Lecture 14: Midterm Review

July 14th, 2021

Alex Kassil

Announcements

- HW 03 Due Wednesday 7/14
- Midterm Exam (Thursday 7/15 @ 5-7PM)
- Cats Project Due Tuesday 7/20 (turn in by Monday 7/19 for extra credit)
- Lab 06 Midterm Review Problems
- Disc 06 Midterm Review Problems
- Tutorial 06 Midterm Review Problems
- Tomorrow chill Q&A during lecture timeslot before the midterm
- Make sure your code.cs61a.org is running version 2.6.3, refresh a few times if it isn’t. If it still doesn’t update, post to Ed.
- My office hours tomorrow extended by 1 hour (2-4pm PT now) to chill and chat before the exam about whatever you want!

What the heck are trees and how do we solve problems with trees?

General tips

- Think about the input, and the output
- Understand
    - that is_leaf() takes a tree
    - a tree with no branches is a leaf
    - branches(t) returns a ***list*** of branches, we can index into branches(t), iterate over branches(t)
        - If a function takes in a tree, passing into it branches(t) will be wrong
- Think it terms of base case and recursive calls
- Assume the recursive calls work
- To do work on whole tree, need to loop over branches for recursive call!
- When creating a tree, we need to have new label and new branches ready before we use the tree constructor

Aggregation

1. Basic examples: max_tree, sum_tree, num_nodes, num_leaves
2. Advanced: odd_row_sum, even_row_sum
    a. Implement `odd_row_sum(t)` and `even_row_sum(t)`, which both take in a tree `t`, and return the sum of the labels on the odd rows of the tree, and the even row of the tree, respectively

```
t = tree(1, [tree(2, [tree(3), tree(4)]), tree(5, [tree(6,
[tree(7, [tree(8)])])])])

print_tree(t)
1
  2
    3
    4
  5
    6
      7
        8
odd_row_sum(t)  == 22 # 1 + 3+4+6 + 8
even_row_sum(t) == 14 # 2+5 + 7
```

Sp20 Final Q7 Expression Tree

Definition: A tree expression for a tree `t` is a string that starts with `t` and contains a Python expression that evaluates to a node label within `t` by using `branches` and `label`.

```
def labels(t):
    """List all tree expressions for tree t.
    >>> t = tree(3, [tree(4, [tree(-1)]), tree(-5)])
    >>> for e in labels(t):
    ...     print(e)
    label(t)
    label(branches(t)[0])
    label(branches(branches(t)[0])[0])
    label(branches(t)[1])
    """
    def traverse(t, e):
        result.append(___________)
        for ___________:
            traverse(branches(t)[i], ___________)
    result = []
    traverse(t, 't')
    return result
```

Booleans

1. Basic: is_tree, is_even_tree # checks if tree has at least one even label
2. Advanced: is_binary_tree
    a. Implement is_binary_tree(t), which takes in a tree t and returns if each node has exactly 2 branches

Fa20 Midterm 2 Q4 Fork It

https://cs61a.org/exam/fa20/mt2/61a-fa20-mt2.pdf

Creating trees

1. Basic: factorial_tree
2. Advanced: factor_tree
   a. Implement a function `factor_tree(n)` which creates a tree factors for a given positive number input `n`

Su19 Final Q4 Combo Nation

https://cs61a.org/exam/su19/final/61a-su19-final.pdfYou may assume the two trees have the same shape (that is, each node has the same number of children).

```
def apply_tree(fn_tree, val_tree):
    """ Creates a new tree by applying each function stored in fn_tree
    to the corresponding labels in val_tree
    >>> double = lambda x: x*2
    >>> square = lambda x: x**2
    >>> identity = lambda x: x
    >>> t1 = tree(double, [tree(square), tree(identity)])
    >>> t2 = tree(6, [tree(2), tree(10)])
    >>> t3 = apply_tree(t1, t2)
    >>> print_tree(t3)
    12
      4
      10
    """

    __________
    __________
    for __________:
        __________
    return __________
```

