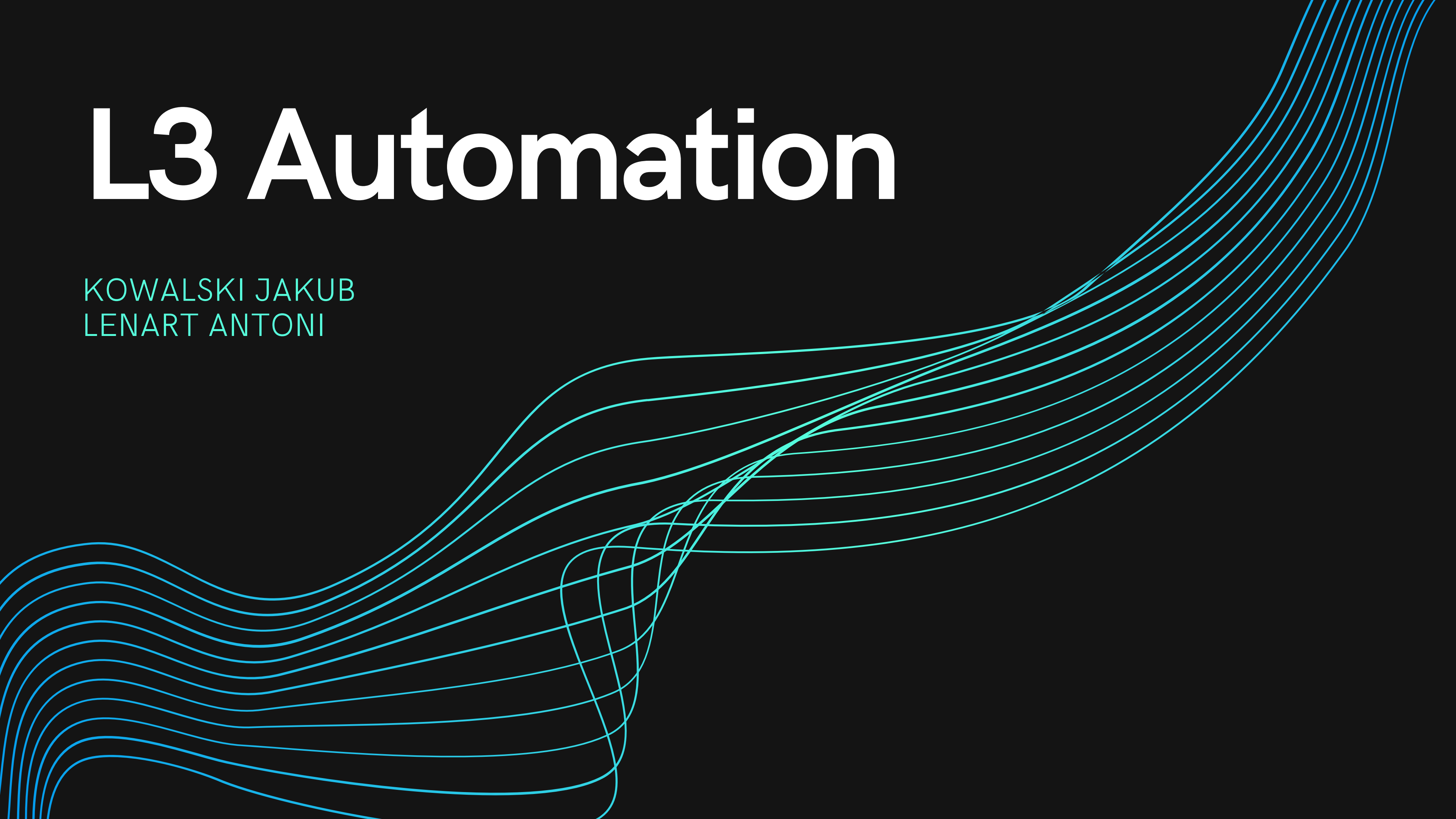


L3 Automation

KOWALSKI JAKUB
LENART ANTONI

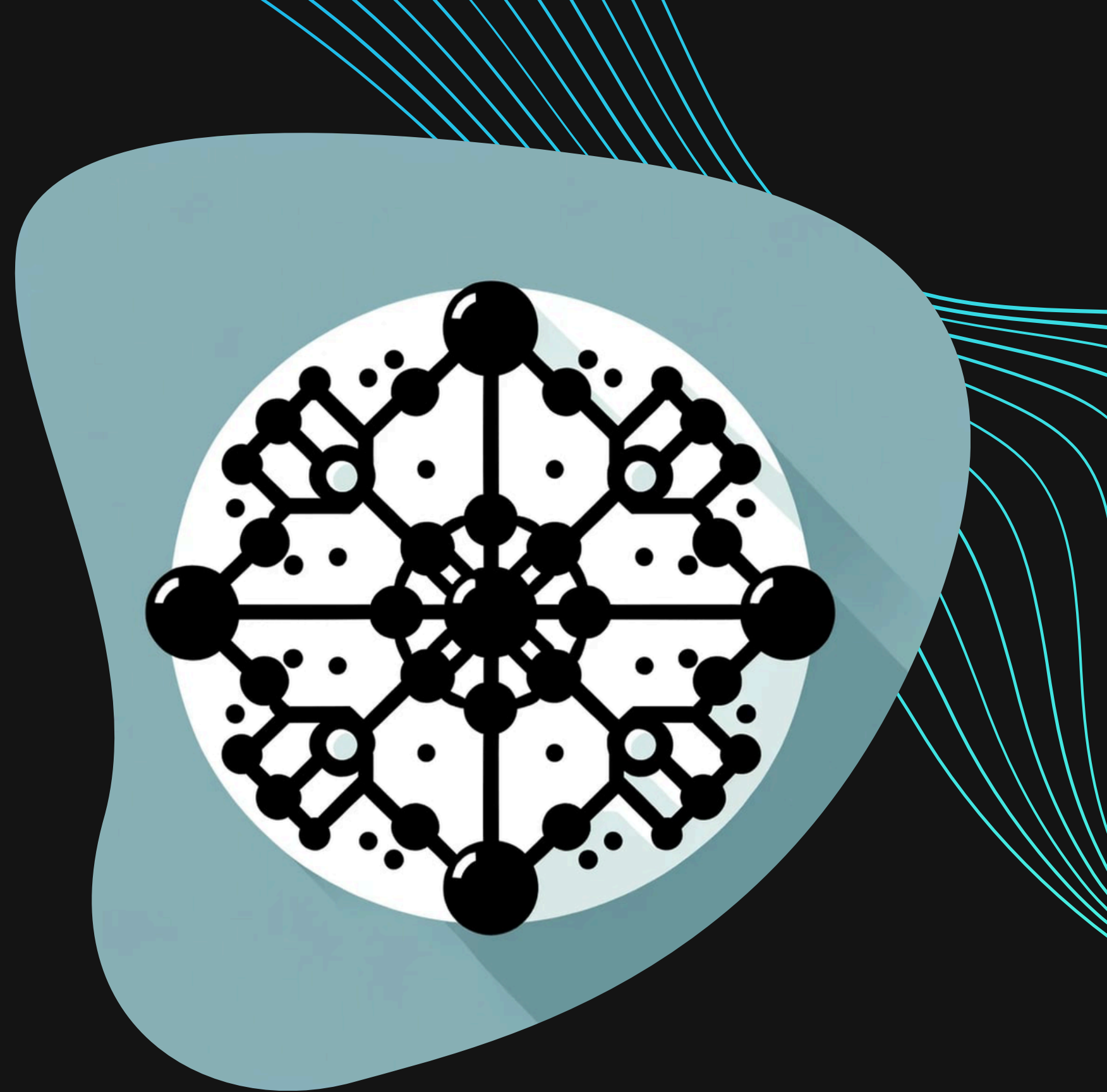


Cele projektu

AUTOMATYZACJA
PROTOKOŁÓW SIECIOWYCH

POŁĄCZENIE PYTHONA Z
ZAGADNIENIAMI SIECIOWYMI

ROZWINIĘCIE WIEDZY
ZDOBYTEJ NA PRZEDMIOCIE
'SIECI IP'





Technologie

GNS 3

Program do symulacji
sieciowych

PYTHON TKINTER

Biblioteka do tworzenia GUI

NETMIKO

Biblioteka do tworzenia sesji
SSH z urządzeniami sieciowymi



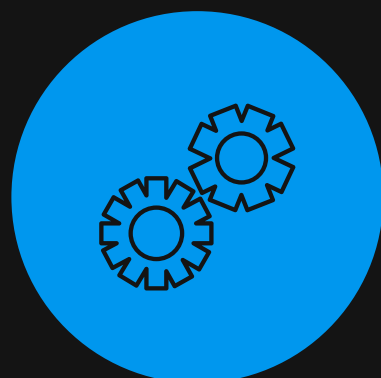
Graphical Network
Simulator-3



Emulacja Sieci i Urządzeń



Wsparcie dla Wielu Platform



Intuicyjny Interfejs Graficzny

GNS 3



GNS3[®]

GNS3: Otwarte Oprogramowanie do Wirtualizacji Sieci

GNS3 to potężne narzędzie, które umożliwia wirtualizację sieci komputerowych w celu testowania, projektowania i symulacji.

- Tworzenie złożonych sieci do testów i eksperymentów.
- Symulacja różnych konfiguracji sieciowych w bezpiecznym środowisku.
- Kompatybilność z wieloma platformami wirtualizacyjnymi i sprzętowymi.
- Dostęp do szerokiej gamy urządzeń sieciowych do konfiguracji.
- Testowanie konfiguracji przed wdrożeniem.
- Nauka i szkolenia w zakresie sieci komputerowych.
- Tworzenie laboratoriów do badania różnych scenariuszy sieciowych.



