



Capacitarte

Introducción a React

Unidad II

JavaScript



Contenidos

- ◉ Introducción.
- ◉ Variables y constantes.
- ◉ Operadores básicos.
- ◉ POO.
- ◉ Funciones.
- ◉ Arreglos.
- ◉ Objetos literales.
- ◉ Promesas.
- ◉ Fetch API.



Clase 2



Javascript es el **lenguaje de programación web** por excelencia.

Decimos que se trata de un lenguaje de programación **interpretado**.

Su uso más conocido es del lado del cliente (**client-side**), corriendo en el navegador web, permite mejoras en la interfaz de usuario.

React está desarrollado en JS, por eso lo llamamos **reactjs**.



¿Qué es?

- Es un lenguaje de programación basado en objetos.
- Interpretado.
- Actualmente también en Back.
- Dialecto de ECMAScript (ES).

¿Para Qué?

- Permite crear, leer, actualizar y eliminar el contenido de los documentos HTML.
- Aporta dinamismo a la UI para hacer más agradable la UX.
- Permite validar datos y verificar los mismos antes de enviarlos al servidor.
- Por qué es el lenguaje que más adeptos ha adquirido en los últimos 3 años.

Fuente (<https://insights.stackoverflow.com/survey/2018/#technology>)

- = / asignación
- + / suma
- - / resta
- */ multiplicación
- / / división
- % / módulo
- ++ / incremento
- -- / decremento
- == / igual simple
- === / igual estricto
- != / distinto
- !== / distinto estricto
- > / mayor
- < / menor
- >= / mayor o igual
- <= / menor o igual
- && / and
- || / or
- ! / negación


```
if (true) {  
    // when is true  
}
```

```
if (true) {  
    // when is true  
} else {  
    // when is false  
}
```

`(test) ? expressionA : expressionB;`

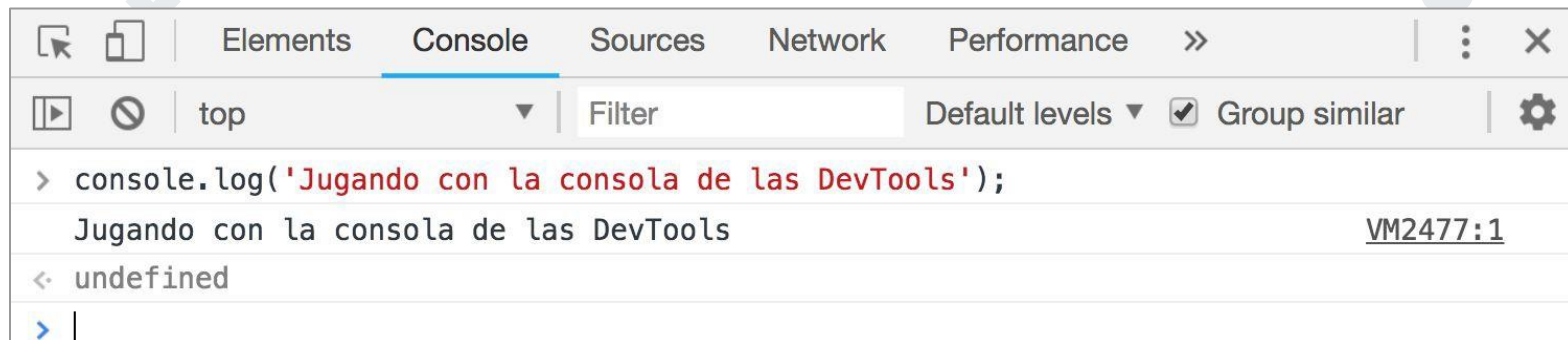
`(test)` : condición a evaluar

`expressionA` : resultado cuando es verdadero

`expressionB` : resultado cuando es falso

```
var isMember = true;  
  
(isMember) ? "$12.50" : "$17.80";  
  
// Retorna "$12.50"
```

“
La mejor manera es usando las
DevTools y el console.log()



```
var isMember = true;  
  
var price = (isMember) ? "$12.50" : "$17.80";  
  
console.log("Precio a cobrar: " + price);  
  
// Precio a cobrar: 12.50 (en la consola)
```

//

¿Cómo vinculamos JS con HTML?

```
<body>  
  ...  
  <script>  
    var mySuperHeroe = "Ada Lovelace";  
    console.log(mySuperHeroe); // "Ada Lovelace"  
  </script>  
</body>
```

```
<body>
```

```
...
```

```
<script src="js/main.js"></script>
```

```
</body>
```



```
// main.js
```

```
window.onload = function () {  
    var mySuperHeroe = "Ada Lovelace";  
    console.log(mySuperHeroe); // "Ada Lovelace"  
}
```

// **onload**, es un evento que permite que todo el script se ejecute cuando se haya cargado por completo el objeto **document** dentro del objeto **window**. En unas clases más adelante profundizaremos sobre los eventos.

