# CSV Files

# "Comma"-Separated Values Files

Say we have data in a comma-separated values file

```
$ cat capitals.dat # could be .dat, .csv, or anything
Japan,Tokyo
France,Paris
Germany,Berlin
U.S.A.,Washington, D.C
```

Can use line-by-line file reading with the split() function we saw earlier to process comma-separated value files

```
$ python
>>> capitals = {} # initialize a dictionary to hold our capitals data
>>> for line in open('capitals.dat', 'r'): # for each line in file
...     k, v = line.split(',')          # split into key and value
...     capitals[k] = v                 # add key:value to dict
...
Traceback (most recent call last):
File "<stdin>", line 2, in <module>
ValueError: too many values to unpack
```

Why didn't it work?

# CSV Separator Characters

We can troubleshoot in the Python interpreter

```
>>> for line in open('capitals.dat', 'r'):
...     print line.split(',')
...
['Japan', 'Tokyo\n']
['France', 'Paris\n']
['Germany', 'Berlin\n']
['U.S.A.', 'Washington', ' D.C\n']
```

There's a comma in Washington, D.C. that was taken as a separator So let's change the capitals.dat file to use semicolons as the separators

```
$ cat capitals.dat
Japan;Tokyo
France;Paris
Germany;Berlin
U.S.A.;Washington, D.C
```

Georgia Tech

# CSV Files in Practice

Now our capitals.dat file is readable as a "comma"-separated values file

```
>>> capitals = {}
>>> for line in open('capitals.dat', 'r'):
...     k, v = line.split(';')
...     capitals[k] = v
...
>>> capitals
{'Japan': ' Tokyo\n', 'U.S.A.': ' Washington, D.C\n', 'Germany': 'Berlin\n',
     'France': ' Paris\n'}
```

- But the values have leading whitespace and trailing \n characters from the data file

# Manually Cleaning CSV Lines

- We can make our code more robust with strip(), which removes leading and trailing whitespace and non-printing chars

```
>>> for line in open('capitals.dat', 'r'):
...     k, v = line.split(';')
...     capitals[k.strip()] = v.strip()
...
>>> capitals
{'Japan': 'Tokyo', 'U.S.A.': 'Washington, D.C', 'Germany': 'Berlin', 'France':
     'Paris'}
```

But we don't need to go to this trouble . . .

# The csv Module

The best way to process CSV files is with the csv module.

```
>>> import csv
>>> scripters = [
...     ['Perl', 'Larry Wall'],
...     ['Python', 'Guido Van Rossum'],
...     ['Ruby', 'Yukihiro Matsumoto']
... ]
>>> with open('scripters', 'wt') as fout:
...     csvout = csv.writer(fout)
...     csvout.writerows(scripters)
...
>>> ^D
$ cat scripters
Perl,Larry Wall
Python,Guido Van Rossum
Ruby,Yukihiro Matsumoto
```

- The with statement creates a context manager
- After the with block ends, the file is automatically closed

# Reading CSV Files

We can read our scripters file with

```
>>> import csv
>>> with open('scripters', 'r') as fin:
...     csvin = csv.reader(fin)
...     scripters = [line for line in csvin]
...
>>> scripters
[['Perl', 'Larry Wall'], ['Python', 'Guido Van Rossum'], ['Ruby',
'Yukihiro Matsumoto']]
>>>
```

Georgia Tech

# Column Headers in CSV Files

Use a DictReader to store the records from the CSV file in a dict.

```
>>> import csv
>>> with open('scripters', 'r') as fin:
...     csvin = csv.DictReader(fin, fieldnames=['langauge', 'creator'])
...     scripters = [line for line in csvin]
...
>>> scripters
[{'creator': 'Larry Wall', 'langauge': 'Perl'}, {'creator': 'Guido Van Rossum',
     'langauge': 'Python'}, {'creator': 'Yukihiro Matsumoto', 'langauge': 'Ruby'}]
```

Georgia Tech

# Writing Dicts to CSV Files

And we can use a DictWriter to write a CSV file with a header line.

```
>>> with open('scripters', 'w') as fout:
...     csvout = csv.DictWriter(fout, fieldnames=['langauge',
'creator'])
...     csvout.writeheader()
...     csvout.writerows(scripters)
...
>>> ^D
$ cat scripters
langauge,creator
Perl,Larry Wall
Python,Guido Van Rossum
Ruby,Yukihiro Matsumoto
```

Georgia Tech

# CSV Details

CSV files can be complex.

- Different delimiters can be used.
- Delimiter characters can appear in fields.
- Fields can be surrounded with "quotes".
- Different operating systems may use different line endings.

The CSV module handles all of these issues for you. Read the CSV module documentation to become familiar with its capabilities.

Georgia Tech