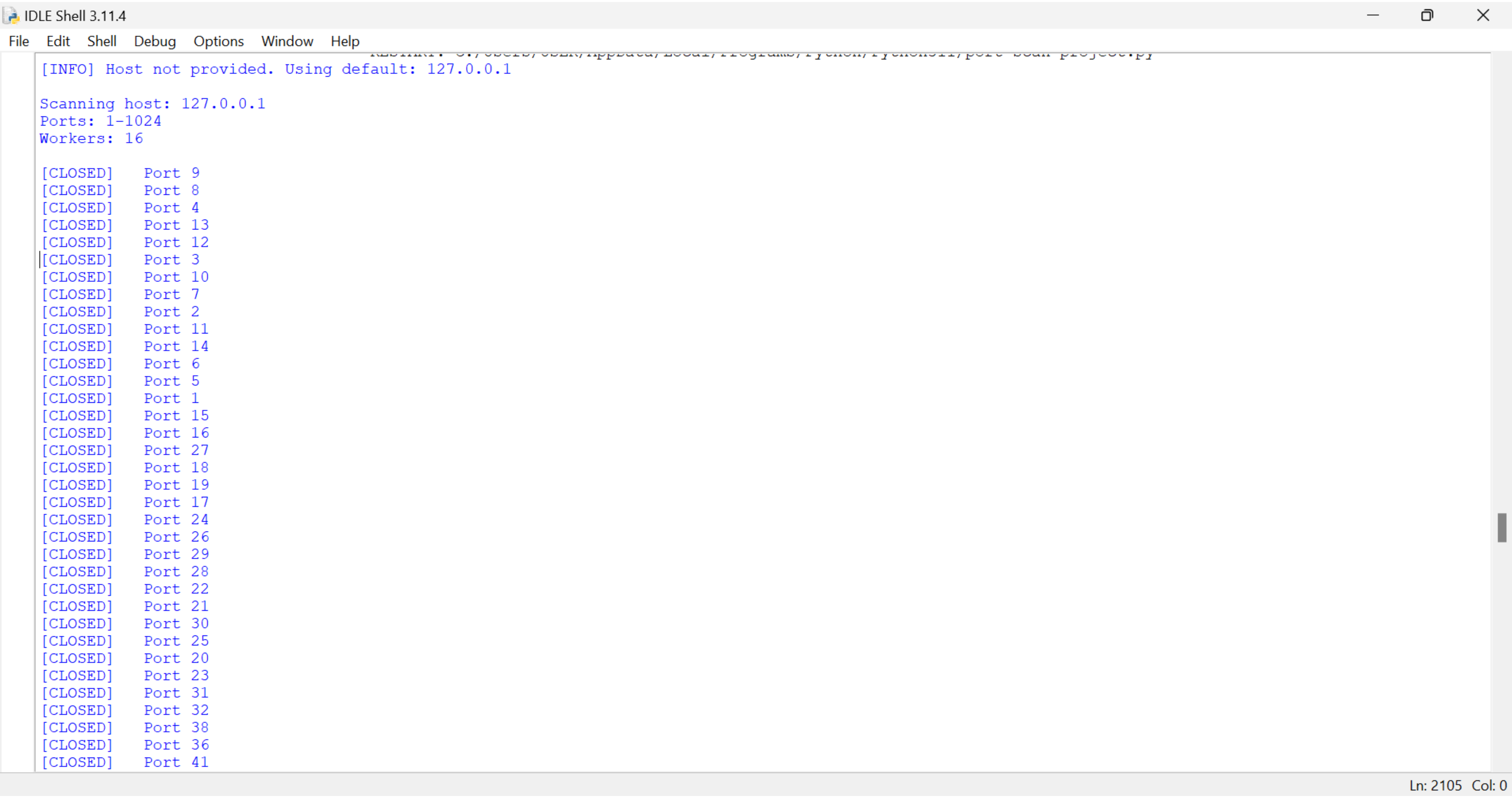
```
    - DNS failure → DEFAULT_HOST
    """

    # --- Direct IP check ---
    try:
        ipaddress.ip_address(host)
        return host
    except ValueError:
        # numeric but invalid (e.g. 256.256.256.256)
        if host.replace('.', '').isdigit():
            logging.error(f"Invalid IP address: {host}")
            return DEFAULT_HOST
    except Exception:
        return DEFAULT_HOST

    # --- Strict hostname validation (RFC-safe, no underscores) ---
    hostname_regex = re.compile(
        r"^(?=.{1,253}$)([a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)(?:\\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$"
    )

    if not hostname_regex.match(host):
        logging.error(f"Invalid hostname syntax: {host}")
        return DEFAULT_HOST

    # --- DNS resolution ---
    try:
        return socket.gethostbyname(host)
    except Exception:
        logging.error(f"DNS resolution failed: {host}")
        return DEFAULT_HOST

