JAVASCRIPT

CLASE 5
Material complementario

OBJETOS

CODER HOUSE

JavaScript: objetos

Objetos: definición

Los objetos son estructuras que podemos definir para agrupar valores bajo un mismo criterio, y asignarles comportamiento. Entendemos por "comportamiento" a funciones asociadas al objeto, las cuales podemos emplear para transformar los valores y estructura del mismo.

Los objetos añaden la posibilidad de codificar elementos a medida, con el objetivo de representar un conjunto de datos significativos al problema a solucionar. Pensemos que estamos desarrollando un programa tipo agenda, donde guardaremos la información de una persona. Nuestro primer enfoque puede ser definir una variable por cada valor a almacenar:

```
let nombre = "Homero";
let edad   = 39;
let calle  = "Av. Siempreviva 742";
```

Pero considerando que las variables están relacionadas entre sí, es conveniente declarar un objeto de la siguiente manera:

```
// Los variables anteriores entran relacionados entre sí, entonces mejor usamos un objeto literal
const persona1 = { nombre: "Homero", edad: 39, calle: "Av. Siempreviva 742" }
```

A esta estructura se la conoce con el nombre de ***objeto literal***, porque el/la programador/a define las propiedades y valores del objeto de forma explícita, asignando dicha estructura a un variable. Para construir esta asignación, primero tenemos que tener en cuenta la apertura de llaves {}, luego a modo de listado definimos ***identificadores*** que identifican los datos asociados a la estructura (nombre, edad y apellido). A dichos identificadores se los conoce con el nombre de **propiedades;** para

cada una se asocia un valor, pero en vez de emplear el igual (=) para asignar, empleamos los dos puntos (:) seguido del valor deseado, que puede ser de cualquier tipo (string, number, boolean, array, etcétera).

Para acceder a los valores del objeto literal asignado (como es una estructura también puede decirse referenciado), a la variable persona1 podemos emplear dos notaciones.

- **Acceso a la propiedades con punto (.):** seguido al identificador de la variables, tipeamos un punto y el nombre de la propiedad. Por ejemplo:

```
console.log(persona1.nombre);
console.log(persona1.edad);
console.log(persona1.calle);
```

- Acceso a la propiedad con corchetes []: ahora, en vez de escribir un punto seguido al nombre del identificador, incluimos los corchetes y dentro especificamos una cadena de caracteres idéntica al nombre de la propiedad buscada. Por ejemplo:

```
console.log(persona1["nombre"]);
console.log(persona1["edad"]);
console.log(persona1["calle"]);
```

Si bien podemos identificar las formas anteriores como equivalentes en la mayoría de los casos, el acceso a la propiedad por corchetes ofrece la posibilidad de que el string de referencia provenga de una variable, permitiendo un acceso por recorrido, como analizaremos más adelante al emplear ***for...in***

Constructores

Declarar objetos de forma literal puede ser viable ante el caso de requerir uno o dos objetos con propiedades similares. Pero si es necesaria una cantidad significativa de

objetos con la misma estructura y funciones asociadas, deben crearse empleando un constructor.

Un constructor es una función que permite crear un objeto. Así como usamos funciones para evitar repetir instrucciones, la función constructora es una manera de simplificar la creación de objetos de un mismo tipo.

El constructor se define siguiendo las reglas de una función, pero teniendo presente que la declaración de propiedades implica usar un prefijo para asociar los valores al objeto correctamente. Veamos un ejemplo para ver de qué se trata:

```
function Persona(nombre, edad, calle) {
    this.nombre = nombre;
    this.edad   = edad;
    this.calle  = calle;
}
```

Podemos notar que el identificador de la función comienza en mayúscula “Persona”, esto puede considerarse una convención cuando definimos constructores. Por otro lado, notamos que las propiedades nombre, edad y calle presentan el prefijo “this.” . La palabra reservada ***this*** que analizaremos más en detalle en la siguiente sección, inicialmente nos permite determinar las propiedades que conforman el objeto.

Una forma coloquial de verlo es que cuando creamos una variable usamos let identificador = valor, pero cuando creamos la propiedad de un objeto usamos this.identificador = valor.