Su19 Final Q4 Combo Nation

Definition. A combo of a non-negative integer n is the result of adding or multiplying the digits of n from left to right, starting with 0. For n = 357, combos include $15 = (((0 + 3) + 5) + 7)$, $35 = (((0 \ast 3) + 5) \ast 7)$, and $0 = (((0 \ast 3) \ast 5) \ast 7)$, as well as 0, 7, 12, 22, 56, and 105. But $36 = ((0 + 3) \ast (5 + 7))$ is not a combo of 357.

```
def is_combo(n, k):
    """ Is k a combo of n? A combo of a non-negative integer n
    is the result of adding or multiplying the digits of n
    from left to right, starting with 0
    >>> [k for k in range(1000) if is_combo(357, k)]
    [0, 7, 12, 15, 22, 35, 56, 105]
    """
    assert n >= 0 and k >= 0
    if __________:
        return True
    if __________:
        return False
    rest, last = n // 10, n % 10
    added = __________ and is_combo(__________, __________)
    multiplied = __________ and is_combo(__________,__________)
    return added or multiplied
```

Su19 Final Q4 Combo Nation

Implement `make_checker_tree` which takes in a tree, t containing digits as its labels and returns a tree with functions as labels (a function tree). When applied to another tree, the function tree should return a new tree with label as True if the label is a combo of the number formed by concatenating the labels from the root to the corresponding node of t. You may use is_combo in your solution.

```
def make_checker_tree(t, so_far=0):
    """ Returns a function tree that, when applied to another tree,
    will create a new tree where labels are True if the label is a
combination
    of the path in t from the root to its corresponding node.
    >>> t1 = tree(5, [tree(2), tree(1)])
    >>> fn_tree = make_checker_tree(t1)
    >>> t2 = tree(5, [tree(10), tree(7)])
    >>> t3 = apply_tree(fn_tree, t2) #5 is a combo of 5, 10 is a combo of 52,
7 isn't a combo of 51
    >>> print_tree(t3)
    True
      True
      False
    """
    new_path = ________________________________________________________
    branches = ________________________________________________________
    fn = ______________________________________________________________
    return tree(fn, branches)
```

Reverse environment diagrams are NOT my friend

General tips

- **Use python tutor!!!!**
    - Fill in the blanks with random/default/generic values.
    - Run through the code line by line to construct your own environment diagram.
    - If your environment diagram doesn't match the image, go back and try to fix each blank one by one.
- Understand which methods mutate a list and which create a copy
- Each line of the environment diagram is a clue!
    - Names show what variables you should have
    - Values show what the expressions should evaluate to eventually
    - Frame names show which function is called
    - Frame numbers show order of program flow

Fa20 Final Q1
The Droids You’re Looking For

https://cs61a.org/exam/fa20/final/61a-fa20-final.pdf#page=3

check out the midterm review session on reverse ED!

