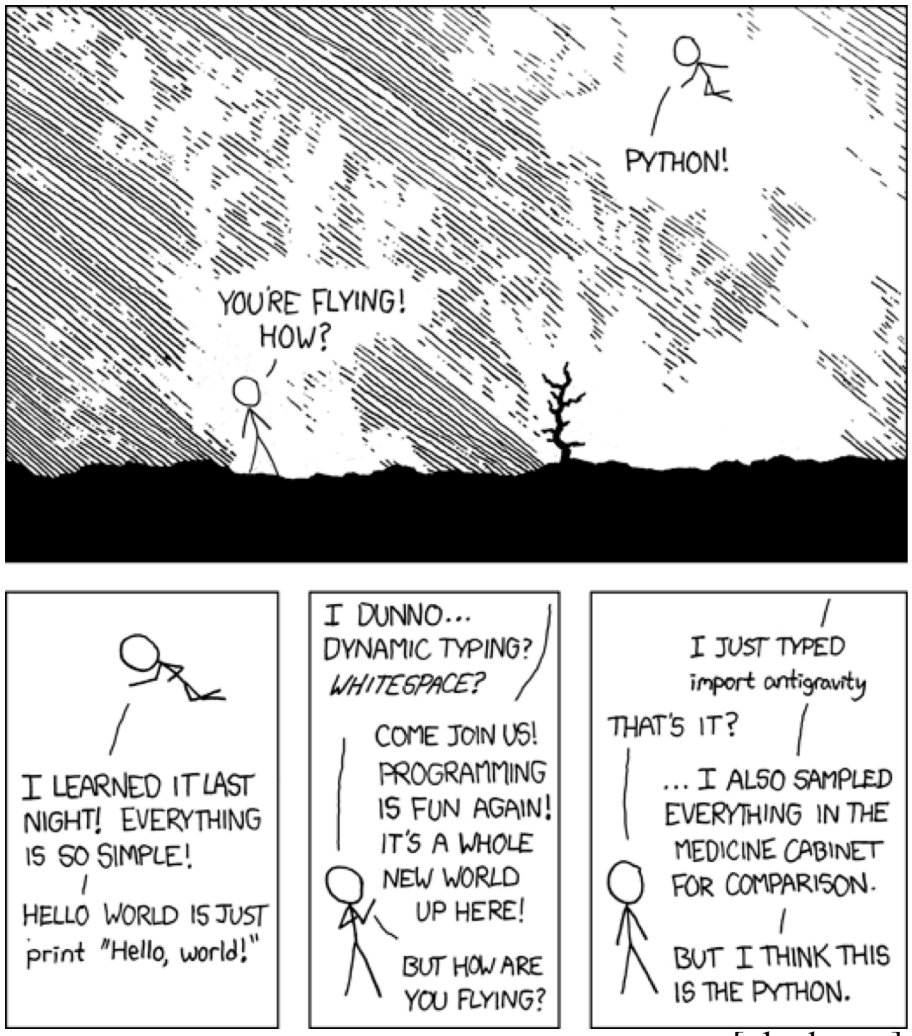
Lecture 3

Functions & Modules

(Optional) Readings

Reading for Next Week

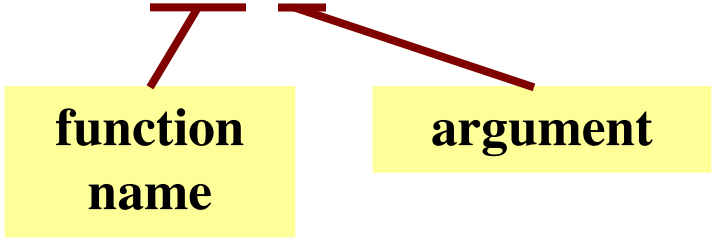
- Chapter 3 in the text
  - But can skip section 3.9
- Browse the Python API
  - Will learn what that is today
  - Do not need to read all of it
- Sections 8.1, 8.2, 8.5, 8.8
  - Strings are needed for A1
  - But Chap 8 mixes easy stuff with advanced stuff



[xkcd.com]

Function Calls

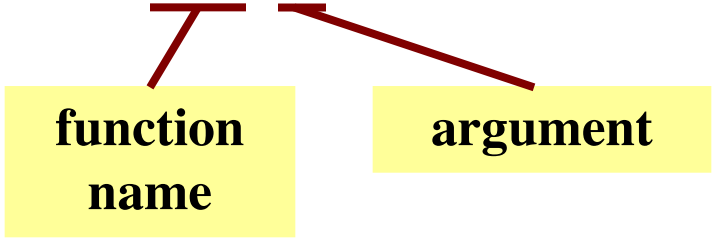
- Python supports expressions with math-like functions
  - A function in an expression is a **function call**
  - Will explain the meaning of this later
- Function expressions have the form **fun**(x,y,…)



