Herencia en JavaScript

Crear objetos a partir de una función constructor

Vamos a estudiar detenidamente el siguiente ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<h2>Declaring a Variable Using let</h2>

<p id="demo"></p>

<script>

function Pokemon(nombre, tipo) {
    this.nombre = nombre;
    this.tipo = tipo;
    this.damePokemon = function() {
    console.log(this.nombre + ' es de tipo ' + this.tipo);
        }

}


var charmander = new Pokemon('charmander', 'fuego')

charmander.damePokemon() // charmander es de tipo fuego

</script>

</body>
</html>
```

Con este código creamos un “constructor” para definir posteriormente objetos de “tipo” Pokemon:

```
function Pokemon(nombre, tipo) {
    this.nombre = nombre;
    this.tipo = tipo;
    this.damePokemon = function() {
    console.log(this.nombre + ' es de tipo ' + this.tipo);
        }

}
```

A continuación, mediante “new” podemos crear todos los objetos Pokemon que queramos. Los nuevos objetos creados tendrán las propiedades *nombre* y *tipo* que hemos definido en la función constructor, así como el método *damePokemon()* que también se ha definido dentro del constructor.

Crear objetos a partir de un constructor, pero con los métodos en su prototype

Para “ahorrar” espacio, es conveniente no definir los métodos que accederán a las propiedades de nuestros objetos en el propio constructor. La forma en la que se suele hacer es la siguiente:

```
<!DOCTYPE html>
<html>
<body>

<h2>Declaring a Variable Using let</h2>

<p id="demo"></p>

<script>

function Pokemon(nombre, tipo) {
    this.nombre = nombre;
    this.tipo = tipo;
}


Pokemon.prototype.damePokemon = function() {
    console.log(this.nombre + ' es de tipo ' + this.tipo);
}


var charmander = new Pokemon('charmander', 'fuego')

charmander.damePokemon() // charmander es de tipo fuego

</script>

</body>
</html>
```

Al hacerlo de esta manera, el método “damePokemon” no estará dentro de cada uno de los objetos creados con new, sino que estará dentro del prototype de Pokemon. Cuando este método sea invocado desde uno de los objetos de tipo Pokemon, primero se buscará dicho método en el propio objeto y si no se encentra, se buscará en el prototipo, es decir en Pokemon.prototype.

Herencia. Crear una nueva función constructor que herede de una función constructor ya creada.

```
<!DOCTYPE html>
<html>
<body>

<h2>Declaring a Variable Using let</h2>

<p id="demo"></p>

<script>

function Pokemon(nombre, tipo) {
    this.nombre = nombre;
    this.tipo = tipo;
    this.damePokemon = function() {
    console.log(this.nombre + ' es de tipo ' + this.tipo);
        }

}


/*Creamos un constructor para Pokedex, que hereda de Pokemon.
Tenemos que llamar al constructor de Pokemon, con call() */
function Pokedex(nombre, tipo, evolucion) {
    Pokemon.call(this, nombre, tipo);
    this.evolucion = evolucion
}



var charmander = new Pokedex('charmander', 'fuego','no')

charmander.damePokemon() // charmander es de tipo fuego

</script>

</body>
</html>
```

En el ejemplo anterior todo funciona bien, ¿pero qué pasaría si en vez de definir el método “damePokemon” dentro de la función “Pokemon” se definiera en el prototipo de “Pokemon” con Pokemon.prototype?
En ese caso, al invocar a “damePokemon” desde “charmander”, se buscaría el método “damePokemon” en el objeto “charmander”, y si no estuviera ahí se buscaría en el prototipo de “charmander”, que es “Pokedex.prototype”, pero en “Pokedex.prototype” no hay definido ningún método.

Herencia a partir de un constructor con métodos en su prototype

```
<!DOCTYPE html>
<html>
<body>

<h2>Declaring a Variable Using let</h2>

<p id="demo"></p>

<script>

function Pokemon(nombre, tipo) {
    this.nombre = nombre;
    this.tipo = tipo;
}


Pokemon.prototype.damePokemon = function() {
    console.log(this.nombre + ' es de tipo ' + this.tipo);
}

function Pokedex(nombre, tipo, evolucion) {
    Pokemon.call(this, nombre, tipo);
    this.evolucion = evolucion
}

/*El prototipo de Pokedex no tiene los métodos
que se han añadido al prototipo de Pokemon.
Por eso hay que añadir el prototype de Pokemon
al prototype de Pokedex.*/

/*Object.create() es otra forma de crear un nuevo
objeto que hereda del objeto que se le pasa como parámetro*/
Pokedex.prototype= Object.create(Pokemon.prototype);

var charmander = new Pokedex('charmander', 'fuego','no')

charmander.damePokemon() // charmander es de tipo fuego

</script>

</body>
</html>
```