Exam Prep 5: Iterators and Generators

Students from past semesters wanted more content and structured time to prepare for exams. Exam Prep sections are a way to solidify your understanding of the week's materials. The problems are typically designed to be a bridge between discussion/lab/homework difficulty and exam difficulty.

Reminder: There is nothing to turn in and there is no credit given for attending Exam Prep Sections.

We try to make these problems **exam level** , so you are not expected to be able to solve them coming straight from lecture without additional practice. To get the most out of Exam Prep, we recommend you **try these problems first on your own** before coming to the Exam Prep section, where we will explain how to solve these problems while giving tips and advice for the exam. Do not worry if you struggle with these problems, **it is okay to struggle while learning**.

You can work with anyone you want, including sharing solutions. We just ask you don't spoil the problems for anyone else in the class. Thanks!

You may only put code where there are underscores for the codewriting questions.

You can test your functions on their doctests by clicking the red test tube in the top right corner after clicking Run in 61A Code. Passing the doctests is not necessarily enough to get the problem fully correct. You must fully solve the stated problem.

Q1: Node Printer

Difficulty: ☆☆

Your friend wants to print out all of the values in some trees. Based on your experience in CS 61A, you decide to come up with an unnecessarily complicated solution. You will provide them with a function that takes in a tree and returns a *node-printing function*. When you call a node-printing function, it prints out the label of one node in the tree. Each time you call the function it will print the label of a different node. You may assume that your friend is polite and will not call your function after printing out all of the tree's node labels. You may print the labels in any order, so long as you print the label of each node exactly once.

(Very) optional challenge: See if you can come up with a solution that prints out all of the nodes from one layer before moving on to the next (hint: it still fits within the skeleton code).

Important: The skeleton code is only a suggestion; feel free to add or remove lines as you see fit. Also, it's okay if your code doesn't pass the doctest; if you run the test case with the green arrow and all 8 values are printed exactly once, then your implementation is fine.

```
3       >>> t1 = Tree(1, [Tree(2,
4       ...                    [Tree(5),
5       ...                     Tree(6, [Tree(8)])]),
6       ...                Tree(3),
7       ...                Tree(4, [Tree(7)])])
8       >>> printer = node_printer(t1)
9       >>> for _ in range(8): # NOTE: it's okay to fail this test if all 8 are printed once
10      ...     printer()
11      1
12      2
13      3
14      4
15      5
16      6
17      7
18      8
```

```
        """
        to_explore = [t]

        def step():
            node = to_explore.pop(0)
            print(node.label)
            to_explore.extend(node.branches)

        return step

```

Q2: Fibonacci Generator

Difficulty: ☆☆

Construct the generator function `fib_gen`, which when called returns a generator that yields elements of the Fibonacci sequence in order. **Hint:** consider using the `zip` function.

IMPORTANT: As a warm-up, try solving this problem iteratively without using the template. Then try to find a recursive solution using the template (you may add an extra line or two, but no extra structure is necessary). We recommend running your code in a local interpreter, as sometimes the online interpreter has bugs with recursive generator functions.

```
def fib_gen():
    """
    >>> fg = fib_gen()
    >>> for _ in range(7):
    ...     print(next(fg))
    0
    1
    1
    2
    3
    5
    8
    """
    yield from [0, 1]
    a = fib_gen()
    next(a)
    for x, y in zip(a, fib_gen()):
        yield x + y

```

Q3: Partition Generator

Difficulty: ☆☆☆

Construct the generator function `partition_gen`, which takes in a number `n` and returns an *n-partition iterator*. An *n-partition iterator* yields partitions of `n`, where a partition of `n` is a list of integers whose sum is `n`. The iterator should only return unique partitions; the order of numbers within a partition and the order in which partitions are returned does not matter

Important: The skeleton code is only a suggestion; feel free to add or remove lines as you see fit.

```
def partition_gen(n):
    """
    >>> for partition in partition_gen(4): # note: order doesn't matter
    ...     print(partition)
    [4]
    [3, 1]
    [2, 2]
    [2, 1, 1]
    [1, 1, 1, 1]
    """

    def yield_helper(j, k):
        if j == 0:
            yield []
        elif k > 0 and j > 0:
            for small_part in yield_helper(j - k, k):
                yield [k] + small_part
            yield from yield_helper(j, k - 1)

```

Q4: Apply That Again

This problem was taken from the Spring 2015 final exam (https://inst.eecs.berkeley.edu//~cs61a/sp15/assets/pdfs/61a-sp15-final.pdf#page=7).

NOTE: We will introduce this problem in section and give you time to work on it then. If you'd like to solve it then, don't look ahead!

Difficulty: ☆

Implement `amplify`, a generator function that takes a one-argument function `f` and a starting value `x`. The element at index *k* that it yields (starting at 0) is the result of applying `f` *k* times to `x`. It terminates whenever the next value it would yield is a false value, such as `0`, `""`, `[]`, `False`, etc.

```
def amplify(f, x):
    """Yield the longest sequence x, f(x), f(f(x)), ... that are all true values

    >>> list(amplify(lambda s: s[1:], 'boxes'))
    ['boxes', 'oxes', 'xes', 'es', 's']
    >>> list(amplify(lambda x: x//2-1, 14))
    [14, 6, 2]
    """
    "*** YOUR CODE HERE ***"

    while x:
```

Q5: Extra Practice

Difficulty: >☆☆☆

Note: several problems overlap with those on this sheet, but some problems are unique or are more difficult variants of the problems above.

Fall 2020 Exam Prep on Generators
(https://drive.google.com/file/d/1F_zxvxVJmVoRGH6trSVxhyXKO7sZhAL4/view?usp=sharing)