Designing Functions

How to Design Programs

Problem Solving Process from *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/

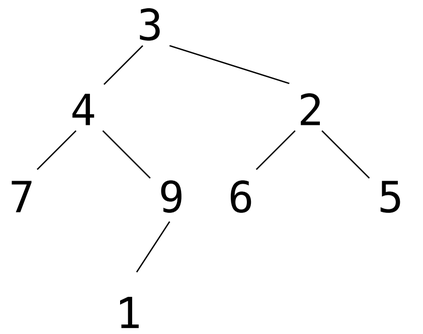
Applying the Design Process

Designing a Function

Implement **as_dict**, which takes a Tree instance t containing **unique** integer labels. It returns a dictionary in which each non-leaf label is a key and its value is a list of the labels of its children.

```
def as_dict(t):  Signature: Tree -> Dictionary from int to list[int]
    """Return a dictionary containing a key for each non-leaf label in t for which the
    corresponding value is a list of its children's labels. Assume all labels in t are unique.

    >>> as_dict(Tree(3, [Tree(4, [Tree(7), Tree(9, [Tree(1)])]), Tree(2, [Tree(6), Tree(5)])]))
    {3: [4, 2], 4: [7, 9], 9: [1], 2: [6, 5]}
    """
    result = {}
    def f(x):  Signature: Tree -> None ; adds a key-value pair to result
        if not x.is_leaf():   "Add {x.label: [list of x's children]} to result and recurse"
            s = []
            result[x.label] = s
            for b in x.branches:
                s.append(b.label)
                f(b)
    f(t)
    return result
```



More Tree Practice

61A Fall 2015 Final Question 3 [Extended Remix]

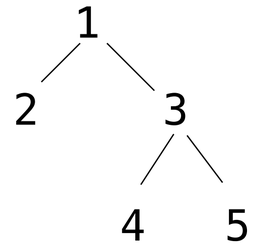
Given a Tree instance t and an integer n, count the number of paths in t such that the labels on the path sum to at least n.

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

Signature: (Tree, int) -> number of paths

```
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])____
```

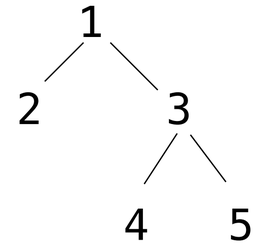



```
def one(b):
    if b:
        return 1
    else:
        return 0
```

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, [])
```



Signature: (Tree, list[int]) -> None

"For a node & partial path, extend the path & maybe print it"