CS 61A Fall 2023

Iterators, Generators

Discussion 6: February 28, 2024

Pick someone in your group to join Discord. It's fine if multiple people join, but one is enough.

Now switch to Pensieve:

- **Everyone**: Go to discuss.pensieve.co and log in with your @berkeley.edu email, then enter your group number. (Your group number is the number of your Discord channel.)

Once you're on Pensieve, you don't need to return to this page; Pensieve has all the same content (but more features). If for some reason Penseive doesn't work, return to this page and continue with the discussion.

Post in the `#help` channel on Discord if you have trouble.

Getting Started

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

A *generator* is an *iterator* that is returned by calling a *generator function*, which is a function that contains `yield` statements instead of `return` statements. The ways to use an *iterator* are to call `next` on it or to use it as an iterable (for example, in a `for` statement).

Q1: 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
```

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 2024.

Talk with each other about what built-in functions might be helpful, such as `map`, `filter`, `list`, `any`, `all`, etc. (Click on these function names to view their documentation.) 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

next(filter(lambda n: n > 2024, gen_fib()))
```

One solution has the form: `next(____(lambda n: n > 2024, ____))` where the first blank uses a built-in function to create an iterator over just large numbers and the second blank creates an iterator over all Fibonacci numbers.

Surprise! There's no hint here. If you're still stuck, it's time to get help from the course staff.

Q2: 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
    """
    last_x = next(t)
    for x in t:
        yield x - last_x
        last_x = x
```

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.

Intermission

We're lazy (like an iterator) and used ChatGPT to generate a generator joke…

Because it was skilled at knowing when to "return" to the recipe and when to "yield" to improvisation!

Q3: 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 str(n)
    if n - m > 0:
        for p in partition_gen(n - m, m):
            yield p + ' + ' + str(m)
    if m > 1:
        yield from partition_gen(n, m-1)
```

Yield a partition with just one element, `n`. Make sure you yield a string.

The first recursive case uses at least one `m`, and so you will need to yield a string that starts with `p` but also includes `m`. The second recursive case only uses parts up to size `m-1`. (You can implement the second case in one line using

`yield from`.)

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).