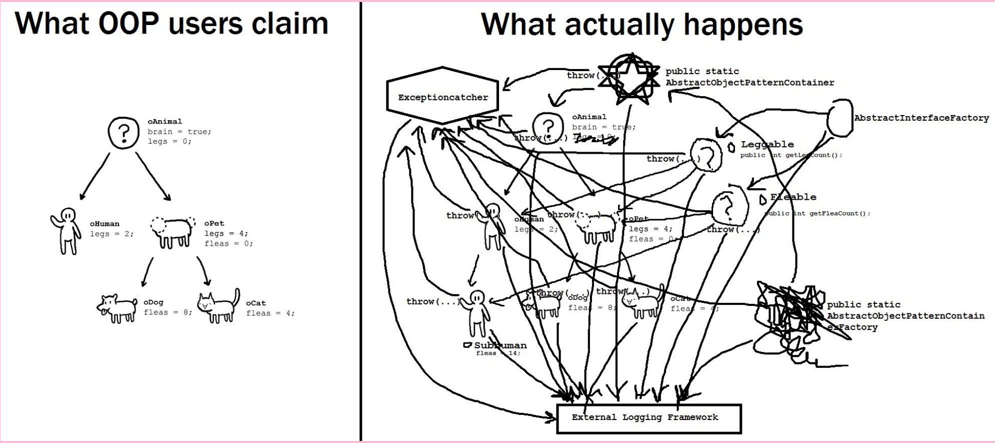


Hi! Welcome to 61A Discussion :)

We will begin at **5:10**!

Attendance: **go.cs61a.org/ben-disc**

Slides: **cs61a.bencuan.me**

Announcements

- Ants phase 1 & HW4 due tonight!

Agenda

- Attendance
- String __repr__esentation
- Trees 🌳🌳🌳

String Representation

String rep: what and why?

- Magic python functions __str__ and __repr__ to convert objects into strings (text)
- Makes debugging a lot easier
- Compare contents of objects

str() vs repr()

str:

- Human friendly
  (easy to read)

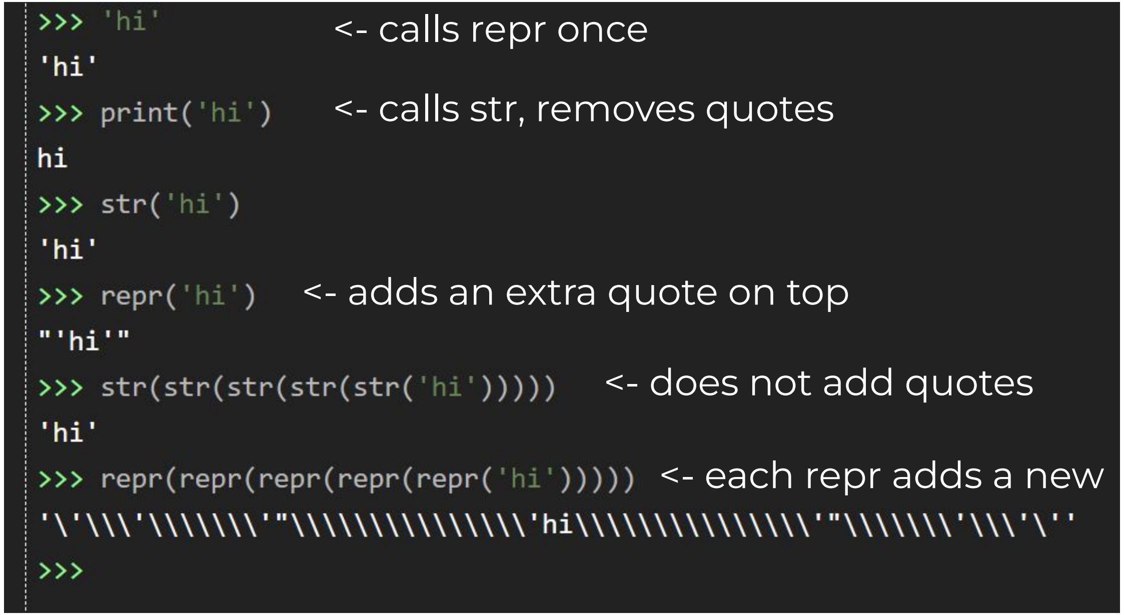
- Called by str() and print()

repr:

