Python Fundamentals

PROJECT: Python OS INFO

Student: David lim

Student code: s7

Python Fundamentals trainer: James

Class code: CFC020823

Introduction to the Project

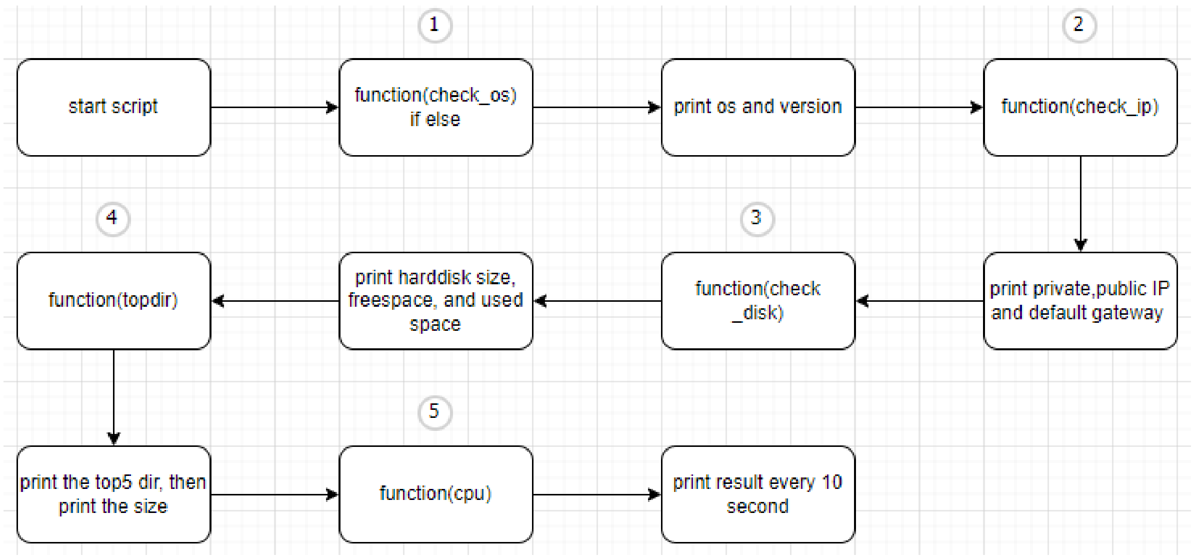
In an era characterized by the relentless pace of technological advancement, the effective management and monitoring of computer systems have become paramount. The complexity of contemporary operating environments demands tools that can swiftly and comprehensively provide vital system information. This project is a response to this need, aiming to create an automation script that can efficiently gather and display key system data.

The project's primary objective is to develop a versatile automation script that operates seamlessly on both Windows and Linux operating systems. This script will encompass five core functionalities:

1. **Operating System Information:** It identifies the operating system in use, providing specific details for Windows and Linux systems.
2. **IP Addresses and Default Gateway:** The script retrieves and displays both private and public IP addresses, along with the default gateway.
3. **Hard Disk Information:** It offers insights into hard disk size, free space, and used space.
4. **Top Directories:** The script lists the top five directories on the system and their respective sizes.
5. **CPU Usage:** It monitors and updates CPU usage in real-time, refreshing every 10 seconds.

This project seeks to streamline and expedite the process of gathering system information, facilitating more efficient system administration and monitoring. Through this report, we will outline the methods employed in the development of the automation script and present the results, demonstrating its practical utility and significance in today's dynamic computing landscape.

Diagram of script


1
2
start script
function(check_os) if else
print os and version
function(check_ip)
4
3
function(topdir)
print harddisk size, freespace, and used space
function(check _disk)
print private,public IP and default gateway
5
print the top5 dir, then print the size
function(cpu)
print result every 10 second

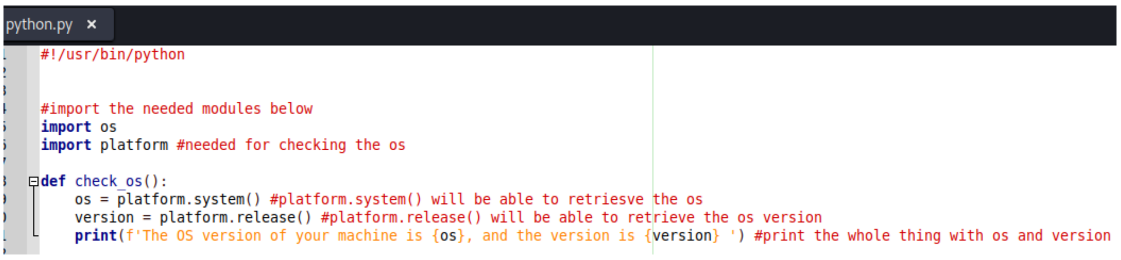
Operating System Information: It identifies the operating system in use, providing specific details for Windows and Linux systems.

import platform module.

