

# File I/O

BBM 101 - Introduction to Programming I

Hacettepe University  
Fall 2015

Fuat Akal, Aykut Erdem, Erkut Erdem, Vahid Garousi

# 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:
    - Linux/Mac: `"/home/rea/class/140/lectures/file_io.pptx"`
    - Windows: `"C:\Users\rea\My Documents\cute_dog.jpg"`
    - Linux/Mac: `"homework3/images/Husky.png"`
    - `"Husky.png"`

# Two Types of Filenames

- An **Absolute** filename gives a specific location on disk: `"/home/rea/class/140/14wi/lectures/file_io.pptx"` Or `"C:\Users\rea\My Documents\homework3\images\Husky.png"`
  - 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 `" images\Husky.png"`
  - 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*

# Examples

Linux/Mac: These could all refer to the same file:

```
"/home/rea/class/140/homework3/images/Husky.png"
```

```
"homework3/images/Husky.png"
```

```
"images/Husky.png"
```

```
"Husky.png"
```

Windows: These could all refer to the same file:

```
"C:\Users\rea\My Documents\class\140\homework3\images\Husky.png"
```

```
"homework3\images\Husky.png"
```

```
"images\Husky.png"
```

```
"Husky.png"
```

# “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 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()  
  
myfile.close() # close the file when done reading
```

*Assumption: file is a sequence of lines*

*Where does Python expect to find this file (note the relative pathname)?*



# Reading a File Example

```
# Count the number of words in a text file
in_file = "thesis.txt"
myfile = open(in_file)
num_words = 0
for line_of_text in myfile:
    word_list = line_of_text.split()
    num_words += len(word_list)
myfile.close()

print "Total words in file: ", num_words
```

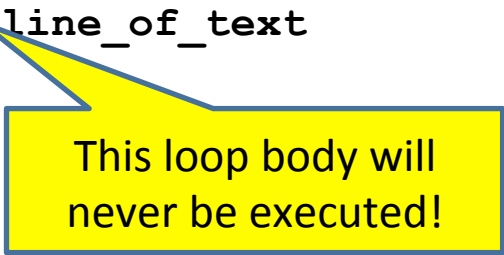
# Reading a File Multiple Times

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!

## How to read a **file** multiple times?

**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 **Writing**  
(no argument, or  
"r", for **Reading**)

# Just like printing output

```
myfile.write("a bunch of data")
```

```
myfile.write("a line of text\n")
```

"\n" means  
end of line  
(**Newline**)

Wrong; results in:

**TypeError: expected a character buffer object**

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

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

Right. Argument  
must be a string

```
myfile.close()
```

close when done  
with all writing