**Project topic:** The project is about network automation script that can be used to gather information from any network devices for reporting task. As a Network Engineer by profession, it is part of my responsibility to generate a report that will be presented to the business to have an overview of the network. The project will talk about the purpose of the automation and how the business and user will benefit from it.

**Technology stack:** Python language

**Functional Requirements**

1. Device Discovery and Inventory Management:

It is capable to gather information like IP addresses, hostnames, device types, and relevant attributes.

to maintain an inventory or for reporting.

1. Configuration Management:

Ability to push configuration changes to network devices.

Support for both individual device configuration and bulk configuration changes.

Ability to rollback configuration changes in case of errors or failures.

1. Monitoring and Alerts:

Real-time monitoring of network device status and performance metrics.

Generation of alerts for predefined threshold violations or network issues.

Integration with existing monitoring systems for alert notifications.

1. Security and Access Control:

Secure authentication and authorization mechanisms for accessing network devices.

Role-based access control (RBAC) to restrict access based on user roles and permissions.

Encryption of sensitive data transmitted between the script and network devices.

1. Automated Tasks and Workflows:

Support for automating routine network tasks such as device provisioning, backups, and firmware upgrades.

Customizable workflows for specific network operations or maintenance activities.

1. Error Handling and Logging:

Comprehensive error handling mechanisms to handle exceptions and unexpected behavior gracefully.

Logging of all script activities, including device interactions, configuration changes, and errors, for troubleshooting and audit purposes.

1. Scalability and Performance:

High scalability to support large-scale network environments with thousands of devices.

Optimization for performance to minimize execution time and resource utilization.

1. Documentation and Reporting:

Automated generation of documentation for network configurations, topology diagrams, and change management records.

Reporting capabilities for analyzing network performance trends, compliance status, and operational metrics.

1. Testing and Validation:

Built-in testing and validation mechanisms to ensure the correctness and reliability of configuration changes.

Support for pre-deployment and post-deployment validation checks to verify the impact of configuration changes on network behavior.

These functional requirements provide a foundation for developing a robust and versatile network automation Python script that meets the needs of modern network environments.

**Non-Functional Requirements**

1. Performance:

The script should be able to handle a large number of network devices efficiently without significant performance degradation.

Response times for device interactions, such as configuration changes or status queries, should be minimal to ensure timely network management.

1. Reliability:

The script should be highly reliable, with minimal downtime or failures during routine operations.

It should include mechanisms for error detection, recovery, and graceful degradation to maintain network availability and stability.

1. Scalability:

The script should scale seamlessly to accommodate growth in the number of network devices and the complexity of network environments.

It should support distributed architectures and parallel processing to handle increased workload without bottlenecks.

1. Security:

Strong security measures should be implemented to protect sensitive network data and credentials stored or transmitted by the script.

Compliance with industry standards and best practices for network security, including encryption, authentication, and access control, should be ensured.

1. Maintainability:

The script should be well-documented, with clear and concise code that is easy to understand, modify, and maintain.

1. Flexibility and Extensibility:

The script should be flexible enough to support a variety of network devices, vendors, and protocols without requiring extensive modifications.

1. Compatibility:

The script should be compatible with a wide range of operating systems, Python versions, and third-party libraries commonly used in network automation.

Compatibility with industry-standard network protocols and technologies should be ensured to facilitate interoperability with existing network infrastructure.

1. Performance Monitoring and Logging:

Comprehensive logging and performance monitoring capabilities should be built into the script to track execution times, resource utilization, and operational metrics.

Integration with monitoring systems or log management platforms should be supported for centralized analysis and reporting of script performance.

