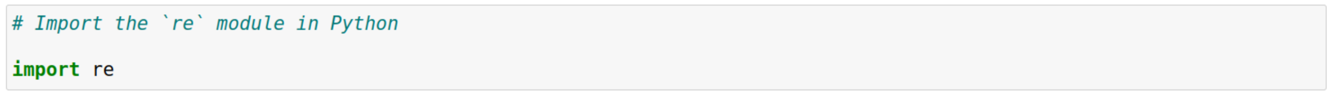
Project Description

In this lab I take the role of a security professional managing vulnerable machines by detecting devices that require updates.

I analyze Python regular expression code used to extract device IDs that match certain characters from a log, these characters correspond to certain operating systems that require an update. I examine the extraction of valid IP addresses from a corrupted log file and then compare those IP addresses to another set of IP addresses flagged in a list.

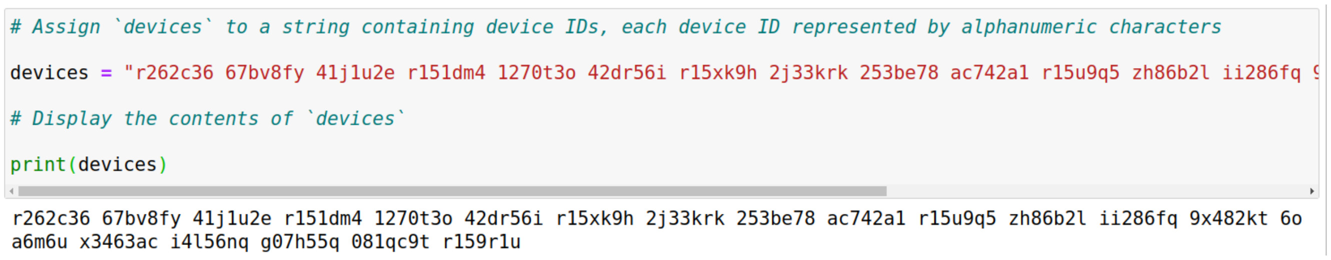
Import the Regular Expressions Module



```python
# Import the `re` module in Python

import re
```

The first step is to make the regular expressions module available in the rest of the lab by importing it.

Examine the Device IDs

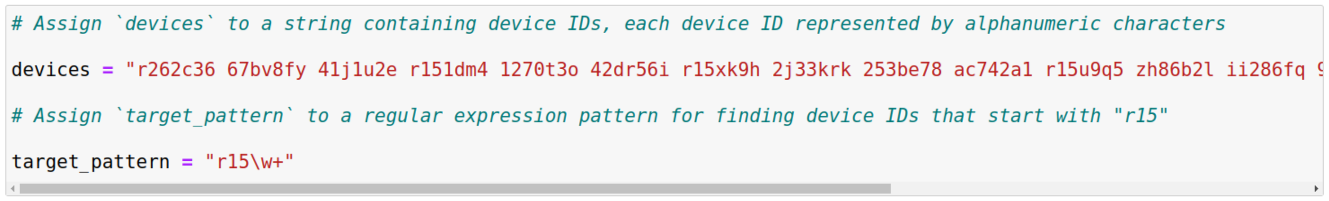


```python
# Assign `devices` to a string containing device IDs, each device ID represented by alphanumeric characters

devices = "r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk 253be78 ac742a1 r15u9q5 zh86b2l ii286fq

# Display the contents of `devices`

print(devices)
```

```
r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk 253be78 ac742a1 r15u9q5 zh86b2l ii286fq 9x482kt 6o
a6m6u x3463ac i4l56nq g07h55q 081qc9t r159r1u
```

I next display the list of device IDs that I am working with.

Task 1: Build a Regular Expression to match Device IDs Needing an Update



```python
# Assign `devices` to a string containing device IDs, each device ID represented by alphanumeric characters

devices = "r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk 253be78 ac742a1 r15u9q5 zh86b2l ii286fq

# Assign `target_pattern` to a regular expression pattern for finding device IDs that start with "r15"

target_pattern = "r15\w+"
```

My goal is to extract device IDs that start with the characters “r15” that indicate the device needs an update.

The regular expression \w matches any alphanumeric character. The + symbol matches one or more occurrences and hence matches alphanumeric characters of any length. Without the \w, the pattern will not match characters after “r15”. Without the +, only the first alphanumeric character after “r15” would match.

