Generator Composition in ES6

Problem

Generators cannot abstract over sync IO.

Two Ways to Iterate

```
var nums = function*() {
  yield 1;
  yield 2;
}
```

Iterable

```
function Iterable(generatorFunction) {
  this[@@iterator] =
generatorFunction;
};
```

Creating Iterables

```
let nums = new Iterable(function*() {
  yield 1;
  yield 2;
  yield 3;
});
```

Iterating Iterables

```
for(let x of nums()) {
   console.log(x);
}
```

...becomes...

```
let iterator = nums()[@@iterator],
   pair;

while(!(pair = iterator.next()).done) {
   console.log(pair.value);
}
```

IO Streams as Iterables

```
function getLines(fileName) {
   return new Iterable(function*() {
      let reader = new SyncReader(fileName);
      try {
         while(!reader.eof) {
            yield reader.readLine();
         }
      }
      finally {
         reader.close();
      }
   })
};
```

Iterable Composition

```
Iterable.prototype.map =
function(projection) {
     let self = this;
    return new Iterable(function*() {
        for(let x of self) {
            yield projection(x);
        }
    });
}
```

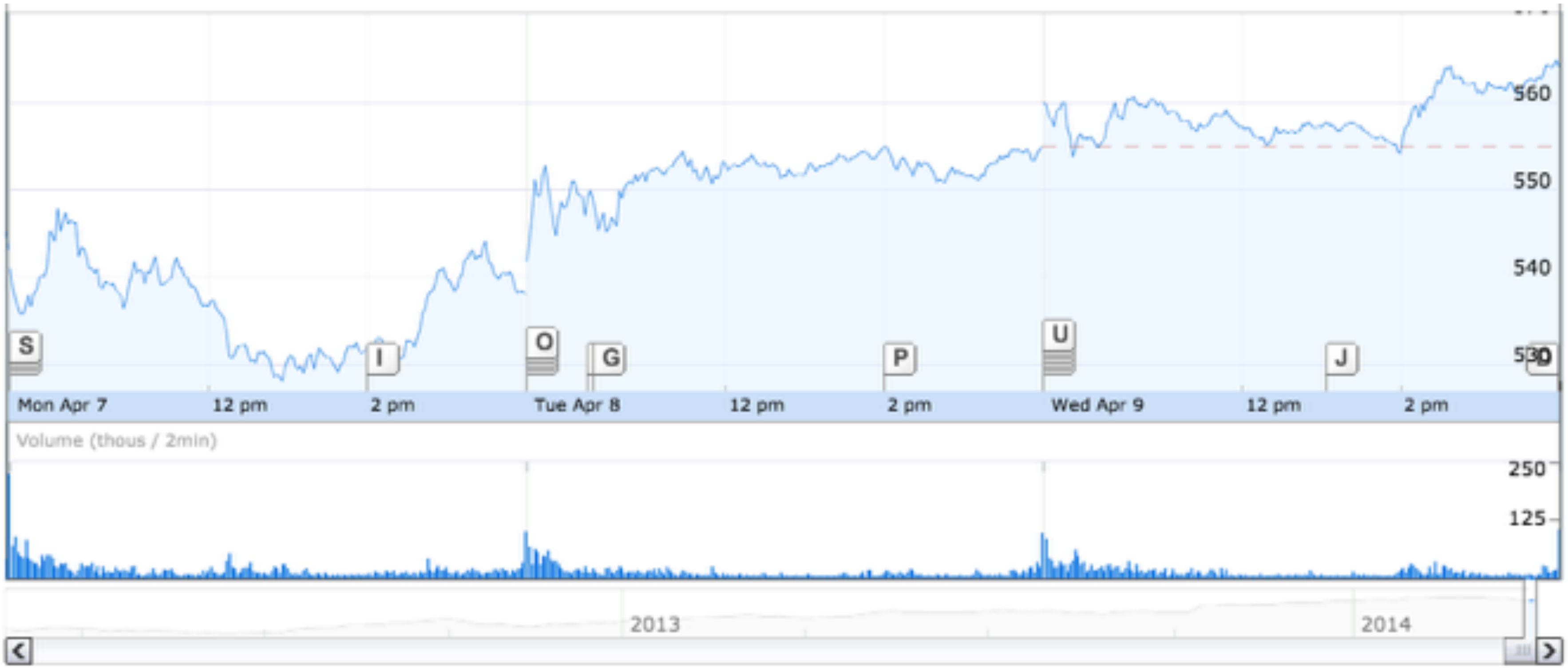
Iterable Composition (cont)

```
Iterable.prototype.takeWhile = function(predicate) {
   let self = this;
   return new Iterable(function*() {
      for(let x of self) {
         if (!predicate(x)) {
            break;
         }
         yield x;
      }
   });
}
```

Historical Stock Ticker Visualizer

GOOG

Symbol filter



Date filter

Created Filtered Iterable

```
// massiveFileOfStockTickerHistory.txt
// {date: 7347832748932, symbol: “BAC”, price: 16.62 }\n
// {date: 7347832748939, symbol: “CTI”, price: 3.48 }\n

let records =
   getLines(“massiveFileOfStockTickerHistory.txt”).
      map(line => JSON.parse(line)).
      filter(({symbol}) => symbol === selectedSymbol).
      skipWhile(({date}) => date > dateFrom).
      takeWhile(({date}) => date < dateTo);


render(records);
```

Lazy evaluation.
Nothing has happened

Render Iterates Data

```
function renderGraph(stocks) {
  // large file is opened.
  for(let stock of stocks) {
    // rendering
  }
}
```

Is file handle closed?

takeWhile Leaks!

```
Iterable.prototype.takeWhile =
    function(predicate) {
        return new
Iterable(function*() {
            for(let x of this) {
                if (!predicate(x)) {
                    break;
                }
                yield x;
            }
        });
    };
```

Suspends

```
function getLines(fileName) {
    return new Iterable(function*()
        let reader =
            new SyncReader(fileName)
        try {
            while(!reader.eof) {
              yield reader.readLine()
            }
        }
        finally {
            reader.close();
        }
    })
};
```

Never executes!

Can not Safely Page Generators!

- Leak exposed by any operator that short-circuits…
  - take
  - takeWhile
- Must iterate to completion to avoid leak
- for…of useless over Generators that abstract over expensive resources

Proposal

- for…of always assumes generator creation
- for…of always terminates generator function

Rationale

- For…of creates iterator
- User-land cannot invoke iterator.return(), because iterator transparently created by for…of
- “If you built it, you bought it.”