

Final Examples

Announcements

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 `big`, 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 big(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)])])])
```

```
    >>> big(a)
```

```
    4
```

```
    """
```

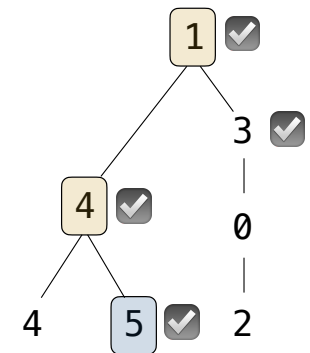
The root label is always larger than all of its ancestors

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

Somehow increment
the total count

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

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



Somehow track a
list of ancestors

Somehow track the
largest ancestor

Solving Tree Problems

Implement `big`, 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 big(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)])])]
>>> big(a)
```

```
4
"""
```

Somehow track the largest ancestor

```
def f(a, x):
    A node in t → max_ancestor
    if a.label > x: node.label > max_ancestors :
```

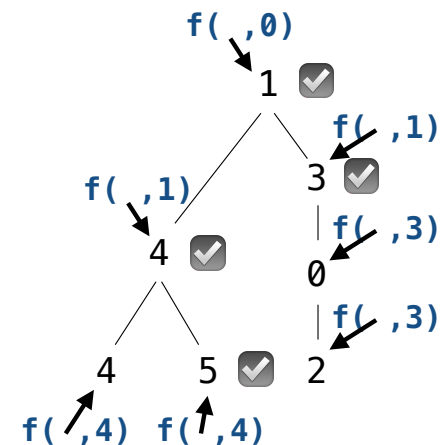
```
    return 1 + sum([f(b, a.label) for b in a.branches])
```

```
else:
    Somehow increment the total count
```

```
    return sum([f(b, x) for b in a.branches])
```

```
return f(t, t.label - 1)
    Root label is always larger than its ancestors
```

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



Recursive Accumulation

Solving Tree Problems

Implement `big`, 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 big(t):  
    """Return the number of nodes in t that are larger than all their ancestors."""  
    n = 0  
    def f(a, x):  
        nonlocal n  
  
        if a.label > x:  
            n += 1  
            for b in a.branches:  
                f(b, max(a.label, x))  
  
    f(t, t.label - 1)  
  
    return n
```

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.

Applying the Design Process

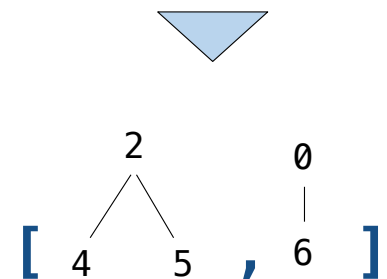
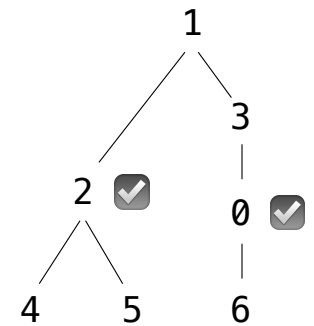
Designing a Function

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.

```
def smalls(t): Signature: Tree -> List of Trees
    """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 = []
    Signature: Tree -> number
    def process(t): "Find smallest label in t & maybe add t to result"
        if t.is_leaf():
            return t.label
        else:
            return min(...)
    process(t)
    return result
```



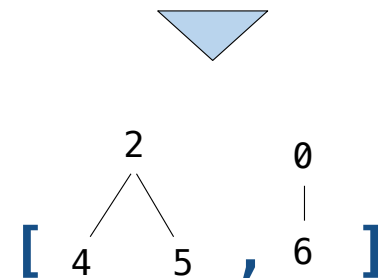
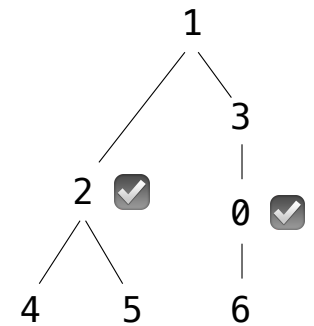
Designing a Function

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.

