

# JAVASCRIPT



**JS**

# Contenido

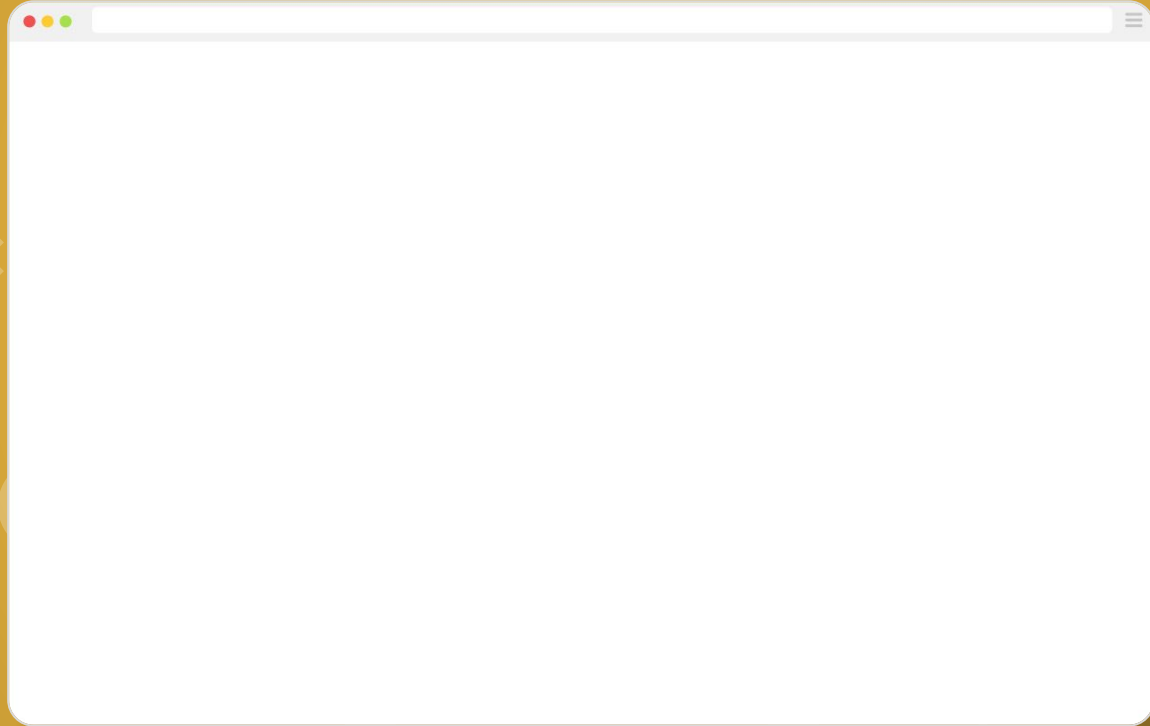
## ★ Objeto Window

- alert()
- prompt()
- confirm()
- location()
- history()

## ★ D.O.M - Vol. 2

- parentElement
- innerText vs innerHtml
- children
- createElement()
- append()
- removeChild()
- Attributes [set & get & has]
- classList()

# Objeto Window





El objeto **window** es soportado por todos los navegadores. Representa la ventana del navegador.

Los objetos, funciones y variables de Javascript que son **globales**, automáticamente se convierten en propiedades y métodos de éste objeto.

# Algunos métodos del objeto window

## ★ alert()

Permite generar un popup que alerta al usuario sobre algo. **Detiene la ejecución del script**. Recibe como parámetro un ***String***. No tiene valor de retorno.

```
window.alert('Hi everyone');
```

```
// ó
```

```
alert('Hi everyone');
```

# Algunos métodos del objeto window

## ★ prompt()

Permite generar un popup que solicita información al usuario (input). **Detiene la ejecución del script.** Recibe como parámetro un ***String***. Retorna el valor del campo.

```
window.prompt('Please. Tell me your name');
```

```
// ó
```

```
prompt('Please. Tell me your name');
```