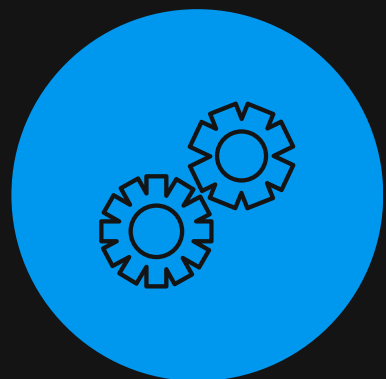
Standardowa biblioteka



Interfejs biblioteki Tk



Dostępny w standardowej
instalacji Pythona



Nie wymaga dodatkowego
instalowania

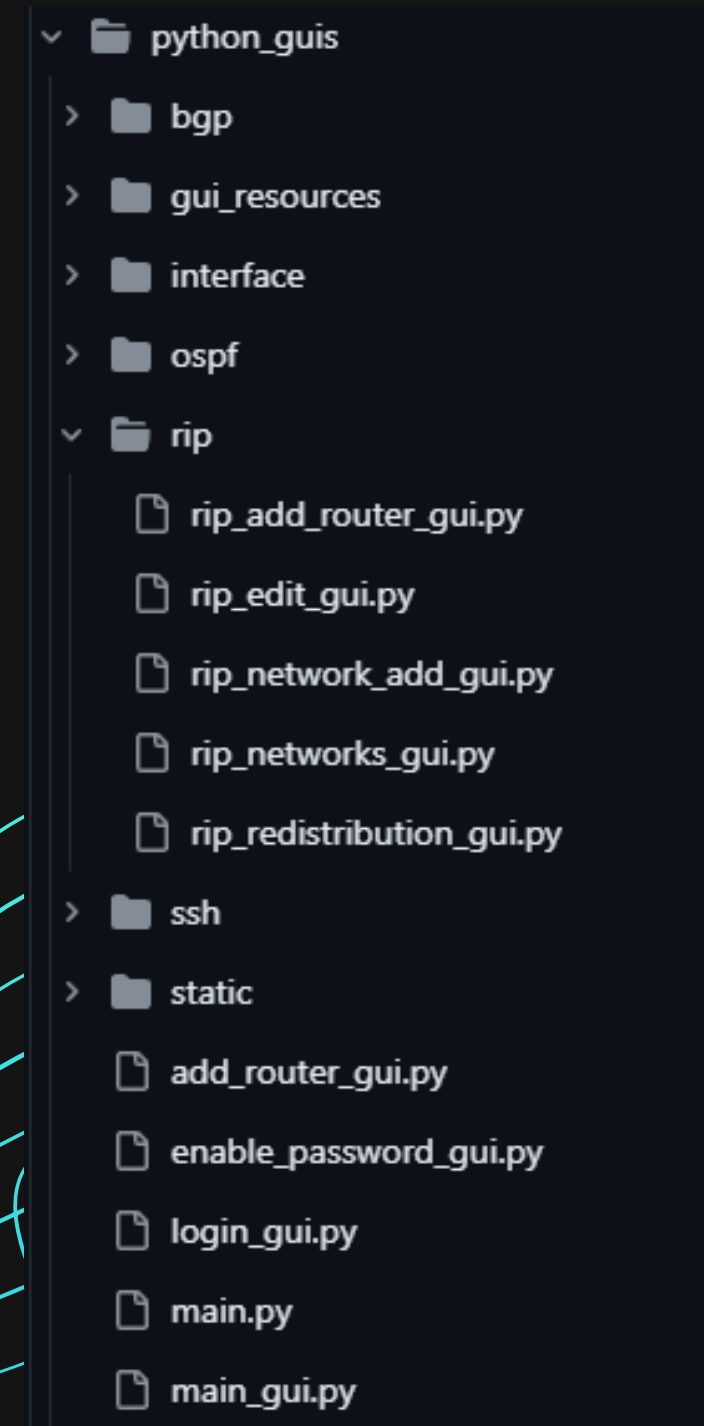
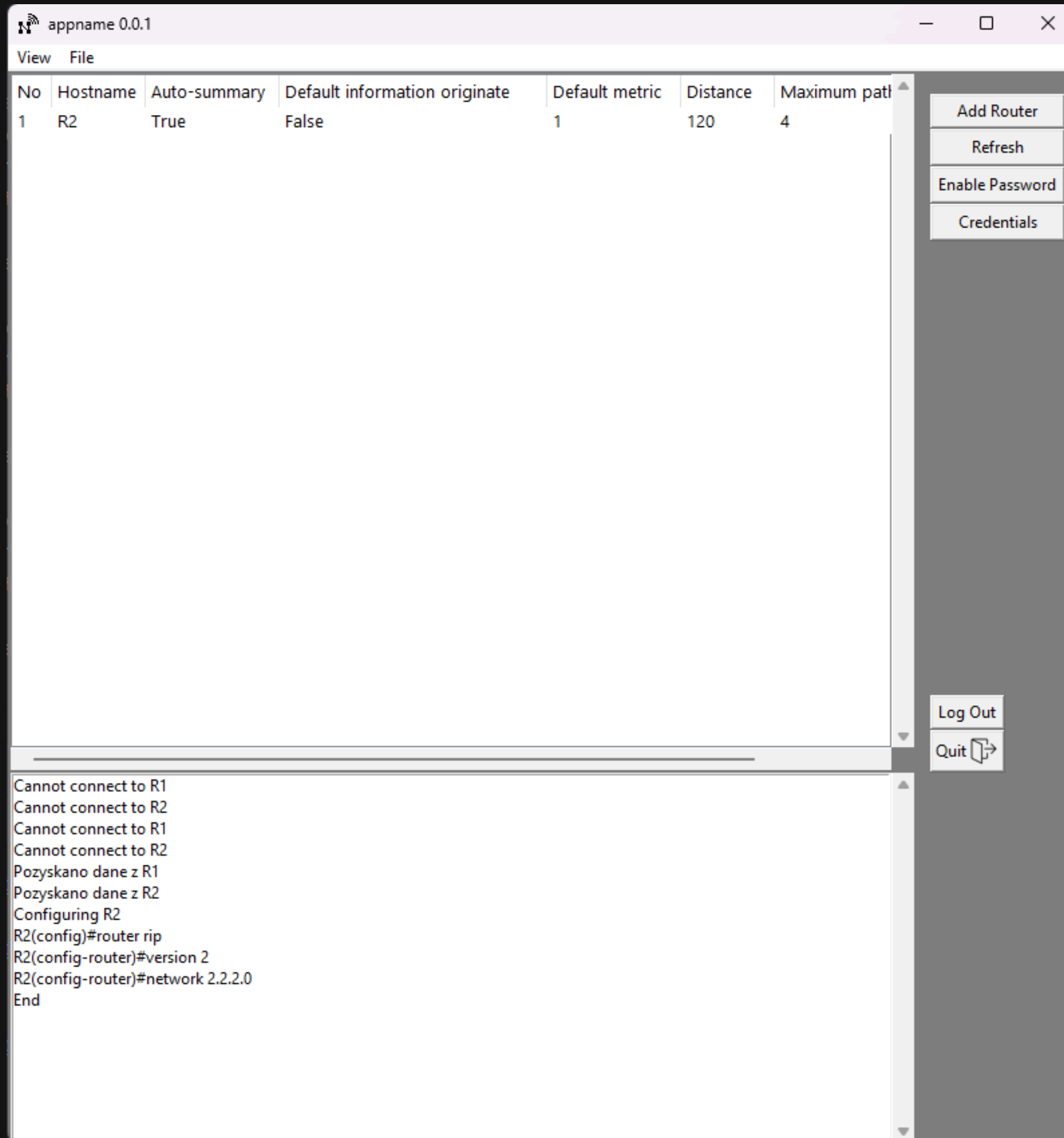
Tkinter (Tk Interface)



