Review 1

Call Frames;
Diagramming Objects
The Big Issue

• Cannot answer questions on this topic without
  ▪ drawing variables
  ▪ drawing frames for function calls
  ▪ drawing 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 3 on prelim 1
  ▪ Question 5 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

- Draw parameters as variables (named boxes)
- Number of statement in the function body to execute **next**
- Starts with 1
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):
    ...
    def addScore(self, value):
        """Add value to score attr""
        self._score = self._score + value
```
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):
    ...
    def addScore(self, value):
        """Add value to score attr"""
        self._score = self._score + value
```
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:]  

Call Stacks: Given a Line to Reach

Execute to here
(Modified) Question from Previous Years

```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)
```

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

```
  a
  id2
  list
```

- Give only one frame per call
- Give the state of the frame at last line before call terminates
Execute the Call \texttt{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)
```

Diagramming Objects (Folders)

**Object Folder**

- **Folder Name** (make it up)
- **id4**
- **classname**
- **Instance Attributes**
- Draw attributes as named box w/ value

**Class Folder**

- **classname**
- No folder name
- **Class Attributes**
- **Method Names**
- Methods must contain parameters
class Time(object):
    """Instance attributes:
    _hr: hour of day [int, 0..23]
    _min: minute of hour [int, 0..59]""

def getMin(self):
    """Return: minute of hour""
    return self._min

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

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

Time

_hr 2
_min 30
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 id (any number will do)
  - Folder goes into the heap

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

• Return *the identifier* 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
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 Definition:
```python
class C(object):
    f = 0
    def __init__(self, k):
        self.f = k
```

Code:
```python
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
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
Code Segment (with Constructors)

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

old reference is lost
 Execute the call: session() 

```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.getCost())
    print ('Are they the same? ' + str(one.getName()==two.getName()))
    print ('Are they the same? ' + str(one.getName()==treat.getName()))
    print ('Are they the same? ' + str(one.getName()==three.getName()))
```
Execute the call: session()

```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.getCost())
    print ('Are they the same? ' +
    str(one.getName()==two.getName())
    print ('Are they the same? ' +
    str(one.getName()==treat.getName())
    print ('Are they the same? ' +
    str(one.getName()==three.getName()))
```
Execute the call: `session()`

```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.getCost())
    print ('Are they the same? ' + str(one.getName()==two.getName()))
    print ('Are they the same? ' + str(one.getName()==treat.getName()))
    print ('Are they the same? ' + str(one.getName()==three.getName()))
```

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'
Example from Fall 2019(!)

```python
class Cornellian(object):
    # Instance attributes:
    # _cuid: Cornell id, an int > 0
    # _name: full name, a nonempty str

    NEXT = 1 # Class Attribute

    def getName(self):
        """Returns the name."""
        return self._name

    ... 

    def __init__(self, n):
        """Initializes Cornellian with name n."""
        self._name = n
        self._cuid = Cornellian.NEXT
        Cornellian.NEXT = Cornellian.NEXT + 1

    ...
```

Execute:

```python
>>> a = Cornellian('Alice')
>>> b = Cornellian('Bob')
```

Pay close attention to class attribute NEXT
Example from Fall 2019(!)

class Cornellian(object):
    # Instance attributes:
    # _cuid: Cornell id, an int > 0
    # _name: full name, a nonempty str

    NEXT = 1 # Class Attribute

def getName(self):
    """Returns the name."""
    return self._name

...  
def __init__(self, n):
    """Initializes Cornellian with name n."""
    self._name = n
    self._cuid = Cornellian.NEXT
    Cornellian.NEXT = Cornellian.NEXT+1

...
Example from Fall 2019(!)

Execute:

```python
>>> a = Cornellian('Alice')
>>> b = Cornellian('Bob')
```

```
  Cornellian
  __init__(self,n)
  getName(self)
  _cuid  1
  _name  'Alice'
```
Example from Fall 2019(!)

Execute:

```python
>>> a = Cornellian('Alice')
>>> b = Cornellian('Bob')
```

```
Cornellian
 getName(self)
__init__(self,n)
```

NEXT

Cornellian

```python
>>> a = Cornellian('Alice')
>>> b = Cornellian('Bob')
```