Week 6 - Wednesday

# COMP 1800

# Last time

- What did we talk about last time?
- Exam!
- Before that:
  - Dictionaries
  - Review for Exam 1

# Questions?

# Assignment 4

# Exam 1 Post Mortem

# Files

# Big data

- We started with programs that had no input
  - All the data was hard-coded into the program
- We started prompting the user for information using the **`input()`** command
- Now we will allow our programs to access data from anywhere
  - Files
  - Online

# Files

- A file is a series of bytes stored on a computer
- Usually, a file is stored on a hard drive or SSD
- It's **persistent**, so it exists after a program is done running
- Files allow us to do input that would be tedious by hand
- Files also allow us to do output that is too long to read in one go

# Text Files

- Files include:
  - Images like .jpg or .png files
  - Audio files like .flac or .mp3 files
  - Movies like .mp4 files
  - Office files like spreadsheets and this PowerPoint
- We're going to start with simple **text files**
  - They only contain unformatted text
  - They're human-readable
  - Programs like Notepad can read them

# Opening a file

- We can open a text file with the **`open()`** function
- It takes two string arguments:
  - File name
  - Mode (reading: **`'r'`**, writing: **`'w'`**, or append: **`'a'`**)
- Append is like writing, except that append writes to the end of the file while writing destroys whatever used to be in the file

```
file = open('data.txt', 'r')
```

# Closing a file

- After you open a file and read from it or write to it, you need to close it
- Files take up resources on the system, so having too many open files is wasteful
- There can be issues with reading or writing a file that another program has open
- Some of your data might get lost if you're writing to a file and forget to close it before your program ends
- To close a file, call the file reference's **`close()`** method

```
file.close()
```

# Using with/as

- Because it's annoying to have to remember to close a file, Python has syntax that makes it unnecessary
- This alternative style starts with the keyword **with**
- Then, code using the file is in an indented block

```
with open('data.txt', 'r') as file:
  # Do the reading you want to do with file
  # Do some calculations
```

- The file is automatically closed after the indented block

# File processing

- Files are often read one line at a time
- Python lets us iterate over the file as if it were a list of lines

```
with open('alice.txt', 'r') as story:
  for line in story:
    print(line)
```

# split()

- We introduced **split()** earlier
- It allows us to break a string into a list of strings
  - With no argument, it will break up the strings based on white space

```
phrase = 'It was a stark and dormy night'
words = phrase.split()
# ['It', 'was', 'a', 'stark', 'and', 'dormy', 'night']
```

  - With a single character as an argument, it will break up strings based on that

```
text = 'potatoes:tomatoes:Barbados'
items = text.split(':')
# ['potatoes', 'tomatoes', 'Barbados']
```

# Using `split()` with files

- Each line of a file might contain several data fields.
- The **`split()`** method can be used to break a line into a list of fields
- For example, a comma-separated-value (CSV) file divides values with commas

```
with open('data.csv', 'r') as data:
  for line in data:
    for column in line.split(','):
        print(column)
```

# File methods

- Here are a few useful file methods that can be used for reading or writing individual lines or characters:
  - **`read()`** Reads entire file as a single string
  - **`read(n)`** Reads **`n`** characters from file as a string
  - **`readline()`** Reads the next line of the file
  - **`readline(n)`** Reads **`n`** characters from the next line of the file
  - **`readlines()`** Reads all the lines of the file as a list of strings
  - **`readlines(n)`** Reads **`n`** lines of the file as a list of strings
  - **`write(s)`** Write the string **`s`** to the file
- Each of these file methods would be called on an open file reference:

```
with open('data.txt', 'r') as data:
  firstLine = data.readline()
```

# Example file

- We have a file called **`starbucks.csv`** that has information about North American Starbucks stored in a CSV format with the following fields:
  - Longitude (*x* location)
  - Latitude (*y* location)
  - Name (in quotes)
  - Address (in quotes)
- Available here: https://introcs.cs.princeton.edu/java/data/starbucks.csv

# Longitudes and latitudes

- Let's find the maximum and minimum longitudes and latitudes
- Algorithm:
  - Open the file for reading
  - Initialize our variables for max and min longitude and latitude
  - Loop over all the lines in the file
    - Split each line
    - Convert the first value in the split-up line to a decimal value for longitude
    - Convert the second value in the split-up line to a decimal value for latitude
    - Update maximums and minimums
  - Print out maximums and minimums

# Drawing Starbucks locations

- We can draw the locations we found in the previous example with turtle graphics
- All we have to do is go to the (longitude, latitude) location and draw a dot (with a size of 3, so that the dot is small)
- A few suggestions that will make the output nicer:
  - Get a screen object and set the world coordinates to have a min *x* of -180, min *y* of 0, max *x* of 0, and max *y* of 90
  - Put the turtle's tail up, set its speed to 0, and hide it
  - Call **`turtle.tracer(100)`** so that it only updates the screen every 100 draw operations, making things much faster

# Upcoming

# Next time…

- **`while`** loops
- Work time for Assignment 4

# Reminders

- Read section 5.3 of the textbook
- Finish Assignment 4
  - **Due Friday before midnight!**