



E-Camp
Powered by Edutecno

ACTUALIZACIONES DE ECMASCRIP

Sesión 07_I



María Alejandra Canache V
Alejandro Quiñones

Prototipos

Los prototipos son un mecanismo mediante el cual los objetos en JavaScript heredan características entre sí.

JavaScript es a menudo descrito como un lenguaje basado en prototipos - para proporcionar mecanismos de herencia, los objetos pueden tener un objeto prototipo, el cual actúa como un objeto plantilla que hereda métodos y propiedades.

Prototipos

Hasta el momento declaramos un objeto por la sintaxis de Object Literal

```
const cliente = {  
    nombre: 'Jose',  
    saldo: 500  
}  
  
console.log(cliente);  
console.log( typeof cliente);
```

Por consola obtenemos...

```
▼ {nombre: 'Jose', saldo: 500} ⓘ  
  nombre: "Jose"  
  saldo: 500  
▶ [[Prototype]]: Object  


---

object
```

Prototipos

Si observamos en consola, todos los objetos tienen un proto

```
▼ [[Prototype]]: Object
  ▶ constructor: f object()
  ▶ hasOwnProperty: f hasOwnProperty()
  ▶ isPrototypeOf: f isPrototypeOf()
  ▶ propertyIsEnumerable: f propertyIsEnumerable()
  ▶ toLocaleString: f toLocalestring()
  ▶ toString: f toString()
  ▶ valueOf: f valueof()
  ▶ __defineGetter__: f __defineGetter__()
  ▶ __defineSetter__: f __defineSetter__()
  ▶ __lookupGetter__: f __LookupGetter__()
  ▶ __lookupSetter__: f __LookupSetter__()
  ▶ __proto__: Object
    ▶ constructor: f Object()
      ▶ assign: f assign()
      ▶ create: f create()
      ▶ defineProperties: f defineProperties()
      ▶ defineProperty: f defineProperty()
      ▶ entries: f entries()
      ▶ freeze: f freeze()
      ▶ fromEntries: f fromEntries()
      ▶ getOwnPropertyDescriptor: f getOwnPropertyDescriptor()
      ▶ getOwnPropertyDescriptors: f getOwnPropertyDescriptors()
```

Todos los objetos tienen ese proto, y todo lo que se encuentra dentro de ese proto, son funciones exclusivas de ese objeto.

Prototipos

Si bien la sintaxis de Object Literal es la más común, también es la menos dinámica, ya que es muy estático. Así que si requieres crear un objeto reutilizable, que puedas utilizar en muchas instancias con muchos datos, lo tendrías que realizar de otra forma. Esa se conoce como el Object Constructor

Prototipos

Object Constructor

Si bien, el resultado es semejante al anterior, esta forma es dinámica y nos permite crear múltiples instancias de diferentes clientes, por ejemplo.

```
function Cliente(nombre, saldo) {  
    this.nombre = nombre;  
    this.saldo = saldo;  
}  
  
//Creamos un nuevo objeto, una instancia de Cliente  
  
const jose = new Cliente('Jose', 500);  
console.log(jose)
```

```
▼ Cliente {nombre: 'Jose', saldo: 500} ⓘ  
  nombre: "Jose"  
  saldo: 500  
▼ [[Prototype]]: object  
  ► constructor: f Cliente(nombre, saldo)  
  ► [[Prototype]]: Object
```

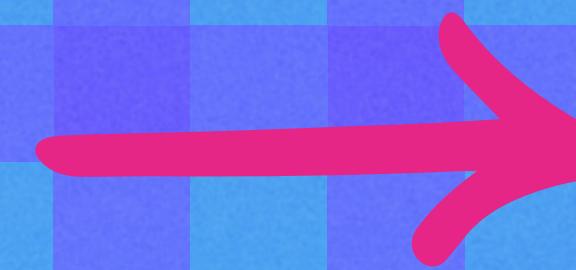
Prototipos

Esta forma de declarar objetos, es lo que anteriormente se conocía como programación orientada a objetos. A partir de ECMAScript 2015, se hace uso de las clases, basándose en la sintaxis de Object Constructor.

