software studio

functionals

Daniel Jackson
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 (aka iterators)

```python
>>> 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
```

```ruby
Ruby

```ruby
s = 0; [1,2,3].each { | e | s+= e }; print s
6=> nil
```

Python

```javascript
each = function (a, body) {
    for (var i = 0; i < a.length; i++) { body(a[i]); }
}

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

how it works (JS and Ruby)

- body of loop is function
- generate takes body as arg

```javascript
var sum = function (a) {
    var result = 0;
    each(a, function (e) {
        result += e;
    });
    return result;
}
```
map

map = function (a, f) {
    var result = [];
    each (a, 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)

```
fold = function (a, f, base) {
  var result = base;
  each (a, 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

```javascript
filter = function (a, p) {
    var result = [];
    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]
find the bug

contains = function (a, e) {
  each(a, function (x) {
    if (x === e) return true;
  });
  return false;
}

> contains([1,2], 1)
false
each to his or her own...

in JQuery
› each (collection, callback(index, value))
› when callback returns false, iteration stops

in ECMAScript 5
› array.forEach (callback(index,value,array))