Final Examples

Announcements

Fall 2021 CATS Contest Winners

`secrued`

`secured`

`thinves`

`things`

3rd Place: (Young Park + Jonathan Yue) and (Charlie Ji + Hans Mao)

2nd Place: Jiayin Lin + Jay Dang

1st Place: Nishant Bhakar

Trees

Tree-Structured Data

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

def label(t):
    return t[0]

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

def is_leaf(t):
    return not branches(t)

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

    def is_leaf(self):
        return not self.branches
```

A tree can contains 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)))
  (. .))

<ul>
  <li>Midterm <b>1</b></li>
  <li>Midterm <b>2</b></li>
</ul>
```

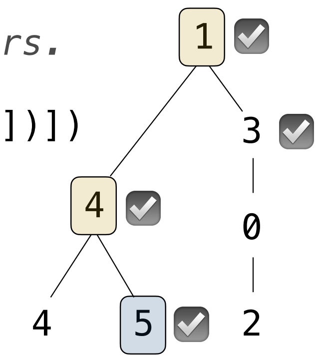
Tree processing often involves recursive calls on subtrees

Tree Processing

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 all labels of their ancestor nodes.

```
def bigs(t):
    """Return the number of nodes in t that are larger than all their ancestors.

    >>> a = Tree(1, [Tree(4, [Tree(4), Tree(5)]), Tree(3, [Tree(0, [Tree(2)])])])
    >>> bigs(a)
    4
```



The root label is always larger than all of its ancestors

```
if t.is_leaf():
    return ___
else:
    return ___([___ for b in t.branches])
```

Somehow increment the total count

Somehow track a list of ancestors

```
if node.label > max(ancestors):
```

Somehow track the largest ancestor

```
if node.label > max_ancestors:
```

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 all labels of their ancestor nodes.

```
def bigs(t):
    """Return the number of nodes in t that are larger than all their ancestors.

    >>> a = Tree(1, [Tree(4, [Tree(4), Tree(5)]), Tree(3, [Tree(0, [Tree(2)])])])
    >>> bigs(a)
    4
    """
    def f(a, x):
        if a.label > x:
            return 1 + sum([f(b, a.label) for b in a.branches])
        else:
            return sum([f(b, x) for b in a.branches])
    return f(t, t.label - 1)
```

A node

max_ancestor

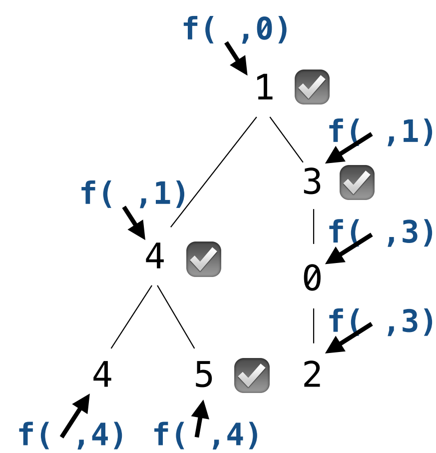
Somehow track the largest ancestor

node.label > max_ancestors

Somehow increment the total count

Root label is always larger than its ancestors

Some initial value for the largest ancestor so far...



Recursive Accumulation

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."""
    n = [0]

    def f(a, x):
        if a.label > x:
            n[0] += 1
        for b in a.branches:
            f(b, max(a.label, x))
    f(t, t.label - 1)

