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

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

Using a Module

<table>
<thead>
<tr>
<th>Module Contents</th>
<th>Python Shell</th>
</tr>
</thead>
<tbody>
<tr>
<td># module.py</td>
<td>&gt;&gt;&gt; import module</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; x</td>
</tr>
<tr>
<td>&quot;&quot;&quot; This is a simple module. It shows how modules work&quot;&quot;</td>
<td>Traceback (most recent call last): File &quot;&lt;stdin&gt;&quot;, line 1, in &lt;module&gt; NameError: name 'x' is not defined</td>
</tr>
<tr>
<td>x = 1+2</td>
<td>&gt;&gt;&gt; module.x</td>
</tr>
<tr>
<td>x = 3*x</td>
<td>9</td>
</tr>
<tr>
<td>x</td>
<td>&gt;&gt;&gt; help(module)</td>
</tr>
</tbody>
</table>

We Write Programs to Do Things

- Functions are the key doers
  - Function Call
    - Command to do the function
      - Example: greet('Walker')
  - Function Definition
    - Defines what function does
      - Example: def greet(n):
          - print 'Hello ' + n + '!
          - print 'How are you?'

Anatomy of a Function Definition

```
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?'
```

Procedures vs. Fruitful Functions

<table>
<thead>
<tr>
<th>Procedures</th>
<th>Fruitful Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions that do something</td>
<td>Functions that give a value</td>
</tr>
<tr>
<td>Call them as a statement</td>
<td>Call them in an expression</td>
</tr>
<tr>
<td>Example: greet('Walker')</td>
<td>Example: x = round(2.56,1)</td>
</tr>
</tbody>
</table>

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 centigrade""

# 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

How Do Functions Work?

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

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

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

Call Frames vs. Global Variables

- This does not work:
def swap(a,b):
  """Swap vars a & b""
  tmp = a
  a = b
  b = tmp

Global Variables

```python
a 1
b 2
```

Call Frame

```python
swap
```

```python
a 2
b 1
```

```
```