Functional, or declarative, programming is a very powerful programming method and is gaining popularity in the software industry. This article introduces some of the relevant functional programming concepts, and provides examples to use those concepts effectively. The author explains how to write elegant code with JavaScript™, which can import constructs and features from functional programming.

Introduction

Functional programming languages have been in academia for quite some time, but historically they do not have extensive tools and libraries available. With the advent of Haskell in the .NET platform, functional programming is becoming more popular. Some traditional programming languages, such as C++ and JavaScript, import constructs and features from functional programming. In many cases, repetitive code in JavaScript leads to clumsy coding. You can avoid all that if you use functional programming. Also, you can write more elegant callbacks with a functional programming style.

Functional programming

Functional programming describes only the operations to be performed on the inputs to the programs, without use of temporary variables to store intermediate results. The emphasis is to capture “what and why” rather than the “how.” It emphasizes the definition of functions rather than the implementation of state machines, in contrast to procedural programming, which emphasizes execution of sequential commands.

Large scale knowledge management applications benefit greatly from using a functional programming style as it simplifies development.

Because functional programming encompasses a very different way of composing programs, programmers who are used to the imperative paradigm can find it difficult to learn. In this article you'll see examples of how to write good, elegant code in JavaScript in a functional style. I'll discuss:

- **Functional programming concepts**, including anonymous functions, different ways to call functions, and how to pass functions as arguments to other functions.
• **Use of functional concepts**, with examples of: extending array sort; elegant code for dynamic HTML generation; and application of a sequence of functions.

**Functional programming concepts**

Many developers know how to code in languages where you specify the method of solving a problem by describing "how." For example, to code a function to compute a factorial, I describe the program by coding a loop or using recursion to find the product of all the numbers. In both cases, the procedure for the computation is detailed in the program. **Listing 1** shows possible C code for a factorial.

**Listing 1. Factorial in procedural style**

```c
int factorial (int n)
{
    if (n <= 0)
        return 1;
    else
        return n * factorial (n-1);
}
```

Such languages are also called *procedural* programming languages because they define the procedure to solve the problem. Functional programming is markedly different from that philosophy. In functional programming, you need to describe the "what" of the problem. Functional programming languages are also called *declarative*. The same program computing factorial would be written as a product of all the numbers up to \( n \). A typical functional program for a factorial looks like the example in **Listing 2**.

**Listing 2. Factorial in functional style**

```c
factorial n, where n <= 0  := 1
factorial n    := foldr * 1 take n [1..]
```

The second statement instructs you to take a list of first \( n \) numbers starting from 1 (\( \text{take} \ n \ [1..] \)), then find their product with 1 as the identity. This definition does not have the loop, or recursion, as with the earlier case. It is like the mathematical definition of the factorial function. Once you know the meaning of the library functions (\( \text{take} \) and \( \text{foldr} \)), and the notation (list notation \([\ ])\), it is very easy to code and also very readable.

You can write a routine to process each element of an n-ary tree using breadth-first or depth-first traversal, depending on a parameter, where the elements could be of any generic type, in just three lines of Miranda code.

Historically, functional programming languages have not been very popular for various reasons. Recently, however, a few of these languages are entering the computer industry. One example is Haskell in the .NET platform. In other cases, some of the existing languages borrow concepts from functional programming languages. Iterators and continuations show up in some implementations of C++ and some functional constructs provided in JavaScript. However, by borrowing functional constructs, the overall programming paradigm of the language does not change. JavaScript has not become a functional programming language because of the addition of the functional constructs.
I will now discuss the various niceties of the functional constructs in JavaScript, and ways to use them in our day-to-day coding and work. You'll start with some basic capabilities, then use them to see some of the more interesting applications.

Anonymous functions

In JavaScript, you have the ability to write anonymous functions, or functions without a name. Why would you need that? Keep reading, but first you'll learn how to write one. If you had the following JavaScript function:

**Listing 3. A typical function**

```javascript
function sum(x, y, z) {
  return (x + y + z);
}
```

Then the corresponding anonymous function would look like:

**Listing 4. An anonymous Function**

```javascript
function(x, y, z) {
  return (x + y + z);
}
```

To use it, you write the following:

**Listing 5. Applying an anonymous function**

```javascript
var sum = function(x, y, z) {
  return (x + y + z);
}(1, 2, 3);
alert(sum);
```

Using functions as values

You could also use functions as values. You could have variables with functions as values assigned to them. In the last example, you could also do the following:

**Listing 6. Using function assignment**

```javascript
var sum = function(x, y, z) {
  return (x + y + z);
};
alert(sum(1, 2, 3));
```

In the example in Listing 6 above, the variable sum is assigned the function definition itself. Therefore, sum is a function and called anywhere you want.

Different ways of calling functions

JavaScript allows you to call a function in two ways, as shown in Listings 7 and 8.

**Listing 7. Typical function application**

```javascript
alert ("Hello, World!");
```
Or

**Listing 8. Using function as an expression**

```javascript
(alert) ("Hello, World!");
```

Therefore you can also write the following:

**Listing 9. Function application just after definition**

```javascript
( function(x,y,z) { return (x+y+z) } ) (1, 2, 3);
```

You might write a function expression in parentheses and then pass arguments to get them evaluated. Though in the example in **Listing 8**, with a function name directly enclosed in parentheses, this need not be the case while using it as shown in **Listing 9**.

