

# Lec 3: Strings, files, functions

**Orange text** on the presentation slides indicate significant updates to something on the note-taking handout.

(The handouts go to press significantly before we finalize lecture content.)

Have your iClicker at the ready.

# Operations for Getting Data from Strings: Indexing and slicing

---

- `s = 'abc d'` (note the space)

0	1	2	3	4
a	b	c		d

- Access portions with `[]`.
  - `s[0]` is 'a'
  - `s[4]` is 'd'
  - `s[5]` causes `IndexError: string index out of range`
  - String slicing: give start index, "don't include" start index
    - `s[0:2]` is 'ab' (excludes c). Everyone forgets this at least once.
    - `s[2:]` is 'c d'.
    - `s[:2]` is 'ab'.

# Finger Exercise

---

- `greet = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is `greet[3:6]`?

A: 'lo a'

B: 'lo '

C: 'lo'

# Finger Exercises

---

- `greet = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is `greet[3:6]`?

A: 'lo a'

B: 'lo ' **CORRECT**

C: 'lo'

# Other Ways to Get Data from Strings

---

- `s1 in s2`
  - Tests if `s1` “a part of” `s`
- `len(s)`
  - Value is # of chars in `s`
- `s1.index(s2)`
  - Position of the 1st instance of `s2` in `s1`
- `s1.count(s2)`
  - Number of times `s2` appears inside `s1`
- `s.strip()`
  - A copy of `s` with white-space removed at ends
  - `s1.strip(s2)` removes the characters in `s2` from ends of `s1`, if there are any.

```
s = 'abracadabra'
```

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

*# 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'
```

# Finger Exercise

---

- `greet = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is `greet.index('l')`?

- How about this?

`greet.index(greet[7:])`

- A: error
- B: 7
- C: True
- D: 2

**(blank to prevent fast-forward)**

---

# Finger Exercise

---

- `greet = 'Hello all'`

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is `greet.index('l')`?

- How about this?

`greet.index(greet[7:])`

A: error

B: 7

C: True

D: 2      **CORRECT**



# 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<sup>nd</sup> ','. (*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* j

# (3) Give an expression for the part of **tail** starting after ','

**(blank to prevent fast-forward)**

---

# String Puzzle Solution

---

**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<sup>nd</sup> ','. (*How can we use the index operation?*)

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

```
j = data.index(',')
```

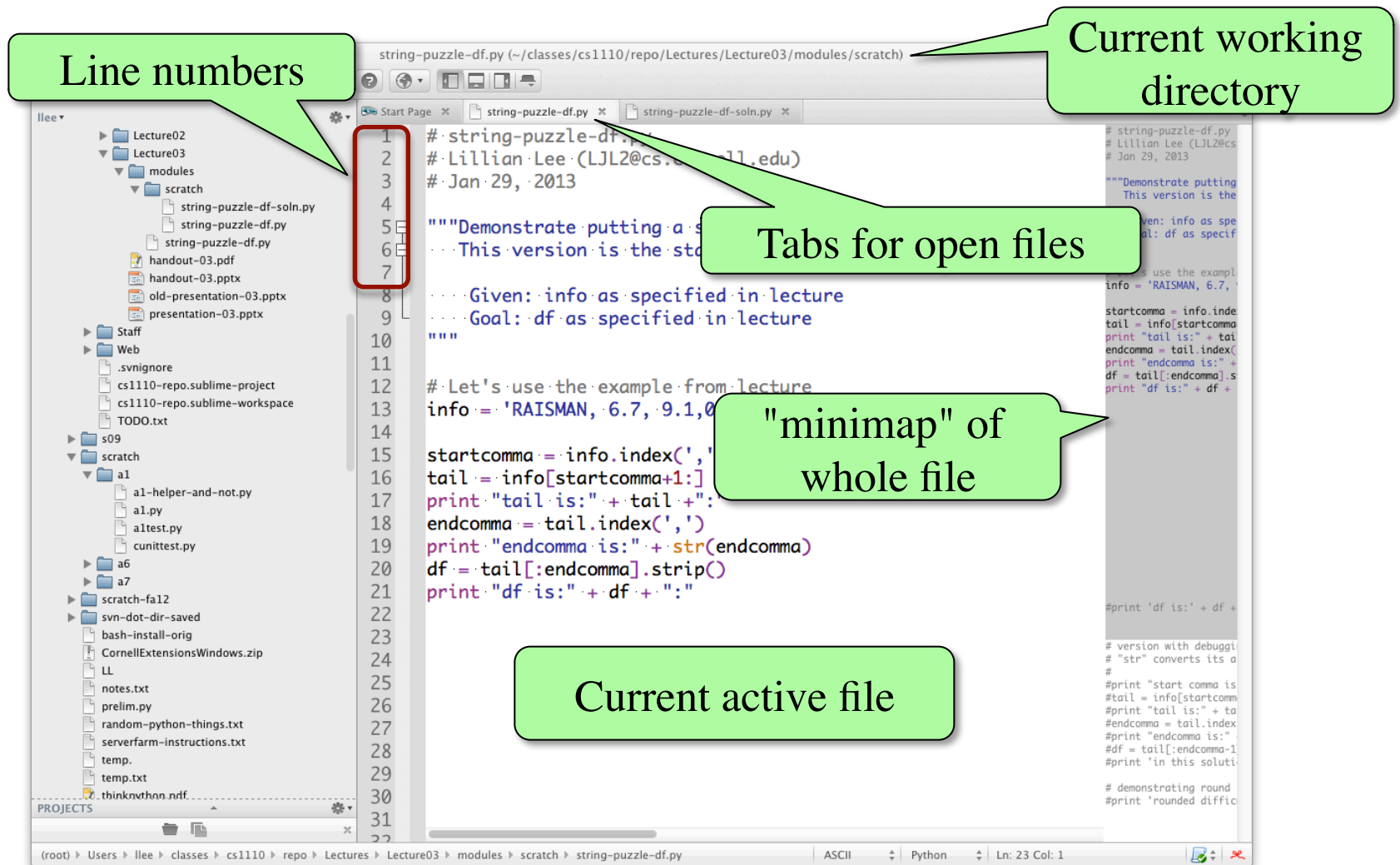
# (2) Store in variable **tail** the part of data starting *after* j

