Functions as Values

We discuss more extensively one of the key features of Javascript, namely that functions are first-class values.

Relevant Links

- Flanagan's book, section 8.4

Notes

- Functions in Javascript are first-class values. This means that they can be used anywhere a value would be used.
- This simple fact has extremely deep consequences. We will start examining them here.
- Here is one example of this:

```
var ops = {
    "add": function(a, b) { return a + b; },
    "sub": function(a, b) { return a - b; },
    "mult": function(a, b) { return a * b; },
    "div": function(a, b) { return a / b; }
};
function doOp(op, v1, v2) {
    if (typeof ops[op] === 'function') {
        return ops[op](v1, v2);
    } else {
        throw new Error("Unknown operation");
    }
}
doOp("add", 5, 6);
```

  In this example, the ops object contains keys whose values are functions. The function doOp uses a string "op" to access the corresponding key, and get back a function. That function is then called on the two values v1 and v2.

- Here is another example, from the Array object. We have an array of strings, and we want to print a message for each one. We could do a for loop, but here is an alternative:

```
var arr = [8, 3, 2, 4];
function printIt(v) { console.log("You are looking at:", v); }
arr.forEach(printIt);
```

  So what happens here is that forEach expects a function as an argument. It then calls the function once for each element of the array, passing that element as an argument. This form of iteration is something you should get used to, and we will spend some more time going over it in a future segment. We could even enter the function directly, without giving it a name:

```
[8, 3, 2, 4].forEach(function(v) {
    console.log("You are looking at:", v);
});
```

But the thing to take from this is that *you can have a function that takes as argument another function*. These are called **higher-order functions** and they are a cornerstone of what is known as "functional programming".

- Here is a different example: In this case, our function actually returns a function:

```
function makeGreeter(name) {
    var greeter = function() {
        console.log("Greetings, " + name + "!");
    };

    return greeter;
}
var greet = makeGreeter("John");
greet();
greet();
```

So our function makeGreeter takes as argument a name string, and returns a function. When that function called, it does a console.log. Notice that nothing happens until we actually call greet.

Some times when we define a local variable only to immediately return it, it is often customary to skip the "defining" step:

```
function makeGreeter(name) {
    return function() {
        console.log("Greetings, " + name + "!");
    };
}
```

Practice

1. Write a function loving that takes a name string as input, and returns a function. That function takes a language string as input, and *returns* the string "<name> loves <language>". So for instance if var f = loving('Skiadas'); then f('Javascript') would result in the string "Skiadas loves Javascript".
2. Write a function map that takes as input a function f. It then returns a function that takes as input an array arr, and proceeds to create a new array by applying f to each element of arr and collecting the results.