    return n[0]
```

Somehow track the largest ancestor

node.label > max_ancestors

Somehow increment the total count

Root label is always larger than its ancestors

Designing Functions

How to Design Programs

From Problem Analysis to Data Definitions
Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

Signature, Purpose Statement, Header
State what kind of data the desired function consumes and produces. Formulate a concise answer to the question *what* the function computes. Define a stub that lives up to the signature.

Functional Examples
Work through examples that illustrate the function's purpose.

Function Template
Translate the data definitions into an outline of the function.

Function Definition
Fill in the gaps in the function template. Exploit the purpose statement and the examples.

Testing
Articulate the examples as tests and ensure that the function passes all. Doing so discovers mistakes. Tests also supplement examples in that they help others read and understand the definition when the need arises—and it will arise for any serious program.

https://htdp.org/2018-01-06/Book/

Applying the Design Process

Designing a Function

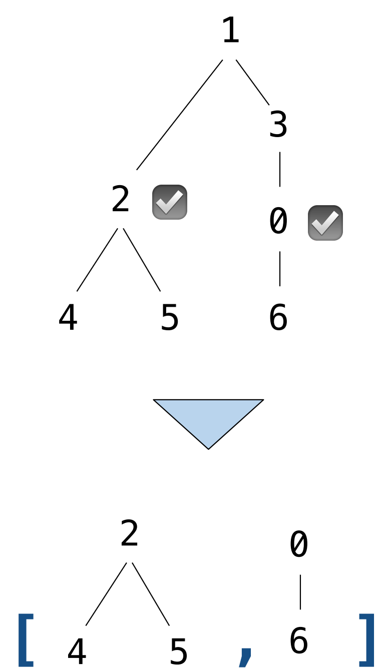
Implement **smalls**, which takes a Tree instance t containing integer labels. It returns the non-leaf nodes in t whose labels are smaller than any labels of their descendant nodes.

Signature: Tree -> List of Trees

```
def smalls(t):
    """Return the non-leaf nodes in t that are smaller than all their descendants.

    >>> a = Tree(1, [Tree(2, [Tree(4), Tree(5)]), Tree(3, [Tree(0, [Tree(6)])])])
    >>> sorted([t.label for t in smalls(a)])
    [0, 2]
    """
    result = []
    def process(t):
        if t.is_leaf():
            return t.label
        else:



            return min(...)
    process(t)
    return result
```

Signature: Tree -> number

"Find smallest label in t & maybe add t to result"



Designing a Function

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.

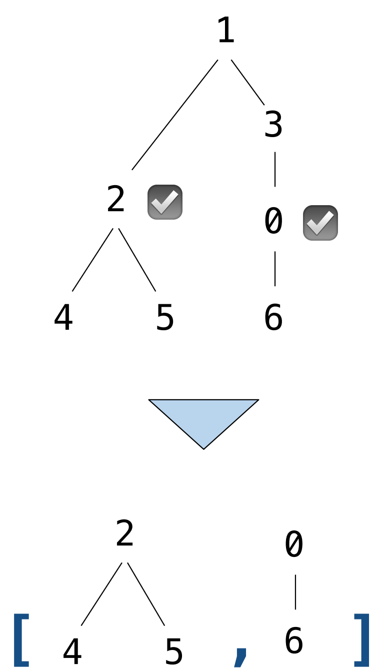
Signature: Tree -> List of Trees

```
def smalls(t):
    """Return the non-leaf nodes in t that are smaller than all their descendants.

    >>> a = Tree(1, [Tree(2, [Tree(4), Tree(5)]), Tree(3, [Tree(0, [Tree(6)])])])
    >>> sorted([t.label for t in smalls(a)])
    [0, 2]
    """
    result = []
    def process(t):
        if t.is_leaf():
            return ___t.label___
        else:
            smallest = ___min([process(b) for b in t.branches])___
            if ___t.label < smallest___:
                ___result.append( t )___
            return min(smallest, t.label)
    process(t)
    return result
```

Signature: Tree -> number

"Find smallest label in t & maybe add t to result"

smallest label in a branch of t



Interpreters

Interpreter Analysis

```
What expressions are passed to scheme_eval when evaluating the following expressions?
```

```
(define x (+ 1 2))

(define (f y) (+ x y))

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

Matching Patterns

Regular Expressions

Which strings match which patterns?

Strings:

Patterns:	abc	cab	bac	baba	ababca	aabcc	abbaba
[abc]*	✅	✅	✅	✅	✅	✅	✅
a*b*c*	✅	❌	❌	❌	❌	✅	❌
ab|[bc]*	❌	❌	❌	❌	❌	❌	❌
(a[bc]+)+a?	✅	❌	❌	❌	✅	❌	✅
(ab|ba)+(ab|[bc])?	✅	❌	✅	✅	❌	❌	✅