software

functionals

Daniel Jackson

For now, all rights reserved. Daniel Jackson, 2011.
functionals

since functions are first class
› can pass functions as arguments

functions that take functions as args
› are called ‘functionals’

can use functionals
› to capture common idioms

examples
› generators: a nice way to iterate over structures
› list functionals: map, fold (reduce), filter
arrays: a refresher

Javascript operations
› push/pop (back)
› unshift/shift (front)
› splicing
› concatenation

autofilling
› if set at index beyond length
› elements in between set to undefined

```javascript
> a = [3,5,7]
[3, 5, 7]
> a.push(9)
4
> a
[3, 5, 7, 9]
> a.unshift(1)
5
> a
[1, 3, 5, 7, 9]
> a.pop()
9
> a
[1, 3, 5, 7]
> a.shift()
1
> a
[3, 5, 7]
> a.splice(1,1,6)
[5]
> a
[3, 6, 7]
> a[4] = 8
8
> a
[3, 6, 7, undefined, 8]
```
generators

simulate Python’s generators in JS
› body of loop is function
› generate takes body as arg

Array.prototype.each = function (body) {
    for (var i = 0; i < this.length; i++) {
        body(this[i]);
    }
}

var sum = function (a) {
    var result = 0;
    a.each(function (e) {
        result += e;
    });
    return result;
}

>>> def elements(a):
...     for i in range(0, len(a)):
...         yield a[i]

>>> for e in elements ([1,2,3]):
...     print e
1
2
3

> sum([1,2,3])
6
map

Array.prototype.map = function (f) {
    var result = [];
    this.each (function (e) {
        result.push(f(e));
    });
    return result;
}

type

> map: list[A] x (A→B) → list[B]

> twice = function (x) {return x * 2;}
function (x) {return x * 2;}
> a = [1,2,3]
[1, 2, 3]
> map (a, twice)
[2, 4, 6]
**fold (or reduce)**

Array.prototype.fold = function (f, base) {
    var result = base;
    this.each (function (e) {
        result = f(e, result);
    });
    return result;
}

**type**

→ fold: list[A] x (A x B→B) x B → B

> times = function (x, y) {return x * y;}
function (x, y) {return x * y;}
> a = [1,2,3]
[1, 2, 3]
> reduce (a, times, 1)
6
filter

Array.prototype.filter = function (p) {
    var result = [];
    this.each (function (e) {
        if (p(e)) result.push(e);
    });
    return result;
}

type

> filter: list[A] x (A→Bool) → list[A]

> a = [1, 3, 5]
[1, 3, 5]
> filter (a, function (e) {return e < 4; })
[1, 3]
early returns?

› not easy with functionals
› always processes entire array

```javascript
var contains = function (a, e) {
  var f = function (x, found) {
    return found || (x === e)
  }
  return reduce(a, f, false);
}
```
find the bug

```javascript
Array.prototype.contains = function (e) {
    this.each(function (x) {
        if (x === e) return true;
    });
    return false;
}

> [1,2].contains(1)
false
```