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**: 660 responded
  - Deadline is tomorrow
  - **Avg Time**: 6.8 hours!
  - **STD Dev**: 3.9 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

---

9/25/18

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
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
```

9/25/18 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

```
import math
```

Heap Space

```
math  id5
```

Global Space

```
pi    3.141592
```

```
e     2.718281
```

functions
Modules vs Objects

Module

math

id2

module

id2

pi

3.141592

e

2.718281

functions

Object

p

id3

Point3

id3

x

5.0

y

2.0

z

3.0

9/25/18

Memory in Python
Modules vs Objects

Module

Math

Module

Object

Id2

Point3

Id3

Math.pi

Math.cos(1)

P

P.x

P.clamp(-1,1)

Id3

X 5.0

Y 2.0

Z 3.0

Pi 3.141592

E 2.718281

9/25/18 Memory in Python
The period (.) means “go inside of the folder”

Functions:
- `math.pi`
- `math.cos(1)`

Module:
- `id2`
- `math`
- `pi`
- `e`
- `functions`

Object:
- `id3`
- `p`
- `id2`
- `Point3`
- `p.x`
- `p.clamp(-1,1)`
- `y 2.0`
- `z 3.0`
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

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

\[ \text{Call: to\_centigrade}(50.0) \]

\[
\text{def to\_centigrade}(x):
    \begin{align*}
    1 & \quad \text{return } 5*(x-32)/9.0
    \end{align*}
\]

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)
```

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)
```

9/25/18

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

- `p` id5

Heap Space

- `Point3`
- `id5`
- `x` 0 1.0

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

Call: `last_name_first('Walker White')`:

- `last_name_first`: 1
- `s`: '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')

Not done. Do not erase!
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>
<tr>
<td>first_name</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>'Walker White'</td>
</tr>
<tr>
<td>end</td>
<td>6</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
1
s
'Walker White'
```

```
first_name
1
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'):

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

ERASE WHOLE 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 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'
```

9/25/18 Memory in Python 24
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.

Frame 1 calls Frame 2
Frame 2 calls Frame 3
Frame 3 calls Frame 4
Frame 4 calls Frame 6

Memory in 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'
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

Anglicize Example

Global Space

Call Stack