Memory in Python
Announcements For This Lecture

**Reading**

- Reread all of Chapter 3

**Assignments**

- Work on your revisions
  - Want done by Sunday
- **Survey**: 501 responded
  - Remaining do by tomorrow
  - **Avg Time**: 7.3 hours
  - **STD Dev**: 3.6 hours
- Assignment 2 also Sunday
  - Scan and submit online
- Assignment 3 up Monday
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
- Call Frame
- Heap Space

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

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

9/21/17  Memory in Python
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**

- `math`
- `id5`

**Heap Space**

- `id5`
  - `module`
- `pi`: 3.141592
- `e`: 2.718281

functions
Modules vs Objects

### Module
- `math`
- `id2`
  - `module`
  - `pi`: 3.141592
  - `e`: 2.718281
  - `functions`

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

Module

```
math
```

id2

```
module

pi
3.141592

e
2.718281

functions

math.pi
math.cos(1)
```

Object

```
Point3

id3

p
```

```

x
5.0

y
2.0

z
3.0

p.x
p.clamp(-1,1)
```
**Modules vs Objects**

### Module vs Object

<table>
<thead>
<tr>
<th>Module</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
<td>id2</td>
</tr>
<tr>
<td>id2</td>
<td>id3</td>
</tr>
<tr>
<td>pi</td>
<td>p</td>
</tr>
<tr>
<td>e</td>
<td>y</td>
</tr>
<tr>
<td></td>
<td>z</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

```
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)`
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)

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

```
# 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
```

```python
# globals.py
# global space
a = 4

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)
```

9/21/17 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 = Point3(0,0,0)
>>> incr_x(p)
```

9/21/17 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 = Point3(0,0,0)
>>> incr_x(p)
```

Global Space

Heap Space

Call Frame

ERASE 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(' ')
    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(' ')  # Call: last_name_first
    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')`:
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
    s                     'Walker White'
    end                  6
    return               'Walker'
```
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>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>'Walker White'</td>
</tr>
<tr>
<td>first</td>
<td>'Walker'</td>
</tr>
</tbody>
</table>

Call: first_name('Walker White'):

<table>
<thead>
<tr>
<th>first_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Walker'</td>
</tr>
</tbody>
</table>

9/21/17 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'):
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
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

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

Frames

- Global frame
  - function anglicize
  - anglicize1000
  - anglicize1000(n)
  - anglicize100to999
  - tens
  - anglicize
  - n 234756
  - anglicize1000
  - n 756
  - anglicize100to999
  - n 756
  - hundreds 56
  - suffix ""
  - anglicize20to99
  - n 56

Objects

- tens
- n 5
```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'
```