- Machine friendly
  (prioritize completeness over readability)

- Called by repr() or by passing an object straight into interpreter

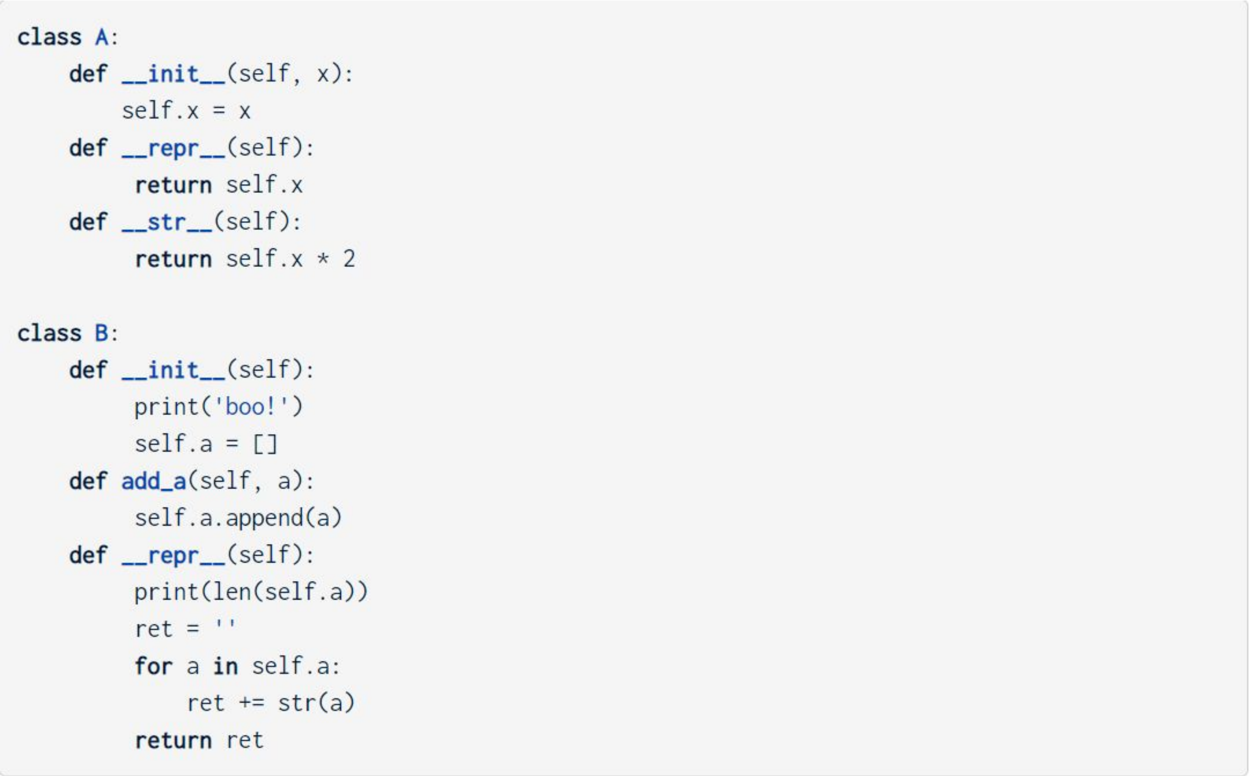
Some very subtle differences



```
>>> 'hi'                    <- calls repr once
'hi'
>>> print('hi')             <- calls str, removes quotes
hi
>>> str('hi')
'hi'
>>> repr('hi')            <- adds an extra quote on top
"'hi'"
>>> str(str(str(str(str('hi')))))      <- does not add quotes
'hi'
>>> repr(repr(repr(repr(repr('hi')))))  <- each repr adds a new set of quotes
'\'\\\'\\\\\\\'"\\\\\\\\\\\\\\\'hi\\\\\\\\\\\\\\\'"\\\\\\\'\\\'\''
>>>
```

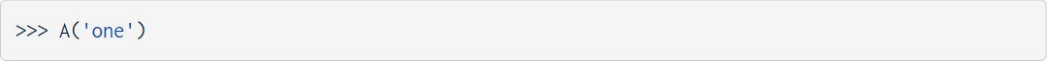
(...uhh whattt????)