Let

Podemos utilizar `let` en lugar de `var` a la hora de declarar una variable, cuando queremos que esta solo sea accedida de manera local en determinado ámbito. Por ejemplo:

```
55  
56 //ES6  
57 let nombre1 = 'Javascript';  
58 let nombre2 = 'awesome';  
59 console.log('Solo quiero decir que ${nombre1} es ${nombre2}');  
60 // Solo quiero decir que Javascript es awesome  
61  
62
```

Función Arrow

```
//ES6
var miFuncion = (num) => num + num;

//ES6
var data = [{...}, {...}, {...}];
data.forEach(elem => {
  console.log(elem);
})
```

En ambos ejemplos se sustituye el uso de la palabra reservada **function** dando simplicidad al código.

(parametro1,parametro2,...,parámetro n) => {Definición de la función}

Clases

```
class Animal {  
  constructor(sonido = 'ola-k-ac') {  
    this.sonidoPrimitivo = sonido;  
  }  
  
  emitirSonido() {  
    console.log(this.sonidoPrimitivo);  
  }  
}
```

```
class Persona extends Animal {  
  constructor(altura) {  
    this.altura = 1.20  
    super('Hola! Soy una persona!');  
  }  
  
  hablar() {  
    return this.emitirSonido();  
  }  
  
  getAltura() {  
    return this.altura;  
  }  
  
  correr() {  
    console.log('Hey! Vamo a correr');  
  }  
}
```

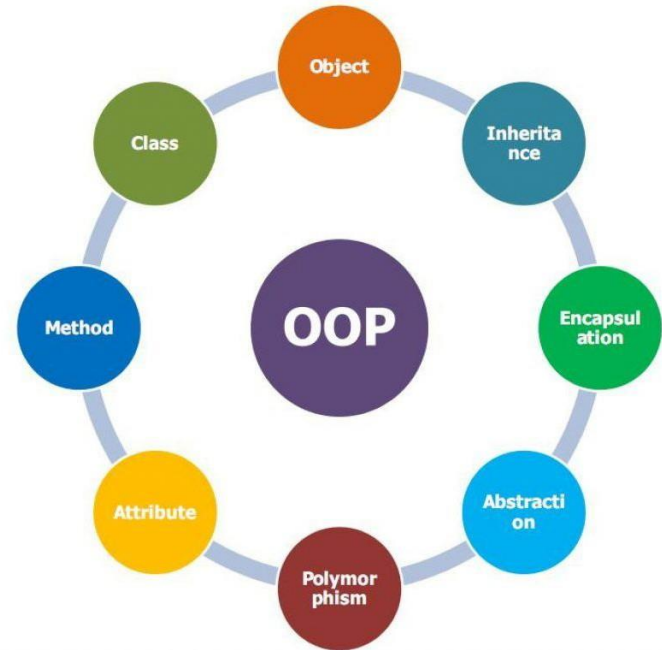
JavaScript también tiene clases. Estas son muy parecidas a las funciones constructoras de objetos que realizamos en el estándar anterior, pero ahora bajo el paradigma de clases, con todo lo que eso conlleva (ejemplo: herencia)



A programar!!!

Introducción a la POO

Vamos a ir familiarizándonos con todos estos conceptos.



Introducción a la POO

Assembler: Programación a muy **bajo nivel**. Sobre el circuito de la máquina.

Programación secuencial/estructurada: Programación en la que una instrucción sigue a la anterior, y dónde el programador debe manejar y controlar todos los procesos “a mano”.

Todo el código está en el mismo lugar. Corregir un error, significa rever y volver a testear todo el código.

Introducción a la POO

Programación procedimental: Se extrapola de la programación secuencial que muchas veces se repiten líneas de código. Se extraen estas repeticiones en procedimientos y funciones. Corregir un error puede ser más sencillo al tener “partes” más reconocibles en el código. Pero de todas formas el código se ve como un todo.

Módulos: Aparecen agrupaciones de procedimientos y funciones. Esto presenta algún grado de abstracción.