Task 1: Extract Matching Device IDs

```
# Assign `devices` to a string containing device IDs, each device ID represented by alphanumeric characters

devices = "r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk 253be78 ac742a1 r15u9q5 zh86b2l ii286fq 9

# Assign `target_pattern` to a regular expression pattern for finding device IDs that start with "r15"

target_pattern = "r15\w+"

# Use `re.findall()` to find the device IDs that start with "r15" and display the results

print(re.findall(target_pattern, devices))
```

```
['r151dm4', 'r15xk9h', 'r15u9q5', 'r159r1u']
```

I use re.findall to match target_pattern against entries in the devices string.

Task 2: Examine a Security Log

```
# Assign `log_file` to a string containing username, date, login time, and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-

# Display contents of `log_file`

print(log_file)
```

```
eraab 2022-05-10 6:03:41 192.168.152.148
iuduike 2022-05-09 6:46:40 192.168.22.115
smartell 2022-05-09 19:30:32 192.168.190.178
arutley 2022-05-12 17:00:59 1923.1689.3.24
rjensen 2022-05-11 0:59:26 192.168.213.128
aestrada 2022-05-09 19:28:12 1924.1680.27.57
asundara 2022-05-11 18:38:07 192.168.96.200
dkot 2022-05-12 10:52:00 1921.168.1283.75
abernard 2022-05-12 23:38:46 19245.168.2345.49
cjackson 2022-05-12 19:36:42 192.168.247.153
jclark 2022-05-10 10:48:02 192.168.174.117
alevitsk 2022-05-08 12:09:10 192.16874.1390.176
jrafael 2022-05-10 22:40:01 192.168.148.115
yappiah 2022-05-12 10:37:22 192.168.103.10654
daquino 2022-05-08 7:02:35 192.168.168.144
```

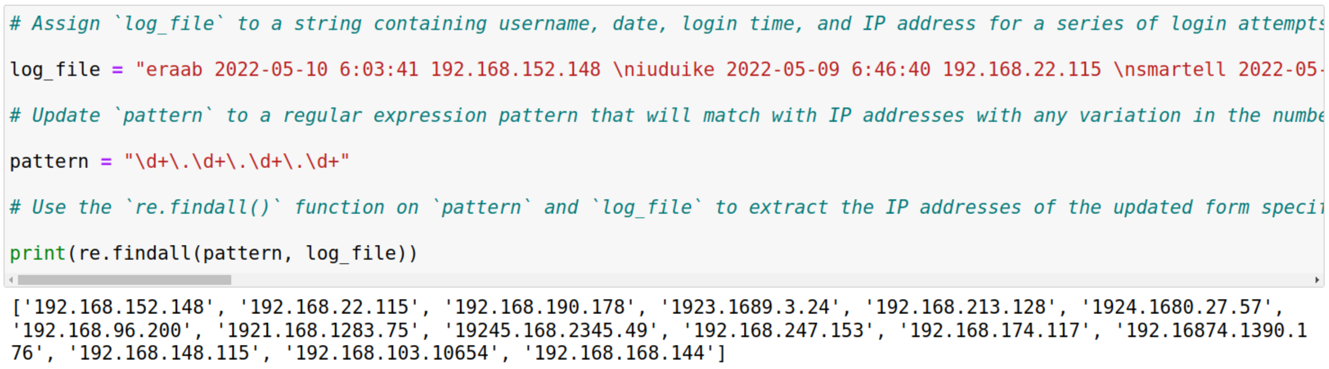
I first print the contents of the security log_file string to see what I am working with.

My goal in this task is to later filter the security log to extract valid data from it, as some of the data in the log is invalid (the IP addresses have four digits in an octet, for example `19245.168.2345.49.`

Task 2: Build a Regular Expression to Match Valid IP Addresses

```
# Assign `log_file` to a string containing username, date, login time, and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-

# Assign `pattern` to a regular expression pattern that will match with IP addresses of the form xxx.xxx.xxx.xxx
pattern = "\d\d\d\.\d\d\d\.\d\d\d\.\d\d\d"
```

