January 25, 2016

7 Foreknown Functions

How I learned to stop writing for-loops.

Mikael Lundin

Higher order functions

1. Map
2. Fold
3. Partition
4. Reduce
5. Expand
6. Collect
7. Scan

What is a for-loop?

mutable

```
var input = new[] { 1, 2, 3, 4, 5 };
var result = new List<int>();
foreach (var item in input)
{
    result.Add(item * 2);
}
```

Difficult to test

A better way

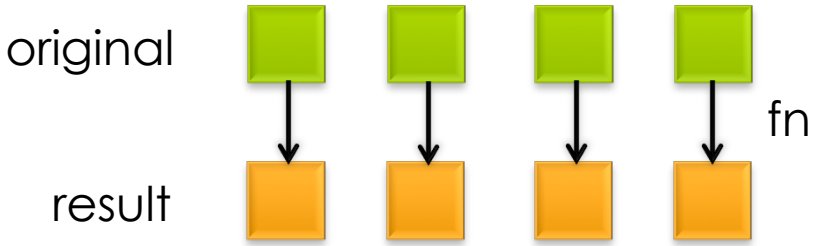
```
var input = new[] { 1, 2, 3, 4, 5 };
var result = input.Map(TimesTwo);

public int TimesTwo(int number)
{
   return number * 2;
}
```

Communicates intent

Test in isolation

1. Map

- Transform each element in a list
  result Map(fn, list)

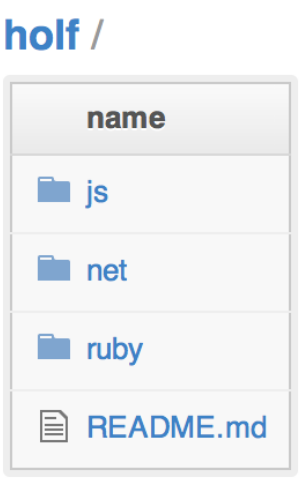


1. Map

- .NET
  ```
  var result = input.Select(i => i * 2);
  ```
- Ruby
  ```
  result = [1, 2, 3].map {|i| i * 2}
  ```
- JavaScript
  (no implementation)

Higher Order List Functions

- https://github.com/miklund/holf



2. Fold

```
var input = new[] { 1, 2, 3, 4, 5 };
var result = 0; // init
foreach (var item in input)
{
   result =  result + item;
}
```

2. Fold

- Aggregate items in a list.
  result Fold(fn, init, list)

result = fn(item3, fn(item2, fn(item1, init)))

2. Fold

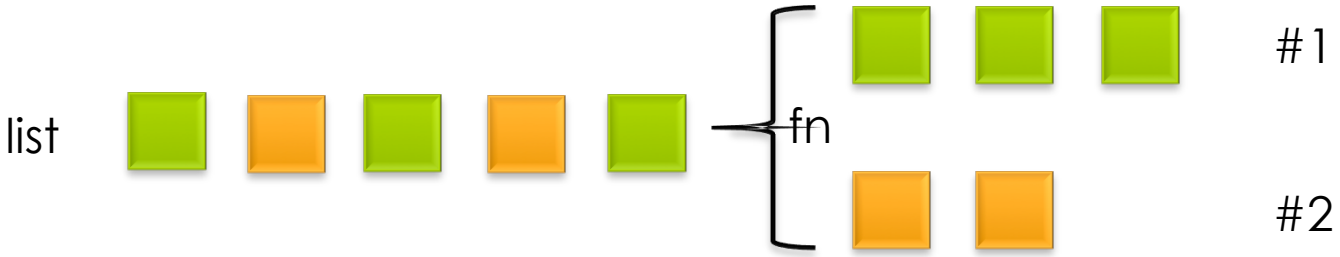
- .NET
  ```
  var result = input.Aggregate(0, (acc, i) => acc + i);
  ```
- Ruby
  ```
  result = input.inject(0) {|acc, i|  acc + i}
  ```
- JavaScript
  (no implementation)

3. Partition

```
var input = new[] { 1, 2, 3, 4, 5 };
var odds = new List<int>();
var evens = new List<int>();
foreach (var item in input) {
    if (item % 2 == 0) {
        evens.Add(item);
    }
    else    {
        odds.Add(item);
    }
}
```