También es importante identificar los parámetros nombre, edad y calle, que representan los valores iniciales que daremos a cada propiedad al momento de crear un objeto con esta funciòn.

Objetos: instancia

Una vez declarada una función constructora, es posible crear nuevos objetos usándola. Veamos un ejemplo de ello:

```
const persona1 = new Persona("Homero", 39, "Av. Siempreviva 742");
const persona2 = new Persona("Marge", 36, "Av. Siempreviva 742");
```

Identificamos que el constructor ***Persona***, se llama como cualquier función, pasando sus respectivos parámetros, los cuales determinan los valores iniciales asignados a las propiedades del nuevo objeto.

En lo que hay que centrarse es en la palabra reservada ***new***, término que se codifica previamente a la llamada de la función constructora para indicar que estamos creando el nuevo objeto con ese constructor. A este proceso podemos llamarlo "**instancia**" del objeto.

Por último, el objeto instanciado se asocia a una variable, para poder acceder a sus propiedades más adelante. Generalmente la variable que referencia un objeto se declara con ***const***, con la intención de no perder la asociación a lo largo del programa.

This

This (del inglés "este") es una palabra reservada que sirve para autoreferenciar, es decir, la usamos para señalar al elemento actual desde el cual estamos programando. Cuando se emplea this en una función constructora, su referencia está enlazada al nuevo objeto instanciado. Entonces, escribir ***this.nombre*** es como decir estoy trabajando con "esta" propiedad, la cual pertenece al objeto instanciado, diferenciándose de la propiedad de otro objeto del mismo tipo o estructura.

Veamos ahora un ejemplo de uso de this y cómo se marca la diferencia en la instancia de un objeto a partir de estructura de un objeto literal recibido por parámetro:

This es muy útil para asegurar que se emplean las propiedades del objeto actual.

```
function Persona(literal) {
```

```
    this.nombre = literal.nombre;
    this.edad   = literal.edad;
    this.calle  = literal.calle;
}
const persona1 = new Persona({ nombre: "Homero", edad: 39, calle:
"Av.Siempreviva 742" });
```

Como podemos observar, el ***literal*** recibido por parámetro tiene exactamente las mismas propiedades que el objeto creado mediante la función constructora, ¿entonces cómo distingo la propiedad del objeto actual del objeto recibido por parámetro? Con ***this***, ya que me permite referenciar al elemento que estoy instanciando, evitando así la ambigüedad en la asignaciòn.

Señalamos que la referencia de ***this*** depende del contexto: cuando estamos trabajando con un objeto, ***this*** referencia al propio objeto instanciado. Pero cuando usamos ***this*** fuera de cualquier bloque, referencia al contexto global; y si lo empleamos dentro de una función, su referencia dependerá de cómo la función es llamada.

Métodos y operaciones con objetos

Mencionamos al principio del documento que los objetos tienen un comportamiento, lo que implica que estos elementos poseen un conjunto de funciones asociadas para operar sobre sus propiedades.

Antes de comenzar a indagar cómo es posible definir métodos personalizados en la función constructora, indagaremos algunas cuestiones generales sobre los objetos y Javascript:

Funciones y métodos

Si bien técnicamente vamos a definir funciones para asociar comportamiento a los objetos, es importante identificar que cuando una función está vinculada a un objeto, se la conoce con el nombre de ***método***.

Para llamar a un método debemos considerar que accedemos a él mediante un objeto instanciado, lo cual implica llamarlo usando la variable de referencia. Por ejemplo, si tenemos un objeto referenciado en la variable ***persona1*** y este tiene un método ***hola***, pero emplearlo debemos escribir ***persona1.hola()***,

Métodos en objetos JavaScript

El lenguaje JavaScript cuenta con sus propios objetos por defecto, que podemos emplear para construir partes del programa. De hecho, ya usamos algunos de estos elementos sin identificarlos como objeto: los strings y las funciones son objetos en JavaScript, así como los arrays que estudiaremos más adelante.

