

Lecture 3

Functions & Modules

Announcements

Reminders

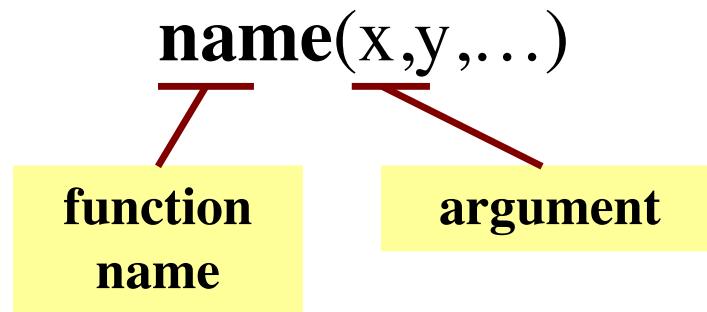
- Have graded **AI quiz**
 - Take now if have not
 - If made 9/10, are okay
 - Else must retake
- **Survey 0** is still open
 - For participation score
 - Must complete 75%
- Must access in CMS

Reading Suggestions

- If reading the text...
 - Chapter 3 (but not 3.9)
 - Sections 8.1, 8.2, 8.5, 8.8
 - As always, text *optional*
- Skim the **Python API**
 - And the intros API
 - See course **Resources**
 - But also not required

Function Calls

- Python supports expressions with math-like functions
 - A function in an expression is a *function call*
- **Function calls** have the form



- **Arguments** are
 - **Expressions**, not values
 - Separated by commas

Built-In Functions

- Python has several math functions
 - `round(2.34)`
 - `max(a+3,24)`
- You have seen many functions already
 - Type casting functions: `int()`, `float()`, `bool()`
- Documentation of all of these are online
 - <https://docs.python.org/3/library/functions.html>
 - Most of these are too advanced for us right now

Functions as Commands/Statements

- Most functions are expressions.
 - You can use them in assignment statements
 - **Example:** `x = round(2.34)`
- But some functions are **commands**.
 - They instruct Python to do something
 - Help function: `help()`
 - Quit function: `quit()`
- How know which one? Read documentation.

These take no
arguments

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
```

To access math
functions

```
>>> math.cos(0)
```

```
1.0
```

```
>>> cos(0)
```

Functions
require math
prefix!

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
```

To access math
functions

```
>>> math.cos(0)
```

```
1.0
```

```
>>> cos(0)
```

Functions
require math
prefix!

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'cos' is not defined

```
>>> math.pi
```

Module has
variables too!

```
3.141592653589793
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

Example: Module math

```
>>> import math
```

To access math functions

```
>>> math.cos(0)
```

Functions require math prefix!

```
1.0
```

```
>>> cos(0)
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'cos' is not defined

```
>>> math.pi
```

Module has variables too!

```
3.141592653589793
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

Other Modules

- **os**
 - Information about your OS
 - Cross-platform features
- **random**
 - Generate random numbers
 - Can pick any distribution
- **introscs**
 - Custom module for the course
 - Will be used a lot at start

Using the from Keyword

