Lecture 7

Memory in Python
Announcements For This Lecture

Readings
- Reread Chapter 3
- No reading for Thursday

Lab
- Work on Assignment 1
  - Credit when submit A1
  - Nothing else to do

Assignment 1
- Moved to Fri, Sep. 19
  - Worried if I push to Sun., people will start too late
  - Resubmit until Sep. 28
- Posted a Survey in CMS
  - Questions on A1
  - Fill it out when done

9/16/14 Python Memory
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
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

Call Frame

Heap Space

Will cover later in this course
Memory and the Python Tutor

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

a = 1
b = 2
max(a, b)
```
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
def to_centigrade(x):
    return 5*(x-32)/9.0
```

```python
>>> to_centigrade
<fun to_centigrade at 0x100498de8>
>>> to_centigrade (32)
0.0
```
• 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

```
import math
```

**Global Space**

**Heap Space**

- `math` id5
- `pi` 3.141592
- `e` 2.718281
- `functions`
Modules vs Objects

Module

<table>
<thead>
<tr>
<th>math</th>
<th>id2</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi</td>
<td>3.141592</td>
</tr>
<tr>
<td>e</td>
<td>2.718281</td>
</tr>
<tr>
<td></td>
<td>functions</td>
</tr>
</tbody>
</table>

Object

<table>
<thead>
<tr>
<th>id3</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
</tr>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>z</td>
</tr>
</tbody>
</table>

Point
Modules vs Objects

Module

Object

9/16/14 Python Memory

Module

Object

math

p

id2

id3

module

Point

math.pi

math.cos(1)

functions

p.x

p.clamp(-1,1)

id2

id3

math

id2

id3

module

Point

pi

x

3.141592

5.0

e

y

2.718281

2.0

z

3.0

functions

3.141592

p.x

2.718281

p.clamp(-1,1)
## Modules vs Objects

### Module

<table>
<thead>
<tr>
<th>math</th>
<th>id2</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi</td>
<td>3.14159</td>
</tr>
<tr>
<td>e</td>
<td>2.718281</td>
</tr>
<tr>
<td>functions</td>
<td>math.pi, math.cos(1)</td>
</tr>
</tbody>
</table>

### Object

<table>
<thead>
<tr>
<th>p</th>
<th>id3</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>2.0</td>
</tr>
<tr>
<td>z</td>
<td>3.0</td>
</tr>
<tr>
<td>p.x</td>
<td>p.clamp(-1,1)</td>
</tr>
</tbody>
</table>

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

- Including *basic values*
  - `int`, `float`, `bool`, `str`
- **Example:**
  ```python
  >>> 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

9/16/14 Python Memory
Recall: 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

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

**Call:** to_centigrade(50.0)

```
<table>
<thead>
<tr>
<th>frame</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_centigrade</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>50.0</td>
</tr>
</tbody>
</table>
```

What is happening here?

```
9/16/14
Python Memory
```
Recall: 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

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

Call: to_centigrade(50.0)
Recall: 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)

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

But don’t actually erase on an exam
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
Call Frames vs. Global Variables

• This does not work:

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

```plaintext
>>> 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

<table>
<thead>
<tr>
<th>swap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

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

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

```python
global a
a = 1
```

```python
global b
b = 2
```

```python
>>> a = 1
```

```python
>>> b = 2
```

```python
>>> swap(a,b)
```

The call frame and global variables are as follows:

Call Frame:
- `swap`
- `a = 1`
- `b = 2`
- `tmp = 1`

Global Variables:
- `a = 1`
- `b = 2`
The specification is false:

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

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

Global Variables:
- a = 1
- b = 2

Call Frame:
- `swap` function
- `a` = 2
- `b` = 2
- `tmp` = 1
Call Frames vs. Global Variables

- The specification is false:

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

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

Global Variables

- a
- b

Call Frame

- a
- b
- tmp

1
2
3

9/16/14
Call Frames vs. Global Variables

- The specification is false:

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

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

Global Variables

- a
- b

Call Frame

- a
- b

9/16/14 Python Memory
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 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
```
Call Frames and Objects

- Mutable objects can be altered in a function call
  - Object vars hold names!
  - Folder accessed by both global var & parameter
- **Example:**
  
  ```python
  def incr_x(q):
      q.x = q.x + 1
  
  p = Point(0,0,0)
  incr_x(p)
  ```
Call Frames and Objects

- Mutable objects can be altered in a function call
  - Object vars hold names!
  - Folder accessed by both global var & parameter

- Example:

```python
def incr_x(q):
    q.x = q.x + 1

>>> p = Point(0,0,0)
>>> incr_x(p)
```

Global Space

Heap Space

Call Frame
Call Frames and Objects

- Mutable objects can be altered in a function call
  - Object vars hold names!
  - Folder accessed by both global var & parameter

- **Example:**

```python
def incr_x(q):
    q.x = q.x + 1

>>> p = Point(0,0,0)
```

```python
>>> incr_x(p)
```

```
Point(1,0,0)
```