substitutions_and_flags

March 18, 2023

1 Substitutions

As we mentioned at the beginning of this lesson, the `re` module also has functions that allow us to modify strings. Regex objects have the `.sub()` method that allows us to replace patterns within a string. Let' see an example.

In the code below we have a multi-line string that contains two instances of the ampersand character, `&`. Let's use the `.sub` method to replace these ampersands with the word `and`. First we will create a regular expression that matches all the `&` characters in our string. Then we will use `regex.sub(r'and', sample_text)` to replace every match of the `regex` expression in the `sample_text` with the raw string `and`. Let's see this in action:

```
In [1]: # Import re module
        import re

        # Sample text
        sample_text = '''
        Ben & Jerry
        Jack & Jill
        '''

        # Create a regular expression object with the regular expression '&'
        regex = re.compile(r'&')

        # Substitute all & in the sample_text with 'and'
        new_text = regex.sub(r'and', sample_text)

        # Print Original and Modified texts
        print('Original text:', sample_text)
        print('Modified text:', new_text)
```

```
Original text:
Ben & Jerry
Jack & Jill

Modified text:
Ben and Jerry
Jack and Jill
```

We can see that we have successfully replaced all the & characters with the word and. Being able to make this kind of substitutions can be really useful and save you a lot of time if you are working with large documents that you need to reformat.

2 Substitutions with Groups

We can do more sophisticated substitutions by using groups. Let's see an example. In the code below we have a multi-line string that contains the names of 4 people. As we can see, some people have middle names but other don't. Let's use the `.sub()` method to replace all names in the string with just the first and last name. For example, the name `John David Smith` should be replaced by `John Smith` and `Alice Jackson` should stay the same.

The first step is to create a regular expression that matches all the names in the list. Now, keeping in mind that we need to be able to make replacements later we will use groups to be able to distinguish between the first name, the middle name, and the last name. Since all names have a first name then we can use this group `([a-zA-z]+)` to match all the first names. Now, not all names have middle names, so having a middle name is optional. Since the first and middle name are separated by a whitespace we also need to indicate that the whitespace is also optional. So, to do indicate that the whitespace and middle name are optional we will include the `?` metacharacter after the whitespace and second group, `[ ]?([a-zA-z]+)?`. After the first or middle name we have a whitespace that we can match with `\[ \]`. Notice that in this case we didn't use the sequence `\\s` since this will match newlines as well and we don't what match those. Finally we make a third group to match the last name. Since all names have last names, we don't need to use the `?` metacharacter. Putting all together we get:

```
In [2]: # Import re module
        import re

        # Sample text
        sample_text = '''
        John David Smith
        Alice Jackson
        Mary Elizabeth Wilson
        Mike Brown
        '''

        # Create a regular expression object with a regular expression that can find all
        # the names in the sample_text and group the first, middle, and
        # last names separately
        regex = re.compile(r'([a-zA-z]+)[ ]?([a-zA-z]+)?[ ]([a-zA-z]+)')

        # Search the sample_text for the regular expression
        matches = regex.finditer(sample_text)

        # Print all the matches
        for match in matches:
            print(match)
```

```
<_sre.SRE_Match object; span=(1, 17), match='John David Smith'>
<_sre.SRE_Match object; span=(18, 31), match='Alice Jackson'>
<_sre.SRE_Match object; span=(32, 53), match='Mary Elizabeth Wilson'>
<_sre.SRE_Match object; span=(54, 64), match='Mike Brown'>
```

