# CMSC201
# Computer Science I for Majors

## Lecture 10 – File I/O

Prof. Katherine Gibson

## Last Class We Covered

- Using **`while`** loops
  - Syntax
  - Using them for interactive loops
- Two different ways to mutate a list
  - **`append()`** and **`remove()`**
- Nested loops
- Two-dimensional lists (lists of lists)

## Any Questions from Last Time?

## Today’s Objectives

- To learn about escape sequences
  - Why we need them
  - How to use them
- To be able to
  - Open a file
  - Read in its data

## Escape Sequences

## "Misbehaving" **`print()`** Function

- There are times when the **`print()`** function doesn't output exactly what we want

```
>>> print("I am 5 feet, 4 inches")
I am 5 feet, 4 inches
>>> print("I am 5'4"")
  File "<stdin>", line 1
    print("I am 5'4"")
                    ^
SyntaxError: EOL while scanning string literal
```

## Special Characters

- Just like Python has special keywords…
  - **`for`**, **`int`**, **`True`**, etc.

- It also has special characters
  - single quote (`'`), double quote (`"`), etc.

## Backslash: Escape Sequences

- The backslash character (\) is used to "*escape*" a special character in Python
  - Tells Python not to treat it as special

- The backslash character goes <u>in front</u> of the character we want to "escape"

```
>>> print("I am 5'4\"")
I am 5'4"
```

## Using Escape Sequences

- There are three ways to solve the problem of printing out our height using quotes

```
>>> print("I am 5'4\"")
I am 5'4"
>>> print('I am 5\'4"')
I am 5'4"
>>> print("I am 5\'4\"")
I am 5'4"
```

# Using Escape Sequences

- There are three ways to solve the problem of printing out our height using quotes

```
>>> print("I am 5'4\"")
I am 5'4"
>>> print('I am 5\'4"')
I am 5'4"
>>> print("I am 5\'4\"")
I am 5'4"
```

escape double quotes (using " for the string)

escape single quotes (using ' for the string)

escape both single and double quotes (works for both ' and ")

# Common Escape Sequences

| Escape Sequence | Purpose |
|---|---|
| `\'` | Print a single quote |
| `\"` | Print a double quote |
| `\\` | Print a backslash |
| `\t` | Print a tab |
| `\n` | Print a new line (“enter”) |
| `"""` | Allows multiple lines of text |

`"""` is not really an escape sequence, but is useful for printing quotes

## Escape Sequences Example

```
tabby_cat = "\tI'm tabbed in."
print(tabby_cat)
     I'm tabbed in.
```

**\t** adds a tab

```
persian_cat = "I'm split\non a line."
print(persian_cat)
I'm split
on a line.
```

**\n** adds a newline

```
backslash_cat = "I'm \\ a \\ cat."
print(backslash_cat)
I'm \ a \ cat.
```

**\\** adds a single backslash

## Escape Sequences Example

```
fat_cat = """
I'll do a list:
\t* Cat food
\t* Fishies
\t* Catnip\n\t* Grass
"""
print(fat_cat)
```

```
I'll do a list:
        * Cat food
        * Fishies
        * Catnip
        * Grass
```

# Escape Sequences Example

```
fat_cat = """
I'll do a list:
\t* Cat food
\t* Fishies
\t* Catnip\n\t* Grass
"""
print(fat_cat)
```

```

I'll do a list:
        * Cat food
        * Fishies
        * Catnip
        * Grass
```

when using triple quotes (`"""`), the times you hit "enter" inside the string will print as newlines

# Escape Sequences Example

```
fat_cat = """
I'll do a list:
\t* Cat food
\t* Fishies
\t* Catnip\n\t* Grass
"""
>>> print(fat_cat)

I'll do a list:
        * Cat food
        * Fishies
        * Catnip
        * Grass
```

**\t** puts in a tab

**\n** adds a newline

## How Python Handles Escape Sequences

- Escape sequences look like two characters to us
- Python treats them as a single character

```
>>> example1 = "dog\n"
>>> example2 = "\tcat"
```

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| d | o | g | \n |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| \t | c | a | t |

## File Input/Output

## Why Use Files?

- Until now, the Python programs you've been writing are pretty simple for input/output
  - User types input at the keyboard
  - Results (output) are displayed in the console
- This is fine for short and simple input…
  - But what if we want to average 50 numbers, and mess up when entering the 37th one?
  - Start all over???