3. Partition

- Partition a list into several lists.
  result = Partition(list, fn)



3. Partition

- .NET
  (no implementation)
- Ruby
  evens, odds = input.partition {|i| i.even?}
- JavaScript
  (no implementation)

4. Reduce

```
var input = new[] { 1, 2, 3, 4, 5 };
var result = input.First();
foreach (var item in input.Skip(1))
{
    if (result < item)
    {
        result = item;
    }
}
```

4. Reduce

- Reduce a collection to one value
  result = Reduce(list, fn)

result = fn(item3, fn(item2, item1))

4. Reduce

- .NET
  (no implementation?)

- Ruby
  result = input.reduce {|a, b| a > b ? a : b}

- JavaScript

```
result = input.reduce(function (a, b) {
  if (a > b) {
    return a;
  } else {
    return b;
  }
});
```

5. Collect

```
var urls = new[]
{
    "http://en.wikipedia.org/wiki/List_of_fantasy_novels_(A-H)",
    "http://en.wikipedia.org/wiki/List_of_fantasy_novels_(I-R)",
    "http://en.wikipedia.org/wiki/List_of_fantasy_novels_(S-Z)"
};

var result = new List<string>();
foreach (var url in urls)
{
    result.AddRange(BookTitlesFromUrl(url));
}

private IEnumerable<string> BookTitlesFromUrl(string url)
{
    // go get book titles from url
}
```

5. Collect

- Collect is an intelligent flatten
  result = Collect(list, fn)

Collect

fn( ) =

fn( ) =

fn( ) =

result =

5. Collect

```
private IEnumerable<string> BookTitlesFromUrl(string url)
{
    /// ... get all book titles from html page
}

var urls = new[] {
    "http://en.wikipedia.org/wiki/List_of_fantasy_novels_(A-H)",
    "http://en.wikipedia.org/wiki/List_of_fantasy_novels_(I-R)",
    "http://en.wikipedia.org/wiki/List_of_fantasy_novels_(S-Z)"
};

var titles = urls.Collect(BookTitlesFromUrl);
```

5. Collect

- .NET
  (not implemented)
- Ruby
  (not implemented)
- JavaScript
  (not implemented)

6. Expand

```
public IEnumerable<int> GetNumbers()
{
    var number = 0;
    while (true)
    {
        yield return number++;
    }
}
```

6. Expand

- Expand a sequence from initial state
  list = Expand(initial, fn)

6. Expand

- .NET
  (not implemented)
- Ruby
  (not implemented)
- JavaScript
  (not applicable)

6. Expand

- Luhn-validation

```
   1 9 3 8 0 8 2 0 9 0 0 5
*  2 1 2 1 2 1 2 1 2 1 2 1
-------------------------
   2+9+6+8+0+8+4+0+18+0+0+5  = 60
                             % 10
                             ----
                               0
```

7. Scan

```
// create a time table for busses
var departure = 609;

// first departure 10:09, 609 minutes after midnight
var timetable = new List<int> { departure };

// nine times
for (int i = 0; i < 9; i++)
{
    // add 20 minutes, 10 minutes
    departure += i % 2 == 0 ? 20 : 10;
    timetable.Add(departure);
}

// print: 10:09, 10:29, 10:39, 10:59, 11:09, 11:29, 11:39, 11:59, 12:09, 12:29,
foreach (var time in timetable)
{
    Console.Write("{0:00}:{1:00}, ", time / 60, time % 60);
}
```

7. Scan

- When you need to accumulate a value through a computation.
  result = Scan(list, fn)

7. Scan

```
//arrange
const int firstDeparture = 609; // 10:09
var intervals = 0.Expand(i => i == 20 ? 10 : 20);

// act
var timetable = intervals
    .Scan(IncreasingEachDeparture, firstDeparture)
    .Map(ToHoursAndMinutesString)
    .Take(5)
    .Reduce(ToCommaSeparatedString);

// assert
Assert.Equal("10:09, 10:29, 10:39, 10:59, 11:09", timetable);
```

7. Scan

- .NET
  (not implemented)
- Ruby
  (not implemented)
- JavaScript
  (not implemented)

Thank you!

- Website: http://litemedia.info
- Twitter: @mikaellundin