Week 6 - Friday

COMP 1800

Last time

- What did we talk about last time?
- Exam 1 post mortem
- Files

Questions?

Assignment 4

File Review

Opening a file 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)
```

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)
```

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

`while` Loops

`while` loop

- The simplest loop in Python is the **`while`** loop
- It looks similar to an **`if`** statement
- The difference is that, when you get to the end of the **`while`** loop, it jumps back to the top
- If the condition in the **`while`** loop is still **`True`**, it executes the body of the loop again

Anatomy of a `while` loop

```
while condition :
```

A whole bunch of statements

```
    statement1
    statement2
    …
    statementn
```

`while` loop syntax

- Just like an **`if`**-statement
  - The condition should be a Boolean
  - The colon after the condition is required
  - All the statements inside the **`while`** loop must be indented

Rules for `while`

- The **`while`** loop executes each statement one by one
- When execution gets to the bottom, it jumps to the top
- If the condition is still **`True`** (i.e., **`i < 100`**), it repeats the loop
- In Python, some tasks can only be done with a **`while`** loop because we don't know how many times they will repeat

Iteration

- The aspect of computing that provides the most power and also represents the most risk
- There are different ways to look at iteration:
  - Definite vs. indefinite
    - In definite iteration (**`for`**) you know at compile time how many iterations there will be (maybe related to a variable, not necessarily constant)
    - In indefinite iteration (**`while`**) you may not know until the end of the final iteration (at runtime) when the loop will stop
  - Explicit vs. implicit
    - Explicit iteration uses control constructs like **`for`** and **`while`**
    - Implicit iteration uses recursion, which we won't cover until chapter 9

Comparison of for and while loops

```
for n in range(10):
    print(n)
```

```
n = 0
while n < 10:
    print(n)
    n += 1
```

```
for line in file:
    print(line)
```

```
line = file.readline()
while len(line) > 0:
    print(line)
    line = file.readline()
```

Observations

- A loop must make progress towards termination
  - With **`for`** loops this happens automatically
  - With **`while`** loops it is the responsibility of the programmer
  - The exit condition must include one or more variables
  - The values of those variables must be modified in the body of the loop in such a way as to move the test closer to being **`False`**
- A **`while`** loop must be primed
  - The variables in the exit condition must be initialized *prior* to reaching the loop

Quiz

Work Time

Upcoming

Next time…

- **`while`** loop examples
- List comprehensions
- Reading data from the Internet

Reminders

- Keep reading section 5.3
- Finish Assignment 4
  - **Due tonight before midnight!**
- Start Assignment 5
  - Due next Friday before midnight