**Lección 5: Javascript IV (Objetos)**

En esta lección cubriremos:

* Introducción a los Objetos
* Métodos
* Bucles for…in
* Palabra clave this
* Objetos en Javascript

**Introducción a los Objetos**

En la anterior lección aprendimos sobre *arrays* o matrices. Las matrices son contenedores que sostienen colecciones de datos. En esta lección, introduciremos otro contenedor de datos, el *Objeto*. Los objetos y las matrices son similares en ciertas cosas, y muy diferentes en otras. Mientras que los array pueden contener múltiples elementos relacionados unos con otros, los objetos contienen mucha información sobre una sola cosa. Los objetos se instancian usando llaves ({}).

**const** nuevoObjeto **=** {};

**Pares Clave:Valor (Key:Value)**

A diferencia de las matrices que tienen elementos valorados en índices, los objetos usan un concepto llamado pares de clave:valor. La clave (*key*) es el identificador y el valor (*value*) es el valor que queremos guardar en esa clave. La sintaxis es “clave: valor”. Los objetos pueden contener muchos pares de clave-valor, deben estar separados por una coma (importante: sin punto y coma dentro de un objeto). Las claves son únicas en un objeto, solo puede haber una clave de ese nombre, aunque, varias claves pueden tener el mismo valor. Los valores pueden ser cualquier tipo de dato de Javascript; cadena, número, booleano, matriz, función o incluso otro objeto. En esta demostración crearemos un objeto usuario.

**const** user **=** {

username: 'juan.perez',

password: 'loremipsumpwd123',

lovesJavascript: **true**,

favoriteNumber: 42

};

**Acceder a los valores**

Una vez que tengamos los pares clave-valor, podemos acceder a esos valores llamando al nombre del objeto y la clave. Hay dos formas diferentes de hacer esto, usando puntos o usando corchetes.

Con la notación de puntos podemos llamar al nombre del objeto, un punto y el nombre de la clave. Así como llamamos a la propiedad .length en una matriz. La propiedad de longitud es un par de clave-valor.

user.lovesJavascript; *// true*

user.username; *// juan.perez*

La notación de corchetes es como llamar a un elemento en una matriz, aunque con corchetes debemos usar una cadena o número, o una variable que apunte a una cadena o número. Se puede llamar a cada clave envolviéndola con comillas:

**const** passString **=** 'password';

user['lovesJavascript']; *// true*

user['username']; *// juan.perez*

user[passString]; *// loremipsumpwd123*

Generalmente, verás que los corchetes casi siempre se usan con variables.

**Asignación de valores**

Asignar valores funciona igual que acceder a ellos. Podemos asignarlos, cuando creamos el objeto, con notación de puntos o con notación de corchetes:

**const** nuevoUsuario **=** {

esNuevo: **true**

}

**const** loveJSString **=** 'lovesJavascript';

nuevoUsuario.username **=** 'otro.nombre.de.usuario';

nuevoUsuario['password'] **=** '12345';

nuevoUsuario[loveJSString] **=** **true**;

**Eliminando propiedades**

Si queremos eliminar una propiedad, podemos hacerlo usando la palabra clave delete:

**const** nuevoObjeto **=** {

eliminarEstaPropiedad: **true**

};

**delete** nuevoObjeto.eliminarEstaPropiedad;

Es raro que veamos el uso de la palabra clave delete, muchos consideran que la mejor práctica es establecer el valor de una palabra clave en undefined. Dependerá de ti cuando llegue el momento.

**Métodos**

En los objetos, los valores se pueden establecer en funciones. Las funciones guardadas en un objeto se denominan métodos. Hemos utilizado muchos métodos hasta ahora a lo largo de este curso. .length, .push, .pop, etc., son todos métodos. Podemos establecer una clave para un nombre y el valor para una función. Al igual que otras veces que llamamos métodos, llamaremos a este método usando notación de puntos y paréntesis finales (Nota: podemos llamar a un método con argumentos como lo haríamos con una función normal):

**const** nuevoObjeto **=** {

decirHola: **function**() {

console.log('Hola a todo el mundo!');

}

}

nuevoObjeto.decirHola(); *//Hola a todo el mundo!*

**Bucles for…in**

A veces queremos iterar sobre cada par clave-valor en nuestro objeto. Con las matrices, utilizamos un estándar para el bucle y una variable de número de índice. Los objetos no contienen índices numéricos, por lo que el bucle estándar no funcionará para los objetos. Javascript tiene un segundo tipo de bucle for integrado llamado “*for … in loop*”. Es una sintaxis ligeramente diferente, comienza igual pero entre paréntesis declararemos una variable, la palabra clave in y el nombre del objeto. Esto recorrerá cada clave del objeto y finalizará cuando se hayan iterado todas las claves. Podemos usar esta clave, y la notación de corchetes, en nuestro bucle for para acceder al valor asociado con esa clave.

**const** usuario **=** {

username: 'juan.perez',

password: 'loremipsumpwd123',

lovesJavascript: **true**,

favoriteNumber: 42

};

**for** (**let** clave **in** usuario){

console.log(clave);

console.log(usuario[clave]);

}

*// username*

*// 'juan.perez'*

*// password*

*// 'loremipsumpwd123'*

*// lovesJavascript*

*// true*

*// favoriteNumber*

*// 42*

**La palabra clave ‘this’**

Los objetos tienen una palabra clave autorreferencial que se puede aplicar en cada objeto llamado this. Cuando se llama dentro de un objeto, se refiere a ese mismo objeto. this puede usarse para acceder a otras claves en el mismo objeto, y es especialmente útil en métodos:

**const** usuario **=** {

username: 'juan.perez',

password: 'loremipsumpwd123',

lovesJavascript: **true**,

favoriteNumber: 42,

decirHola: **function**(){

console.log( **this**.username **+** ' manda saludos!');

}

};

usuario.decirHola(); *// 'juan.perez manda saludos!'*

Nota: la palabra clave this a veces puede ser uno de los temas más difíciles en Javascript. Lo estamos usando muy básicamente aquí, pero el tema se vuelve mucho más complejo muy pronto.