```
>>> import math
```

```
>>> math.pi
```

```
3.141592653589793
```

```
>>> from math import pi
```

```
>>> pi
```

No prefix needed
for variable pi

```
3.141592653589793
```

```
>>> from math import *
```

```
>>> cos(pi)
```

```
-1.0
```

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, introcs
 - Both have func. Turtle()

Reading the Python Documentation

The screenshot shows a web browser displaying the Python documentation for the `math` module. The URL is `docs.python.org/3/library/math.html`. The page title is "9.2. math — Mathematical functions". The left sidebar contains a "Table Of Contents" for the `math` module, listing sections such as Number-theoretic and representation functions, Power and logarithmic functions, Trigonometric functions, Angular conversion, and Hyperbolic functions. The main content area starts with a brief introduction: "This module is always available. It provides access to the mathematical functions defined by the C 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." Below this, a note states: "The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats."

`math.ceil(x)`

Return the ceiling of x , the smallest integer greater than or equal to x .

Next topic

9.3. `cmath` — Mathematical functions for complex numbers

This Page

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

Return a float with the magnitude (absolute value) of x but the sign of y . On platforms that support signed zeros, `copysign(1.0, -0.0)` returns `-1.0`.

`math.fabs(x)`

Return the absolute value of x .

`math.factorial(x)`

Return x factorial. Raises `ValueError` if x is not integral or is negative.

`math.floor(x)`

Return the floor of x , the largest integer less than or equal to x . If x is not a float, delegates to `x.floor()`, which should return an `Integral` value.

`math.fmod(x, y)`

Return `fmod(x, y)`, as defined by the platform C library. Note that the C standard is that `fmod(x, y)` be exactly (mathematically; to infinite precision) equal to $x - ny$ for some integer n such that the result has the same sign as x and magnitude less than `abs(y)`. Python's `x % y` returns a result with the sign of y instead, and may not be exactly computable for float arguments. For example, `fmod(-1e-100, 1e100)` is `-1e-100`, but the result of Python's `-1e-100 % 1e100` is `1e100-1e-100`, which cannot be represented exactly as a float.

<http://docs.python.org/3/library>

Reading the Python Documentation

The screenshot shows a web browser displaying the Python documentation for the `math` module. The URL is `http://docs.python.org/3/library/math.html`. The page title is "9.2. math — Mathematical functions". A green callout box labeled "Function name" points to the `math.ceil(x)` function. Another green callout box labeled "Possible arguments" points to the parameter `x` in the function definition. A third green callout box labeled "Module" points to the `math` module name. A green callout box labeled "What the function evaluates to" points to the description of the function's return value: "Return the ceiling of `x`, the smallest integer greater than or equal to `x`". The page also contains descriptions for other functions like `floor` and `fmod`.

Function name

Possible arguments

Module

What the function evaluates to

`math.ceil(x)`

Return the ceiling of `x`, the smallest integer greater than or equal to `x`.

`math.floor(x)`

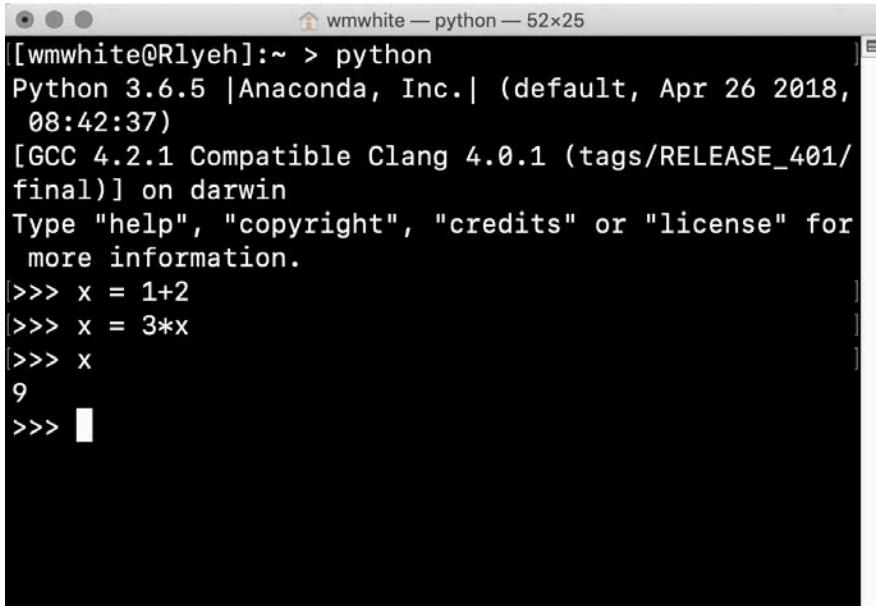
Return the floor of `x`, the largest integer less than or equal to `x`. If `x` is not a float, delegates to `x.__floor__()`, which should return an `Integral` value.

`math.fmod(x, y)`

Return `fmod(x, y)`, as defined by the platform C library. Note that the C standard is that `fmod(x, y)` be exactly (mathematically; to infinite precision) equal to `x - n*y` for some integer `n` such that the result has the same sign as `x` and magnitude less than `abs(y)`. Python's `x % y` returns a result with the sign of `y` instead, and may not be exactly computable for float arguments. For example, `fmod(-1e-100, 1e100)` is `-1e-100`, but the result of Python's `-1e-100 % 1e100` is `1e100-1e-100`, which cannot be

<http://docs.python.org/3/library>

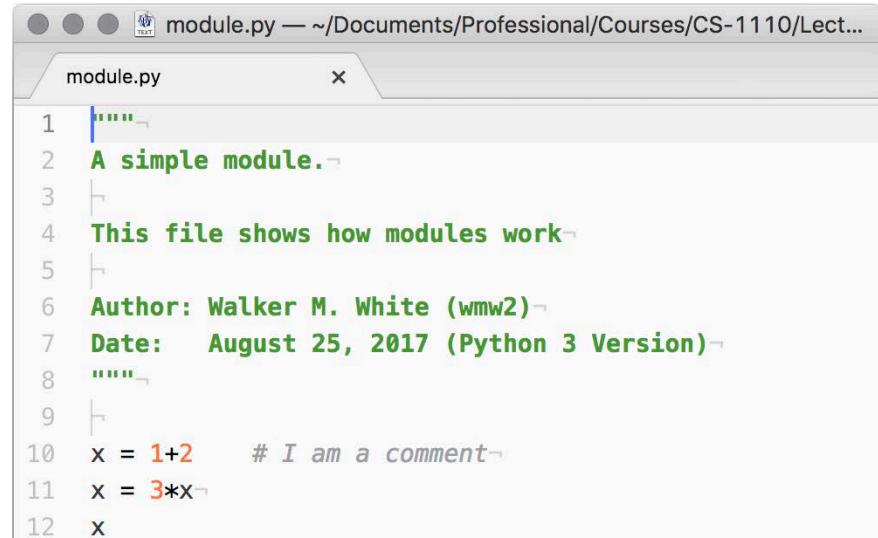
Interactive Shell vs. Modules



A screenshot of a terminal window titled "wmwhite — python — 52x25". The window shows the Python 3.6.5 interactive shell. The user has typed several commands:

```
[wmwhite@Rlyeh]:~ > python
Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018,
 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/
final)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.

>>> x = 1+2
>>> x = 3*x
>>> x
9
>>> 
```



A screenshot of a code editor window titled "module.py". The file contains the following Python code:

```
"""
A simple module.

This file shows how modules work

Author: Walker M. White (wmw2)
Date: August 25, 2017 (Python 3 Version)

"""

x = 1+2      # I am a comment
x = 3*x
x
```

- 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
```

Single line comment

(not executed)

```
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
```

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
```

“**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>
```

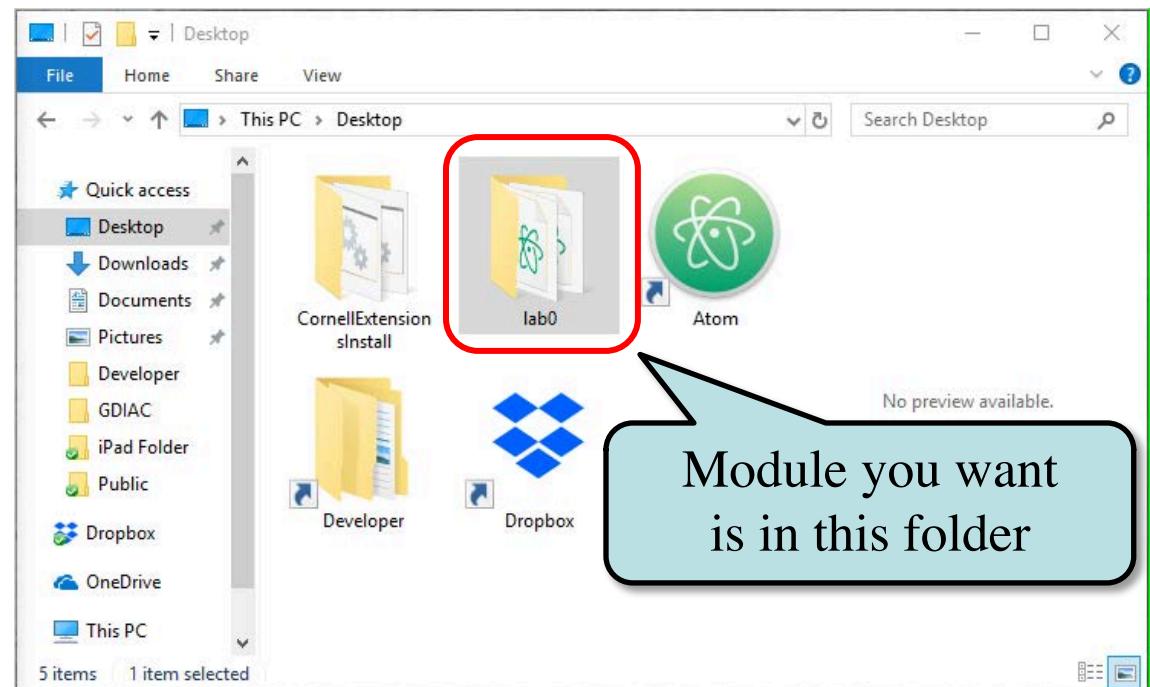
```
NameError: name 'x' is not defined
```

```
>>> module.x
```

