Designing Functions

Announcements

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

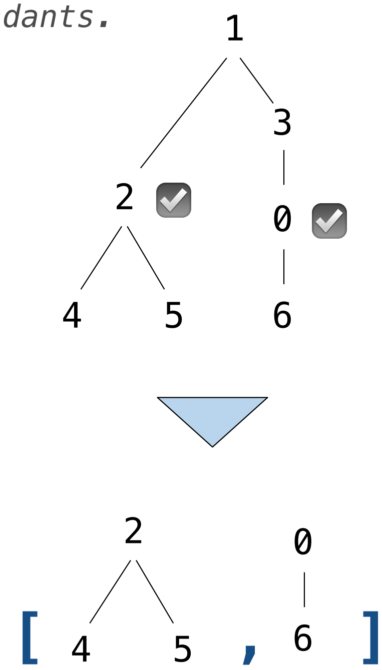
Implement **smalls**, which takes a Tree instance t containing integer labels. It returns the non-leaf <u>nodes</u> in t whose labels are smaller than any labels of their descendant nodes.

Signature: Tree -> List of Trees

```python
def smalls(t):
    """Return a list of 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.

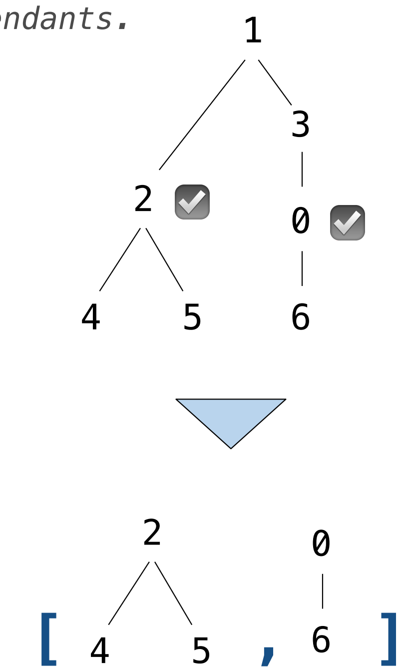
Signature: Tree -> List of Trees

```python
def smalls(t):
    """Return a list of 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



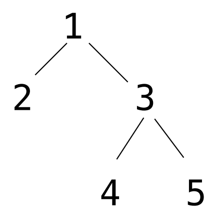
More Tree Practice

61A Fall 2015 Final Question 3 [Extended Remix]

Definition. A *full path* through a Tree is a list of adjacent node labels that starts with the root label and ends with a leaf label.

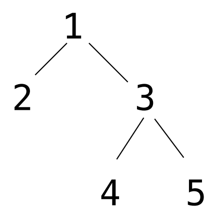
```
def count_big(t, n):
    """Return the number of paths in t that have a sum larger or equal to n.

    >>> t = Tree(1, [Tree(2), Tree(3, [Tree(4), Tree(5)])])
    >>> count_big(t, 3)
    3
    >>> count_big(t, 6)
    2
    >>> count_big(t, 9)
    1
    """
    if t.is_leaf():
        return one(t.label >= n)
    else:
        return sum([count_big(b, n-t.label) for b in t.branches])
```

```
def one(b):
    if b:
        return 1
    else:
        return 0
```



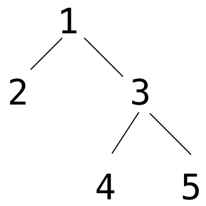
61A Fall 2015 Final Question 3 [Extended Remix]

```
def print_big(t, n):
    """Print the paths in t that have a sum larger or equal to n.

    >>> t = Tree(1, [Tree(2), Tree(3, [Tree(4), Tree(5)])])
    >>> print_big(t, 3)
    [1, 2]
    [1, 3, 4]
    [1, 3, 5]
    >>> print_big(t, 6)
    [1, 3, 4]
    [1, 3, 5]
    >>> print_big(t, 9)
    [1, 3, 5]
    """
    def helper(t, p):
        p = p + [t.label]
        if t.is_leaf():
            if sum(p) >= n:
                print(p)
        else:
            for b in t.branches:
                helper(b, p)
    helper(t, [])
```



61A Fall 2015 Final Question 3 [Extended Remix]

```
def big_links(t, n):
    """Yield the paths in t that have a sum larger or equal to n as linked lists.

    >>> t = Tree(1, [Tree(2), Tree(3, [Tree(4), Tree(5)])])
    >>> for p in big_links(t, 3):
    ...     print(p)
    <1 2>
    <1 3 4>
    <1 3 5>
    >>> for p in big_links(t, 6):
    ...     print(p)
    <1 3 4>
    <1 3 5>
    >>> for p in big_links(t, 9):
    ...     print(p)
    <1 3 5>
    """
    if ___t.is_leaft() and t.label >= n___:
        yield ___Link(t.label)___
    for b in t.branches:
        for x in ___big_links(b, n - t.label)___:
            yield ___Link(t.label, x)___
```



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))
```