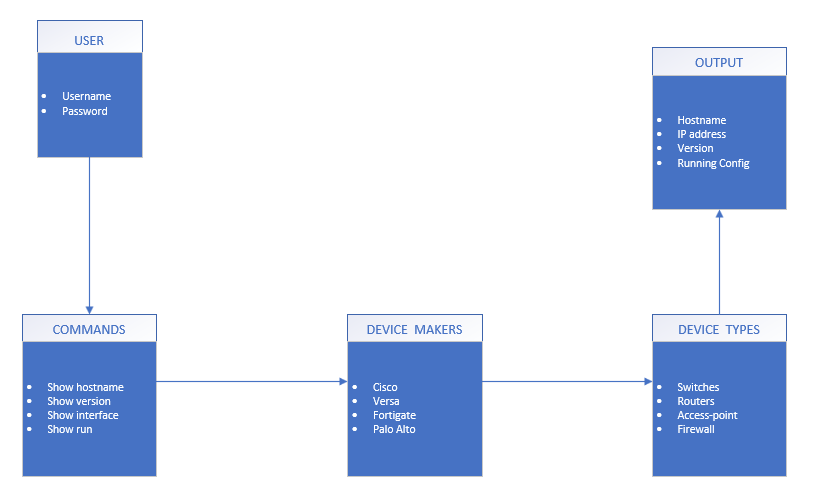
1. Compliance and Regulatory Requirements:

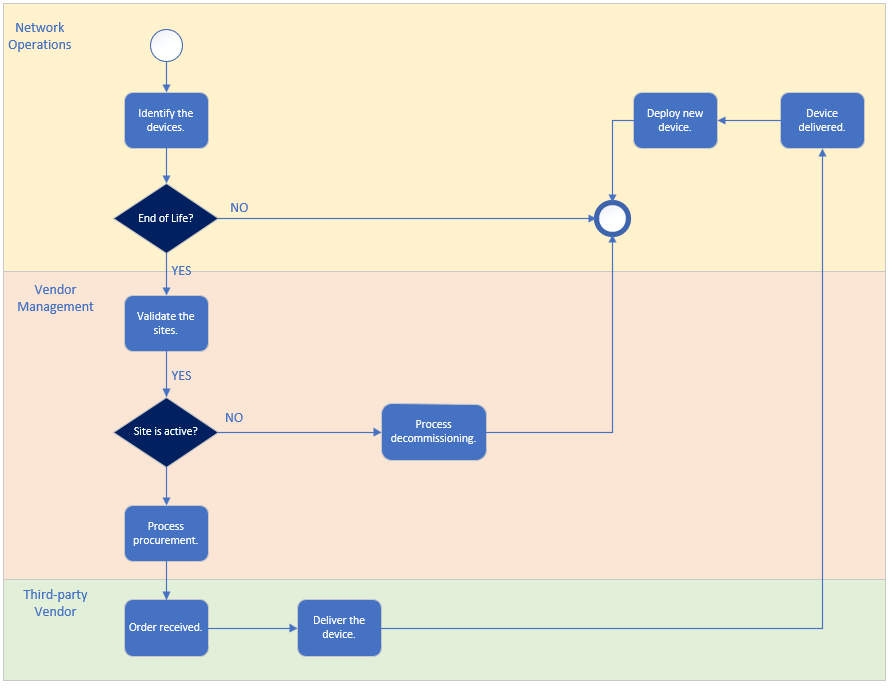
The script should comply with relevant regulatory requirements, industry standards, and organizational policies governing network management and data privacy.

Regular audits and compliance checks should be conducted to ensure adherence to established guidelines and frameworks.

These non-functional requirements are essential for ensuring the effectiveness, reliability, and security of a network automation Python script in real-world deployment scenarios.

**Diagrams**





**Testing Methods and Approaches**

For this case I used Black box testing. Black box testing is a software testing technique where the tester examines the functionality of an application without knowing its internal code structure. Below was my guide for this kind of testing.

