HOW TO SURVIVE AND HAVE FUN WITHOUT FOR LOOP

I CAN’T REMEMBER WHEN WAS THE LAST TIME I WRITE FOR LOOP ON MY WORK

I MEAN THIS FOR LOOP

```
for (int i = 0; i < array.length; i++) {
        ...
}
```

HOW?

LET USE WHILE LOOP!

```
int i = 0;
while (i < array.length) {
        ...
        i++;
}
```

THANK YOU!

NO...

RECURSIVE?

```
void recurseMe(int i) {
        if (i < array.length) {
        ...
                return recurseMe(i++);
        }
}
```

THAT WORKS, BUT NOT SO FUN...

Nuttanart Pornprasitsakul

(Tap)

@visibletrap

2 years with Java

3 years with Ruby and little bit of Javascript

PLEASE INTERRUPT ME

WHAT'S YOUR MOST FAMILIAR LANGUAGE?

JAVA?

C#?

C++?

C?

PHP?

JAVASCRIPT?

OBJECTIVE-C?

OTHERS?

I’D LIKE TO INVITE YOU TO TRY

REPLACING YOUR FOR LOOP WITH HIGHER-ORDER FUNCTION

~ Lambda, Closure

DEFINITION

From Wikipedia:

a higher-order function is a function that does at least one of the following:

- *takes one or more functions as an input*
- *outputs a function*

IN ACTION

VIDEO API

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        likeCount:          100,
        dislikeCount:          10
        },
        {
        id:          'aJjYLc1gjTE',
        likeCount:          300,
        dislikeCount:          50
        }
];
```

LIKE - DISLIKE

```
var likeDiffVideos = [];
for (var i = 0; i < videos.length; i++) {
        var videoWithDiff = {
        id: videos[i].id,
        likeCount: videos[i].likeCount,
        dislikeCount: videos[i].dislikeCount,
        diffCount: videos[i].likeCount - videos[i].dislikeCount
        };
        likeDiffVideos.push(videoWithDiff);
}
console.log(likeDiffVideos);
```

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        likeCount:          100,
        dislikeCount:          10
        },
        {
        id:          'aJjYLc1gjTE',
        likeCount:          300,
        dislikeCount:          50
        }
];
```

RESULT

```
[ { id: 'BE8-8f6lxWo',
          likeCount: 100,
          dislikeCount: 10,
          diffCount: 90 },
  { id: 'aJjYLc1gjTE',
          likeCount: 300,
          dislikeCount: 50,
          diffCount: 250 } ]
```

MAP

```
var likeDiffVideos = videos.map(function(video) {
        return {
        id: video.id,
        likeCount: video.likeCount,
        dislikeCount: video.dislikeCount,
        diffCount: video.likeCount - video.dislikeCount
        };
});
console.log(likeDiffVideos);
```

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        likeCount:          100,
        dislikeCount:          10
        },
        {
        id:          'aJjYLc1gjTE',
        likeCount:          300,
        dislikeCount:          50
        }
];
```

VIDEO API

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        publishedAt:          '2014-10-01T22:45:24.000Z'
        },
        {
        id:          'aJjYLc1gjTE',
        publishedAt:          '2014-09-06T02:35:12.000Z'
        }
];
```

PUBLISHED AT >

```
var thisMonthVideos = [];
for (var i = 0; i < videos.length; i++) {
        if (videos[i].publishedAt > '2014-10-01T00:00:00.000Z') {
        thisMonthVideos.push(videos[i]);
        }
}
console.log(thisMonthVideos);
```

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        publishedAt:          '2014-10-01T22:45:24.000Z'
        },
        {
        id:          'aJjYLc1gjTE',
        publishedAt:          '2014-09-06T02:35:12.000Z'
        }
];
```

RESULT

```
[ { id: 'BE8-8f6lxWo', publishedAt: '2014-10-01T22:45:24.000Z' } ]
```

FILTER

```
var thisMonthVideos = videos.filter(function(video) {
        return video.publishedAt > '2014-10-01T00:00:00.000Z';
});
console.log(thisMonthVideos);
```

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        publishedAt:          '2014-10-01T22:45:24.000Z'
        },
        {
        id:          'aJjYLc1gjTE',
        publishedAt:          '2014-09-06T02:35:12.000Z'
        }
];
```