The 'platform' module is used for accessing information about the host operating system. In the script, it is employed to determine the current OS and its version.

This function is responsible for identifying the OS and its version. It first utilizes 'platform.system()' to retrieve the name of the OS (e.g., 'Windows', 'Linux', 'Darwin' for macOS). It then uses 'platform.release()' to obtain the OS version. Finally, it prints a message to the console indicating the detected OS and its version.

The printed message is in the format: "The OS version of your machine is {os}, and the version is {version}" where '{os}' is replaced with the actual OS name and '{version}' is replaced with the OS version.



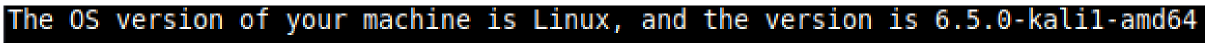
```python
#!/usr/bin/python


#import the needed modules below
import os
import platform #needed for checking the os

def check_os():
    os = platform.system() #platform.system() will be able to retriesve the os
    version = platform.release() #platform.release() will be able to retrieve the os version
    print(f'The OS version of your machine is {os}, and the version is {version} ') #print the whole thing with os and version
```

Figure 1.1 A Screenshot of the check_os():

The script is ran on a linux machine, the script is able to retrieve the OS and the version



```
The OS version of your machine is Linux, and the version is 6.5.0-kali1-amd64
```

Figure 1.2 A Screenshot of the result of check_os():

IP Addresses and Default Gateway: The script retrieves and displays both private and public IP addresses, along with the default gateway.

first import subprocess module.

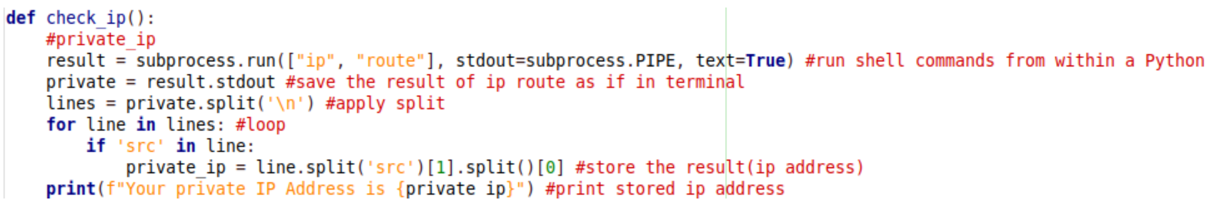
1. For private ip, use subprocess.run(["ip", "route"] because this function runs the ip route command to get information about the system's network routes. It will save the output, then spilt and store as lines (remove the \n). It then searches for a line containing 'src' (which typically indicates the source IP address) and extracts the private IP address from that line. Then print the private IP address.

2. For public ip, use subprocess.run(["curl", "-s", "ifconfig.me"] because this function uses the curl command to query the "ifconfig.me" service, which returns the public IP address of the system. It captures the output of the curl command and returns the public IP address.

3. For the default gateway, use subprocess.run(["ip", "route", "show", "0.0.0.0/0"] because this function run the ip route show 0.0.0.0/0 command to display the default gateway information. It extracts the default gateway IP address from the output and returns it.

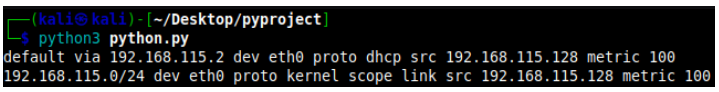
The subprocess module, allows the users to run shell commands from within a Python script.
It defines a function called private_ip that performs the following steps:

1. Use subprocess.run to run ip route command and captures its output (the routing table) as a string.
2. It splits the output into lines.
3. It iterates through each line and checks if the word 'src' is present in the line.
4. If 'src' is found in a line, it splits the line by spaces and extracts the IP address that follows 'src'.
5. Print the private IP address found.