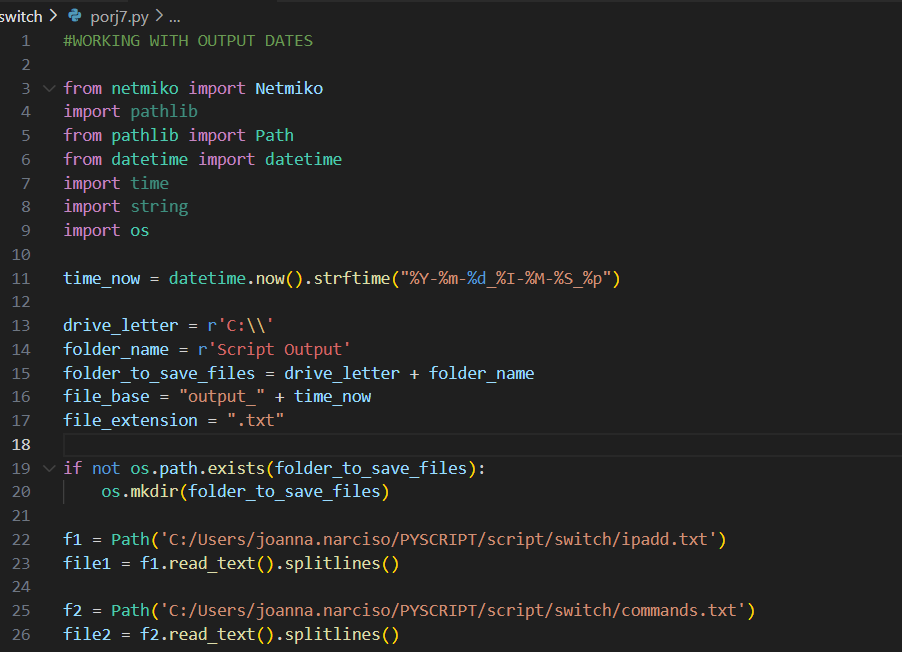
1. Understand Requirements: Gather detailed requirements of the software application or system. Understand what the software is supposed to do and its expected behavior.
2. Identify Test Scenarios: Based on the requirements, identify test scenarios. These are specific functionalities or features of the software that need to be tested.
3. Create Test Cases: Write test cases to cover each identified test scenario. Test cases should include input data, expected outputs, and any other necessary conditions for testing.
4. Design Test Data: Design test data to be used during testing. This includes both valid and invalid input data to evaluate how the software handles different situations.
5. Execute Test Cases: Execute the test cases against the software without any knowledge of its internal workings. This involves interacting with the software through its user interface or APIs, depending on the type of application.
6. Record Results: Record the actual results of each test case, including any deviations from expected behavior or errors encountered during testing.

Analyze Results: Analyze the test results to identify defects or areas of improvement in the software. Document any issues found during testing.

1. Report Defects: Report any defects or issues found during testing to the development team. Provide detailed information about the problem, including steps to reproduce it and any relevant test data.
2. Regression Testing: After fixes are made to address reported defects, perform regression testing to ensure that the changes did not introduce any new issues.
3. Iterate: Repeat the testing process as needed, incorporating feedback from stakeholders and making adjustments to test cases as necessary.



**Actual Script**

****

**Unit Tests**

*PS C:\Users\joanna.narciso\PYSCRIPT\script> cd c:/Users/joanna.narciso/PYSCRIPT/script/switch*

*PS C:\Users\joanna.narciso\PYSCRIPT\script\switch> & "C:/Program Files/Python310/python.exe" c:/Users/joanna.narciso/PYSCRIPT/script/switch/porj7.py*