Se “modulariza” aún más y mejora el tema de corrección de errores y testeos.




Introducción a la POO

Programación Orientada a Objetos - Cambia el paradigma.

- En este paradigma, se plantea dividir el problema a resolver, en problemas menores, que se representarán con Objetos.
- Estos Objetos tendrán relación a elementos del contexto real del problema a resolver.
- A estos Objetos se los dotará de características y de responsabilidades para que puedan operar y responder a la necesidad particular que nos interese.
- Estos Objetos podrán interactuar entre ellos.

Introducción a la POO

Cuando organicemos/pensemos/diseñemos el código según el paradigma Orientado a Objetos, el resultado en el momento de ejecución será el mismo que de otra forma, pero el código será:

- » **mucho más mantenible**
 - » **legible**
 - » **escalable**
 - » **fácil de corregir y actualizar.**
- 
- Decorative orange wavy lines at the bottom of the slide.

Introducción a la POO

En un ejemplo de un sistema para una **Veterinaria**. Seguramente identificaré objetos como **AlimentoBalanceado**, **AnimalDomestico**, **Perro**, **Gato**, **Canario**, **Empleado**, **Cliente**, **Veterinario**...

Estos Objetos podrán relacionarse entre si.

Por ejemplo el **Veterinario**, podrá alimentar al **Perro**.

Decorative orange wavy lines at the bottom of the slide.

Introducción a la POO

Estos **Objetos** podrán hacer cosas (porque tendrán **responsabilidades**), y podrán responder según su Estado.

Por ejemplo, un **Canario**, “podrá decirnos su nombre”, porque tendrá un *String nombre* = “Tweety” como una de sus características.

Corrección de Errores / Testeo

Con este diseño, por ejemplo si tengo algún inconveniente con el funcionamiento de mis Gatos. Sabré exactamente en qué código ir a buscar. No tendré que modificar (y volver a testear), el código de los otros Objetos.

- OBJETOS LITERALES

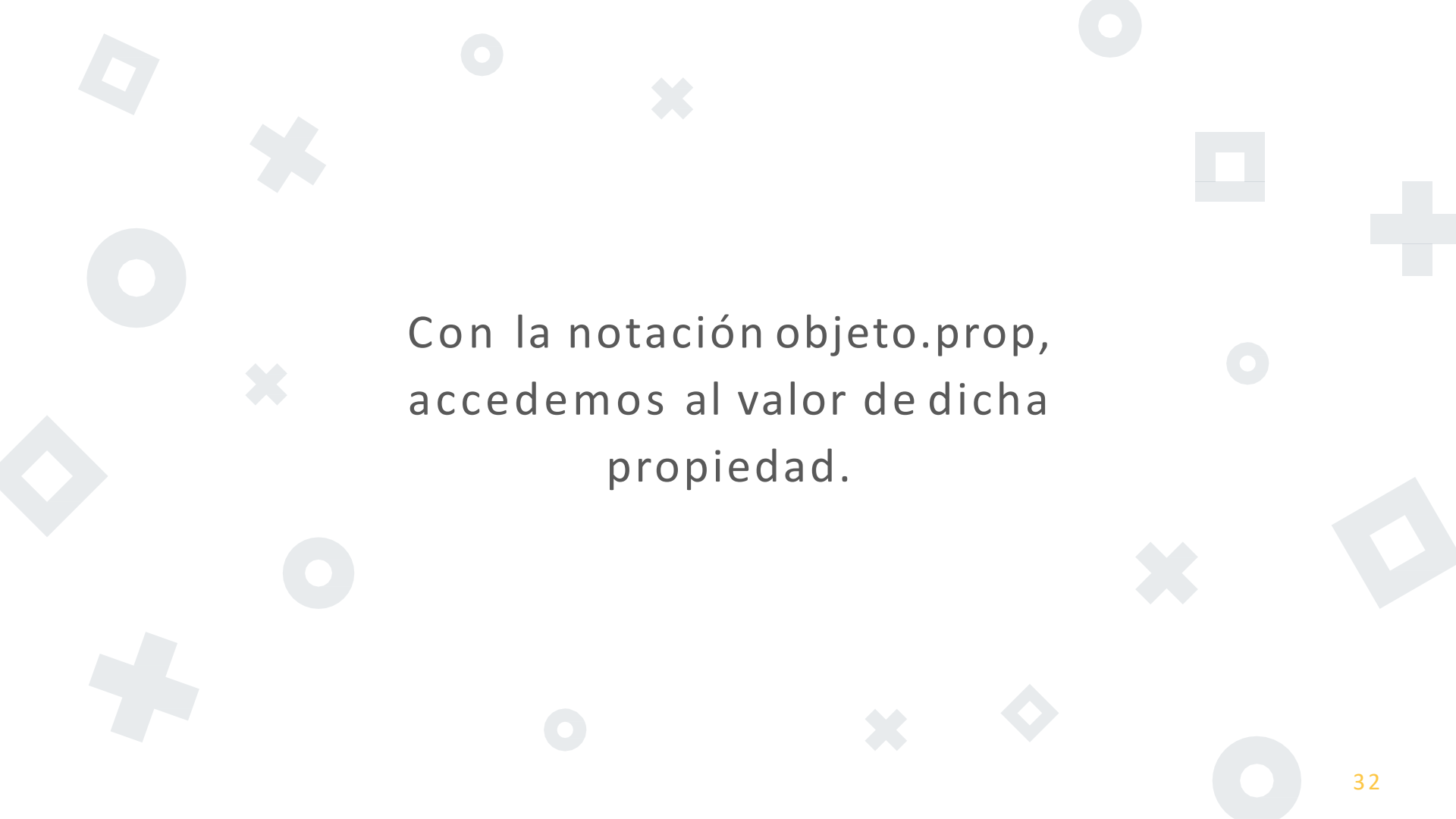
Representación en código de un
elemento de la vida real