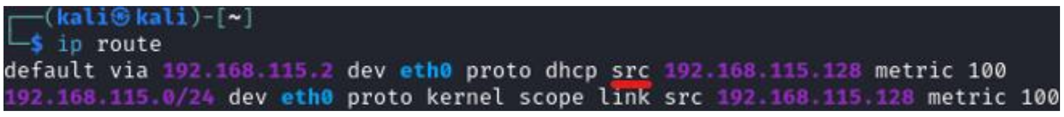
```
def check_ip():
    #private_ip
    result = subprocess.run(["ip", "route"], stdout=subprocess.PIPE, text=True) #run shell commands from within a Python
    private = result.stdout #save the result of ip route as if in terminal
    lines = private.split('\n') #apply split
    for line in lines: #loop
        if 'src' in line:
            private_ip = line.split('src')[1].split()[0] #store the result(ip address)
    print(f"Your private IP Address is {private_ip}") #print stored ip address
```

Figure 2.1 A Screenshot of check_ip():



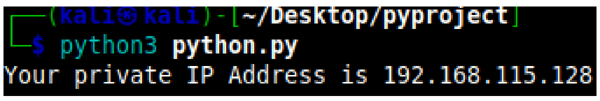
```
┌──(kali㉿kali)-[~/Desktop/pyproject]
└─$ python3 python.py
default via 192.168.115.2 dev eth0 proto dhcp src 192.168.115.128 metric 100
192.168.115.0/24 dev eth0 proto kernel scope link src 192.168.115.128 metric 100
```

Figure 2.2 A Screenshot of result of output



```
┌──(kali㉿kali)-[~]
└─$ ip route
default via 192.168.115.2 dev eth0 proto dhcp src 192.168.115.128 metric 100
192.168.115.0/24 dev eth0 proto kernel scope link src 192.168.115.128 metric 100
```

Figure 2.3 A Screenshot of ip route in terminal highlighting src 192.168.115.128

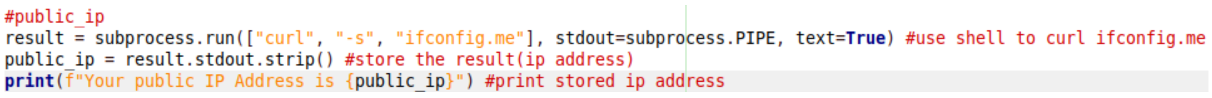


```
┌──(kali㉿kali)-[~/Desktop/pyproject]
└─$ python3 python.py
Your private IP Address is 192.168.115.128
```

Figure 2.4 A Screenshot of result of private IP address

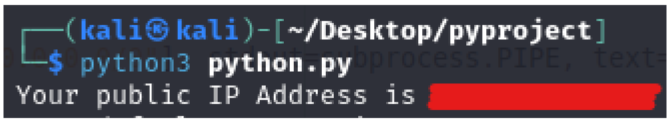
For the public ip address

1. Use subprocess.run to curl -s ifconfig.me and captures its output.
2. Store the result of the output.
3. Print the private IP address found.



```
#public_ip
result = subprocess.run(["curl", "-s", "ifconfig.me"], stdout=subprocess.PIPE, text=True) #use shell to curl ifconfig.me
public_ip = result.stdout.strip() #store the result(ip address)
print(f"Your public IP Address is {public_ip}") #print stored ip address
```

Figure 2.5 A Screenshot of public_ip

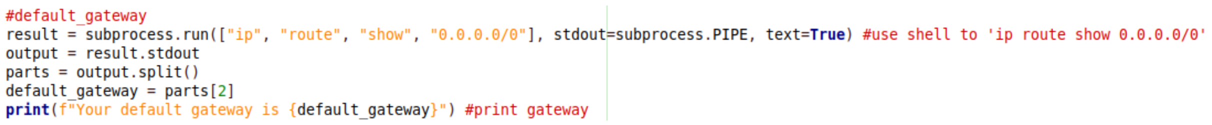


```
┌──(kali㉿kali)-[~/Desktop/pyproject]
└─$ python3 python.py
Your public IP Address is [redacted]
```

Figure 2.6 A Screenshot of print(f"Your public IP Address is {public_ip}")

