Regular Expressions in Practice: Python

The module for using regular expressions in Python is re. It is part of the Python standard library, so you don't have to install
anything. To be able to use the module, import it first. Also import the csv module, since we'll be reading in a csv file. Then
let's read in the leader data file we just converted from .txt to .csv:

```
>>> import re
>>> import csv

>>> with open('example_data_leaders.csv', 'rU') as f:
...     reader = csv.reader(f)
...     data = [row for row in reader]
...
```

The re module contains several functions to enable us to match strings by regular expressions, find and replace instances
of a pattern, etc. Let's start with the function `match()`. It takes two arguments: the regular expression pattern to match,
then the string that we want to match to that pattern.

Let's say that we want to find all of our leaders' prior experience that involved a cabinet minister position. We'll use
a simple for loop to iterate through the rows in our data, look at the cell for prior experience, and print it out if it matches
a regular expression for "minister". (To keep this very simple, we're just looking at the last cell in each row.)

Recall that regular expressions are case sensitive, so we can either convert the input string to lowercase, or include both
uppercase and lowercase versions in the regular expression (e.g. `[Mm]inister`). Let's just convert everything to lowercase,
so we don't miss some possible capitalizations (like all caps).

```
>>> for row in data:
...     if re.match('minister', row[-1].lower()): print(row[-1])
...
```

```
Minister of Public Works
```

Why did we only get one position, when we should have more than one? The function `match()` only matches from the beginning of
the given string. To find a sequence anywhere within the given string, we can use the function `search()`, with the same arguments:

```
>>> for row in data:
...     if re.search('minister', row[-1].lower()): print(row[-1])
...
```

```
economist, business executive, Minister of Energy, Minister of Economy, Prime
Minister
lawyer, legislator, party leader, Minister of Education, Minister of Justice
economist, professor, Minister of Finance
physician, professor, government agency official, cabinet minister, Vice
President
legislator, mayor, party leader, cabinet minister
Minister of Public Works
```

Great, we found a lot. But we see now that it's printing out the leader's entire list of positions, and we might only want
to look at the position that had "minister" in it. Let's capture the specific position in a group and just print out that
group. We want to make sure we capture the full position, not just the word "minister". So we want to get all of
the characters before and after that word that are not a comma (since the positions are separated by commas in the string).

The functions match() and search() return a match object, that contains information about what matching sequence
was found in the given string. If there is no match, the function returns the value None, which is why we were able to treat
it like a logical value in the examples above. (We used an `if` statement to say that if there was any match found, we
wanted to print out the whole string.)

Now we want to assign the function's output to a variable, so that we can retrieve certain components from it. We'll make sure
the function did return an object (or we'll get an error, if we try to retrieve components from None). Then if there was a match,
we'll use the method .group() which returns the sequence from the input string that matched the regular expression.

As you might imagine, .group() can be used to retrieve a specific numbered group, if we use parenthesis to separate parts of the
expression. But we'll just call .group() with no arguments, so it returns the entire matching sequence.

```
>>> for row in data:
...     match = re.search('[^,]*minister[^,]*', row[-1].lower())
...     if match:
...         print(match.group())
...
```

```
 minister of energy
 minister of education
 minister of finance
 cabinet minister
 cabinet minister
minister of public works
```

We're getting more positions, but we're still only getting one per leader, and some leaders held more than one cabinet office. We can use
the function `findall()` to find all matches to an expression within the same string. The function `findall()` just returns a list of
strings that represent the sequences from the original string matching the regular expression. It isn't an object, so we don't use the method .group(), we can just print out the list.

(Note: If we created groups using parentheses, `findall()` would return a list with separate items for each group. We're keeping it simple here by not having any groups. You might also notice that we're getting a leading space for some of these results, because we grabbed everything after the last comma. If we just want a neat list of positions, we could use the string method .strip() at the end. Or we could add an initial space that we don't capture, i.e. outside of a group parentheses, but we won't worry about that for now.)

```
>>> for row in data:
...     match = re.findall('[^,]*minister[^,]*', row[-1].lower())
...     if match:
...         print(match)
...
```

```
[' minister of energy', ' minister of economy', ' prime minister']
[' minister of education', ' minister of justice']
[' minister of finance']
[' cabinet minister']
[' cabinet minister']
['minister of public works']
```

What if we want to replace these different positions with a common term, e.g. use `cabinet minister` for all of them? The re module
comes with a function sub(), which works like the string function `replace()`, except that we can use regular expressions. The function
sub() takes three arguments: a regular expression pattern to find, a string to substitute wherever that expression matches, and a main
string in which we want to find and replace.

Let's use the same regular expression from above, but now we'll use sub() and add a replacement string `cabinet minister`.

```
>>> for row in data:
...     re.sub('[^,]*minister[^,]*', 'cabinet minister', row[-1].lower())
...
```

```
'economist, business executive,cabinet minister,cabinet minister,cabinet
minister'
'military officer'
'legislator, party leader'
'economist, consultant for international organizations'
'lawyer, legislator, party leader,cabinet minister,cabinet minister'
'mathematician, professor, dean'
'economist, professor,cabinet minister'
'physician, professor, government agency official,cabinet minister, vice
president'
'military colonel'
'governor, ministry official, chancellor/dean, vice president'
'legislator, mayor, party leader,cabinet minister'
'legislator, head of legislature'
'head of police service, college instructor, mayor'
'cabinet minister'
'legislator, professor'
```

The function sub() returns the input string with the substitutions made. It doesn't change the original string object. So since
we didn't save that output, it just printed to the screen. Let's put that output back into the last cell in our data row.

We can see from the printed output, though, that we've ended up with several copies of "cabinet office" where there were multiple
minister positions. We might want to replace multiple copies with just one copy. And we replaced the initial space when the position
came after a comma, so let's add a space whenever we see an instance of "cabinet office" after a comma. We'll save this to the
original data row as well, but we can print them out to make sure they look right.

```
>>> for row in data:
...     row[-1] = re.sub('[^,]*minister[^,]*', 'cabinet minister',
row[-1].lower())
...     row[-1] = re.sub('(,cabinet minister)+', ', cabinet minister', row[-1])
...     print(row[-1])
...
```

```
economist, business executive, cabinet minister
military officer
legislator, party leader
economist, consultant for international organizations
lawyer, legislator, party leader, cabinet minister
mathematician, professor, dean
economist, professor, cabinet minister
physician, professor, government agency official, cabinet minister, vice
president
military colonel
governor, ministry official, chancellor/dean, vice president
legislator, mayor, party leader, cabinet minister
legislator, head of legislature
head of police service, college instructor, mayor
cabinet minister
legislator, professor
```

Now this looks right!