Lecture 3

# Functions & Modules

# Labs this Week

- Lab 1 is due at the **beginning** of your lab
  - If it is not yet by then, you cannot get credit
  - Only exception is for students who added late (Those students should talk to me)
- Should spend time ***entirely*** on Lab 2
  - Getting behind this early is bad
  - We are getting you ready for Assignment 1

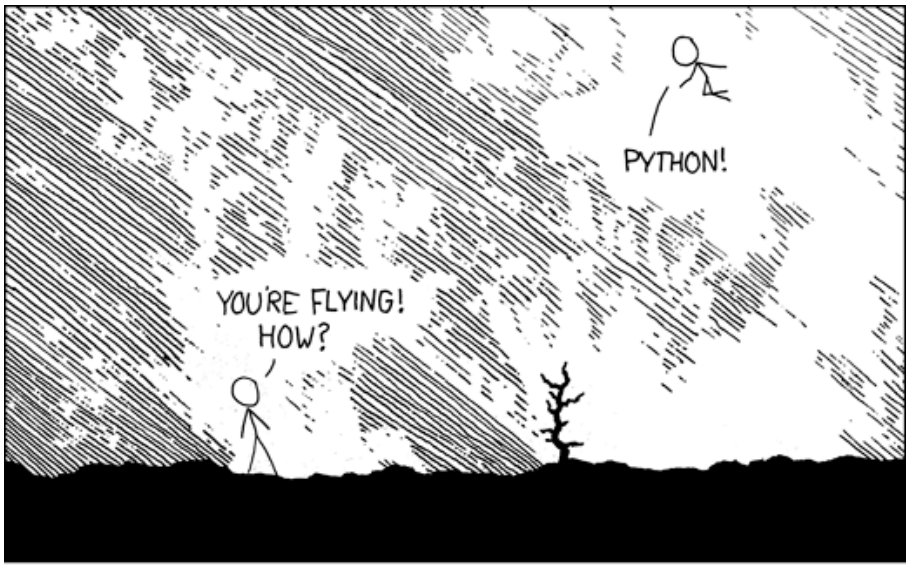
# Readings for Next Few Lectures

## Thursday Reading

- Chapter 3
  - But can skip 3.10
- Browse the Python API
  - Will learn what that is today
  - Do not need to read all of it

## Next Week

- Sections 8.1, 8.2, 8.4, 8.5

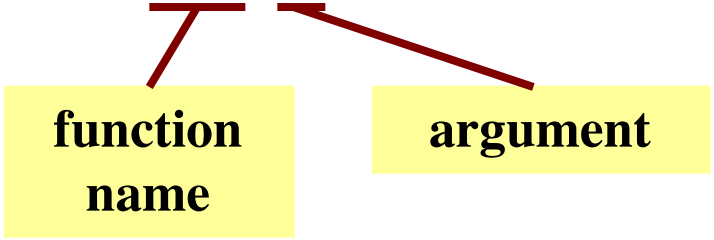




[xkcd.com]

# Function Calls

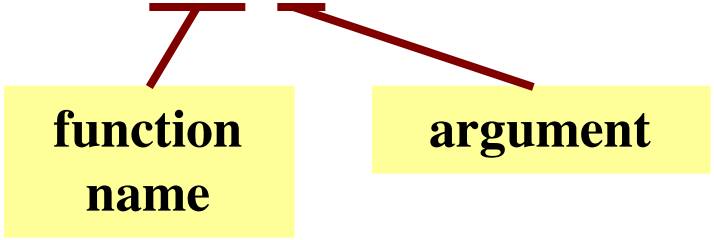
- Python supports expressions with math-like functions
  - A function in an expression is a **function call**
  - Will explain the meaning of this later
- Function expressions have the form **fun**(x,y,…)




- **Examples** (math functions that work in Python):
  - `round(2.34)`
  - `max(a+3,24)`

# Function Calls

- Python supports expressions with math-like functions
  - A function in an expression is a **function call**
  - Will explain the meaning of this later
- Function expressions have the form **fun**(x,y,…)




- **Examples** (math functions that work in Python):
  - `round(2.34)`
  - `max(a+3,24)`

Arguments can be any **expression**

# Built-In Functions

- You have seen many functions already
  - Type casting functions: int(), float(), bool()
  - Dynamically type an expression: type()
  - Help function: help()
  - Quit function: quit()

Arguments go in (), but name() refers to function in general

- print <string> is **not** a function call
  - It is simply a statement (like assignment)
  - But it is in Python 3.x: print(<string>)

# Built-in Functions vs Modules

- The number of built-in functions is small
  - http://docs.python.org/2/library/functions.html
- Missing a lot of functions you would expect
  - **Example**: `cos()`, `sqrt()`
- **Module**: file that contains Python code
  - A way for Python to provide optional functions
  - To access a module, the `import` command
  - Access the functions using module as a *prefix*

# Example: Module **math**

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

# Example: Module **math**

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

To access math functions

# Example: Module **math**

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

To access math functions

Functions require math prefix!

# Example: Module **math**

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

To access math functions

Functions require math prefix!

Module has variables too!

