File Manipulation

Reading Files

- Open the file

```
infile = open('data1.txt', 'r')
```

- Loop over lines of the file

```
for line in infile:
    # do something with line
```

- We can read all lines to a list:

```
lines = infile.readlines()
```

```
lines = []
for line in infile:
    lines.append(line)
```

Try It

- Read in the file 'numbers.txt'
- Compute the mean of the numbers with

```
mean = 0
for number in lines:
    mean = mean + number
mean = mean/len(lines)
```

- How well does it work?

Type Casting

- Error Message:

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- Each line is read as a string, but mean is an integer

```
mean = 0
for line in lines:
    number = float(line)
    mean = mean + number
mean = mean/len(lines)
```

Parsing Files

- For any string, you can strip trailing whitespace characters with the 'strip' function
  - string.strip()
  - Default is whitespace
- You can also split a string into a list
  - string.split(delimiter)

Practicals

1) Read words from 'words.txt' into a list and sort the list

- Don't forget about the sorted() routine
- Lists also have a built-in sort method.

2) Read the comma separated file 'numbers.cvs' and store the 1st and 4th column in a list

Reading Block Text

- If data is stored in columns, you can read it using a module function
  - numpy.loadtxt(file_str, keyword_args)
  - numpy.genfromtxt(file_str, keyword_args)
- Keywords
  - dtype: string that tells what type the file
- Can only be one string
- If you use None for genfromtxt, it will make its best guess for each column
  - usecols: list of which columns to read
  - delimiter: string of which delimiters to use

Writing to a File

- Use open again, but with 'w' or 'a'
- print >> file, “Printed Values”
- file.write(“Printed Values\n”)

- Contents not written into file until the file is closed!

Dictionaries

- Hash tables
- Similar to lists, but can be indexed by anything
- Built using a 'key : value' pair
  - D = {}   # Creating blank dictionary
  - D = {'my_key' : 5}   # One entry
  - D['my_key']   # Accessing entry

Dictionary Example

- City Temperatures

```
temps = {'Oslo': 13, 'London': 15.4, 'Paris': 17.5}
# or
temps = dict(Oslo=13, London=15.4, Paris=17.5)
```

- Adding a value

```
temps['Madrid'] = 26.0
```

  – Note that Madrid didn't exist until we indexed it and added a value

Looping over a Dictionary

- For key in dictionary:

```
>>> for city in temps:
...     print 'The temperature in %s is %g' % (city, temps[city])
...
The temperature in Paris is 17.5
The temperature in Oslo is 13
The temperature in London is 15.4
The temperature in Madrid is 26
```

- Is a key in the dictionary?

```
>>> if 'Berlin' in temps:
...     print 'Berlin:', temps['Berlin']
... else:
...     print 'No temperature data for Berlin'
...
No temperature data for Berlin
```

Practicals

3) Read in the file 'block_data.txt' using either loadtxt or genfrom txt. Print the month and year that made the most money

4) Print cos(x) to a file in pairs

```
x1, y1
X2, y2
…
```

5) Read words from 'words.txt' and count how many times they occur

Storing Data Objects

- Sometimes, you don't want to store your data in text format.
- Instead, you can store the data object directly, and load it directly as well
  - Pickle library
- pickle.dump(file_object)
- pickle.load(file_object)

- Real World Example
  - Spike.pck