

### The Three "Areas" of Memory

```

1 def max(x,y):
2     if x > y:
3         return x
4     return y
5
6 a = 1
7 b = 2
8 max(a,b)
    
```

The diagram illustrates the memory layout. On the left, a code editor shows the execution of a `max` function. On the right, a memory diagram shows the **Global Space** containing a `Global frame` with variables `a` (1) and `b` (2), and a `function max(x, y)` object. Below it, the **Call Stack** contains a `max` frame with local variables `x` (1) and `y` (2). To the right, **The Heap** is shown as a separate area.

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
- 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

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*

4

### Modules and Global Space

- Importing a module: `import math`
  - 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

5

### Functions and Global Space

- A function **definition...** `def to_centiGrade(x):`
  - 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
            
```

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

**Call:** to\_centigrade(50.0)

to_centigrade	1
x	50.0

What is happening here?

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

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

Global Space (for globals.py)	a 4
----------------------------------	-----

change_a	a 3.5
----------	-------

```
# globals.py
"""Show how globals work"""
a = 4 # global space

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

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]`

**Call:** last\_name\_first('Walker White')

last_name_first	4
s	'Walker White'

first_name	10
s	'Walker White'

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')

last_name_first	5
s	'Walker White'
first	'Walker'

last_name	15
s	'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

Frame 1
Frame 2
Frame 3
Frame 4
Frame 5

}

}

}

}

}

11

### Anglicize Example

```
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'
    elif n == 9:
        return 'ninety'
```

Global frame	n 5
anglicize000	n 234756
anglicize009	n 756
anglicize019	n 756
anglicize020999	hundreds 56
tens	a100
anglicize030999	n 56
anglicize040999	n 56
tens	n 5

12