# Algunos métodos del objeto window

## ★ confirm()

Permite generar un popup que solicita confirmación del usuario (button). **Detiene la ejecución del script.** Recibe como parámetro un ***String***. Retorna un booleano.

```
window.confirm('Did you read the terms?');
```

```
// ó
```

```
confirm('Did you read the terms?');
```

# Algunos métodos del objeto window

## ★ location()

Location es un objeto que **representa la URL**. Por medio de sus propiedades podemos acceder a las distintas partes de la URL como también navegar a otro documento.

```
window.location // Retorna todo el objeto
```

```
location.href // Retorna la URL
```

```
location.href = 'https://youtube.com' // Redirección a Youtube
```



# Algunos métodos del objeto window

## ★ location()

`location.host` // Retorna el dominio. Ej: `www.something.com`

`location.pathname` // Retorna la ruta del recurso. Ej. `about/we.php`

`location.search` // Retorna el query string

# Algunos métodos del objeto window

## ★ history()

El objeto **history** representa el historial de documentos visitados en la actual sesión del browser.

```
history.length; // Retorna la cantidad de ítems del historial
```

```
history.back(); // Vuelve al documento anterior
```

```
history.forward(); // Va al documento siguiente
```

```
history.go(i); // vuelve (-i) ó avanza en el historial (i)
```

**¡A practicar!**

Ejercicios "window"

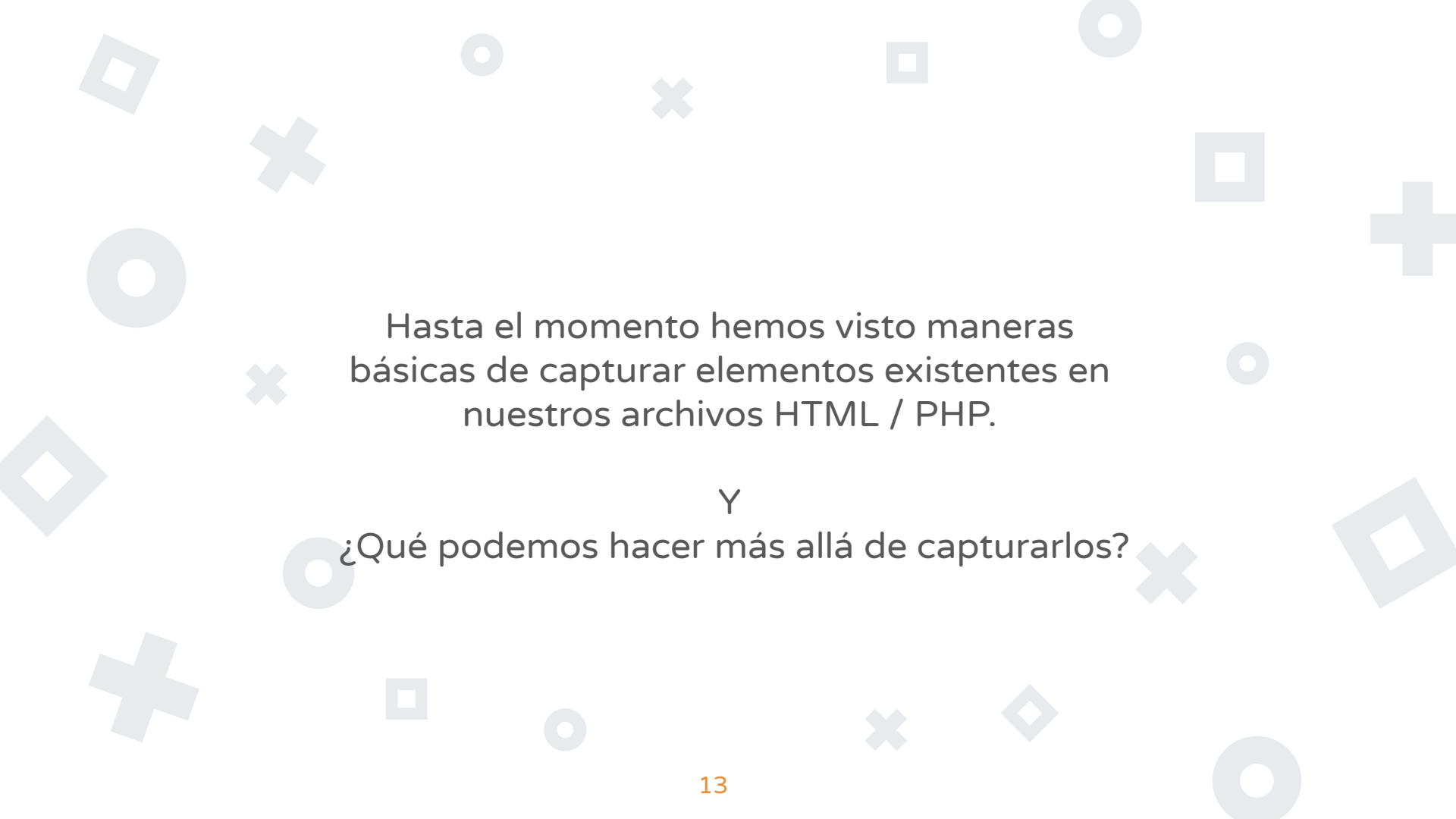


```
// To Do  
console.log("Practice Time");
```



