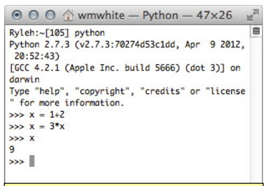
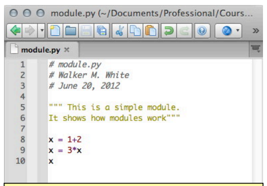
One-on-One Sessions

- Starting next week: 1/2-hour one-on-one sessions
  - Bring computer and work with instructor, TA or consultant
  - Hands on, dedicated help with Lab 2 and/or Lab 3
  - To prepare for assignment, **not for help on assignment**
- **Limited availability: we cannot get to everyone**
  - **Students with experience or confidence should hold back**
- Sign up online in CMS: first come, first served
  - Choose assignment One-on-One
  - Pick a time that works for you; will add slots as possible
  - Can sign up starting at 1pm **THURSDAY**

Python Shell vs. Modules



```
Ryleh:~[105] python
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012,
 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on
darwin
Type "help", "copyright", "credits" or "license
" for more information.
>>> x = 1+2
>>> x = 3*x
>>> x
9
>>>
```



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

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

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

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

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**

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

    Precondition: n is a string
    representing a person's name"""
    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

- Functions that **do** something
- Call them as a **statement**
- Example: `greet('Walker')`

Fruitful Functions

- Functions that give a **value**
- Call them in an **expression**
- 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
```

Module Example: Temperature Converter

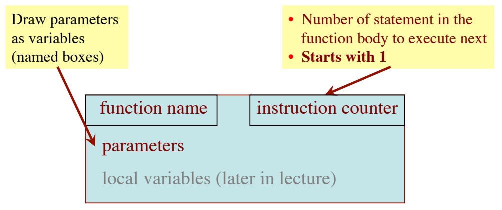
```
# temperature.py
"""Conversion functions between fahrenheit and centrigrade"""

# 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

How Do Functions Work?

Draw template on a piece of paper

- **Function Frame**: Representation of function call
- A **conceptual model** of Python

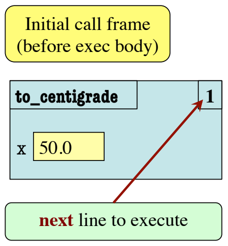


Draw parameters as variables (named boxes)
• Number of statement in the function body to execute next
• Starts with 1
function name
instruction counter
parameters
local variables (later in lecture)

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

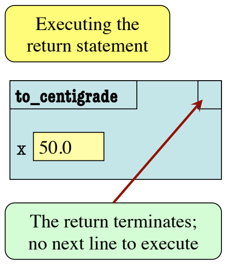
```
   def to_centigrade(x):
1      return 5*(x-32)/9.0
```



Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
   def to_centigrade(x):
1      return 5*(x-32)/9.0
```



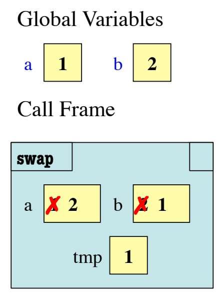
Executing the return statement
to_centigrade
x
50.0
The return terminates; no next line to execute

Call Frames vs. Global Variables

- This does not work:

```
   def swap(a,b):
       """Swap vars a & b"""
1      tmp = a
2      a = b
3      b = tmp
```

```
>>> a = 1
>>> b = 2
>>> swap(a,b)
```



Global Variables
a
1
b
2
Call Frame
swap
a
2
b
1
tmp
1