Lecture 4

Defining Functions

Academic Integrity Quiz

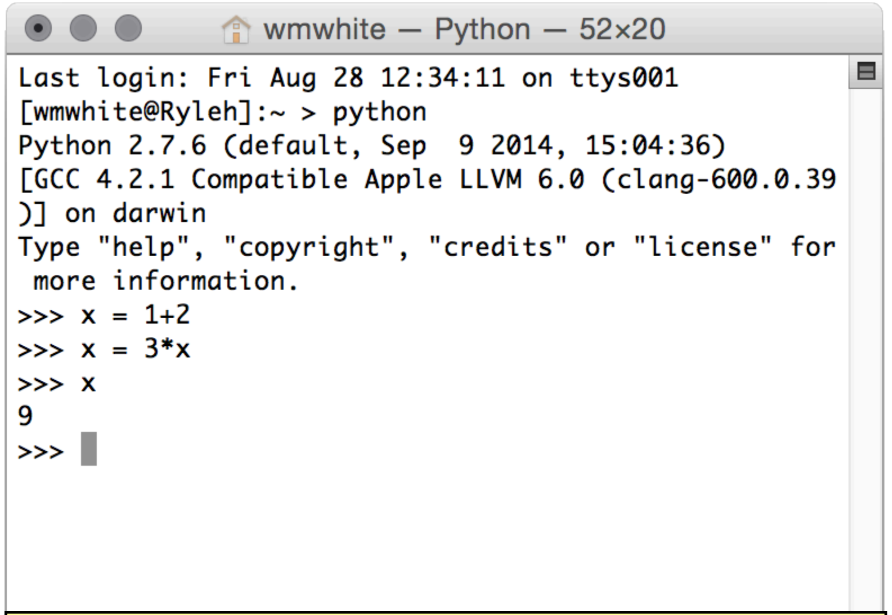
- **Remember**: quiz about the course AI policy
  - Have posted grades for completed quizes
  - Right now, missing ~70 enrolled students
  - If did not receive perfect, take it again
- If you are not aware of the quiz
  - Go to http://www.cs.cornell.edu/courses/cs1110/
  - Click **Academic Integrity** in side bar
  - Read and take quiz in CMS

Recall: Modules

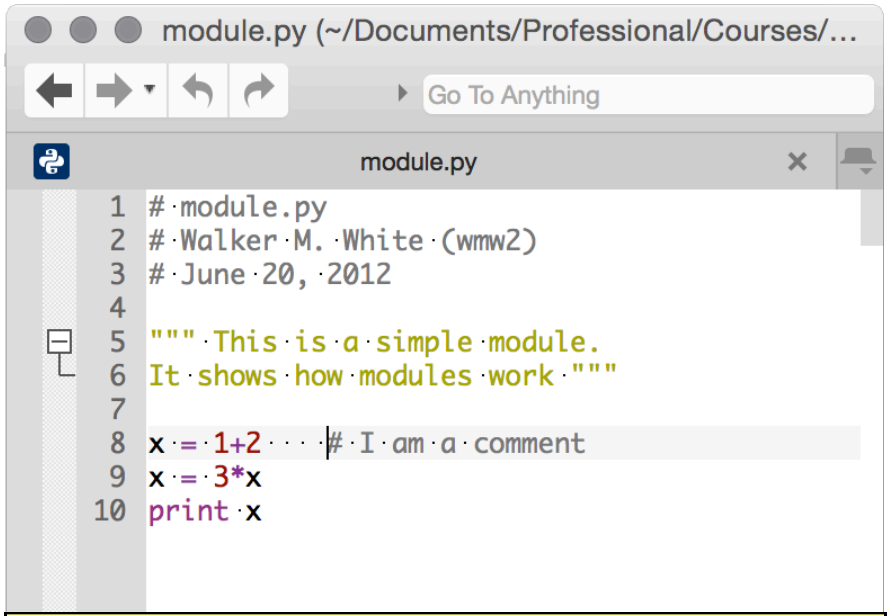
- Modules provide extra functions, variables
  - **Example**: math provides math.cos(), math.pi
  - Access them with the `import` command
- Python provides a lot of them for us
- **This Lecture**: How to make modules
  - Komodo Edit to *make* a module
  - Python to *use* the module