Let’s try some WWPD!

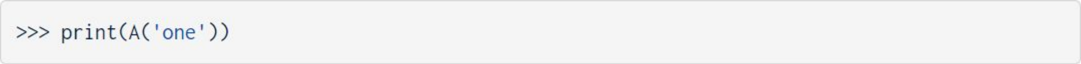


```
class A:
    def __init__(self, x):
        self.x = x
    def __repr__(self):
         return self.x
    def __str__(self):
         return self.x * 2

class B:
    def __init__(self):
         print('boo!')
         self.a = []
    def add_a(self, a):
         self.a.append(a)
    def __repr__(self):
         print(len(self.a))
         ret = ''
         for a in self.a:
             ret += str(a)
         return ret
```

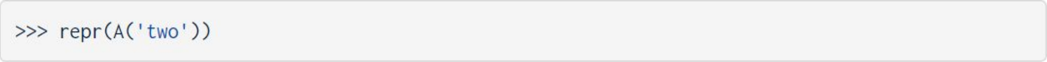
Let’s try some WWPD!



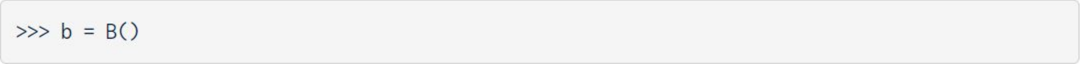
```
>>> A('one')
```



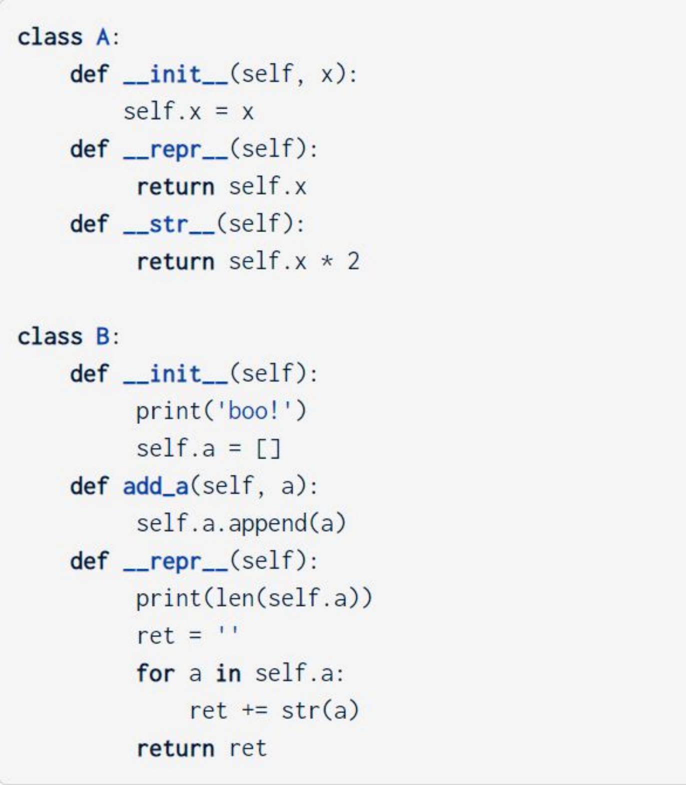
```
>>> print(A('one'))
```



```
>>> repr(A('two'))
```



```
>>> b = B()
```

