TREES, NONLOCAL, ORDERS OF GROWTH

COMPUTER SCIENCE 61A

October 4, 2016

1 Trees

1.1 Questions

1. Write a function `num_occurences` that takes in a tree `t` and a number `x`. It returns the number of times that `x` appears in `t`

```
def num_occurences(t, x):
    """Returns number of times x appears in t

    >>> t = tree(1, [tree(3),
    ... tree(3, [tree(4, [tree(5, [tree(1)])]),
    ...    tree(7)])])
    >>> num_occurences(t, 1)
    2
    >>> num_occurences(t, 2)
    0
    """
```

Solution:

```
    count = 0
    if root(t) == x:
        count += 1
    for b in branches(t):
        count += num_occurences(b, x)
    return count
```

2. Write a function `has_path` that takes in a tree `t` and a string `word`. It returns `True` if there is a path that starts from the root where the entries along the path spell out the word, and `False` otherwise.

```
def has_path(t, word):
    """Return whether there is a path in a tree where the
       entries along the path spell out a particular word.

    >>> greetings = tree('h', [tree('i'),
    ...   tree('e', [tree('l', [tree('l', [tree('o')])]),
    ...        tree('y')])])
    >>> print_tree(greetings)
    h
      i
      e
        l
          l
            o
        y
    >>> has_path(greetings, 'h')
    True
    >>> has_path(greetings, 'i')
    False
    >>> has_path(greetings, 'hi')
    True
    >>> has_path(greetings, 'hello')
    True
    >>> has_path(greetings, 'hey')
    True
    >>> has_path(greetings, 'bye')
    False

    """
```

Solution:

```
    if root(t) != word[0]:
        return False
    elif len(word) == 1:
        return True
    for b in branches(t):
        if has_path(b):
            return True
    return False
```

3. In the first week of class, we learned that expressions like `mul(sub(4, 5), add(2, 3))` can be represented as expression trees. In this problem we use our `tree` abstract data structure to further explore this idea.
   Write a function `evaluate`, which takes a binary expression tree, `exp` and returns what that expression would evaluate to. You may assume that all operators will take exactly 2 arguments, and that nodes always have either 0 or 2 children (never 1).

```
def evaluate(exp):
    """Evaluates exp, which is an expression tree.

    >>> from operator import add, sub, mul
    >>> exp = tree(3) # 3
    >>> evaluate(exp)
    3
    >>> exp = tree(add, [tree(3), tree(4)]) # add(3, 4)
    >>> evaluate(exp)
    7
    >>> exp = tree(mul, [tree(add, [tree(3), tree(5)]),
        tree(sub,
        [tree(5), tree(2)])]) # mul(add(3, 5), sub(5, 2))

    >>> evaluate(exp)
    24
    """
```

Solution:

```
    if is_leaf(exp):
        return root(exp)
    return root(exp)(evaluate(branches(exp)[0]), evaluate(
       branches(exp)[1]))
```

