ENGINEERING
TEXAS A&M UNIVERSITY

Week 10

File Input/Output

Learning Objectives

ENGINEERING
TEXAS A&M UNIVERSITY

File Input and Output

- Use ‘open’ and ‘close’ commands within Python to read and write external files
- Define and utilize file open designators (e.g., ‘r’, ‘w’, ‘a’)
- Write to files using write command
- Read from files using read, readline, readlines commands
- Use for or while loops to read complete or required contents of a file
- Process strings using split command

File operation: reading

Several options are available—all read in as strings:

`<string variable> = <fileID>.read()`

Will read the entire file into **one single string** (could be **REALLY** big!)

`<list variable> = <fileID>.readlines()`

Creates **a list of strings**—each line of the program is saved as a string in a new element of the list

`<list variable> = list(<fileID>)`

Creates **a list of strings**—each line of the program is saved as a string in a new element of the list

`<string variable> = <fileID>.readline()`

Reads **one line** of the file, everything until a newline character (`\n`) is found

File operation: reading

```
<string variable> = <fileID>.read()
```

Will read the entire file into one single string (could be **REALLY** big!)

```
<list variable> = <fileID>.readlines()
```

Creates a list of strings—each line of the program is saved as a string in a new element of the list

```
<list variable> = list(<fileID>)
```

Creates a list of strings—each line of the program is saved as a string in a new element of the list

These 3 read the whole file at once

File operation: .read()

```
with open('myfile.txt', 'r') as datFile:
    stringy_var = datFile.read()
```

What is `stringy_var`?

myfile.txt

1, 2
3, 4
5, 6

File operation: .read()

```
with open('myfile.txt', 'r') as datFile:
    stringy_var = datFile.read()
```

stringy_var = “1, 2/n3, 4/n5, 6”

myfile.txt

1, 2
3, 4
5, 6

File operation: .readlines()

```
with open('myfile.txt', 'r') as datFile:
    listy_var = datFile.readlines()
```

What is `listy_var`?

myfile.txt

1, 2
3, 4
5, 6

File operation: .readlines()

```
with open('myfile.txt', 'r') as datFile:
    listy_var = datFile.readlines()
```

```
listy_var = ['1, 2\n', '2, 4\n', '4, 5']
```

myfile.txt

1, 2
3, 4
5, 6

File operation: list()

```
with open('myfile.txt', 'r') as datFile:
    listy_var = list(datFile)
```

What is `listy_var`?

myfile.txt

1, 2
3, 4
5, 6

File operation: list()

```
with open('myfile.txt', 'r') as datFile:
    listy_var = list(datFile)
```

```
listy_var = ['1, 2\n', '2, 4\n', '4, 5']
```

myfile.txt

1, 2
3, 4
5, 6

File operation: reading

This one reads one line at a time…

```
<string variable> = <fileID>.readline()
```

Reads one line of the file, everything until a newline character (`\n`) is found

File operation: reading

This one reads one line at a time…

so why is it the most used?

```
<string variable> = <fileID>.readline()
```

Reads one line of the file, everything until a newline character (`\n`) is found

File operation: .readline()

```
with open('myfile.txt', 'r') as datFile:
    stringy_var = datFile.readline()
```

What is `stringy_var`?

myfile.txt

1, 2
3, 4
5, 6

File operation: .readline()

```
with open('myfile.txt', 'r') as datFile:
    stringy_var = datFile.readline()
```

```
stringy_var = '1, 2\n'
```

myfile.txt

1, 2
3, 4
5, 6

File operation: .readline()

```
with open('myfile.txt', 'r') as datFile:
    stringy_var = datFile.readline()
    stringy_var = datFile.readline()
```

`stringy_var = ?`

myfile.txt

1, 2
3, 4
5, 6

File operation: .readline()

```
with open('myfile.txt', 'r') as datFile:
    stringy_var = datFile.readline()
    stringy_var = datFile.readline()
```

```
stringy_var = '3, 4\n'
```

myfile.txt

1, 2
3, 4
5, 6

Reading multiple lines

Often we want to process an entire file, and read all (or many) lines of the same format (e.g., data)

We can use the for loop!

```
for <lineID> in <fileID>:
    #Do stuff with the string lineID
```

It's structured the same, but for each iteration <lineID> is a line in the file.

File operation:

```
with open('myfile.txt', 'r') as datFile:
    for stringy_var in datFile:
        parts = stringy_var.split(',')
        print(int(parts[0])+int(parts[1]))
```

?

myfile.txt

1, 2
3, 4
5, 6

Reading multiple lines

Two versions that essentially work the same way:

```
### OPTION 1
for next_line in myfile:
    #Do stuff with the string next_line
```

```
### OPTION 2
next_line = myfile.readline()
while next_line != '':
    #Do stuff with the string next_line
    next_line = myfile.readline()
```

CSV Files:

Python has a csv module to help us read from and write to CSV files:

```
import csv
```

Then, instead of working directly from the file we open, we work with a thing the module creates:

```
with open('sheetyfile.csv', 'r') as datFile:
    sheet_reader = csv.reader(datFile, delimiter=',')
    for row_var in sheet_reader:
        print(row_var)
```

Reading CSV Files:

```
import csv
with open('sheetyfile.csv', 'r') as datFile:
    sheet_reader = csv.reader(datFile, delimiter=',')
    for row_var in sheet_reader:
        print(row_var)
```

sheetyfile.csv

1, 2
3, 4
5, 6

Reading CSV Files:

```
import csv
with open('sheetyfile.csv', 'r') as datFile:
    sheet_reader = csv.reader(datFile, delimiter=',')
    for row_var in sheet_reader:
        print(row_var)
```

```
['1',' 2']
['3',' 4']
['5',' 6']
```

sheetyfile.csv

1, 2
3, 4
5, 6

Writing CSV Files:

In Windows, if you don't do this you'll get an extra blank line between each written line

```
with open('sheetyfile.csv', 'w', newline='') as datFile:
    sheet_writer = csv.writer(datFile, delimiter=',')
    row_var = ['100', '50', '75']
    sheet_writer.writerow(row_var)
    tupperware = ['-5', '6', 'squirrel']
    sheet_writer.writerow(tupperware)
```

sheetyfile.csv

100,50,75
-5,6,squirrel

File location

By default, files are created in the same directory as the .py file.

- To specify a different directory, provide the location as part of the file name when opening.

```
#Mac OS X
infile = open('data/data.txt', 'r')
```

```
#Windows
infile = open('data\\data.txt', 'r')
```

A double backslash (\\) represents a single backslash inside a string (remember the 'escape character'?)

File location

By default, files are created in the same directory as the .py file.

- To specify a different directory, provide the location as part of the file name when opening.

```
#Mac OS X
infile = open('data/data.txt', 'r')
```

```
#Windows
infile = open('data\\data.txt', 'r')
```

A double backslash (\\) represents a single backslash inside a string (remember the 'escape character'?)

Example in Windows:

```
with open('C:\\Users\\Oblivious\\PycharmProjects\\ENGR_102\\temp2.csv','w') as datFile:
```