Lecture 9

Memory and Call Stacks
## Announcements for Today

### Assignment 1

- We have started grading!
  - Should have your grade tomorrow morning
  - Resubmit until correct
- If you were close…
  - Will get feedback in CMS
  - Fix your assignment
- If you were very wrong…
  - Will be contacted tonight!
  - Will hold one-on-ones Fri

### Reading

- Reread Chapter 3
- No reading for Tuesday

### More Assignments

- A2 due next week (Tues)
- A3 posted this Thursday
  - Due 2 weeks from Fri
  - Before leave for Fall Break
## Announcements for Today

### Assignment 1
- We have started grading!
  - Should have your grade tomorrow morning
  - Resubmit until correct
- If you were close…
  - Will get feedback in CMS
  - Fix your assignment
- If you were very wrong…
  - Will be contacted tonight!
  - Will hold one-on-ones Fri

### Reading
- Reread Chapter 3
- No reading for Tuesday
- A2 due next week (Tues)
- A3 posted this Thursday
  - Due 2 weeks from Fri
  - Before leave for Fall Break

**Complete the Survey on Assignment 1!**
Modeling Storage in Python

• **Global Space**
  - What you “start with”
  - Stores global variables
  - Also *modules & functions!*
  - Lasts until you quit Python

• **Call Frame**
  - Variables in function call
  - Deleted when call done

• **Heap Space**
  - Where “folders” are stored
  - Have to access indirectly

9/26/13
Modeling Storage in Python

• Global Space
  ▪ What you “start with”
  ▪ Stores global variables
  ▪ Also modules & functions!
  ▪ Lasts until you quit Python

• Call Frame
  ▪ Variables in function call
  ▪ Deleted when call done

• Heap Space
  ▪ Where “folders” are stored
  ▪ Have to access indirectly

Global Space
- p
- id2

Call Frame
- incr_x
- q
- id2

Heap Space
- id2
- x
- 2.0
- x
- 3.0

Will cover later in this course
Memory and the Python Tutor

Global Space

Heap Space

Call Frame

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

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

Objects

Frames
Functions and Global Space

- A function definition...
  - Creates a global variable (same name as function)
  - Creates a **folder** for body
  - Puts folder id in variable
- Variable vs. Call

```python
>>> max
<fun max at 0x100498de8>
>>> max(1,2)
2
```

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

9/26/13
Modules and Global Space

- Importing a module:
  - Creates a global variable (same name as module)
  - Puts contents in a folder
    - Module variables
    - Module functions
  - Puts folder id in variable
- *from* keyword dumps contents to global space

```python
import math
```

**Global Space**

<table>
<thead>
<tr>
<th>id5</th>
<th>math</th>
</tr>
</thead>
</table>

**Heap Space**

<table>
<thead>
<tr>
<th>id5</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
</tr>
<tr>
<td>pi</td>
</tr>
<tr>
<td>e</td>
</tr>
</tbody>
</table>

functions
Modules vs Objects

Module

Object

pi
3.141592

id2

id3

math

x
5.0

e
2.718281

id2

y
2.0

module

z
3.0

functions

Point

id2

p

id3

Call Stacks
Modules vs Objects

Module

Object

9/26/13

math

id2

id3

p

Point

id2

module

id2

module

p

id3

math

pi

3.141592

math.pi

x

5.0

p.x

e

2.718281

math.cos(1)

y

2.0

p.clamp(-1,1)

z

3.0

functions

9/26/13
**Modules vs Objects**

### Module
- **math**: id2
  - **pi**: 3.14159
  - **e**: 2.71828
  - **functions**: `math.pi`, `math.cos(1)`

### Object
- **id3**
  - **Point**
    - **p**:
      - **x**: p.x
      - **y**: 2.0
      - **z**: 3.0
    - **functions**: `p.clamp(-1,1)`

---

The period (.) means "go inside of the folder"
Recall: Everything is an Object!

- Including *basic values*
  - int, float, bool, str

- **Example:**
  >>> x = 2.5
  >>> id(x)

- But basics are *immutable*
  - Contents cannot change
  - Distinction between *value* and *identity* is immaterial
  - So we can ignore the folder
When Do We Need to Draw a Folder?

Yes

• Variable holds a
  ▪ function
  ▪ module
  ▪ object
  ▪ (more????)

No

• Variable holds a
  ▪ base type
  ▪ bool, int, float, str

9/26/13  Call Stacks
Review: Call Frames

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

Call: `to_centigrade(50.0)`

What is happening here?