VIDEO API

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        diffCount:          90
        },
        {
        id:          'aJjYLc1gjTE',
        diffCount:          250
        }
];
```

SUM LIKE DIFFS

```
var sumDiff = 0;
for (var i = 0; i < videos.length; i++) {
        sumDiff += videos[i].diffCount;
}
console.log(sumDiff);
```

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        diffCount:          90
        },
        {
        id:          'aJjYLc1gjTE',
        diffCount:          250
        }
];
```

RESULT

```
340
```

REDUCE

```
var sumDiff = videos.reduce(function(immediateResult, video) {
        return immediateResult + video.diffCount;
}, 0);
console.log(sumDiff);
```

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        diffCount:          90
        },
        {
        id:          'aJjYLc1gjTE',
        diffCount:          250
        }
];
```

PUT THEM ALL TOGETHER

```
var videos = [
        {
        id:          'BE8-8f6lxWo',
        publishedAt:          '2014-10-01T22:45:24.000Z',
        likeCount:          100,
                dislikeCount: 10
        },
        {
        id:          'aJjYLc1gjTE',
        publishedAt:          '2014-09-06T02:35:12.000Z',
        likeCount:          300,
        dislikeCount:          50
        }
];
```

```
var sumDiff = 0;
for (var i = 0; i < videos.length; i++) {
        if (videos[i].publishedAt > '2014-10-01T00:00:00.000Z') {
           sumDiff += videos[i].likeCount - videos[i].dislikeCount;
        }
}
console.log(sumDiff);
```

```
var sumDiff = videos.filter(function(video) {
        return video.publishedAt > '2014-10-01T00:00:00.000Z';
}).map(function(video) {
        return { diffCount: video.likeCount - video.dislikeCount };
}).reduce(function(immediateResult, video) {
        return immediateResult + video.diffCount;
}, 0);
console.log(sumDiff);
```

```
var sumDiff = 0;
for (var i = 0; i < videos.length; i++) {
        if (videos[i].publishedAt > '2014-10-01T00:00:00.000Z') {
           sumDiff += videos[i].likeCount - videos[i].dislikeCount;
        }
}
console.log(sumDiff);
```

```
var fromLastMonth = function(video) {
        return video.publishedAt > '2014-10-01T00:00:00.000Z';
};
var calDiffCount = function(video) {
        return { diffCount: video.likeCount - video.dislikeCount };
};
var sumDiffCount = function(result, video) {
        return result + video.diffCount;
};
var sumDiff = videos.filter(fromLastMonth)
        .map(calDiffCount)
        .reduce(sumDiffCount,        0);
console.log(sumDiff);
```

BASIC INTERNAL IMPLEMENTATION

FOR LOOP!

```
function mapper1(coll, f) {
        var newColl = [];
        for (var i = 0; i < coll.length; i++) {
        newColl[i] = f(coll[i]);
        }
        return newColl;
}
```

```
var numbers = [1, 2, 3, 4, 5];
var increment = function(i) {
        return i + 1;
};
var incrementedNumbers1 = mapper1(numbers, increment);
console.log(incrementedNumbers1);
// => [ 2, 3, 4, 5, 6 ]
```

RECURSIVE

```
function mapper2(coll, f) {
        if (coll.length === 0) {
                return [];
        } else {
                var head = coll[0];
                var tail = coll.slice(1,coll.length);
                return [f(head)].concat(mapper2(tail,f));
        }
}
```

```
var numbers = [1, 2, 3, 4, 5];
var increment = function(i) {
        return i + 1;
};
var incrementedNumbers2 = mapper2(numbers, increment);
console.log(incrementedNumbers2);
// => [ 2, 3, 4, 5, 6 ]
```

PROS

AVOID BORING, DUPLICATED FOR LOOP

```
for (int i = 0; i < array.length; i++) {
        ...
}
```

COMPOSIBILITY

```
var sumDiff = 0;
for (var i = 0; i < videos.length; i++) {
        if (videos[i].publishedAt > '2014-10-01T00:00:00.000Z') {
           sumDiff += videos[i].likeCount - videos[i].dislikeCount;
        }
}
console.log(sumDiff);
```