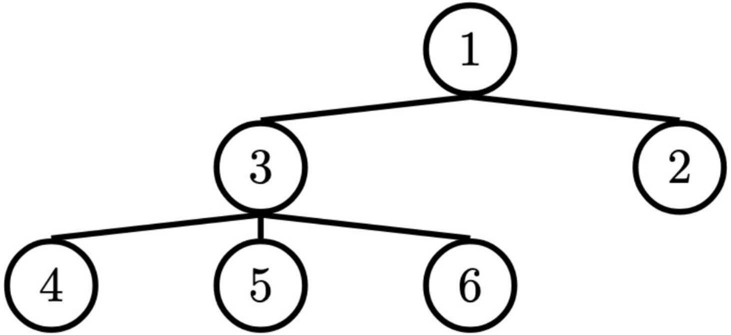


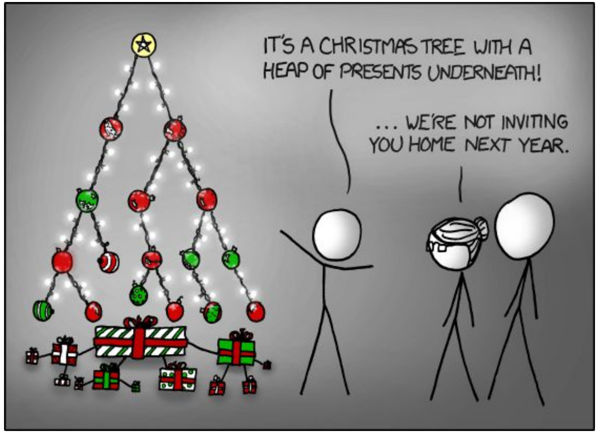
```
class A:
    def __init__(self, x):
        self.x = x
    def __repr__(self):
         return self.x
    def __str__(self):
         return self.x * 2

class B:
    def __init__(self):
         print('boo!')
         self.a = []
    def add_a(self, a):
         self.a.append(a)
    def __repr__(self):
         print(len(self.a))
         ret = ''
         for a in self.a:
             ret += str(a)
         return ret
```

Trees

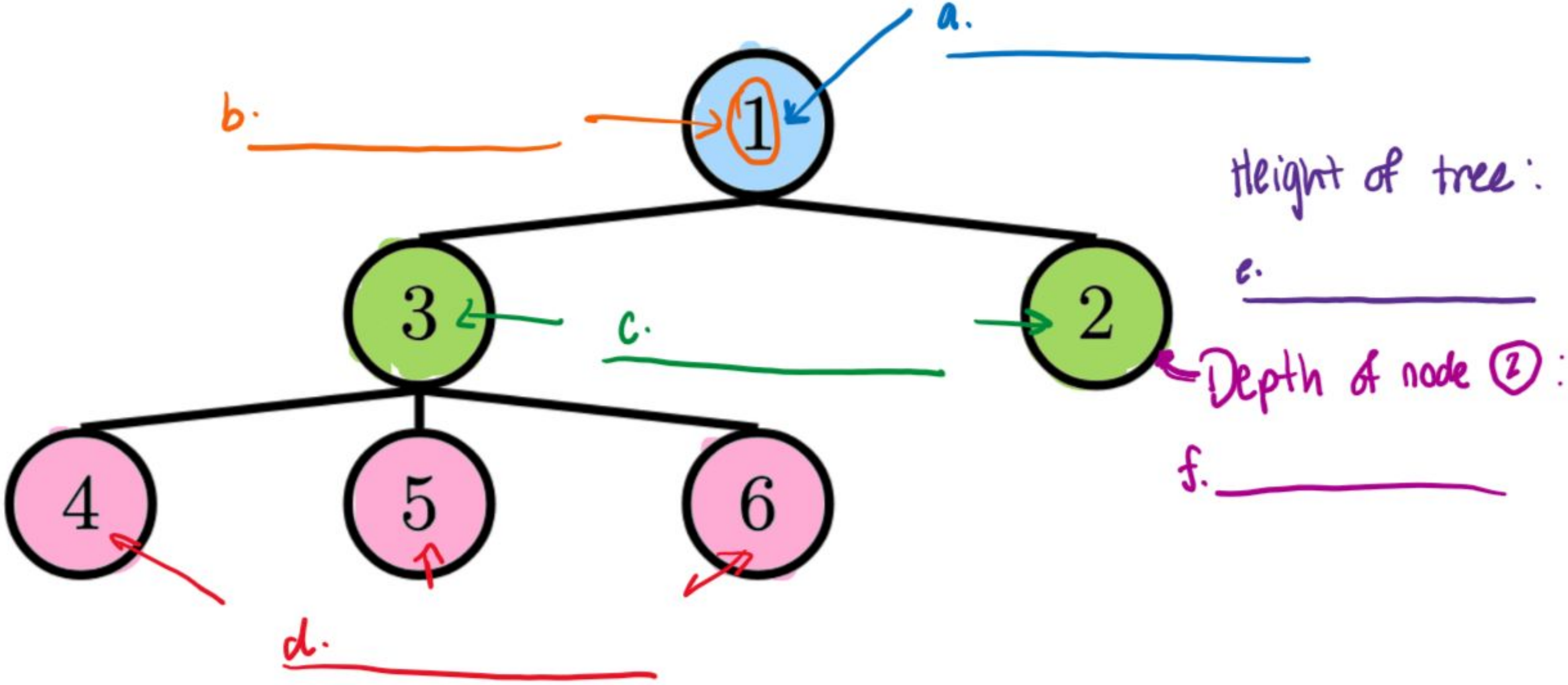
What are trees?

