CS 61A
Fall 2023

Iterators, Generators

Discussion 6: October 4, 2023

Getting Started

Someone from your group should join Discord.

To get help from a TA, send a message to the `discuss-queue` channel with the @discuss tag and your discussion group number.

If you have only 1 or 2 people in your group, you can join the other group in the room with you.

If it's too hot in your room, you're welcome to move outside. If your whole group hasn't arrived yet, leave a note to let latecomers know where you went.

Q1: Ice Breaker

Say your name and share a favorite place on the Berkeley campus or surrounding city that you've discovered. Try to pick a place that others might not have been yet. (But if the room you're in now is your favorite place on campus, that's ok too.)

McCone Hall has a nice view from the 5th floor balcony.

Generators

Q2: Big Fib

This generator function yields all of the Fibonacci numbers.

```
def gen_fib():
    n, add = 0, 1
    while True:
        yield n
        n, add = n + add, n
```

As a warm-up, explain the following expression to each other so that everyone understands how it works. (It creates a list of the first 10 Fibonacci numbers.)

```
(lambda t: [next(t) for _ in range(10)])(gen_fib())
```

Then, complete the expression below by writing only names and parentheses in the blanks so that it evaluates to the smallest Fibonacci number that is larger than 2023.

Talk with each other about what built-in functions might be helpful. Try to figure out the answer without using Python. Only run the code when your group agrees that the answer is right. This is not the time for guess-and-check.

```
def gen_fib():
    n, add = 0, 1
    while True:
        yield n
        n, add = n + add, n

____lambda n: n > 2023, ____
```

Discussion Time. When your group finds a solution, discuss what each function and each set of parentheses is doing. If you're not sure, ask the course staff. Then, evaluate the expression and post its value in your group's Discord channel's text chat.

One solution has the form: `next(____(lambda n: n > 2020, ____))` where the first blank uses a built-in function to create an iterator and the second blank creates an iterator over Fibonacci numbers.

Surprise! There's no hint here. If you're still stuck, it's time to get help from a TA by sending a message to the `discuss-queue` channel with the @discuss tag and your discussion group number.

Q3: Something Different

Implement `differences`, a generator function that takes `t`, a non-empty iterator over numbers. It yields the differences between each pair of adjacent values from `t`. If `t` iterates over a positive finite number of values `n`, then `differences` should yield `n-1` times.

```
def differences(t):
    """Yield the differences between adjacent values from iterator t.

    >>> list(differences(iter([5, 2, -100, 103])))
    [-3, -102, 203]
    >>> next(differences(iter([39, 100])))
    61
    """
    "*** YOUR CODE HERE ***"
```

Add to the following implementation by initializing and updating `previous_x` so that it is always bound to the value of `t` that came before `x`.

```
for x in t:
    yield x - previous_x
```

Presentation Time. Work together to explain why `differences` will always yield `n-1` times for an iterator `t` over `n` values. Pick someone who didn't present to the course staff last week to present your group's answer, and then send a message to the discuss-queue channel with the @discuss tag, your discussion group number, and the message "We beg to differ!" and a member of the course staff will join your voice channel to hear your description and give feedback.

Q4: Partitions

Tree-recursive generator functions have a similar structure to regular tree-recursive functions. They are useful for iterating over all possibilities. Instead of building a list of results and returning it, just `yield` each result.

You'll need to identify a *recursive decomposition*: how to express the answer in terms of recursive calls that are simpler. Ask yourself what will be yielded by a recursive call, then how to use those results.

Definition. For positive integers `n` and `m`, a *partition* of `n` using parts up to size `m` is an addition expression of positive integers up to `m` in non-decreasing order that sums to `n`.

Implement `partition_gen`, a generator functon that takes positive `n` and `m`. It yields the partitions of `n` using parts up to size `m` as strings.

Reminder: For the `partitions` function we studied in lecture (video), the recursive decomposition was to enumerate all ways of partitioning `n` using at least one `m` and then to enumerate all ways with no `m` (only `m-1` and lower).

```
def partition_gen(n, m):
    """Yield the partitions of n using parts up to size m.

    >>> for partition in sorted(partition_gen(6, 4)):
    ...     print(partition)
    1 + 1 + 1 + 1 + 1 + 1
    1 + 1 + 1 + 1 + 2
    1 + 1 + 1 + 3
    1 + 1 + 2 + 2
    1 + 1 + 4
    1 + 2 + 3
    2 + 2 + 2
    2 + 4
    3 + 3
    """
    assert n > 0 and m > 0
    if n == m:
        yield ____
    if n - m > 0:
        "*** YOUR CODE HERE ***"



    if m > 1:
        "*** YOUR CODE HERE ***"
```

Presentation Time. If you have time, work together to explain why this implementation of `partition_gen` does not include base cases for `n < 0`, `n == 0`, or `m == 0` even though the original implementation of `partitions` from lecture (video) had all three. Pick someone who didn't present to the course staff this week or last week to present your group's answer, and then send a message to the discuss-queue channel with the @discuss tag, your discussion group number, and the message "We're positive!" and a member of the course staff will join your voice channel to hear your description and give feedback.

Document the Occasion

Please all fill out the attendance form (one submission per person per week).

If you finish early, you could all go check out someone's favorite place together.