Tkinter

Front - czyli GUI

Graphical User Interface




```
42  ✓ class MainGUI:
43      ALL = 'all'
44      RIP = 'rip'
45      OSPF = 'ospf'
46      BGP = 'bgp'
47
48  ✓ def __init__(self):
49      self.root = tk.Tk()
```

```
def open_project():
    files = [('L3 Project Files', '*.jkal')]
    file = askopenfilename(filetypes=files)
    if not file:
        return
    self.project.file_path = file
    root = tk.Tk()
    root.withdraw()
    aes_key = SHA256.new(tk.simpledialog.askstring("Password", "Enter password:").encode()).hexdigest()[:32]
    root.destroy()
    self.project.open_project(aes_key)
    self.show_view_all()
```

main_gui

Kręgosłup całej aplikacji, w niej uruchomiony jest główna instancja Tkinter i w niej przechowywane są dane o urządzeniach.

```
def show_view_ospf(self) -> None:
    self.clear_tree()
    self.current_view = MainGUI.OSPF
```

VIEWS

Dla każdego protokołu sieciowego pokazywany jest inny widok w main_gui, aby zwiększyć przejrzystość konfiguracji urządzeń.

```
def show_view_rip(self) -> None:
    self.clear_tree()
    self.current_view = MainGUI.RIP
```

```
def show_view_bgp(self) -> None:
    self.clear_tree()
    self.current_view = MainGUI.BGP
```

```
def show_view_all(self) -> None:
    self.clear_tree()
    self.current_view = MainGUI.ALL
```

Komunikacja z back-endem

```
def apply_network():
    if validate_network():
        network = get_network()
        mask = get_mask()
        threading.Thread(target=add_rip_networks,
                        args=(main_gui, rip_networks_gui, router, user, [network])).start()

    clean_entries()
```

```
def add_rip_networks(main_gui, rip_networks_gui, router: Router, user: User, networks: list[str]) -> None:
    try:
        completed, output = universal_router_commands.add_rip_networks(router, user, networks)
    except netmiko.exceptions.NetMikoTimeoutException and netmiko.exceptions.ReadTimeout:
        main_gui.console_commands(f'Cannot connect to {router.name} due to timeout')
        return None
    except netmiko.exceptions.NetMikoAuthenticationException:
        main_gui.console_commands(f'Cannot connect to {router.name} due to authentication error')
        return None

    if completed:
        main_gui.console_commands(output)
        router.rip = universal_router_commands.get_rip(None, router, user)
        rip_networks_gui.update_window()

    return None
```

Netmiko



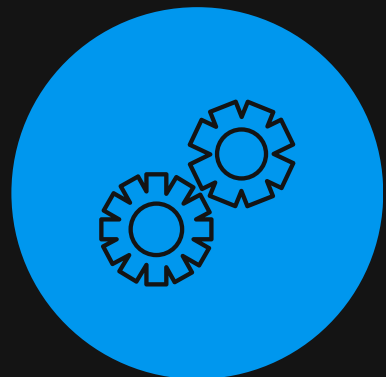
Automatyzacja Zarządzania
Siecią



Wsparcie dla Wielu
Dostawców



Prostota i Łatwość
Użycia



Integracja z Narzędziami
DevOps



NETMiko

Netmiko

Biblioteka netmiko jest popularnym narzędziem używanym do automatyzacji zadań sieciowych za pomocą skryptów Python. Opiera się na bibliotece paramiko, która zapewnia funkcjonalność SSH. netmiko jest zoptymalizowana do pracy z urządzeniami sieciowymi różnych producentów, takich jak Cisco, Juniper, Arista, HP, i wielu innych.