Prototipos

¿Cuál es el problema que soluciona el prototipo?. Ejemplo

Partimos de acá



Supongamos que queremos crear una función que muestre el nombre y el saldo.

```
function Cliente(nombre, saldo) {  
    this.nombre = nombre;  
    this.saldo = saldo;  
}
```

```
const jose = new Cliente('Jose', 500);
```

```
function formatearCliente(cliente) {  
    const {nombre, saldo} = cliente;  
    return `El Cliente ${nombre} tiene un saldo de ${saldo}`;  
}  
  
console.log(formatearCliente(jose));
```

Prototipos

¿Cuál es el problema que soluciona el prototipo?. Ejemplo

Ahora supongamos que nuestro proyecto va creciendo mas, y ahora tenemos empresas.

```
function Empresa(nombre, saldo, categoria) {  
    this.nombre = nombre;  
    this.saldo = saldo;  
    this.categoría;  
}  
  
const servicio = new Empresa('Agua Service Spa', 50000, 'servicio público');  
console.log(servicio)
```

Prototipos

¿Cuál es el problema que soluciona el prototipo?. Ejemplo

Ahora si queremos utilizar la función `formatearCliente()` con `empresa`, que ocurre... funciona pero

```
function Empresa(nombre, saldo, categoria) {  
    this.nombre = nombre;  
    this.saldo = saldo;  
    this.categoría;  
}  
  
const servicio = new Empresa('Agua Service Spa', 50000, 'servicio público');  
console.log(formatearCliente(servicio));
```

¿Qué observas?



El Cliente Jose tiene un saldo de 500

El Cliente Agua Service Spa tiene un saldo de 50000

Prototipos

¿Cuál es el problema que soluciona el prototipo?. Ejemplo

Funciona pero no muestra la categoría por lo que tendría que crear otra función para que muestre los datos completos de empresa, para que funcione.

```
function formatearEmpresa(empresas) {  
  const {nombre, saldo, categoria} = empresas;  
  return `El Cliente ${nombre} tiene un saldo de ${saldo} y tiene una categoria de ${categoria}`;  
}  
  
console.log(formatearEmpresa(servicio));
```

Prototipos

¿Cuál es el problema que soluciona el prototipo?. Ejemplo

El problema de este tipo de código es que debes ir realizando ajustes, y cuando son empresas, que son varias personas que le hacen mantenimiento al código, no saben que función deben utilizar por ejemplo para empresa y cual para cliente. Y puedes decir...bueno tenemos que documentar el código, pero si son muchas personas trabajando en el código, estarán agregando funciones y no todas las van a documentar.

Prototipos

¿Cuál es el problema que soluciona el prototipo?. Ejemplo

Eso es lo que soluciona el prototype. Porque en el prototype puedes agregar funciones que son exclusivas de clientes y otras exclusivas de empresa, por ejemplo.

Prototipos

Creando un prototype

Partimos del cliente

Accedimos al constructor

```
function Cliente(nombre, saldo) {  
    this.nombre = nombre;  
    this.saldo = saldo;  
}  
//instanciarlo  
  
const paula = new Cliente('Paula', 5000);
```

```
▼ Cliente {nombre: 'Paula', saldo: 5000} ⓘ  
  nombre: "Paula"  
  saldo: 5000  
  ▼ [[Prototype]]: Object  
    ► constructor: f Cliente(nombre, saldo)  
    ► [[Prototype]]: Object
```

Prototipos

Creando un prototype

Ahora, como le hacemos para crear mas funciones que sean exclusivas del objeto cliente? Pues debemos crear un prototype.

```
Cliente.prototype.tipoCliente = function(){
    console.log('Desde mi nuevo proto')
}
```

Prototipos

Creando un prototype

ahora si podemos llamar a la funcion tipoCliente desde cualquier objeto que sea de tipo cliente

```
 paula.tipoCliente();
```