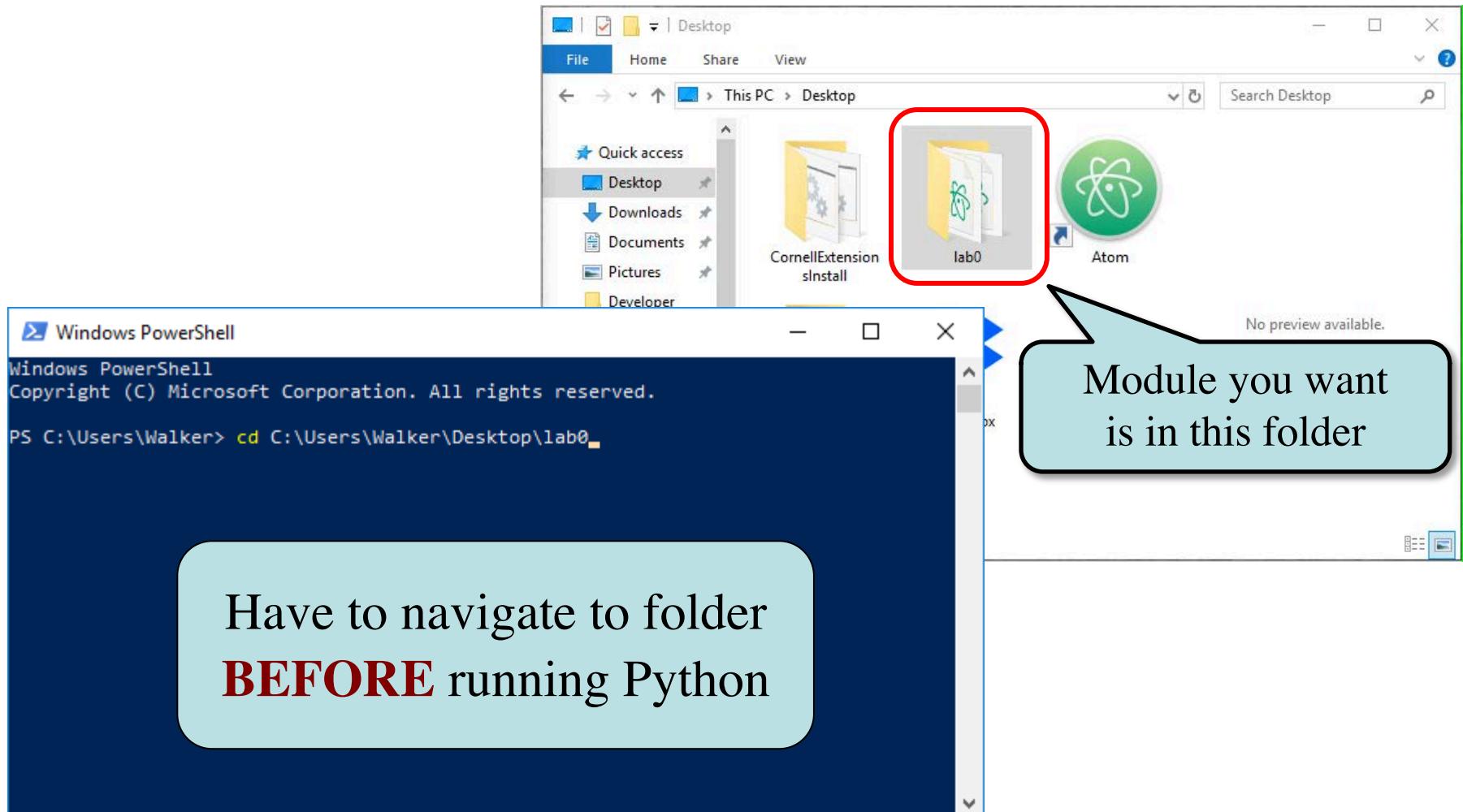
```
9
```

```
>>> help(module)
```

Modules Must be in Working Directory!