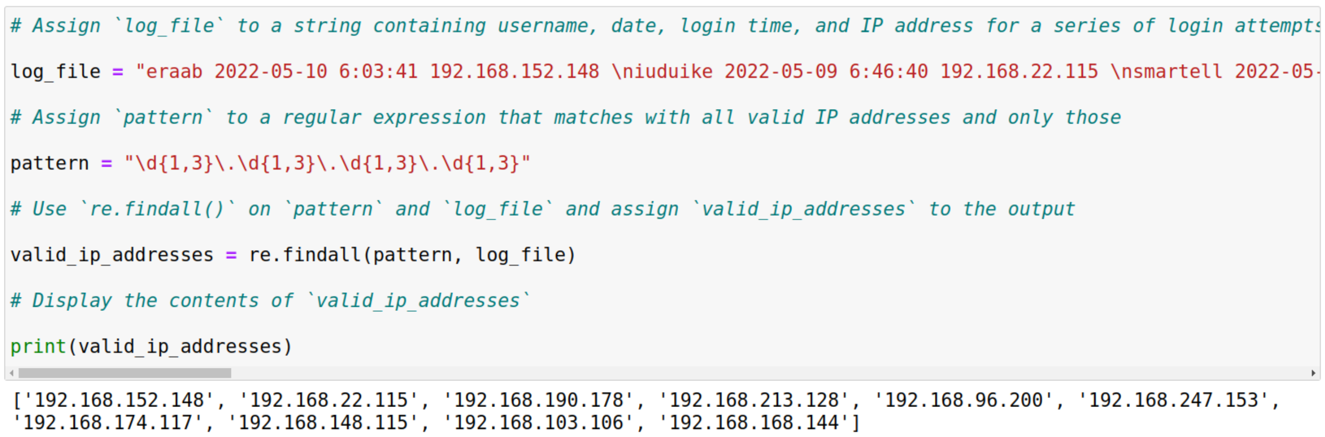
The \d pattern matches a digit, it is repeated three times. Using \d\d\d would only match when there are three digits and fail to match with two digits, for example `1921.168.1283.75.`



```
# Assign `log_file` to a string containing username, date, login time, and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-

# Update `pattern` to a regular expression pattern that will match with IP addresses with any variation in the numbe

pattern = "\d+\.\d+\.\d+\.\d+"

# Use the `re.findall()` function on `pattern` and `log_file` to extract the IP addresses of the updated form speci

print(re.findall(pattern, log_file))
```

```
['192.168.152.148', '192.168.22.115', '192.168.190.178', '1923.1689.3.24', '192.168.213.128', '1924.1680.27.57',
'192.168.96.200', '1921.168.1283.75', '19245.168.2345.49', '192.168.247.153', '192.168.174.117', '192.16874.1390.1
76', '192.168.148.115', '192.168.103.10654', '192.168.168.144']
```

The pattern \d+ is slightly better as it matches two digit octets but unfortunately also captures invalid IP addresses with four digits per octet such as `1921.168.1283.75`

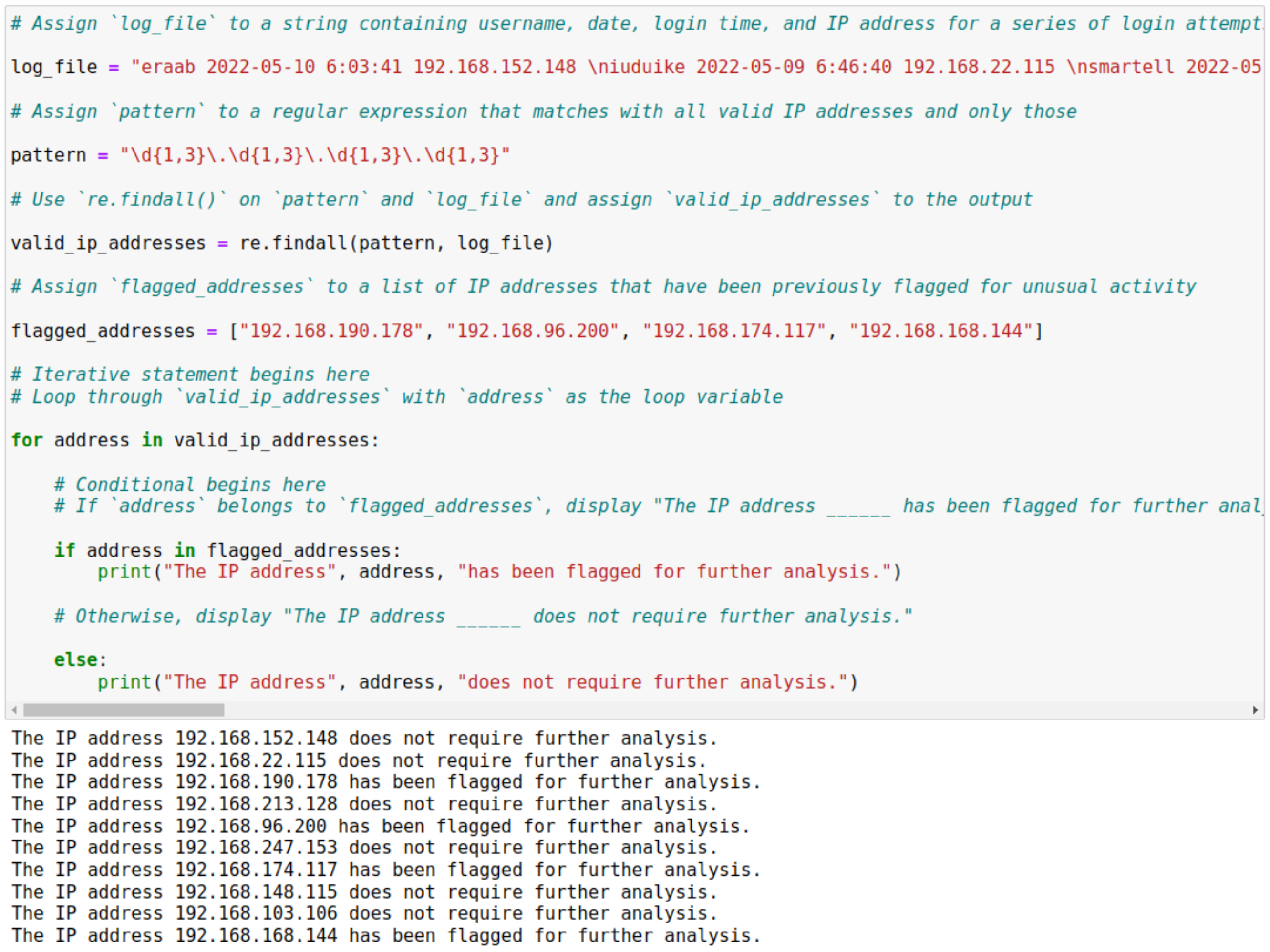


```
# Assign `log_file` to a string containing username, date, login time, and IP address for a series of login attempts

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-

# Assign `pattern` to a regular expression that matches with all valid IP addresses and only those

pattern = "\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"

# Use `re.findall()` on `pattern` and `log_file` and assign `valid_ip_addresses` to the output

valid_ip_addresses = re.findall(pattern, log_file)

# Display the contents of `valid_ip_addresses`

print(valid_ip_addresses)
```

```
['192.168.152.148', '192.168.22.115', '192.168.190.178', '192.168.213.128', '192.168.96.200', '192.168.247.153',
'192.168.174.117', '192.168.148.115', '192.168.103.106', '192.168.168.144']
```

