Lecture 13 — Data from Files and Web Pages

Overview

- Review of string operations
- Files on your computer
    - Opening and reading files
    - Closing
    - Writing files
- Accessing files across the web
- Parsing basics
- Parsing html

Our discussion is only loosely tied to Chapter 8 of the text.

Review — String operations often used in file parsing

Let's review and go over some very common string operations that are particularly useful in parsing files.

- Remove characters from the beginning, end or both sides of a string with `lstrip`, `rstrip` and `strip`:

```
>>> x = "red! Let's go red! Go red! Go red!"
>>> x.strip("red!")
" Let's go red! Go red! Go "
>>> x.strip("edr!")
" Let's go red! Go red! Go "
>>> x.lstrip("red!")
" Let's go red! Go red! Go red!"
>>> x.rstrip("red!")
"red! Let's go red! Go red! Go "
>>> "     Go red!       ".strip()
'Go red!'
>>> " \n   Go red!    \t   ".strip()
'Go red!'
>>> " \n   Go red!    \t   ".strip(' ')
'\n   Go red!    \t'
```

With no arguments, the `strip` functions remove all white space characters.

- Split a string using a delimiter, and get a list of strings. Whitespace is the default delimiter:

```
>>> x = "Let's go red! Let's go red! Go red! Go red!"
>>> x.split()
["Let's", 'go', 'red!', "Let's", 'go', 'red!', 'Go', 'red!', 'Go', 'red!']
>>> x.split("!")
["Let's go red", " Let's go red", ' Go red', ' Go red', '']
>>> x.split("red!")
["Let's go ", " Let's go ", ' Go ', ' Go ', '']
>>> "Let's go red! \n Let's go    red! Go red! \t Go red!".split(" ")
["Let's", 'go', 'red!', '\n', "Let's", 'go', '', '', '', 'red!', 'Go', 'red!', '\t',
'Go', 'red!']
>>> "Let's go red! \n Let's go    red! Go red! \t Go red!".split()
["Let's", 'go', 'red!', "Let's", 'go', 'red!', 'Go', 'red!', 'Go', 'red!']
```

`split` returns the strings before and after the delimiter string in a list.

- The `find` function returns the first location of a substring in a string, and returns -1 if the substring is not found. You can also optionally give a starting and end point to search from:

```
>>> x
"Let's go red! Let's go red! Go red! Go red!"
>>> x.find('red')
9
>>> x.find('Red')
-1
>>> x.find("edr!")
-1
>>> x.find('red',10)
23
>>> x.find('red',10,12)
-1
>>> 'red' in x
True
>>> 'Red' in x
False
```

Opening and Reading Files

- On to our main topic....
- Given the name of a file as a string, we can open it to read:

```
f = open('abc.txt')
```

This is the same as

```
f = open('abc.txt','r')
```

  - Variable `f` now “points” to the first line of file `abc.txt`.
  - The `'r'` tells Python we will be reading from this file — this is the default.
- One way to access contents of a file is by doing so one input line at a time. In particular,

  ```
  line = f.readline()
  ```

  reads in the next line up to and including the end-of-line character, and “advances” `f` to point to the next line of file `abc.txt`.
- By contrast,

  ```
  s = f.read()
  ```

  reads the entire **remainder** of the input file as a single string,

  - storing the one (big) string in `s`, and
  - advancing `f` to the end of the file!
- When you are at the end of a file, `f.read()` and `f.readline()` will both return the empty string: `""`.

Reading the contents of a file

- The most common way to read a file is illustrated in the following example that reads each line from a file and prints it on the screen:

  ```
  f = open('abc.txt')
  for line in f:
      print(line)
  ```

  (Of course you can replace the call to `print()` with any other processing code since each `line` is just a string!)
- You can combine the above steps into a single for loop:

  ```
  for line in open('abc.txt'):
      print(line)
  ```

Closing and Reopening Files

- The code below opens, reads, closes and reopens a file

```
f = open('abc.txt')

# Insert whatever code is need to read from the file
# and use its contents ...

f.close()
f = open('abc.txt')
```

- `f` now points again to the beginning of the file.
- This can be used to read the same file multiple times.

Example: Computing the Average Score

- We are given a file that contains a sequence of integers representing test scores, one score per line. We need to write a program that computes the average of these scores.
- Here is one solution

```
file_name = input("Enter the name of the scores file: ")
file_name = file_name.strip()    # Elminate extra white space that the user may have
typed
print(file_name)

num_scores = 0
sum_scores = 0
for s in open(file_name):
    sum_scores += int(s)
    num_scores += 1

print("Average score is {:.1f}".format( sum_scores / num_scores ))
```

- In class we will discuss:
  - Getting the file name from the user
  - The importance of using `strip`
  - The rest of the program.

