ACTIVITY_Use regular expressions to find patterns

April 11, 2024

1 Activity: Use regular expressions to find patterns

1.1 Introduction

Security analysts often analyze log files, including those that contain information about login attempts. For example as an analyst, one might flag IP addresses that relate to unusual attempts to log in to the system.

Another area of focus in cybersecurity is detecting devices that require updates. Software updates help prevent security issues due to vulnerabilities.

Using regular expressions in Python can help automate the processes involved in both of these areas of cybersecurity. Regular expression patterns and functions can be used to efficiently extract important information from strings and files.

1.2 Scenario

In this lab, I'm working as a security analyst and your main tasks are as follows: - extracting device IDs containing certain characters from a log; these characters correspond with a certain operating system that requires an update. - extracting all IP addresses from a log and then comparing them to those that are flagged in a list.

1.3 Task

In order to work with regular expressions in Python, I'll start by importing the `re` module. This module contains many functions that will help you work with regular expressions.

[1]:
```python
# Import the `re` module in Python

import re
```

1.4 Task

In my work as a cybersecurity analyst, I'm responsible for updating devices. A device ID that begins with the characters `"r15"` indicates that the device has a certain operating system that must be updated. I'm given a log of device IDs, stored in a variable named `devices`. The eventual goal is to extract the device IDs that start with the characters `"r15"`.

[2]:
```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 9x482kt 6oa6m6u x3463ac i4l56nq␣
↪g07h55q 081qc9t r159r1u"

# Display the contents of `devices`

print(devices)
```

```
r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk 253be78 ac742a1
r15u9q5 zh86b2l ii286fq 9x482kt 6oa6m6u x3463ac i4l56nq g07h55q 081qc9t r159r1u
```

1.5 Task

In this task, I'll write a pattern to find devices that start with the character combination of `"r15"`. I'll use the regular expression symbols `\w` and `+` to create the pattern, and store it as a string in a variable named `target_pattern`.

[3]:
```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 9x482kt 6oa6m6u x3463ac i4l56nq␣
↪g07h55q 081qc9t r159r1u"

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

target_pattern = "r15\w+"
```

1.6 Task

I'll now use the `findall()` function from the `re` module to find the device IDs that the `target_pattern` matches with.

[4]:
```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 9x482kt 6oa6m6u x3463ac i4l56nq␣
↪g07h55q 081qc9t r159r1u"
```

```
# 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']
```

1.7 Task

Now, the next task is analyzing a network security log file and determining which IP addresses have been flagged for unusual activity. I'm given the log file as a string stored in a variable named `log_file`. There are some invalid IP addresses in the log file due to issues in data collection. My eventual goal is to use regular expressions to extract the valid IP addresses from the string.

[5]:
```
# 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 2022-05-09 19:30:32 192.168.190.178 \narutley␣
↪2022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.
↪128 \naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:
↪38:07 192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard␣
↪2022-05-12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.
↪247.153 \njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08␣
↪12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115␣
↪\nyappiah 2022-05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35␣
↪192.168.168.144"

# 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
```

1.8 Task

In this task, I'll build a regular expression pattern that I can use later on to extract IP addresses that are in the form of xxx.xxx.xxx.xxx. In other words, I'll extract all IP addresses that contain four segments of three digits that are separated by periods.

I'll write a regular expression pattern that will match with these IP addresses and store it in a variable named `pattern`. I will use the regular expression symbols `\d` and `\.` in the pattern. The symbol `\d` matches with digits, in other words, any integer between 0 and 9.

[6]:
```
# 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 2022-05-09 19:30:32 192.168.190.178 \narutley␣
↪2022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.
↪128 \naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:
↪38:07 192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard␣
↪2022-05-12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.
↪247.153 \njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08␣
↪12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115␣
↪\nyappiah 2022-05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35␣
↪192.168.168.144"

# 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"
```

1.9 Task

In this task, I'll use the `re.findall()` function on the regular expression pattern stored in the `pattern` variable and the provided `log_file` to extract the corresponding IP addresses.

