Call Frames; Diagramming Objects
The Big Issue

• Cannot answer questions on this topic unless you
  ▪ draw variables
  ▪ draw frames for function calls
  ▪ draw objects when they are created

• Learning to do this is useful in general
  ▪ Helps you “think like a computer”
  ▪ Easier to find errors in your programs.
What Do You Need to Know?

• Major topics
  - local variables (in a function body)
  - function call (call frames, call stack)
  - constructor call (in addition to call frames)

• Examples from previous exams
  - Question 5 on prelim 1
  - Questions 3 and 5b on prelim 2
Important

• Code execution is an important part of the final
• You need to know how to
  ▪ draw variables
  ▪ draw call frames
  ▪ draw objects

The purpose of such questions on executing statements with constructs and function calls is to test your understanding of how Python programs are executed
The Frame (box) for a Function Call

- **Function Frame**: Representation of function call
- A **conceptual model** of Python

```
function name
parameters
local variables (when assigned)
```

- **Number of statement in the function body to execute** next
- **Starts with 1**

Draw parameters as variables (named boxes)
To Execute the Method: x.addScore(100)

1. Draw a frame for the call
2. Assign the arguments to the parameters (in frame)
3. Execute the method body
   - Look for variables in frame
   - If an attribute, follow the name into Heap Space
4. Erase the frame

```python
class Score(object):
    _score = 0

    def addScore(self, value):
        self._score = self._score + value
```

12/2/12  Review 1
To Execute the Method: \texttt{x.addScore(100)}

1. Draw a frame for the call
2. Assign the arguments to the parameters (in frame)
3. Execute the method body
   - Look for variables in frame
   - If an attribute, follow the name into Heap Space
4. Erase the frame

\begin{verbatim}
class Score(object):
    _score = 0
...
    def addScore(self, value):
        self._score = self._score + value
\end{verbatim}
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 last_name(s):
    """Prec: see last_name_first""
    end = s.find(' ')
    return s[end+1:]
```
def reverse(b):
    """Reverse elements of b in place
    (does not make a copy)
    Pre: b is a list"
    reverse_part(b,0,len(b)-1)

def reverse_part(b,h,k):
    """Reverse b[h..k] in place
    Pre: b is a list; h, k are in b"
    if h >= k:
        return
    temp = b[h]
    b[h] = b[k]
    b[k] = temp
    reverse_part(b,h+1,k-1)
```

- Execute the call
  - a = [5,7,3]; reverse(a)
  - Use ‘folder’ for list a below
  - Stop at (before) line 2
  - **Draw call frame at that time!**

**a**

<table>
<thead>
<tr>
<th>23457811</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0]</td>
</tr>
<tr>
<td>a[1]</td>
</tr>
<tr>
<td>a[2]</td>
</tr>
</tbody>
</table>

Give only one frame per call

Give the state of the frame at last line before call terminates
Execute the Call `reverse([5,7,3])` to Line 2

```python
def reverse(b):
    """Reverse elements of b in place (does not make a copy)"
    Pre: b is a list"
    reverse_part(b,0,len(b)-1)

def reverse_part(b,h,k):
    """Reverse b[h..k] in place"
    Pre: b is a list; h, k are in b"
    if h >= k:
        return
    temp = b[h]
    b[h] = b[k]
    b[k] = temp
    reverse_part(b,h+1,k-1)
```

```
23457811
a[0] 3
a[1] 7
a[2] 5
```

12/2/12
Review 1
Diagramming Objects (Folders)

Folder Name
(make it up)

4300517584

classname

Attributes:
• Unhidden fields
• Properties with a getter

Methods:
• Name+parameter names
• But do not include self

Draw attributes as named box w/ value (even for properties)

Include operators here (but only if defined)
class Time(object):
    hr = 0      # Hour of day
    _min = 0    # Minute of hour

@property  # getter only
def min(self):
    return self._min

def __init__(self, h, m=0):
    """Constructor: new time h:m""
    self._hr = h; self._min = m

def __str__(self):
    """Returns string '<hr>:<min>' ""
    return `self._hr` + ':' + `self._min`

12/2/12 Review 1
Evaluation of a Constructor Call

3 steps to evaluating the call C(args)

• **Create a new folder** (object) of class C
  ▪ Give it with a unique name (any number will do)
  ▪ Folder goes into heap space

• Execute the *method* `__init__`(args)

• Yield *the name* of the object as *the value*
  ▪ A constructor call is an *expression*, not a command
  ▪ Does not put name in a variable unless you *assign it*
Code Segment (with Constructors)

\[ a = 3 \]
\[ x = C(a) \# C \text{ is a class} \]
\[ y = C(a) \]
\[ x = y \]

First thing to do? draw all of the local variables
class C(object):
    f = 0
    def __init__(self, k):
        self.f = k

a = 3
x = C(a) # C a class
y = C(a)
x = y
class C(object):
    f = 0
    def __init__(self, k):
        self.f = k

a = 3
x = C(a)  # C a class
y = C(a)
x = y

aliasing
Execute the call:

```python
def session()
    one = Item('ipod', 20)
    two = Item('wii', 32)
    treat = two
    three = one
    three.add(4)
    print one
    print 'Cost of item one: ' + str(one.cost)
    print ('Are they the same? ' + str(one.name== two.name))
    print ('Are they the same? ' + str(one.name== treat.name))
    print ('Are they the same? ' + str(one.name== three.name))
```
Execute the call: session()

def session()
    one = Item('ipod', 20)
    two = Item('wii', 32)
    treat = two
    three = one
    three.add(4)
    print one
    print 'Cost of item one: ' + str(one.cost)
    print ('Are they the same? ' +
            str(one.name== two.name))
    print ('Are they the same? ' +
            str(one.name== treat.name))
    print ('Are they the same? ' +
            str(one.name== three.name))
Execute the call:

```python
def session():
    one = Item('ipod', 20)
    two = Item('wii', 32)
    treat = two
    three = one
    three.add(4)
    print(one)
    print('Cost of item one: ' + str(one.cost))
    print('Are they the same? ' + str(one.name == two.name))
    print('Are they the same? ' + str(one.name == treat.name))
    print('Are they the same? ' + str(one.name == three.name))
```

Output:

6 : 'ipod:24'
7 : 'Cost of item one: 24'
8 : 'Are they the same? False'
9 : 'Are they the same? False'
10 : 'Are they the same? True'