Lecture 4: Defining Functions (Ch. 3.4-3.11)

CS 1110
Introduction to Computing Using Python

Announcements

• Zoom polls not appearing, and not using browser?
  • "a little icon shows up on the bottom … sometimes you have to click it to see the poll."(Thanks, CS1110 student for the tip!)

From Last Time: Function Calls

• Function calls have the form:
  \[
  \text{best\_function\_ever}(x, y, ...)
  \]
  - Arguments: values given as inputs
    • Separated by commas
    • Can be any expression
  A function might have 0, 1, … or many arguments

Let’s define our own functions!

Anatomy of a Function Definition

Python keyword

def increment(n):
  """Returns: the value of n+1""
  return n+1

function name

function parameters: variables for storing input

function header

function body: statements to execute when called.
Indented relative to function header

Docstring specification

The return Statement

• Passes a value from the function to the caller
• Format: \text{return <expression>}
• Any function body statements placed after a return statement will be ignored
• Optional (if absent, special value None will be sent back)
### Organization of a Module

- Function definition goes before any code that calls that function
- There can be multiple function definitions
- Can organize function definitions in any order

```python
# simple_math.py
def increment(n):
    return n+1
increment(2)
```

### Function Definitions vs. Calls

**Function definition**
- Defines what function will do
- Declaration of parameters (n in this case)
- Parameter: variable where input to function is stored

**Function call**
- Command to do the function
- Argument to assign to function parameter (Argument 2 to be assigned to parameter n in this case)
- Argument: an input value to assign to the function parameter when it is called

```python
# simple_math.py
def increment(n):
    return n+1
increment(2)
```

### Executing the script `simple_math.py`

```bash
C:/> python simple_math.py
```  
```python
def increment(n):
    return n+1
x = increment(2)
```

### Understanding How Functions Work

- We draw pictures to show what is in memory
- **Call Frame**: representation of function call

- Draw parameters as variables (named boxes)
- Line number of the next statement in the function body to execute
- Starts with 1st statement in function body

- function name
- instruction counter
- parameters
- local variables (later in lecture)

**Not just a pretty picture!**
The information in this picture depicts exactly what is stored in memory on your computer.

*Note: slightly different than in the book (3.9) Please do it this way.*

### Example: get_feet in `height.py` module

```python
>>> import height
>>> height.get_feet(68)
```

```python
# height.py
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```
Example: get_feet(68) (slide 1)

```python
>>> import height
>>> height.get_feet(68)
```

**PHASE 1: Set up call frame**

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Indicate next line to execute

```python
# height.py
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```

**Example: get_feet(68) (slide 2)**

```python
>>> import height
>>> height.get_feet(68)
```

**PHASE 2: Execute function body**

- Return statement creates a special variable for result
- The return terminates; no next line to execute

```python
# height.py
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```

**Example: get_feet(68) (slide 3)**

```python
>>> import height
5
>>> import height
```

**PHASE 3: Delete (cross out) call frame**

```python
# height.py
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```

**Local Variables (1)**

Call frames can contain “local” variables

- A variable created in the function

```python
>>> import height2
>>> height2.get_feet(68)
```

```python
# height2.py
def get_feet(ht_in_inches):
    feet = ht_in_inches // 12
    return feet
```

**Local Variables (2)**

Call frames can contain “local” variables

```python
>>> import height2
>>> height2.get_feet(68)
```

```python
# height2.py
def get_feet(ht_in_inches):
    feet = ht_in_inches // 12
    return feet
```

**Local Variables (3)**

Call frames can contain “local” variables

```python
>>> import height2
>>> height2.get_feet(68)
```

```python
# height2.py
def get_feet(ht_in_inches):
    return ht_in_inches // 12
```
Variables are gone! This function is over.

```python
# height2.py
def get_feet(ht_in_inches):
    feet = ht_in_inches // 12
    return feet
```

Call frames can contain “local” variables

```python
>>> import height2
>>> height2.get_feet(68)
5
```

Python interactive mode evaluates the expression and reports

Local Variables (4)

Exercise #1

<table>
<thead>
<tr>
<th>Function Definition</th>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def foo(a,b):</code></td>
<td><code>&gt;&gt;&gt; foo(3,4)</code></td>
</tr>
<tr>
<td>1 ( x = a )</td>
<td></td>
</tr>
<tr>
<td>2 ( y = b )</td>
<td></td>
</tr>
<tr>
<td>3 ( return x*y+y )</td>
<td></td>
</tr>
</tbody>
</table>

Which One is Closest to Your Answer?

A: `foo`  
\[
\begin{array}{c}
\text{a} \\
\text{x}
\end{array}
\begin{array}{c}
\text{b} \\
\text{y}
\end{array}
\]

B: `foo`  
\[
\begin{array}{c}
\text{a} \\
\text{x}
\end{array}
\begin{array}{c}
\text{b} \\
\text{y}
\end{array}
\]

C: `foo`  
\[
\begin{array}{c}
\text{a} \\
\text{x}
\end{array}
\begin{array}{c}
\text{b} \\
\text{y}
\end{array}
\]

D: `foo`  
\[
\begin{array}{c}
\text{a} \\
\text{x}
\end{array}
\begin{array}{c}
\text{b} \\
\text{y}
\end{array}
\]

Exercise #2

<table>
<thead>
<tr>
<th>Function Definition</th>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def foo(a,b):</code></td>
<td><code>&gt;&gt;&gt; foo(3,4)</code></td>
</tr>
<tr>
<td>1 ( x = a )</td>
<td></td>
</tr>
<tr>
<td>2 ( y = b )</td>
<td></td>
</tr>
<tr>
<td>3 ( return x*y+y )</td>
<td></td>
</tr>
</tbody>
</table>

Exercise Time (no poll, just discuss)

<table>
<thead>
<tr>
<th>Function Definition</th>
<th>Function Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def foo(a,b):</code></td>
<td><code>&gt;&gt;&gt; foo(3,4)</code></td>
</tr>
<tr>
<td>1 ( x = a )</td>
<td></td>
</tr>
<tr>
<td>2 ( y = b )</td>
<td></td>
</tr>
<tr>
<td>3 ( return x*y+y )</td>
<td></td>
</tr>
</tbody>
</table>

Which One is Closest to Your Answer?