*CONNECTING TO 10.142.70.17*

*\*\*\* Sending show ip interface brief \*\*\**

*Interface IP-Address OK? Method Status Protocol*

*Vlan1 unassigned YES NVRAM administratively down down*

*Vlan70 10.142.70.17 YES NVRAM up up*

*GigabitEthernet0/0 unassigned YES NVRAM administratively down down*

*GigabitEthernet1/0/1 unassigned YES unset down down*

*GigabitEthernet1/0/2 unassigned YES unset up up*

*GigabitEthernet1/0/3 unassigned YES unset down down*

*GigabitEthernet1/0/4 unassigned YES unset down down*

*GigabitEthernet1/0/5 unassigned YES unset down down*

*GigabitEthernet1/0/6 unassigned YES unset down down*

*GigabitEthernet1/0/7 unassigned YES unset up up*

*GigabitEthernet1/0/8 unassigned YES unset down down*

*GigabitEthernet1/0/9 unassigned YES unset down down*

*GigabitEthernet1/0/10 unassigned YES unset down down*

*GigabitEthernet1/0/11 unassigned YES unset down down*

*GigabitEthernet1/0/12 unassigned YES unset down down*

*GigabitEthernet1/0/13 unassigned YES unset down down*

*GigabitEthernet1/0/14 unassigned YES unset down down*

*GigabitEthernet1/0/15 unassigned YES unset up up*

*GigabitEthernet1/0/16 unassigned YES unset up up*

*GigabitEthernet1/0/17 unassigned YES unset down down*

*GigabitEthernet1/0/18 unassigned YES unset down down*

*GigabitEthernet1/0/19 unassigned YES unset up up*

*GigabitEthernet1/0/20 unassigned YES unset down down*

*GigabitEthernet1/0/21 unassigned YES unset down down*

*GigabitEthernet1/0/22 unassigned YES unset up up*

*GigabitEthernet1/0/23 unassigned YES unset down down*

*GigabitEthernet1/0/24 unassigned YES unset down down*

*GigabitEthernet1/1/1 unassigned YES unset down down*

*GigabitEthernet1/1/2 unassigned YES unset down down*

*GigabitEthernet1/1/3 unassigned YES unset down down*

*GigabitEthernet1/1/4 unassigned YES unset down down*

*Te1/1/1 unassigned YES unset up up*

*Te1/1/2 unassigned YES unset up up*

*Te1/1/3 unassigned YES unset down down*

*Te1/1/4 unassigned YES unset down down*

*\*\*\* Sending show clock \*\*\**

