Kierunki Rozwoju Telekomunikacji - projekt

Dokumentacja projektowa

Skrypt interpretujący adres IP

Projekt wykonali:

Mateusz Furgała

Adam Czerwiec

Paweł Kuźniar

Prowadzący: mgr inż. Natalia Cieplińska

1. Spis Treści

2. Opis ogólny

Skrypt jest narzędziem w Pythonie służącym do analizy i przetwarzania adresów IP w różnych formatach, w tym CIDR i maski dziesiętne. Program umożliwia:

- Obliczenie i wyświetlenie szczegółowych informacji o adresie IP, takich jak adres sieci, maska sieci, adres rozgłoszeniowy, liczba hostów w sieci, oraz reprezentacje binarne i szesnastkowe adresu IP.
- Sprawdzenie, czy dany adres IP należy do specjalnych zakresów (np. adresy prywatne, multicast, loopback, itp.).
- Sprawdzenie, czy adres IP należy do zadanej podsieci.

Skrypt obsługuje różne formaty wejściowe, umożliwiając użytkownikowi wygodne podanie adresu w formacie CIDR (np. 192.168.1.1/24) lub z maską dziesiętną (np. 192.168.1.1 255.255.255.0). Dodatkowo, wyświetla reprezentację binarną i szesnastkową adresu IP.

3. Wymagania

Skrypt wymaga Pythona w wersji 3.x oraz modułu ipaddress, który jest standardowym modułem w Pythonie do pracy z adresami IP.

4. Użyte moduły

- ipaddress – Wbudowany moduł w Pythonie do pracy z adresami IP. Używany do walidacji adresów IP, konwersji do różnych formatów oraz obliczania informacji o sieciach.
- argparse – Moduł do obsługi argumentów wiersza poleceń, pozwala na łatwe zarządzanie wejściami użytkownika.

5. Opis funkcji

a. validate_netmask(mask)

- Opis: Sprawdza poprawność maski sieci w formacie dziesiętnym.
- Argumenty: mask – Maska sieci w formacie dziesiętnym (np. 255.255.255.0).
- Zwraca: Wartość prefiksu sieci (np. 24).
- Podnosi wyjątek: ValueError w przypadku nieprawidłowej maski.

b. parse_ip_and_mask(ip, mask=None)

- Opis: Rozpoznaje, czy podano CIDR (np. 192.168.1.1/24), czy adres z maską dziesiętną (np. 192.168.1.1 255.255.255.0) oraz przetwarza je na obiekt ip_interface.
- Argumenty:
  - ip – Adres IP.
  - mask – Opcjonalna maska sieci.
- Zwraca: Obiekt ip_interface (np. IPv4Interface('192.168.1.1/24')).
- Podnosi wyjątek: ValueError w przypadku nieprawidłowego formatu adresu.

c. check_special_address(ip_obj)

- Opis: Sprawdza, czy adres IP należy do specjalnych kategorii adresów, takich jak: loopback, multicast, prywatny (RFC1918), link-local (APIPA), zarezerwowany, czy nieokreślony.
- Argumenty: ip_obj – Obiekt reprezentujący adres IP.
- Zwraca: String z nazwą kategorii, do której należy adres IP (np. "Prywatny (RFC1918)").

d. ip_info(ip_obj)

- Opis: Wyświetla szczegółowe informacje o adresie IP, w tym adres sieciowy, maskę sieci, adres rozgłoszeniowy, liczbę hostów w sieci oraz reprezentację binarną i szesnastkową.
- Argumenty: ip_obj – Obiekt reprezentujący adres IP.
- Zwraca: Brak, funkcja jedynie wyświetla informacje.

e. is_ip_in_subnet(ip, subnet)

- Opis: Sprawdza, czy adres IP należy do zadanej podsieci.
- Argumenty:
  - ip – Adres IP.
  - subnet – Podsiec w formacie CIDR (np. 192.168.1.0/24).
- Zwraca: True jeśli adres IP należy do podsieci, False w przeciwnym razie.

f. main()

- Opis: Główna funkcja, która przetwarza argumenty wejściowe, wywołuje odpowiednie funkcje oraz wyświetla wyniki.
- Argumenty: Brak, argumenty są przekazywane przez wiersz poleceń.
- Zwraca: Brak, funkcja jedynie wyświetla informacje.

