CS 61A Challenge Problems: Iterators, Iterables, and Generators (and Streams)

Solutions at https://alextseng.net/teaching/cs61a/

Alex Tseng

1 Iterators and Iterables

(a) Complete the following class `PrimeIterator` so that it correctly iterates through the prime numbers in the interval `[start, end)` one by one. You may assume that the function `is_prime` is already written for you.

```
class PrimeIterator:
    def __init__(self, start, end):
        self.start = start
        self.end = end

    def __next__(self):
```

(b) If we wanted to use a `PrimeIterator` instance in a `for` loop or in the function `list`, then what method do we need to add? Write this method.

(c) `p = PrimeIterator(2, 15)`. Eventually what happens if we keep calling `next` on `p`?

(d) Complete the `PrimeIterable` class that is an iterable and has the same functionality as `PrimeIterator`. This means that we can use it in a `for` loop, call `list` on it, etc.

```
class PrimeIterable:
    def __init__(self, start, end):
        self.start = start
        self.end = end


    def __getitem__(self, i):
        # Assume this is already implemented for you
```

(e) `q = PrimeIterable(2, 15)`. What happens if keep calling `next` on `q`? What is the result of calling `list(q)`?

(f) Implement the class `Vowels` that takes in a word and allows you to step through each of the vowels in the word in order. `Vowels` *is both an iterator and an iterable* that also supports indexing.

```
import re
def get_vowels(word):
    # A bit of RegEx magic to isolate all the vowels of a word in order
    return re.sub(r'[^aeiou]', '', word)

class Vowels:
```

(g) What is the result of the following code?

```
list(Vowels("facetious"))

Vowels("aardvark")[3]

next(Vowels("sciatic"))
```

2 Generators

(a) Write the generator function `randoms` that can generate `num` random integers in the interval `[low, high)`.

(b) Write a generator expression that gives the same result as the function.

3 Streams

What is the 4th element in this stream? Assume 1-indexing.

```
(define (mystery foo)
  (let ((bar (+ (* foo 3) 1)))
    (cons-stream bar
                 (mystery bar))))

(mystery 3)
```