ConnectHandler

```
from netmiko import ConnectHandler

connection = ConnectHandler(
    device_type='cisco_ios',
    host='10.10.10.10',
    username='admin',
    password='password',
    secret='secret', # jeśli jest wymagana
    port=22, # domyślnie 22, ale można zmienić
```

```
)
```


Netmiko

Przydatne metody ConnectHandler

Wysłanie komendy do urządzenia sieciowego i uzyskanie outputu z połączenia vty

```
output = connection.send_command('show ip interface brief')
```

Przejdźcie do trybu privilege exec mode

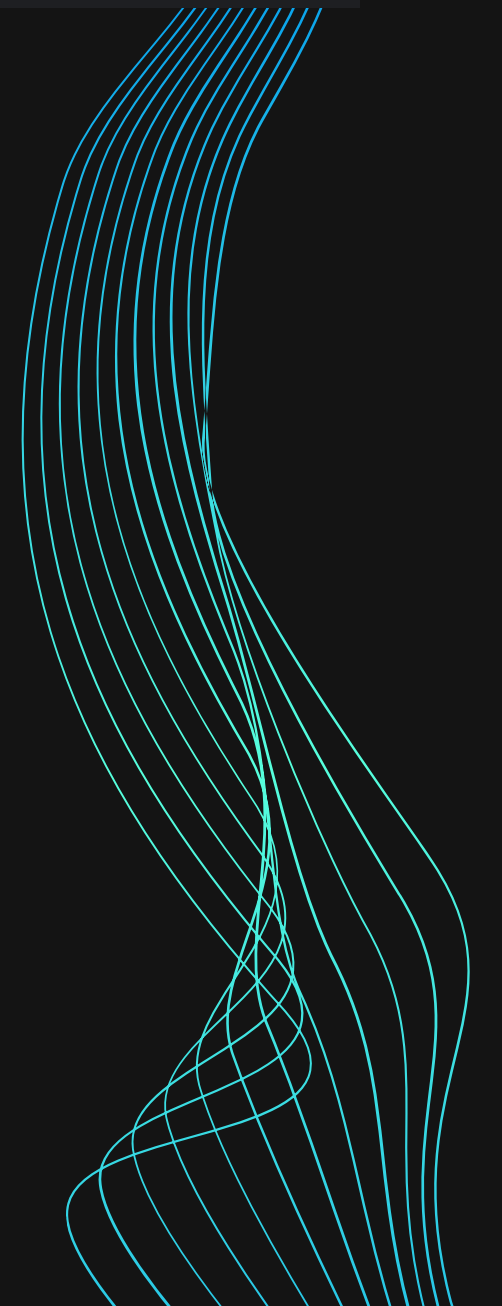
```
connection.enable()
```

Wyjście z trybu privilege exec mode

```
connection.exit_enable_mode()
```

Wysłanie komend z pliku (po przejściu do privilege exec mode)

```
output = connection.send_config_from_file('config.txt')
```



Netmiko


Przydatne metody ConnectHandler

Wysłanie listy komend do urządzenia

```
config_commands = [  
    'interface GigabitEthernet0/1',  
    'description Connected to Router1',  
    'no shutdown'  
]  
output = connection.send_config_set(config_commands)
```

Zakończenie połączenia

```
connection.disconnect()
```



Context Manager

Pozwalają na dynamiczne alokowanie i zwalnianie zasobów. Idealnie nadają się w sytuacjach, w których fragment naszego kodu wykonuje się pomiędzy stałą częścią początkową i końcową.

Stosowany ze słowem kluczowym **with**.

Realizacja połączenia i wykonania poleceń bez context manager. Rozbicie kodu na funkcje w celu zwiększenia czytelności.

Funkcja początkowa

```
def create_connection_to_router(router: Router, user: User) -> netmiko.BaseConnection:
    for key, ssh_ip in router.ssh_information.ip_addresses.items():
        try:
            conn_handler = ConnectHandler(host=ssh_ip,
                                           port=router.ssh_information.port,
                                           username=user.username,
                                           password=user.ssh_password,
                                           secret=router.enable_password,
                                           device_type=router.type
                                           )

            except TimeoutError as e:
                continue
            return conn_handler
        raise Exception("Cannot connect to router")
```

Funkcja końcowa

```
def close_connection(connection: netmiko.BaseConnection) -> None:
    connection.disconnect()
```

Context Manager

Docelowa funkcja

```
def execute_conf_commands(router: Router, user: User, commands: str | list[str],
                           connection: netmiko.BaseConnection | None = None) -> str:
    is_connection = False if connection is None else True
    if not is_connection:
        connection = create_connection_to_router(router, user)
        connection.enable()

    output: str = connection.send_config_set(commands)

    if not is_connection:
        close_connection(connection)

    output: list = output.splitlines()[2:-2]
    output.insert(0, f'Configuring {router.name}')
    output.append(f'End')

    return '\n'.join(output)
```