$\Big\}$ Two different programs

Python Shell vs. Modules



```
Last login: Fri Aug 28 12:34:11 on ttys001
[wmwhite@Ryleh]:~ > python
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39
)] on darwin
Type "help", "copyright", "credits" or "license" for
 more information.
>>> x = 1+2
>>> x = 3*x
>>> x
9
>>> 
```



```
# module.py
# Walker M. White (wmw2)
# June 20, 2012

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

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

- 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
- Run 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

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

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

Single line comment
(not executed)

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

Using a Module

Module Contents

```
# module.py

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

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

Single line comment
(not executed)

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

Commands
Executed on 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
```

Single line comment
(not executed)

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

Commands
Executed on import

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
```

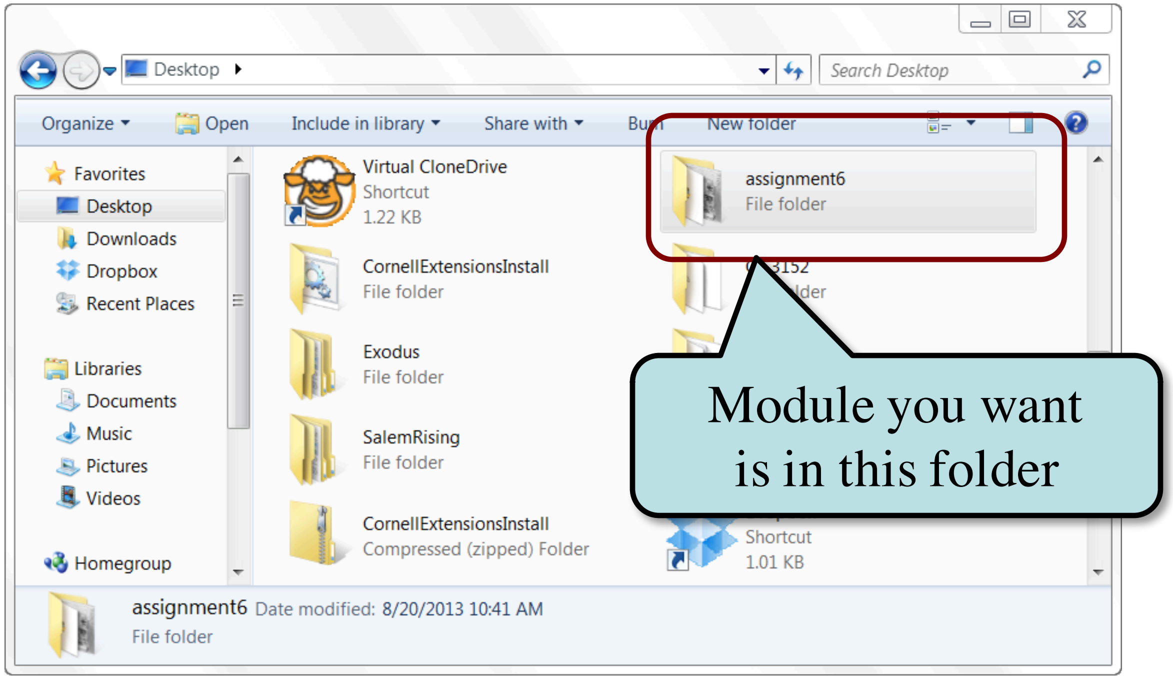
"**Module data**" must be prefixed by module name