Modules Must be in Working Directory!



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
```

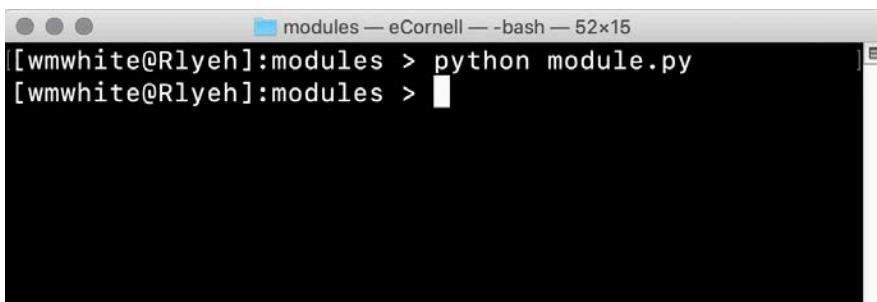
```
x = 3*x
```

```
print(x)
```



Scripts and Print Statements

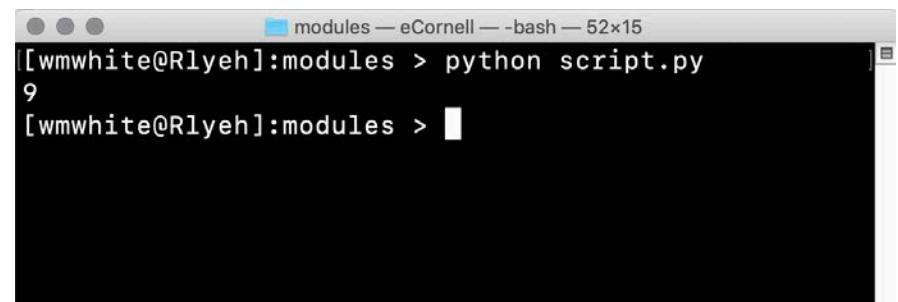
module.py



```
[wmwhite@Rlyeh]:modules > python module.py
[wmwhite@Rlyeh]:modules > 
```

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

script.py



```
[wmwhite@Rlyeh]:modules > python script.py
9
[wmwhite@Rlyeh]:modules > 
```

- 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

```
[wmwhite@Rlyeh]:modules > python module.py  
[wmwhite@Rlyeh]:modules > 
```

script.py

```
[wmwhite@Rlyeh]:modules > python script.py  
9  
[wmwhite@Rlyeh]:modules > 
```

- Looks like a function definition
- Python
 - Executed the assignments
 - Skipped the last line ('x' is not a statement)

When you run a script,
only statements are executed

- this time!
- the following:
- Executed the assignments
 - Executed the last line
(Prints the contents of x)

User Input

```
>>> input('Type something')
```

```
Type somethingabc
```

```
'abc'
```

No space after the prompt.

```
>>> input('Type something: ')
```

```
Type something: abc
```

```
'abc'
```

Proper space after prompt.

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

```
Type something: abc
```

```
>>> x
```

Assign result to variable.

```
'abc'
```

9/5/19

Making a Script Interactive

====

A script showing off input.

This file shows how to make a script interactive.

====

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

```
[wmw2] folder> python script.py
```

```
Give me something: Hello
```

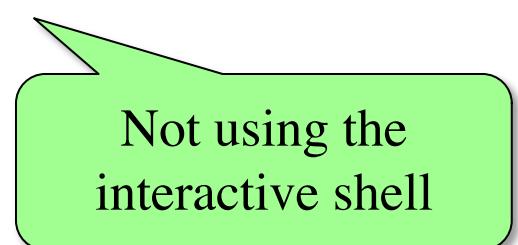
```
You said: Hello
```

```
[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.