*11:51:01.150 UTC Wed Jun 5 2024*

*CONNECTING TO 10.142.70.19*

*\*\*\* Sending show ip interface brief \*\*\**

*Interface IP-Address OK? Method Status Protocol*

*Vlan1 unassigned YES NVRAM administratively down down*

*Vlan70 10.142.70.19 YES NVRAM up up*

*GigabitEthernet0/0 unassigned YES NVRAM administratively down down*

*GigabitEthernet1/0/1 unassigned YES unset down down*

*GigabitEthernet1/0/2 unassigned YES unset down down*

*GigabitEthernet1/0/3 unassigned YES unset up up*

*GigabitEthernet1/0/4 unassigned YES unset down down*

*GigabitEthernet1/0/5 unassigned YES unset up up*

*GigabitEthernet1/0/6 unassigned YES unset down down*

*GigabitEthernet1/0/7 unassigned YES unset down down*

*GigabitEthernet1/0/8 unassigned YES unset up up*

*GigabitEthernet1/0/9 unassigned YES unset up up*

*GigabitEthernet1/0/10 unassigned YES unset down down*

*GigabitEthernet1/0/11 unassigned YES unset down down*

*GigabitEthernet1/0/12 unassigned YES unset up up*

*GigabitEthernet1/0/13 unassigned YES unset down down*

*GigabitEthernet1/0/14 unassigned YES unset down down*

*GigabitEthernet1/0/15 unassigned YES unset up up*

*GigabitEthernet1/0/16 unassigned YES unset down down*

*GigabitEthernet1/0/17 unassigned YES unset down down*

*GigabitEthernet1/0/18 unassigned YES unset up up*

*GigabitEthernet1/0/19 unassigned YES unset down down*

*GigabitEthernet1/0/20 unassigned YES unset up up*

*GigabitEthernet1/0/21 unassigned YES unset up up*

*GigabitEthernet1/0/22 unassigned YES unset down down*

*GigabitEthernet1/0/23 unassigned YES unset down down*

*GigabitEthernet1/0/24 unassigned YES unset up up*

*GigabitEthernet1/1/1 unassigned YES unset down down*

*GigabitEthernet1/1/2 unassigned YES unset down down*

*GigabitEthernet1/1/3 unassigned YES unset down down*

*GigabitEthernet1/1/4 unassigned YES unset down down*

*Te1/1/1 unassigned YES unset up up*

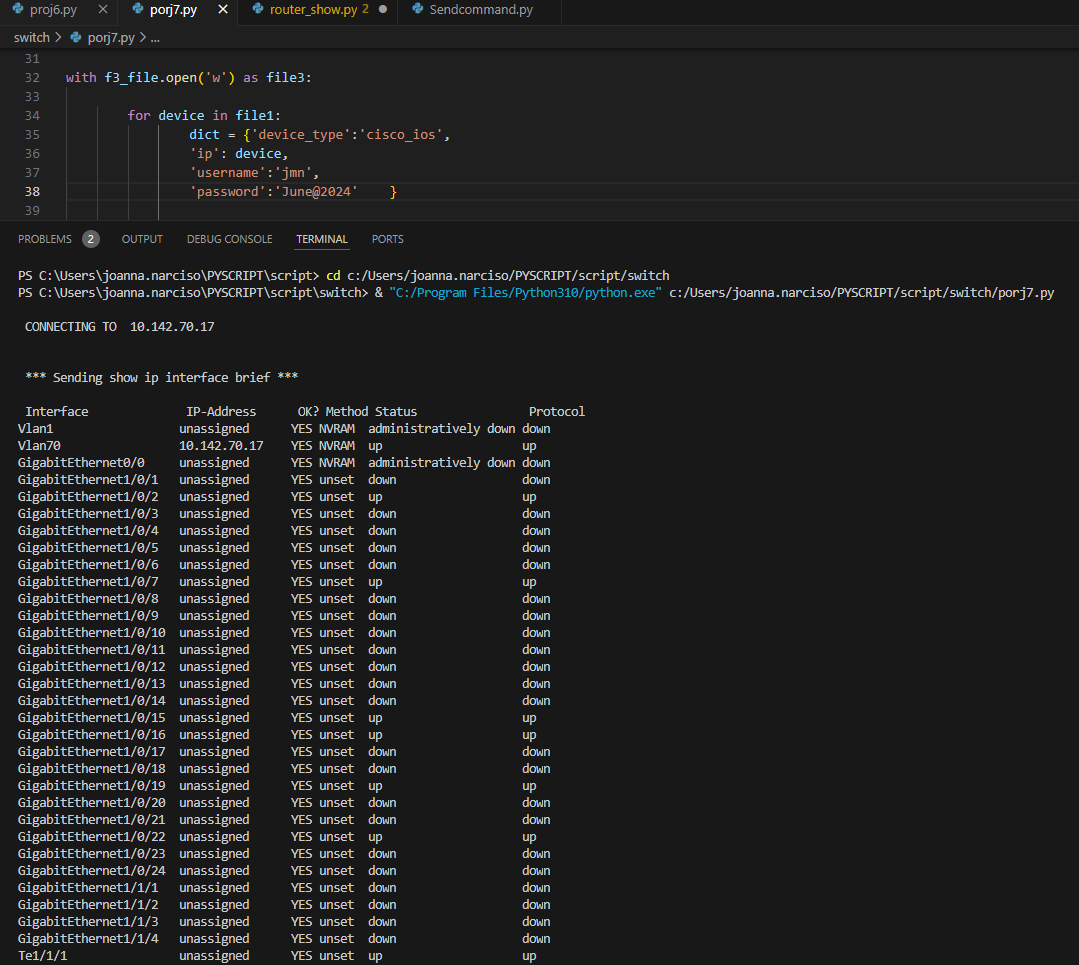
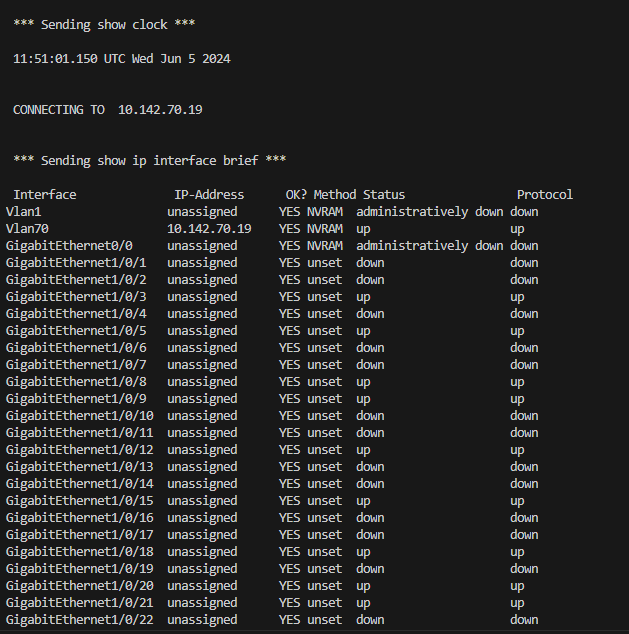
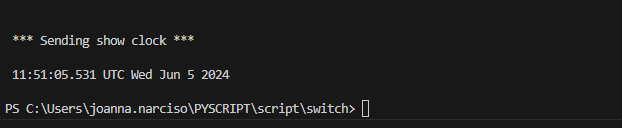
*Te1/1/2 unassigned YES unset up up*

