Doing things in series (async.waterfall)

We can run things in parallel, but what if we want to run things in series?

Example: read a file, transform it using a transformation service that doesn't support streams, then write the transformed file somewhere else.

Callbacks

caolan's async gives us `async.waterfall`

Assuming that

```
service.transform = function(string, callback)
```

```
function transformFile(inPath, outPath, callback) {
    async.waterfall([
        fs.readFile.bind(fs, inPath, 'utf8'),
        service.transform,
        fs.writeFile.bind(fs, outPath)
    ], callback);
}
transformFile(input, output, function(err) {
    if (!err) console.log("All ok!");
})
```

The callback is called with no error and the result of last task after all operations are complete , or when the first error is encountered.

Promises

Thanks to `.then`'s behavior when returning a promise, we can chain async operations without any helper tools:

```
function transformFile(input, output) {
    return fs.readFileAsync(input, 'utf8')
        .then(service.transformAsync)
        .then(fs.writeFileAsync.bind(fs, output));
}
transformFile(fileIn, fileOut).done(function() {
    console.log("All ok!");
}, function(err) {
    console.error(err);
});
```

The resulting promise is fulfilled when all the promises in the chain are fulfilled or is rejected with an error when the first error is encountered.

Notes

Admitedly, this code may be a bit hard to follow. Lets write it in a way that makes it easier - first, `async.waterfall`

```
function transformFile(inPath, outPath, done) {
    async.waterfall([
        function(callback) {
            fs.readFile(inPath, 'utf8', callback); },
        function(data, callback) {
            service.transform(data, callback); },
        function(transformed, callback) {
            fs.writeFile(outPath, transformed, callback); }
    ], done);
}
```

and also, the promises implementation:

```
function transformFile(input, output) {
    return fs.readFileAsync(input, 'utf8')
```

```
        .then(function(data) {
            return service.transformAsync(data);
        })
        .then(function(transformed) {
            return fs.writeFileAsync(output, transformed)
        });
}
```

`async.waterfall()` is pretty cool. It starts by giving us a callback, then when that callback is called with a result, it passes that result onto the next function in the list. But it also passes another callback to that function, which it can use to pass a result to the 3rd function in the list and so on.

Promises are also pretty cool. As we already know, when we return a promise from inside `.then()`, we get a promise for the same value on the outside. We can immediately attach another `.then()` on the outside and continue chaining.

Both of these patterns flatten our code.