# Example: Module **math**

```
>>> import math
>>> math.cos(0)
1.0
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

To access math functions

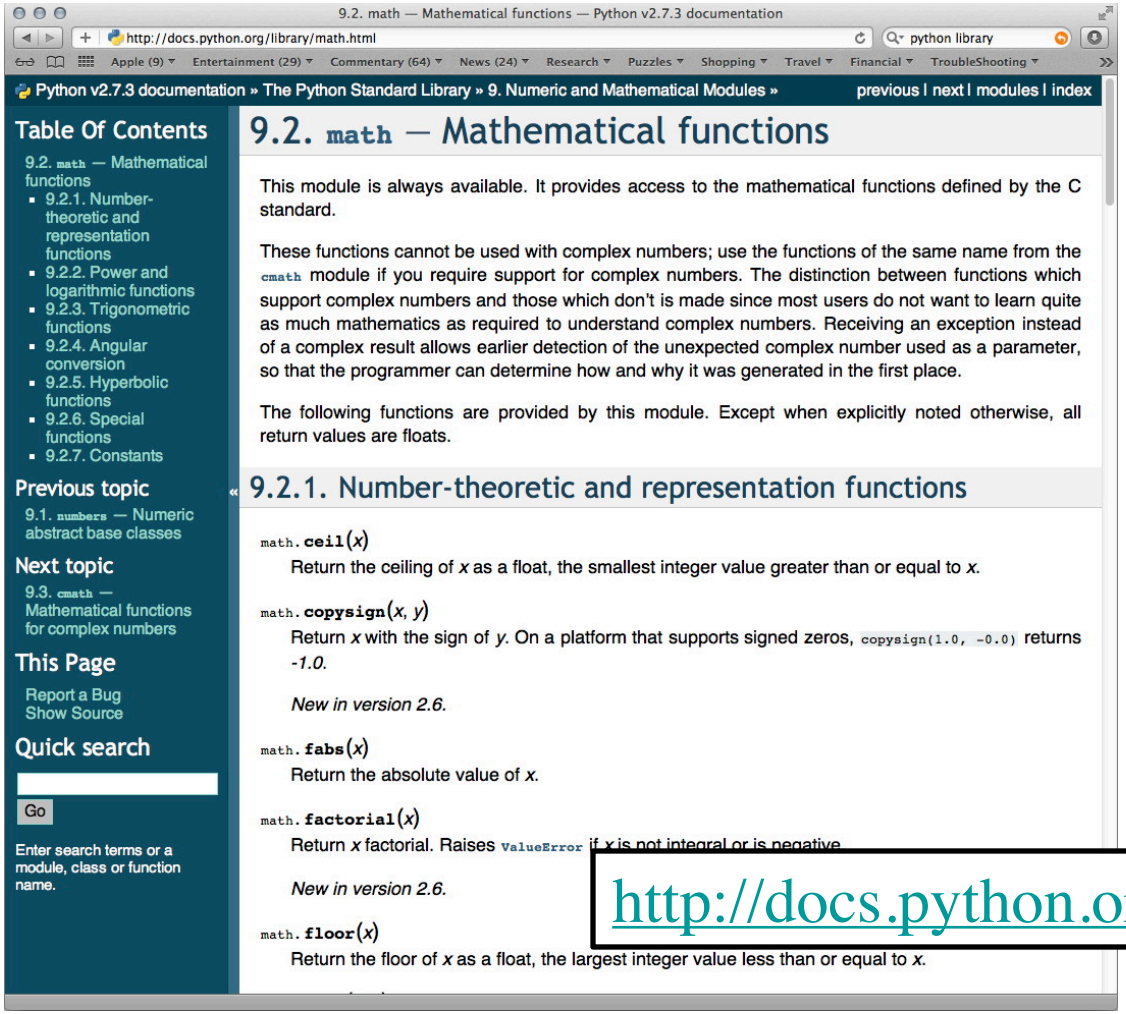
Functions require math prefix!

Module has variables too!

## Other Modules

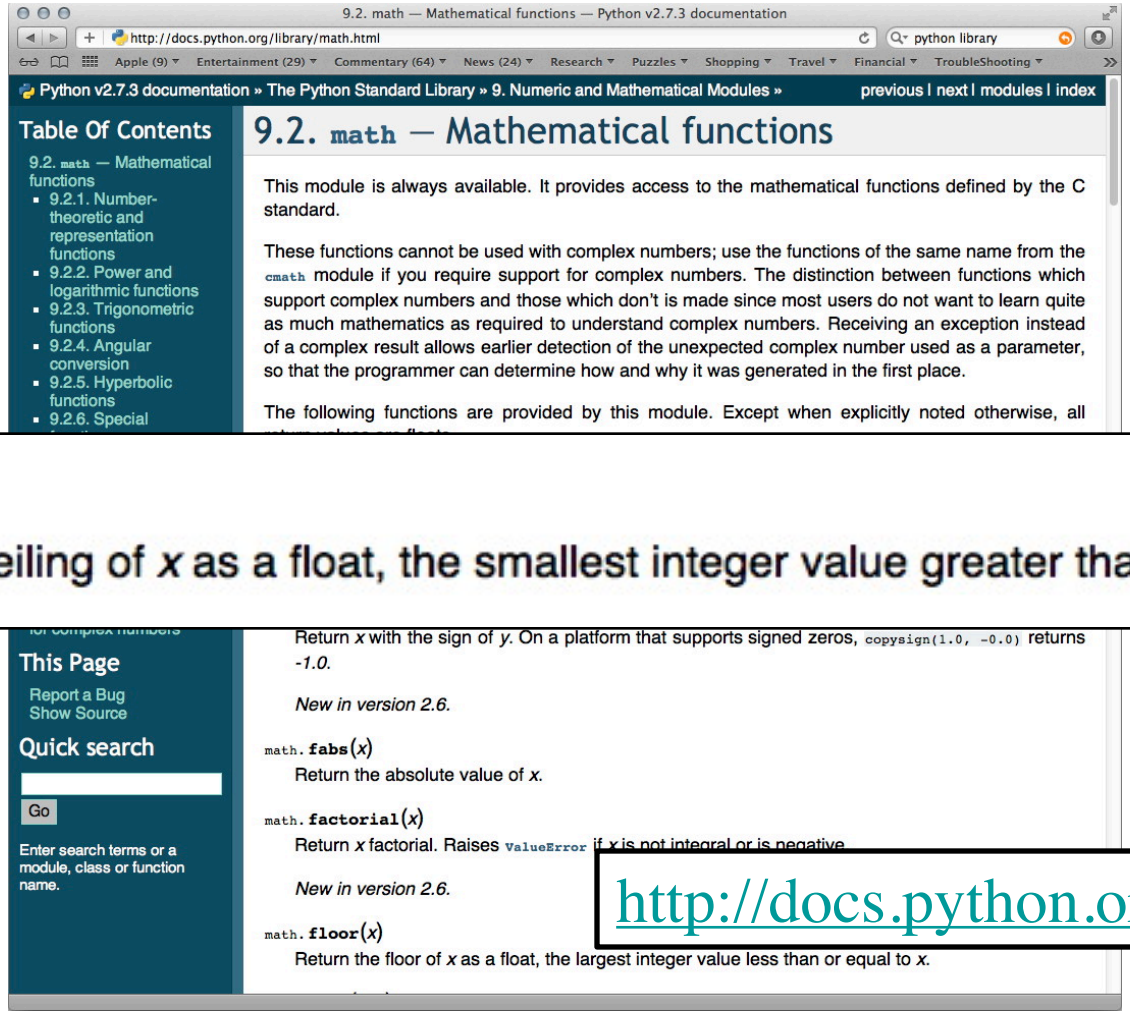
- io
  - Read/write from files