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

Only at the End!
**Text (Section 3.10) vs. Class**

No instruction counter

Variables are not boxes

---

**Definition:**

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

**Call:** `to_centigrade(50.0)`
Aside: What Happens Each Frame Step?

• The instruction counter **always** changes
• The contents only **change** if
  ▪ You add a new variable
  ▪ You change an existing variable
  ▪ You delete a variable
• If a variable refers to a **mutable object**
  ▪ The contents of the folder might change
Frames and Helper Functions

```python
def last_name_first(s):
    """Precondition: s in the form
    <first-name> <last-name>""
    first = first_name(s)
    last = last_name(s)
    return last + ', ' + first

def first_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[0:end]
```

Call: last_name_first('Walker White'):
def last_name_first(s):
    """Precondition: s in the form
    <first-name> <last-name>""
    first = first_name(s)
    last = last_name(s)
    return last + ',' + first

def first_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[0:end]
def last_name_first(s):
    """Precondition: s in the form <first-name> <last-name>""
    first = first_name(s)
    last = last_name(s)
    return last + ',' + first

def first_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[0:end]

Call: last_name_first('Walker White'):
Frames and Helper Functions

```python
def last_name_first(s):
    """Precondition: s in the form <first-name> <last-name>""
    first = first_name(s)
    last = last_name(s)
    return last + ',' + first

def first_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[0:end]
```

**Call: last_name_first('Walker White'):**

```
<table>
<thead>
<tr>
<th>Function</th>
<th>Call Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>last_name_first</td>
<td>1</td>
</tr>
<tr>
<td>s</td>
<td>'Walker White'</td>
</tr>
<tr>
<td>first_name</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>'Walker White'</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>Ready to return</td>
</tr>
</tbody>
</table>
```

9/26/13 Call Stacks
Frames and Helper Functions

```python
def last_name_first(s):
    """Precondition: s in the form <first-name> <last-name>""
    first = first_name(s)
    last = last_name(s)
    return last + ',' + first

def first_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[0:end]
```

Call: last_name_first('Walker White'):

```
last_name_first: 2
  s: 'Walker White'
  first: 'Walker'
```

Call Stacks
def last_name_first(s):
    """Precondition: s in the form <first-name> <last-name>""
    first = first_name(s)
    last = last_name(s)
    return last + '.' + first

def last_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[end+1:]
The Call Stack

• Functions are “stacked”
  ▪ Cannot remove one above w/o removing one below
  ▪ Sometimes draw bottom up (better fits the metaphor)

• Stack represents memory as a “high water mark”
  ▪ Must have enough to keep the entire stack in memory
  ▪ Error if cannot hold stack
The Call Stack

- Functions are “stacked”
  - Cannot remove one above w/o removing one below
  - Sometimes draw bottom up (better fits the metaphor)
- Stack represents memory as a “high water mark”
  - Must have enough to keep the entire stack in memory
  - Error if cannot hold stack
The Call Stack

- Functions are “stacked”
  - Cannot remove one above w/o removing one below
  - Sometimes draw bottom up (better fits the metaphor)
- Stack represents memory as a “high water mark”
  - Must have enough to keep the entire stack in memory
  - Error if cannot hold stack
The Call Stack

- Functions are “stacked”
  - Cannot remove one above w/o removing one below
  - Sometimes draw bottom up (better fits the metaphor)
- Stack represents memory as a “high water mark”
  - Must have enough to keep the entire stack in memory
  - Error if cannot hold stack

Frame 1
Frame 2
Frame 3
Frame 4
Frame 6

Book adds a special “frame” called module. This is WRONG!
Module is global space
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 show_a():
    print a # shows 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
```

Global Space
(for globals.py)

```
<table>
<thead>
<tr>
<th>change_a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
</tbody>
</table>

```
Errors and the Call Stack

```python
# error.py

def function_1(x,y):
    return function_2(x,y)

def function_2(x,y):
    return function_3(x,y)

def function_3(x,y):
    return x/y  # crash here

if __name__ == '__main__':
    print function_1(1,0)
```

9/26/13

Call Stacks
Errors and the Call Stack

# error.py

```python
def function_1(x,y):
    return function_2(x,y)

def function_2(x,y):
    return function_3(x,y)

def function_3(x,y):
    return x/y  # crash here

if __name__ == '__main__':
    print function_1(1,0)
```

Crashes produce the call stack:

Traceback (most recent call last):
File "error.py", line 20, in <module>
print function_1(1,0)
File "error.py", line 8, in function_1
return function_2(x,y)
File "error.py", line 12, in function_2
return function_3(x,y)
File "error.py", line 16, in function_3
return x/y

Make sure you can see line numbers in Komodo. Preferences ➔ Editor
Errors and the Call Stack

Crashes produce the call stack:

Traceback (most recent call last):
File "error.py", line 20, in <module>
    print function_1(1,0)
File "error.py", line 8, in function_1
    return function_2(x,y)
File "error.py", line 12, in function_2
    return function_3(x,y)
File "error.py", line 16, in function_3
    return x/y

Application code. Not a frame!

Where error occurred (or where was found)

Make sure you can see line numbers in Komodo. Preferences ➔ Editor