Temporary Files

- `tempfile.TemporaryFile` takes several optional arguments.
  - `mode='w+b'`
  - `buffering=None`
  - `encoding=None`
  - `newline=None`
  - `suffix=''`
  - `prefix='tmp'`
  - `dir=None`
- temporary files are not guaranteed to exist on the disk; closing temporary files deletes them
- `tempfile.NamedTemporaryFile` creation of file is guaranteed to exist on the disk.

Here's an example. Suppose we create a named text file. If has a location, which we can see with the `name` method.

```
>>> import tempfile
>>> tmp = tempfile.NamedTemporaryFile('w+t', prefix='example')
>>> tmp.name
'/var/folders/hf/3nb9cj8x4b36l7hdt1sfqslh0000gn/T/examplea4rho8cc'
```

If we investigate the folder give to us by `tmp.name` then we'll see the file.

```
$ pushd /var/folders/hf/3nb9cj8x4b36l7hdt1sfqslh0000gn/T/
/var/folders/hf/3nb9cj8x4b36l7hdt1sfqslh0000gn/T
$ ls examplea4rho8cc
examplea4rho8cc
```

Let's write something to the file using the print statement (notice here we use the named parameter `file` to actually send our string to the file instead of standard output — by default `file=sys.stdout`).

```
>>> print("The number {} is even".format(2),file=tmp)
```

Now we'll investigate the contents of the file.

```
$ more examplea4rho8cc
```

The file is empty. That's because Python buffers the content. We can force it to flush it to the file, however.

```
>>> tmp.flush()
```

And now can check the file again.

```
$ more examplea4rho8cc
The number 2 is even
```

If we close the file

```
>>> tmp.close()
```

it disappears.

```
$ more examplea4rho8cc
examplea4rho8cc: No such file or directory
```

CSV

Comma Separated Value (CSV) is a common data file format that represents data as a row of values, separated by a delimiter, which is typically a comma. Data in spreadsheets and databases matches this format nicely, so CSV is often used as an export format.

For example, here is CSV data representing financial information from Apple Computer

```
Date,Open,High,Low,Close,Volume,Adj Close
2009-12-31,213.13,213.35,210.56,210.73,88102700,28.40
2009-12-30,208.83,212.00,208.31,211.64,103021100,28.52
2009-12-29,212.63,212.72,208.73,209.10,111301400,28.18
2009-12-28,211.72,213.95,209.61,211.61,161141400,28.51
```

header many CSV files start with an initial header row, which gives column names for the data

data data in CSVs is separated by commas, but any delimiter can be used.

Readers

Suppose the the contents of the above CSV were in a file called `aapl.csv`. One could open that CSV and stream through the data using the following syntax. The `reader` object is iterable—each row is a list of strings.

```
1 reader = csv.reader(aapl.csv}
2 [row for row in reader]
```

```
[['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'],
 ['2009-12-31', '213.13', '213.35', '210.56', '210.73', '88102700', '28.40'],
 ['2009-12-30', '208.83', '212.00', '208.31', '211.64', '103021100', '28.52'],
 ['2009-12-29', '212.63', '212.72', '208.73', '209.10', '111301400', '28.18'],
 ['2009-12-28', '211.72', '213.95', '209.61', '211.61', '161141400', '28.51']]
```

Suppose that the CSV data, however, is in a string `data`, instead of a file. In this case, one would use the `io.StringIO` type to wrap the string inside something that behaves like a *file object*. You can think of this as *buffering* the string.

```
1 reader = csv.reader(io.StringIO(data))
2 for row in reader:
3   print(",".join(row))
```

Reader Options

There are many options when creating a CSV reader. Here are some, with definitions coming directly from the API[1]:

`delimiter` A one-character string used to separate fields. It defaults to ','.

`doublequote` Controls how instances of `quotechar` appearing inside a field should be themselves be quoted. When `True`, the character is doubled. When `False`, the `escapechar` is used as a prefix to the `quotechar`. It defaults to `True`. On output, if `doublequote` is `False` and no `escapechar` is set, Error is raised if a `quotechar` is found in a field.

[1] `https://docs.python.org/3.4/library/csv.html#csv-fmt-params`

escapechar A one-character string used by the writer to escape the delimiter if quoting is set to `QUOTE_NONE` and the `quotechar` if `doublequote` is `False`. On reading, the `escapechar` removes any special meaning from the following character. It defaults to `None`, which disables escaping.

lineterminator The string used to terminate lines produced by the writer. It defaults to `'\r\n'`. Note The reader is hard-coded to recognise either `'\r'` or `'\n'` as end-of-line, and ignores line terminator. This behavior may change in the future.

quotechar A one-character string used to quote fields containing special characters, such as the delimiter or `quotechar`, or which contain new-line characters. It defaults to "".

As an extreme example, suppose we wanted to represent a bunch of data that was just commas. One could use a different delimiter

```
,|,,|,
,,|,|,,
```

and use `csv.reader(filename.csv, delimiter="|")` to create the correct reader. We could also escape the commas

```
\,,\,\,,\,
\,\,,\,,\,\,
```

and use `csv.reader(filename.csv, escapechar="\\")` to create the correct reader. Notice that we need to escape the backslash inside the character string.

Writers

CSV Writer objects accept any object that has a `write` method (file objects, `StringIO` objects, etc.) and formats CSV data using the `writerow` or `writerows` method. Here's an example. Suppose that data is a list of NESCAC school information.

```
[['Williams', 'Ephs', 'Purple Cows'],
 ['Amherst', 'Lord Jefs', 'Lord Jeffrey Amherst'],
 ['Middlebury', 'Panthers', 'Panther']]
```

To write this to the file called `nescac.csv` we would use the following code

```
1 import csv
2 with open('nescac.csv', 'w', newline='') as csvfile:
3     writer = csv.writer(csvfile, delimiter=',')
4     writer.writerow(['School', 'Nickname', 'Mascot'])
5     writer.writerows(data)
```

Practice

Suppose you had a list of constellations and their galactic coordinates (right ascension and declination) in CSV format.

```
constellation, right ascension, declination
Sagittarius,19,-25
Taurus, 4.9, 19
Perseus, 3, 45
```

Write a function that takes a filename `file` in CSV format and returns a list of constellations. Suppose that you know one of the headers is labelled `constellation`, but not which one. Suppose further that you can easily fit all the data in memory.

```
1 with open(file) as fp:
2   data = [row for row in csv.reader(file)]
3   col = data[0].index('constellation')
4   return [row[col] for row in data[1:]]
```