File IO

Text File IO

- File IO is done in Python with the built-in `File` object which is returned by the built-in open function
- Use the 'w' open mode for writing

```
$ python
>>> f = open("hello.txt", w ) # open for writing, create if necessary
>>> f.write("Hello, file!\n") # write string to file; notice \n ending
>>> f.close()                 # close file, causing it to write to disk
>>> exit()
$ cat hello.txt
Hello, file!
```

Use the 'r' open mode for reading

```
$ python
>>> f = open("hello.txt", "r") # open for reading in text mode
>>> contents = f.read()
# slurp the whole file into memory
>>> contents
 Hello , file !\ n
>>> exit()
```

Reading Lines from Text Files

- Text files often have data split into lines
- the `readlines()` function reads all lines into memory as a list

```
>>> f = open(lines.txt, r)
>>> f.readlines()
[line 1\n, line 2\n, line 3\n]
```

- `readline()` reads one line at a time, returns empty string when fully read
- re-open file or use seek() to go back to beginning of file

```
>>> f = open(lines.txt, r)
>>> f.readline()
 line 1\n
>>> f.readline()
 line 2\n
>>> f.readline()
 line 3\n
>>> f.readline()
''
>>> f.seek(0)
>>> f.readline()
 line 1\n
```

Processing Lines in a Text File

Could use readlines() and iterate through list it returns

```
>>> f = open(lines.txt, r)
>>> for line in f.readlines():
...     print line
...
line 1
line 2
line 3
```

Better to take advantage of fact `File` is `Iterable`

```
>>> for line in open(lines.txt, r):
...     print line
...
line 1
line 2
line 3
```

Files are Buffered

Try a little experiment. create a subdirectory named foo, cd to your new empty `foo` directory, lauch a Python shell, create open a new file named bar, and write something to it:

```
$ mkdir foo
$ cd foo
$ python3
Python 3.4.0 (v3.4.0:04f714765c13, Mar 15 2014, 23:02:41) ...
>>> bar = open("bar",  w )
>>> bar.write("last call!")
10
>>>
```

At this point, open another command shell or use your graphical file explorer to view the contents of the bar file. It's empty. Now go back to your Python shell and do:

```
>>> bar.close()
```

Now view the contents of the `bar` file again. It has the text from the previous `write()` call. Files are buffered, and the buffer isn't (guaranteed to be) flushed to disk until the file object is closed or the `File` Object goes out of scope or the program terminates (gracefully).

Context Management with `with`

Python has context managers to close resources automatically. A context manager has the form

```
with expression as variable:
    block
```

which is equivalent to

```
variable = expression
block
variable.close()
```

For example, the previous bar example is:

```
>>> with open( bar , w ) as bar:
...     bar.write( last call! )
...
```

And the file is closed and flushed to disk automatically after the block under the with statement finishes.

Listing Files in a directory

```
import os
dir = 'some_dir'
for path in os.listdir(dir):
    if os.path.isdir(submission_dir):
        print(path + '/')
    else:
        print(path)
```

Moving and Copying Files

```
import shutil
shutil.move(source, destination)
shutil.copy(source, destination)
```

Making directories

```
import shutil
dir = 'some_dir'
shutil.mkdir(dir)
```