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>

```
>>> 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

```
def plus(n):
    """Returns the number n+1
    Precondition: n is a number"
    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

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

A More Complex Example

```
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

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

Text (Section 3.10) vs. Class

<table>
<thead>
<tr>
<th>Textbook</th>
<th>This Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>to_centigrade</code></td>
<td><code>to_centigrade</code></td>
</tr>
</tbody>
</table>

| `x = 50.0` | `x = 50.0` |

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

What is in the box?

- A: 2
- B: 3
- C: 16
- D: Nothing!
- E: I do not know

---

The vertical line indicates indentation

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

---

```
def plus(n):
    """Returns the number n+1
    Precondition: n is a number"
    x = n+1
    return x
```
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):
    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):
    return 5*(x-32)/9.0
```

Call Frames vs. Global Variables

The specification is a lie:

```
def swap(a,b):
    """Swap global a & b""
    tmp = a
    a = b
    b = tmp
```

Global Variables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Call Frame

<table>
<thead>
<tr>
<th>swap</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Calling `swap(a,b)`

```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

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

Function Definition

Function Call

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

What does the frame look like at the start?