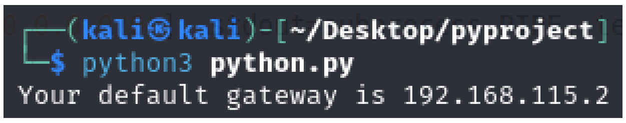
For the default gateway

1. Use subprocess.run to runs ip route show and captures its result.
2. Splits the string on the whitespace.
3. extracts the third element from the parts list
4. Print the gateway.



```
#default_gateway
result = subprocess.run(["ip", "route", "show", "0.0.0.0/0"], stdout=subprocess.PIPE, text=True) #use shell to 'ip route show 0.0.0.0/0'
output = result.stdout
parts = output.split()
default_gateway = parts[2]
print(f"Your default gateway is {default_gateway}") #print gateway
```

Figure 2.7 A Screenshot of default_gateway



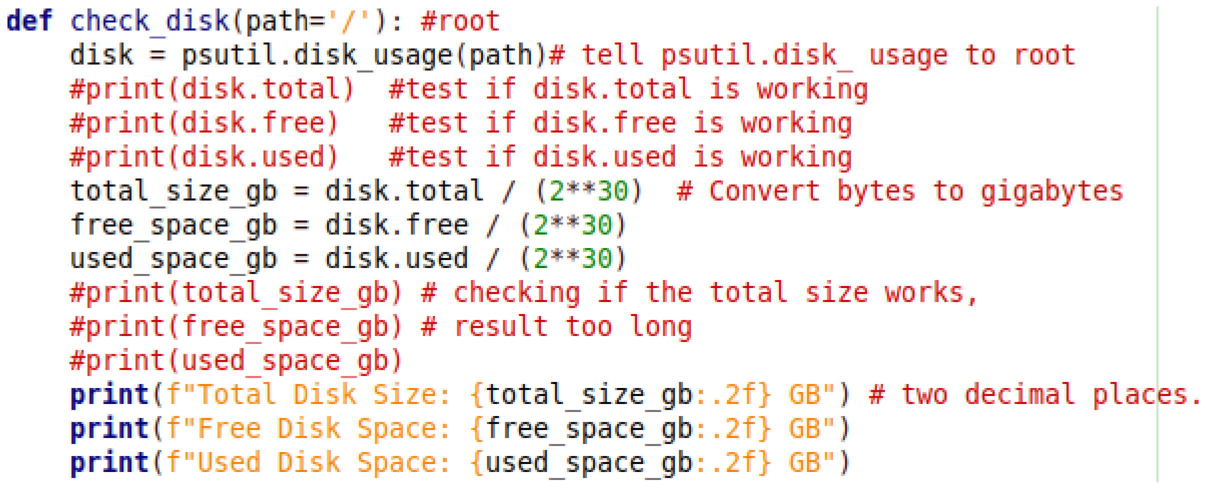
```
┌──(kali㉿kali)-[~/Desktop/pyproject]
└─$ python3 python.py
Your default gateway is 192.168.115.2
```

Figure 2.8 A Screenshot of print(f"Your default gateway is {default_gateway}")

Hard Disk Information: It offers insights into hard disk size, free space, and used space.

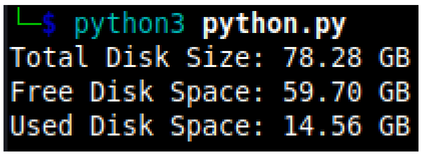
1. First import the psutil module, which is used to gather system information
2. Define a function check_disk() that takes a path parameter (defaulting to the root directory, "/").
3. Inside the function, use psutil.disk_usage to get information about the disk at the specified path.
4. Using disk.total, disk.free, and disk.used to get the size.
5. Calculate the total disk size, free disk space, and used disk space in gigabytes (GB).
6. Print the results with two decimal places for a neater output.

When this script is run, it will display the total disk size, free disk space, and used disk space for the specified path (or the root directory if no path is provided). The sizes are displayed in gigabytes with two decimal places.



```python
def check_disk(path='/'): #root
    disk = psutil.disk_usage(path)# tell psutil.disk_ usage to root
    #print(disk.total)  #test if disk.total is working
    #print(disk.free)   #test if disk.free is working
    #print(disk.used)   #test if disk.used is working
    total_size_gb = disk.total / (2**30)  # Convert bytes to gigabytes
    free_space_gb = disk.free / (2**30)
    used_space_gb = disk.used / (2**30)
    #print(total_size_gb) # checking if the total size works,
    #print(free_space_gb) # result too long
    #print(used_space_gb)
    print(f"Total Disk Size: {total_size_gb:.2f} GB") # two decimal places.
    print(f"Free Disk Space: {free_space_gb:.2f} GB")
    print(f"Used Disk Space: {used_space_gb:.2f} GB")
```

