File I/O

PS 452: Text as Data

Fall 2014

Department of Political Science, Stanford University

Created by **Frances Zlotnick**

Please contact me at zlotnick@stanford.edu with questions or comments.

This brief tutorial will introduce you to reading and writing files in python. For now we will stick mainly to regular text (i.e. not CSVs).

Reading Text

There are several functions available for reading text from a file. Which one you chose depends on the nature of the text, the size of the data and what you plan to do with it. A few things to think about before you start:

- Are line breaks meaningful? Do you want to preserve the structure imposed by the line breaks in the data, or is that just noise?
- How big is your data relative to your RAM?
- What kind of operations do you intend to do on your text once you've read it in, and what is the most efficient way to store it based on the functions you will be using later?

First we need to locate and open the file we want to read from. The `open` function takes two string arguments, the file name and a "mode." The mode defines what you will be able to do to this file. Here we just want to read from it, so our mode is "r". You may see code useing "rb", which allows you to read binary data files; for the type of data we are working with, this is not important.

In [1]:
```
import os
#set working directory
os.chdir("/Users/franceszlotnick/Dropbox/TextAsData/Sections/week1/")

f = open("textExample1.txt", "r")
```

This opens up a stream for you to read from. The stream itself contains none of your data, you have to call another function to read from it.

In [2]:
```
print f
```

```
<open file 'textExample1.txt', mode 'r' at 0x103355540>
```

read

The `read` function returns a single string, terminating at the last end of line (EOL) character.

```
In [3]: f.read()
```

```
Out[3]: 'This file contains several lines.\nThis is line 1!\nThis is line 2.\nThis i
        s....wait for it....line 3.\nLine 4!!!!!\nOk this is the last line.'
```

If you call this function again, you'll get an empty string. Why? Because you've already reached the end of file, and there's no more data to read. To reset the stream, you have to close it and reopen it.

```
In [4]: f.read()
```

```
Out[4]: ''
```

```
In [5]: f.close()

        f = open("textExample1.txt", "r")
        f.read()
```

```
Out[5]: 'This file contains several lines.\nThis is line 1!\nThis is line 2.\nThis i
        s....wait for it....line 3.\nLine 4!!!!!\nOk this is the last line.'
```

```
In [6]: f.close()
```

You can also use python's `with` syntax to open the file, which will automatically close the file afterwards. But the text won't automatically print to the console this way, so if you want to see it, you'll have to save it as an object and then print the object.

```
In [7]: with open("textExample1.txt", "r") as f:
            lines = f.read()

        lines
```

```
Out[7]: 'This file contains several lines.\nThis is line 1!\nThis is line 2.\nThis i
        s....wait for it....line 3.\nLine 4!!!!!\nOk this is the last line.'
```

readline

The `readline` function will return as a string the contents of the file up to and including the first EOL character.

```
In [8]: file = open("textExample1.txt", "r")
        file.readline()
```

```
Out[8]: 'This file contains several lines.\n'
```

To get the next line, you must call the function again. But you will lose the previous one, so make sure to save it as an object if you want to do something with it later. Once you get to the end of the document, this function will start returning empty strings.

```
In [9]: file.readline()
```

```
Out[9]: 'This is line 1!\n'
```

```
In [10]: file.close()
```

readlines

The `readlines` function will read the entire document and store each line as an element in a single list object.

```
In [11]: file = open("textExample1.txt","r")

         lines = file.readlines()
         lines
```

```
Out[11]: ['This file contains several lines.\n',
          'This is line 1!\n',
          'This is line 2.\n',
          'This is....wait for it....line 3.\n',
          'Line 4!!!!!\n',
          'Ok this is the last line.']
```

This is a really useful format if you don't have too much data and you want to be able to iterate over all of your lines.

Writing files

To write a file from python, we once again open a file, but we use use the "w" mode, for "write." As with reading mode, you may come across code using mode "wb", which allows you to write binary files. This is not important for our purposes.

The function you use to write to the file will depend on the structure of your data.

```
In [12]: fileToWrite = open("newFile.txt", "w")
```

write

We can use the write function to write strings (and only strings) to the file. Your data won't actually be written to the file until you close the stream, so don't forget that step.

```
In [13]: fileToWrite.write("Hello! Welcome to my new file.")
         fileToWrite.close()
```

You can also use the with syntax if you don't want to worry about closing your file afterwards.

If you try to write something other than a string using this function, it will throw an error.

In [14]:
```
with open("newFile.txt", "w") as fileToWrite:
    fileToWrite.write(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-46de1bcba692> in <module>()
      1 with open("newFile.txt", "w") as fileToWrite:
----> 2     fileToWrite.write(1)

TypeError: expected a character buffer object
```

You can coerce other data types into strings either by surrounding them in quotation marks or wrapping them in str().

In [15]:
```
with open("newFile.txt", "w") as fileToWrite:
    fileToWrite.write("1")

#is equivalent to
with open("newFile.txt", "w") as fileToWrite:
    fileToWrite.write(str(1))
```

Note that it is **extremely** easy to accidentally overwrite files and lose your data this way, so be careful when writing files. Change the mode argument "a" if you want to append to a file rather than replace it.

In [16]:
```
with open("newFile.txt", "a") as fileToWrite:
    fileToWrite.write("This text will be appended.")
```

writelines

The `writelines` function will accept lists as well as strings, but the elements of the list still need to be strings.

In [17]:
```
l = ["hi", "there","this", "is", "a","list"]

with open("newFile.txt", "a") as fileToWrite:
    fileToWrite.writelines(l)

#this will write "hitherethisisalist"
```

Keep in mind that you may want to add in line breaks or spaces between your list elements; by default they will be concatenated. The join function concatenates the elements of a list with a specified separator and returns the resulting string. Note the weird syntax: **(arguments).function(object)** rathern than the more typical **object.function(arguments)**.

In [18]:
```
#to write with spaces between the words
with open("newFile.txt", "a") as fileToWrite:
    fileToWrite.write((" ").join(l))

#to write with line breaks between the words
with open("newFile.txt", "a") as fileToWrite:
    fileToWrite.write(("\n").join(l))
```

You can even use this to method to write CSVs if you want, but there are packages that are better for this task.

In [19]:
```
#to write a single line of a CSV
with open("newFile.txt", "a") as fileToWrite:
    fileToWrite.write((",").join(l))
```

Summary

Reading from file

- "r" mode
- read vs readline vs readlines

Writing to file

- "w" or "a" mode
- write vs writeline