

### 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
 

```
>>> to_centigrade
<fun to_centigrade at 0x100498de8>
>>> to_centigrade(32)
0.0
```

### Modules vs Objects

#### Module

math **id2**

**id2**

pi 3.141592

e 2.718281

functions

math.pi

math.cos(1)

#### Object

p **id3**

**id3**

x 5.0

y 2.0

z 3.0

Point

p.x

p.clamp(-1,1)

### Recall: Call Frames

- Draw a frame for the call
- Assign the argument value to the parameter (in frame)
- Execute the function body
  - Look for variables in the frame
  - If not there, look for global variables with that name
- Erased the frame for the call

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

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

### 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

### 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:**

```
def incr_x(q):
    q.x = q.x + 1
1 |
>>> p = Point(0,0,0)
>>> incr_x(p)
```

**Global Space**

p id5

**Heap Space**

id5

Point

x 0.0

...

**Call Frame**

incr\_x 1

q id5

### 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]
```

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

last\_name\_first 1

s 'Walker White'

first\_name 1

s 'Walker White'

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

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

last\_name\_first 2

s 'Walker White'

first 'Walker'

first\_name 1

s 'Walker White'

ERASE WHOLE FRAME

### 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.rfind(' ')
    2 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'

### 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

calls

calls

calls

calls

calls

### 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'
    return ''

def anglicize(n):
    """Returns: the string representing the integer n in English
    Parameter: the integer to anglicize
    Precondition: n in 0..999999999"""
    if n == 0:
        return ''
    elif n < 10:
        return '' + tens(n)
    elif n < 100:
        return tens(n // 10) + ' ' + tens(n % 10)
    elif n < 1000:
        return tens(n // 1000) + ' hundred' + anglicize(n % 1000)
    else:
        return tens(n // 1000000) + ' million' + anglicize(n % 1000000)
```

110 def tens(n):

111 """Returns: tens-word for n

112 Parameter: the integer to anglicize

113 Precondition: n in 2..9"""

114 if n == 2:

115 return 'twenty'

116 elif n == 3:

117 return 'thirty'

118 elif n == 4:

119 return 'forty'

120 elif n == 5:

121 return 'fifty'

122 elif n == 6:

123 return 'sixty'

124 elif n == 7:

125 return 'seventy'

126 elif n == 8:

127 return 'eighty'

128 elif n == 9:

129 return 'ninety'

130 return ''

**Global Space**

anglicize

anglicize100

anglicize1019

anglicize20099

anglicize1000999

...

**Call Stack**

anglicize n: 234756

anglicize1000 n: 756

anglicize1000999 n: 756

hundreds 56

units ...

anglicize20099 n: 56

tens n: 5