- **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

- One of the most important functions is `print()`
  - `print(exp)` displays value of `exp` on screen
  - Will see later why this is important

Built-in Functions vs Modules

- The number of built-in functions is small
  - http://docs.python.org/3/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

Using the from Keyword

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

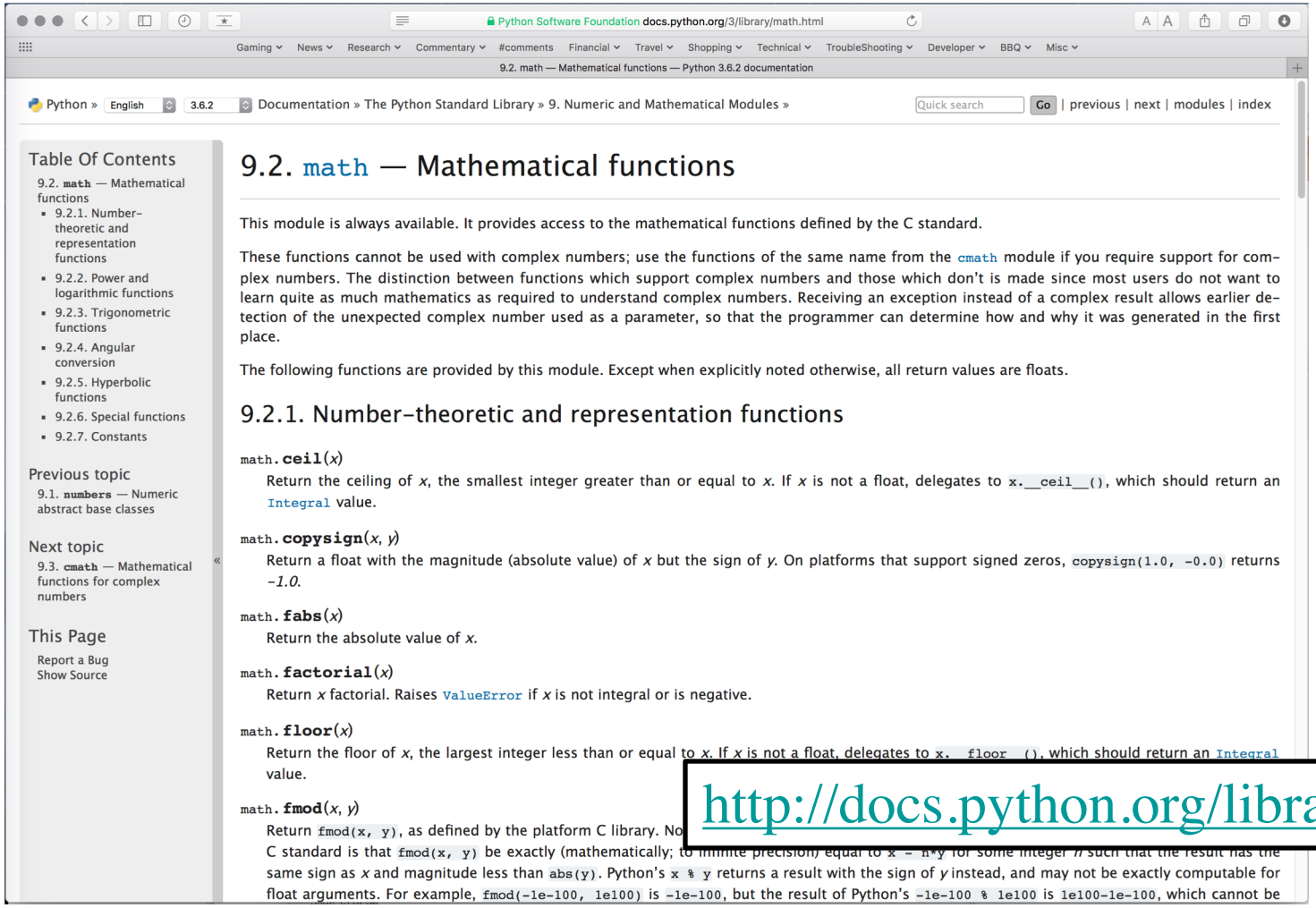
Must prefix with module name

No prefix needed for variable pi

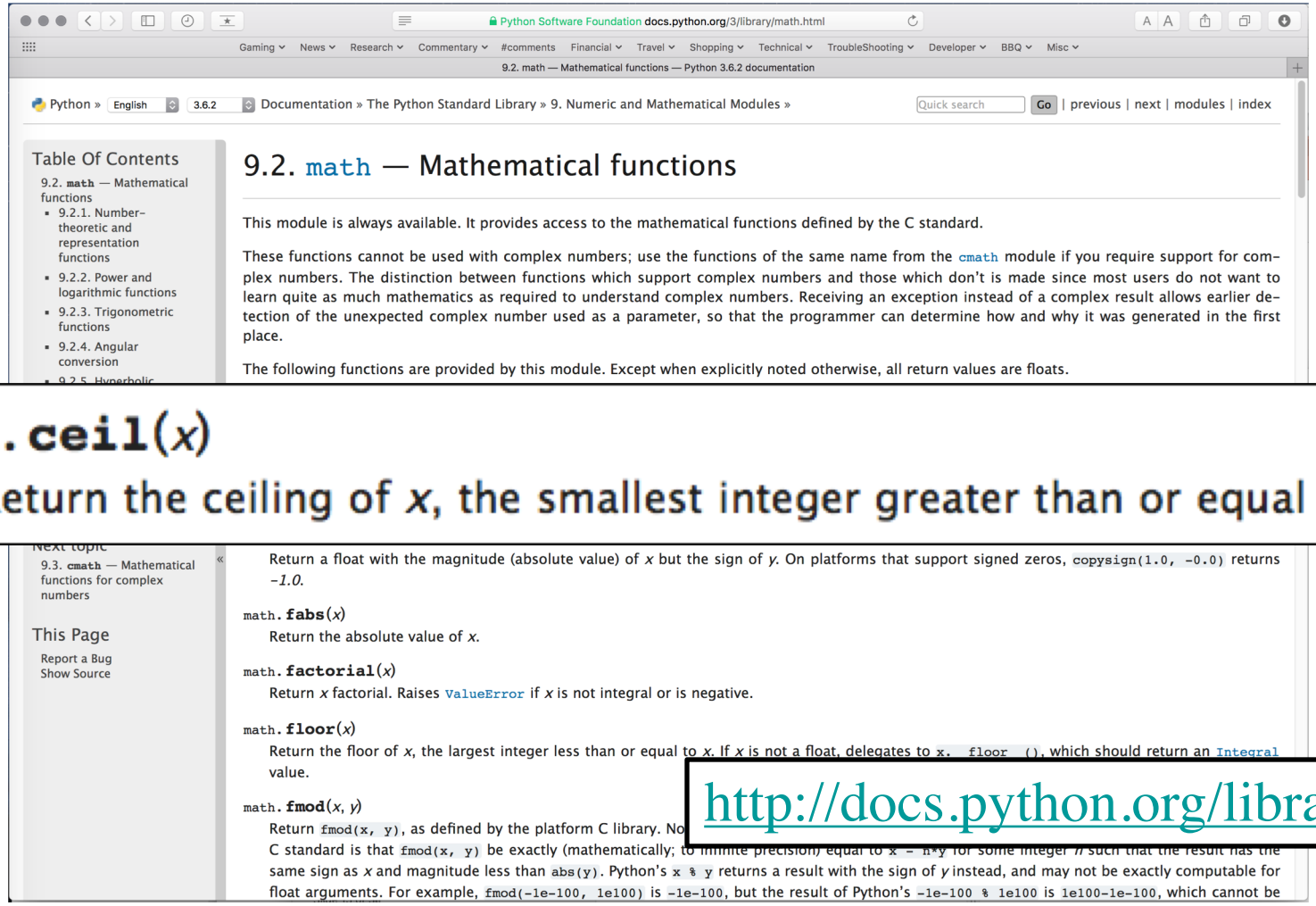
No prefix needed for anything in math

- 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()

