software studio

# 3 closure examples

Daniel Jackson

# counter, from before

```
> seq = function () {
     seq.c += 1; return seq.c;}
function () {seq.c += 1; return
seq.c;}
> seq.c = 0
0
> seq()
1
> seq()
2
```

**note: violation of encapsulation!**

# counter, revisited

```
make_seq = function (c) {
    return function () {
        c += 1;
        return c;
    }
}
```

**what's going on?**

› local var is updated inside fun
› can't be accessed outside
› said to be 'encapsulated'

```
> seq = make_seq(0)
...
> seq()
1
> seq()
2
```

```
make_seq = function () {
    var c = 0;
    return function () {
        c += 1;
        return c;
    }
}
```

**suppose we always want to start at 0. how to do this?**

# fibonacci

## fibonacci function
- what scope is fib bound in?

## note use of var
- by default, you should make all variables local

## a problem
- testing golden ratio property
- try fib(20)/fib(19) etc
- at fib(34), gets very slow...

```
var fib = function (i) {
    if (i < 2) return 1;
    return fib(i-1) + fib(i-2);
}
```

## memoizing to rescue!

```
var memoize = function (f) {
    var memo = [];
    var fm = function (i) {
        if (memo[i]) return memo[i];
        result = f(i);
        memo[i] = result;
        return result;
    }
    return fm;
}

var mfib = memoize(function (i) {
    if (i < 2) return 1;
    return mfib(i-1) + mfib(i-2);
});
```

now mfib(1000) is instantaneous

## an abstract type

```
Sample = function () {
  var total = 0;
  var count = 0;
  result = {
    add: function (v) { total += v; count++ },
    avg: function () { return total/count; },
    sum: function () { return total; }
  };
  return result;
};
```

```
> var s = Sample ();
> s.add(1);
  s.add(2);
  s.add(6);
undefined
> s.avg();
3
> s.sum();
9
```

how robust is this ADT?
what can the client break?

but see: property accessors in ECMAScript 5