## What is File I/O?

- One solution is to read the information in from a file on your computer
  - You could even write information to a file
- This process is called *File I/O*
  - "I/O" stands for "input/output“
  - Python has built-in functions that make this easy

## File I/O Example Usage

- "Read" in a file using a word processor
  - File opened
  - Contents read into memory (RAM)
  - File closed
  - IMPORTANT: Changes to the file are made to the copy stored in memory, <u>not</u> the original file on the disk

## File I/O Example Usage

- “Write” a file using a word processor
  - (Saving a word processing file)
  - Original file on the disk is reopened in a mode that will allow writing
    - This actually erases the old contents!
  - Copy the version of the document stored in memory to the original file on disk
  - File is closed

## File Processing

- In order to do interesting things with files, we need to be able to perform certain operations:
  - Associate an external file with a program object
    - Opening the file
  - Manipulate the file object
    - Reading from or writing to the file object
  - Close the file
    - Making sure the object and file match

## Syntax: Opening a File

## Syntax for `open()` Function

```
myFile = open(FILE_NAME [, ACCESS_MODE][, BUFFERING])
```

**`FILE_NAME`**

- This argument is a string the contains the name of the file you want to access
  - `"input.txt"`
  - `"numbers.dat"`
  - `"roster.txt"`

UMBC

AN HONORS UNIVERSITY IN MARYLAND

# Syntax for **`open()`** Function

```
myFile = open(FILE_NAME [, ACCESS_MODE][, BUFFERING])
```

**`ACCESS_MODE`** (optional argument)

- This argument is a string that determines which of the modes the file is to be opened in
  - **`"r"`** (open for reading)
  - **`"w"`** (open for writing)
  - **`"a"`** (open for appending)

# Syntax for **`open()`** Function

```
myFile = open(FILE_NAME [, ACCESS_MODE][, BUFFERING])
```

**`BUFFERING`** (<u>optional</u> argument)

- This argument is an integer that specifies to desired buffer size for the file
  - 0 (unbuffered)
  - 1 (line buffered)
  - >1 (buffer of approximately that size in bytes)

we won't be using buffering much (if at all) in this class

# Examples of Using `open()`

- In general, we will use commands like:

```
testFile = open("scores.txt")
dataIn   = open("old_stats.dat")
dataOut  = open("stats.dat", "w")
```

- We will ignore the optional buffering argument

an example input file

```
scores.txt
2.5 8.1  7.6 3.2 3.2
3.0 11.6 6.5 2.7 12.4
8.0 8.0  8.0 8.0 7.5
```

## File Processing: Reading

## Using File Objects to Read Files

```
myFile = open("myStuff.txt")
```

- This line of code does three things:
  1. Opens the file “myStuff.txt”
  2. In the “reading” mode (which is the default)
  3. Assigns the opened file to the variable **`myFile`**
     - **`myFile`** is a variable of type file object
- Once the file is open, we can start reading it

# Three Ways to Read a File

- There are three different ways to read in a file:

1. Read the whole file in as one big long string
   `myFile.read()`
2. Read the file in one line at a time
   `myFile.readline()`
3. Read the file in as a list of strings (each is one line)
   `myFile.readlines()`

# Entire Contents into One String

```
>>> info = open("hours.txt")
>>> wholeThing = info.read()
>>> wholeThing
'123 Susan 12.5 8.1 7.6 3.2\n456 Brad 4.0
11.6 6.5 2.7 12\n789 Jenn 8.0 8.0 8.0 8.0
7.5\n'
```

it's literally one giant string!

our input file

**hours.txt**
```
123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

# Entire Contents into One String

```
>>> info = open("hours.txt")
>>> wholeThing = info.read()
>>> wholeThing
'123 Susan 12.5 8.1 7.6 3.2\n456 Brad 4.0
11.6 6.5 2.7 12\n789 Jenn 8.0 8.0 8.0 8.0
7.5\n
```

it's literally one giant string!

notice that escape sequence (**\n**) is being printed, instead of the text starting on a new line

our input file

```
hours.txt
123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

## One Line at a Time

```
>>> info = open("hours.txt")
>>> lineOne = info.readline()
>>> lineOne
'123 Susan 12.5 8.1 7.6 3.2\n'
>>> lineTwo = info.readline()
'456 Brad 4.0 11.6 6.5 2.7 12\n'
```

there's actually an easier way to do this... can you guess what it is?

