Every program has a shape. On a small scale, this

shape is determined by its division into functions and

the blocks inside those functions. Programmers have a

lot of freedom in the way they structure their programs.

Shape follows more from the taste of the programmer

than from the program’s intended functionality.

*Modules* divide programs into clusters of code that, by *some* criterion, be-

long together.

MOTIVATION

who aren’t yet familiar with the code find

what they are looking for and makes it easier for the programmer to keep

things that are related close together.

Some programs are even organized along

of

a project, when you are not quite

Namespacing

only this function can see it). JavaScript does

not. Thus, by default, everything that needs to be visible outside of the scope

of a top-level function is visible *everywhere*.

Namespace pollution, the problem of a lot of unrelated code having

to share a single set

Namespace pollution

**Namespacing**

Most modern programming languages have a scope level between *global* (everyone

can see it) and *local* (only this function can see it). JavaScript does

not. Thus, by default, everything that needs to be visible outside of the scope

of a top-level function is visible *everywhere*.

Though JavaScript provides no actual module construct yet, objects can

be used to create publicly accessible subnamespaces, and functions can be

used to create an isolated, private namespace inside of a module. Later in

this chapter, I will discuss a way to build reasonably convenient, namespaceisolating

modules on top of the primitive concepts that JavaScript gives us.

USING FUNCTIONS AS NAMESPACE/Module

Functions are the only thing in javascript that create a new scope.

Consider this module for returning day of week.

Var days=[‘monday’,]

Var dayNames=function(number){

return days[number]

}

The dayName function is part of the modules interface,the days array is not,we would prefer not to spill it into the global scope.

We can do this

Var dayName=(function(){

Var days=[‘monday’];

return function(number){

return days[number];

}

})()

Objects as interfaces

Now imagine that we would like to add another function to our day-of-week module, what we can do is to wrap all the returned functions in an object

Var weekday=(function(){

Var names=[‘sunday’,’monday’,’tuesday’];

return {

name: function(number){return names[number]},

number:function(name){return names.indexOf(name)}

}

})()

The alternative is to declare an object,conventionally named exports, and add properties to that whenever we need something that needs to be exported