¿Qué es?

Un objeto de JavaScript es un bloque de código que tiene propiedades, las cuales a su vez tienen un valor determinado.

Si una propiedad recibe como valor una función, a esto lo llamamos un método.

- `var student =`
 `{ name:`
 `"Juana",`
 `lastName:`
 `"Heinz",`
 `}`



Con la notación `objeto.prop`,
accedemos al valor de dicha
propiedad.


```
var student = {  
  name: "Juana",  
  lastName: "Heinz",  
}
```

```
console.log(student.name); // "Juana"
```



¿Cómo se ven los métodos
de un objeto?

```
var student = {  
  name: "Juana",  
  lastName: "Heinz",  
  fullName: function () {  
    return student.name + " " + student.lastName;  
  }  
}
```



¿Cómo podemos ejecutar
los métodos?

```
var student = {  
  name: "Juana",  
  lastName: "Heinz",  
  fullName: function () {  
    return student.name + " " + student.lastName;  
  }  
}  
console.log(student.fullName()); // "Juana Heinz"
```

DOM

DOM (Document Object Model o modelo de objetos del navegador) nos sirve para **acceder** a cualquiera de los **componentes** que hay dentro de una página.

Por medio del DOM podremos **controlar al navegador** en general y a los distintos elementos que se encuentran en la página.

¿Qué hace JS con el D.O.M.?

- Modificar elementos, atributos y estilos de una página.
- Borrar cualquier elemento y atributo.
- Agregar nuevos elementos o atributos.
- Reaccionar a todos los eventos HTML de la página.
- Crear nuevos eventos HTML en la página.

window

El objeto ***window*** es lo **primero** que se carga en el navegador. El objeto *window* tiene propiedades como *length*, *innerWidth*, etc.

document

El objeto ***document*** representa al html, php u otro documento que será cargado en el navegador. El *document* es cargado dentro del objeto *window* y tiene propiedades como *title*, *url*, *cookie*, etc.

¿Cómo capturamos a los
elementos existentes en elHTML?

Para acceder a los elementos de una página utilizamos selectores.

Cada selector puede retornar un solo elemento o una lista de elementos.

Algunos selectores básicos son (*reciben un string como parámetro*):

- `document.querySelector();`
- `document.querySelectorAll();`
- `document.getElementById();`

```
<body>  
  <h1 class="title">¡Hola mundo!</h1>  
</body>
```

```
// main.js  
var titulo = document.querySelector(".title");
```



¿Cuál es el objetivo de capturar a algún elemento de HTML?

Lo primero que podríamos hacer sería cambiar los estilos CSS de un elemento a nuestro antojo.

```
// main.js
```

```
var titulo = document.querySelector(".title");  
titulo.style.color = "pink";  
titulo.style.textAlign = "center";  
titulo.style.fontSize = "23px";  
titulo.style.backgroundColor = "#f2f2f2";
```

Gracias!



¿Consultas?