Prints **docstring** and module contents

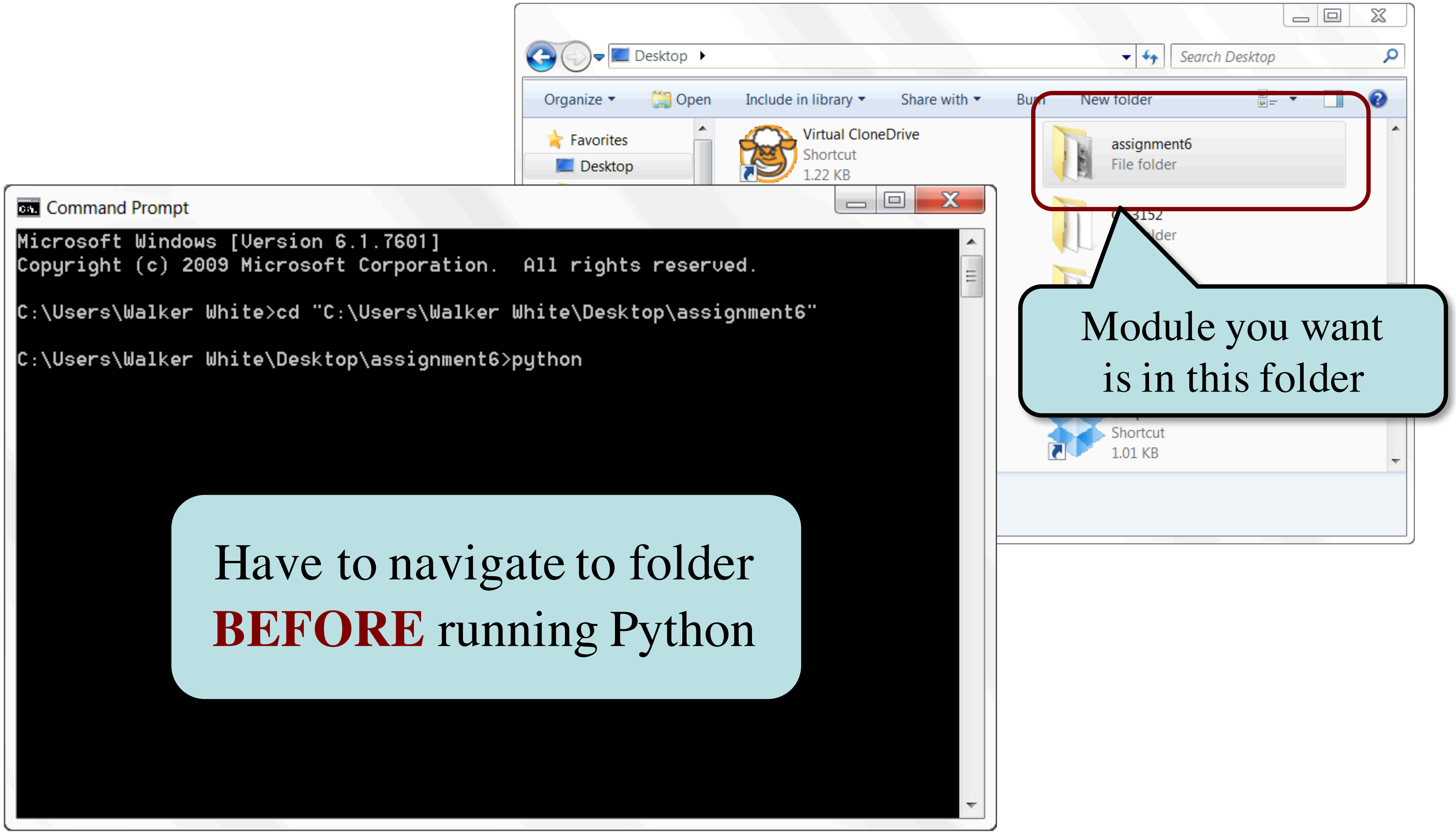
Python Shell

```
>>> 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!



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Walker White>cd "C:\Users\Walker White\Desktop\assignment6"

C:\Users\Walker White\Desktop\assignment6>python
```

Have to navigate to folder **BEFORE** running Python

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
greet('Walker')
```

Function Definition

- Defines what function **does**

```
def greet(n):
    print 'Hello '+n+'!'
```

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
greet('Walker')
```

Function Definition

- Defines what function **does**

```
def greet(n):
      print 'Hello '+n+'!'
```

Function **Header**

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
greet('Walker')
```

Function Definition

- Defines what function **does**

```
def greet(n):
    print 'Hello '+n+'!'
```

Function **Header**

Function **Body** (indented)

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
greet('Walker')
```

Function Definition

- Defines what function **does**

```
def greet(n):
    print 'Hello '+n+'!'
```

Function **Header**

declaration of **parameter** n

Function **Body** (indented)