(we'll show you soon)

our input file

```
hours.txt
123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

## As a List of Strings

```
>>> info = open("hours.txt")
>>> listOfLines = info.readlines()
>>> listOfLines
['123 Susan 12.5 8.1 7.6 3.2\n',
 '456 Brad 4.0 11.6 6.5 2.7 12\n',
 '789 Jenn 8.0 8.0 8.0 8.0 7.5\n']
```

our input file

```
hours.txt
123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

## Using `for` Loops to Read in Files

- Remember, `for` loops are great for iterating
- With a list, the `for` loop iterates over…
  - Each element of the list (in order)
- Using a `range()`, the `for` loop iterates over…
  - Each number generated by the range (in order)
- And with a file, the `for` loop iterates over…
  - Each line of the file (in order)

## A Better Way to Read One Line at a Time

- Instead of reading them manually, use a **`for`** loop to iterate through the file line by line

```
>>> info = open("hours.txt")
>>> for eachLine in info:
...     print(eachLine)
...
123 Susan 12.5 8.1 7.6 3.2

456 Brad 4.0 11.6 6.5 2.7 12

789 Jenn 8.0 8.0 8.0 8.0 7.5
```

## A Better Way to Read One Line at a Time

- Instead of reading them manually, use a **`for`** loop to iterate through the file line by line

```
>>> info = open("hours.txt")
>>> for eachLine in info:
...       print(eachLine)
...
123 Susan 12.5 8.1 7.6 3.2

456 Brad 4.0 11.6 6.5 2.7 12

789 Jenn 8.0 8.0 8.0 8.0 7.5

```

why are there all these empty lines???

now that we're calling **`print()`**, the **`\n`** is printing out as a new line

## Whitespace

## Whitespace

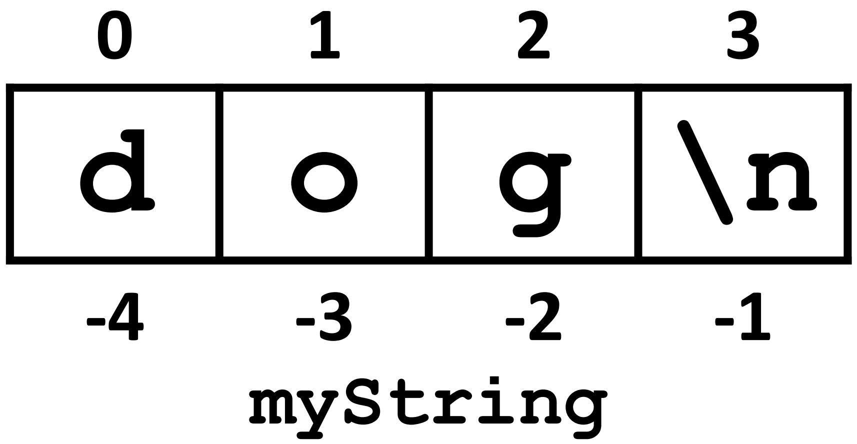
- Whitespace is any “blank” character, that represents space between other characters
- For example: tabs, newlines, and spaces
  `"\t"` `"\n"` `" "`

- When we read in a file, we can get whitespace
  - Sometimes, we don’t want to keep it

## Removing the Newline from the End

- To remove the escaped newline sequence (**\n**) from a string we read in, we can use slicing

```
myString = myString[:-1]
```



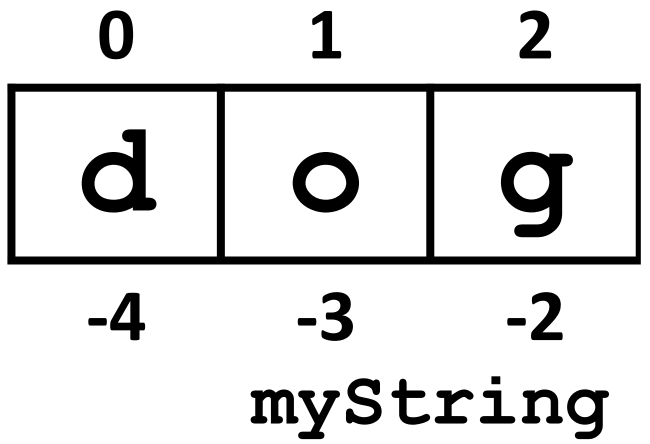
# Removing the Newline from the End

- To remove the escaped newline sequence (**`\n`**) from a string we read in, we can use slicing

```
myString = myString[:-1]
```

