We can attach a predefined function to a predefined object (in which case only that object can call the function, not any other object derived from the same prototype):

```
myObj.doSomethingNew=doSomething;
myObj.doSomethingNew(x,y,z);
```

We can also attach functions such that every instance of a class can access them, by adding the function (either predefined or declared inline) to the new object in the constructor, as we saw in section B.2.2, or by attaching it to the prototype.

Once we've done this, though, they aren't very strongly attached, as we will see.

B.3.3 Borrowing functions from other objects

Giving functions first-class object status alters the capabilities of a language significantly. Furthermore, an understanding of these alterations is important when coding GUI event handling, so most Ajax programmers would do well to understand it.

So what are these new capabilities? Well, first off, one object can borrow another's function and call it on itself. Let's define a class to represent species of tree in a taxonomic system:

```
function Tree(name, leaf, bark){
  this.name=name;
  this.leaf=leaf;
  this.bark=bark;
}
```

Next, we'll add a function to it that provides a quick description of the tree:

```
Tree.prototype.describe=function(){
  return this.name+": leaf="+this.leaf+", bark="+this.bark;
}
```

If we now instantiate a Tree object and ask it to describe itself, we get a fairly predictable response:

```
var Beech=new Tree("green, serrated edge","smooth");
alert(Beech.describe());
```

The alert box will display the text `Beech: leaf=green, serrated edge, bark=smooth`. So far, so good. Now let us define a class to represent a dog:

```
function Dog(name,bark){
  this.name=name;
  this.bark=bark;
}
```