6. Opis argumentów wejściowych

Skrypt obsługuje następujące argumenty wejściowe:

- ip (wymagane): Adres IP w jednym z następujących formatów:
  - Adres IP: 192.168.1.1 (Podejście klasowe – adres traktowany jako Klasy C).
  - CIDR: 192.168.1.1/24 (adres IP z CIDR).
  - Adres IP i maska dziesiętna: 192.168.1.1 255.255.255.0 (adres IP z maską).
  - CIDR z odstępem: 192.168.1.1 /24 (adres IP z prefiksem po odstępie).
- -n, --network (opcjonalnie): Określa adres podsieci, do której należy sprawdzić przynależność adresu IP (np. 192.168.1.0/24).
- -h, –help: wyświetla pomoc opisując użycie funkcji w sposób przedstawiony poniżej:

```
C:\Users\pkuzniar\PycharmProjects\IPInfo>python ip.py -h
usage: ip.py [-h] [-n ADRES [MASKA ...]] ip [ip ...]

Podręczne narzędzie sieciowe – sprawdzanie adresów IP i przynależności do sieci.
Użycie: <ADRES IP SPRAWDZANY> <ADRES DOCELOWY/CIDR> lub <ADRES DOCELOWY> <MASKA>

positional arguments:
  ip                    Adres IP w jednym z formatów:
                          • IP (Podejście klasowe):  192.168.1.1
                          • CIDR:                    192.168.1.1/24
                          • IP + maska dziesiętna:   192.168.1.1 255.255.255.0

options:
  -h, --help            show this help message and exit
  -n ADRES [MASKA ...], --network ADRES [MASKA ...]
                        Sprawdź przynależność podanego adresu IP do danej sieci:

                        Sieć docelową można określić za pomocą:
                          • CIDR:              192.168.1.0/24
                          • ADRES + MASKA:     192.168.1.0 255.255.255.0

                        Przykłady użycia:
                          python ip.py 192.168.1.100 -n 192.168.1.0/24
                          python ip.py 192.168.1.100 -n 192.168.1.0 255.255.255.0

                        Jeśli adres sieci jest nieprawidłowy, zgłoszony zostanie błąd.
```

7. Opis wyników

Skrypt wyświetla następujące informacje na temat adresu IP:

- Typ adresu: IPv4 lub IPv6.
- Adres IP: Wartość adresu IP.
- Adres sieci: Adres sieciowy wynikający z podanego adresu IP i maski.
- Maska sieci: Maska sieciowa.
- Adres rozgłoszeniowy: Adres broadcastowy dla sieci IPv4, dla IPv6 jest to "N/A".
- Liczba hostów w sieci: Liczba hostów dostępnych w danej sieci (dla IPv4 z wyłączeniem adresów sieciowego i rozgłoszeniowego).
- Reprezentacja binarna: Reprezentacja binarna adresu IP.
- Reprezentacja szesnastkowa: Reprezentacja szesnastkowa adresu IP.
- Kategoria adresu: Określenie, czy adres IP należy do specjalnej kategorii (loopback, multicast, prywatny, itp.).

Ponadto, skrypt wyświetla informację, czy dany adres IP należy do wskazanej podsieci (jeśli opcja --network została użyta).

