Didn't finish or missed lab? As stated on the lab handout, "If you do not finish during the lab, you have until the beginning of lab next week to finish it. You should show it to your lab instructor at the beginning of that next lab."

Lab handouts:

Contact your lab instructor for lab issues; email addresses here:
http://www.cs.cornell.edu/Courses/cs1110/2014sp/about/times.php

Readings and references for today and next time:
Strings: 8.1-8.2; 8.4; 8.8-8.9. Don't worry about the "method" terminology (yet).
Python functions: 3.1-3.6; 4.9
Operations for Getting Data from Strings: Indexing and slicing

- **s = 'abc d'**
  
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- Access portions with [].
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes IndexError: string index out of range
  - s[0:2] is 'ab' (excludes c). Everyone forgets this at least once.
  - s[2:] is 'c d'. Called “string slicing”
Other Ways to Get Data from Strings

- `s_1 in s_2`
  - Tests if `s_1` “a part of” `s`
- `len(s)`
  - Value is # of chars in `s`
- `s_1.index(s_2)`
  - Position of the first instance of `s_2` in `s_1`
- `s_1.count(s_2)`
  - Number of times `s_2` appears inside of `s_1`
- `s.strip()`
  - A copy of `s` with white-space removed at ends
- `s_1.strip(s_2)` removes the characters in `s_2` from ends of `s_1` if there are any.

A ' #' marks a comment for the reader (including the code's author). Python ignores the rest of the line.

```
s = 'abracadabra'

# the following all evaluate to True
'a' in s
'cad' in s
not('foo' in s)
len(s) == 11
s.index('a') == 0
s.index('rac') == 2
s.count('a') == 5
' cs1110 '.strip() == 'cs1110'
s.strip('a') == 'bracadabr'
```
A String Puzzle (Extraction Practice)

Given: variable `data` contains a string with at least two '','s.

Ex: `data="LL, '14, 1-800-OPYTHON, 1-555-TYPHOON"`

Goal: give an expression for the part of the string after the 2\(^{\text{nd}}\) ','. (How can we use the index operation?)

# (1) Store in variable `j` the index of the first comma.

# (2) Store in variable `tail` the part of `data` starting after \textit{after} \textbf{j}

# (3) Give an expression for the part of `tail` starting after `',"`
**Given:** info contains a comma-separated string with last name, difficulty, execution, and penalty.

- **Example:** info = 'RAISMAN, 6.7, 9.1,0'

**Goal:** store the difficulty as a string, with no extra spaces or punctuation, in variable df

Where, in the following sequence of commands, does the first (conceptual) error occur?

A: startcomma = info.index(',,')
B: tail = info[startcomma+1:]
C: endcomma = tail.index(',,')
D: df = tail[:endcomma-1].strip()
E: this sequence achieves the goal
The Komodo Edit Interface

- Current working directory
- Tabs for open files
- "minimap" of whole file
- Current active file
- Line numbers
""" Demonstrates putting a sequence of commands into a ...."""

```python
x = 1+2
x = 3*x
```

Note: Unlike with the command prompt, evaluating an expression produces nothing when a Python file (script, module) is run. The author probably wanted `print x` here.
Start Python in Your Script's Directory!

Module you want is in this folder

Have to navigate to folder **BEFORE** running Python

Module you want is in this folder
Function Calls

- Python supports expressions with math-like functions
- Function expressions have the form `fun(x, y, ...)`

Examples of built-in functions:
- Numerical functions: `round(number)`, `pow(base, exp)`
- Getting user input: `raw_input()`
- Help function: `help()`
Using a Function From Another File
(such files are called *modules*)

Example: what if we want 'Raisman', not 'RAISMAN'?

*Lucky us:* someone has written a module (file) `string` that contains a function `capwords`.

```python
import string  # Tell Python to access this module
name = info[:info.find(',')]]  # name is 'RAISMAN'
print string.capwords(name)  # output is 'Raisman'
```
Python Comes with Many Modules

- **io**
  - Read/write from files
- **math**
  - Mathematical functions
- **random**
  - Generate random numbers
  - Can pick any distribution
- **string**
  - Useful string functions
- **sys**
  - Information about your OS
- **Complete list:**
- [http://docs.python.org/library](http://docs.python.org/library)
- **Library**: built-in modules
  - May change each release
  - Why version #s are an issue
Reading the Python Documentation

- 9.2.1. Number-theoretic and representation functions
- 9.2.2. Power and logarithmic functions
- 9.2.3. Trigonometric functions
- 9.2.4. Angular conversion
- 9.2.5. Hyperbolic functions

These functions cannot be used with complex numbers; use the functions of the same name from the `cmath` module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don’t is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

**Function name**

- `math.ceil(x)`
  - **Argument list**: `x`
  - **What the function evaluates to**: Return the ceiling of `x` as a float, the smallest integer value greater than or equal to `x`.

**Module**

- `math`

**What the function evaluates to**

- `math.fabs(x)`
  - **What the function evaluates to**: Return the absolute value of `x`.

*New in version 2.6.*