- random
  - Generate random numbers
  - Can pick any distribution
- string
  - Useful string functions
- sys
  - Information about your OS

# Reading the Python Documentation

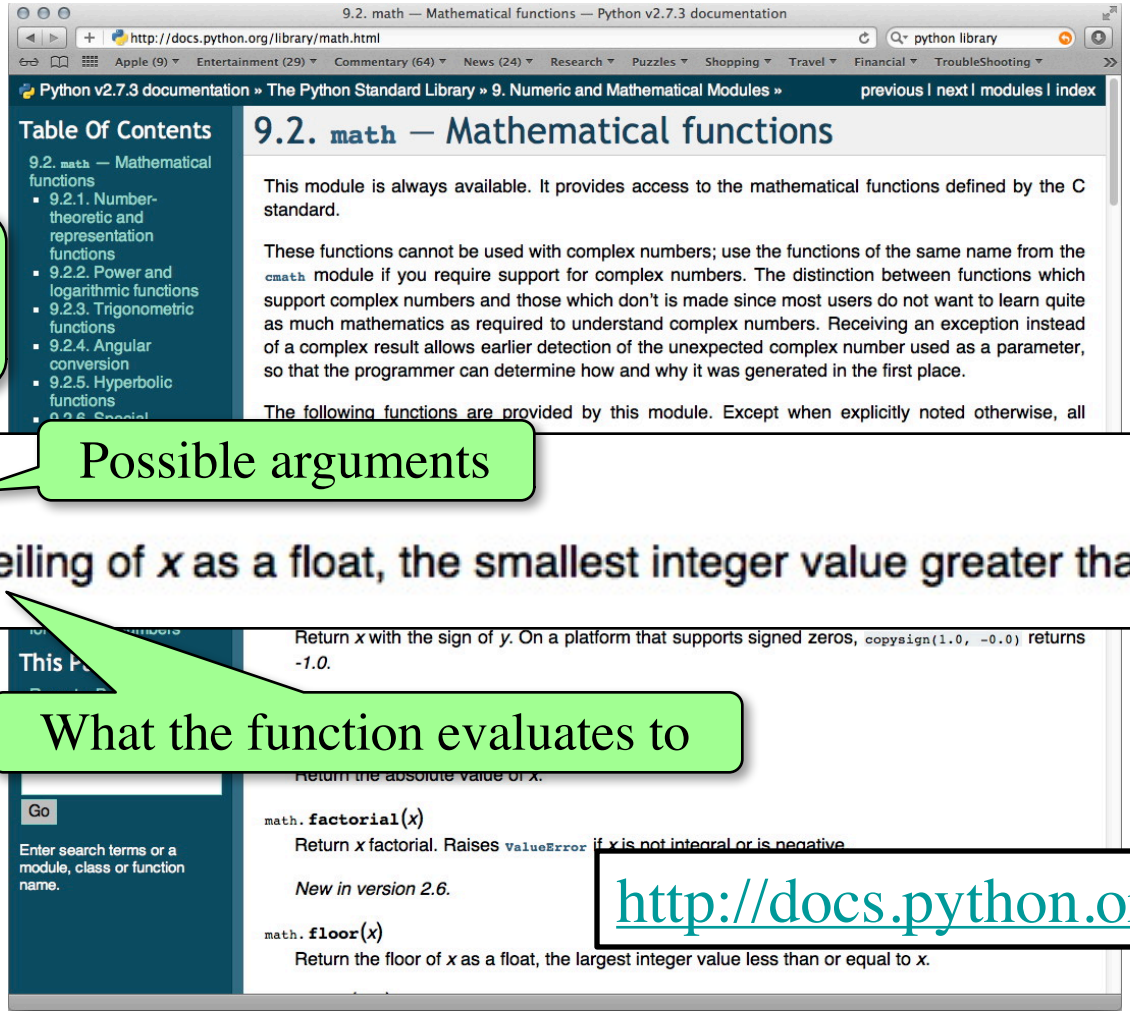



http://docs.python.org/library

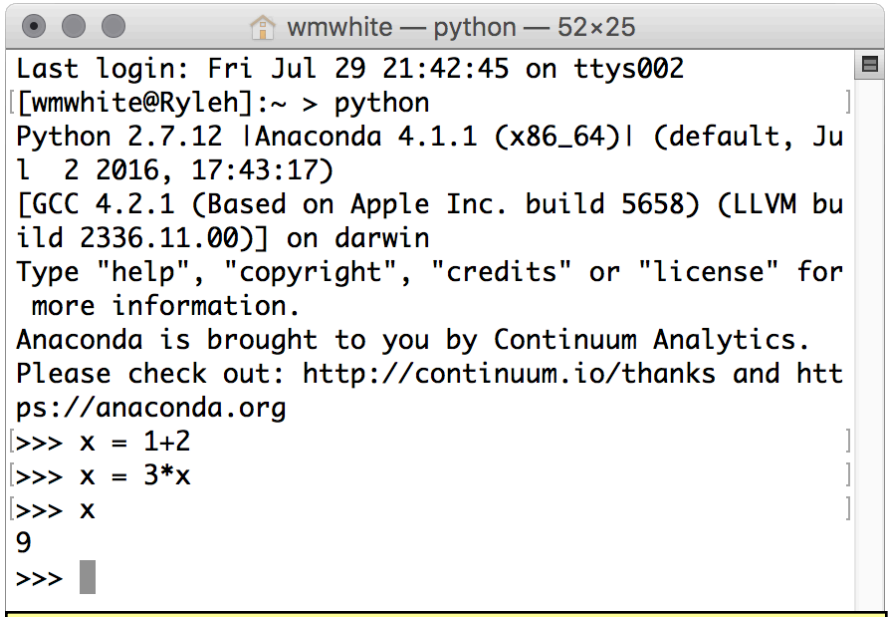
# Reading the Python Documentation



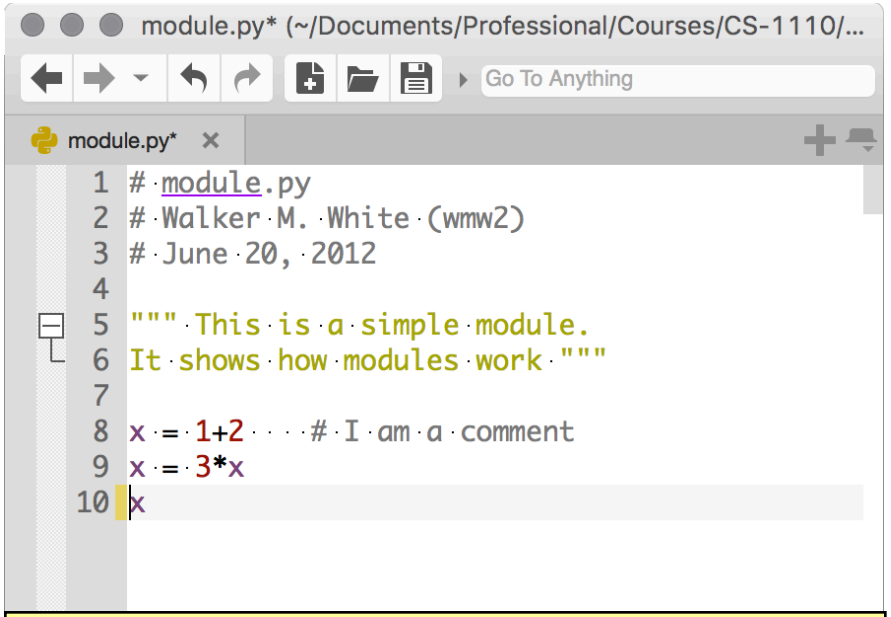
# Reading the Python Documentation



# Interactive Shell vs. Modules




- Launch in command line
- Type each line separately
- Python executes as you type




- **Write in a text editor**
  - We use Komodo Edit
  - But anything will work
- Load module with `import`

# Using a Module

## Module Contents