- A recursively defined object
- Two instance attributes: **label** and **branches**
- **Branches = list of more Trees!**
- **Leaf:** a tree with no branches

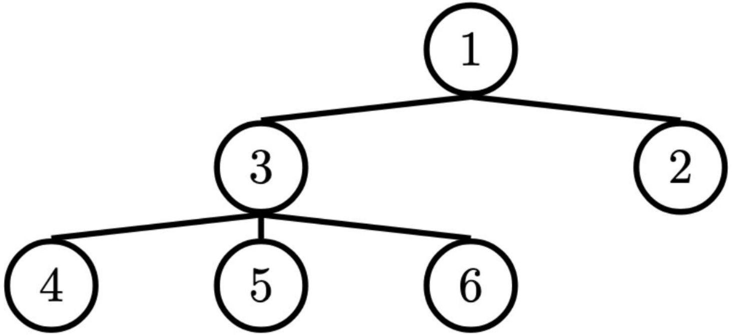



IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.

Label the tree!



The Tree Class



Tree(label, branches): Creates a new Tree object (runs __init__)

t.label: The label in this tree's node

t.branches: A **list** of **Trees** (child nodes)

t.is_leaf(): A **function** that returns True if t.branches is empty

IMPORTANT: Data Types!

Tree(label, branches)

- label can be anything.
- branches must be a **list of trees.**
- Returns a Tree object.

t.label

- can be any type (usually a number)

t.branches

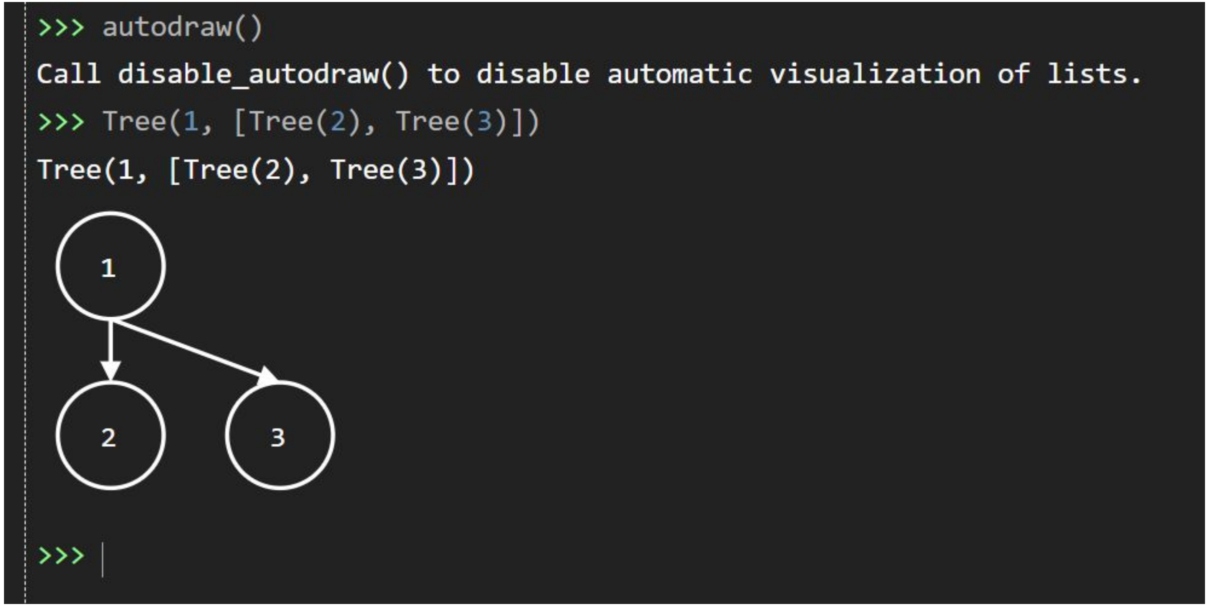
- must always be a **list of trees** (branches of t)

