ITERATORS, GENERATORS, AND STREAMS

COMPUTER SCIENCE MENTORS CS 61A

April 22 to April 24, 2019

1 Iterators and Generators

1. What Would Python Display?

```
class SkipMachine:
    skip = 1
    def __init__(self, n=2):
        self.skip = n + SkipMachine.skip

    def generate(self):
        current = SkipMachine.skip
        while True:
            yield current
            current += self.skip
            SkipMachine.skip += 1

p = SkipMachine()
twos = p.generate()
SkipMachine.skip += 1
twos2 = p.generate()
threes = SkipMachine(3).generate()
```

   (a) `next(twos)`

   Solution: 2

   (b) `next(threes)`

Solution: 2

(c) **next**(twos)

Solution: 5

(d) **next**(twos)

Solution: 8

(e) **next**(threes)

Solution: 7

(f) **next**(twos2)

Solution: 5

2. What does the following code block output?

```
def foo():
    a = 0
    if a < 10:
        print("Hello")
        yield a
        print("World")

for i in foo():
    print(i)
```

Solution:

```
Hello
0
World
```

3. How can we modify `foo` so that it satisfies the following doctests?

```
>>> a = list(foo())
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Solution: Change the **`if`** to a **`while`** statement, and make sure to increment `a`. This looks like:

```
def foo():
    a = 0
    while a < 10:
        a += 1
        yield a
```

4. Define `filter_gen`, a generator that takes in iterable `s` and one-argument function `f` and yields every value from `s` for which `f` returns `True`

```
def filter_gen(s, f):
    """
    >>> list(filter_gen([1, 2, 3, 4, 5],
                                lambda x: x % 2 == 0))
    [2, 4]
    >>> list(filter_gen((1, 2, 3, 4, 5), lambda x: x < 3))
    [1, 2]
    """
```

Solution:

```
        for x in s:
            if f(x):
                yield x
```

CS61A Spring 2019: Catherine Cang and Alex Stennet, with
Albert Xu, Albert Yu, Alex Yang, Allan Yu, Amy Hung, Chris Niu, Danelle Nachum, Evan Corriere, Jack Coyle, Jade Singh, Jason Chang, Jemmy Zhou, Kevin Wang, Lucas Pan, Mary Xu, Michelle Cheung, Olivia Shiah, Patricia Ouyang, Richard Roggenkemper, Samy Cherfaoui, Shreya Sahoo, Varun Jhunjhunwalla

5. Define `tree_sequence`, a generator that iterates through a tree by first yielding the root value and then yielding the values from each branch.

```
def tree_sequence(t):
    """
    >>> t = tree(1, [tree(2, [tree(5)]), tree(3, [tree(4)])])
    >>> print(list(tree_sequence(t)))
    [1, 2, 5, 3, 4]
    """
```

Solution:

```
        yield label(t)
        for branch in branches(t):
                for value in tree_sequence(branch):
                        yield value
```

Alternate solution:

```
        yield label(t)
        for branch in branches(t):
                yield from tree_sequence(branch)
```

2 Streams

1. (a) What are the advantages or disadvantages of using a stream over a linked list?

Solution: Lazy evaluation. We only evaluate up to what we need.

(b) What's the maximum size of a stream?

Solution: Infinite

(c) What's stored in `first` and `rest`? What are their types?

Solution: `first` is a value, `rest` is another stream (either a method to calculate it, or an already calculated stream). In the case of Scheme, this is called a promise.

(d) When is the next element actually calculated?

Solution: Only when it's requested (and hasn't already been calculated)

CS61A Spring 2019: Catherine Cang and Alex Stennet, with
Albert Xu, Albert Yu, Alex Yang, Allan Yu, Amy Hung, Chris Niu, Danelle Nachum, Evan Corriere, Jack Coyle, Jade Singh, Jason Chang, Jemmy Zhou, Kevin Wang, Lucas Pan, Mary Xu, Michelle Cheung, Olivia Shiah, Patricia Ouyang, Richard Roggenkemper, Samy Cherfaoui, Shreya Sahoo, Varun Jhunjhunwalla

2. What Would Scheme Display?

(a) `scm> (define (foo x)(+ x 10))`

Solution: `foo`

(b) `scm> (define bar (cons-stream (foo 1)(cons-stream (foo 2)bar )))`

Solution: `bar`

(c) `scm> (car bar)`

Solution: `11`

(d) `scm> (cdr bar)`

Solution: `#[promise (not forced)]`

(e) `scm> (define (foo x)(+ x 1))`

Solution: `foo`

(f) `scm> (cdr-stream bar)`

Solution: `(3 . #[promise (not forced)])`

(g) `scm> (define (foo x)(+ x 5))`

Solution: `foo`

(h) `scm> (car bar)`

Solution: `11`

(i) `scm> (cdr-stream bar)`

Solution: `(3 . #[promise (not forced)])`

(j) `scm> (cdr bar)`

Solution: `#[promise (forced)]`

3 Code Writing for Streams

1. Implement `double-naturals`, which is a returns a stream that evaluates to the sequence 1, 1, 2, 2, 3, 3, etc.

```
(define (double-naturals)
    (double-naturals-helper 1 #f)
)
```

Solution:

```
(define (double_naturals_helper first go-next)
    (if go-next
        (cons-stream first (double_naturals_helper (+ 1
           first) #f))
        (cons-stream first (double_naturals_helper first #t
           ))))
```

2. Implement `interleave`, which returns a stream that alternates between the values in stream1 and stream2. Assume that the streams are infinitely long.

Solution:

```
(define (interleave stream1 stream2)
    (cons-stream
        (car stream1)
        (interleave stream2 (cdr-stream stream1))
    )
)

(define (interleave stream1 stream2)
    (cons-stream (car stream1)
        (cons-stream (car stream2)
            (interleave (cdr-stream stream1) (cdr-stream
               stream2))))
)
```