Intro to Computer Science

Previous

- File I/O

```
for i in fp:
    line = fp.readline()
```

Readings	
Gaddis	• Chapter 6

Next

- File I/O
  - Review
  - CSV files

Readings	
Gaddis	• Chapter 6

Text files: the basics

```
Bob 1-212-555-1212
Mary +971-56-856-2245
Nancy +44-1523-4456
```

What the file looks like in a program that reads and presents

What the file actually looks like under the covers

Each character is a "byte"; a file is a *sequence* of bytes

```
Bob 1-212-555-1212\nMary +971-56-856-2245\nNancy +44-1523-4456
```

This file happens to 60 bytes long

- Includes letters and numbers
- Includes spaces
- Includes new lines

Methods for reading files

Method	Notes
`read()`	• Reads an entire file (in one call) • Use with caution: if the file is bigger than your memory space you'll have problems
`read(n)`	• Read n bytes (or less) from a file
`readline()`	• Read a single line from a given file • Subsequent calls read subsequent lines (maintains internal pointer) • *Of all string methods, this is likely to be the most useful*
`readlines()`	• Returns the entire file as a list of strings • Each element in the list corresponds to a line in the file • List order corresponds to line order

Example:

```
fp = open('file.txt')
single_line = fp.readline()
entire_file = fp.read()
```

Methods return empty string to signify end-of-file (EOF)

Let Python do the work

file.txt

This is the first line
This is the second line
This is the third line

Like calling readline() over and over again, on your behalf!

```
fp = open('file.txt')
for line in fp:
    sLine = line.strip()
    print(sLine)
```

iteration 1 → This is the first line

iteration 2 → This is the second line

iteration 3 → This is the third line

You have to strip

- The readline methods return a string that includes the newline terminator
- You almost always want to get rid of this!

file.txt

This is the first line
This is the second line

```
>>> fp = open('file.txt')
>>> line = fp.readline()
>>> print(line[-1])
```

The last character is probably not what you think!

```
>>> line
'This is the first line\n'
>>> print(line.strip())
This is the first line
>>>
```

The newline character is part of the string!

Writing

- Two common ways of writing files: write, print
- Assume a writable open file,

```
fp = open('file.txt', 'w')
```

methods are as follows:

Method	Usage	Notes
`write`	`fp.write('Hello file!')`	Writes the string to the file using the object method
`print`	`print('Hello file!', file=fp)`	Writes a string to a file using the print function • Advantage: casting (to string) is implicit • Disadvantage: must remember to use the "file" parameter

CSV files

- *Comma-separated value* is a file format containing delimiter separated tabular data
  - Delimiter is usually a comma
- Very common file format for interchanging data
  - Spreadsheet program writes CSV
  - Database program reads CSV
  - Underlying data is maintained
- Nice because it's plain text and it's simple

CSV files

Table (spreadsheet)

Col_1	Col_2	…	Col_n
Val_{11}	Val_{12}		Val_{1n}
Val_{21}	Val_{22}		Val_{2n}
Val_{31}	Val_{32}		Val_{3n}
…			
Val_{m1}	Val_{m2}		Val_{mn}

CSV file (plain text)

$Col_1,Col_2,\ldots,Col_n$

$Val_{11},Val_{12},\ldots,Val_{1n}$

$Val_{21},Val_{22},\ldots,Val_{2n}$

$Val_{31},Val_{32},\ldots,Val_{3n}$

…

$Val_{m1},Val_{m2},\ldots,Val_{mn}$

Move from binary, proprietary, format to something easily read by other programs… including your own!

Make friends with split and join

split(delimiter)

- String method
  - Operates on a string
  - Takes a string as a parameter
- Turns a string into a list

```
>>> s = 'col 1,col 2,col 3'
>>> s.split(',')
['col 1', 'col 2', 'col 3']
```

join(list)

- String method
  - Operates on a string
  - Takes a list as a parameter
- Joins list using a given string

```
>>> l = ['a', '24', 'val 7']
>>> s = ','
>>> s.join(l)
'a,24,value 7'
```

All values in the list must be strings!

Working with CSV files

```
fp = open('file.csv')
for line in fp:
    row = line.strip().split(',')
    # row[0] is the first element
fp.close()
```

1. strip the newline from the input
2. split the string by comma

“Use the header, Luke”

- Headers are optional
  - The CSV author should specify if the first column is a header
  - But you can generally tell by looking
- Headers are useful
  - Allow you to use a *name* when referring to a given *value*
  - You can use the headers as keys into a dictionary

CSV files

Table (spreadsheet)

Col_1	Col_2	…	Col_n
Val_{11}	Val_{12}		Val_{1n}
Val_{21}	Val_{22}		Val_{2n}
Val_{31}	Val_{32}		Val_{3n}
…			
Val_{m1}	Val_{m2}		Val_{mn}

CSV file (plain text)

Col_1,Col_2,…,Col_n ← Header!

Val_{11},Val_{12},…,Val_{1n}

Val_{21},Val_{22},…,Val_{2n}

Val_{31},Val_{32},…,Val_{3n}

…

Val_{m1},Val_{m2},…,Val_{mn}

Working with headers

- Use readline() to extract the header
- Treat the remainder of the file as if it were a sequence
- Build a dictionary using the extracted header (keys) and the current line (values)

```
fp = open('file.csv')
hdr = fp.readline()
hdr = hdr.strip().split(',')
for line in fp:
    row = line.strip().split(',')
    d = {}
    for i in range(len(hdr)):
        key = hdr[i]
        value = row[i]
        d[key] = value
```

- readline obtains a line from the file
- It's called first: it obtains the first line
- strip and split make it a list

- Create a new dictionary
- Loop through the row, obtaining the key and value from the header and row, respectively
- Since the header and row have the same number of columns, using the range of the header is okay
- Build the dictionary

An example

file.csv

```
name,age,hair,eyes
bob,23,brown,blue
john,25,red,green
```

```
fp = open('file.csv')
hdr = fp.readline()
hdr = hdr.strip().split(',')
for line in fp:
    row = line.strip().split(',')
    d = {}
    for i in range(len(hdr)):
        key = hdr[i]
        value = row[i]
        d[key] = value
```

line 0 →

```
hdr = ['age',
       'name',
       'hair',
       'eyes']
```

line 1 →

```
d = {'age': '23',
     'name': 'bob',
     'hair': 'brown',
     'eyes': 'blue'}
```

line 2 →

```
d = {'age': '25',
     'name': 'john',
     'hair': 'red',
     'eyes': 'green'}
```

Don’t forget your cast

- Note that by default all values are strings
- *You must explicitly cast the input if you want something different!*

age should probably be a number

```
d = {'age': '23',
     'name': 'bob’,
     'hair': 'brown’,
     'eyes': 'blue'}

d = {'age': '25',
     'name': 'john',
     'hair': 'red',
     'eyes': 'green'}
```