Lecture 10

Memory in Python
Announcements For This Lecture

Assignment 1
- Work on your revisions
  - Read feedback carefully
  - Want done by tomorrow
  - Partial credit after Wed.
- Survey: 501 responded
  - Deadline is tomorrow
  - Avg Time: 7.3 hours
  - STD Dev: 3.6 hours

More Assignments
- Assignment 2 TONIGHT
  - Scan and submit online
  - Upload before midnight
  - Late: -10% per day
  - No lates after Thursday
- Assignment 3 is posted
  - Due week from Friday
  - Before you go on Fall Break
  - Graded when you get back
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

Memory in Python

Global Space

Heap Space

Call Frame
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
```
**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 **

- `math`
- `id5`

** Global Space **

- `pi`: 3.141592
- `e`: 2.718281
- `functions`

9/25/18 Memory in Python
Modules vs Objects

Module

<table>
<thead>
<tr>
<th>math</th>
<th>id2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>module</td>
</tr>
<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>
</tbody>
</table>

Object

<table>
<thead>
<tr>
<th>id3</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>id3</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>
</tbody>
</table>
Modules vs Objects

Module

Object

id2

p

pi

x

3.141592

5.0

e

y

2.718281

2.0

functions

math.pi

math.cos(1)

Point3

z

p.x

3.0

p.clamp(-1,1)

Memory in Python
# Modules vs Objects

<table>
<thead>
<tr>
<th>Module</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
<td>id2</td>
</tr>
<tr>
<td>pi</td>
<td>id2</td>
</tr>
<tr>
<td>e</td>
<td>p</td>
</tr>
<tr>
<td>functions</td>
<td>p.x</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

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

Call: to_centigrade(50.0)

```
to_centigrade
  x  50.0
  RETURN 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)

Memory in Python

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`: 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 `locals.py`)  

<table>
<thead>
<tr>
<th>show_a</th>
<th>1</th>
</tr>
</thead>
</table>

```python
a  4
```
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)

```
a  4

c change_a  
a  3.5
```

9/25/18 Memory in Python 15
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
- `p` id5

Heap Space
- `id5`
- `x` 0.0
- `...`
- `Point3`

Call Frame
- `incr_x`
- `q` id5
- `1`
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 = Point3(0,0,0)
incr_x(p)
```

Global Space

Heap Space

Call Frame
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(' ')  # Call: last_name_first('Walker White'):
    s[0:end]
```

9/25/18  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')

Not done. Do not erase!
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
```
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>Frame</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s</code></td>
<td>'Walker White'</td>
</tr>
<tr>
<td><code>end</code></td>
<td>6</td>
</tr>
<tr>
<td><code>s[0:end]</code></td>
<td>'Walker'</td>
</tr>
</tbody>
</table>

9/25/18  
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'):

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

ERASE WHOLE FRAME

9/25/18

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'):

```

<table>
<thead>
<tr>
<th>last_name_first</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>'Walker White'</td>
</tr>
<tr>
<td>first</td>
<td>'Walker'</td>
</tr>
</tbody>
</table>
```

```

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

9/25/18 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 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

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

```
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