Objects/ Clousure

objects have a special hidden property [[Prototype]]
t __proto__ is *not the same* as [[Prototype]]. That's a getter/setter for it.

```
let animal = {
  eats: true
};
let rabbit = {
  jumps: true
};

rabbit.__proto__ = animal; // (*)
```

Here the line (*) sets **animal to be a prototype of rabbit.**

```
// we can find both properties in rabbit now:
alert( rabbit.eats ); // true (**)
alert( rabbit.jumps ); // true
```

or

==========

```
let animal = {
  eats: true,
  walk() {
    alert("Animal walk");
  }
};

let rabbit = {
  jumps: true,
  __proto__: animal
};

let longEar = {
  earLength: 10,
  __proto__: rabbit
}

// walk is taken from the prototype chain
longEar.walk(); // Animal walk
alert(longEar.jumps); // true (from rabbit)
```

==========

```
function Person(x){
this.x =x;
}

Person.prototype.syaNmae = function () {
console.log(“”);
}

Person p1 = new Person (“dd”);
Person p2 = new Person (“mm”);
==========================================


function Rectangle (len,w){
this.len =len;
this.w = w;
}

function Squer(s){
this.len =s;
this.w = s;
}

Square.prototype = new Rectangle();

Square.prototype.construcgor = Square;
==========================================
rabbit.prototype = Object.create(animal.prototype, {constructor: {value:rabbit,
        configurable:true,numerable:true,writable:true}});

==========================================

// strict mode is important, as otherwise instead of a TypeError
// you will just have a silent failure

'use strict'

const obj = {
  answer: 42
}

Object.freeze(obj)
```

obj.answer = 43 // throws TypeError about read only property
obj.newProperty = 'foo' // throws TypeError about object not being extensible
Object.defineProperty(obj, 'bar', {value: 'bar'}) // the same TypeError

==========

Module Pattern

Closure : closure are block that access the data outside of its scope

1 Creating a module

First we start using a **anonymous closure**.
Anonymous closures are just functions that wrap our code and create an enclosed scope around it.
Closures help keep any state or privacy within that function.
Closures are one of the best and most powerful features of JavaScript.

```
(

function() {
   'use strict';
   // Your code here
   // All function and variables are scoped to this function
}

());
```

This pattern is well known as a **Immediately Invoked Function Expression** or IIFE. The function is evaluated then immediately invoked. Its also a good practice to run your modules in ES5 strict mode. Strict mode will protect you from some of the more dangerous parts in JavaScript.

2 **Exporting our module**
Next we will want to export our module. This basically assigns the module to a variable that we can use to call our modules methods.

```
var myModule =
(

function() {
   'use strict';
```

```
    // Your code here
    // All function and variables are scoped to this function
}

());
```

3 Next lets create a **public method for our module to call**. To expose this method to code outside our module we return an Object with the methods defined.

```
var myModule = (function() {
    'use strict';

    return {
        publicMethod: function() {
            console.log('Hello World!');
        }
    };
}());

myModule.publicMethod();    // outputs 'Hello World'
```

4Private methods & properties

JavaScript **does not have a private keyword by default but using closures we can create private methods and private state.**

An important feature of closures is that an inner function still has access to the outer function's variables.

```
var myModule = (function() {
    'use strict';

    var _privateProperty = 'Hello World';

    function _privateMethod() {
        console.log(_privateProperty);
    }

    return {
        publicMethod: function() {
            _privateMethod();
        }
    };
```

```
}());

myModule.publicMethod();                    // outputs 'Hello World'
console.log(myModule._privateProperty);     // is undefined protected by the module
closure
myModule._privateMethod();                  // is TypeError protected by the
```

==========

Revealing Module Pattern

The Revealing Module Pattern is one of the most popular ways of creating modules.
Using the return statement we can return a object literal that 'reveals' only the methods or properties we want to be publicly available.

Either way is fine var x = (function(){} **()**);

or

Either way is fine var x = (function(){})**()**;

```
var myModule = (function() {
    'use strict';

    var _privateProperty = 'Hello World';
    var publicProperty = 'I am a public property';

    function _privateMethod() {
        console.log(_privateProperty);
    }

          function publicMethod() {
          _privateMethod();
          }

    return {
        publicMethod: publicMethod,
        publicProperty: publicProperty
    };
}());
```

```
var myModule = (function() {
    'use strict';

    var _privateProperty = 'Hello World';
    var publicProperty = 'I am a public property';

    function _privateMethod() {
        console.log(_privateProperty);
    }

            function publicMethod() {
            _privateMethod();
            }

    return {
        publicMethod: publicMethod,
        publicProperty: publicProperty
    };
})();


myModule.publicMethod();                    // outputs 'Hello World'
console.log(myModule.publicProperty);       // outputs 'I am a public property'
console.log(myModule._privateProperty);     // is undefined protected by the module
closure
myModule._privateMethod();                  // is TypeError protected by the module closure
```

The benefit to the Revealing Module Pattern is that we can look at the bottom of our modules and quickly see what is publicly available for use.

JavaScript Hoisting

‹ Previous
Next ›

JavaScript Declarations are Hoisted

In JavaScript, a variable/function can be declared after it has been used.
In other words; a variable/function can be used before it has been declared.

```
x = 5; // Assign 5 to x
```

```
elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                     // Display x in the element

var x; // Declare x
```
Try it Yourself »

```
D(‘aa’)

function D(val) {
}
```