Can Your Programming Language Do This?

This item ran on the Joel on Software homepage on Tuesday, August 01, 2006

One day, you're browsing through your code, and you notice two big blocks that look almost exactly the same. In fact, they're exactly the same, except that one block refers to "Spaghetti" and one block refers to "Chocolate Moose."

```
// A trivial example:

alert("I'd like some Spaghetti!");
alert("I'd like some Chocolate Moose!");
```

These examples happen to be in JavaScript, but even if you don't know JavaScript, you should be able to follow along.

The repeated code looks wrong, of course, so you create a function:

```
function SwedishChef( food )
{
    alert("I'd like some " + food + "!");
}

SwedishChef("Spaghetti");
SwedishChef("Chocolate Moose");
```

OK, it's a trivial example, but you can imagine a more substantial example. This is better code for many reasons, all of which you've heard a million times. Maintainability, Readability, Abstraction = Good!

Now you notice two other blocks of code which look almost the same, except that one of them keeps calling this function called BoomBoom and the other one keeps calling this function called PutInPot. Other than that, the code is pretty much the same.

```
alert("get the lobster");
PutInPot("lobster");
PutInPot("water");

alert("get the chicken");
BoomBoom("chicken");
BoomBoom("coconut");
```

Now you need a way to pass an argument to the function which itself is a function. This is an important capability, because it increases the chances that you'll be able to find common code that can be stashed away in a function.

```
function Cook( i1, i2, f )
{
    alert("get the " + i1);
    f(i1);
    f(i2);
}

Cook( "lobster", "water", PutInPot );
Cook( "chicken", "coconut", BoomBoom );
```

Look! We're passing in a function as an argument.

Can your language do this?

Wait... suppose you haven't already defined the functions PutInPot or BoomBoom. Wouldn't it be nice if you could just write them inline instead of declaring them elsewhere?

```
Cook( "lobster",
      "water",
      function(x) { alert("pot " + x); } );
Cook( "chicken",
      "coconut",
      function(x) { alert("boom " + x); });
```

Jeez, that is handy. Notice that I'm creating a function there on the fly, not even bothering to name it, just picking it up by its ears and tossing it into a function.

As soon as you start thinking in terms of anonymous functions as arguments, you might notice code all over the place that, say, does something to every element of an array.

```
var a = [1,2,3];

for (i=0; i<a.length; i++)
{
    a[i] = a[i] * 2;
}

for (i=0; i<a.length; i++)
{
    alert(a[i]);
}
```

Doing something to every element of an array is pretty common, and you can write a function that does it for you:

```
function map(fn, a)
{
    for (i = 0; i < a.length; i++)
    {
        a[i] = fn(a[i]);
    }
}
```

Now you can rewrite the code above as:

```
map( function(x){return x*2;}, a );
map( alert, a );
```

Another common thing with arrays is to combine all the values of the array in some way.

```
function sum(a)
{
    var s = 0;
    for (i = 0; i < a.length; i++)
        s += a[i];
    return s;
}

function join(a)
{
    var s = "";
    for (i = 0; i < a.length; i++)
        s += a[i];
    return s;
}

alert(sum([1,2,3]));
alert(join(["a","b","c"]));
```

sum and **join** look so similar, you might want to abstract out their essence into a generic function that combines elements of an array into a single value:

```
function reduce(fn, a, init)
{
    var s = init;
    for (i = 0; i < a.length; i++)
        s = fn( s, a[i] );
    return s;
}

function sum(a)
{
    return reduce( function(a, b)
                       { return a + b; },
                   a, 0 );
}

function join(a)
{
    return reduce( function(a, b)
                       { return a + b; },
                   a, "" );
}
```

Many older languages simply had no way to do this kind of stuff. Other languages let you do it, but it's hard (for example, C has function pointers, but you have to declare and define the function somewhere else). Object-oriented programming languages aren't completely convinced that you should be allowed to do anything with functions.

Java required you to create a whole object with a single method called a functor if you wanted to treat a function like a first class object. Combine that with the fact that many OO languages want you to create a whole file for each class, and it gets really klunky fast. If your programming language requires you to use functors, you're not getting all the benefits of a modern programming environment. See if you can get some of your money back.

How much benefit do you really get out of writting itty bitty functions that do nothing more than iterate through an array doing something to each element?

Well, let's go back to that **map** function. When you need to do something to every element in an array in turn, the truth is, it probably doesn't

matter what order you do them in. You can run through the array forward or backwards and get the same result, right? In fact, if you have two CPUs handy, maybe you could write some code to have each CPU do half of the elements, and suddenly **map** is twice as fast.

Or maybe, just hypothetically, you have hundreds of thousands of servers in several data centers around the world, and you have a really big array, containing, let's say, again, just hypothetically, the entire contents of the internet. Now you can run **map** on thousands of computers, each of which will attack a tiny part of the problem.

So now, for example, writing some really fast code to search the entire contents of the internet is as simple as calling the **map** function with a basic string searcher as an argument.

The really interesting thing I want you to notice, here, is that as soon as you think of **map** and **reduce** as functions that everybody can use, and they use them, you only have to get one supergenius to write the hard code to run **map** and **reduce** on a global massively parallel array of computers, and all the old code that used to work fine when you just ran a loop still works only it's a zillion times faster which means it can be used to tackle huge problems in an instant.

Lemme repeat that. By abstracting away the very concept of looping, you can implement looping any way you want, including implementing it in a way that scales nicely with extra hardware.
And now you understand something I wrote a while ago where I complained about CS students who are never taught anything but Java:

Without understanding functional programming, you can't invent MapReduce, the algorithm that makes Google so massively scalable. The terms Map and Reduce come from Lisp and functional programming. MapReduce is, in retrospect, obvious to anyone who remembers from their 6.001-equivalent programming class that purely functional programs have no side effects and are thus trivially parallelizable. The very fact that Google invented MapReduce, and Microsoft didn't, says something about why Microsoft is still playing catch up trying to get basic search features to work, while Google has moved on to the next problem: building Skynet^H^H^H^H^H^H the world's largest massively parallel supercomputer. I don't think Microsoft completely understands just how far behind they are on that wave.

Ok. I hope you're convinced, by now, that programming languages with first-class functions let you find more opportunities for abstraction, which means your code is smaller, tighter, more reusable, and more scalable. Lots of Google applications use MapReduce and they all benefit whenever someone optimizes it or fixes bugs.

And now I'm going to get a little bit mushy, and argue that the most productive programming environments are the ones that let you work at *different levels of abstraction*. Crappy old FORTRAN really didn't even let you write functions. C had function pointers, but they were ugleeeeee and not anonymous and had to be implemented somewhere else than where you were using them. Java made you use functors, which is even uglier. As Steve Yegge points out, Java is the Kingdom of Nouns.

Correction: The last time I used FORTRAN was 27 years ago. Apparently it got functions. I must have been thinking about GW-BASIC.

About the Author: I'm your host, Joel Spolsky, a software developer in New York City. Since 2000, I've been writing about software development, management, business, and the Internet on this site. For my day job, I run Fog Creek Software, makers of FogBugz - the smart bug tracking software with the stupid name, and Fog Creek Copilot - the easiest way to provide remote tech support over the Internet, with nothing to install or configure.