Contenido

Uno de los aspectos más importantes del lenguaje Javascript es el concepto de **objeto**, puesto que prácticamente todo lo que utilizamos en Javascript, son objetos. Sin embargo, tiene ligeras diferencias con los objetos de otros lenguajes de programación, así que vamos a comenzar con una explicación sencilla y más adelante ampliaremos este tema en profundidad.

¿Qué son los objetos?

En Javascript, existe un tipo de dato llamado **object**. Una primera forma de verlo, es como una variable especial que puede contener más variables en su interior. De esta forma, tenemos la posibilidad de organizar **múltiples variables** de la misma temática en el interior de un objeto.

En muchos lenguajes de programación, para crear un objeto se utiliza la palabra clave **new**. En Javascript también se puede hacer, pero pospondremos su uso para cuando entremos en el capítulo de Programación orientada a objetos.

```
const objeto = new Object();    // Evitar esta sintaxis en Javascript (no se suele usar)
```

En Javascript, siempre que podamos, se prefiere utilizar la **notación literal**, una forma abreviada para crear objetos (*u otros tipos de datos que veremos más adelante*), sin necesidad de utilizar la palabra **new**.

Declaración de un objeto

Los literales de los objetos en Javascript son las llaves **{}**. Este ejemplo es equivalente al anterior, pero es más corto, rápido y cómodo, por lo que se aconseja declararlos siempre así:

```
const objeto = {};    // Esto es un objeto vacío
```

Sin embargo, en esta ocasión hemos creado un objeto con nombre **objeto** que está vacío. Vamos a crear un nuevo objeto llamado **player**, que contenga variables con información en su interior:

```
const player = {
  name: "Manz",
  life: 99,
  power: 10,
};
```

Estas variables dentro de los objetos se suelen denominar **propiedades**. Como se puede ver, un objeto en Javascript nos permite encapsular en su interior información relacionada rápidamente, para posteriormente poder acceder a ella de forma sencilla e intuitiva.

Propiedades de un objeto

Una vez tengamos un objeto, podemos acceder a sus propiedades de dos formas diferentes: a través de la notación con **puntos** o a través de la notación con **corchetes**.

```
// Notación con puntos (preferida)
console.log(player.name);    // Muestra "Manz"
console.log(player.life);    // Muestra 99

// Notación con corchetes
console.log(player["name"]);  // Muestra "Manz"
console.log(player["life"]);  // Muestra 99
```

El programador puede utilizar la notación que más le guste. La más utilizada en Javascript suele ser la **notación con puntos**, mientras que la notación con corchetes se suele conocer en otros lenguajes como «arrays asociativos» o «diccionarios».

A algunos programadores puede resultarles confuso utilizar objetos con la notación de corchetes, ya que en otros lenguajes de programación los objetos y los diccionarios son cosas diferentes, sin embargo en Javascript ambos conceptos se mezclan en uno solo.

OJO: Hay ciertos casos en los que sólo se puede utilizar la **notación con corchetes**, como por ejemplo cuando se utilizan espacios en el nombre de la propiedad. Es imposible hacerlo con la notación con puntos.

Añadir propiedades

También podemos añadir **propiedades** al **objeto** después de haberlo creado, y no sólo en el momento de crear el objeto. Veamos un ejemplo equivalente al anterior donde crearemos las propiedades, pero tras haber creado el objeto:

```
// FORMA 1: A través de notación con puntos
const player = {};

player.name = "Manz";
player.life = 99;
player.power = 10;

// FORMA 2: A través de notación con corchetes
const player = {};

player["name"] = "Manz";
```

```
player["life"] = 99;
player["power"] = 10;
```

Las **propiedades** del objeto pueden ser utilizadas como variables. De hecho, utilizar los objetos como elementos para organizar múltiples variables suele ser una primera buena práctica de organización en Javascript.

Métodos de un objeto

Hasta ahora, solo hemos visto como crear objetos «genéricos» **object** en Javascript. También hemos visto que es posible añadir propiedades a un objeto, que no son más que variables dentro del objeto en cuestión.