```
var fromLastMonth = function(video) {
        return video.publishedAt > '2014-10-01T00:00:00.000Z';
};
var calDiffCount = function(video) {
        return { diffCount: video.likeCount - video.dislikeCount };
};
var sumDiffCount = function(result, video) {
        return result + video.diffCount;
};
var sumDiff = videos.filter(fromLastMonth)
        .map(calDiffCount)
        .reduce(sumDiffCount,        0);
console.log(sumDiff);
```

```
var sumDiff = 0;
for (var i = 0; i < videos.length; i++) {
        if (videos[i].publishedAt > '2014-10-01T00:00:00.000Z' &&
        videos[i].publishedAt <          '2014-11-01T00:00:00.000Z') {
           sumDiff += videos[i].likeCount - videos[i].dislikeCount;
        }
}
console.log(sumDiff);
```

```
var fromLastMonth = function(video) {
        return video.publishedAt > '2014-10-01T00:00:00.000Z';
};
var beforeThisMonth = function(video) {
        return videos[i].publishedAt < '2014-11-01T00:00:00.000Z';
};
var calDiffCount = function(video) {
        return { diffCount: video.likeCount - video.dislikeCount };
};
var sumDiffCount = function(result, video) {
        return result + video.diffCount;
};
var sumDiff = videos.filter(fromLastMonth)
        .filter(beforeThisMonth)
        .map(calDiffCount)
        .reduce(sumDiffCount,        0);
console.log(sumDiff);
```

PARALLELISM

Required isolation in each function

E.g. MapReduce

CONS

- Not as fast
- Familiarity

MORE FUN

MARTIN FOWLER'S COLLECTION PIPELINE

collect

*Alternative name for **map**, from Smalltalk. Java 8 uses "collect" for a completely different purpose: a terminal that collects elements from a stream into a collection.*

see map

concat

Concatenates collections into a single collection

more…

difference

Remove the contents of the supplied list from the pipeline

more…

distinct

Removes duplicate elements

more…

drop

*A form of **slice** that returns all but the first n elements*

see slice

filter

Runs a boolean function on each element and only puts those that pass into the output.

more…

flat-map

Map a function over a collection and flatten the result by one-level

more…

flatten

Removes nesting from a collection

more…

fold

*Alternative name for **reduce** Sometimes seen as foldl (fold-left) and foldr (fold-right).*

see reduce

group-by

Runs a function on each element and groups the elements by the result.

more…

inject

*Alternative name for **reduce**, from Smalltalk's `inject:into:` selector.*

see reduce

intersection

Retains elements that are also in the supplied collection

more…

map

Applies given function to each element of input and puts result in output

more…

mapcat

*Alternative name for **flat-map***

see flat-map

reduce

Uses the supplied function to combine the input elements, often to a single output value

more…

reject

*Inverse of **filter**, returning elements that do not match the predicate.*

see filter

select

*Alternative name for **filter**.*

see filter

slice

Return a sub-sequence of the list between the given first and last positions.

more…

sort

Output is sorted copy of input based on supplied comparator

more…

take

*A form of **slice** that returns the first n elements*

see slice

union

returns elements in this or the supplied collection, removing duplicates

more…

CHECK YOUR FAVORITE LANGUAGE DOCUMENTATION OR LIBRARY

REPLICATE A SEQUENCE

```
(= (__ [1 2 3] 2) '(1 1 2 2 3 3))

(= (__ [:a :b] 4) '(:a :a :a :a :b :b :b :b))

(= (__ [4 5 6] 1) '(4 5 6))

(= (__ [[1 2] [3 4]] 2) '([1 2] [1 2] [3 4] [3 4]))

(= (__ [44 33] 2) [44 44 33 33])
```

LITTLE BIT OF CLOJURE

```
functionName(arg1, args2, ...)
```

```
(function-name arg1 args2 ...)
```

REPLICATE A SEQUENCE

```
(= (__ [1 2 3] 2) '(1 1 2 2 3 3))

(= (__ [:a :b] 4) '(:a :a :a :a :b :b :b :b))

(= (__ [4 5 6] 1) '(4 5 6))

(= (__ [[1 2] [3 4]] 2) '([1 2] [1 2] [3 4] [3 4]))

(= (__ [44 33] 2) [44 44 33 33])
```

```
(= (__ [1 2 3] 2) '(1 1 2 2 3 3))

