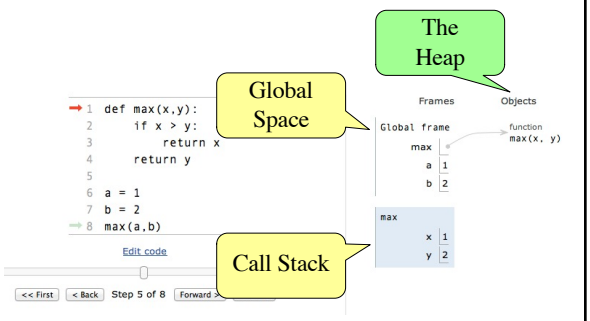
The Three "Areas" of Memory



1

Global Space

- This is the **area you "start with"**
  - First memory area you learned to visualize
  - A place to store "global variables"
  - Lasts until you quit Python

p | id2

- What are **global variables**?
  - **Any assignment not in a function definition**
  - Also **modules & functions!**
  - Will see more on this in a bit

2

The Call Stack

- The area **where call frames live**
  - Call frames are created on a function call
  - May be several frames (functions call functions)
  - Each frame deleted as the call completes
- Area of volatile, temporary memory
  - Less permanent than global space
  - Think of as "scratch" space
- Primary focus of Assignment 2

incr_x	2
q	id2

3

Heap Space or "The Heap"

- **Where the "folders" live**
  - Stores *only* folders
- Can only **access indirectly**
  - Must have a variable with identifier
  - Can be in global space, call stack
- MUST have **variable with id**
  - If no variable has id, it is *forgotten*
  - Disappears in Tutor immediately
  - But not necessarily in practice
  - Role of the *garbage collector*

id2 → Point3

x	0.0
y	0.0
z	0.0

4

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
```

Global Space: math | id5

Heap Space

id5 → module

pi	3.141592
e	2.718281

functions

5

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
```

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

Body

Global Space: to_centigrade | id6

Heap Space

id6 → function

Body

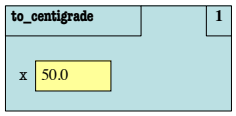
6

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)

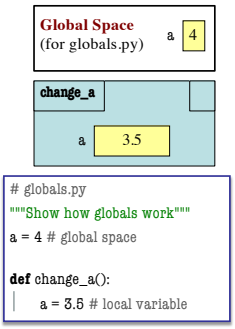


What is happening here?

7

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
  - Makes a *new local variable*!
  - Why we limit to constants

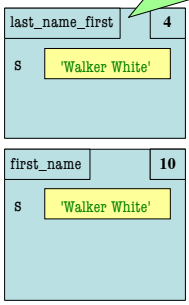


8

Frames and Helper Functions

```
1.  def last_name_first(s):
2.      """Precond: s in the form
3.      'first-name last-name' """
4.      first = first_name(s)
5.      last = last_name(s)
6.      return last + ',' + first
7.
8.  def first_name(s):
9.      """Precond: see above"""
10.     end = s.find(' ')
11.     return s[0:end]
```

Not done. Do not erase!

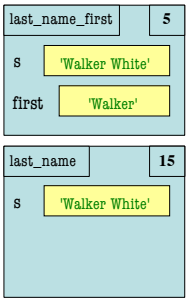


9

Frames and Helper Functions

```
1.  def last_name_first(s):
2.      """Precond: s in the form
3.      'first-name last-name' """
4.      first = first_name(s)
5.      last = last_name(s)
6.      return last + ',' + first
        . . .
13. def last_name(s):
14.     """Precond: see above"""
15.     end = s.rfind(' ')
16.     return s[end+1:]
```

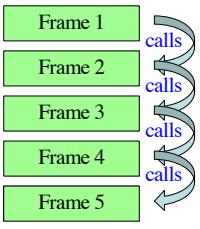
Call: last_name_first('Walker White'):



10

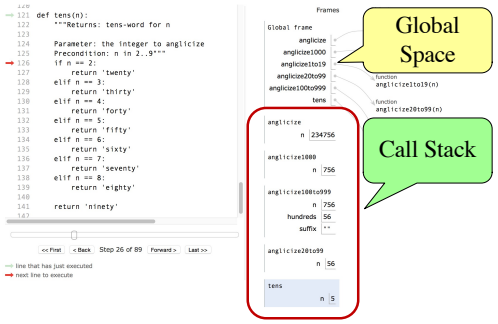
The Call Stack

- 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



11

Anglicize Example



12