http://helephant.com/2008/08/19/functions-are-first-class-objects-in-javascript/

Functions are first class objects in javascript

Functions in javascript are [first class objects](http://en.wikipedia.org/wiki/First-class_function). This means that javascript functions are just a special type of object that can do all the things that regular objects can do.

Really, just like any other variable

Here are a few of the important objects things that you can do with a function in javascript.

A function is an instance of the Object type:

1. **function** feedCat(){
2. alert("Kibble, tinned food and water");
3. }
4. alert(feedCat instanceof Object);

A function can have properties and has a link back to its constructor method:

1. feedCat.food = "kibble";
2. alert(feedCat.food);
3. alert(feedCat.constructor);

You can store the function in a variable:

1. **function** feedCat(){
2. alert("Kibble, tinned food and water");
3. }
4. **var** eveningChore = feedCat;eveningChore();

You can pass the function as a parameter to another function:

1. **function** doEveningChores(chores){
2. for(**var** x=0; x<chores .length; x++)
3. chores[x]();
4. }
5. doEveningChores([feedCat]);

You can return the function from a function:

1. **function** tonightChores(){
2. return feedCat;
3. }
4. **var** tonight = tonightChores();
5. tonight();

[Complete example](http://helephant.com/wp-content/uploads/2008/11/function.html).

Can reduce repetitive code

Being able to pass logic around an application in the form of a function means it’s possible to move a lot of repetitive code into a library function. It makes it easier to separate the unique pieces of logic from the generally useful logic.

For example, imagine you have a list of chocolate bars and you want to find all the ones that are made by Mars because you love M&Ms and you want to find out what other chocolatey goodness you could be enjoying (I have been on a diet for seven months.. I content myself with imagining chocolate..).

You could write a loop to iterate through the complete list and apply your item selection logic like this:

1. **var** chocolateBars = [
2. {name: "Galaxy", manufacturer: "Mars"},
3. …];
4. **var** marsChocolate = [];
5. for(**var** x=0; x<chocolatebars .length; x++){
6. if(chocolateBars[x].manufacturer == "Mars")
7. marsChocolate.push(chocolateBars[x]);
8. }

That’s great! Problem solved. Only now it’s Christmas time in your application and you have to sort the naughty list from the nice list. You’re always so busy at this time of year! Plus you learnt in programmer school that [you shouldn’t have to write the same thing twice](http://c2.com/cgi/wiki?DontRepeatYourself).

This sounds like the job for a library function! Let’s split the thing that will be the same each time (walking the existing list, building the new list) from the thing that will be different each time (applying the filter). The walking the list and building the new list logic can go into a library function. The filtering logic can be passed in as a parameter:

1. **var** array\_helper = {
2. filter: **function**(list, filter)
3. {
4. **var** matches = [];
5. for(**var** x=0; x<list .length; x++)
6. {
7. if(filter(list[x]))
8. matches.push(list[x]);
9. }
10. return matches;
11. }
12. };
13. **var** marsChocolate = array\_helper.filter(chocolateBars,
14. **function**(item) { return item.manufacturer == "Mars"
15. });
17. **var** naughtyList = array\_helper.filter(childrenOfTheWorld,
18. **function**(item) { return item.naughtiness &gt; 50;
19. });
21. **var** niceList = array\_helper.filter(childrenOfTheWorld,
22. **function**(item) { return item.naughtiness &lt;= 50;
23. });

[Complete example](http://helephant.com/wp-content/uploads/2008/11/function2.html) (except for the naughty list – I wouldn’t presume to take Santa’s job).

Now the code that actually needs to be written for each list that needs filtering is really simple. If there’s a problem in this logic it will be dead easy to spot. Any problems with the logic in the library function can be fixed in one place in the code. Plus you can get a list of delicious confectionery any time you want and the good children of the world will all get their presents at Christmas time.

The filter function is such a useful idea that it’s a part of popular javascript libraries like [JQuery](http://docs.jquery.com/Traversing/filter" \l "fn) and [Dojo](http://api.dojotoolkit.org/jsdoc/dojo/HEAD/dojo.filter). It will also be included in the browser as a standard part of [Javascript 1.6](http://developer.mozilla.org/en/docs/New_in_JavaScript_1.6#Array_extras). However the same idea can be applied in a lot of different places to reduce the amount of repeated code that’s not really pulling its weight.

Methods are properties that contain functions

Object methods are nothing special in javascript. They just are properties that happen to contain a function rather than something like a string or number value:

1. **var** sabby = {
2. name : "Sabby",
3. species: "cat",
4. hello : **function**() { alert("hissss"); }
5. };

This was pretty clever of the javascript language designers because it meant that they didn’t need to do anything special to support object methods. This is part of the reason why [javascript can have objects without having classes](http://helephant.com/2008/08/building-simple-objects/).

Makes javascript flexible and dynamic

A lot of javascript’s flexibility comes from being able to treat functions as first class objects. Pretty much every example in the [javascript objects articles](http://helephant.com/2008/08/how-javascript-objects-work/) takes advantage of being able to treat a function just like a regular object.

All of the basic techniques for using objects in javascript rely on this one thing. Simple objects, constructor functions and prototypes all involve assigning methods to the properties of an object.

It also helps javascript to be flexible and lightweight. Useful ideas from other languages like namespacing, custom events, static methods and extension methods can all be simulated because it’s possible to create a function and put it wherever you need.

Further reading

The best reference I could find about this was an article about [functional javascript](http://www.hunlock.com/blogs/Functional_Javascript). It explains a heap of the [functional language](http://en.wikipedia.org/wiki/Functional_programming) features that javascript has.

Raganwald has an [interesting article](http://weblog.raganwald.com/2007/01/closures-and-higher-order-functions.html) with some more general information about why techniques like this are so useful. It compares ruby and java but the logic still applies to javascript.

What’s next?

[Anonymous functions](http://helephant.com/2008/08/javascript-anonymous-functions/) are functions that are dynamically created at runtime using the function operator. Anonymous functions go hand in hand with functions being first class objects because these are two of the big things that javascript such a flexible and dynamic language.

This article is part of a set of related posts about [How javascript objects work](http://helephant.com/2008/08/how-javascript-objects-work/).