We Write Programs to Do Things

- Functions are the **key doers**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Function Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Command to do the function</td>
<td>• Defines what function does</td>
</tr>
</tbody>
</table>

```python
>>> plus(23)
24
```

- **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

<table>
<thead>
<tr>
<th>name</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>def plus(n):</td>
<td></td>
</tr>
</tbody>
</table>

```python
# Returns the number n+1

Parameter n: number to add to
Precondition: n is a number
```

```python
x = n+1
return x
```

The return Statement

- **Format**: `return <expression>`
  - Used to evaluate function call (as an expression)
  - Also stops executing the function!
  - Any statements after a `return` are ignored

**Example**: temperature converter function

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

A More Complex Example

```python
def foo(a,b):
    #"Return something"
    Param a: number
    Param b: number
    x = a
    y = b
    return x*y+y
```

Understanding How Functions Work

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

Definition: `to_centigrade`

Text (Section 3.10) vs. Class

<table>
<thead>
<tr>
<th>Textbook</th>
<th>This Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>to_centigrade(x)</code></td>
<td><code>x -&gt; 50.0</code></td>
</tr>
<tr>
<td><code>to_centigrade</code></td>
<td><code>1</code></td>
</tr>
<tr>
<td><code>x</code></td>
<td><code>50.0</code></td>
</tr>
</tbody>
</table>

Definition: `to_centigrade(x):`

```
def to_centigrade(x):
    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

```python
def to_centigrade(x):
    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

```
to_centigrade
x  50.0
next line to execute
```

---

Call Frames vs. Global Variables

The specification is a lie:
```python
def swap(a,b):
    """Swap global a & b""
    tmp = a
da = b
b = tmp
```

Global Variables:
```
a  1
b  2
```

Call Frame:
```
swap
a  2
b  1
tmp 1
```

Example:
```python
>>> a = 1
>>> b = 2
>>> swap(a,b)
```

---

Function Access to Global Space

- All function definitions are in some module
- Call can access global space for that module
  - math.cos: global for math
  - temperature.to_centigrade uses global for temperature
- But cannot change values
  - Assignment to a global makes a new local variable!
  - Why we limit to constants

Exercise Time

Function Definition
```python
def foo(a,b):
    """Return something"
    Param x: a number
    Param y: a number"
    a = 3.5 # global space
    def change_a():
        a = 3.5 # local variable
        return a
```

Function Call
```python
>>> x = foo(3,4)
```

"""Show how globals work""
```python
get_a
```
```python
# globals.py
"""Show how globals work""
```python
# Change a variable!
```python
change_a()
```
```python
# return a # returns global
```

What does the frame look like at the start?