# D.O.M - Vol. 2

Manipulación a fondo de los elementos  
del documento



Hasta el momento hemos visto maneras  
básicas de capturar elementos existentes en  
nuestros archivos HTML / PHP.

Y

¿Qué podemos hacer más allá de capturarlos?

# Obtener al padre de un elemento

## ★ parentElement

Este método, **retorna el elemento padre** de un elemento hijo.

```
// html
```

```
<p> <a href="http://google.com">Click on me</a> </p>
```

```
// javascript
```

```
var btn = document.querySelector('a');
```

```
var btnParent = btn.parentElement; // Retorna el <p> y su contenido
```

# Obtener o modificar el contenido

## ★ innerText vs innerHTML

**innerText.** Sirve como **get** y **set**. Retorna el texto plano dentro de un elemento

```
// html
```

```
<p> <a href="http://google.com"> <em>Click on me</em> </a> </p>
```

```
// javascript
```

```
var btn = document.querySelector('a');
```

```
btn.innerText // Retorna "Click on me"
```

```
btn.innerText = 'Venga, ¡dame clic!'
```

# Obtener o modificar el contenido

## ★ innerText vs innerHTML

**innerHTML.** Sirve como **get** y **set**. Retorna el contenido HTML dentro de un elemento

```
// html
```

```
<p> <a href="http://google.com"> <em>Click on me</em> </a> </p>
```

```
// javascript
```

```
var btn = document.querySelector('a');
```

```
btn.innerHTML // Retorna "<em>Click on me</em>"
```

```
btn.innerHTML = '<strong>CLICK CLICK</strong>' // Retorna el nuevo elemento.
```

```
btn.parentElement.innerHTML = '<strong>CLICK CLICK</strong>' // ¿?
```



# Obtener todos los hijos de un elemento

## ★ children

El método **children** nos permite capturar una colección de elementos HTML que será igual a los hijos directos del objeto capturado.

```
// html
```

```
<ul>
```

```
  <li>Item 01</li>
```

```
  <li>Item 02</li>
```

```
  <li>Item 03</li>
```

```
</ul>
```

```
// javascript
```

```
var lista = document.querySelector('ul');
```

```
var hijosLista = lista.children;
```

```
console.log(hijosLista); // colección HTML
```