is_leaf(t)

- returns a **boolean** (True or False)

Autodraw demo

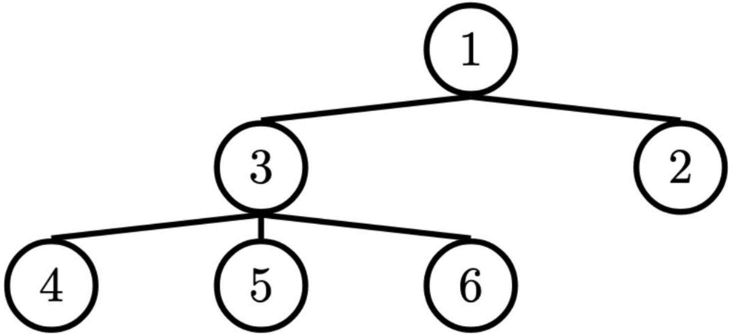
run autodraw() on code.cs61a.org to visualize trees!



```
>>> autodraw()
Call disable_autodraw() to disable automatic visualization of lists.
>>> Tree(1, [Tree(2), Tree(3)])
Tree(1, [Tree(2), Tree(3)])
```

```
>>>
```

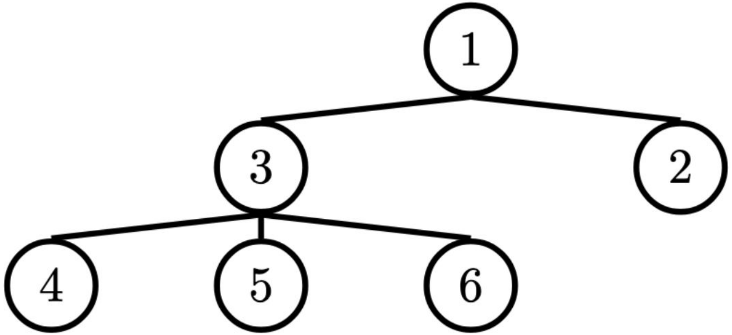
Tree coding



```
def tree_stuff(t):

    if t.is_leaf():
        return _______ (base case)

    else:
        result = [tree_stuff(b) for b in t.branches]
        return _______ (do something with the result)
```

Height (Q2)

Write a function that returns the height of a tree. Recall that the height of a tree is the length of the longest path from the root to a leaf.



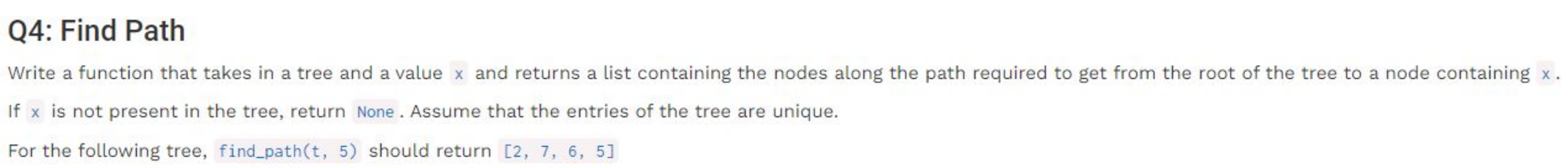
tree(label, branches)
label(t)
branches(t)
is_leaf(t)

base case?
what to do with result?