8. Przykłady użycia

```
============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 192.168.1.100
------------------------------------------------------------
============ Informacje o adresie 192.168.1.100 ============
Klasa adresu:               Klasa C
Adres prywatny:             Tak
Typ adresu:                 IPv4
Maska sieci:                255.255.255.0
Adres sieci:                192.168.1.0
Adres rozgłoszeniowy:       192.168.1.255
Liczba hostów w sieci:      254
Liczba adresów w sieci:     256
Reprezentacja binarna:      11000000101010000000000101100100
Reprezentacja szesnastkowa: 0xc0a80164
Kategoria adresu:           Prywatny (RFC1918)
============================================================
============================================================
```

```
============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 169.254.0.1
------------------------------------------------------------
============ Informacje o adresie 169.254.0.1 ============
Klasa adresu:               Klasa B
Adres prywatny:             Tak
Typ adresu:                 IPv4
Maska sieci:                255.255.0.0
Adres sieci:                169.254.0.0
Adres rozgłoszeniowy:       169.254.255.255
Liczba hostów w sieci:      65534
Liczba adresów w sieci:     65536
Reprezentacja binarna:      10101001111111100000000000000001
Reprezentacja szesnastkowa: 0xa9fe0001
Kategoria adresu:           APIPA (Link-local)
============================================================
============================================================
```

```
============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 100.230.8.73/17
------------------------------------------------------------
============ Informacje o adresie 100.230.8.73 ============
Klasa adresu:               Brak
Adres prywatny:             Nie
Typ adresu:                 IPv4
Maska sieci:                255.255.128.0
Adres sieci:                100.230.0.0
Adres rozgłoszeniowy:       100.230.127.255
Liczba hostów w sieci:      32766
Liczba adresów w sieci:     32768
Reprezentacja binarna:      01100100111001100000100001001001
Reprezentacja szesnastkowa: 0x64e60849
Kategoria adresu:           Brak specjalnej kategorii
============================================================
============================================================
```

```
============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 74.125.200.100 255.255.252.0
------------------------------------------------------------
============ Informacje o adresie 74.125.200.100 ============
Klasa adresu:               Brak
Adres prywatny:             Nie
Typ adresu:                 IPv4
Maska sieci:                255.255.252.0
Adres sieci:                74.125.200.0
Adres rozgłoszeniowy:       74.125.203.255
Liczba hostów w sieci:      1022
Liczba adresów w sieci:     1024
Reprezentacja binarna:      01001010011111101110010000 1100100
Reprezentacja szesnastkowa: 0x4a7dc864
Kategoria adresu:           Brak specjalnej kategorii
============================================================
============================================================
```

```
============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 2001:db8::1/64
------------------------------------------------------------
============ Informacje o adresie 2001:db8::1 ============
Klasa adresu:               Brak
Adres prywatny:             Tak
Typ adresu:                 IPv6
Maska sieci:                /64
Adres sieci:                N/A [IPv6]
Adres rozgłoszeniowy:       N/A [IPv6]
Liczba hostów w sieci:      18446744073709551616
Liczba adresów w sieci:     18446744073709551616
Reprezentacja binarna:      00100000000000010000110110111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
Reprezentacja szesnastkowa: 2001:0db8:0000:0000:0000:0000:0000:0001
Kategoria adresu:           Documentation (2001:db8::/32)
============================================================
============================================================
```

```
============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py ff02::1/128
------------------------------------------------------------
============ Informacje o adresie ff02::1 ============
Klasa adresu:               Brak
Adres prywatny:             Nie
Typ adresu:                 IPv6
Maska sieci:                /128
Adres sieci:                N/A [IPv6]
Adres rozgłoszeniowy:       N/A [IPv6]
Liczba hostów w sieci:      N/A
Liczba adresów w sieci:     1
Reprezentacja binarna:      11111111000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
Reprezentacja szesnastkowa: ff02:0000:0000:0000:0000:0000:0000:0001
Kategoria adresu:           Multicast
============================================================
============================================================
```

```
============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py fe80::1/64
------------------------------------------------------------
============ Informacje o adresie fe80::1 ============
Klasa adresu:               Brak
Adres prywatny:             Tak
Typ adresu:                 IPv6
Maska sieci:                /64
Adres sieci:                N/A [IPv6]
Adres rozgłoszeniowy:       N/A [IPv6]
Liczba hostów w sieci:      18446744073709551616
Liczba adresów w sieci:     18446744073709551616
Reprezentacja binarna:      11111110100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
Reprezentacja szesnastkowa: fe80:0000:0000:0000:0000:0000:0000:0001
Kategoria adresu:           APIPA (Link-local)
============================================================
============================================================
```

```
============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 192.168.1.100 -n 192.168.1.0/24
------------------------------------------------------------
Adres IP 192.168.1.100 należy do sieci 192.168.1.0/24
============================================================


============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 192.168.1.100 -n 10.0.0.0/8
------------------------------------------------------------
Adres IP 192.168.1.100 NIE należy do sieci 10.0.0.0/8
============================================================


============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 192.168.1.100 -n 192.168.1.0 255.255.255.0
------------------------------------------------------------
Adres IP 192.168.1.100 należy do sieci 192.168.1.0 255.255.255.0
============================================================


============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 192.168.1.100 -n 192.168.0.0 255.255.0.0
------------------------------------------------------------
Adres IP 192.168.1.100 należy do sieci 192.168.0.0 255.255.0.0
============================================================
```