# ====================== PORT SCANNING =======================
def scan_port(host: str, port: int, timeout: int = DEFAULT_TIMEOUT) -> str:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
            sock.settimeout(timeout)
            result = sock.connect_ex((host, port))
            if result == 0:
                return f"[OPEN]     Port {port}"
            return f"[CLOSED]   Port {port}"
    except socket.timeout:
```


port scan project.py - C:/Users/USER/AppData/Local/Programs/Python/Python311/port scan project.py (3.11.4)
File Edit Format Run Options Window Help
Ln: 71 Col: 1

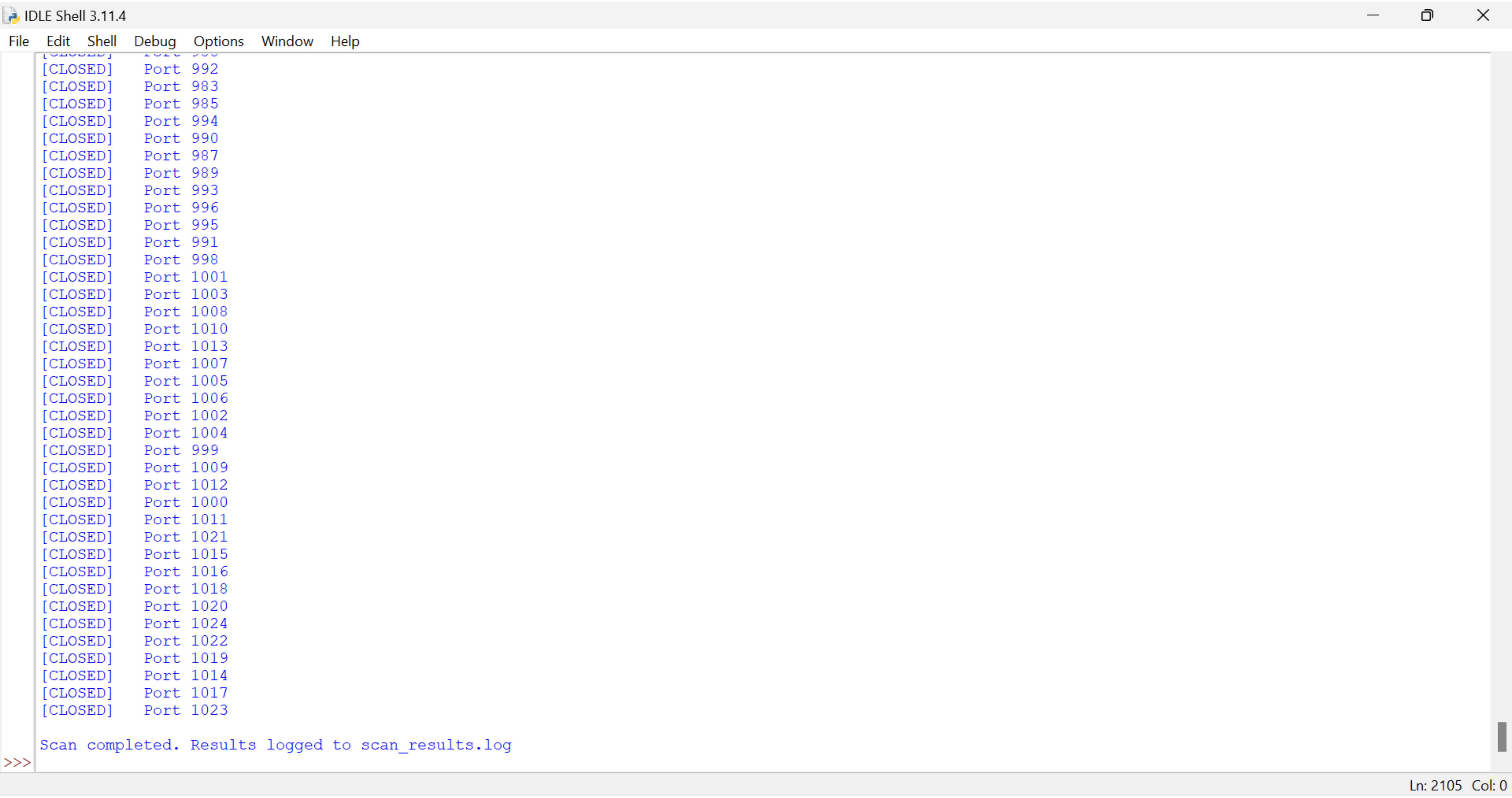
```python
    except socket.timeout:
        return f"[TIMEOUT]  Port {port}"
    except Exception as e:
        return f"[ERROR]    Port {port}: {e}"

# ====================== SCANNER ENGINE ======================
def start_scan(host: str, start_port: int, end_port: int) -> None:
    ports = list(range(start_port, end_port + 1))

    if len(ports) > MAX_PORTS_PER_SCAN:
        print(f"[ERROR] Port range too large ({len(ports)}). Limit is {MAX_PORTS_PER_SCAN}.")