Q3: Maximum Path Sum

Write a function that takes in a tree and returns the maximum sum of the values along any path in the tree. Recall that a path is from the tree's root to any leaf.

tree(label, branches)
label(t)
branches(t)
is_leaf(t)

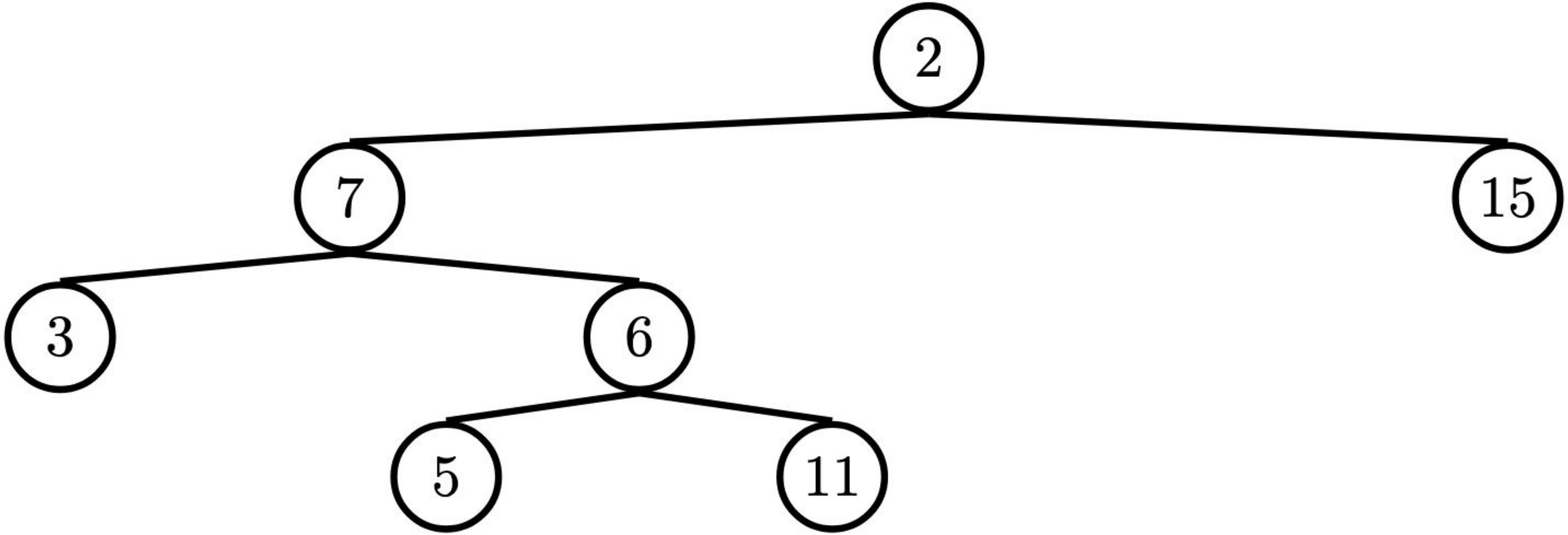


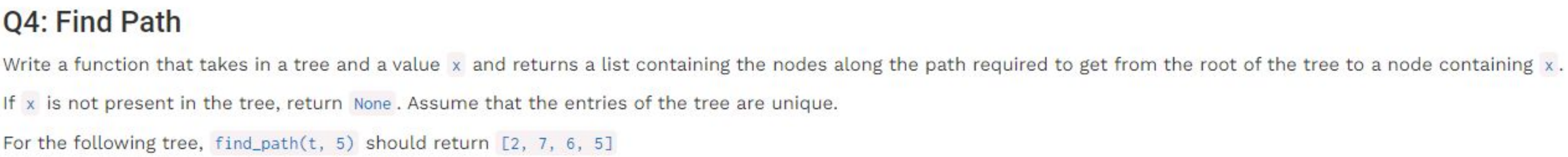
Q4: Find Path

Write a function that takes in a tree and a value `x` and returns a list containing the nodes along the path required to get from the root of the tree to a node containing `x`.