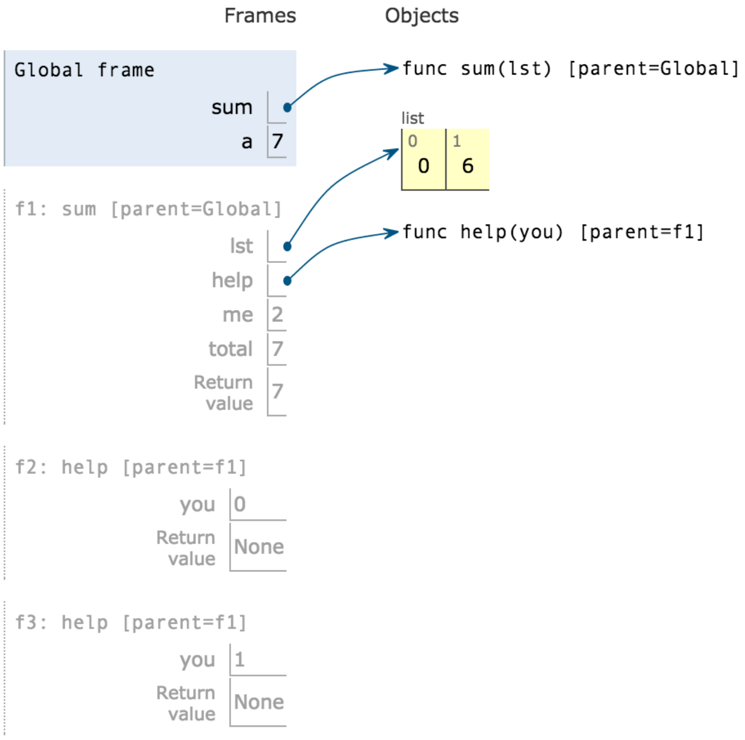
2 Nonlocal

2.1 Questions

1. Draw the environment diagram for the code below:

```
def sum(lst):
    total = 0
    def help(you):
        nonlocal total
        total += lst[you]
        lst[you] = total - lst[you]
    me = 0
    while me < len(lst):
        help(me)
        me += 1
    return total
a = sum([6, 1])
```

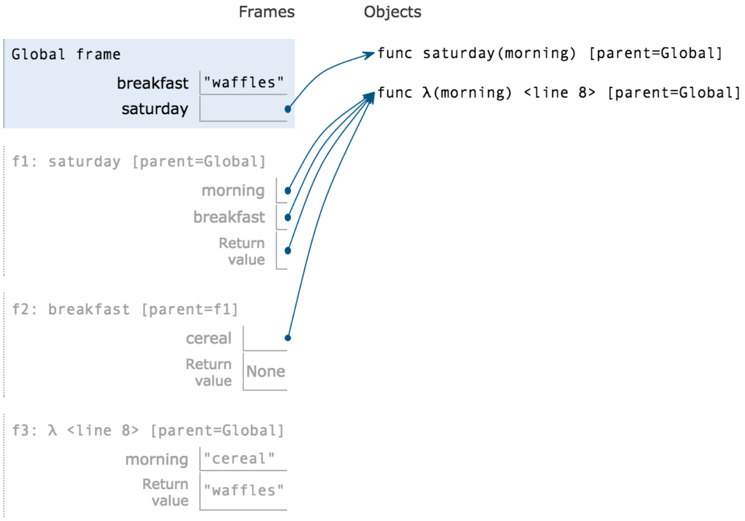
Solution:



2. Draw the environment diagram for the code below:

```
breakfast = 'waffles'
def saturday(morning):
    def breakfast(cereal):
        nonlocal breakfast
        breakfast = cereal
    breakfast(morning)
    return breakfast
saturday(lambda morning: breakfast)('cereal')
```

Solution:


Frames
Objects
Global frame
breakfast
"waffles"
saturday
func saturday(morning) [parent=Global]
func λ(morning) <line 8> [parent=Global]
f1: saturday [parent=Global]
morning
breakfast
Return value
f2: breakfast [parent=f1]
cereal
Return value
None
f3: λ <line 8> [parent=Global]
morning
"cereal"
Return value
"waffles"

3 Orders of Growth

3.1 Questions

1. What is the order of growth for a call to `fizzle(n)`?

```
def fizzle(n):
    if n <= 0:
        return n
    elif n % 23 == 0:
        return n
    return fizzle(n - 1)
```

Solution: $\theta(1)$

2. What is the order of growth for a call to `explode(n)`?

```
def boom(n):
    if n == 0:
        return "BOOM!"
    return boom(n - 1)

def explode(n):
    if n == 0:
        return boom(n)
    i = 0
    while i < n:
        boom(n)
        i += 1
    return boom(n)
```

Solution: $\theta(n^2)$

3. What is the order of growth for a call to `dreams(n)`?

```
def dreams(n):
    if n <= 0:
        return n
    if n > 0:
        return n + dreams(n // 2)
```

Solution: $\theta(log n)$