Lecture 10

Memory in Python
### Announcements For This Lecture

#### Reading
- Reread all of Chapter 3

#### Assignments
- Work on your revisions
  - Want done by Sunday
- **Survey**: 487 responded
  - Remaining do by tomorrow
  - **Avg Time**: 7.0 hours
  - **STD Dev**: 4.9 hours
- Assignment 2 also Sunday
  - Scan and submit online
- Assignment 3 up Monday

9/22/16 Memory in Python
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
Memory and the Python Tutor

Global Space

Heap Space

Call Frame

Memory in Python
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
>>> to_centigrade
<fun to_centigrade at 0x100498de8>  
>>> to_centigrade(32)
0.0
```

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

**Heap Space**
- `id5`
  - `module`
  - `pi` 3.141592
  - `e` 2.718281

**Global Space**
- `math`
- `id5`
Memory in Python

Modules vs Objects

**Module**

- math
- module
- pi: 3.141592
- e: 2.718281
- functions

**Object**

- p
- Point3
- id3
  - x: 5.0
  - y: 2.0
  - z: 3.0
Modules vs Objects

Module

<table>
<thead>
<tr>
<th>id2</th>
<th>module</th>
</tr>
</thead>
<tbody>
<tr>
<td>id2</td>
<td></td>
</tr>
<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>
<tr>
<td></td>
<td>math.pi</td>
</tr>
<tr>
<td></td>
<td>math.cos(1)</td>
</tr>
</tbody>
</table>

Object

<table>
<thead>
<tr>
<th>id3</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>id3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>5.0</td>
</tr>
<tr>
<td>y</td>
<td>2.0</td>
</tr>
<tr>
<td>z</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>p.x</td>
</tr>
<tr>
<td></td>
<td>p.clamp(-1,1)</td>
</tr>
</tbody>
</table>
The period (.) means "go inside of the folder"
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)`

What is happening here?

Only at the End!
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)

![Diagram of call frame with variables x=50.0 and return value 10.0]
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
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
```

Global Space
(for globals.py)

```
show_a
```

```
<table>
<thead>
<tr>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>show_a</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
```
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 = Point3(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)
>>> incr_x(p)
```

Global Space

- `p` id5

Heap Space

- `p` id5
- `x` id5
- `0.0` 1.0
- `...`

Call Frame

- `incr_x`
- `q` id5
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)
  >>> incr_x(p)
  ```
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
  1

s      'Walker White'
```

9/22/16 Memory in 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(' ')  # Not done. Do not erase!
    return s[0:end]
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 | 1
--------------  
s             | 'Walker White'
```

```
first_name | 2
-----------  
s             | 'Walker White'
end              | 6
```

9/22/16  Memory in Python
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(' ')  # Add space for clarity
    return s[0:end]
```

Call: last_name_first('Walker White'):

```
last_name_first                   1
s                                   'Walker White'

first_name                        
end                                6
RETURN                             'Walker'
```

9/22/16 Memory in Python
**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'):**

- s: 'Walker White'
- first: 'Walker'

9/22/16

Memory in Python
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 last_name(s):
    """Prec: see last_name_first""
    end = s.rfind(' ')
    return s[end+1:]
```

Call: `last_name_first('Walker White')`

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

last_name
1
s     'Walker White'
```
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

9/22/16

Memory in Python
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 without 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.
def tens(n):
    """Returns: tens-word for n
    Parameter: the integer to anglicize
    Precondition: n in 2..9""
    if n == 2:
        return 'twenty'
    elif n == 3:
        return 'thirty'
    elif n == 4:
        return 'forty'
    elif n == 5:
        return 'fifty'
    elif n == 6:
        return 'sixty'
    elif n == 7:
        return 'seventy'
    elif n == 8:
        return 'eighty'
    return 'ninety'
Anglicize Example

```python
120 def tens(n):
121     '''Returns: tens-word for n
122     Parameter: the integer to anglicize
123     Precondition: n in 2..9'''
124     if n == 2:
125         return 'twenty'
126     elif n == 3:
127         return 'thirty'
128     elif n == 4:
129         return 'forty'
130     elif n == 5:
131         return 'fifty'
132     elif n == 6:
133         return 'sixty'
134     elif n == 7:
135         return 'seventy'
136     elif n == 8:
137         return 'eighty'
138     return 'ninety'
```

Global Space

Call Stack