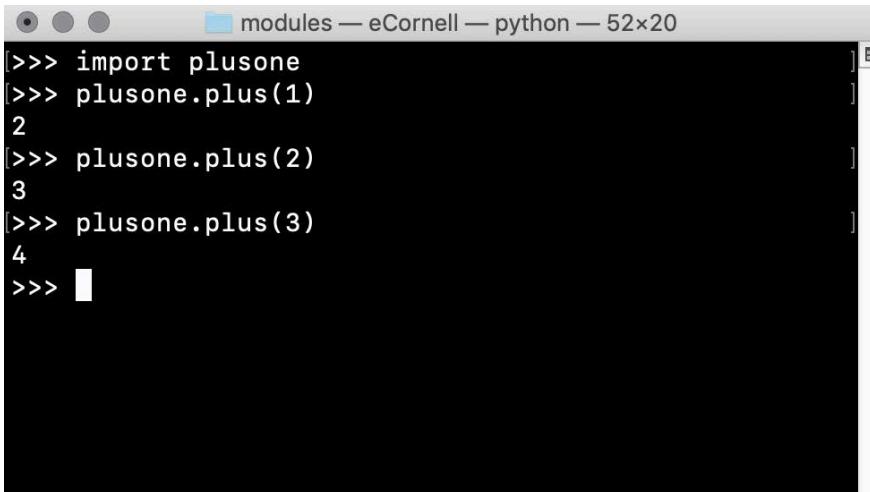
# Next Time: Defining Functions

---

## Function Call

---

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



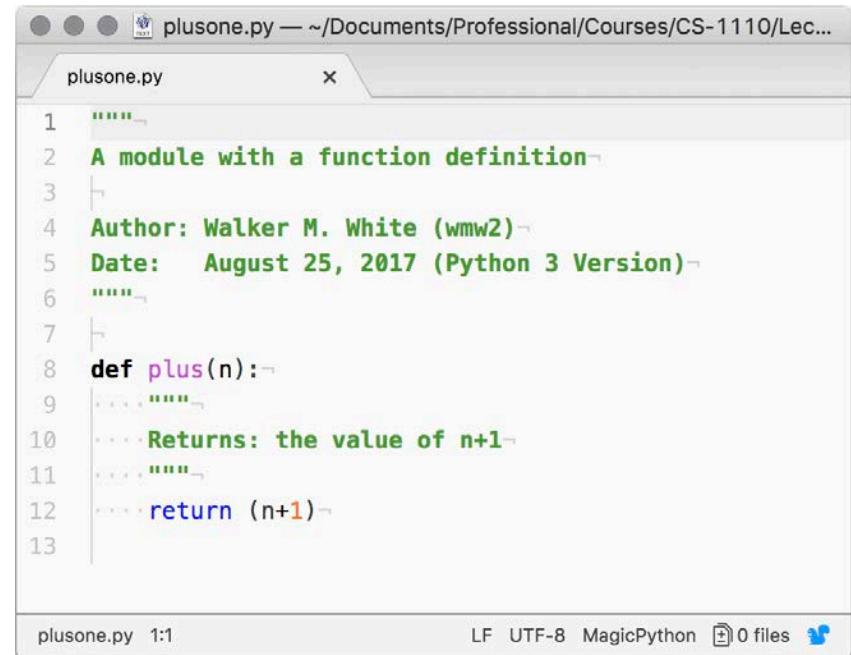
```
modules — eCornell — python — 52x20
>>> import plusone
>>> plusone.plus(1)
2
>>> plusone.plus(2)
3
>>> plusone.plus(3)
4
>>>
```

A screenshot of a terminal window titled "modules — eCornell — python — 52x20". It shows four lines of Python code being run at the prompt (>>>). The output is the result of each call to the "plus" function: 2, 3, and 4 respectively.

## 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
```

A screenshot of a code editor window titled "plusone.py". The code defines a function named "plus" that takes a parameter "n" and returns "n+1". The code is annotated with docstrings and comments. The status bar at the bottom indicates the file is 1:1, uses LF line endings, and is in UTF-8 encoding.

# Next Time: Defining Functions

## Function Call

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

A screenshot of a terminal window titled "modules — eCornell — python — 52x20". The window contains the following Python code:

```
>>> import plusone
>>> plusone.plus(1)
2
>>> plusone.plus(2)
3
>>> plusone.plus(3)
4
>>>
```

A red curly brace is placed over the three calls to `plusone.plus()`, with the text "arguments inside ()" written next to it.

**Can call as many times as you want**

## Function Definition

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

A screenshot of a code editor window titled "plusone.py — ~/Documents/Professional/Courses/CS-1110/Lec...". The window contains the following Python code:

```
1
2
3
4
5
6
7
8 def plus(n):
9 """
10 Returns: the value of n+1
11 """
12 return (n+1)
13
```

A blue rounded rectangle highlights the first two lines of the function definition, with the text "But only define function **ONCE**" written inside it.