# File I/O

Michael Ernst

UW CSE 140

Winter 2013

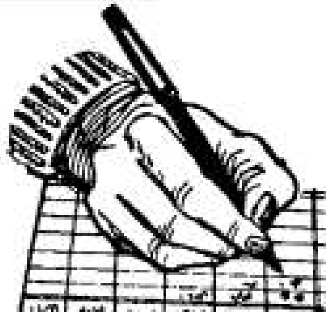
## File Input and Output

- As a programmer, when would one use a file?
- As a programmer, what does one do with a file?

## Files store information when a program is not running

Important operations:

- open a file
- close a file
- read data
- write data



## Files and filenames

- A file object represents data on your disk drive
  - Can read from it and write to it
- A filename (usually a string) states where to find the data on your disk drive
  - Can be used to find/create a file
  - Examples:
    - `"/home/mernst/class/140/lectures/file_io.pptx"`
    - `"C:\Users\mernst\My Documents\cute_cat.gif"`
    - `"lectures/file_io.pptx"`
    - `"cute_cat.gif"`

## Read a file in python

```
# Open takes a filename and returns a file.
# This fails if the file cannot be found & opened.
myfile = open("datafile.dat")

# Approach 1:
for line_of_text in myfile:
   … process line_of_text

# Approach 2:
all_data_as_a_big_string = myfile.read()
```

*Assumption: file is a sequence of lines*
*Where does Python expect to find this file (note the relative pathname)?*

# Two types of filename

- An Absolute filename gives a specific location on disk:
  **`"/home/mernst/class/140/13wi/lectures/file_io.pptx"`**
  or **`"C:\Users\mernst\My Documents\cute_cat.gif"`**
  - Starts with "/" (Unix) or "C:\" (Windows)
  - Warning: code will fail to find the file if you move/rename files or run your program on a different computer
- A Relative filename gives a location relative to the *current working directory*:
  **`"lectures/file_io.pptx"`** or **`"cute_cat.gif"`**
  - Warning: code will fail to find the file unless you run your program from a directory that contains the given contents
- A relative filename is usually a better choice

# “Current Working Directory” in Python

The directory from which you ran Python

To determine it from a Python program:

```
>>> import os   # "os" stands for "operating system"
>>> os.getcwd()
'/Users/johndoe/Documents'
```

*Can be the source of confusion:  where are my files?*

# Reading a file multiple times

You can iterate over a list as many times as you like:

```
mylist = [ 3, 1, 4, 1, 5, 9 ]
for elt in mylist:
 … process elt
for elt in mylist:
 … process elt
```

Iterating over a file uses it up:

```
myfile = open("datafile.dat")
for line_of_text in myfile:
  … process line_of_text
for line_of_text in myfile:
  … process line_of_text       # This loop body will never be executed!
```

Solution 1:  Read into a list, then iterate over it

```
myfile = open("datafile.dat")
mylines = []
for line_of_text in myfile:
  mylines.append(line_of_text)
… use mylines
```

Solution 2:  Re-create the file object (slower, but a better choice if the file does not fit in memory)

```
myfile = open("datafile.dat")
for line_of_text in myfile:
  … process line_of_text
myfile = open("datafile.dat")
for line_of_text in myfile:
  … process line_of_text
```

# Writing to a file in python

```
#  Replaces any existing file of this name
myfile = open("output.dat", "w")
```

open for **W**riting (no argument, or **"r"**, for **R**eading)

```
#  Just like printing output
myfile.write("a bunch of data")
myfile.write("a line of text\n")
```

"\n" means end of line (**N**ewline)

```
myfile.write(4)
```

Wrong; results in:

```
TypeError: expected a character buffer object
```

```
myfile.write(str(4))
```

Right. Argument must be a string