*Te1/1/3 unassigned YES unset down down*

*Te1/1/4 unassigned YES unset down down*

*\*\*\* Sending show clock \*\*\**

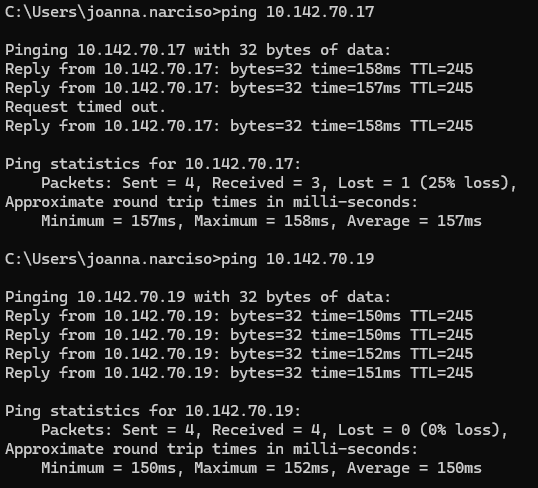
*11:51:05.531 UTC Wed Jun 5 2024*

**** **** 

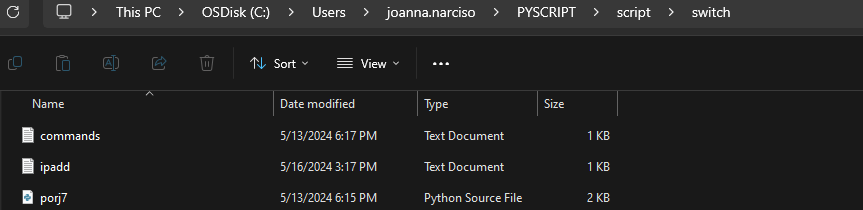


**Implementation of the project**

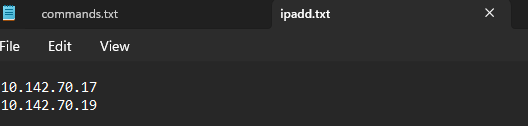
1. Make sure the device that you want to access are reachable.



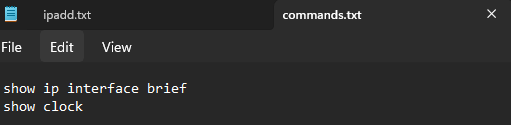
1. Make sure the file path defined in the script and the ipadd and commands text file are the same.

****

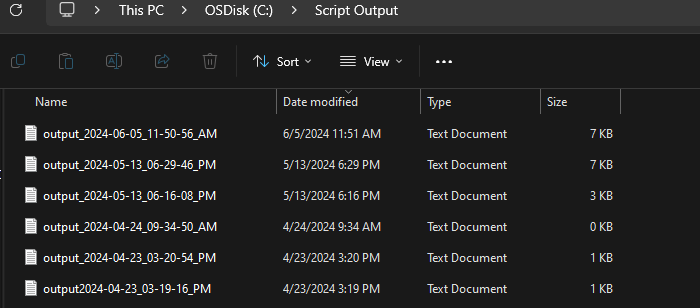
1. Ipadd.text 🡪 list of ip the device you want to access



Commands.text 🡪 list of commands you want to run



1. Make sure the username and password in the script is correct.
2. Run the script.
3. The output will be saved in C drive Script Output folder. If the folder does not exist it will create a new one.