http://links.cs61a.org/midterm-review-sessions

Fa20 Midterm 2 Q1 Political Environment

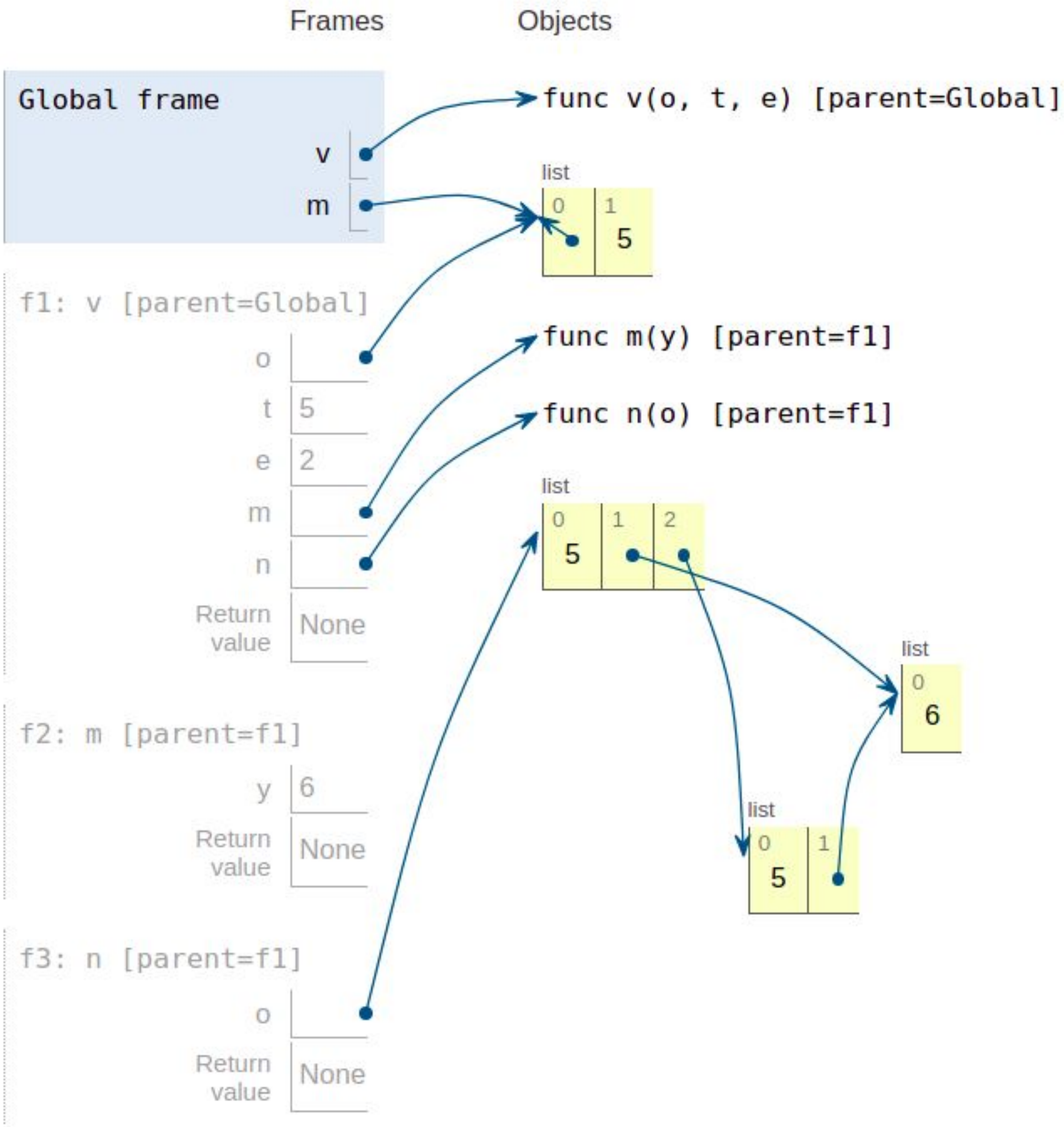
https://cs61a.org/exam/fa20/mt2/61a-fa20-mt2.pdf#page=3 (modified to remove nonlocal)

You may not write any numbers or arithmetic operators (+, -, *, /, //, **) in your solution.

```
def v(o, t, e):
    def m(y):
        _________ #(a)
    def n(o):
        o.append(_____)#(b)
        o.append(_____)#(c)
    m(e)
    n([t])
    e = 2
m = [3, 4]
v(m, 5, 6)
```

Blank (c) choose all that apply

```
o
[o]
list(o)
list([o])
o + []
[o[0], o[1]]
o[:]
```



You may not write any numbers or arithmetic operators (+, -, *, /, //, **) in your solution.

```
def v(o, t, e):
    def m(y):
       o[:] = [o, t] #(a)
    def n(o):
        o.append([e])#(b)
        o.append(o[:])#(c)
    m(e)
    n([t])
    e = 2
m = [3, 4]
v(m, 5, 6)
```

