Week 10

Pre-Lecture Slides: File Input/Output and String Processing

Last week: one method to open a file

```
with open("<File Name>", "<designator>") as <fileID>:
    # Stuff to do while file is open
    # More stuff
# Now file is closed
```

- When you finish with the indented portion, the file is automatically closed.
- The fileID variable can be used to refer to the file within the indented portion of the code

Now, another method

```
<fileID> = open("<File Name>", "<designator>")
# Stuff to do while file is open
# More stuff
<fileID>.close()   # Now file is closed
```

- The file does not automatically close; you must remember to do so.
- This insures the file is left in a valid condition
- The fileID variable can be used to refer to the file *only* before it is closed.

Same designators as before:

`r`	Reading (we will read data from an existing file)
`w`	Writing (we will write data to a new file, or overwrite an existing file)
`a`	Appending (we will append data to an existing file)
`rb`, `wb`, `ab`	Read/write/append BINARY data *(note: we won't in this class)*
`r+`	We will read from AND write to the file
<nothing>	If there is no mode designator, then 'r' is assumed

Examples

Open a file named Measurements.dat for reading, and assign the variable `myfile` as the file id:

```
myfile = open('Measurements.dat', 'r')
```

Open a file named Results.out so that it can be written to, and assign the variable `output_file` as the file id:

```
output_file = open('Results.out', 'w')
```

Open a file named data for reading and writing in binary, and assign the variable `df` as the file id:

```
df = open('data', 'rb+')
```

The two alternatives:

```
# OPTION 1
myfile = open("data.dat",r+)
#Do stuff with myfile - read/write
myfile.close()
```

```
# OPTION 2
with open("data.dat",r+) as myfile:
    #Do stuff with myfile - read/write
```

Advantages and Disadvantages

Think of advantages and disadvantages of each method

1

```
myfile = open("data.dat",r+)
#Do stuff with myfile - read/write
myfile.close()
```

2

```
with open("data.dat",r+) as myfile:
    #Do stuff with myfile - read/write
```

Write these down and bring them to class.

String Processing

- Since Python inputs data as strings, we find ourselves often needing to "break" these strings into parts.
- One useful operation for strings is the "split" method.
  - Converts a string into a list of strings
  - Programmer specifies the separator
  - Everything found between separators becomes a new element in the list.

String splitting

Format:

```
<list variable> = <string variable>.split(<thing to split on>)
```

We start with a string variable

String splitting

Format:

```
<list variable> = <string variable>.split(<thing to split on>)
```

Then we put .split()

ENGINEERING
TEXAS A&M UNIVERSITY

String splitting

Format:

```
<list variable> = <string variable>.split(<thing to split on>)
```

Inside the parentheses is what we want to use to decide how to split up the string. This is a string.

String splitting

Format:

```
<list variable> = <string variable>.split(<thing to split on>)
```

The result is a list of strings

ENGINEERING
TEXAS A&M UNIVERSITY

Example

```
s = "1,2,3,4"
elems = s.split(',')
print(elems)
```

Console

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

Example: Split a Date

Say we have a date, in a string of the form: month/day/year, and we want to get three variables, one with the month, one with the day, one with the year. How would we do that?

Example: Split a Date

```python
date = "10/31/2019"
parts = date.split('/')
month = parts[0]
day = parts[1]
year = parts[2]
print("Day:",day,"Month:",month,"Year:",year)
```

Console

```
Day: 31 Month: 10 Year: 2019
```

Example: Split a Paragraph

How can you take a paragraph, written by a user, and print it one word per line to a txt file?

E.g., `input_string = ‘I do not like them, Sam-I-am. I do not like green eggs and ham.’`

```
I
do
not
like
...
```

String Stripping

Since reading lines often results in \n characters at the end of lines, we often need a simple way to remove them. *(Note: the last option is most common for our use this week.)*

Format:

`<variable> = <string variable>.strip()`	Removes any \n and \r from left <u>and</u> right side
`<variable> = <string variable>.lstrip()`	Removes any \n and \r from left side
`<variable> = <string variable>.rstrip()`	Removes any \n and \r from right side