Esta característica de objeto que tienen los elementos mencionados implica que existen propiedades y métodos asociados a ellos, que podemos utilizar para hacer operaciones. El caso más común es emplear los métodos de string para crear nuevas cadenas con cierto formato:

```
let cadena = "HOLA CODER";
//Propiedad de objeto String: Largo de la cadena.
console.log(cadena.length);
//Método de objeto String: Pasar a minúscula.
console.log(cadena.toLowerCase());
//Método de objeto String: Pasar a mayúscula.
console.log(cadena.toUpperCase());
```

Haz clic aquí para verificar todos los mètodos útiles que tiene el objeto string

Definición de métodos personalizados:

Para crear un método personalizado es necesario referenciar una función a una propiedad en la función constructora. Veamos un ejemplo de este procedimiento:

```
function Persona(nombre, edad, calle) {
    this.nombre = nombre;
    this.edad   = edad;
    this.calle  = calle;
    this.hablar = function(){ console.log("HOLA SOY "+ this.nombre)}
}
const persona1 = new Persona("Homero", 39, "Av. Siempreviva 742");
const persona2 = new Persona("Marge", 36, "Av. Siempreviva 742");
persona1.hablar();
persona2.hablar();
```

Respecto a la declaración del método personalizado en la función constructora, se deben señalar las siguientes cuestiones:

- El método personalizado se refiere a una propiedad del objeto: es necesario crear una propiedad, en este caso ***this.hablar***, a la cual le asociamos una función, definiendo así un método que se puede emplear desde todos los objetos instanciados con este constructor.
- Es posible referenciar una función anónima o una tradicional: podemos definir la función que representa el comportamiento del objeto como función anónima, declarandola en la asignación, o bien podemos detallar el nombre de una función convencional previamente creada. No obstante, para emplear el método es necesario llamarlo usando el identificador de la propiedad. En el ejemplo vemos ese uso en las últimas instrucciones: persona1.hablar() y persona2.hablar()
- Para emplear las propiedades del objeto en los mètodos es necesario usar ***this***: como observamos en ***console.log("HOLA SOY "+ this.nombre)***, para referenciar el valor de la propiedad nombre del objeto desde el cual se efectúa la llamada al

mètodo, es necesario emplear la autorreferencia con this.

Clases

Evaluamos en la sección anterior cómo podemos crear métodos personalizados, pero con la función constructora tenemos la desventaja de requerir una propiedad por cada método. Esto puede generar una complejidad de declaración ante la necesidad de trabajar con objetos con un número significativo de métodos asociados. También tenemos el problema de distinguir propiedades que representan los datos del objeto, y aquellas que son comportamientos.

Dado que el empleo de la función constructora presenta las particularidades mencionadas, desde la versión ES6 podemos emplear la notación de clase para crear objetos. Vemos a la clase como un equivalente de la función constructora, más ordenada en términos de definición. Analizamos cómo quedaría la función constructora declarada en la sección anterior como una clase:

```
class Persona{
    constructor(nombre, edad, calle) {
        this.nombre = nombre;
        this.edad   = edad;
        this.calle  = calle;
    }
    hablar(){
        console.log("HOLA SOY "+ this.nombre);
    }
}
const persona1 = new Persona("Homero", 39, "Av. Siempreviva 742");
persona1.hablar();
```

Lo primero que notamos es que la forma de instanciar un objeto es idéntica, independientemente de si se crea desde una función constructora o desde una clase. Lo

importante de las clases es que los métodos se declaran aparte, sin la necesidad de asociarlos a una propiedad en el constructor.

Esto permite una declaración más ordenada, y al momento de crear un nuevo método personalizado podemos simplemente agregarlo sin modificar el constructor de la clase, como se observa en la siguiente declaración de la clase ***Producto***, y sus dos métodos personalizado ***sumaIva()*** y ***vender()***:

```
class Producto {
    constructor(nombre, precio) {
        this.nombre   = nombre.toUpperCase();
        this.precio   = parseFloat(precio);
        this.vendido = false;
    }
    sumaIva() {
        this.precio = this.precio * 1.21;
    }
    vender() {
        this.vendido = true;
    }
}
const producto1 = new Producto("arroz", "125");
const producto2 = new Producto("fideo", "50");
producto1.sumaIva();
producto2.sumaIva();
producto1.vender();
```

Asimismo, el empleo de clase nos permite aplicar técnicas avanzadas de programación al trabajar con ciertos objetos, como el uso de herencia de clase.