Generators

Announcements

Tree Practice

Spring 2023 Midterm 2 Question 4(a)

Implement exclude, which takes a tree t and a value x. It returns a tree containing the root node of t as well as each non-root node of t with a label not equal to x. The parent of a node in the result is its nearest ancestor node that is not excluded.

```
def exclude(t, x):
    """Return a tree with the non-root nodes of tree t labeled anything but x.

    >>> t = tree(1, [tree(2, [tree(2), tree(3), tree(4)]), tree(5, [tree(1)])])
    >>> exclude(t, 2)
    [1, [3], [4], [5, [1]]]
    >>> exclude(t, 1)  # The root node cannot be excluded
    [1, [2, [2], [3], [4]], [5]]
    """
    filtered_branches = map(lambda y: ____________, branches(t))
    bs = []
    for b in filtered_branches:
        if ____________:
            bs.____(____)
        else:
            bs.append(b)
    return tree(label(t), bs)
```

exclude(y, x)

In Spring 2023, 20% of students got this right

label(b) == x

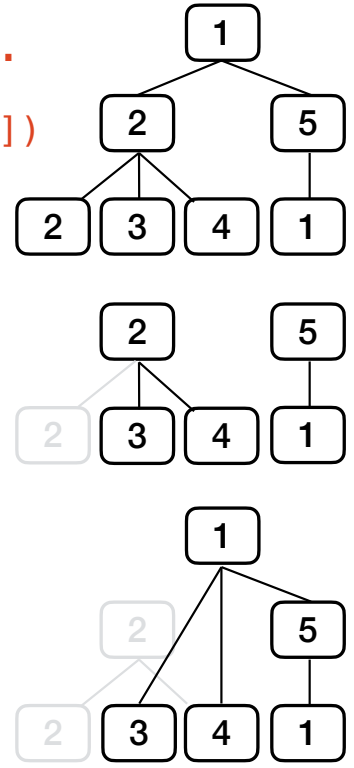
37% of students got this right

30% got it right; 1 of 4 options

extend

branches(b)

24% got it right



Generators

Generators and Generator Functions

```
>>> def plus_minus(x):
...     yield x
...     yield -x

>>> t = plus_minus(3)
>>> next(t)
3
>>> next(t)
-3
>>> t
<generator object plus_minus ...>
```

A *generator function* is a function that **yield**s values instead of **return**ing them

A normal function **return**s once; a *generator function* can **yield** multiple times

A *generator* is an iterator created automatically by calling a *generator function*

When a *generator function* is called, it returns a *generator* that iterates over its yields

(Demo)

Spring 2023 Midterm 2 Question 5(b)

Definition. When parking vehicles in a row, a motorcycle takes up 1 parking spot and a car takes up 2 adjacent parking spots. A string of length n can represent n adjacent parking spots using % for a motorcycle, <> for a car, and . for an empty spot.

For example: '.%%.<><>' (Thanks to the Berkeley Math Circle for introducing this question.)

Implement **park**, a generator function that yields all the ways, represented as strings, that vehicles can be parked in n adjacent parking spots for positive integer n.

```
def park(n):
    """Yield the ways to park cars and motorcycles in n adjacent spots.

    >>> sorted(park(1))
    ['%', '.']
    >>> sorted(park(2))
    ['%%', '%.', '.%', '..', '<>']
    >>> len(list(park(4)))  # some examples: '<><>', '.%%.', '%<>%', '%.<>'
    29
    """
```

Example: Call Expressions

Problem Definition

From Discussion 0:

Imagine you can call only the following three functions:

- f(x): Subtracts one from an integer x

- g(x): Doubles an integer x

- h(x, y): Concatenates the digits of two different positive integers x and y. For example, h(789, 12) evaluates to 78912 and h(12, 789) evaluates to 12789.

Definition: A *small expression* is a call expression that contains only f, g, h, the number 5, and parentheses. All of these can be repeated. For example, h(g(5), f(f(5))) is a small expression that evaluates to 103.

What's the shortest *small expression* you can find that evaluates to 2023?

A Simple Restatement:

You start with 5. You can:

- Subtract 1 from a number

- Double a number

- Glue two numbers together

How do you get to 2023?

5➡10➡20

5➡4➡3➡2

5➡4➡3

A Computational Approach

Try all the small expressions with 4 function calls, then 5 calls, then 6 calls, etc.

```
f(f(f(f(5)))) -> 1
g(f(f(f(5)))) -> 4
f(g(f(f(5)))) -> 5
g(g(f(f(5)))) -> 12
f(f(g(f(5)))) -> 6
g(f(g(f(5)))) -> 14
f(g(g(f(5)))) -> 15
g(g(g(f(5)))) -> 32
f(f(f(g(5)))) -> 7
g(f(f(g(5)))) -> 16
f(g(f(g(5)))) -> 17
g(g(f(g(5)))) -> 36
f(f(g(g(5)))) -> 18
g(f(g(g(5)))) -> 38
f(g(g(g(5)))) -> 39
g(g(g(g(5)))) -> 80
```

```
f(h(f(5),f(5))) -> 43
g(h(f(5),f(5))) -> 88
f(h(f(5),g(5))) -> 409
g(h(f(5),g(5))) -> 820
f(h(g(5),f(5))) -> 103
g(h(g(5),f(5))) -> 208
f(h(g(5),g(5))) -> 1009
g(h(g(5),g(5))) -> 2020
```

```
h(f(5),f(f(5))) -> 43
h(f(5),g(f(5))) -> 48
h(f(5),f(g(5))) -> 49
h(f(5),g(g(5))) -> 420
h(g(5),f(f(5))) -> 103
h(g(5),g(f(5))) -> 108
h(g(5),f(g(5))) -> 109
h(g(5),g(g(5))) -> 1020
h(f(f(5)),f(5)) -> 34
h(f(f(5)),g(5)) -> 310
h(g(f(5)),f(5)) -> 84
h(g(f(5)),g(5)) -> 810
h(f(g(5)),f(5)) -> 94
h(f(g(5)),g(5)) -> 910
h(g(g(5)),f(5)) -> 204
h(g(g(5)),g(5)) -> 2010
```

Reminder: f(x) subtracts 1; g(x) doubles; h(x, y) concatenates