Lecture 3

Defining Functions
Academic Integrity Quiz

• **Remember**: quiz about the course AI policy
  - Have posted grades for completed quizzes
  - Right now, missing ~40 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/cs1133/](http://www.cs.cornell.edu/courses/cs1133/)
  - 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

Two different programs
Functions are the **key doers**

**Function Call**
- Command to *do* the function

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

**Function Definition**
- Defines what function *does*

```python
def plus(n):
    return n+1
```

- **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**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Function Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Command to <strong>do</strong> the function</td>
<td>• Defines what function <strong>does</strong></td>
</tr>
<tr>
<td>&gt;&gt;&gt; <strong>plus</strong>(23)</td>
<td><strong>def</strong> <strong>plus</strong>(n):</td>
</tr>
<tr>
<td>24</td>
<td><strong>return</strong> n+1</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td></td>
</tr>
</tbody>
</table>

• **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**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Function Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Command to <strong>do</strong> the function</td>
<td>• Defines what function <strong>does</strong></td>
</tr>
</tbody>
</table>
| >>> `plus(23)` | `def plus(n):`
| 24 | `return n+1`
| >>> | 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

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

**Argument** to assign to n

**Function Definition**
- Defines what function **does**

```python
def plus(n):
    return n+1
```

**Declaration of parameter** 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
Anatomy of a Function Definition

**Function Definition**

```python
def plus(n):
    """Returns the number n+1
    Parameter n: number to add to
    Precondition: n is a number"
    x = n+1
    return x
```

- **Name**: plus
- **Parameters**: n
- **Docstring**: """"Returns the number n+1
    Parameter n: number to add to
    Precondition: n is a number"
"""
- **Statements to execute when called**: x = n+1
  return x
Anatomy of a Function Definition

**Function Header**

```python
def plus(n):
    """Returns the number n+1
    Parameter n: number to add to
    Precondition: n is a number"
    x = n+1
    return x
```

**Docstring Specification**

- The vertical line indicates indentation

**Statements to execute when called**

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

**Defining Functions**

9/1/16
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
```

9/1/16

Defining Functions
## A More Complex Example

<table>
<thead>
<tr>
<th>Function Definition</th>
<th>Function Call</th>
</tr>
</thead>
</table>
| ```python
def foo(a, b):
    """Return something
    Param a: number  
    Param b: number""
    x = a
    y = b
    return x*y+y
``` | ```
>>> x = 2
>>> foo(3, 4)
``` |

What is in the box?
A More Complex Example

Function Definition

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

Function Call

```python
>>> x = 2
>>> foo(3, 4)
```

What is in the box?

A: 2
B: 3
C: 16
D: Nothing!
E: I do not know
**A More Complex Example**

---

### Function Definition

```python
def foo(a,b):
    
    """Return something
    Param a: number
    Param b: number"

    x = a
    y = b
    return x*y+y
```

### Function Call

```python
>>> x = 2
>>> foo(3,4)
```

What is in the box?

- **A:** 2  **CORRECT**
- **B:** 3
- **C:** 16
- **D:** Nothing!
- **E:** I do not know
Understanding How Functions Work

- **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**

<table>
<thead>
<tr>
<th>function name</th>
<th>instruction counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameters</td>
<td>local variables (later in lecture)</td>
</tr>
</tbody>
</table>
Text (Section 3.10) vs. Class

<table>
<thead>
<tr>
<th>Textbook</th>
<th>This Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>to_centigrade</strong></td>
<td></td>
</tr>
<tr>
<td>x → 50.0</td>
<td></td>
</tr>
<tr>
<td><strong>Defining Functions</strong></td>
<td></td>
</tr>
<tr>
<td><code>def to_centigrade(x):</code></td>
<td><code>to_centigrade</code></td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x 50.0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Call:</strong> to_centigrade(50.0)</td>
<td></td>
</tr>
</tbody>
</table>

```
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
```

![Initial call frame (before exec body)]

![next line to execute]
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
```

Executing the return statement

Return statement creates a special variable for result
Example: \texttt{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
```

Executing the return statement

The return terminates; no next line to execute

9/1/16 Defining Functions
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
```

But don’t actually erase on an exam
The specification is a lie:

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

>>> a = 1
>>> b = 2
>>> swap(a,b)
```
The specification is a lie:

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

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

Global Variables

```
a  1
b  2
```

Call Frame

```
swap 2
```

```
a  1
b  2
tmp 1
```
The specification is a lie:

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

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

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

```
\[
\begin{array}{c}
\text{swap} \\
a \times 2 \\
tmp \\
\end{array}
\]
```

```
\[
\begin{array}{c}
a \times 2 \\
\text{swap} \\
tmp \\
\end{array}
\]
```
The specification is a **lie**:

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

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

Global Variables

```
a  1   b  2
```

Call Frame

```
swap
```

```
a  2   b  1
```

```
tmp  1
```
The specification is a lie:

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

>>> 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