If `x` is not present in the tree, return `None`. Assume that the entries of the tree are unique.

For the following tree, `find_path(t, 5)` should return `[2, 7, 6, 5]`





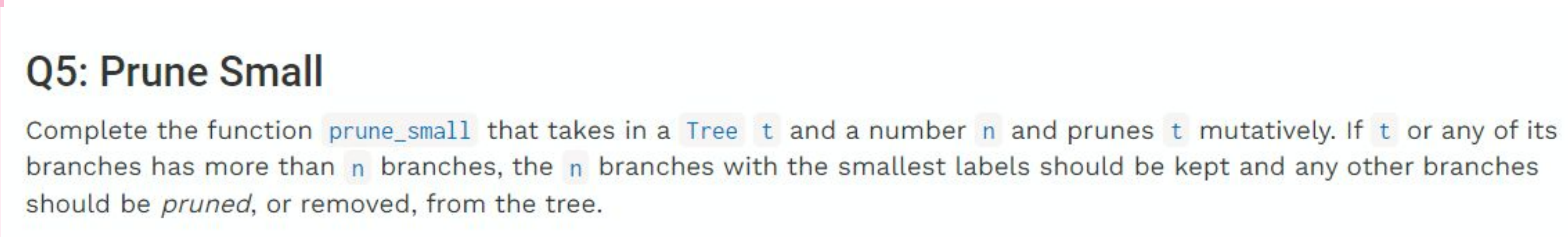
Q4: Find Path

Write a function that takes in a tree and a value `x` and returns a list containing the nodes along the path required to get from the root of the tree to a node containing `x`.

If `x` is not present in the tree, return `None`. Assume that the entries of the tree are unique.

For the following tree, `find_path(t, 5)` should return `[2, 7, 6, 5]`

```
def find_path(t,x):

    if
        return

    for
        path =

        if
            return
```

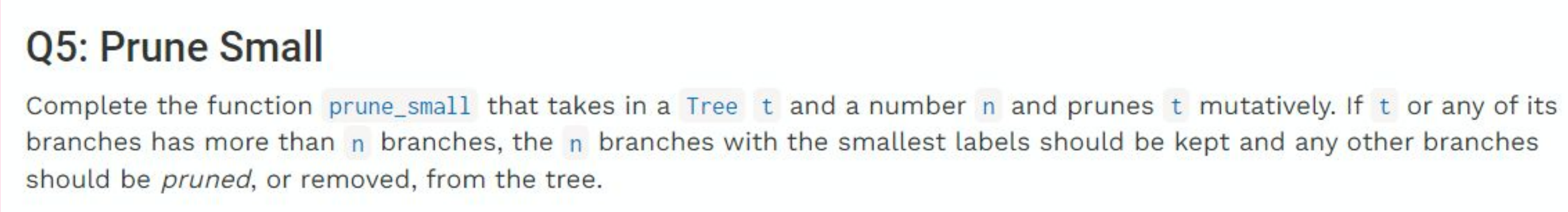


Q5: Prune Small

Complete the function `prune_small` that takes in a `Tree` `t` and a number `n` and prunes `t` mutatively. If `t` or any of its branches has more than `n` branches, the `n` branches with the smallest labels should be kept and any other branches should be *pruned*, or removed, from the tree.

mutate, no need to use returns anywhere!

- remember list mutation functions (pop, append, remove…)



Q5: Prune Small

Complete the function `prune_small` that takes in a `Tree` `t` and a number `n` and prunes `t` mutatively. If `t` or any of its branches has more than `n` branches, the `n` branches with the smallest labels should be kept and any other branches should be *pruned*, or removed, from the tree.

mutate, no need to use returns anywhere!

- remember list mutation functions (pop, append, remove...)

```
def prune_small(t, n):

    while
        largest = max(                    , key=lambda b: b.label)

    for
```