Regular expressions in Python

We've already learned a lot about working with strings.
This includes working with their positional indices and slicing them.
In the previous video, we applied these to extract the first three digits from a list of IP addresses.
In this video, we're going to focus on a more advanced way to search through strings.
We'll learn about searching for patterns in strings through regular expressions.
A regular expression, shortened to regex, is a sequence of characters that forms a pattern.
This pattern can be used when searching within log files.
We can use them to search for any kind of pattern.
For example, we can find all strings that start with a certain prefix, or we can find all strings that are a certain length.
We can apply this to a security context in a variety of ways.
For example, let's say we needed to find all IP addresses with a network ID of 184.
Regular expressions would allow us to efficiently search for this pattern.
We'll examine another example throughout this video.
Let's say that we want to extract all the email addresses containing a log.
If we try to do this through the index method, we would need the exact email addresses we were searching for.
As security analysts, we rarely have that kind of information.
But if we use a regular expression that tells Python how an email address is structured, it would return all the strings that have the same elements as an email address.
Even if we were given a log file with thousands of lines and entries, we could extract every email in the file by searching for the structure of an email address through a regular expression.
We wouldn't need to know the specific emails to extract them.
Let's explore the regular expression symbols that we need to do this.
To begin, let's learn about the plus sign.
The plus sign is a regular expression symbol that represents one or more occurrences of a specific character.
Let's explain that through an example pattern.
The regular expression pattern a+ matches a string of any length in which "a" is repeated.
For example, just a single "a", three "a's" in a row, or five "a's" in a row.
It could even be 1000 "a's" in a row.
We can start working with a quick example to see which strings this pattern would extract.
Let's start with this string of device IDs.
These are all the instances of the letter "a" written once or multiple times in a row.
The first instance has one "a", the second has two "a's", the third one has one "a", and the fourth has three "a's".
So, if we told Python to find matches to the a+ sign regular expression, it would return this list of

"a's".
The other building block we need is the \w symbol.
This matches with any alphanumeric character, but it doesn't match symbols.
"1", "k", and "i" are just three examples of what "\w" matches.
Regular expressions can easily be combined to allow for even more patterns in a search.
Before we apply this to our email context, let's explore the patterns we can search for if we combine the "\w" with the plus sign.
"\w" matches any alphanumeric character, and the plus sign matches any number of occurrences of the character before it.
This means that the combination of "\w+" matches an alphanumeric string of any length.
"\w" provides flexibility in the alphanumeric characters that this regular expression matches, and the plus sign provides flexibility in the length of the string that it matches.
The strings "192", "abc123", and "security" are just three possible strings that match to "\w+".
Now let's apply these to extracting email addresses from a log.
Email addresses consist of text separated by certain symbols, like the @ symbol and the period.
Let's learn how we can represent this as a regular expression.
To start, let's think about the format of a typical email address; for example, user1@email1.com.
The first segment of an email address contains alphanumeric characters, and the number of alphanumeric characters may vary in length.
We can use our regular expression "\w+" for this portion to match to an alphanumeric string of any length.
The next segment in an email address is the @ symbol.
This segment is always present.
We'll enter this directly in our regular expression.
Including this is essential for ensuring that Python distinguishes email addresses from other strings.
After the @ symbol is the domain name.
Just like the first segment, this one varies depending on the email address, but it always contains alphanumeric characters, so we can use "\w+" again to allow for this variation.
Next, just like the @ symbol, a period is always part of an email address.
But unlike the @ symbol, in regular expressions, the period has a special meaning.
For this reason, we need to use backslash period here.
When we add a backslash in front of it, we let Python know that we are not intending to use it as an operator, and that our pattern should include a period in this location.
For the last segment, we can also use "\w+".
This final part of an email address is often "com" but might be other strings like "net." When we put the pieces together, we get the regular expression we'll use to find email addresses in our row.
This pattern will match all email addresses.
It will exclude everything else in our string.
This is because we've included the @ symbol and the period where they appear in the structure of an email address.
Let's bring this into Python.
We'll use regular expressions to extract email addresses from a string.
Regular expressions can be used when the re module is imported into Python, so we begin with that step.
Later, we'll learn how to import and open files like logs.
But for now, we've restored our log as a string variable named email_log.
Because this is a multi-line string, we're using three sets of quotation marks instead of just one.

Next, we'll apply the findall() function from the re module to a regular expression.
re.findall() returns a list of matches to a regular expression.
Let's use this with the regular expression we created earlier for email addresses.
The first argument is the pattern that we want to match.
Notice that we place it in quotation marks.
The second argument indicates where to search for the pattern.
In this case, we're searching through the string contained within the email log variable.
When we run this, we get a list of all the emails in the string.
Imagine applying this to a log with thousands of entries.
Pretty useful, right?
This was just an introduction to the power of regular expressions.
There are many more symbols you can use.
I encourage you to explore regular expressions on your own and learn more.

Revision #1
Created 26 December 2023 14:01:37 by naruzkurai
Updated 27 December 2023 11:35:24 by naruzkurai