Si dentro de una variable del objeto metemos una función (*o una variable que contiene una función*), tendríamos lo que se denomina un **método de un objeto**:

```
const user = {
  name: "Manz",
  talk: function() { return "Hola"; }
};

user.name;      // Es una variable (propiedad), devuelve "Manz"
user.talk();    // Es una función (método), se ejecuta y devuelve "Hola"
```

Si alguien ya tiene experiencia en el mundo de la programación, esto le resultará muy similar a un concepto que veremos más adelante llamado Clase. De momento nos estamos saltando ese concepto, y estamos creando directamente un objeto. Más adelante abordaremos las clases y sus instancias en el apartado de Programación orientada a objetos.

El método toString

Simplemente por generar una variable de tipo **object**, esa variable «hereda» una serie de métodos que existen en cualquier variable que sea de tipo **object**. Un buen ejemplo,

sería el método **.toString()**, un método que intenta representar la información de ese objeto en un **string**.

Si creamos un objeto vacío y ejecutamos dicho método, comprobaremos que ocurre lo siguiente:

```
const objeto = {};
objeto.toString();   // Devuelve "[object Object]" (representación textual de un objeto genérico)
```

Observa que en ningún momento hemos añadido una función **.toString()** al objeto, pero aún así existe y la podemos ejecutar. Esto ocurre también con otros tipos de dato que a priori no son object, sino por ejemplo **number, boolean, regexp**.

```
const number = 42;     // Tipo Number
number.toString();     // Devuelve "42"

const booleano = true;  // Tipo Boolean
booleano.toString();   // Devuelve "true"

const regexp = /.+/;   // Tipo RegExp
booleano.toString();   // Devuelve "/.+/"
```

Al crear una variable de un determinado tipo de dato, la variable será siempre también de tipo **object**, ya que todas las variables heredan de este tipo de dato. Por lo tanto, nuestra variable tendrá:

- Los métodos que implementemos nosotros personalmente
- Los métodos heredados de su propio tipo de dato
- Los métodos heredados del tipo **object**

Observa el siguiente ejemplo:

```
const number = 42.5;
number.toString();        // Devuelve "42.5" (Método de variables de tipo Object)
number.toLocaleString();  // Devuelve "42,5" (Método de variables de tipo Object)
number.toFixed(3);        // Devuelve "42.500" (Método de variables de tipo Number)
```

Hemos definido una variable numérica, de tipo number, con el valor con decimales **42.5**. En la siguiente línea, ejecutamos el método.**toString()** un método heredado de los **object** para mostrarlo como texto. Lo mismo ocurre con el método **.toLocaleString()**. Sin embargo, el método **toFixed()** es un método heredado del tipo de dato **number** , y como el tipo de dato de la variable **number** es **number** tenemos disponible ese método.

Creando nuestro toString

Si quisieramos, podríamos crear una variable **toString** dentro de nuestro objeto, que contenga una función que muestre el texto ideal para representar nuestro objeto de información. Por ejemplo:

```
const player = {
  name: "Manz",      // Nombre del jugador
  life: 4,           // Cantidad de vida actual
  totalLife: 6,      // Máximo de vida posible
  toString: function() {
    return `${this.name} (${this.life}/${this.totalLife})`;
  }
};
```

Observa que en la función **.toString()** devolvemos un **string** que contiene el **nombre** del jugador y entre paréntesis, la **vida actual** del jugador y tras un **/** el **máximo posible** de vida que podría tener. De esta forma, el método **.toString()** representa el objeto de la mejor forma posible para este caso particular:

```
console.log("Mi jugador es " + player);      // "Mi jugador es Manz (4/6)"
```

Aquí, el **console.log()** está concatenando un **string** y un objeto **object** . Como el objeto **player** no es de tipo **string**, llama automáticamente al método **.toString()** y obtiene un **string** para representarlo.