Figure 3.1 A Screenshot of check_disk script

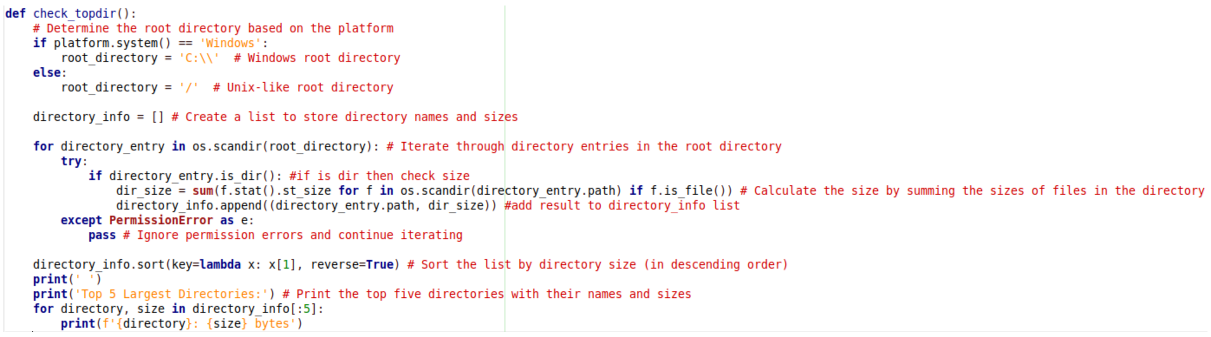


```
└─$ python3 python.py
Total Disk Size: 78.28 GB
Free Disk Space: 59.70 GB
Used Disk Space: 14.56 GB
```

Figure 3.1 A Screenshot of result of total disk size, free disk space and used disk space

Top Directories: The script lists the top five directories on the system and their respective sizes.

1. The script starts by determining the root directory based on the platform. If the platform is Windows, it sets the root directory to 'C:\', and for Unix-like systems, it sets it to '/'.
2. Then create a list called directory_info to store directory names and their sizes.
3. The script then iterates through the directory entries in the root directory using the os.scandir function. For each directory entry, it checks if it's a directory using directory_entry.is_dir().
4. If the entry is a directory, it calculates the size of the directory by summing the sizes of all the files within that directory. This is done using a list comprehension that iterates through the files in the directory and adds up their sizes.
5. The directory path and its size are then appended as a tuple to the directory_info list.
6. If a PermissionError occurs during this process (for example, if the script lacks permission to access a directory), it is caught and ignored, allowing the script to continue processing other directories.
7. After collecting the directory information, the script sorts the directory_info list in descending order based on the directory size.
8. Finally, it prints the top 5 largest directories, displaying the directory name and its size in bytes.



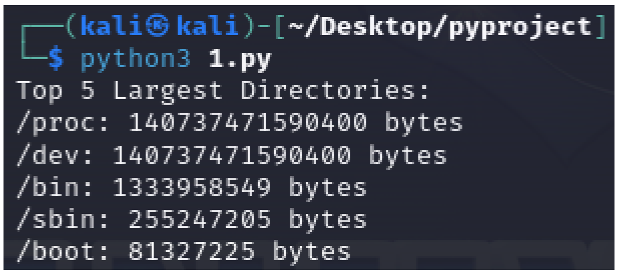
```python
def check_topdir():
    # Determine the root directory based on the platform
    if platform.system() == 'Windows':
        root_directory = 'C:\\'  # Windows root directory
    else:
        root_directory = '/'  # Unix-like root directory

    directory_info = [] # Create a list to store directory names and sizes

    for directory_entry in os.scandir(root_directory): # Iterate through directory entries in the root directory
        try:
            if directory_entry.is_dir(): #if is dir then check size
                dir_size = sum(f.stat().st_size for f in os.scandir(directory_entry.path) if f.is_file()) # Calculate the size by summing the sizes of files in the directory
                directory_info.append((directory_entry.path, dir_size)) #add result to directory_info list
        except PermissionError as e:
            pass # Ignore permission errors and continue iterating

    directory_info.sort(key=lambda x: x[1], reverse=True) # Sort the list by directory size (in descending order)
    print(' ')
    print('Top 5 Largest Directories:') # Print the top five directories with their names and sizes
    for directory, size in directory_info[:5]:
        print(f'{directory}: {size} bytes')
```