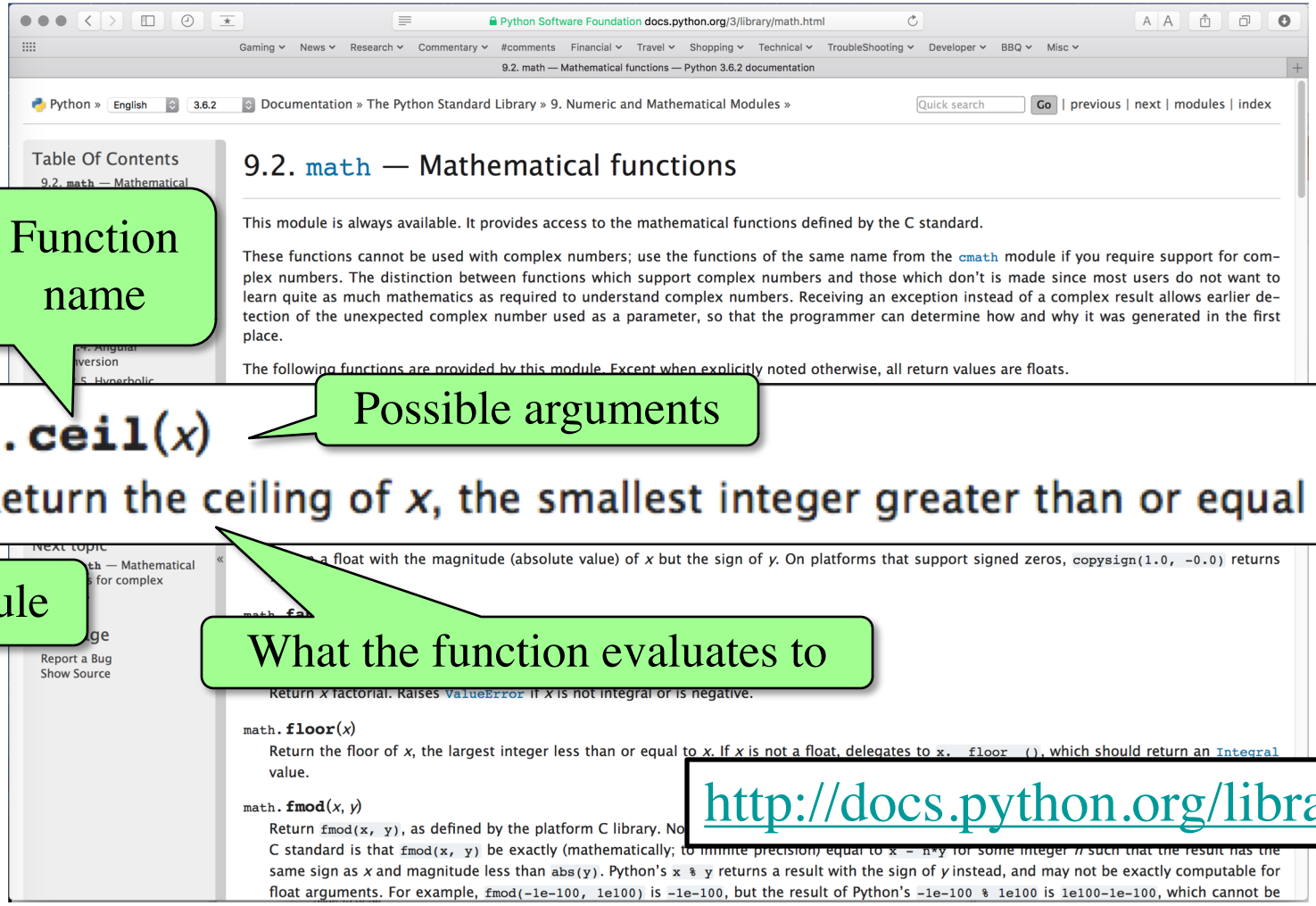
Reading the Python Documentation



Reading the Python Documentation

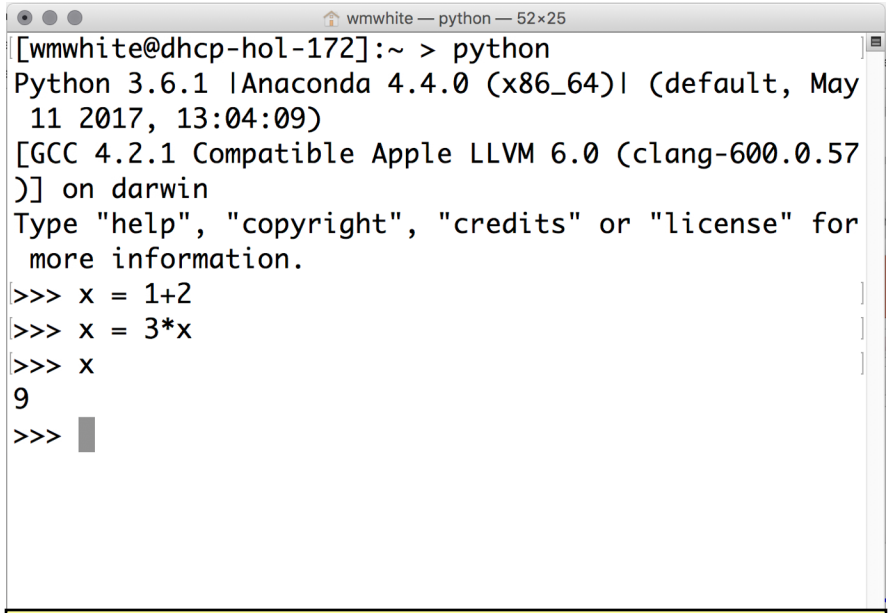


Reading the Python Documentation

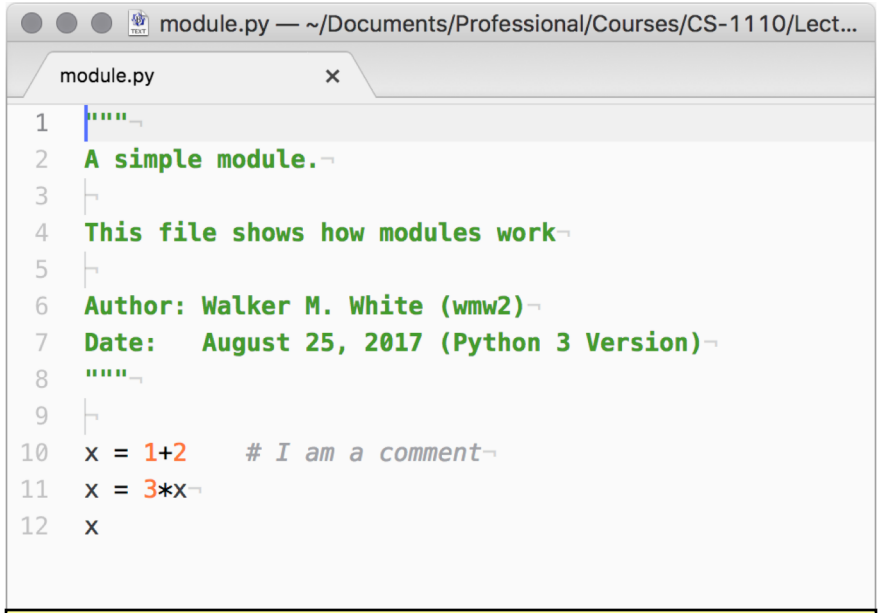


http://docs.python.org/library

Interactive Shell vs. Modules



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



- **Write in a code editor**
  - We use Atom Editor
  - But anything will work
- Load module with `import`

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""


# This is a comment
x = 1+2
x = 3*x
x
```

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

Single line comment
(not executed)

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

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

Single line comment
(not executed)

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

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

Single line comment
(not executed)

Commands
Executed on import

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

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

Single line comment
(not executed)

Commands
Executed on import

Not a command.
import **ignores this**

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""


# This is a comment
x = 1+2
x = 3*x
x
```

Python Shell

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

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""


# This is a comment
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

```
""" A simple module.

This file shows how modules work
"""


# This is a comment
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

```
""" A simple module.

This file shows how modules work
"""