Context Manager

Realizacja context manager z wykorzystaniem klasy

```
class Connection:
    # Jakubkkk12
    def __init__(self, router: Router, user: User) -> None:
        for key, ssh_ip in router.ssh_information.ip_addresses.items():
            try:
                conn_handler: ConnectHandler = ConnectHandler(host=ssh_ip,
                                                                port=router.ssh_information.port,
                                                                username=user.username,
                                                                password=user.ssh_password,
                                                                secret=router.enable_password,
                                                                device_type=router.type)

                conn_handler.enable()
            except Exception:
                continue
            self.connection = conn_handler
            return None
        raise Exception("Cannot connect to router")

    # Jakubkkk12
    def __enter__(self) -> ConnectHandler:
        return self.connection

    # Jakubkkk12
    def __exit__(self, exc_type, exc_value, traceback) -> None:
        self.connection.disconnect()
        return None
```


Context Manager

Realizacja context manager z wykorzystaniem funkcji i dekoratora @contextmanager

```
@contextmanager
def open_connection(router: Router, user: User):
    connection = None
    for key, ssh_ip in router.ssh_information.ip_addresses.items():
        try:
            conn_handler: ConnectHandler = ConnectHandler(host=ssh_ip,
                                                            port=router.ssh_information.port,
                                                            username=user.username,
                                                            password=user.ssh_password,
                                                            secret=router.enable_password,
                                                            device_type=router.type)

            conn_handler.enable()
        except Exception:
            continue
    connection = conn_handler
    break
    if connection is None:
        raise Exception("Cannot connect to router")

    try:
        yield connection
    finally:
        connection.disconnect()
```

Context Manager

Docelowa funkcja

```
def execute_conf_commands(router: Router, user: User, commands: str | list[str],
                           connection: netmiko.BaseConnection | None = None) -> str:
    is_connection = False if connection is None else True
    if is_connection:
        output: list = connection.send_config_set(commands).splitlines()[2:-2]
    else:
        with open_connection(router=router, user=user) as connection:
            output: list = connection.send_config_set(commands).splitlines()[2:-2]

    output.insert(__index: 0, __object: f'Configuring {router.name}')
    output.append(f'End')
    return '\n'.join(output)
```

Wyrażenia Regularne

Wyrażenia regularne pomagają znajdować dane wzory (pattern) w fragmencie tekstu.

Standardowym zastosowaniem to sprawdzenie adresu emaila

```
# Wzorzec regularny na email
email_regex = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
email = 'jakbol234@gmail.com'

# Funkcja do sprawdzenia, czy email jest poprawny
if re.search(email_regex, email):
    # Jest poprawny
```

Wyrażenia Regularne

Elementy wyrażen regularnych

```
# Znaki liter i cyfr
re.search(pattern: r'a', string: 'cat') # Wynik: a
# Kropka (.) Pasuje do dowolnego znaku (z wyjątkiem nowej linii).
re.search(pattern: r'c.t', string: 'cat') # Wynik: cat
# Gwiazdka (*) Pasuje do 0 lub więcej wystąpień poprzedzającego znaku
re.search(pattern: r'ac*', string: 'zacccc') # Wynik: acccc
# Plus (+) Pasuje do 1 lub więcej wystąpień poprzedzającego znaku
re.search(pattern: r'ac+', string: 'zaccc') # Wynik: accc
# Znak zapytania (?) Pasuje do 0 lub 1 wystąpienia poprzedzającego znaku.
re.search(pattern: r'ac?', string: 'zaccc') # Wynik: ac
# Klasy znaków ([]) Pasuje do dowolnego znaku w nawiasach.
re.search(pattern: r'[adg]', string: 'zaccc') # Wynik: a
re.search(pattern: r'[a-z]', string: 'zaccc') # Wynik: z
# Negacja (^) Pasuje do dowolnego znaku, który nie znajduje się w nawiasach.
re.search(pattern: r'[ ^a-z]', string: 'zaCcc') # Wynik: C
# Kotwica (^) Pasuje na początku stringa.
re.search(pattern: r'^pies', string: 'pies i owca') # Wynik: pies
# Kotwica ($) Pasuje na końcu stringa.
re.search(pattern: r'owca$', string: 'pies i owca') # Wynik: owca
# Alternatywa (|) Pasuje do jednego z kilku wzorców
re.search(pattern: r'owca|pies', string: 'pies i owca') # Wynik: pies
# Dowolna liczba wystąpień ({min,max}) Pozwala na określenie minimalnej i maksymalnej liczby wystąpień.
re.search(pattern: r'a{2,3}', string: 'zaaaccc') # Wynik: aaa
```

Wyrażenia Regularne

Elementy wyrażeń regularnych

```
# \d dowolna cyfra
# \s dowolny znak biały
# \w dowolny znak alfanumeryczny i _ (podłoga)
# \D dowolny znak nie cyfry
# \S dowolny znak nie biały
# \W dowolny znak nie alfanumeryczny i _ (podłoga)
```

