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.

```python
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: exclude(y, x), branches(t))
    bs = []
    for b in filtered_branches:
        if label(b) == x:
            bs.extend(branches(b))
        else:
            bs.append(b)
    return tree(label(t), bs)
```

In Spring 2023, 20% of students got this right.

37% of students got this right.

30% got it right; 1 of 4 options

24% got it right.
Min Practice
Match the description to the code

```python
w = {...}  # a dict with unique keys and values
m = {v: k for k, v in w.items()}
```

Which expression evaluates to?

1. The key that has the smallest value in `w`
   - `min(w.keys(), key=lambda k: w[k])`

2. The value that has the smallest key in `w`
   - `min(w.keys(), key=lambda k: m[k])`

3. The smallest absolute difference between a key and its value
   - `min(w.values(), key=lambda v: w[v])`
   - `min(w.values(), key=lambda v: m[v])`
   - `min(w.keys(), key=lambda k: abs(k - w[k]))`
   - `min(w.keys(), key=lambda k: abs(k - m[k]))`
   - `min(map(lambda k: abs(k - w[k]), w.keys()))`
   - `min(map(lambda k: abs(k - m[k]), w.keys()))`

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 yields values instead of returning them.
A normal function returns 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)
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 $n$.

```python
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?

**(Demo)**

**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 2024?
A Computational Approach

def f(x):
    return x - 1

def g(x):
    return 2 * x

def h(x, y):
    return int(str(x) + str(y))

def smalls(n):
    """Yield all call expressions involving f, g, h, and 5 that have n calls."
    if n == 0:
        yield '5'
    else:
        for operand in smalls(n-1):
            yield 'f(' + operand + ')
            yield 'g(' + operand + ')
        for k in range(n):
            for first in smalls(k):
                for second in smalls(n-k-1):
                    if eval(first) > 0 and eval(second) > 0:
                        yield 'h(' + first + ',', second + ')'

>>> [exp for exp in smalls(7) if eval(exp) == 2024]
['g(h(g(5), g(g(f(f(5))))))']

"""