The \d{1,3} pattern captures octets with 1 to 3 digits and the \. pattern matches the IP address octet separator character ".".

Task 3: Flag certain IP addresses for further analysis



```
# Assign `log_file` to a string containing username, date, login time, and IP address for a series of login attempt

log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:46:40 192.168.22.115 \nsmartell 2022-05

# Assign `pattern` to a regular expression that matches with all valid IP addresses and only those

pattern = "\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"

# Use `re.findall()` on `pattern` and `log_file` and assign `valid_ip_addresses` to the output

valid_ip_addresses = re.findall(pattern, log_file)

# Assign `flagged_addresses` to a list of IP addresses that have been previously flagged for unusual activity

flagged_addresses = ["192.168.190.178", "192.168.96.200", "192.168.174.117", "192.168.168.144"]

# Iterative statement begins here
# Loop through `valid_ip_addresses` with `address` as the loop variable

for address in valid_ip_addresses:

    # Conditional begins here
    # If `address` belongs to `flagged_addresses`, display "The IP address ______ has been flagged for further anal

    if address in flagged_addresses:
        print("The IP address", address, "has been flagged for further analysis.")

    # Otherwise, display "The IP address ______ does not require further analysis."

    else:
        print("The IP address", address, "does not require further analysis.")
```

```
The IP address 192.168.152.148 does not require further analysis.
The IP address 192.168.22.115 does not require further analysis.
The IP address 192.168.190.178 has been flagged for further analysis.
The IP address 192.168.213.128 does not require further analysis.
The IP address 192.168.96.200 has been flagged for further analysis.
The IP address 192.168.247.153 does not require further analysis.
The IP address 192.168.174.117 has been flagged for further analysis.
The IP address 192.168.148.115 does not require further analysis.
The IP address 192.168.103.106 does not require further analysis.
The IP address 192.168.168.144 has been flagged for further analysis.
```

The list of flagged addresses is specified in the variable flagged_addresses.

A for loop is used to iterate through addresses in the valid_ip_addresses list generated in task 2. Each IP address there is compared against the the list of flagged addresses and an alert is printed if there is a match.

Summary

I learned about regular expressions in this lab, expressions used to match specific character combinations. Specifically I learned about the \w, \d, \., + and {x, y} operators. I also examined the use of the re regular expressions module, and the re.findall() function to check regular expressions against given strings to extract matches.

References

Google Cybersecurity Certificate (2023) Use regular expressions to find patterns.