Lecture 4

Defining Functions
• **Remember**: quiz about the course AI policy
  - Have posted grades for completed quizzes
  - Right now, missing ~125 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/cs11110/](http://www.cs.cornell.edu/courses/cs11110/)
  - 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
We Write Programs to Do Things

• Functions are the **key doers**

**Function Call**

• Command to **do** the function

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

**Function Definition**

• Defines what function **does**

```
def plus(n):
    return n+1
```
We Write Programs to Do Things

• 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**

**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**

**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
```

**Function Header**
- declaration of parameter `n`

**Function Body** (indented)
- 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
Anatomy of a Function Definition

def plus(n):

    """Returns the number n+1
    Parameter n: number to add to
    Precondition: n is a number"

    x = n + 1
    return x
Anatomy of a Function Definition

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

The vertical line indicates indentation
```

Use vertical lines when you write Python on exams so we can see indentation
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
  ```
## 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 More Complex Example

Function Definition

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

Function Call

>>> 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

function name  instruction counter
parameters
local variables (later in lecture)
Textbook vs. This Class

to_centigrade

x -> 50.0

Call: to_centigrade(50.0)

Definition:

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

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
```

Initial call frame (before exec body)

<table>
<thead>
<tr>
<th>to_centigrade</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
</tbody>
</table>

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

```
to_centigrade
x 50.0  RETURN 10.0
```

Return statement creates a special variable for result
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

The return terminates; no 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
```

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
```

```python
>>> 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
```

```console
>>> 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`: 3
- `a`: 2
- `b`: 2
- `tmp`: 1
Call Frames vs. Global Variables

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)
```

Global Variables

Call Frame

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>tmp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The specification is a **lie**:

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

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

---

Defining Functions
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
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
x 3
```

D:

```
foo
a 3 b 4
x y
```
Which One is Closest to Your Answer?

A: 
```
<table>
<thead>
<tr>
<th>foo</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
</tr>
</tbody>
</table>
```

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>
```

C: 
```
<table>
<thead>
<tr>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>x</td>
</tr>
</tbody>
</table>
```

D: 
```
<table>
<thead>
<tr>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>x</td>
</tr>
</tbody>
</table>
```

E: 
```
\_(ツ)_/¯
```
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 3</td>
<td>b 4</td>
</tr>
</tbody>
</table>
```
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)

B:

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

A: 

```
foo
```

```
a 3 b 4
```

```
2
```

B: 

```
foo
```

```
a 3 b 4
```

```
x 3
```

```
1
```

C: 

```
foo
```

```
a 3 b 4
```

```
x 3
```

```
2
```

D: 

```
foo
```

```
a 3 b 4
```

```
x 3 y
```

```
2
```
**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:**

![Diagram showing the function call and variable assignments]

---

9/1/16  Defining Functions
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:

```
What is the next step?
```

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

<table>
<thead>
<tr>
<th></th>
<th>foo</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a 3</td>
<td>b 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x 3</td>
<td>y 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>foo</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a 3</td>
<td>b 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x 3</td>
<td>y 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>foo</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a 3</td>
<td>b 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x 3</td>
<td>y 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

RETURN
## 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
```

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

### Function Call

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>
<tr>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td>y</td>
<td>4</td>
</tr>
</tbody>
</table>
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)
```

A:

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

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

A: foo 3
   RETURN 16

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

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

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)
C:
```

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

RETURN 16
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:

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td></td>
<td>4</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
```
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:

x
```

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:

Variable in global space

x

ERASE THE FRAME
Visualizing Frames: The Python Tutor

```python
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)
```

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

Objects
- function max(x, y)
  - 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
6  a = 1
7  b = 2
8  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)

Global Space

Call Frame

Variables from second lecture go in here
Visualizing Frames: The Python Tutor

```python
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)
```

Missing line numbers!
Visualizing Frames: The Python Tutor

Line number marked here (sort-of)

Missing line numbers!
Next Time: Concrete Examples