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
greet('Walker')
```

argument to assign to n

Function Definition

- Defines what function **does**

```
def greet(n):
    print 'Hello '+n+'!'
```

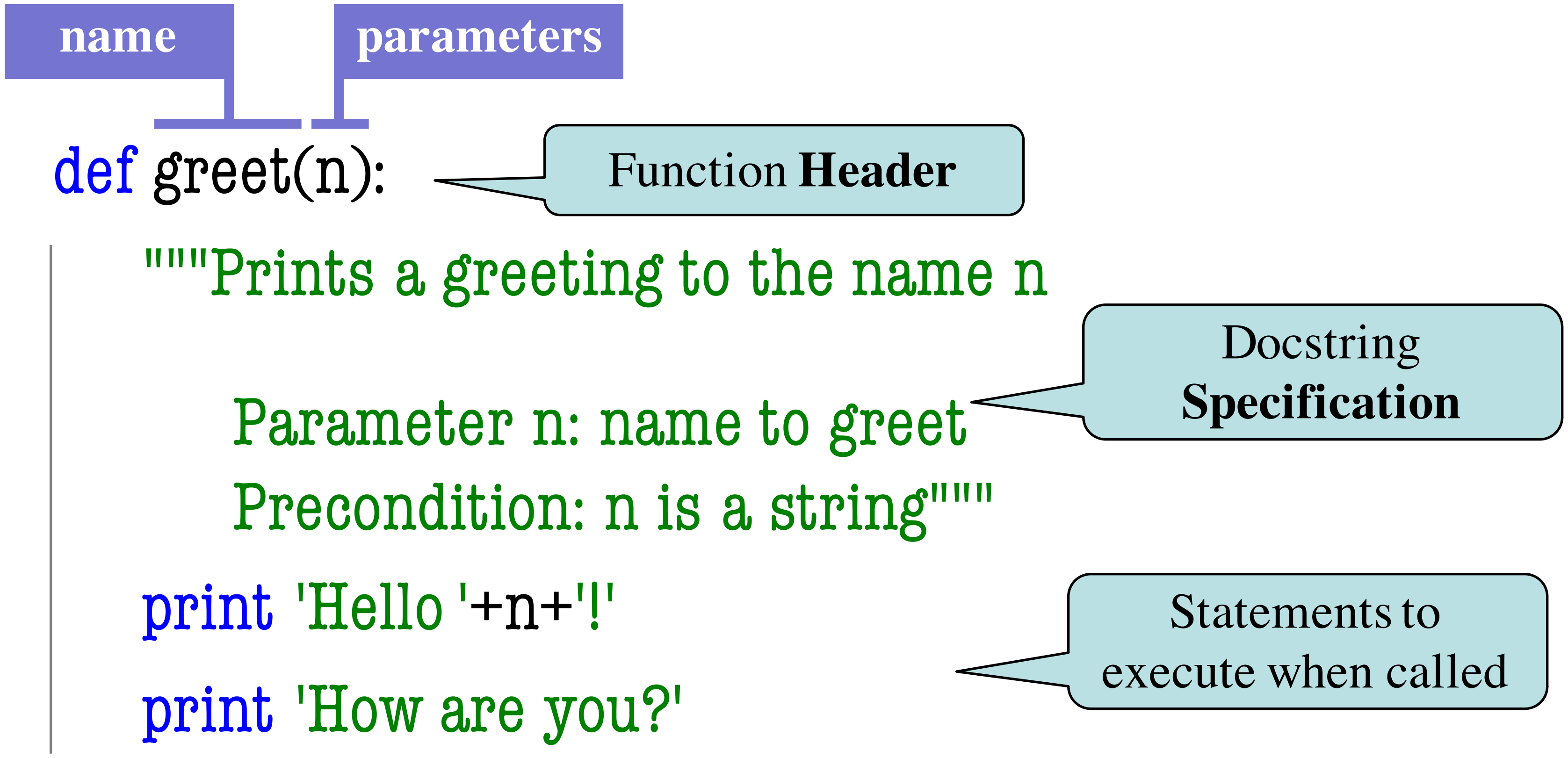
Function **Header**

declaration of **parameter** n

Function **Body** (indented)

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

Anatomy of a Function Definition


name
parameters
Function **Header**
Docstring **Specification**
Statements to execute when called