Writing to a File

- In order to write to a file we must first open it and associate it with a file variable, e.g.

```
f_out = open("outfile.txt","w")
```

- The `"w"` signifies *write mode* which causes Python to completely delete the previous contents of `outfile.txt` (if the file previously existed).
- It is also possible to use *append mode*:

```
f_out = open("outfile.txt","a")
```

  which means that the contents of `outfile.txt` are kept and new output is added to the end of the file.
- Write mode is much more common than append mode.
- To actually write to a file, we use the `write` method:

```
f_out.write("Hello world!\n")
```

  - Each call to `write` passes only a **single string**.
  - Unlike what happens when using `print`, spacing and newline characters are required explicitly
  - You can use the `format` method of each string before you print it.
- You must close the files you write! Otherwise, the changes you made will not be recorded!

```
f_out.close()
```

Lecture Exercise 1:

You will have a few minutes to work on the first lecture exercise.

Opening Static Web Pages

- We can use the `urllib` module to access web pages.
- We did this with our very first “real” example:

```
import urllib.request
word_url = 'http://www.greenteapress.com/thinkpython/code/words.txt'
word_file = urllib.request.urlopen(word_url)
```

- Once we have `words_file` we can use the `read`, `readline`, and `close` methods just like we did with "ordinary" files.
- When the web page is dynamic, we usually need to work through a separate API (application program interface) to access the contents of the web site.

Parsing

- Before writing code to read a data file or to read the contents of a web page, we must know the format of the data in the file.
- The work of reading a data file or a web page is referred to as *parsing*.
- Files can be of a fixed well-known format
    - Python code
    - C++ code
    - HTML (HyperText Markup Language, used in all web pages)
    - JSON (Javascript Object Notation, a common data exchange format)
    - RDF (resource description framework)
- Often there is a parser module for these formats that you can simply use instead of implementing them from scratch.
- For code, parsers check for syntax errors.

Short tour of data formats

- Python code:
    - Each statement is on a separate line
    - Changes in indentation are used to indicate entry/exit to blocks of code, e.g. within `def`, `for`, `if`, `while` ...
- HTML: Basic structure is a mix of text with commands that are inside "tags" `< ... >`.

  Example:

```
<html>
   <head>
      <title>HTML example for CSCI-100</title>
   </head>
   <body>
      This is a page about <a href="http://python.org">Python</a>.
      It contains links and other information.
   </body>
</html>
```

- Despite the clean formatting of this example, html is in fact free-form, so that, for example, the following produces exactly the same web page:

```
<html><head><title>HTML   example for CSCI-100</title>
</head> <body> This is a page about <a
href="http://python.org">Python</a>.  It contains   links
and other   information. </body> </html>
```

- JSON: used often with Python in many Web based APIs:

```
{
   "class_name": "CSCI 1100"
   , "lab_sections" : [
          { "name": "Section 01"
             , "scheduled": "T 10AM-12PM"
             , "location": "Sage 2704"
          }
          , { "name": "Section 02"
             , "scheduled": "T 12PM-2PM"
             , "location": "Sage 2112"
          } ]

 }
```

  Similar to HTML, spaces do not matter.

  - Json is a simple module for converting between a string in JSON format and a Python variable:

```
>>> import json
>>> x = ' [ "a", [ "b", 3 ] ] '
>>> json.loads(x)
['a', ['b', 3]]
```

Parsing ad-hoc data formats - Regular tabular data

We will examine some simple formats that you may have already seen in various contexts.

- Parsing files with fixed format in each line, delimited by a character

  Often used: comma (csv), tab or space

  - One example is a file of comma separated values, Each line has a label of a soup type and a number of cans we have available:

```
chicken noodle, 2
clam chowder, 3
```

    - Here is pseudo code for processing such files:

      ```
      for each line of the file
        split into a list using the separator
        process the entries of the list into the desired form
      ```

- **Practice Problem:** write a simple parser for the soup list that returns a list of the form:

  ```
  ["chicken noodle", "chicken noodle", "clam chowder", "clam chowder", "clam chowder"]
  ```

Parsing ad-hoc data formats - Irregular tabular data

- Parsing files with one line per row of information, different columns containing unknown amount of information separated with a secondary delimiter
    - Example: Yelp data with the name of a restaurant, the lattitude, the longitude, the address, the URL, and a sequence of reviews

      Meka's Lounge|42.74|-73.69|407 River Street+Troy, NY 12180|http://www.yelp.com/biz/mekas-lounge-troy|Bars|5|2|4|4|3|4|5

  Information after column 5 are all reviews

  The address field is separated with a plus sign

  Pseudo code:

  ```
  for each line of the file
      split using the separator
      split the entry with secondary separator
      for each value in the column
          handle value
  ```

- **Practice Problem:** Write a function that returns a list of lists for a file containing Yelp data. Each list should contain the name of the restaurant and the average review.

Summary

- You should now have enough information to easily write code to open, read, write and close files on local computers.
- Once text (or HTML) files found on the web are opened, the same reading methods apply just as though the files were local. Binary files such as images require special modules.
- Parsing a file requires understanding its format, which is, in a sense, the "language" in which it is written.
- You will practice with file reading and writing in future labs and homework assignments.