Video 31 : Regular Expressions 5

In this video we cover advanced Regular Expressions techniques. I'll cover Back References, Back Reference Substitutions, Look Ahead, Look Behind, and Negative Look Ahead & Behind.

Everything We Have Learned

```
# Did you find a match
# if re.search("REGEX", my_string)

# Get list of matches
# print("Matches :", len(re.findall("REGEX", my_string)))

# Get a pattern object
# regex = re.compile("REGEX")

# Substitute the match
# my_string = regex.sub("substitution", my_string)

# [ ]   : Match what is in the brackets
# [^ ]  : Match anything not in the brackets
# ( )   : Return surrounded submatch
# .     : Match any 1 character or space
# +     : Match 1 or more of what proceeds
# ?     : Match 0 or 1
# *     : Match 0 or More
# *?    : Lazy match the smallest match
# \b    : Word boundary
# ^     : Beginning of String
# $     : End of String
# \n    : Newline
# \d    : Any 1 number
# \D    : Anything but a number
# \w    : Same as [a-zA-Z0-9_]
# \W    : Same as [^a-zA-Z0-9_]
# \s    : Same as [\f\n\r\t\v]
# \S    : Same as [^\f\n\r\t\v]
# {5}   : Match 5 of what proceeds the curly brackets
# {5,7} : Match values that are between 5 and 7 in length
# ($m)  : Allow ^ on multiline string
```

Back References

A back reference allows you to to reuse the expression that proceeds it

```
# Grab a double word
rand_str = "The cat cat fell out the window"

# Match a word boundary, 1 or more characters followed
# by a space if it is then followed by the same
# match that is surrounded by the parentheses
regex = re.compile(r"(\b\w+)\s+\1")
```

```
matches = re.findall(regex, rand_str)

print("Matches :", len(matches))

for i in matches:
    print(i)
```

Back Reference Substitutions

```
# Replace the bold tags in the link with no tags
rand_str = "<a href='#'><b>The Link</b></a>"

# Regex matches bold tags and grabs the text between
# them to be used by the back reference
regex = re.compile(r"<b>(.*?)</b>")

# Replace the tags with just the text between them
rand_str = re.sub(regex, r"\1", rand_str)

print(rand_str)
```

Another Back Reference Substitution

```
# Receive this string
rand_str = "412-555-1212"

# Match the phone number using multiple subexpressions
regex = re.compile(r"([\d]{3})-([\d]{3}-[\d]{4})")

# Output (412)555-1212
rand_str = re.sub(regex, r"(\1)\2", rand_str)

print(rand_str)
```

Python Problem for you to Solve

To solve this problem I want you to receive a string like this :

```
rand_str = "https://www.youtube.com http://www.google.com"
```

And, then Grab the URL and provide the following output using a back reference substitution :

```
<a href='https://www.youtube.com'>www.youtube.com</a>
<a href='https://www.google.com'>www.google.com</a>
```

Solution

```
regex = re.compile(r"(https?://([\w.]+))")

rand_str = re.sub(regex, r"<a href='\1'>\2</a>\n", rand_str)

print(rand_str)
```

Look Ahead

A look ahead defines a pattern to match but not return. You define the expression to look for but not return like this (?=expression)

```
rand_str = "One two three four"

# Grab all letters and numbers of 1 or more separated
# by a word boundary but don't include it
regex = re.compile(r"\w+(?=\b)")

matches = re.findall(regex, rand_str)

for i in matches:
    print(i)
```

Look Behind

The look behind looks for what is before the text to return, but doesn't return it. It is defined like (?<=expression)

```
rand_str = "1. Bread 2. Apples 3. Lettuce"

# Find the number, period and space, but only return
# the 1 or more letters or numbers that follow
regex = re.compile(r"(?<=\d.\s)\w+")

matches = re.findall(regex, rand_str)

for i in matches:
    print(i)
```

Look Ahead & Behind

```
rand_str = "<h1>I'm Important</h1> <h1>So am I</h1>"

# Use the look behind, get 1 or more of anything,
# and use the look ahead
regex = re.compile(r"(?<=<h1>).+?(?=</h1>)")

matches = re.findall(regex, rand_str)

for i in matches:
    print(i)
```

Negative Look Ahead & Behind

These are used to look for text that doesn't match the pattern

(?!expression) : Negative Look Ahead
(?<!expression) : Negative Look Behind

```
rand_str = "8 Apples $3, 1 Bread $1, 1 Cereal $4"
```

```
# Grab the total number of grocery items by ignoring the $
regex = re.compile(r"(?<!\$)\d+")

matches = re.findall(regex, rand_str)

print(len(matches))

# Convert from a string list to an int list
matches = [int(i) for i in matches]

from functools import reduce

# Sum the items in the list with reduce
print("Total Items {}".format(reduce((lambda x, y: x + y), matches)))
```

That's it for now. In the next video, I finish my coverage of Regular Expressions. We'll look at Or, Group, Named Groups, More Match Object Functions and then we'll solve some problems.