Obsługa błędów:

```
============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 300.300.300.300
------------------------------------------------------------
Błąd: Nieprawidłowy adres IP: 300.300.300.300
============================================================


============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 192.168.1.100 -n 192.168.1.0 255.300.255.0
------------------------------------------------------------
Nieprawidłowa sieć: 192.168.1.0 255.300.255.0 (Nieprawidłowa sieć lub maska: 192.168.1.0 255.300.255.0)
============================================================


============================================================
Uruchamianie: C:\Python312\python.exe C:\Users\pkuzniar\PycharmProjects\IPInfo\ip.py 192.168.1.100 -n 100.100.100.300 255.255.265.255
------------------------------------------------------------
Nieprawidłowa sieć: 100.100.100.300 255.255.265.255 ('100.100.100.300 255.255.265.255' does not appear to be an IPv4 or IPv6 network)
============================================================
```

9. Wnioski

Skrypt jest elastycznym narzędziem, które umożliwia szybkie analizowanie adresów IP, sprawdzanie ich przynależności do specjalnych zakresów oraz wykonywanie obliczeń na temat podsieci. Dzięki zastosowaniu modułów ipaddress i argparse, program jest łatwy w użyciu i może być wykorzystany zarówno przez administratorów sieci, jak i programistów zajmujących się sieciami komputerowymi