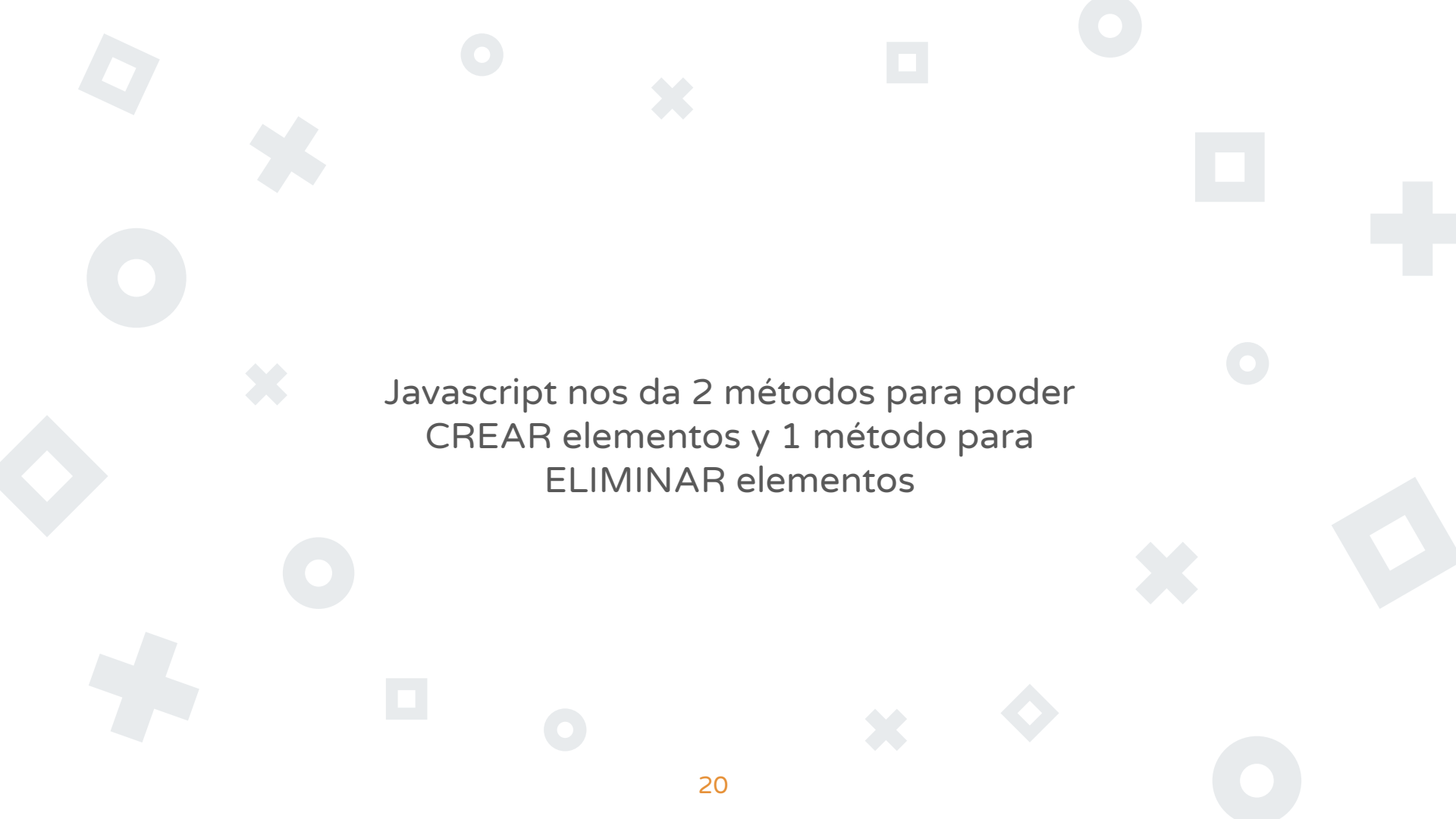
**¡A practicar!**

Ejercicios D.O.M.



```
// To Do  
console.log("Practice Time");
```





Javascript nos da 2 métodos para poder  
CREAR elementos y 1 método para  
ELIMINAR elementos

# Crear elementos

## ★ createElement()

El método **createElement** nos permite crear nuevos elementos HTML. Recibe como parámetro un ***String*** con el nombre de una etiqueta de HTML (a, div, span, li, ul, etc).

```
var lista = document.querySelector('ul');
```

```
var nuevoLi = document.createElement('li');
```

# Insertando elementos creados

## ★ `append()`

El método **`append()`** nos permite insertar el elemento creado, dentro de un elemento previamente capturado.

```
var lista = document.querySelector('ul');
```

```
var nuevoLi = document.createElement('li');
```

```
nuevoLi.innerText = 'Soy un nuevo LI :D';
```

```
lista.append(nuevoLi);
```