[7]:
```
# 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 2022-05-09 19:30:32 192.168.190.178 \narutley␣
↪2022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.
↪128 \naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:
↪38:07 192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard␣
↪2022-05-12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.
↪247.153 \njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08␣
↪12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115␣
↪\nyappiah 2022-05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35␣
↪192.168.168.144"

# 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"

# Use the `re.findall()` function on `pattern` and `log_file` to extract the IP␣
↪addresses of the form xxx.xxx.xxx.xxx and display the results

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

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

1.10 Task

There are some valid IP addresses in the `log_file` that haven't been extracted yet. This is because each segment of digits in a valid IP address can have anywhere between one and three digits.

I'll have to adjust the regular expression in the `pattern` to allow for variation in the number of digits in each segment. I'll do this by using the `+` symbol after the `\d` symbol.

[8]:
```
# 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 2022-05-09 19:30:32 192.168.190.178 \narutley␣
↪2022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.
↪128 \naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:
↪38:07 192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard␣
↪2022-05-12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.
↪247.153 \njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08␣
↪12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115␣
↪\nyappiah 2022-05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35␣
↪192.168.168.144"
```

```
# Update `pattern` to a regular expression pattern that will match with IP␣
↪addresses with any variation in the number of digits per segment

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

# Use the `re.findall()` function on `pattern` and `log_file` to extract the IP␣
↪addresses of the updated form specifed above and display the results

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', '192.16874.1390.176',
'192.168.148.115', '192.168.103.10654', '192.168.168.144']
```

1.11 Task

I noted that all the IP addresses are now extracted but they also include invalid IP addresses with more than three digits per segment.

In this task, I'll update the `pattern` using curly brackets instead of the `+` symbol. In regular expressions, curly brackets can be used to represent an exact number of repetitions between two numbers. For example, `{2,4}` in a regular expression means between 2 and 4 occurrences of something. Applying this to an example, `\w{2,4}` would match with two, three, or four alphanumeric characters.

[9]:
```
# 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 2022-05-09 19:30:32 192.168.190.178 \narutley␣
↪2022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.
↪128 \naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:
↪38:07 192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard␣
↪2022-05-12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.
↪247.153 \njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08␣
↪12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115␣
↪\nyappiah 2022-05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35␣
↪192.168.168.144"

# 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']
```

1.12 Task

Now, all of the valid IP addresses have been extracted. The next step is to identify flagged IP addresses.

I'm given a list of IP addresses that have been previously flagged for unusual activity, stored in a variable named `flagged_addresses`. When these addresses are encountered, they should be investigated further.

[10]:
```
# 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"]

# Display the contents of `flagged_addresses`

print(flagged_addresses)
```

```
['192.168.190.178', '192.168.96.200', '192.168.174.117', '192.168.168.144']
```

1.13 Task

Finally, I will write an iterative statement that loops through the `valid_ip_addresses` list and checks if each IP address is flagged. In the following code, the `address` will be the loop variable. I'll also include a conditional that checks if the `address` belongs to the `flagged_addresses` list. If so, it should display `"The IP address ______ has been flagged for further analysis."` If not, it should display `"The IP address ______ does not require further analysis."`

[11]:
```
# Assign `log_file` to a string containing username, date, login time, and IP␣
↪address for a series of login attempts

```

```python
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 2022-05-09 19:30:32 192.168.190.178 \narutley 2022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.128 \naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:38:07 192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153 \njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08 12:09:10 192.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115 \nyappiah 2022-05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35 192.168.168.144"

# 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 analysis."

    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.
```

1.14 Conclusion

What are your key takeaways from this lab?

- Regular expressions in Python allow you to create patterns that you can then use to find important strings.
- Regular expression patterns can be built to match specific characters and character combinations.
- Examples of regular expression symbols practiced in this lab:
  - `\w` represents any alphanumeric character.
  - `+` represents one or more occurrences of the previous character in the regular expression.
  - `\d` represents any digit.
  - `\.` represents a period.
  - `{x,y}` represents anywhere between x and y number of occurrences of the previous character in the regular expression. The x and y can be replaced with any two positive integers to indicate an exact range for the number of occurrences.
- The `re` module in Python contains functions that are useful when working with regular expressions.
  - One example is the `re.findall()` function, which takes in a regular expression pattern as well as a string, checks for all instances in the string that match with the pattern and outputs a list of the matches.