Desde mi nuevo proto

```
▶ Cliente {nombre: 'Paula', saldo: 5000}
```

Prototipos

Creando un prototype

Ahora si revisamos empresa

```
function Empresa(nombre, saldo, categoria) {  
    this.nombre = nombre;  
    this.saldo = saldo;  
    this.categoría = categoría;  
}  
  
const servicio = new Empresa('Agua Service Spa', 50000, 'servicio público');  
console.log(servicio)
```

observamos que no tiene la función tipoCliente

```
▼ Empresa {nombre: 'Agua Service Spa', saldo: 50000, categoría: 'servicio público'} ⓘ  
  categoría: "servicio público"  
  nombre: "Agua Service Spa"  
  saldo: 50000  
▼ [[Prototype]]: Object  
  ► constructor: f Empresa(nombre, saldo, categoría)  
  ► [[Prototype]]: Object
```

Prototipos

Creando un prototype

Por lo tanto las instancias de Empresa no podrán acceder a la función tipoCliente.

Un dato interesante, es que los prototype utilizan es function y no arrow function, porque las arrow function buscaran los datos en la ventana global mientras que la function, solo buscará los datos en el bloque de código indicado.

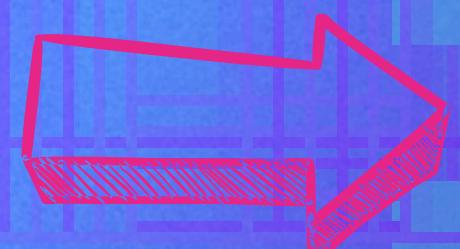
Prototipos

Creando un prototype

El prototype a través de la función puede acceder a las propiedades del objeto

```
Cliente.prototype.tipoCliente = function(){  
    console.log(this.saldo)  
}
```

Esto ocurre porque el prototype mantiene la referencia al objeto actual



5000

Prototipos

Creando un prototype

Por ejemplo, podemos evaluar que tipo de cliente es Paula, según su saldo

```
Cliente.prototype.tipoCliente = function(){  
    let tipo;  
    if(this.saldo > 10000){  
        tipo = 'Gold'  
    }else if(this.saldo > 5000) {  
        tipo = 'Silver'  
    } else {  
        tipo = 'Bronce'  
    }  
  
    return tipo;  
}  
  
console.log(paula.tipoCliente())
```

y así puedes ir creando funciones que solo se apliquen a ese objeto.

Prototipos

No solo puedes hacer referencia a propiedades de un objeto sino también a otros prototype

```
Cliente.prototype.tipoCliente = function(){
    let tipo;
    if(this.saldo > 10000){
        tipo = 'Gold'
    }else if(this.saldo > 5000) {
        tipo = 'Silver'
    } else {
        tipo = 'Bronce'
    }

    return tipo;
}

Cliente.prototype.nombreClienteSaldo = function(){
    return `El cliente ${this.nombre} tiene un saldo de ${this.saldo} y es tipo ${this.tipoCliente()}`;
}

console.log(paula.nombreClienteSaldo())
```

Prototipos

También puedes escribir prototype que tomen valores no solo parámetros

```
Cliente.prototype.retiraSaldo = function(retira) {  
    this.saldo -= retira;  
}  
  
paula.retiraSaldo(1000);  
  
console.log(paula.nombreClienteSaldo())
```

Prototipos

Crea una web para una inmobiliaria,(HTML, CSS y JS) donde el cliente pueda obtener la cotización de un seguro para la propiedad que va a comprar. Para ello debe tener la oportunidad de seleccionar la propiedad y a partir de ello, poder seleccionar el seguro básico o completo para la misma. El rango de propiedades va desde una casa hasta terrenos.

En este reto debe aplicar la creación de prototipos.

COTIZA TU SEGURO DE PROPIEDADES

PROPIEDAD: - Seleccionar -
- Seleccionar -
Casa
Departamento
Local Comercial

AÑO:

BÁSICO COMPLETO

Cotizar Seguro