# File Input/Output

L435/L555

Dept. of Linguistics, Indiana University

Fall 2016

# Reading from files



A simple way to read over file contents is:

```
for line in open(filename):
    line = line.rstrip()
    print(line)
```

We'll return to this after we've looked over other ways ...

# Reading from files

```
myfile = open('cd6.pos')

text = myfile.read()

myfile.close()
```

# Read methods



- `read()`
  reads in the whole text – only advisable if the text is small
- `readline()`
  reads one line at a time (the '\n' is kept)
- `readlines()`
  reads in the whole text using `readline()` and returns a list of strings

# Iterating over file input

```
myfile = open('cd6.pos')

line = myfile.readline()
while line:
    print(line)
    line = myfile.readline()

myfile.close()
```

Note how `line` is your loop control variable

# Newlines

When you input a file's lines, they always end with `'\n'`

- `.rstrip()` (right-side stripping of white space) or `.strip()` (both-side) can handle this

```
myfile = open('cd6.pos')

line = myfile.readline()
while line:
    line = line.rstrip()
    print(line)
    line = myfile.readline()

myfile.close()
```

We'll talk more about some of this with strings

# Writing into files



Default parameter for `open` is `'r'` (read)

- `'w'` makes it writable

```
words = ['take', 'on', 'me', 'i', "'ll"
         'be', 'gone', 'in', 'a', 'day']
myfile = open('newfile.txt', 'w')

for word in words:
    myfile.write(word)

myfile.close()
```

# Write methods



- `write(<str>)`
  writes the string to a file
- `writelines(<seq>)`
  writes a list or any other sequence to a file – it does not add newlines

# Iterating over file contents



Returning to this way to iterate:

```
for line in open(filename):
    line = line.rstrip()
    print(line)
```

Note that:

- Python handles the file closing
- There's no explicit file variable

# Piping with files



The module `fileinput` increases our capabilities:

```
import fileinput
for line in fileinput.input():
    line = line.rstrip()
    print(line)
```

This program allows us to pipe input from the terminal through our python program, e.g.,

```
cat file.txt | python3 testing.py
```

It can also be run with the file as an argument to the program:

```
python3 testing.py file.txt
```

# sys



One module you can import is sys

```
import sys

sys.stdout.write('print to terminal or file\n')
sys.stderr.write('print to terminal no matter what\n')
```

If this is stored in `example.py`, try running

```
python3 example.py > output.txt
```

# More on `sys`



In general, `sys` interacts with the python interpreter

- `sys.argv` gives you a list of command line arguments
- e.g., for `python3 example.py --cool def` `sys.argv` gives:
  `['example.py', '--cool', 'def']`

If you want more options for passing in arguments, see the `argparse` module