Final Examples

Announcements

Tree-Structured Data

def tree(label, branches=[]):
    return [label] + list(branches)

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:] 

class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        selfbranches = list(branches)

A tree can contain other trees:

[5, [6, 7], 8, [[9], 10]]
(+ 5 (- 6 7) 8 (* (- 9) 10))

(S NP (JJ Short) (NNS cuts))
(VP (VBP make) (NP (JJ Long) (NNS delays))

Tree processing often involves recursive calls on subtrees.

Trees

Solving Tree Problems

Implement bigs, which takes a Tree instance t containing integer labels. It returns the number of nodes in t whose labels are larger than any labels of their ancestor nodes.

def bigs(t):
    """Return the number of nodes in t that are larger than all their ancestors."
    tlabel = t.label
    if t.is_leaf():
        return ___
    return ___(f(b, tlabel)
    for b in t.branches)

The root label is always larger than all of its ancestors.

Implement bigs, which takes a Tree instance t containing integer labels. It returns the number of nodes in t whose labels are larger than any labels of their ancestor nodes.

Recursive Accumulation
Implement `bigs`, which takes a Tree instance `t` containing integer labels. It returns the number of nodes in `t` whose labels are larger than any labels of their ancestor nodes.

```scheme
(define (bigs t)
  (if (is-leaf? t) 0
      (+ (if (not (is-leaf? t)) 1 (bigs (first-branch t)))
          (bigs (rest-branches t))))
```

Implement `smalls`, which takes a Tree instance `t` containing integer labels. It returns the non-leaf nodes in `t` whose labels are smaller than any labels of their descendant nodes.

```scheme
(define (smalls t)
  (if (is-leaf? t) t
      (if (not (is-leaf? t))
          (let ((smaller (first-branch t))
                (rest (rest-branches t)))
              (if (smaller? (label smaller) (label t))
                  (cons smaller (smalls (rest-branches t)))
                  (smalls (rest-branches t))))))
```

### Applying the Design Process

**Signature, Purpose Statement, Header**

```
(f (if (> 3 2) 4 5))
```

**Function Template**

```
def f(x, y):
  return x + y
```

**Function Definition**

```
def f(x, y):
  return x + y
```

**Testing**

```
assert f(3, 2) == 5
assert f(4, 5) == 9
```

**Testing**

```
assert f(3, 2) == 5
assert f(4, 5) == 9
```

### Interpreter Analysis

How many times does `scheme_eval` get called when evaluating the following expressions?

```
(define x 5)
define y 6
```

```
define if y (if y 5 6)
define if x (if x 5 6)
```

```
(define if y (if y 5 6))
define if x (if x 5 6)
```

```
(define if y (if y 5 6))
define if x (if x 5 6)
```

---

### Expression Trees

How can you design a function to process a tree and return its non-leaf nodes whose labels are smaller than any labels of their descendant nodes? Consider the following tree:

```
  1
 / \
2   3
 / \ /
4   5 6
```

What is the signature of this function? How would you write the function template and define it? Provide examples to test your function.