# Eliminando elementos

## ★ removeChild()

El método **removeChild()** nos permite eliminar un hijo dentro de un elemento previamente capturado. Recibe como parámetro el **nodo** que queremos eliminar.

```
var lista = document.querySelector('ul');  
  
var nodosHijos = lista.children; // Colección nodos HTML  
var nodoABorrar = nodosHijos[2]; // Elemento HTML  
  
lista.removeChild(nodoABorrar);
```

Así como podemos crear o eliminar elementos del HTML, también podemos manipular los **atributos** de los mismos.



# Seteando un atributo y su valor

## ★ `setAttribute()`

`setAttribute()` nos permite setear un atributo para un elemento. Recibe dos parámetros *String*. Nombre de atributo y valor.

```
var lista = document.querySelector('ul');
```

```
lista.setAttribute('class', 'my-list');
```

```
lista.setAttribute('id', 'main-list');
```

# Obteniendo el valor de un atributo

## ★ `getAttribute()`

`getAttribute()` nos retorna el valor del atributo por el cual estamos consultando.

```
var lista = document.querySelector('ul');
```

```
lista.getAttribute('class'); // Retorna "my-list"
```

```
lista.getAttribute('title'); // Retorna null
```

# Eliminando un atributo por completo

## ★ `removeAttribute()`

`removeAttribute()` elimina por completo un atributo y su(s) valor(es) existente(s).

```
var lista = document.querySelector('ul');
```

```
lista.removeAttribute('class'); // Elimina el atributo class
```

# Preguntando si existe un atributo

## ★ `hasAttribute()`

`hasAttribute()` recibe como parámetro el atributo por el que queremos preguntar. Retorna un booleano.

```
var lista = document.querySelector('ul');
```

```
lista.hasAttribute('class'); // false
```

```
lista.hasAttribute('id'); // true
```