```
def greet(n):
    """Prints a greeting to the name n

       Parameter n: name to greet
       Precondition: n is a string"""
    print 'Hello '+n+'!'
    print 'How are you?'
```

Anatomy of a Function Definition

name

parameters

```
def greet(n):
    """Prints a greeting to the name n

       Parameter n: name to greet
       Precondition: n is a string"""
    print 'Hello '+n+'!'
    print 'How are you?'
```

Function **Header**

Docstring **Specification**

Statements to execute when called

The vertical line indicates indentation

Use vertical lines when you write Python on **exams** so we can see indentation

Procedures vs. Fruitful Functions

Procedures	Fruitful Functions
• Functions that **do** something	• Functions that give a **value**
• Call them as a **statement**	• Call them in an **expression**
• Example: `greet('Walker')`	• Example: `x = round(2.56,1)`

Historical Aside

- Historically "function" = "fruitful function"
- But now we use "function" to refer to both

The **return** Statement

- Fruitful functions require a **return statement**
- **Format**: return <*expression*>
  - Provides value when call is used in an expression
  - Also stops executing the function!
  - Any statements after a **return** are ignored
- **Example**: temperature converter function

```
def to_centigrade(x):
    """Returns: x converted to centigrade"""
    return 5*(x-32)/9.0
```

Print vs. Return

Print

- Displays a value on screen
  - Used primarily for **testing**
  - Not useful for calculations

```
def print_plus(n):
    print (n+1)
>>> x = plus_one(2)
3
>>>
```

Return

- Defines a function's value
  - Important for **calculations**
  - But does not display anything

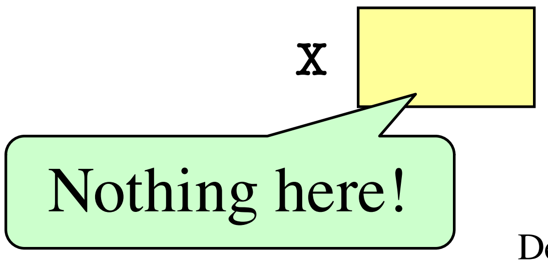
```
def return_plus(n):
    return (n+1)
>>> x = plus_one(2)
>>>
```

Print vs. Return

Print

- Displays a value on screen
  - Used primarily for **testing**
  - Not useful for calculations

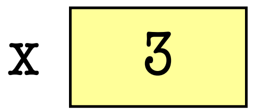
```
def print_plus(n):
    print (n+1)
>>> x = plus_one(2)
3
>>>
```



Return

- Defines a function's value
  - Important for **calculations**
  - But does not display anything

```
def return_plus(n):
    return (n+1)
>>> x = plus_one(2)
>>>
```



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 command shell
3. Move to the directory with this file
4. Start Python (type `python`)
5. Import the module
6. Try out the function

Aside: Constants

- Modules often have variables outside a function
  - We call these global variables
  - Accessible once you import the module
- Global variables should be **constants**
  - Variables that never, ever change
  - Mnemonic representation of important value
  - **Example**: `math.pi`, `math.e` in `math`
- In this class, constant names are **capitalized**!
  - So we can tell them apart from non-constants

Module Example: Temperature Converter

```
# temperature.py
"""Conversion functions between fahrenheit and centrigrade"""

# Functions
def to_centigrade(x):
    """Returns: x converted to centigrade"""
    return 5*(x-32)/9.0

def to_fahrenheit(x):
    """Returns: x converted to fahrenheit"""
    return 9*x/5.0+32

# Constants
FREEZING_C = 0.0    # temp. water freezes
```

Style Guideline:
Two blank lines between function definitions

Example from Previous Lecture

```
def second_in_list(s):
    """Returns: second item in comma-separated list
    The final result does not have any whitespace on edges
    Parameter s: The list of items
    Precondition: s is a string of items separated by commas."""
    startcomma = s.index(',')
    tail = s[startcomma+1:]
    endcomma = tail.index(',')
    item = tail[:endcomma].strip()
    return item
```

See commalist.py