Blank (c) choose all that apply

`o`

`[o]`

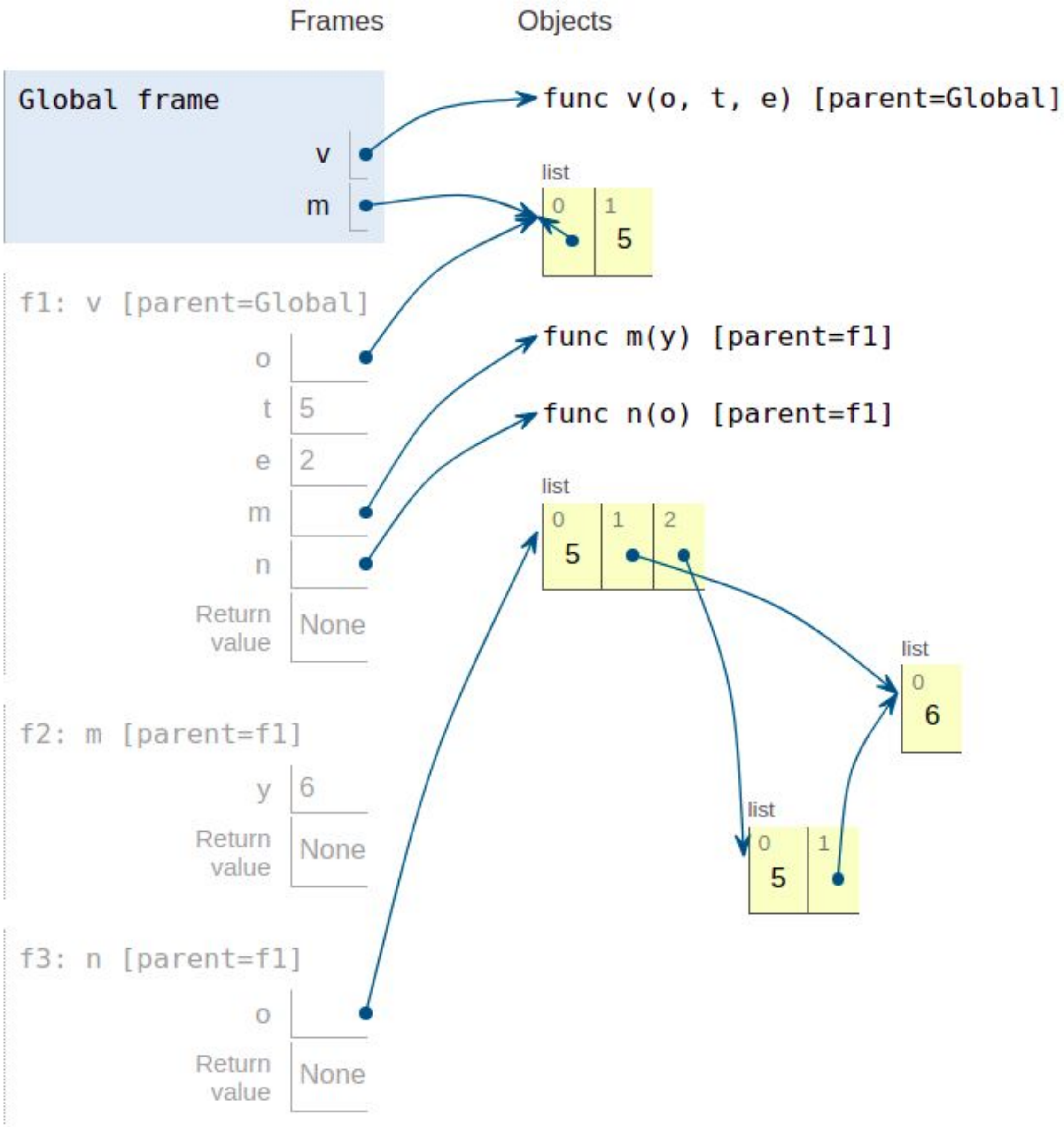
`list(o)`

`list([o])`

`o + []`

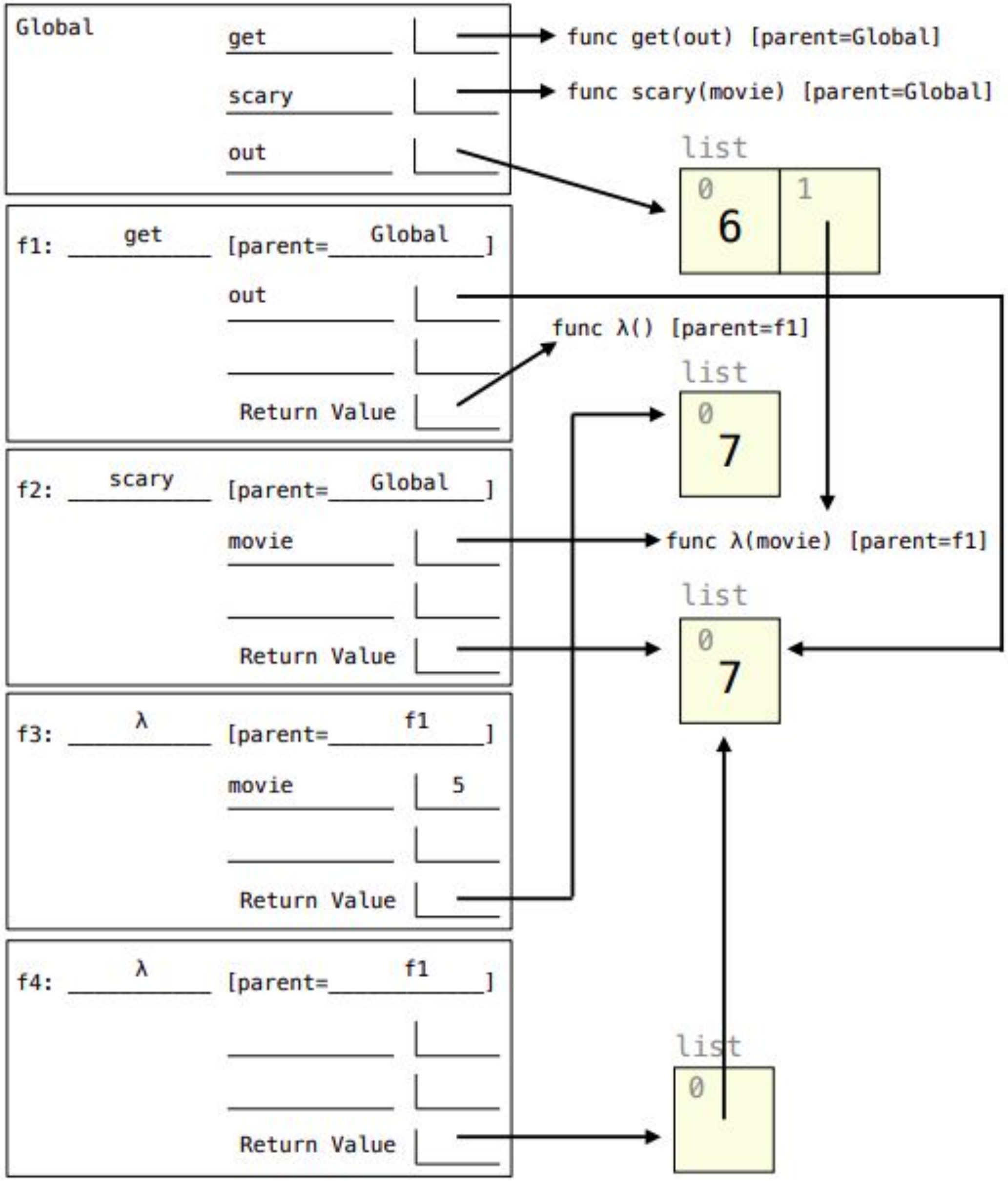
`[o[0], o[1]]`

`o[:]`