# This is a comment
x = 1+2
x = 3*x
x
```

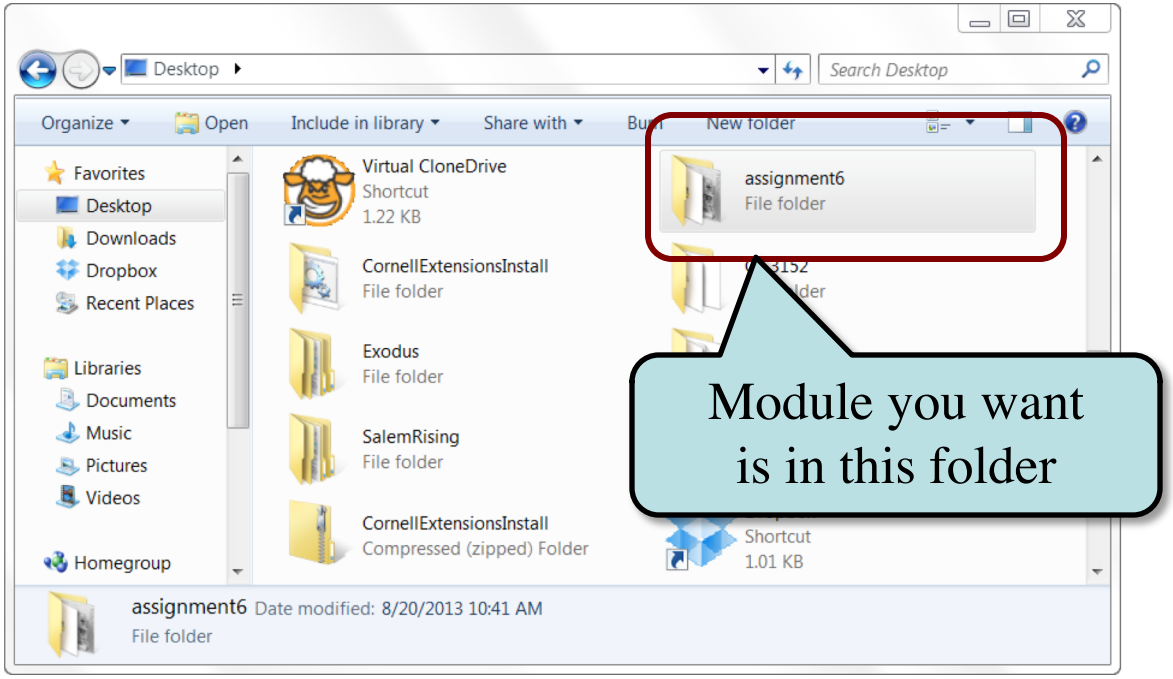
Python Shell

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

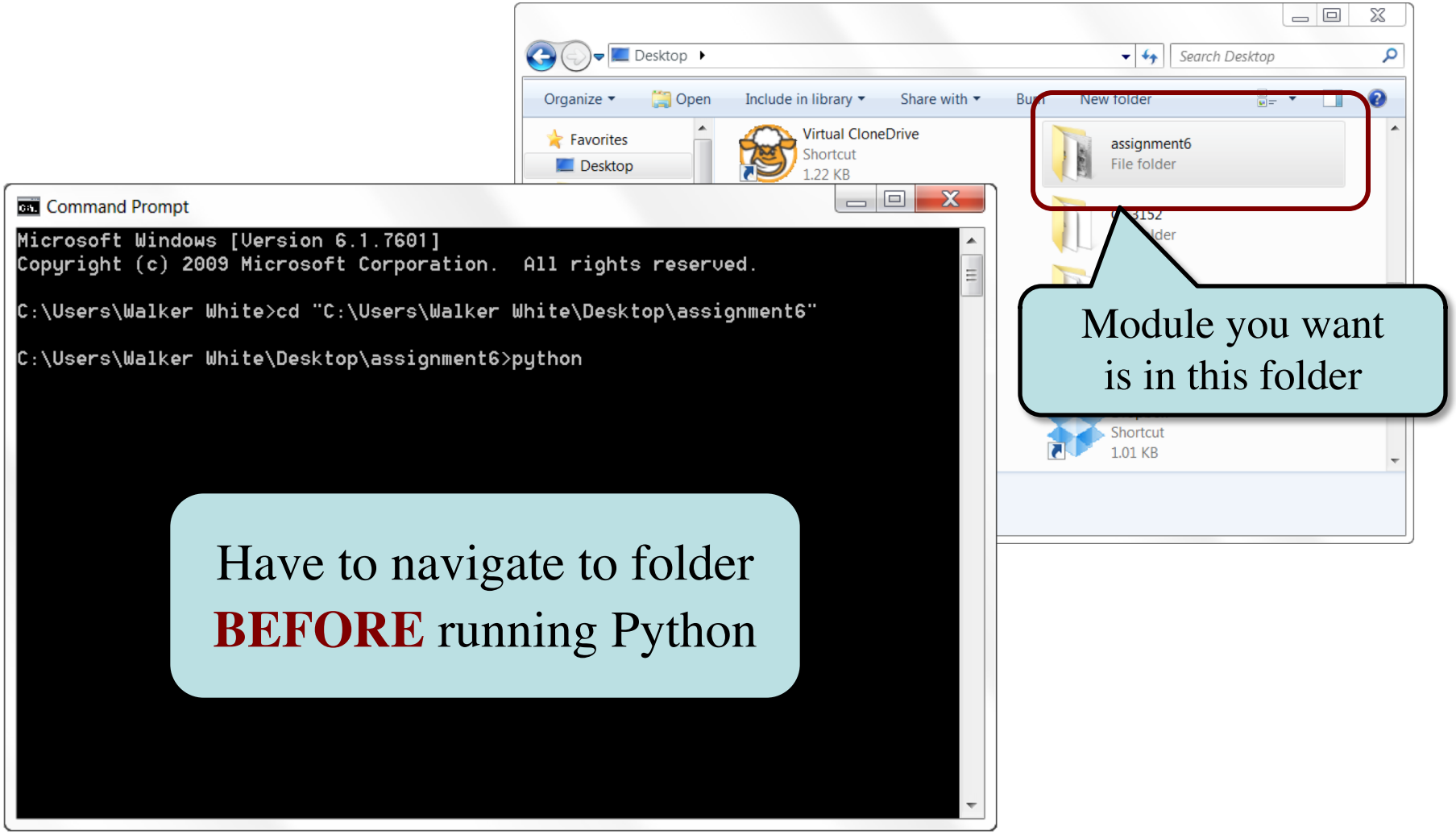
"**Module data**" must be prefixed by module name

Prints **docstring** and module contents

Modules Must be in Working Directory!



Modules Must be in Working Directory!



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

```
""" A simple module.

This file shows how modules work
"""


# This is a comment
x = 1+2
x = 3*x
x
```

script.py

```
""" A simple script.

This file shows why we use print
"""


# This is a comment
x = 1+2
x = 3*x
print(x)
```

Scripts and Print Statements

module.py

```
""" A simple module.