```
tail = data[j+1: ]
```

# (3) Give an expression for the part of **tail** starting after ','

```
tail[tail.index(',')+1: ]
```

# Install Komodo Edit on your laptops!



# Req'd Format for CS1110 Python Files

---

```
# string_puzzle_df.py
# Lillian Lee (LJL2) and Steve Marschner (SRM2)
# Thu Jan 30, 2014
```

## Header:

file name, authoring info

```
""" Demonstrates putting a sequence
of commands into a ... """
```

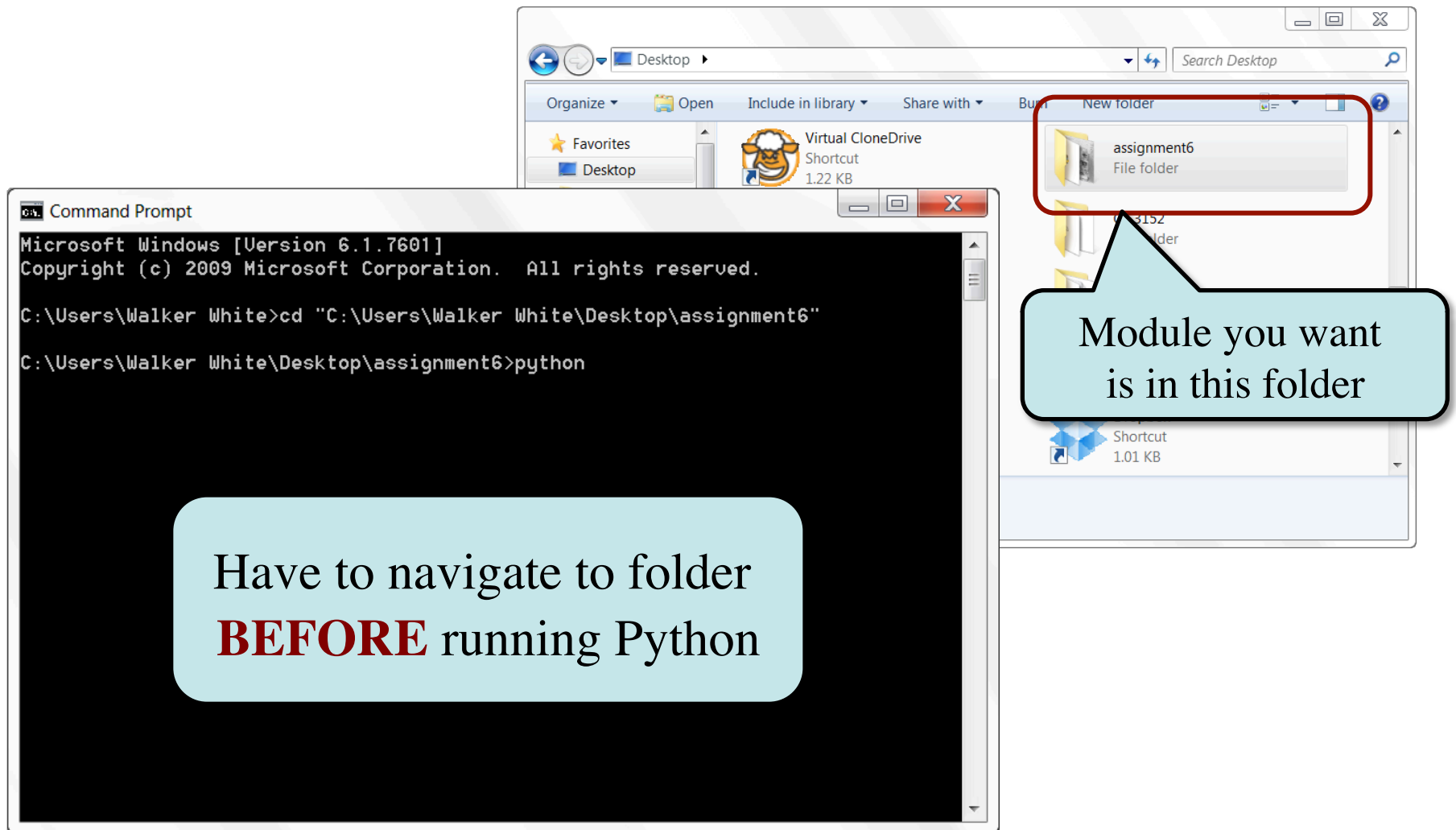
## Docstring: *note the triple quotes.*

Multi-line comment explaining the purpose & function of the file.

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

Note: Unlike with the command prompt, evaluating an expression produces nothing when a Python file is run. **Writing just `x` wouldn't do anything.**

# Start Python in Your Script's Directory!



# Running Python Commands from a File

---

- At the terminal prompt (*not* >>>):  
`python string_puzzle_soln.py`

**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:]` # extra space OK

C: `endcomma = tail.index(',')`

D: `df = tail[:endcomma-1].strip()`

E: this sequence achieves the goal



# Using a Function From Another File

## (such files are called *modules*)

---

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

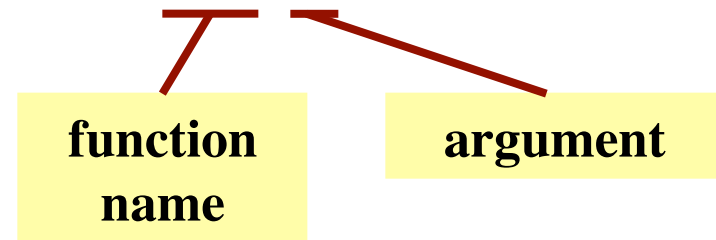
*Lucky us:* someone has written a module (file) string that contains a function capwords.

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

# 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()`

# 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>
- **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
- 9.2.6. Miscellaneous functions
- 9.2.7. Constants

standard.

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.

Function name

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

Argument list

`math.ceil(x)`

Return the ceiling of `x` as a float, the smallest integer value greater than or equal to `x`.

Module

What the function evaluates to

`copysign(x, y)`

From that supports signed zeros, `copysign(1.0, -0.0)` returns

*New in version 2.6.*

`math.fabs(x)`

Return the absolute value of `x`.

`math.factorial(x)`

This Page

[Report a Bug](#)  
[Show Source](#)

Quick search

Go

# Print Statements:

## Useful Inspection Tool (in Python Files)

`print <expression>` evaluates `<expression>`, converts it to a string, and displays it.

```
data = 2  
print data
```

2

```
data = 'this has two trailing spaces '  
print data
```

this has two trailing spaces

```
print 'data is:' + str(data) + ':'
```

data is:this has two trailing spaces :