**Technical Documentation of the code**

Python Overview

Python is a clear and powerful object-oriented programming language, comparable to Perl, Ruby, Scheme, or Java.

Some of Python's notable features:

Uses an elegant syntax, making the programs you write easier to read.

Is an easy-to-use language that makes it simple to get your program working. This makes Python ideal for prototype development and other ad-hoc programming tasks, without compromising maintainability.

Comes with a large standard library that supports many common programming tasks such as connecting to web servers, searching text with regular expressions, reading and modifying files.

Python's interactive mode makes it easy to test short snippets of code. There's also a bundled development environment called IDLE.

Is easily extended by adding new modules implemented in a compiled language such as C or C++.

Can also be embedded into an application to provide a programmable interface.

Runs anywhere, including Mac OS X, Windows, Linux, and Unix, with unofficial builds also available for Android and iOS.

Is free software in two senses. It doesn't cost anything to download or use Python, or to include it in your application. Python can also be freely modified and re-distributed because while the language is copyrighted it's available under an open-source license.

Some programming-language features of Python are:

A variety of basic data types are available: numbers (floating point, complex, and unlimited-length long integers), strings (both ASCII and Unicode), lists, and dictionaries.

Python supports object-oriented programming with classes and multiple inheritances.

Code can be grouped into modules and packages.

The language supports raising and catching exceptions, resulting in cleaner error handling.

Data types are strongly and dynamically typed. Mixing incompatible types (e.g. attempting to add a string and a number) causes an exception to be raised, so errors are caught sooner.

Python contains advanced programming features such as generators and list comprehensions.

Python's automatic memory management frees you from having to manually allocate and free memory in your code.

I am using version of 3.10 of Python in my machine.

Summary – Release highlights for the version of 3.10

New syntax features:

PEP 634, Structural Pattern Matching: Specification

PEP 635, Structural Pattern Matching: Motivation and Rationale

PEP 636, Structural Pattern Matching: Tutorial

bpo-12782, Parenthesized context managers are now officially allowed.

New features in the standard library:

PEP 618, Add Optional Length-Checking To zip.

Interpreter improvements:

PEP 626, Precise line numbers for debugging and other tools.

New typing features:

PEP 604, Allow writing union types as X | Y

PEP 612, Parameter Specification Variables

PEP 613, Explicit Type Aliases

PEP 647, User-Defined Type Guards

Important deprecations, removals or restrictions:

PEP 644, Require OpenSSL 1.1.1 or newer

PEP 632, Deprecate distutils module.

PEP 623, Deprecate and prepare for the removal of the wstr member in PyUnicodeObject.

PEP 624, Remove Py\_UNICODE encoder APIs

PEP 597, Add optional EncodingWarning

Documentation of Netmiko Library that was used:

Netmiko

Library Purpose

The purposes of this library are the following:

Successfully establish an SSH connection to the device.

Simplify the execution, retrieval, and formatting of show commands.

Simplify the execution of configuration commands.

Abstract away much of the low-level mechanics of interacting with devices.

Provide a (relatively) uniform API for interacting with devices.

Do the above across a broad set of networking vendors and platforms.

Supported Platforms

Netmiko currently supports about eighty different platforms.

In addition to SSH support, Netmiko also supports Secure Copy, telnet connections, and serial connections. The platform support for each of these is more limited than SSH. Once again see the PLATFORMS.md file for more details on the platforms supported for these use cases.

Getting Started

Import "ConnectHandler" from the Netmiko library. You can think of ConnectHandler as your main entry point into the library. It picks the right class for you, creates a Netmiko object based on that class, and establishes an SSH connection to the remote device.

You would then use ConnectHandler to pick the class and establish the SSH connection.



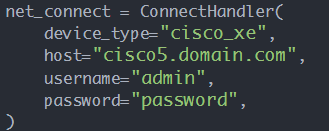
This requires us to pass in certain arguments namely our:

device\_type

host (hostname or IP)

username

password



At this point the variable 'net\_connect' should be a usable SSH connection to the remote device.