A: `foo`  
\[
\begin{array}{c}
\text{a} \\
\text{x}
\end{array}
\begin{array}{c}
\text{b} \\
\text{y}
\end{array}
\]

B: `foo`  
\[
\begin{array}{c}
\text{a} \\
\text{x}
\end{array}
\begin{array}{c}
\text{b} \\
\text{y}
\end{array}
\]

C: `foo`  
\[
\begin{array}{c}
\text{a} \\
\text{x}
\end{array}
\begin{array}{c}
\text{b} \\
\text{y}
\end{array}
\]

D: `foo`  
\[
\begin{array}{c}
\text{a} \\
\text{x}
\end{array}
\begin{array}{c}
\text{b} \\
\text{y}
\end{array}
\]
Exercise #3

Function Definition

```python
def foo(a, b):
    x = a
    y = b
    return x*y+y
```

Function Call

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

What is the next step?

Exercise Time (no poll, just discuss)

Function Definition

```python
def foo(a, b):
    x = a
    y = b
    return x*y+y
```

Function Call

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

What is the next step?

Global Space

= the purple box we previously labeled “What Python can access directly”

- Top-most location in memory
- Variables in Global Space called Global Variables
- Functions can access anything global space (see next slides)

```
C:\> python
>>> x = 7
```
Python just started. It has all the built-in functions. It hasn’t read any of the module yet.

```
# height3.py
INCHES_PER_FT = 12

def get_feet(ht_in_inches):
    feet = ht_in_inches // INCHES_PER_FT
    return feet

answer = get_feet(68)
print(answer)
```

C:\> python height3.py

Python just read line 1 of the module. A variable has been added to the Global Space.

```
# height3.py
INCHES_PER_FT = 12

def get_feet(ht_in_inches):
    feet = ht_in_inches // INCHES_PER_FT
    return feet

answer = get_feet(68)
print(answer)
```

Python just read line 2 of the module. A new function has been added to the Global Space. Note: python has not yet looked inside the function.

```
# height3.py
INCHES_PER_FT = 12

def get_feet(ht_in_inches):
    feet = ht_in_inches // INCHES_PER_FT
    return feet

answer = get_feet(68)
print(answer)
```

Python has just executed line 3. A new local variable feet has been created inside get_feet's Call Frame.

```
# height3.py
INCHES_PER_FT = 12

def get_feet(ht_in_inches):
    feet = ht_in_inches // INCHES_PER_FT
    return feet

answer = get_feet(68)
print(answer)
```

Python has just executed line 4. A return value has been created.
Function Access to Global Space (7)

```python
# height3.py
INCHES_PER_FT = 12

def get_feet(ht_in_inches):
    feet = ht_in_inches // INCHES_PER_FT
    return feet

answer = get_feet(68)
print(answer)
```

Python has just executed line 5. A new global variable `answer` has been created. The call frame for `get_feet` has been deleted.

Function Access to Global Space (8)

```python
# height3.py
INCHES_PER_FT = 12

def get_feet(ht_in_inches):
    feet = ht_in_inches // INCHES_PER_FT
    return feet

answer = get_feet(68)
print(answer)
```

C:\> python height3.py
5

Function Access to Global Space (9)

```python
# height3.py
INCHES_PER_FT = 12

def get_feet(ht_in_inches):
    feet = ht_in_inches // INCHES_PER_FT
    return feet

answer = get_feet(68)
print(answer)
```

C:\> python height3.py
5

Q: what about this??

What if a local variable inside a function has the same name as a global variable?

```python
# height5.py

def get_feet(ht_in_inches):
    feet = ht_in_inches // 12
    return feet

C:\> python
>>> feet = "plural of foot"
>>> import height5
>>> height5.get_feet(68)
```

A: Look, but don't touch!

Can't change global variables in a function! Assignment to a global makes a new local variable!

```python
# height5.py

def get_feet(ht_in_inches):
    feet = ht_in_inches // 12
    return feet
```

C:\> python
>>> feet = "plural of foot"
>>> import height5
>>> height5.get_feet(68)

Use Python Tutor to help visualize

Lots of code for today:
https://www.cs.cornell.edu/courses/cs1110/2022sp/schedule/lecture/lec04/lec04.html
Paste it into the Python Tutor (http://cs1110.cs.cornell.edu/tutor/#mode=edit)
- Visualize the code as is
- Change the code
  - Try something new!
  - Insert an error! (misspell `ht_in_inches` or `feet`)
- Visualize again and see what is different
# bad_swap.py

def swap(a,b):
    """Bad attempt at swapping globals a & b""
    tmp = a
    a = b
    b = tmp

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

Question: Does this work?
What exactly gets swapped with function swap?
Paste this into the Python Tutor and see for yourself!

# my_module.py

def foo(x):
    return x+1

x = 1+2
x = 3*x

A: 9
B: 10
C: 1
D: Nothing
E: Error

# silly.py

def foo(a,b):
    x = a
    y = b
    return x*y+y

x = 1+2
x = 3*x

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

# module.py

def foo(x):
    x = 1+2
    x = 3*x
    return x+1

x = foo(0)

A: 9
B: 10
C: 1
D: Nothing
E: Error