10. Dodatek A (Kod programu)

```
import ipaddress
import argparse

# Predefiniowane sieci specjalne
special_networks = [
    ("Loopback", lambda ip: ip.is_loopback),
    ("Test-Net (192.0.2.0/24)", lambda ip: ip in
ipaddress.IPv4Network("192.0.2.0/24")),
    ("Test-Net (198.51.100.0/24)", lambda ip: ip in
ipaddress.IPv4Network("198.51.100.0/24")),
    ("Test-Net (203.0.113.0/24)", lambda ip: ip in
ipaddress.IPv4Network("203.0.113.0/24")),
    ("Zarezerwowany (240.0.0.0/4)", lambda ip: ip in
ipaddress.IPv4Network("240.0.0.0/4")),
    ("Documentation (2001:db8::/32)", lambda ip: ip in
ipaddress.IPv6Network("2001:db8::/32")),
    ("Multicast", lambda ip: ip.is_multicast),
    ("APIPA (Link-local)", lambda ip: ip.is_link_local),
    ("Zarezerwowany (Reserved)", lambda ip: ip.is_reserved),
    ("Nieokreślony (Unspecified)", lambda ip: ip.is_unspecified),
    ("Prywatny (RFC1918)", lambda ip: ip.is_private),
]

# Predefiniowane kolory
COLORS = {
    "header": "\033[1;92m",    # Zielony pogrubiony
    "key": "\033[1;96m",       # Cyan pogrubiony
    "accent": "\033[1;93m",    # Żółty pogrubiony
    "error": "\033[1;91m",     # Czerwony pogrubiony
    "reset": "\033[0m"
}


def format_text(text, color="reset"): # Funkcja formatująca tekst wyjściowy
    return f"{COLORS[color]}{text}{COLORS['reset']}"


def get_classful_mask(ip):
    """Zwraca domyślną maskę sieci i klasę adresu."""
    first_octet = int(ip.split('.')[0])

    if 1 <= first_octet <= 127:   # Klasa A
        is_private = ipaddress.IPv4Address(ip).is_private
        return "255.0.0.0", "Klasa A", is_private

    elif 128 <= first_octet <= 191:  # Klasa B
        is_private = ipaddress.IPv4Address(ip).is_private
```

```
        return "255.255.0.0", "Klasa B", is_private

    elif 192 <= first_octet <= 223:  # Klasa C
        is_private = ipaddress.IPv4Address(ip).is_private
        return "255.255.255.0", "Klasa C", is_private

    elif 224 <= first_octet <= 239:  # Klasa D (Multicast)
        return None, "Klasa D (Multicast - Brak maski)", False

    elif 240 <= first_octet <= 255:  # Klasa E (Eksperymentalne)
        return None, "Klasa E (Zarezerwowana - Brak maski)", False

    else:
        raise ValueError(f"Adres IP {ip} nie należy do standardowych klas.")


def validate_netmask(mask):  # Sprawdzenie poprawności maski sieci
    try:
        network = ipaddress.IPv4Network(f"0.0.0.0/{mask}", strict=False)
        return network.prefixlen
    except ValueError:
        raise ValueError(f"Nieprawidłowa maska sieci: {mask}")


def convert_to_cidr(network_str): # Konwersja IP + maska dziesiętna na CIDR
    try:
        ip, mask = network_str.split()
        prefixlen = validate_netmask(mask)
        return f"{ip}/{prefixlen}"
    except (ValueError, IndexError):
        raise ValueError(f"Nieprawidłowa sieć lub maska: {network_str}")


def parse_ip_and_mask(ip, mask=None): # Sprawdzenie formatu IP z obsługą klas
IPv4 i adresów prywatnych
    try:
        if mask and mask.startswith("/"):  # Gdy podano CIDR po spacji
            prefixlen = int(mask[1:])
            ip_obj = ipaddress.ip_interface(f"{ip}/{prefixlen}")
            ip_class = "CIDR"
            is_private = ip_obj.ip.is_private
            return ip_obj, ip_class, is_private

        if mask is None:
            if "/" in ip:  # Gdy podano CIDR bez spacji
                ip = ip.replace(" ", "")
                ip_obj = ipaddress.ip_interface(ip)
                ip_class = "CIDR"
                is_private = ip_obj.ip.is_private
```

```
                return ip_obj, ip_class, is_private
            elif ":" not in ip:  # Gdy podano IPv4 (podejście klasowe)
                mask, ip_class, is_private = get_classful_mask(ip)
                ip_obj = ipaddress.ip_interface(f"{ip}/{mask}")
                return ip_obj, ip_class, is_private
            else:
                raise ValueError("Adres IPv6 wymaga prefiksu CIDR(np.
2001:db8::1/64).") # Gdy podano IPv6 bez prefiksu CIDR

        # Jeśli podano maskę dziesiętną, konwersja na prefiks CIDR
        prefixlen = validate_netmask(mask)
        ip_obj = ipaddress.ip_interface(f"{ip}/{prefixlen}")
        ip_class = "CIDR"
        is_private = ip_obj.ip.is_private
        return ip_obj, ip_class, is_private

    except ipaddress.AddressValueError as e:
        raise ValueError(f"Nieprawidłowy adres IP: {e}")
    except ipaddress.NetmaskValueError as e:
        raise ValueError(f"Nieprawidłowa maska sieci: {e}")
    except ValueError as e:
        raise ValueError(f"Ogólny błąd: {e}")


def check_special_address(ip_obj):  # Sprawdzenie, czy adres należy do
specjalnych kategorii
    ip = ip_obj.ip

    # Dopasowanie kategorii sieci
    for category, condition in special_networks:
        if condition(ip):
            return category

    return "Brak specjalnej kategorii"


def ip_info(ip_obj, ip_class, is_private):
    ip_version = ip_obj.version
    ip_address = ip_obj.ip

    netmask = str(ip_obj.network.netmask) if ip_version == 4 else
f"/{ip_obj.network.prefixlen}"
    network_address = ip_obj.network.network_address if ip_version == 4 else