don't remove anything from the beginning

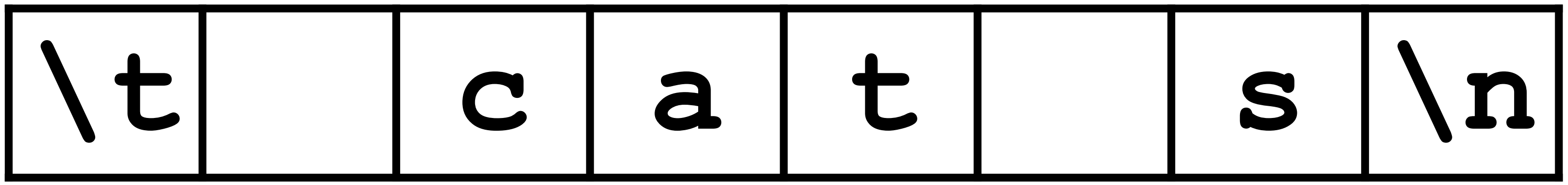
just remove the very last character



# Removing Whitespace

- To remove all whitespace from the <u>start and end</u> of a string, we can use **`strip()`**

```
spacedOut = spacedOut.strip()
```

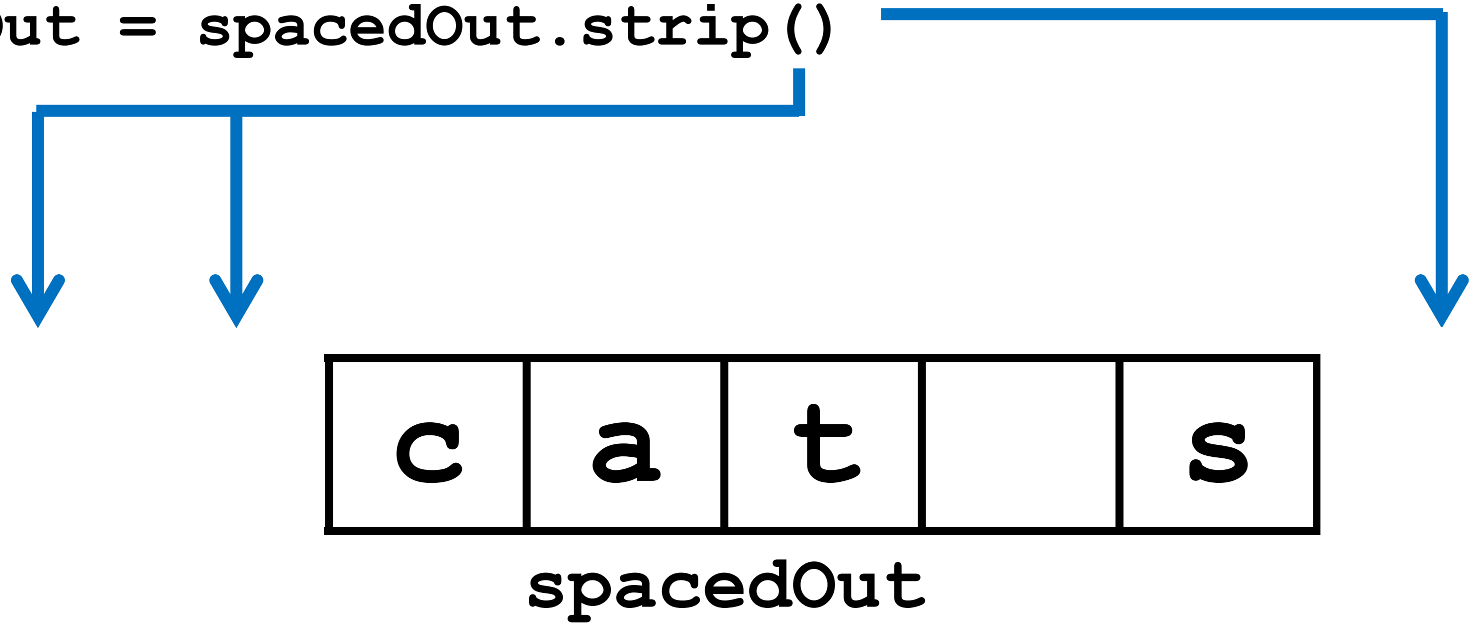


spacedOut

## Removing Whitespace

- To remove all whitespace from the <u>start and end</u> of a string, we can use **`strip()`**