```
# globals.py
"""Show how globals work""
a = 4  # global space

def get_a():
    return a  # returns global
```
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

```python
# globals.py
"""Show how globals work""
a = 4  # global space
def change_a():
    a = 3.5  # local variable
    return a
```
# Exercise Time

## Function Definition

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

## Function Call

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

What does the frame look like at the `start`?
Which One is Closest to Your Answer?

A: 

```
foo  
| a 3 | b 4 |
|     |     |
|     |     |
|     |     |
|     |     |
```

B: 

```
foo  
| a 3 | b 4 |
|     |     |
|     |     |
|     |     |
```

C: 

```
foo  
| a 3 | b 4 |
|     |     |
|     |     |
|     |     |
```

D: 

```
foo  
| a 3 | b 4 |
|     |     |
| x   | y   |
```

9/1/16
Defining Functions
28
### Exercise Time

#### Function Definition

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

#### Function Call

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

B:

```
<table>
<thead>
<tr>
<th>foo</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
</tr>
</tbody>
</table>
```
def foo(a, b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x*y+y

>>> x = foo(3,4)

B:

What is the next step?
Which One is Closest to Your Answer?

A:

```
foo
a 3 b 4
```

B:

```
foo
a 3 b 4
x 3
```

C:

```
foo
a 3 b 4
x 3
```

D:

```
foo
a 3 b 4
x 3 y
```
Exercise Time

Function Definition

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

Function Call

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

C:

```
     2
    foo
      a  3  b  4
     x  3
```

9/1/16  Defining Functions  32
def foo(a,b):
    '''Return something
    Param x: a number
    Param y: a number'''
    x = a
    y = b
    return x*y+y

>>> x = foo(3,4)

C:
What is the next step?
Which One is Closest to Your Answer?

A: foo
   a  3  b  4
   x  3  y  4
   RETURN 16

B: foo
   a  3  b  4
   x  3  y  4
   RETURN

C: foo
   a  3  b  4
   x  3  y  4
   RETURN

D: ERASE THE FRAME
Exercise Time

Function Definition

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

Function Call

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

A:

```
foo
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>
```

9/1/16  Defining Functions  35
## Exercise Time

### Function Definition

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

1. `x = a`
2. `y = b`
3. `return x*y+y`

### Function Call

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

**A:**

<table>
<thead>
<tr>
<th>foo</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>x</td>
<td>3</td>
</tr>
</tbody>
</table>

What is the **next step**?
Which One is Closest to Your Answer?

A: foo

RETURN 16

3

B: foo

RETURN 16

3

a  3  b  4
x  3  y  4

C: foo

RETURN 16

a  3  b  4
x  3  y  4

D: ERASE THE FRAME

9/1/16 Defining Functions 37
Exercise Time

Function Definition

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

Function Call

>>> x = foo(3,4)

C:

```
<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td>y</td>
<td>4</td>
</tr>
<tr>
<td>RETURN</td>
<td>16</td>
</tr>
</tbody>
</table>
```

"""
Function Definition

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

Function Call

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

C:

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
<td>b</td>
</tr>
<tr>
<td>x</td>
<td>3</td>
<td>y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RETURN</td>
</tr>
</tbody>
</table>
```

What is the next step?
Which One is Closest to Your Answer?

A: foo
RETURN 16

B: ERASE THE FRAME

C: foo
x 16

D: x 16
ERASE THE FRAME
def foo(a,b):
    """Return something
    Param x: a number
    Param y: a number"
    x = a
    y = b
    return x*y+y

>>> x = foo(3,4)

D:

ERASE THE FRAME
Exercise Time

Function Definition

```python
def foo(a,b):
    """Return something"
    Param x: a number
    Param y: a number"

    x = a
    y = b
    return x*y+y
```

Function Call

```python
>>> x = foo(3,4)
D:
Variable in global space
```

Variable in global space

16

ERASE THE FRAME
Visualizing Frames: The Python Tutor

```
1. def max(x, y):
2.     if x > y:
3.         return x
4.     return y
5.
6. a = 1
7. b = 2
8. max(a, b)
```

Diagram showing frames and objects:

- Global frame:
  - `max`
  - `a 1`
  - `b 2`

- Frames:
  - `max`

- Objects:
  - `max`
  - `x 1`
  - `y 2`
Visualizing Frames: The Python Tutor

```
1 def max(x,y):
2     if x > y:
3         return x
4     return y

5 a = 1
6 b = 2
7 max(a,b)
```

Global Space

Call Frame
Visualizing Frames: The Python Tutor

1. `def max(x,y):
   if x > y:
       return x
   return y

2. `a = 1
3. `b = 2
4. `max(a,b)``

Variables from second lecture go in here

Global Space

Call Frame
Visualizing Frames: The Python Tutor

```
def max(x, y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a, b)
```

Frames

Objects

Global frame:

```
max
```

```
x 1
y 2
```
Visualizing Frames: The Python Tutor

Line number marked here (sort-of)

Missing line numbers!
Next Time: Concrete Examples