Lecture 10

Memory in Python
# Announcements For This Lecture

## Reading
- Reread all of Chapter 3

## Assignments
- Finishing Assignment 1
- Assignment 2 now online
  - Has objects, lists, dicts
  - Also if and for-loops
  - Due Sunday, March 12
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
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
  >>> to_centigrade
  <fun to_centigrade at 0x100498de8>
  >>> to_centigrade (32)
  0.0

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

Global Space
```
to_centigrade id6
```

Heap Space
```
to_centigrade (x-32)/9.0
```

9/22/16 Memory in Python 5
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
Modules vs Objects

Module

```
math
id2
```

```
module
id2
```

```
pi  3.141592
```

```
e   2.718281
```

```
functions
```

Object

```
p
id3
```

```
id3
Point3
```

```
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>pi</td>
<td>3.141592</td>
</tr>
<tr>
<td>e</td>
<td>2.718281</td>
</tr>
<tr>
<td></td>
<td>math.pi, math.cos(1)</td>
</tr>
</tbody>
</table>

Object

<table>
<thead>
<tr>
<th>id3</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<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>p</td>
<td>p.x, p.clamp(-1,1)</td>
</tr>
</tbody>
</table>
## Modules vs Objects

<table>
<thead>
<tr>
<th>Module</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>math</code></td>
<td><code>id2</code></td>
</tr>
<tr>
<td><code>id2</code></td>
<td></td>
</tr>
<tr>
<td><code>pi</code></td>
<td><code>y</code></td>
</tr>
<tr>
<td><code>e</code></td>
<td><code>z</code></td>
</tr>
<tr>
<td></td>
<td><code>p</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The period (.) means **“go inside of the folder”**

```
math.pi
math.cos(1)
p.x
p.clamp(-1,1)
```
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

```
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

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

Call: to_centigrade(50.0)

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
```
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)
```

9/22/16 Memory in Python
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)
```

9/22/16 Memory in Python
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
- `id5
  - `Point
  - `x 0 1.0`

Call Frame

9/22/16 Memory in Python
Frames and Helper Functions

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 19
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')
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(' ')
    return s[0:end]
```

Call: last_name_first('Walker White'):

```
<table>
<thead>
<tr>
<th>last_name_first</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>'Walker White'</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>first_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>RETURN</td>
</tr>
</tbody>
</table>
```
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(' ')  # Fixing the missing colon
    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 last_name(s):
    """Prec: see last_name_first""
    end = s.rfind(' ')  
    return s[end+1:]
```

Call: last_name_first('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”
  - Calls must be made from the bottom up
  - Sometimes drawn 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

Book adds a special “frame” called module. This is WRONG! Module is global space.

Frame 1
Frame 2
Frame 3
Frame 4
Frame 6
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'
    else:
        return 'ninety'

Global frame

<table>
<thead>
<tr>
<th>function</th>
<th>anglicize</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>anglicize1000</td>
</tr>
<tr>
<td>function</td>
<td>anglicize1099</td>
</tr>
<tr>
<td>function</td>
<td>anglicize20to99</td>
</tr>
<tr>
<td>function</td>
<td>anglicize100to999</td>
</tr>
</tbody>
</table>

Frames

Objects

<table>
<thead>
<tr>
<th>anglicize</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
</tr>
<tr>
<td>function</td>
</tr>
<tr>
<td>function</td>
</tr>
<tr>
<td>function</td>
</tr>
<tr>
<td>function</td>
</tr>
<tr>
<td>function</td>
</tr>
</tbody>
</table>

n | 234756

n | 756

n | 56

n | 5

9/22/16 Memory in Python
Anglicize Example

```python
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'
```

Global Space

Call Stack