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)`

   (b) `next(threes)`

   (c) `next(twos)`

(d) `next(twos)`

(e) `next(threes)`

(f) `next(twos2)`

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)
```

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]
```

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]
    """
```

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]
    """
```

2 Streams

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

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

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

   (d) When is the next element actually calculated?

2. What Would Scheme Display?

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

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

(c)
```
scm> (car bar)
```

(d)
```
scm> (cdr bar)
```

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

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

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

(h)
```
scm> (car bar)
```

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

(j)
```
scm> (cdr bar)
```

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)
)
(define (double-naturals-helper first go-next)
```

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

```
(define (interleave stream1 stream2)
```