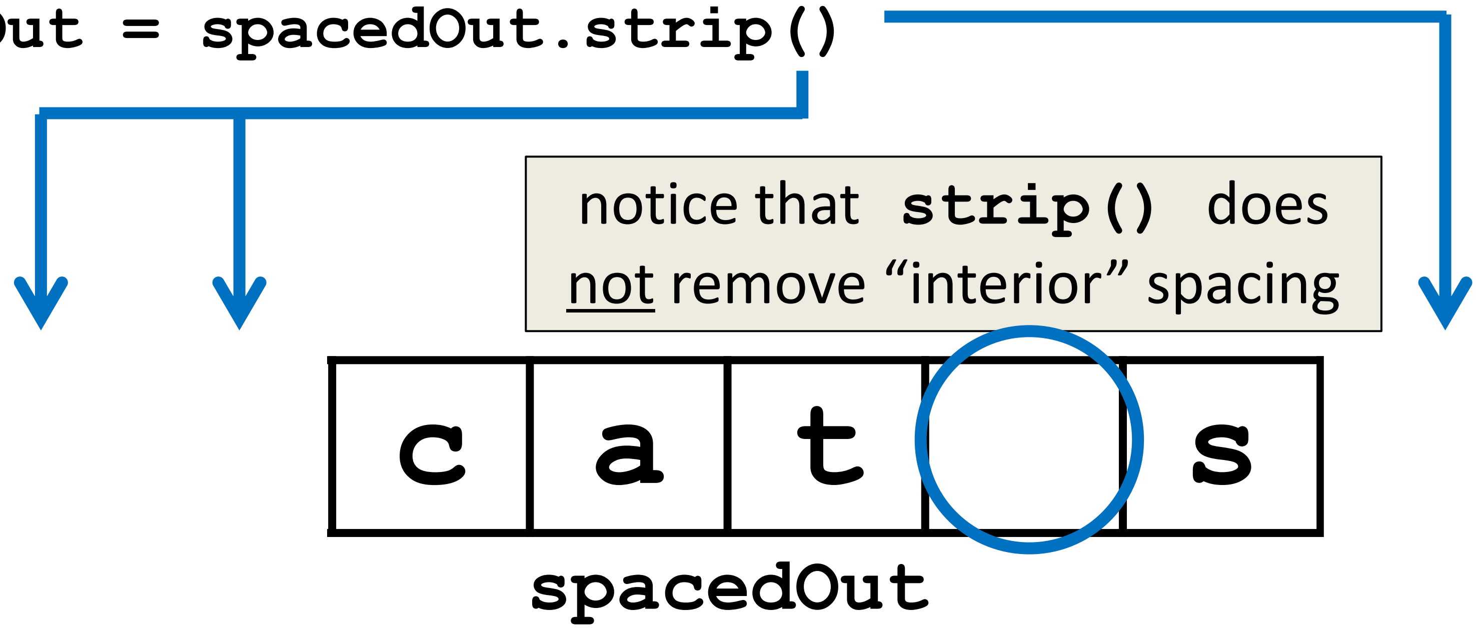
```
spacedOut = spacedOut.strip()
```



# Removing Whitespace

- To remove all whitespace from the start and end of a string, we can use **`strip()`**

```
spacedOut = spacedOut.strip()
```



## Miscellaneous (and Exercises!)

## Getting a Filename from a User

- Instead of putting the filename straight in the code, we can ask the user for the filename
- Save their response in a variable, and call the **`open()`** function with it

```
# printfile.py
#     Prints a file to the screen.

def main():
    fname = input("Enter filename: ")
    infile = open(fname, 'r')
    data = infile.read()
    print(data)

main()
```

# Exercise: Jabberwocky

- Write a program that goes through a file and reports the longest line in the file

Example Input File: **caroll.txt**

```
Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,
Beware the JubJub bird and shun
the frumious bandersnatch.
```

Example Output:

```
>>> longest.py
longest line = 42 characters
the jaws that bite, the claws that catch,
```

## Jabberwocky Solution

```
def main():
    input = open("carroll.txt")
    longest = ""
    for line in input:
        if len(line) > len(longest):
            longest = line

    print("Longest line =", len(longest))
    print(longest)
main()
```

## Announcements

- (Pre) Lab 5 has been released on Blackboard
  - Future ones will be available the weekend prior
- Homework 4 is out
  - Due by Tuesday (Oct 6th) at 8:59:59 PM
- Homework 1 re-grade and re-submit petitions must be made to your TA before Friday @ 3 PM