"N/A [IPv6]"
    broadcast_address = ip_obj.network.broadcast_address if ip_version == 4
else "N/A [IPv6]"

    if ip_obj.network.prefixlen == ip_obj.network.max_prefixlen:
        num_hosts = "N/A"
```

```
    else:
        num_hosts = ip_obj.network.num_addresses - (2 if ip_version == 4 else
0)

    num_addresses = ip_obj.network.num_addresses
    hex_representation = hex(int(ip_address)) if ip_version == 4 else
ip_address.exploded

    private_status = "Tak" if is_private else "Nie"

    print(format_text(f"============ Informacje o adresie {ip_address}
============", "header"))

    info_list = [
        ("Klasa adresu:", ip_class),
        ("Adres prywatny:", private_status),
        ("Typ adresu:", f"IPv{ip_version}"),
        ("Maska sieci:", netmask),
        ("Adres sieci:", network_address),
        ("Adres rozgłoszeniowy:", broadcast_address),
        ("Liczba hostów w sieci:", num_hosts),
        ("Liczba adresów w sieci:", num_addresses),
        ("Reprezentacja binarna:", ''.join(f'{int(octet):08b}' for octet in
ip_address.packed)),
        ("Reprezentacja szesnastkowa:", hex_representation),
        ("Kategoria adresu:", check_special_address(ip_obj)),
    ]

    for key, value in info_list:
        print(f"{format_text(key.ljust(27), 'key')} {value}")

    print(format_text("==========================================================
===", "header"))


def main():
    import argparse

    parser = argparse.ArgumentParser(
        description=(
            "Podręczne narzędzie sieciowe - sprawdzanie adresów IP i
przynależności do sieci.\n"
            "Użycie: <ADRES IP SPRAWDZANY> <ADRES DOCELOWY/CIDR> lub <ADRES
DOCELOWY> <MASKA>"
        ),
        formatter_class=argparse.RawTextHelpFormatter
    )

    # Argument IP
```

```
    parser.add_argument(
        "ip",
        help=(
            "Adres IP w jednym z formatów:\n"
            "  • CIDR:                  192.168.1.1/24\n"
            "  • IP + maska dziesiętna: 192.168.1.1 255.255.255.0"
        ),
        nargs='+'
    )

    # Argument -n / --network
    parser.add_argument(
        "-n", "--network",
        nargs="+",
        metavar=("ADRES", "MASKA"),
        help=(
            "Sprawdź przynależność podanego adresu IP do danej sieci:\n"
            "\n"
            "Sieć docelową można określić za pomocą:\n"
            "  • CIDR:               192.168.1.0/24\n"
            "  • ADRES + MASKA:      192.168.1.0 255.255.255.0\n"
            "\n"
            "Przykłady użycia:\n"
            "  python ip.py 192.168.1.100 -n 192.168.1.0/24\n"
            "  python ip.py 192.168.1.100 -n 192.168.1.0 255.255.255.0\n"
            "\n"
            "Jeśli adres sieci jest nieprawidłowy, zgłoszony zostanie błąd."
        )
    )

    args = parser.parse_args()
    args.ip = [ip.strip() for ip in args.ip]

    try:
        # Rozpoznanie formatu wejściowego
        if len(args.ip) == 1:
            ip_obj, ip_class, is_private = parse_ip_and_mask(args.ip[0])  #
CIDR lub classful
        elif len(args.ip) == 2:
            ip_obj, ip_class, is_private = parse_ip_and_mask(args.ip[0],
args.ip[1])  # IP + maska dziesiętna
        else:
            raise ValueError("Nieprawidłowy format wejściowy. Podaj adres w
formacie CIDR lub z maską dziesiętną.")

        # Sprawdzanie przynależności do sieci
        if args.network:
            try:
                # Przechowujemy oryginalny format wejściowy
```

```
                original_network_format = " ".join(args.network)

                # Obsługa CIDR lub maski dziesiętnej
                if len(args.network) == 2:
                    network_cidr = convert_to_cidr(original_network_format)
                elif len(args.network) == 1:
                    network_cidr = args.network[0]
                else:
                    raise ValueError(f"Nieprawidłowa sieć:
{original_network_format}")

                # Sprawdzenie przynależności do sieci
                network = ipaddress.ip_network(network_cidr, strict=False)

                # Wyświetlenie wyniku w oryginalnym formacie
                if ip_obj.ip in network:
                    print("Adres IP " +
                          format_text(f"{ip_obj.ip} ", "accent") +
                          format_text("należy", "header") +
                          " do sieci " +
                          format_text(f"{original_network_format}", "accent"))
                else:
                    print("Adres IP " +
                          format_text(f"{ip_obj.ip} ", "accent") +
                          format_text("NIE należy", "error") +
                          " do sieci " +
                          format_text(f"{original_network_format}", "accent"))
            except ValueError as e:
                print(format_text(f"Nieprawidłowa sieć:
{original_network_format} ({e})", "error"))
        else:
            ip_info(ip_obj, ip_class, is_private)  # Wyświetlenie informacji o
adresie IP

    except ValueError as e:
        print(f"Błąd: {e}")


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