```
def smalls(t): Signature: Tree -> List of Trees
    """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 = []
    Signature: Tree -> number
    def process(t): "Find smallest label in t & maybe add t to result"
        if t.is_leaf():
            return t.label
        else:
            smallest = min([process(b) for b in t.branches])
            smallest label in a branch of t if t.label < smallest:
                result.append( t )
            return min(smallest, t.label)
    process(t)
    return result
```



Society

Privacy Policies and Laws

Mark Zuckerberg in San Francisco, January 8, 2010

"People have really gotten comfortable not only sharing more information and different kinds, but more openly and with more people. That social norm is just something that has evolved over time."

Tim Cook in Brussels, October 24, 2018

"We at Apple are in full support of a comprehensive federal privacy law in the United States. There, and everywhere, it should be rooted in four essential rights:

- First, the right to have **personal data minimized**. Companies should challenge themselves to de-identify customer data, or not to collect it in the first place.
- Second, the **right to knowledge**. Users should always know what data is being collected and what it is being collected for. This is the only way to empower users to decide what collection is legitimate and what isn't. Anything less is a sham.
- Third, the **right to access**. Companies should recognize that data belongs to users, and we should all make it easy for users to get a copy of, correct, and delete their personal data.
- And fourth, the **right to security**. Security is foundational to trust and all other privacy rights."

Perils of Sharing

A persistent source of privacy breaches: sending a message to an unintended recipient

Grandmas keep accidentally tagging themselves as Grandmaster Flash on Facebook



Grandmaster Flash was mentioned in a post.



Darla Smeltekop

July 5 · 🌐

Happy birthday Cassie and Jessie. it is hard to believe 20 years have gone by so fast. Wish we could be their . Love Grandpa and [Grandmaster Flash](#)

Share

👍 3 people like this.

Grandmaster Flash was mentioned in a post.



Evelyn Shoemaker

July 5 · 🌐

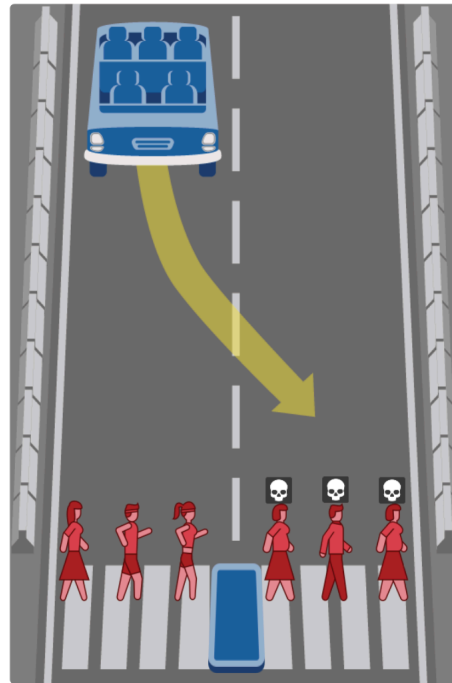
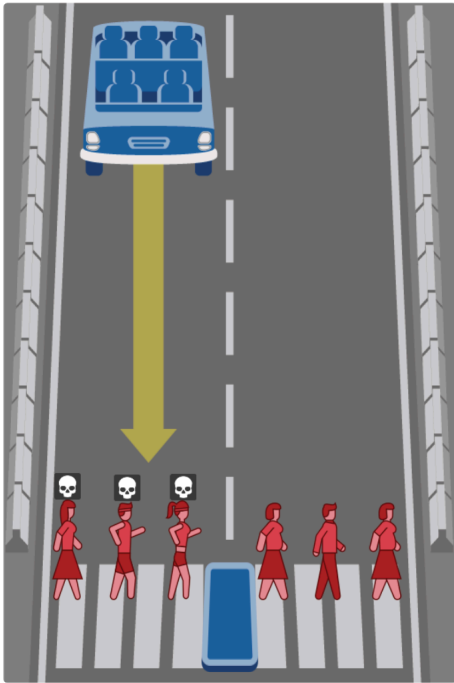
Happy bdat Jaden. Have a great day. Your card has been mailed. Love you. [Grandmaster Flash](#)

Share

Software

Automated Decision Making

What should the self-driving car do?



Self Driving Vehicle (SDV) Overview

Top mounted lidar units provide a 360° 3-dimensional scan of the environment

Side and rear facing cameras work in collaboration to construct a continuous view of the vehicle's surroundings

Roof mounted antennae provide GPS positioning and wireless data capabilities

Forward facing camera array focus both close and far field, watching for braking vehicles, crossing pedestrians, traffic lights, and signage

360° radar coverage

Custom designed compute and storage allow for real-time processing of data while a fully integrated cooling solution keeps components running optimally

Self-Driving System Sensors

- Rear facing cameras for lane changes
- x5 wide FOV cameras for 360° medium range imaging
- x4 OEM surround view cameras for 360° close range imaging
- x1 narrow FOV forward stereo camera for long range sensing
- x12 ultrasonic sensors on sides for additional coverage
- x8 ultrasonic sensors on front/rear bumper for close range sensing
- x1 central LIDAR for 360° medium range sensing (blind spot close to vehicle)
- x8 narrow, long range radar (wide, medium range mode not shown) for 360° sensing

UBER ATG
CONFIDENTIAL BUSINESS INFORMATION. EXEMPT FROM DISCLOSURE UNDER FOIA.
Confidential Treatment Requested / Exempt from Disclosure under FOIA
UBER_NTSSB_0002435

Life