Functional JS

Estimated time for completion: 45 minutes

Overview:

In this lab you will edit a simple HTML file whose purpose is to help understand how JavaScript's `reduce` and `reduceRight` methods work. The `reduce` function is one of the pillars of functional programming (you may have heard of "map/reduce") so understanding how it works can help you apply FP techniques in your code or understand other code written using those techniques. This is mostly an academic exercise, but should involve enough usage of functions and closures to help reinforce how powerful they can be.

Goals:

- Understand how the `reduce` and `reduceRight` methods work
- Populate a table full of information derived from intercepting call backs made by those methods

Tracing reduce and reduceRight

Implement the tracing behavior by intercepting calls made by `reduce` and `reduceRight`.

Criteria:

- Populate the table in the HTML file with calls made by `reduce` or `reduceRight`.

Steps:

1. Open functional.html in the `~/before` folder in your favorite browser. The page doesn't actually do anything. It does contain controls that let you specify which method to trace, what array to invoke the method on, a call back function, and an initial value.
    a. The page also contains a table that, so far, only contains a header row.
    b. Your main goal is to populate that table.
2. Open functional.html in the `~/before` in your favorite text editor. Look for the TODO comment at the bottom of the file.
3. Start by converting the values in the form into actual JavaScript objects.

    a. HTML form controls give values back as strings, but you need an array object, a function object, and some other object (initalValue).
    b. You can convert those strings to objects using JavaScript's `eval()` function. This function is normally considered "evil" but it's just fine for a learning exercise like this.
4. To actually perform the reduction, you can use code like this: `array[method](callback, initialValue);`. This assumes you previously defined `array`, `method`, `callback`, and `initialValue` variables by extracting values from the form controls and `eval()`'ing them as appropriate.
    a. Try this and put the method's return value in the result form control.
    b. The result for the values that the HTML page comes with should be 10.
5. Now that you're able to invoke the method and get back a result, you want to trace all calls to the callback function passed in to the method. With each callback invocation, you'll add a new row to the table.
    a. This is where it gets tricky. Since your goal is to intercept calls the method makes to your callback, you'll have to introduce something in between the method and your callback. Your goal is not to replace `reduce` or `reduceRight`, but to replace your callback with something that can do the tracing and then call your callback.
    b. Hint: `array[method](function(p, c, i, a) { return callback(p, c, i, a); }, initialValue);` intercepts all calls to the callback function, but doesn't do any tracing.
6. Since your code is running on page load, try wrapping it in a function and invoking that function whenever you change any of the values in the form.
    a. Handle the "change" or "input" (if you're using a modern browser) events to detect when those controls change values.
    b. Try leaving the initialValue control empty. Write code to detect when that control is empty so that you can omit the second argument to the method when it is. This is key to understanding how `reduce` and `reduceRight` works.
7. Does playing with the fully functional form help at all? Can you think of examples where you could use `reduce`? Can you "see" the closure in the code you wrote?

Solutions:

The final solution for this lab is available in the `~/after` directory.