This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x
```

script.py

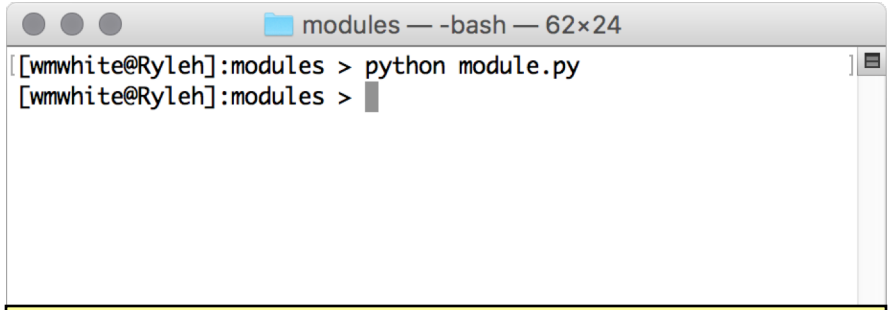
```
""" A simple script.

This file shows why we use print
"""

# This is a comment
x = 1+2
x = 3*x
print(x)
```

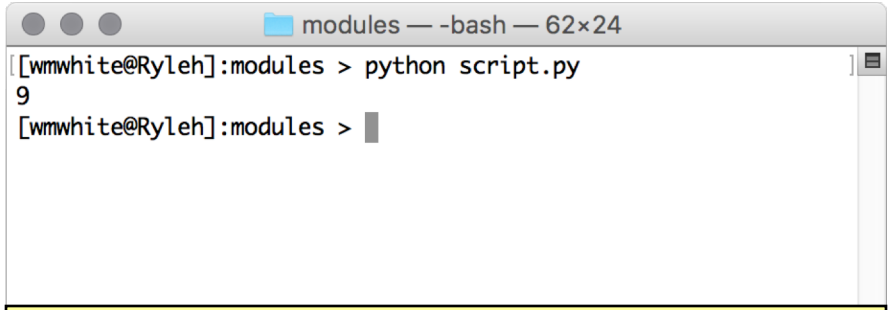
Scripts and Print Statements

module.py



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

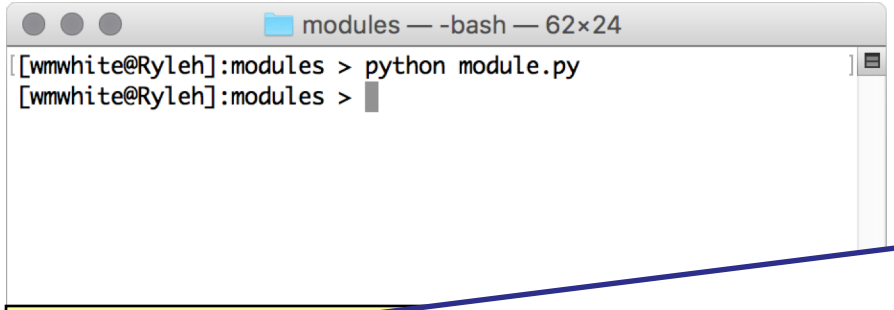
script.py



- 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… assignments
  - 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

User Input

```
>>> input('Type something')
Type somethingabc
'abc'
>>> input('Type something: ')
Type something: abc
'abc'
>>> x = input('Type something: ')
Type something: abc
>>> x
'abc'
```

No space after the prompt.

Proper space after prompt.

Assign result to variable.

Making a Script Interactive

```
"""
A script showing off input.

This file shows how to make a script
interactive.
"""

x = input("Give me a something: ")
print("You said: "+x)
```

```
[wmw2] folder> python script.py
Give me something: Hello
You said: Hello
[wmw2] folder> python script.py
Give me something: Goodbye
You said: Goodbye
[wmw2] folder>
```

Not using the interactive shell

Numeric Input

- input returns a string
  - Even if looks like int
  - It cannot know better
- You must convert values
  - int(), float(), bool(), etc.
  - Error if cannot convert
- One way to program
  - But it is a *bad* way
  - Cannot be automated