```
# module.py

""" This is a simple module.
It shows how modules work"""

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

# Using a Module

## Module Contents

```
# module.py
```

**Single line comment**
(not executed)

```
""" This is a simple module.
It shows how modules work"""

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

# Using a Module

## Module Contents

```
# module.py
```

**Single line comment**
(not executed)

```
""" This is a simple module.
It shows how modules work"""
```

**Docstring** (note the Triple Quotes)
Acts as a multiple-line comment
Useful for *code documentation*

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

# Using a Module

## Module Contents

```
# module.py
```

**Single line comment**
(not executed)

```
""" This is a simple module.
It shows how modules work"""
```

**Docstring** (note the Triple Quotes)
Acts as a multiple-line comment
Useful for *code documentation*

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

**Commands**
Executed on import

# Using a Module

## Module Contents

```
# module.py
```

**Single line comment**
(not executed)

```
""" This is a simple module.
It shows how modules work"""
```

**Docstring** (note the Triple Quotes)
Acts as a multiple-line comment
Useful for *code documentation*

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

**Commands**
Executed on import

```
x
```

Not a command.
import **ignores this**

# Using a Module

## Module Contents

```
# module.py

""" This is a simple module.
It shows how modules work"""

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

## Python Shell

```
>>> import module
>>> x
```

# Using a Module

## Module Contents

```
# module.py

""" This is a simple module.
It shows how modules work"""

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

## Python Shell

```
>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

# Using a Module

## Module Contents

```
# module.py

""" This is a simple module.
It shows how modules work"""

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

"**Module data**" must be prefixed by module name

## Python Shell

```
>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> module.x
9
```

# Using a Module

## Module Contents

```
# module.py


""" This is a simple module.
It shows how modules work"""


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

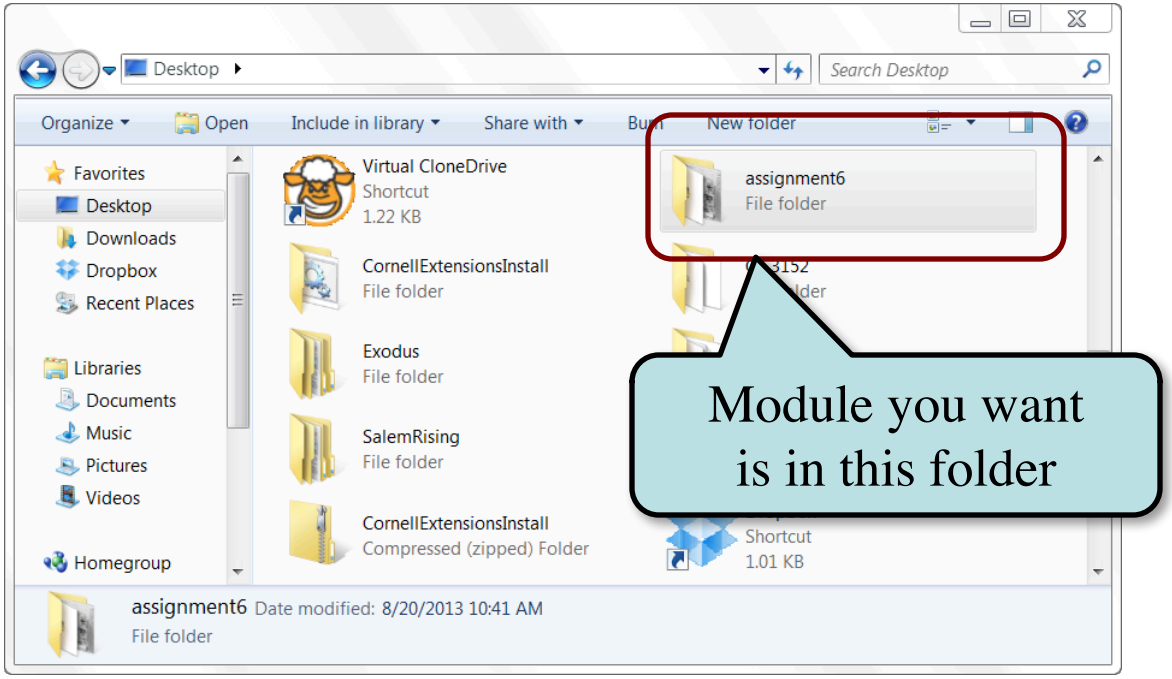
"**Module data**" must be prefixed by module name

Prints **docstring** and module contents

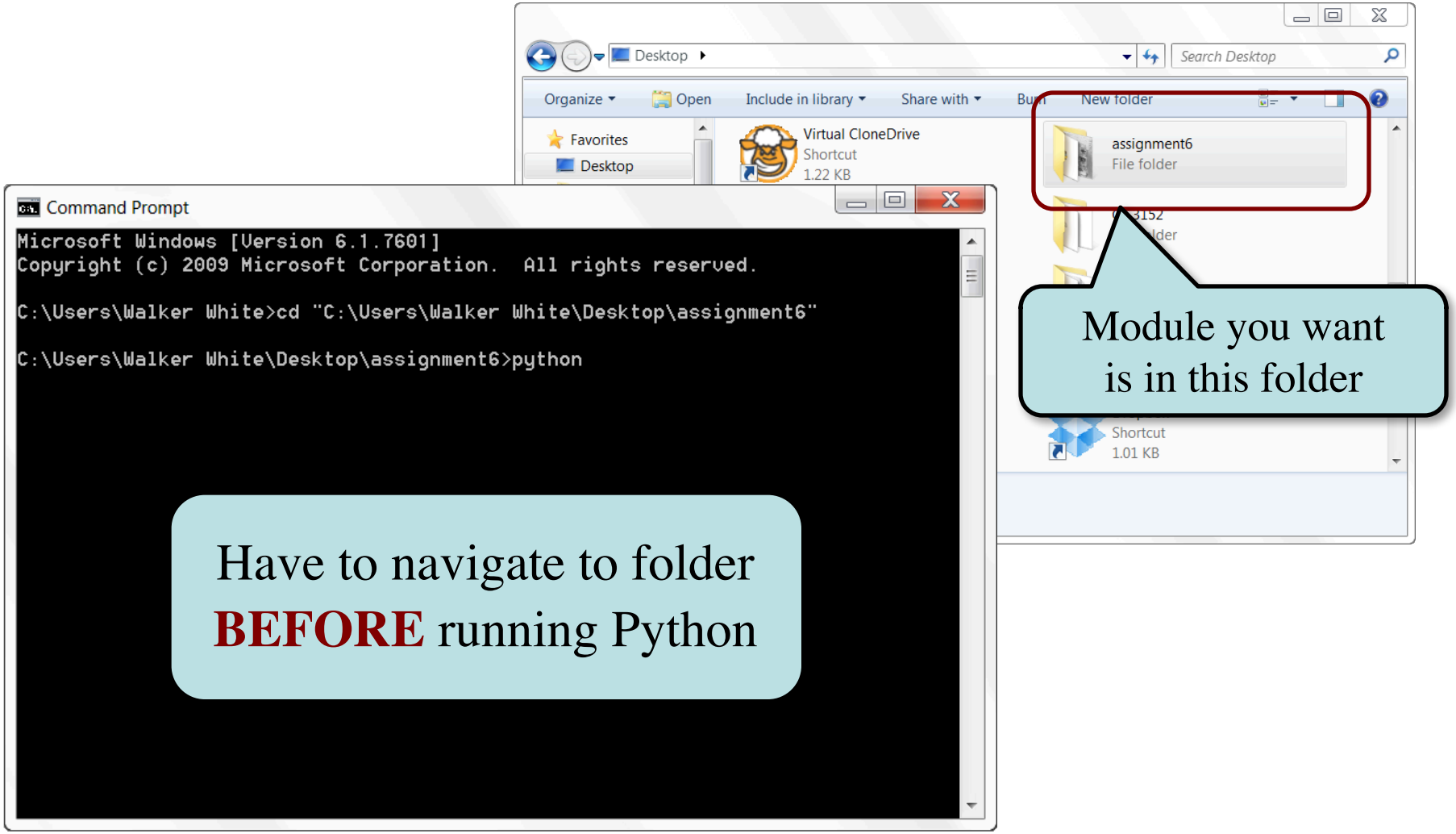
## Python Shell

```
>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> module.x
9
>>> help(module)
```

