PROGRAMACIÓN AVANZADA EN JAVASCRIPT

Módulo 3

OTEC PLATAFORMA 5 CHILE

MÓDULO 3

CLASE 8

Introducción

Methods in .prototype

Creando Objetos

Herencia clásica

Herencia en JavaScript

Introducción

Continuamos con los conceptos empezados en la clase anterior.

Aunque usted no lo crea, ha avanzado muchísimo y llegar a comprender estos conceptos es algo muy complejo. Así que no se frustre si tiene que leerlo una y otra vez, es parte del proceso de aprender.

Esta es la última clase de contenido contundente. Estamos llegando a la cima del aprendizaje, para luego entrar de lleno a la práctica con nuevos conceptos.

Methods in .prototype

Se habrá preguntado qué tiene que ver el objeto prototype y los constructores que hasta ahora vimos.
Cuando el constructor crea su instancia, y la retorna, le asigna una propiedad __proto__, ya que esta instancia va a ser un objeto. Esta propiedad proto va a apuntar a la propiedad prototype de la función constructora.

Entonces, inicialmente, prototype solo va a tener el objeto constructor dentro. Si agregamos nuevos métodos al prototype, la instancia creada también los podrá acceder mediante el prototype chain.

Methods in .prototype

Veamos un ejemplo:

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

// agrego al objeto prototype del constructor el método hablar
Persona.prototype.hablar = function () {
 return this.nombre + " tiene " + this.edad + " años.";
};

// en el momento de la invocación con 'new' se le asigna a 'flor' el prototype
// de la función, entonces la instancia tendrá acceso al método
var flor = new Persona("flor", 29);

flor.hablar(); // 'flor tiene 29 años.'

flor.__proto__.hablar;
```

Methods in .prototype

Mediante el objeto prototype puede asignarle métodos a las instancias. Como estas hacen referencia a prototype, los métodos no tienen que ser agregados antes de las instancias.

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

var flor = new Persona("flor", 29);

Persona.prototype.hablar = function () {
 return this.nombre + " tiene " + this.edad + " años.";
};

flor.hablar(); // 'flor tiene 29 años.'
```

Methods in .prototype

Y conociendo el prototype chain podemos agregar elementos a cualquier tipo de dato que podamos apuntar.

```
// Puedo seleccionar el constructor de "strings" por ejemplo
String.prototype.saludar = function() {
  return 'Hola soy un string, digo: ' + this;
}

'como estas?'.saludar() // 'Hola soy un string, digo: como estas?'
```

Si quisiera saber a qué prototype corresponde, podría usar el método getPrototypeOf().

```
// siguiendo el ejemplo de más arriba
Object.getPrototypeOf(facu) // Object {hablar: 'function', constructor:
'function'}
// lo que hace es mostrar el `prototype` del objeto pasado
Object.getPrototypeOf(facu) === Persona.prototype // true
```

Creando Objetos

Hasta ahora siempre que teníamos que crear un objeto nuevo declaramos un object literal. Ahora vas a ver otros métodos que brinda el prototype de Object para cumplir con esa tarea.

Constructor Object

Si quiere un objeto, puede generarlo a partir del constructor original:

```
var obj = Object()
obj // {}
```

Object.create

El método *create* de los objetos permite crear uno nuevo a partir de un prototype específico:

```
// creo un objeto con un objeto vacío como proto
var obj = Object.create({});

obj; // Object {}

// creo que un objeto a partir de un proto de Objeto
var obj =
Object.create(Object.prototype);
// que es lo mismo que crear un objeto vacío literal
var obj = {};
```

Object.assign

El método *assign* de los objetos permite agregar propiedades a un objeto pasado por parámetro:

```
var obj = {};

// No hace falta guardar el resultado
porque los objetos se pasan por
`referencia`
Object.assign(obj, { nombre: "flor",
apellido: "páez });

obj.nombre; // 'flor'
```

Herencia Clásica

En el paradigma de Programación Orientada a Objetos un tema muy importante es la Herencia y Polimorfismo de las clases (los vamos a llamar constructores por ahora).

Cuando hacemos referencia a Herencia nos referimos a la capacidad de un constructor de heredar propiedades y métodos de otro constructor.

Polimorfismo es la capacidad de que objetos distintos puedan responder a un llamado igual, de acuerdo a su propia naturaleza.

Herencia en JavaScript

En JS, a diferencia de la herencia clásica, nos manejamos con prototipos, que van a tomar los métodos pasados por sus 'padres' mediante la Prototype Chain.

Cuando generamos un arreglo nuevo podemos acceder a métodos, como map o slice, gracias a que los heredamos del Objeto Array. El mismo está vinculado en la propiedad __proto__ y es el siguiente en el Prototype Chain.

También podemos generar nuestros propios constructores:

```
function Persona(nombre, apellido,
ciudad) {
 this.nombre = nombre;
 this.apellido = apellido;
 this.ciudad = ciudad;
}

Persona.prototype.saludar = function ()
{
 console.log("Soy " + this.nombre + " de
" + this.ciudad);
};

var flor = new Persona("Flor", "Páez",
"Santiago");

flor.saludar(); // 'Soy Flor de
Santiago'
```

MÓDULO 3

GRACIAS

Aquí finaliza la clase n°8 del módulo 3

OTEC PLATAFORMA 5 CHILE