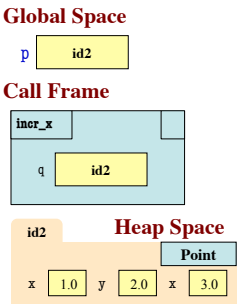
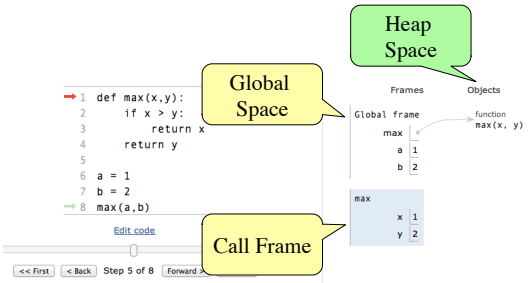
Modeling Storage in Python

- **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

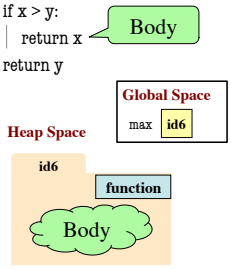


Functions and Global Space

- A function definition…
  - Creates a global variable (same name as function)
  - Creates a **folder** for body
  - Puts folder id in variable
- Variable vs. Call

```
>>> max
<fun max at 0x100498de8>
>>> max(1,2)
2
```

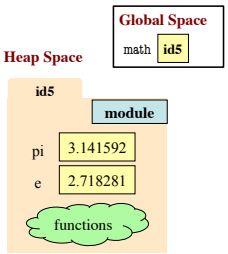
```
def max(x,y):
    if x > y:
        return x
    return y
```



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
```



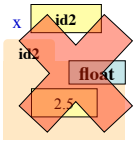
When Do We Need to Draw a Folder?

Yes

- Variable holds a
  - function
  - module
  - object
  - (more????)

No

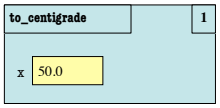
- Variable holds a
  - base type
  - bool, int, float, str



Review: 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)



What is happening here?

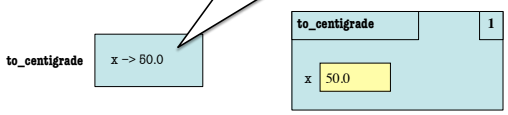
Only at the End!

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

Text (Section 3.10) vs. Class

Textbook | Class

No instruction counter
Variables are not boxes



Definition:

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

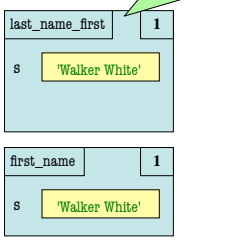
Call: to_centigrade(50.0)

Frames and Helper Functions

```
def last_name_first(s):
    """Precondition: s in the form
    <first-name> <last-name>"""
1   first = first_name(s)
2   last = last_name(s)
3   return last + '.' + first

def first_name(s):
    """Prec: see last_name_first"""
1   end = s.find(' ')
2   return s[0:end]
```

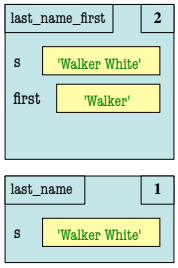
Not done. Do not erase!



Frames and Helper Functions

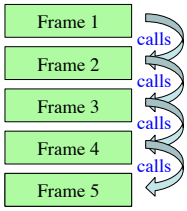
```
def last_name_first(s):
    """Precondition: s in the form
    <first-name> <last-name>"""
1   first = first_name(s)
2   last = last_name(s)
3   return last + ',' + first

def last_name(s):
    """Prec: see last_name_first"""
1   end = s.find(' ')
2   return s[end+1:]
```



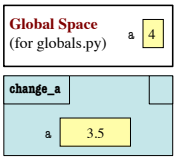
The Call Stack

- 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



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 change_a():
    a = 3.5 # local variable
```

Errors and the Call Stack

```
# error.py

def function_1(x,y):
    return function_2(x,y)

def function_2(x,y):
    return function_3(x,y)

def function_3(x,y):
    return x/y # crash here

if __name__ == '__main__':
    print function_1(1,0)
```

Crashes produce the call stack:

```
Traceback (most recent call last):
  File "error.py", line 20, in <module>
    print function_1(1,0)
  File "error.py", line 8, in function_1
    return function_2(x,y)
  File "error.py", line 12, in function_2
    return function_3(x,y)
  File "error.py", line 16, in function_3
    return x/y
```

Make sure you can see line numbers in Komodo. Preferences ➔ Editor