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
  <function to_centigrade at 0x100498de8>
  >>> to_centigrade(32)
  0.0

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(60.0)

Memory and the Python Tutor

- 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

Function Access to Global Space

- Global Space (for globals.py)
  - Only at the End!
- Change a
  - # global for math
  - temperature.to_centigrade uses global for temperature
  - But cannot change values
  - # global for math
  - s = 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 = Point(0,0,0)
>>> 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):
    """Precondition: s in the form <first-name> <last-name>""
    end = s.find(' ')
    return s[0:end]
```

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