**this y el Execution Context**

* **Contexto global inicial**

Este es el caso cuando ejecutamos código en el contexto global (afuera de cualquier función). En este caso this hace referencia al objeto global, en el caso del browser hace referencia a window.

*// En el browser esto es verdad:*

**>** console.log(**this** **===** window);

**<** **true**

**>** **this**.a **=** 37;

**>** console.log(window.a);

**<** 37

* **En el contexto de una función**

Cuando estamos dentro de una función, el valor de this va a depender de *cómo sea invocada la función*.

**>** **function** f1(){

**return** **this**;

}

**>** f1() **===** window;

**<** **true**

**>** window.fi() **===** window;

**<** **true**

En este ejemplo la función es invocada por el objeto global por lo tanto this hará referencia a window.

Si usamos el modo strict de Javascript, el ejemplo de arriba va a devolver undefined, ya que no le deja al interprete *asumir* que this es el objeto global.

* **Como método de un objeto**

Cuando usamos el *keyword* this dentro de una función que es un método de un objeto, this toma hace referencia al objeto sobre el cual se llamó el método:

**>** **var** o **=** {

prop: 37,

f: **function**() {

**return** **this**.prop;

}

};

**>** console.log(o.f());

**<** 37

*// this hace referencia a `o`*

En este caso, *no depende* donde hayamos definido la función, lo único que importa es que la función haya sido invocada como método de un objeto. Por ejemplo, si definimos la función afuera:

**>** **var** o **=** {prop: 37};

*// declaramos la función*

**>** **function** loguea() {

**return** **this**.prop;

}

*//agregamos la función como método del objeto `o`*

**>** o.f **=** loguea;

**>** console.log(o.f());

**<** 37

*// el resultado es le mismo!*

De todos modos, hay que tener cuidado con el keyword this, ya que pueden aparecer casos donde es contra intuitivo ( Como varias cosas de JavaScript ). Veamos el siguiente ejemplo:

**>** **var** obj **=** {

nombre: 'Objeto',

log: **function**(){

**this**.nombre **=** 'Cambiado'; *// this se refiere a este objeto, a `obj`*

console.log(**this**) *// obj*

**var** cambia **=** **function**( str ){

**this**.nombre **=** str; *// Uno esperaria que this sea `obj`*

}

cambia('Hoola!!');

console.log(**this**);

}

}

Si ejecutamos el código de arriba, vamos a notar que después de ejecutar el código, la propiedad nombre de obj contiene el valor Cambiado y no 'Hoola!!'. Esto se debe a que el keyword this dentro de la función cambia **NO hace referencia a obj**, si no que hace referencia al objeto global. No podemos considerar al this como obj porque la función no es método de este Objeto, fíjense que no podemos hacer obj.cambia. De hecho, si buscamos dentro del objeto global la variable nombre, vamos a encontrar con el valor 'Hoola!!' que seteamos con la función cambia. Esto quiere decir que no importa en donde estuvo declarada la función, si no **cómo la invocamos**.

Prácticamente, no podemos saber a ciencia cierta que valor va a tomar el keyword hasta el momento de ejecución de una función. Porque depende fuertemente de cómo haya sido ejecutada.

Para resolver este tipo de problemas existe un patrón muy común, y se basa en guardar la referencia al objeto que está en this antes de entrar a una función donde no sé a ciencia cierta que valor puede tomar this:

**var** obj **=** {

nombre: 'Objeto',

log : **function**(){

**this**.nombre **=** 'Cambiado'; *// this se refiere a este objeto, a `obj`*

console.log(**this**); *// obj*

**var** that **=** **this**; *// Guardo la referencia a this*

**var** cambia **=** **function**( str ){

that.nombre **=** str; *// Uso la referencia dentro de esta funcion*

}

cambia('Hoola!!');

console.log(**this**);

}

}

De esta forma, that (puede tener cualquier nombre) va a apuntar al objeto obj (this apuntaba a ese objeto cuando hicimos la asignación). Ahora si, podemos usar that en vez de this y estar seguros qué es lo que va a tener adentro.

**Objetos en Javascript**

En esta lección aprendimos qué son los Objetos y las muchas formas que existen para acceder a los valores, llamar a los métodos y asignar valores. Muchas de estas técnicas parecían muy familiares, como si las hubiéramos usado en prácticamente todos los aspectos de nuestros aprendizajes hasta ahora. Aquí hay un patrón, eso es porque TODO en Javascript es un Objeto. Las matrices son solo objetos con teclas numéricas, las cadenas son objetos bajo el capó con métodos incorporados, las funciones son en realidad objetos con sus propias propiedades especiales, todo el tiempo de ejecución de Javascript es un objeto (window en un navegador o global en el Node.js). Cuanto más trabajes con Javascript, más comenzará a tener sentido para ti. Solo recuerda, todo es un objeto.

**Abre la carpeta “homework” y completa la tarea descrita en el archivo README**

[Homework](https://github.com/atralice/Curso.Prep.Henry/tree/master/05-JS-IV/homework)

**Recursos adicionales**

* [MDN: Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object)
* [MDN: this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this)
* [MDN: for…in Loop](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in)