Lesson 2, Week 4:
Obtaining and manipulating the words in a text file

AIM

— To read a text file, extract the words only, and manipulate them

After this lesson, you will be able to

* Depunctuate a text file line by line
* Turn a file of depunctuated text strings into a file of words
* Count various words and phrases in a file of words

Words only: removing the punctuation

Reading in the data

Reminder: the file `prideandprejudiceextract.txt` is used to provide data for this week's work. Let's read in the data with `open` and `readlines` :

```
fh = open("prideandprejudiceextract.txt")
pridelines = readlines(fh)
```

`close(fh)` and then use `filter!` to get rid of the empty lines:

```
filter!(x -> !isempty(x), pridelines)
```

Take small steps

A good habit: work on small pieces of data when you develop code[1]. So before depunctuating the whole file, let's pick a line with several marks, for instance line 10:

[1] If possible!

`sampleline = pridelins[10]`
and let's see it character by character: `samplechars = [x for x in sampleline]`.

Removing the full stop

So an easy character to work on would be the full stop. We can again use `filter!`. It has to operate on an array, so we use `samplechars`. The test `x == '.'` is true when the (local) name `x` is bound to the value `'.'` (programmers usually use the shorthand "when x is a full stop"), but we want `filter!` to retain all the other values, so the anonymous function we need is `x -> !(x == '.')`. Putting it all together, we reach `filter!(x -> !(x == '.'), samplechars)`. [DEMO]

But that the result is an array, so `join` is needed to get the string back:
`join(filter!(x -> !(x == '.'), samplechars))`.

Scaling up: removing the other marks

To filter out the commas as well, we note that the logical expression we need is `!(x == '.'  || x == ',')` [DEMO]

That leaves the quote characters. Note that they are not double quote characters—but then how to find out what they are? You can simply index for it: `sampleline[1]`. [DEMO]

Unfortunately, unless we use `'\u201c'`, that doesn't help us to type it in. One thing to try is whether it might come up as a "\ + letters + Tab" character, and it turns out that simply starting with \quo does the trick. [DEMO]

For the full text, we also need to remove the question marks, colons, semi-colons, exclamation marks—but there's more, see below

Scaling up to the full text

We now face two problems: filtering the other marks, and iterating over all the lines. Let's start with iterating over all the lines. For this, we can use a loop starting with `for line in pridelines` but then we have a problem: `line` is a local variable inside the loop. But we want the results of the depunctuation available globally!

No problem: we create a global array, start it empty, and use `push!` inside the loop to extend it:

```
depunclines = []
for line in pridelines
 newline = join(filter!(x -> !(x == '.'  || x == ',' || x == '\' || x == '"'), [x for x in line]))
push!(depunclines, newline)
```

```
end
```

The logical test in `filter!` is getting longer and longer. One could repeat it several time, for different punctuation marks each time. But an alternative is to just to break it up over several lines:

```
depunclines = [];
for line in pridelines
  newline = join(filter!(x -> !(x == '.'  || x == ','
        || x == '\' || x == '"' || x == '?'  || x == '!'
        || x == '_' || x == ':'  || x == ';' || x = '\'' ),
        [x for x in line]));
push!(depunclines, newline)
end
```

Escaping the single quote character this way is not the best thing to do. Another option, only slightly better, is to replace it with a space, which we leave to the exercises.

Turning the array of strings into an array of words

So now we have the array of `depunclines`. Each line is a string, eg. `depunclines[10]`, which we can split into words with `split(depunclines[10], ' ')`.

Actually, we can turn use `join` to join up all the strings in `depunclines` into one string, and then split it in the same way.

Naively, we could try `split(join(depunclines), ' ')`. However, although in the main it looks good it seems to join some words that should stay separate. Debugging time! The reason is that there is no space between the end of one line and the start of the next. We can put space in with the `join` function, leading to

```
pridewords = split(join(depunclines, ' '), ' ')
```

Counting some of the interesting words and phrases

A simple way to get all occurrences of Mr Bingley's name is by using the `filter` function: `filter(x -> x == "and", pridewords)`.

[DEMO: do this for other name, common words like "I", "the", "and", etc.]

Even more interesting is to search for phrases. One might for instance ask for the phrases "Mr Bennet" and "Mrs Bennet". The problem now is that a `pridewords` does not contain phrases, but only words. So we need a different approach.

One way is to check every pair of words. We loop over the file, from the first to the second-last word, and compare the current pair to the target. However, we cannot simply use `for word in pridewords`

because that only gives us one word at a time. The solution is to use a range, and to index the pair we need as a sub-array. The code below looks for the phrase “Mrs Bennet”

```
count = 0; numwords = length(pridewords)
for wordnumber in 1:numwords-1
  if join(pridewords[wordnumber:wordnumber+1], ' ') == "Mrs Bennet"
    global count = count + 1
  end
end
count
```

Review and summary

* Use `filter!` with test functions similar to `x -> !(x == '.')` to remove punctuation marks
* Filter out the punctuation line by line
* Form an array of strings containing words only into an array of words
* Count given words and given phrases