(= (__ [:a :b] 4) '(:a :a :a :a :b :b :b :b))

(= (__ [4 5 6] 1) '(4 5 6))

(= (__ [[1 2] [3 4]] 2) '([1 2] [1 2] [3 4] [3 4]))

(= (__ [44 33] 2) [44 44 33 33])
```

```
function(seq, n) {
        var result = [];
        for (var i = 0; i < seq.length; i++) {
                for (var j = 0; j < n; j++) {
        result.push(seq[i]);
        }
        }
        return result;
}
```

```clojure
(fn [seqn n] (mapcat (partial repeat n) seqn))
```

```javascript
function(seq, n) {
        var result = [];
        for (var i = 0; i < seq.length; i++) {
                for (var j = 0; j < n; j++) {
        result.push(seq[i]);
        }
        }
        return result;
}
```

```clojure
(fn [seqn n] (mapcat (partial repeat n) seqn))
```

```clojure
(mapcat function collection)
;; collection.mapcat(function)
```

```clojure
(repeat 5 "x")
;=> ("x" "x" "x" "x" "x")
```

```clojure
(partial repeat 2)
;=> #<core$partial$fn__4228 clojure.core$partial$fn__4228@79c853cf>
```

```clojure
(map (partial repeat 2) [1 2 3])
;=> ((1 1) (2 2) (3 3))
```

```clojure
(mapcat (partial repeat 2) [1 2 3])
;=> (1 1 2 2 3 3)
```

```
(= (__ [1 2 3] 2) '(1 1 2 2 3 3))

(= (__ [:a :b] 4) '(:a :a :a :a :b :b :b :b))

(= (__ [4 5 6] 1) '(4 5 6))

(= (__ [[1 2] [3 4]] 2) '([1 2] [1 2] [3 4] [3 4]))

(= (__ [44 33] 2) [44 44 33 33])
```

```
(mapcat (partial repeat 4) [:a :b])
;=> (:a :a :a :a :b :b :b :b)
```

DROP EVERY NTH ITEM

```
(= (__ [1 2 3 4 5 6 7 8] 3) [1 2 4 5 7 8])

(= (__ [:a :b :c :d :e :f] 2) [:a :c :e])

(= (__ [1 2 3 4 5 6] 4) [1 2 3 5 6])
```

```
function(coll, n) {
        var result = [];
        for (var i = 0; i < coll.length; i++) {
                if ((i+1) % 3 !== 0) {
        result.push(coll[i]);
        }
        }
        return result;
}
```

```
(fn [coll n] (mapcat (partial take (dec n)) (partition-all n coll)))
```

```clojure
(fn [coll n] (mapcat (partial take (dec n)) (partition-all n coll)))
```

```clojure
(dec 3)
;=> 2
```

```clojure
(take 2 [1 2 3])
;=> (1 2)
```

```clojure
(partition-all 3 [1 2 3 4 5 6 7 8])
;=> ((1 2 3) (4 5 6) (7 8))
```

```clojure
(mapcat (partial take (dec 3)) (partition-all 3 [1 2 3 4 5 6 7 8]))
;=> (1 2 4 5 7 8)
```

```
(= (__ [1 2 3 4 5 6 7 8] 3) [1 2 4 5 7 8])

(= (__ [:a :b :c :d :e :f] 2) [:a :c :e])

(= (__ [1 2 3 4 5 6] 4) [1 2 3 5 6])
```

```
(mapcat (partial take (dec 2)) (partition-all 2 [:a :b :c :d :e :f]))
;=> (:a :c :e)
```

```
(->> [:a :b :c :d :e :f]
     (partition-all 2)
     (mapcat (partial take (dec 2))))
```

FUN?

AS A CODER, WE NEED TO HAVÉ FUN

HAPPY CODING!

THANK YOU!

CREDITS

Wikipedia: Higher-order function:
http://en.wikipedia.org/wiki/Higher-order_function

Martin Fowler: Collection Pipeline:
http://martinfowler.com/articles/collection-pipeline

Replicate a Sequence: https://www.4clojure.com/problem/33

Drop Every Nth Item: https://www.4clojure.com/problem/41

Q/A

Nuttanart Pornprasitsakul
@visibletrap