Organizing your code

Namespacing

Let's say we have this code

```
var visitors = 0;

function visit() {
    visitors++;
}

function reset() {
    visitors=0;
}
```

Namespacing

Namespaces allow us to group code and help us to avoid name-collisions.

*Unfortunately, everything you create in JavaScript is by default **global**. Now obviously, this is a recipe for disaster.*

*To avoid this behaviour, you can create a **single global object** for your app and make all functions and variables properties of that global object*

IRON HACK

Namespacing

```
var myApp = {};

myApp.visitors = 0;

myApp.visit = function() {
    return myApp.visitors++;
}

myApp.reset = function() {
    myApp.visitors = 0;
}
```

Exercise

Take the following code that is in danger of collisions and put it inside a namespace

```
var cats = 15;
var rabbits = 12;
var dogs = 21;
var adoptDog = function(){
    dogs--;
    alert("A dog has found a home!");
};
```

The Module Pattern

The module pattern strives to improve the reduction of globally scoped variables, thus decreasing the chances of collision with other code throughout an application.

On top of that, it supports the ability to focus on public and private access to methods & variables.

The Module Pattern

Let's use our last example.

```
var myApp = {};

myApp.visitors = 0;

myApp.visit = function() {
    return myApp.visitors++;
}

myApp.reset = function() {
    myApp.visitors = 0;
}
```

The Module Pattern

```
var myApp = (function(){
     var visitors = 0;
     var my_public_object = {};

     my_public_object.visit = function() {
          return visitors++;
     }

     my_public_object.reset = function() {
          visitors = 0;
     }
     my_public_object.say_visits = function() {
          console.log(visitors);
     }
     return my_public_object;
})();
```

Exercise

Let's convert the previous namespace into a module, making sure that only the adoptDog method is public.