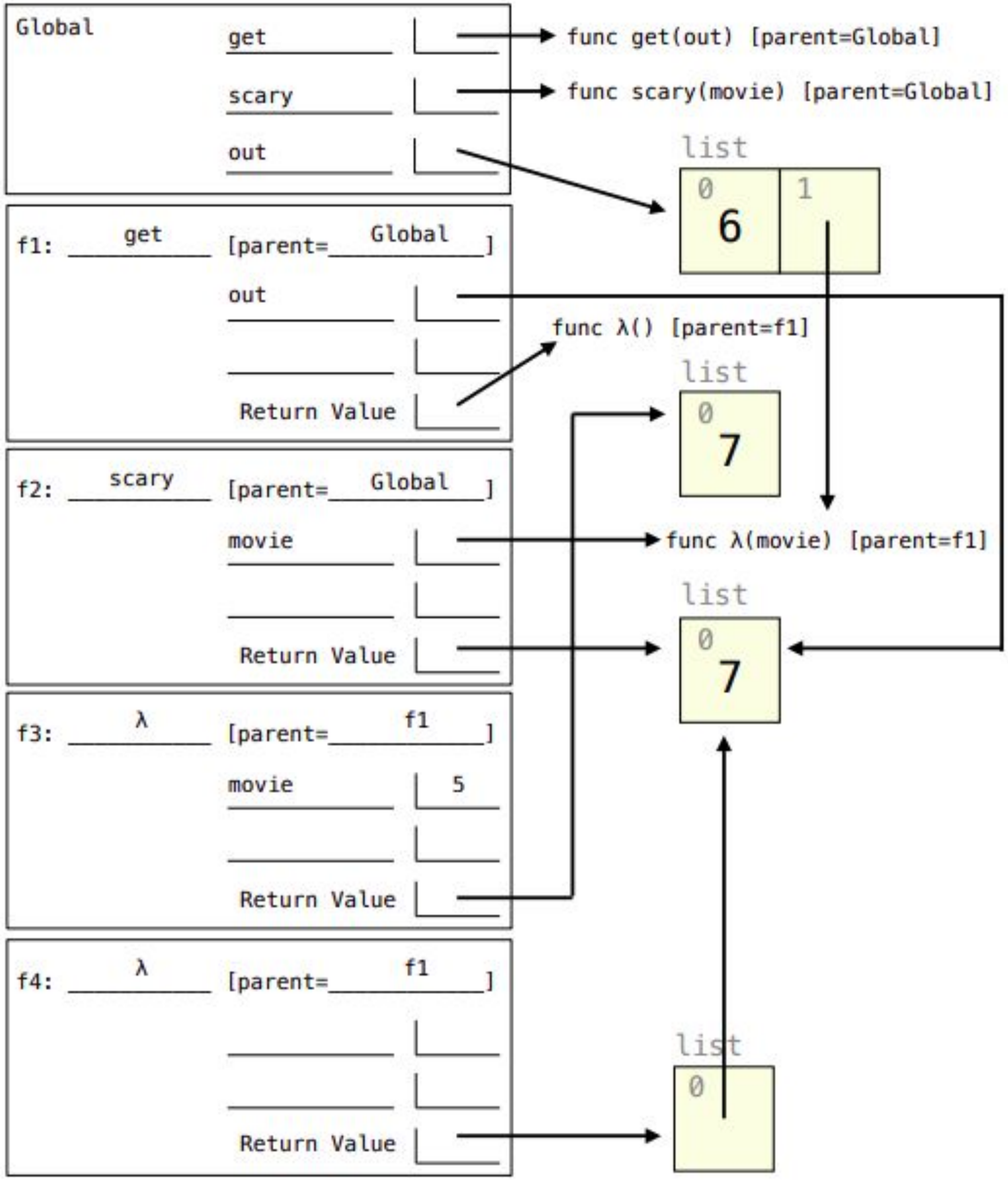
```
def get(out):
   out.pop()
   out = _____
   return lambda: [out]
def scary(movie):
   out.append(movie)
   return _____

out = [6]
_____
```



```
def get(out):
    out.pop()
    out = scary(lambda movie:
    out)
    return lambda: [out]
def scary(movie):
    out.append(movie)
    return movie(5)[:1]

out = [6]
get([7, 8])()
```



More Midterm Review