Funkcje z biblioteki re

```
# Przeszukuje cały string w poszukiwaniu pierwszego wystąpienia wzorca.
re.search(pattern, string, flags=0)
re.search(pattern: r'\d+', string: 'Numer to 123.') # 123
# Sprawdza, czy wzorzec pasuje na początku stringa
re.match(pattern, string, flags=0)
re.match(pattern: r'\d+', string: '188 to twój numer.') # 188
# Sprawdza, czy wzorzec pasuje do całego stringa
re.fullmatch(pattern, string, flags=0)
re.fullmatch(pattern: r'\d+', string: '1453') # 1453
```


Wyrażenia Regularne

Funkcje z biblioteki re

```
# Zwraca wszystkie pokrywające się wystąpienia wzorca jako listę
re.findall(pattern, string, flags=0)
re.findall(pattern: r'\d+', string: '123 i 456 to numery') # ['123', '456']
# Zwraca iterator z wszystkimi pokrywającymi się wystąpieniami wzorca.
re.finditer(pattern, string, flags=0)
# Dzieli string na podstawie wzorca.
re.split(pattern, string, maxsplit=0, flags=0)
re.split(pattern: r'\s+', string: 'To\tjest to') # ['To', 'jest', 'to']
# Zastępuje wystąpienia wzorca w stringu podaną wartością
re.sub(pattern, repl, string, count=0, flags=0)
re.sub(pattern: r'\d+', repl: '@', string: 'Numery: 12 i 23') # 'Numery: @ i @'
```

Flagi

```
re.IGNORECASE # Ignoruje wielkość liter.
re.MULTILINE # Zmienia zachowanie ^ i $, aby odpowiadały na początku i końcu każdej linii.
re.DOTALL # Zmienia zachowanie . tak, aby pasował także do nowej linii.
re.NOFLAG # Brak flagi wartość 0
```

Wyrażenia Regularne

Obiekt Match - przykład

```
import re
text = """
Witaj jakub@gmail.com!
Przesyłam listę emaili zgodnie z prośbą:
asd@gmail.com
Janek.kol@gmail.com
OgalB45@interia.pl
Pozdrawiam aga67.sk@goli.com
"""

email_pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
matches = re.finditer(email_pattern, text, flags=re.MULTILINE)
print("Przesłane emaile w wiadomości:")
for match in matches:
    print(f>Email: {match.group()} na pozycji {match.start()}-{match.end()}")

# match.group() zwraca znaleziony ciąg odpowiadający wzorcowi
# match.start() zwraca początkowy index znalezionego ciągu
# match.end() zwraca końcowy index znalezionego ciągu
```

Przesłane emaile w wiadomości:

Email: asd@gmail.com na pozycji 65-78

Email: Janek.kol@gmail.com na pozycji 79-98

Email: OgalB45@interia.pl na pozycji 99-117

Wyrażenia Regularne

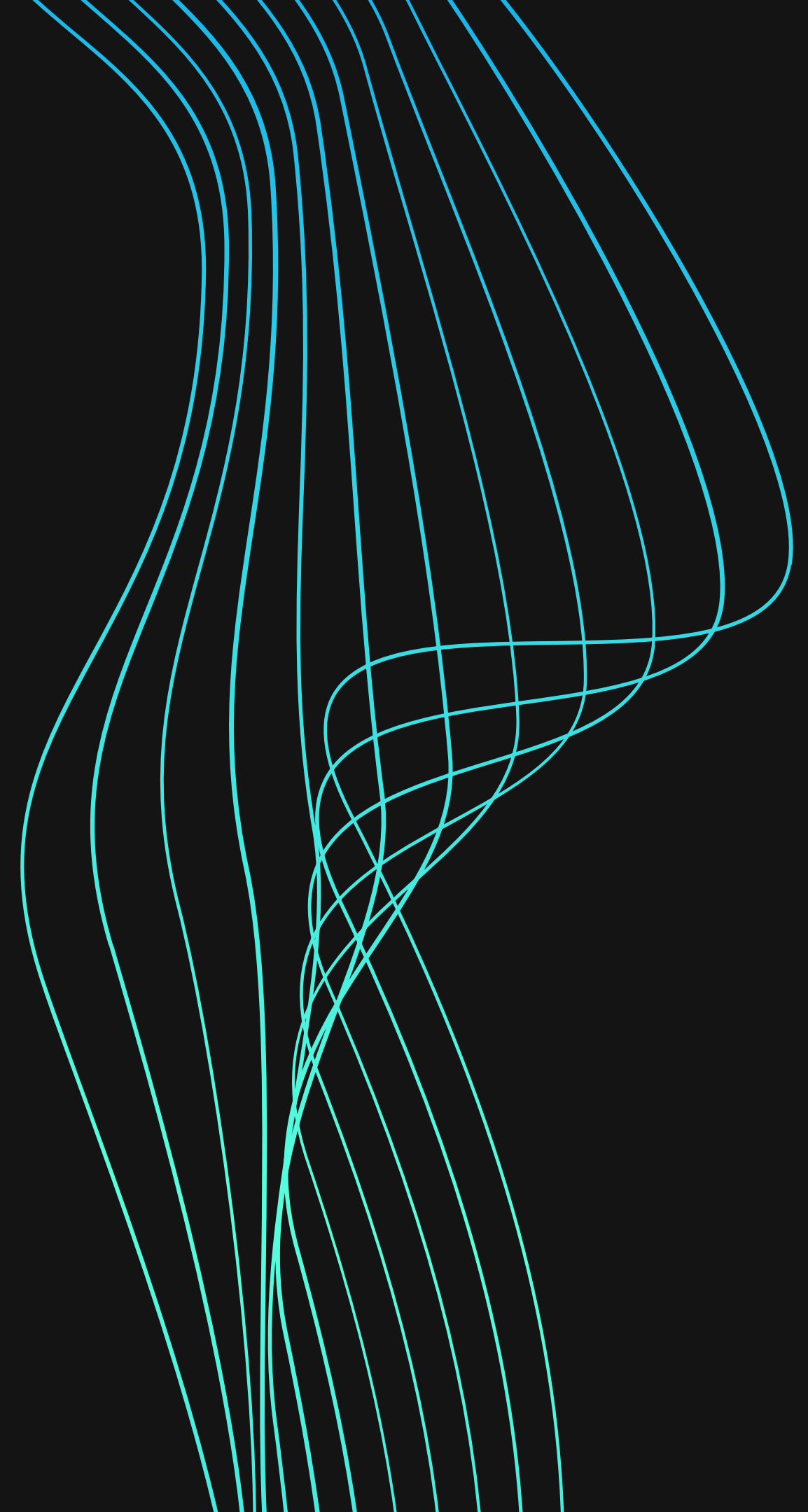
Zastosowane w naszym projekcie

Uzyskanie router_id

```
def get_ospf_router_id(sh_ip_ospf_database_output: str) -> str | None:
    pattern = r'(OSPF Router with ID ).*(Pro)'
    match = re.search(pattern, sh_ip_ospf_database_output)
    if match:
        router_id = match.group()[len('OSPF Router with ID ') + 1:-len(' (Pro')]
        return router_id
    return None
```

Uzyskanie ilości runts

```
def get_interface_statistics_runts(sh_int_name_output: str) -> int | None:
    pattern = r'\d*( runs)'
    match = re.search(pattern, sh_int_name_output)
    if match:
        runts = int(match.group()[:-len(' runs')])
        return runts
    return None
```



Wyrażenia Regularne

Zastosowane w naszym projekcie

Sprawdzenie next-hop-self dla sąsiada w BGP

```
def get_bgp_neighbor_next_hop_self(neighbor: str, sh_run_sec_bgp_output: str) -> bool:
    pattern = f'(neighbor {neighbor} next-hop-self)'
    if re.search(pattern, sh_run_sec_bgp_output):
        return True
    return False
```

Uzyskanie sieci
rozgłaszanych w BGP

```
def get_bgp_networks(sh_run_sec_bgp_output: str) -> dict[str, Network] | None:
    pattern = r'(network .*)'
    match = re.findall(pattern, sh_run_sec_bgp_output)
    if not match:
        return {}
    # 'network 11.0.0.0 mask 255.255.255.0'
    # 'network 11.0.0.0'
    networks_list: list[list[str]] = [[line.split(' ')[1], line.split(' ')[-1]] for line in match]
    networks: dict[str, Network] = {}
    for net in networks_list:
        if net[0] == net[1]:
            f_octet: int = int(net[0].split('.')[0])
            if 0 < f_octet <= 127:
                net[1] = '255.0.0.0'
            elif 128 < f_octet <= 191:
                net[1] = '255.255.0.0'
            else:
                net[1] = '255.255.255.0'

        networks[f"{net[0]} {net[1]}"] = Network(network=net[0],
                                                mask=NETWORK_MASK_REVERSED[net[1]])

    return networks
```

Prezentacja Projektu



GITHUB

<https://github.com/Jakubkkk12/JPWP>

Zadania

ZADANIE 1 (FRONTEND - AL)

Zrefaktoruj kod tak aby zrobić jedno uniwersalne okienko RedistributionGUI (zamiast BGPRedistributionGUI, OSPFRedistributionGUI i RIPRedistributionGUI) i móc je zaaplikować do wszystkich protokołów routingu.

ZADANIE 2 (BACKEND - JK)

Stosując przyjętą w projekcie konwencję napisz odpowiednie funkcje umożliwiające obsłużenie dodania AREA w OSPF. Popatrz na `ospf_area_add_gui.py` aby zobaczyć jakie dane są zwracane z okienka. Prawidłowo rozwiązane zadanie powinno składać się z dopisanych funkcji w: `frontend_backend_functions.py`, `universal_router_commands.py`, `commands.py` oraz `getting_ospf.py`.

ZADANIE 3 (FRONTEND+BACKEND - AL+JK)

Stosując Tkinter oraz przyjętą w projekcie konwencją na backend zaimplementuj możliwość stworzenia loopback interface (na urządzeniach cisco)

Źródła:

<https://docs.python.org/3/reference/>

https://book.pythontips.com/en/latest/context_managers.html

https://pyneng.readthedocs.io/en/latest/book/18_ssh_telnet/netmiko.html

<https://pypi.org/project/netmiko/1.4.1/>

https://ggoralski.gitlab.io/python-wprowadzenie/czesc_ii/2_01-regex/

<https://docs.python.org/3/library/re.html#>

<https://chatgpt.com/c/fc29caa6-ed56-4bcb-a1f9-1406d5800ab2>