# Preguntando si el elemento posee hijos

## ★ hasChildNodes()

**hasChildNodes()** no recibe parámetros y nos retorna un booleano si el elemento posee hijos directos.

```
// html
```

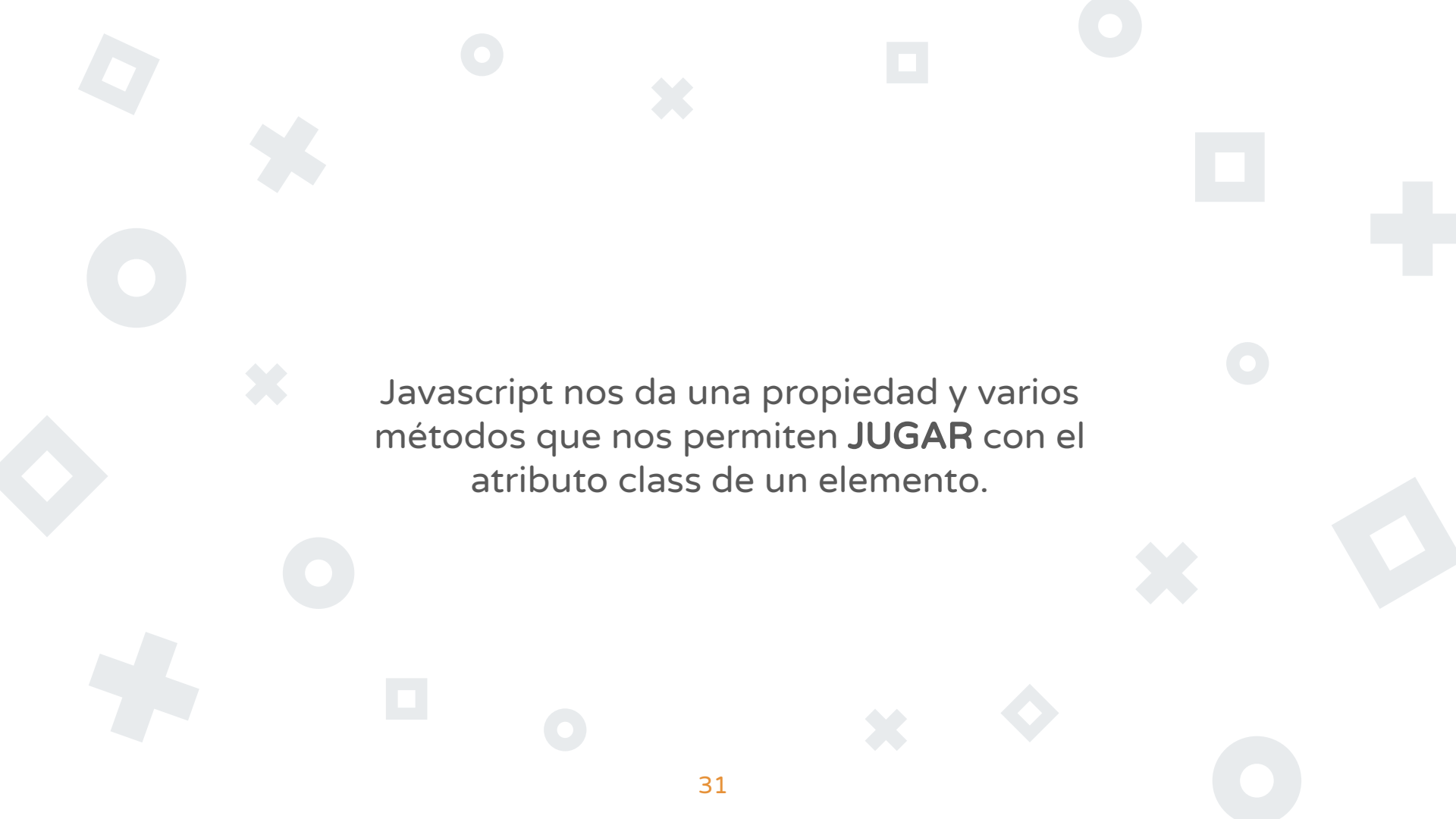
```
<ul>
  <li>Item 01</li>
  <li>Item 02</li>
  <li>Item 03</li>
</ul>
```

```
// javascript
```

```
var lista = document.querySelector('ul');

lista.hasChildNodes(); // true
```



The background of the slide is white and decorated with various light gray geometric shapes scattered across the surface. These shapes include squares, circles, and crosses, some of which are solid and others are hollow. The shapes are of different sizes and are oriented in various directions, creating a playful and abstract pattern.

Javascript nos da una propiedad y varios métodos que nos permiten **JUGAR** con el atributo class de un elemento.

# Manipulando el atributo class

## ★ `element.classList.add()`

`element.classList.add()` nos permite pasar como parámetro el nombre de clase que queremos agregar.

```
// html
```

```
<ul> ... </ul>
```

```
// javascript
```

```
var lista = document.querySelector('ul');
```

```
lista.classList.add('nueva-clase');
```



# Manipulando el atributo class

## ★ `element.classList.remove()`

`element.classList.remove()` no permite pasar como parámetro el nombre de clase que queremos eliminar.

```
// html
```

```
<ul class="clase-base nueva-clase"> ... </ul>
```

```
// javascript
```

```
var lista = document.querySelector('ul');
```

```
lista.classList.remove('clase-base');
```

# Manipulando el atributo class

## ★ `element.classList.toggle()`

`element.classList.toggle()` nos permite alternar la aplicación de una clase determinada. Va muy de la mano con el evento **onclick**.

```
// html
<ul class="nueva-clase"> ... </ul>

// javascript
btn.onclick = function () {
    lista.classList.toggle('visible');
}
```

# Manipulando el atributo class

## ★ `element.classList.contains()`

`element.classList.contains()` recibe como parámetro el valor por el que queremos consultar. Retorna un booleano.

```
// html
```

```
<ul class="nueva-clase"> ... </ul>
```

```
// javascript
```

```
if ( lista.classList.contains('nueva-clase') ) {  
    // do some stuff  
}
```

**¡A practicar!**

Ejercicio Integrador



```
// To Do  
console.log("Practice Time");
```