We can clearly see that we matched all the four names in our list. Now, the cool thing about using groups is that we can reference them individually from the Match Objects using the `.group()` method. The `.group(N)` method selects the `N`th group in the match. Therefore, in our particular case, for each match, `.group(1)` will select the first name, `.group(2)` will select the middle name, and `.group(3)` will select the last name. Let's see how this works in the code below:

```
In [3]: # Import re module
        import re

        # Sample text
        sample_text = '''
        John David Smith
        Alice Jackson
        Mary Elizabeth Wilson
        Mike Brown
        '''

        # Create a regular expression object with a regular expression that can find all
        # the names in the sample_text and group the first, middle, and
        # last names separately
        regex = re.compile(r'([a-zA-z]+)[ ]?([a-zA-z]+)?[ ]([a-zA-z]+)')

        # Search the sample_text for the regular expression
        matches = regex.finditer(sample_text)

        # For each match print the first, middle, and last name separately
        for match in matches:
            print('\nFirst Name: '+ match.group(1))

            if match.group(2) is None:
                print('Middle Name: None')
            else:
                print('Middle Name: '+ match.group(2))
            print('Last Name: '+ match.group(3))
```

```
First Name: John
Middle Name: David
Last Name: Smith
```

```
First Name: Alice
Middle Name: None
Last Name: Jackson

First Name: Mary
Middle Name: Elizabeth
Last Name: Wilson

First Name: Mike
Middle Name: None
Last Name: Brown
```

We can see that for each of the four matches we can selectively choose the first, middle, or last name. We should also mention that `.group(0)` (or equivalently `.group()`) selects all the groups at once.

Now, that we know how to select groups individually for each match, we are ready to use the `.sub()` method to make substitutions. Remember, `regex.sub(r'string', sample_text)` will replace every match of the `regex` expression in the `sample_text` with the raw string `string`. So, what we want to do in our case, is to replace every match with only the first and last names, or equivalently replace every match with the first and third groups. We can refer to each group in the `string` by using the backslash. For example, `regex.sub(r'\1', , sample_text)` will replace every match with the first group. Here we have reference the first group by using `\1` inside the `string`. Let's put it all together to see how it works:

```
In [4]: # Import re module
        import re

        # Sample text
        sample_text = '''
        John David Smith
        Alice Jackson
        Mary Elizabeth Wilson
        Mike Brown
        '''

        # Create a regular expression object with a regular expression that can find all
        # the names in the sample_text and group the first, middle, and
        # last names separately
        regex = re.compile(r'([a-zA-z]+)[ ]?([a-zA-z]+)?[ ]([a-zA-z]+)')

        # Substitute all names in the sample_text with the first and last name
        new_text = regex.sub(r'\1 \3', sample_text)

        # Print the modified text
        print(new_text)
```

```
John Smith
```

```
Alice Jackson
Mary Wilson
Mike Brown
```

3 Flags

We saw at the beginning of this lesson that regexes are case sensitive, therefore we often have to use regexes with both uppercase and lower case letters. However, the `re.compile(pattern, flags)` function, has a `flag` keyword that can be used to allow more flexibility. For example, the `re.IGNORECASE` flag can be used to perform **case-insensitive** matching. In the code below we have a string that contains the name Walter written in two different combinations of upper and lower case letters. In order to be able to find this two renditions of Walter, we will probably have to use a long character set to be able to account for all possible combinations of lower and upper case letters. However, in this case we can use the `re.IGNORECASE` to indicate that we don't care about the case of the letters, we just want to find the name Walter no matter how it is written. Let's see how this works:

```
In [5]: # Import re module
        import re

        # Sample text
        sample_text = 'Alice and WaLtEr Brown are talking with wAlTer Jackson.'

        # Create a regular expression object with the regular expression 'walter'
        # that ignores the case of the letters
        regex = re.compile(r'walter', re.IGNORECASE)

        # Search the sample_text for the regular expression
        matches = regex.finditer(sample_text)

        # Print all the matches
        for match in matches:
            print(match)
```

```
<_sre.SRE_Match object; span=(10, 16), match='WaLtEr'>
<_sre.SRE_Match object; span=(40, 46), match='wAlTer'>
```

We can clearly see that we were able to match both renditions of `walter` without any fancy regular expression.

We have seen a lot in this lesson and we have just began to scratch the surface of regular expressions. For more information on regexes make sure to check out the Python Regex Documentation

```
In [ ]:
```