```
>>> x = input('Number: ')
Number: 3
>>> x
'3'
>>> x + 1
TypeError: must be str, not int
>>> x = int(x)
>>> x+1
4
```

Value is a string.

Must convert to int.

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

plusone.py — ~/Documents/Professional/Courses/CS-1110/Lec...

plusone.py

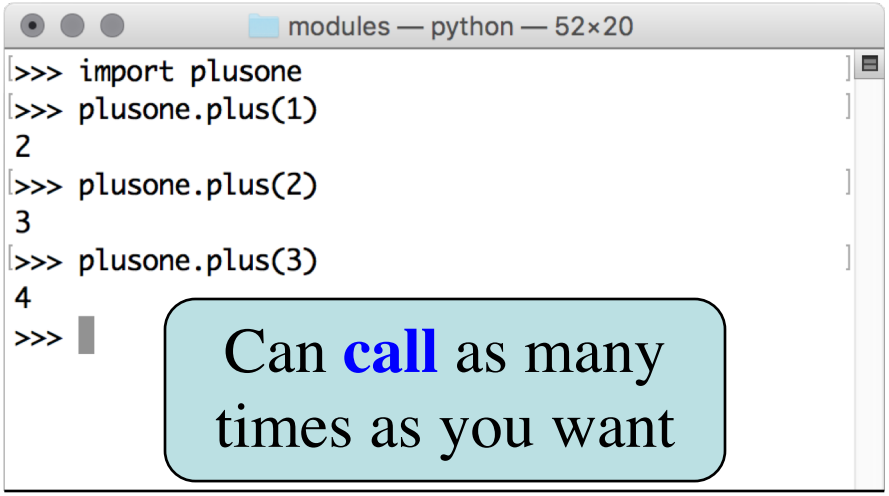
```
1  """
2  A module with a function definition
3
4  Author: Walker M. White (wmw2)
5  Date:   August 25, 2017 (Python 3 Version)
6  """
7
8  def plus(n):
9      """
10     Returns: the value of n+1
11     """
12     return (n+1)
13
```

plusone.py 1:1 LF UTF-8 MagicPython 0 files

Next Time: Defining Functions

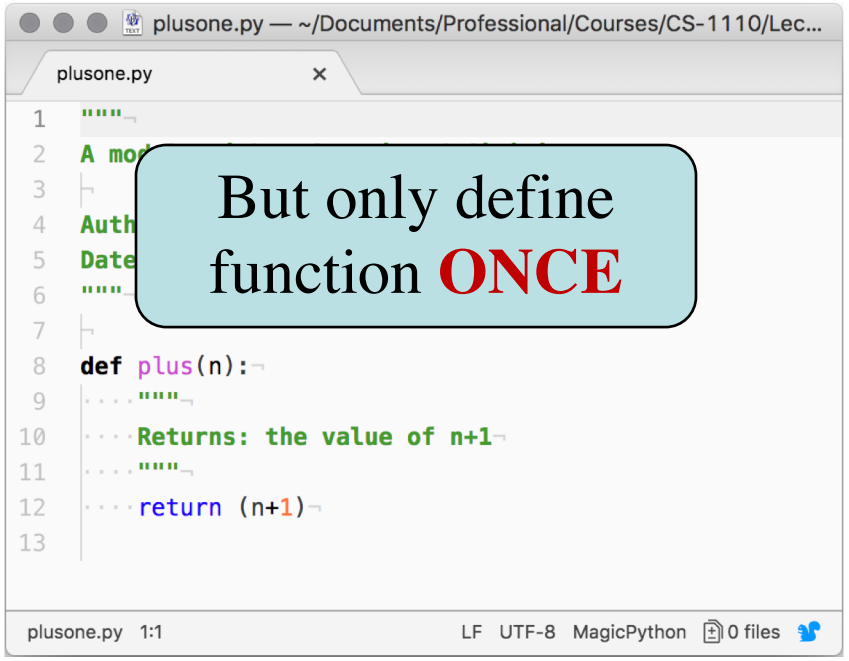
Function Call

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



Function Definition

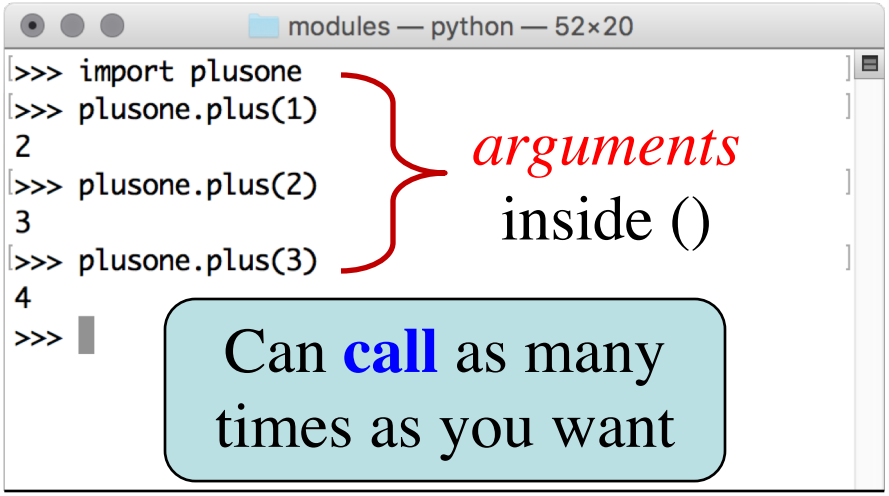
- Command to **do** the function
- Belongs inside a module



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