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,…)

function name — **argument**

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

Arguments can be any **expression**

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
```

To access math functions

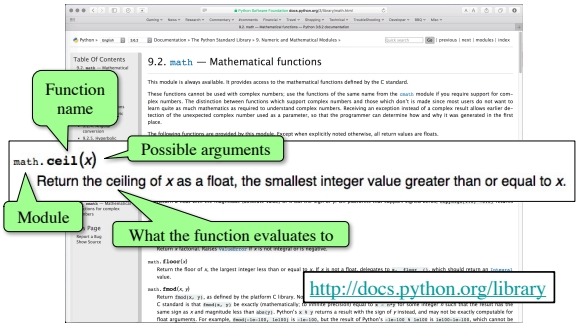
Functions require math prefix!

Module has variables too!

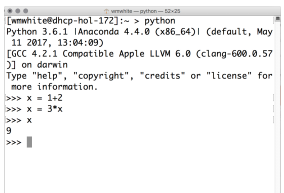
Other Modules

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

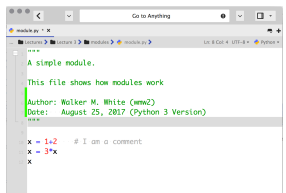
Reading the Python Documentation



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

```
""" 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
```

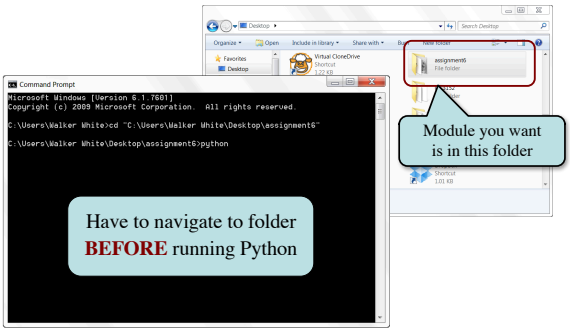
"**Module data**" must be prefixed by module name

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 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.y
```

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)
```

Only difference

Next Time: Defining Functions

Function Call

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

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

arguments inside ()

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

Function Definition

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

```
def plus(n):
    """
    Returns: the value of n+1
    """
    return (n+1)
```

But only define function **ONCE**

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