# Modules Must be in Working Directory!



# Modules Must be in Working Directory!



# Using the from Keyword

```
>>> import math
>>> math.pi
3.141592653589793
>>> from math import pi
>>> pi
3.141592653589793
>>> from math import *
>>> cos(pi)
-1.0
```

Must prefix with module name

No prefix needed for variable pi

No prefix needed for anything in math

- Be careful using from!
- Using import is *safer*
  - Modules might conflict (functions w/ same name)
  - What if import both?
- **Example**: Turtles
  - Used in Assignment 4
  - 2 modules: turtle, tkturtle
  - Both have func. Turtle()

# Modules vs. Scripts

## Module

- Provides functions, variables
  - **Example**: temp.py
- import it into Python shell

```
>>> import temp
>>> temp.to_fahrenheit(100)
212.0
>>>
```

## Script

- Behaves like an application
  - **Example**: helloApp.py
- Run it from command line:

```
python helloApp.py
```

# Modules vs. Scripts

## Module

- Provides functions, variables
  - **Example**: temp.py
- import it into Python shell

```
>>> import temp
>>> temp.to_fahrenheit(100)
212.0
>>>
```

## Script

- Behaves like an application
  - **Example**: helloApp.py
- Run it from command line:

```
python helloApp.py
```


Files look the same. Difference is how you use them.

# Scripts and Print Statements

## module.py

```
# module.py

""" This is a simple module.
It shows how modules work"""

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

## script.py

```
# script.py

""" This is a simple script.
It shows why we use print"""

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

# Scripts and Print Statements

## module.py

```
# module.py

""" This is a simple module.
It shows how modules work"""

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

## script.py

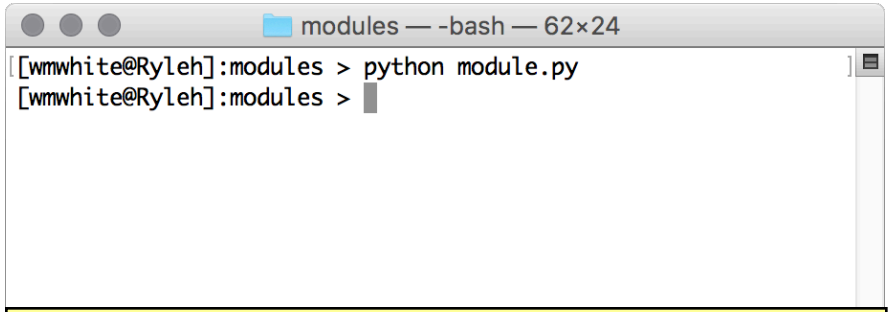
```
# script.py

""" This is a simple script.
It shows why we use print"""

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

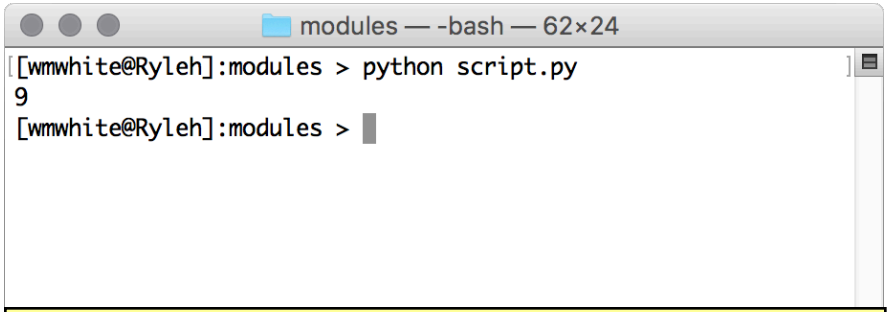
# Scripts and Print Statements

## module.py




- Looks like nothing happens
- Python did the following:
  - Executed the assignments
  - Skipped the last line
    (‘x’ is not a statement)

