Function Decoration

Memoization and other useful tricks.

A brownbag workshop at

by Seth House

@whiteinge
seth@eseth.com

Function Decoration

Decorators.

```
// var decoratedFunction = decorate(regularFunction);
function decorate(f) {
    // One-time initialization.
    // Called only when decorate() is called.
    var privateDataHere = 'private!';

    // Function that wraps our 'f' and does something extra.
    return function(...args) {
        // Logic here that decides
        // if/when/how/why to call 'f' with 'args'
        // (or even with a modified 'args'!).
        return f(...args)
    }
}
```

Memoization.

```
// A very basic memoization function:
// var myMemoedFunction = memo(myFunction)
function memo(f) {
    // One-time initialization. Called only when memo() is called.
    var savedArgs = {};

    return function(...args) {
        // Warning: stringify'ing objects is not stable!
        // If a problem must sort keys
        // or choose another caching strategy.
        var argVal = JSON.stringify(args)
        if (savedArgs[argVal] === undefined) {
            savedArgs[argVal] = f(...args)
        }
        return savedArgs[argVal];
    }
}
```

Exchange memory for CPU.

```
// Block the UI thread on purpose!
// var memoedSleep = memo(sleep);
// memoedSleep(3000);
// memoedSleep(3000);
function sleep(time) {
    var now = Date.now();
    while (Date.now() < (now + time)) {}
    return 'done'
}
```

Once.

```
// Only invoke myFunction one time
// (ever! ...for as long as it's in scope):
// var myGuardedFun = once(myFunction)
function once(f) {
    var wasCalled = false

    return function(...args) {
        if (wasCalled === false) {
            wasCalled = true
            return f(...args)
        }
        return
    }
}
```

Throttle.

```
// Don't invoke myFunction more often than once every three seconds:
// var myThrottledFunction = throttle(myFunction, 3000)
function throttle(f, time) {
    var lastCalled = 915148798;

    return function(...args) {
        var now = Date.now();

        if (now - lastCalled > time) {
            var ret = f(...args)
            lastCalled = now;
            return ret;
        }
    }
}
```

Aspect-oriented programming.

```
// var loggingFunction = (ret) ⇒
//   { console.log('Logging. Got return value', ret); return ret; }
// var myLoggingFunction = after(myFunction, loggingFunction);
function after(fn, afterFn) {
    return function(...args) {
        var ret = fn(...args)
        return afterFn(ret)
    }
}
```

Cheap, simple ajax caching.

```
// Memoize JSON responses to avoid duplicate HTTP calls:
// Note, this is storing promise objects in the memoize cache!
const cacheJSON = memo((...args) ⇒
    fetch(...args).then(x ⇒ x.json()))
cacheJSON('https://api.github.com/users').then(console.log)
```