        return

    print(f"\nScanning host: {host}")
    print(f"Ports: {start_port}-{end_port}")
    print(f"Workers: {MAX_WORKERS}\n")

    # --- Fast path with hard crash protection ---
    try:
        with ThreadPoolExecutor(max_workers=MAX_WORKERS) as executor:
            futures = [executor.submit(scan_port, host, p) for p in ports]
            for future in as_completed(futures):
                print(future.result())
    except RuntimeError:
        # Absolute fallback — never crashes
        print("[WARN] Thread limit hit. Falling back to sequential scan.")
        for p in ports:
            print(scan_port(host, p))

    print("\nScan completed. Results logged to scan_results.log")

# ====================== CLI ================================
def main() -> None:
    parser = argparse.ArgumentParser(description="Fast, Safe TCP Port Scanner")
    parser.add_argument("host", nargs="?", help="Target host (IP or hostname)")
    parser.add_argument("-p", "--ports", default=DEFAULT_PORT_RANGE, help="Port range (e.g. 1-1024)")

    args = parser.parse_args()

    host_input = args.host if args.host else DEFAULT_HOST
    if not args.host:
```


port scan project.py - C:/Users/USER/AppData/Local/Programs/Python/Python311/port scan project.py (3.11.4)
File Edit Format Run Options Window Help
Ln: 71 Col: 1

```
    host_input = args.host if args.host else DEFAULT_HOST
    if not args.host:
        print(f"[INFO] Host not provided. Using default: {DEFAULT_HOST}")

    target_ip = validate_host(host_input)

    try:
        start_port, end_port = map(int, args.ports.split("-"))
        if not (1 <= start_port <= end_port <= 65535):
            raise ValueError
    except ValueError:
        print("[ERROR] Invalid port range. Falling back to 1-1024")
        start_port, end_port = 1, 1024

    start_scan(target_ip, start_port, end_port)

# ======================= SELF TESTS ==========================
def _self_test() -> None:
    assert DEFAULT_HOST == "127.0.0.1"

    # invalid numeric IP
    assert validate_host("256.256.256.256") == DEFAULT_HOST

    # invalid hostname syntax
    assert validate_host("this_is_not_a_real_host_name") == DEFAULT_HOST

    # valid IP
    assert validate_host("127.0.0.1") == "127.0.0.1"

    # valid hostname
    assert isinstance(validate_host("localhost"), str)

    # port scan sanity
    r = scan_port("127.0.0.1", 1)
    assert r.startswith("[CLOSED]") or r.startswith("[TIMEOUT]")


if __name__ == "__main__":
    _self_test()
    main()
```



```
[INFO] Host not provided. Using default: 127.0.0.1

Scanning host: 127.0.0.1
Ports: 1-1024
Workers: 16

[CLOSED]    Port 9
[CLOSED]    Port 8
[CLOSED]    Port 4
[CLOSED]    Port 13
[CLOSED]    Port 12
[CLOSED]    Port 3
[CLOSED]    Port 10
[CLOSED]    Port 7
[CLOSED]    Port 2
[CLOSED]    Port 11
[CLOSED]    Port 14
[CLOSED]    Port 6
[CLOSED]    Port 5
[CLOSED]    Port 1
[CLOSED]    Port 15
[CLOSED]    Port 16
[CLOSED]    Port 27
[CLOSED]    Port 18
[CLOSED]    Port 19
[CLOSED]    Port 17
[CLOSED]    Port 24
[CLOSED]    Port 26
[CLOSED]    Port 29
[CLOSED]    Port 28
[CLOSED]    Port 22
[CLOSED]    Port 21
[CLOSED]    Port 30
[CLOSED]    Port 25
[CLOSED]    Port 20
[CLOSED]    Port 23
[CLOSED]    Port 31
[CLOSED]    Port 32
[CLOSED]    Port 38
[CLOSED]    Port 36
[CLOSED]    Port 41
```



```
[CLOSED]   Port 992
[CLOSED]   Port 983
[CLOSED]   Port 985
[CLOSED]   Port 994
[CLOSED]   Port 990
[CLOSED]   Port 987
[CLOSED]   Port 989
[CLOSED]   Port 993
[CLOSED]   Port 996
[CLOSED]   Port 995
[CLOSED]   Port 991
[CLOSED]   Port 998
[CLOSED]   Port 1001
[CLOSED]   Port 1003
[CLOSED]   Port 1008
[CLOSED]   Port 1010
[CLOSED]   Port 1013
[CLOSED]   Port 1007
[CLOSED]   Port 1005
[CLOSED]   Port 1006
[CLOSED]   Port 1002
[CLOSED]   Port 1004
[CLOSED]   Port 999
[CLOSED]   Port 1009
[CLOSED]   Port 1012
[CLOSED]   Port 1000
[CLOSED]   Port 1011
[CLOSED]   Port 1021
[CLOSED]   Port 1015
[CLOSED]   Port 1016
[CLOSED]   Port 1018
[CLOSED]   Port 1020
[CLOSED]   Port 1024
[CLOSED]   Port 1022
[CLOSED]   Port 1019
[CLOSED]   Port 1014
[CLOSED]   Port 1017
[CLOSED]   Port 1023

Scan completed. Results logged to scan_results.log
>>>
```