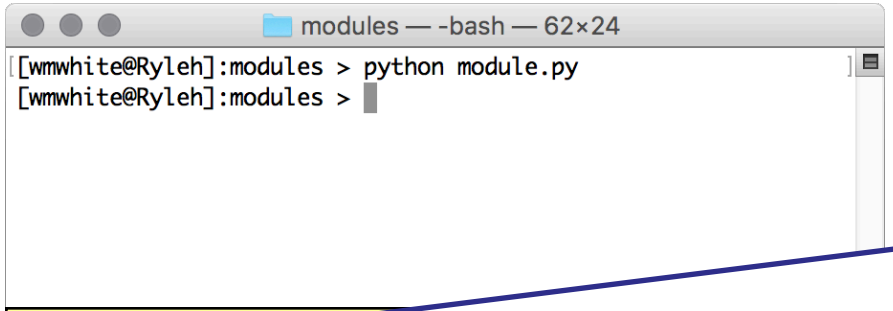
## script.py




- We see something this time!
- Python did the following:
  - Executed the assignments
  - Executed the last line
    (Prints the contents of x)

# Scripts and Print Statements

## module.py




- Looks l
- Python
  - Execu ssignments
  - Skipped the last line
    (‘x’ is not a statement)

## script.py


his time!

llowing:

- Executed the assignments
- Executed the last line
  (Prints the contents of x)

When you run a script,
only statements are executed

# Next Time: Defining Functions

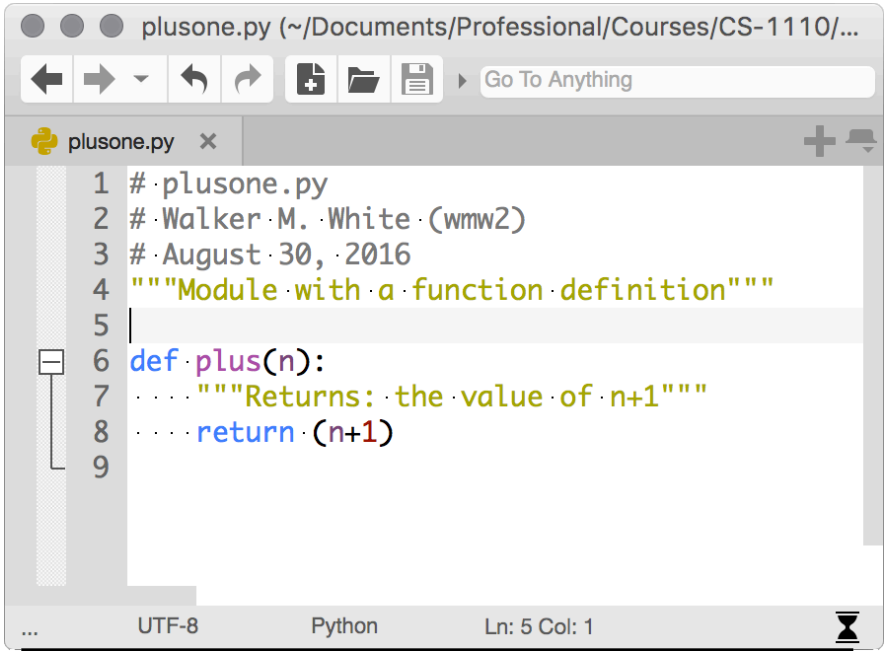
## Function Call

- Command to **do** the function
- Can put it anywhere
  - In the Python shell
  - Inside another module

```
modules — python — 52×20
>>> import plusone
>>> plusone.plus(1)
2
>>> plusone.plus(2)
3
>>> plusone.plus(3)
4
>>>
```

## Function Definition

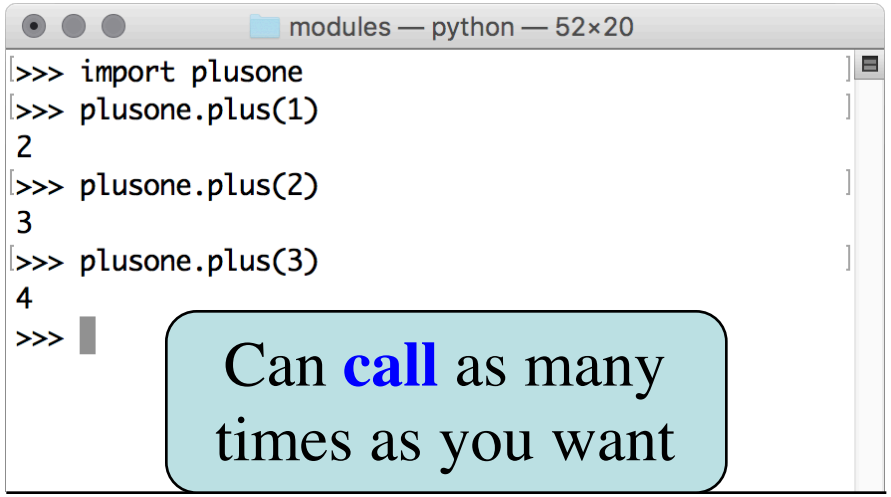
- Command to **do** the function
- Belongs inside a module