**Passing functions as arguments to other functions**

You can also pass functions as argument to other functions. Though this is not a new concept, it is used in the subsequent examples extensively. You can pass function arguments as shown in **Listing 10**.

**Listing 10. Passing function as a parameter and applying it**

```javascript
var passFunAndApply = function (fn,x,y,z) { return fn(x,y,z); };
var sum = function(x,y,z) {
  return x+y+z;
};
alert( passFunAndApply(sum,3,4,5) ); // 12
```

Execution of the last alert statement prints the value 12.

**Using functional concepts**

The previous section showed some programming concepts using the functional style. The examples are by no means complete coverage of all the concepts, nor are they in any order of importance, but are the relevant concepts for this discussion. To quickly summarize, with the functional style in JavaScript:

- Functions need not have names all the time.
- Functions can be assigned to variables like other values.
- A function expression can be written and enclosed in parentheses for application later.
- Functions can be passed as arguments to other functions.

This section shows some examples of how you can use the concepts effectively to write good and elegant code in JavaScript. (Using a functional style of JavaScript, you can do many other things that are out of the scope of this discussion.)

**Extending array sort**

Let's plan to write a sort method that can sort the array elements according to their date.

Writing this is very simple in JavaScript. The sort method of array objects takes an optional
parameter that is the comparison function. In this case, you need the comparison function in Listing 11.

**Listing 11. Comparison function**

```javascript
function (x,y) {
  return x.date â‰¥ y.date;
}
```

To get the required function, use the example in Listing 12.

**Listing 12. Extension of the sort function**

```javascript
arr.sort( function (x,y) { return x.date â‰¥ y.date; } );
```

Where `arr` is an object of type array. This will sort all the objects in `arr` according to their respective dates. The comparison function, along with its definition, is passed to the sort function to complete the sort operation. With this function:

- Each JavaScript object has a date property.
- The sort function of array type in JavaScript takes an optional parameter, which is the comparison function used for sorting. This is similar to the `qsort` function in the C library.

**Elegant code for dynamic HTML generation**

In this example, you'll see how to write elegant code to generate dynamic HTML from arrays. You can generate tables from the values obtained from an array. Or, you might generate ordered and unordered lists using the contents of an array. You can also generate vertical or horizontal menu items.

The coding style in Listing 13 is commonly used to generate dynamic HTML from arrays.

**Listing 13. Normal code for generating dynamic HTML**

```javascript
var str=' ‘;
for (var i=0;i<arr.length;i++) {
  var element=arr[i];
  str+=... HTML generation code...
}
document.write(str);
```

You can replace this code using the code in Listing 14.

**Listing 14. Generic way of generating dynamic HTML**

```javascript
Array.prototype.fold=function(templateFn) {
  var len=this.length;
  var str=' ‘;
  for (var i=0 ; i<len ; i++)
    str+=templateFn(this[i]);
  return str;
}

function templateInstance(element) {
  return ... HTML generation code ...
}

document.write(arr.fold(templateInstance));
```
I've used the prototype property of `Array` type to define the new function `fold`. You can now use this function in any of the arrays that are defined later.

**Application of a sequence of functions**

Consider the case where you want to use a set of functions as the callback function. For this purpose you'll use a `window.setTimeout` function, which has two parameters. The first parameter is the function to be called after the number of milliseconds indicated by the second parameter. **Listing 15** shows one way to do this.

**Listing 15. Calling a set of functions in a callback**

```javascript
window.setTimeout(function(){alert('First!');alert('Second!');}, 5000);
```

**Listing 16** shows the more elegant way of doing it.

**Listing 16. Elegant way of calling a sequence of functions**

```javascript
Function.prototype.sequence=function(g) {
    var f=this;
    return function() {
        f();g();
    }
};
function alertFrst() { alert('First!'); }
function alertSec() { alert('Second!'); }
setTimeout( alertFrst.sequence(alertSec), 5000);
```

You can also use an extension of the code in **Listing 16** if you want to call one callback after calling another while handling an event. Perhaps that is an exercise for you to pursue, now that your interest is sparked.

**In conclusion**

In many areas, you can apply functional programming concepts in JavaScript to accomplish day-to-day activities in an elegant way. The examples in this article show only a few of the cases. If you identify the right scenarios for functional programming, and apply the concepts, you'll make a lot of sense and can increase your elegance level.
Related topics

- "Functional programming in the Java language: Use closures and higher order functions to write modular Java code" (developerworks, July 2004): Learn how to use functional programming constructs such as closures and higher order functions to write well-structured, modular code in the Java language.
- "Beginning Haskell" (developerWorks, September 2001): Take this tutorial, which provides an introduction to the paradigm of functional programming, with specific illustrations in the Haskell 98 language.
- **Tutorial Papers in Functional Programming**: Find several resources on topics related to functional programming.
- **Free downloads and learning resources**: Improve your work with software downloads from developerWorks.