Figure 4.1 A Screenshot of check_topdir():

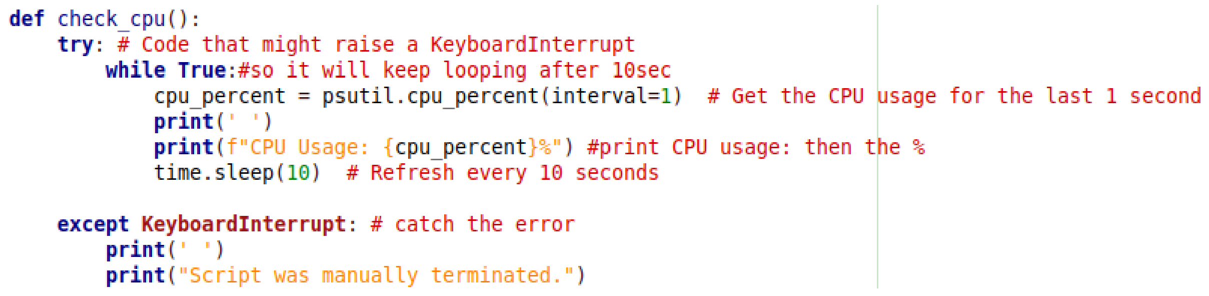


```
┌──(kali㉿kali)-[~/Desktop/pyproject]
└─$ python3 1.py
Top 5 Largest Directories:
/proc: 140737471590400 bytes
/dev: 140737471590400 bytes
/bin: 1333958549 bytes
/sbin: 255247205 bytes
/boot: 81327225 bytes
```

Figure 4.2 A Screenshot of result of check_topdir() with the Top 5 Largest Directories printed

CPU Usage: It monitors and updates CPU usage in real-time, refreshing every 10 seconds.

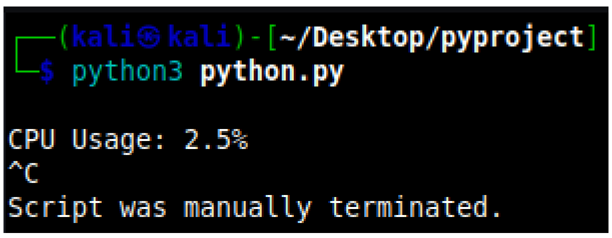
1. import psutil and import time: These lines import the psutil library, which is used to gather system information, and the time library for controlling the timing of the script.
2. the try block is used to handle exceptions. Inside a while True loop, the script continuously monitors CPU usage using psutil.cpu_percent(interval=1). It updates and prints the CPU usage every 10 seconds using time.sleep(10).
3. except KeyboardInterrupt will catches a KeyboardInterrupt exception, which is raised when pressing Ctrl+C. When this exception is caught, it prints a message indicating that the script was manually terminated.



```
def check_cpu():
    try: # Code that might raise a KeyboardInterrupt
        while True:#so it will keep looping after 10sec
            cpu_percent = psutil.cpu_percent(interval=1)  # Get the CPU usage for the last 1 second
            print(' ')
            print(f"CPU Usage: {cpu_percent}%") #print CPU usage: then the %
            time.sleep(10)  # Refresh every 10 seconds

    except KeyboardInterrupt: # catch the error
        print(' ')
        print("Script was manually terminated.")
```

Figure 5.1 A Screenshot of check_cpu:



```
┌──(kali㉿kali)-[~/Desktop/pyproject]
└─$ python3 python.py

CPU Usage: 2.5%
^C
Script was manually terminated.
```

Figure 5.2 A Screenshot of the result of check_cpu: when pressing Ctrl+C

CONCLUSIONS

In the course and this project, I learn the application of Python 3 gained a understanding of its versatility and power as a programming language. This project provided a practical platform for honing my Python skills. I successfully imported and utilized various Python modules, which allowed me to leverage the capabilities of external libraries and tools to enhance my scripts. However, I recognize that further research is needed to fully harness the potential of these modules.

A significant part of the project involved creating automation scripts that performed various tasks, including directory checking, OS version verification, and other automated operations. This hands-on experience enabled me to streamline repetitive processes and improve efficiency.