# Next Time: Defining Functions

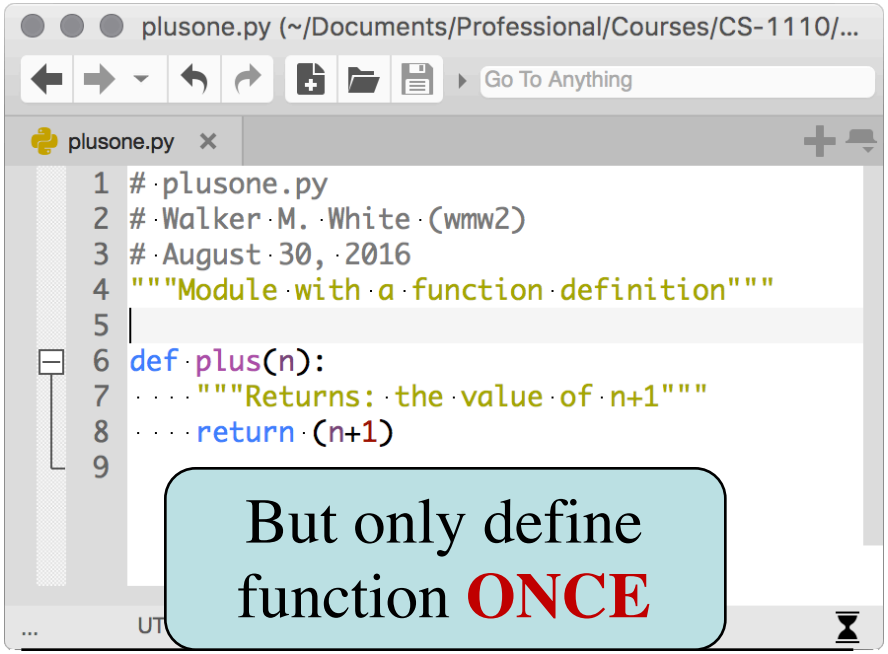
## Function Call

- Command to **do** the function
- Can put it anywhere
  - In the Python shell
  - Inside another module




## Function Definition

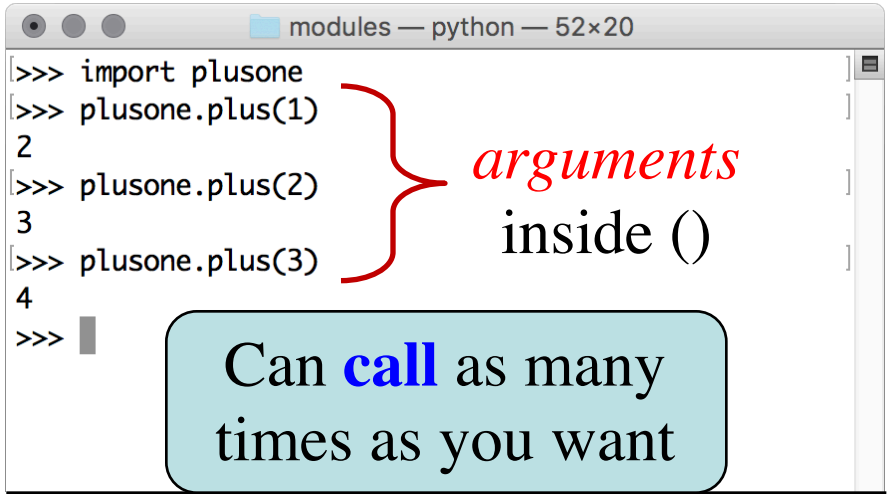
- Command to **do** the function
- Belongs inside a module



# Next Time: Defining Functions

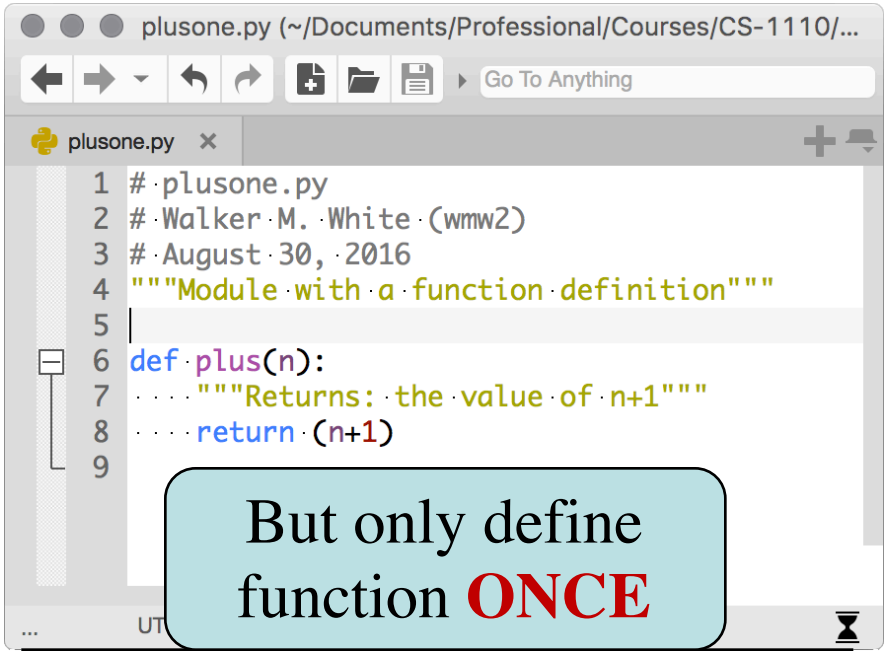
## Function Call

- Command to **do** the function
- Can put it anywhere
  - In the Python shell
  - Inside another module




## Function Definition

- Command to **do** the function
- Belongs inside a module



# Functions and Modules

- Purpose of modules is **function definitions**
  - Function definitions are written in module file
  - Import the module to call the functions
- Your Python workflow (right now) is

1. Write a function in a module (a .py file)
2. Open up the Terminal/Command Prompt
3. Move to the directory with this file
4. Start Python (type `python`)
5. Import the module
6. Try out the function