



JavaScript para React





Importante

En este módulo vamos a ver sólo las características mínimas de JavaScript que tenemos que saber para poder aprender React. JavaScript es un lenguaje que nos brinda muchas más funcionalidades y un funcionamiento interno en el que no vamos a entrar en profundidad

1

Sentencias vs Expresiones

Cuando trabajamos con React, estamos permitidos a utilizar expresiones dentro de JSX, pero no así sentencias. Por lo tanto es muy importante tener clara la diferencia

Sentencias vs Expresiones

Expresiones	Sentencia
<p>Porción de código de JavaScript que produce un valor.</p> <p>Una expresión pueda contener expresiones</p>	<p>Instrucción para que la computadora haga algo en particular</p> <p>Contienen “espacios” para expresiones, por ejemplo al guardar un valor en una variable</p>

2

Interpolación de strings

Interpolación de strings

- ▶ Utilizando el operador “+”
- ▶ Utilizando template strings “\${}”
- ▶ Dentro del espacio dinámico podemos utilizar cualquier expresión de JavaScript

3

Funciones

Podemos definirlas de distintas maneras

Cómo definir funciones

- ▶ Declaracion
- ▶ Expresion
- ▶ Arrow function
- ▶ Método de un objeto
- ▶ Constructores

4

Object Destructuring

Nos permite extraer propiedades de
objetos de una manera más “limpia”

5

Property Value Shorthand

Cuando creamos un objeto, podemos omitir el valor, si comparten el mismo nombre con la propiedad.

El resultado final es el mismo, pero un poco más conciso.

6

Array Destructuring

Nos permite sacar un elemento de un array y asignarlo en una variable.

7

Verdadero o falso?

Valores “falsos” o “falsy” values

- ▶ false
- ▶ NaN
- ▶ null
- ▶ undefined
- ▶ "" (string vacío)
- ▶ 0

Todo valor que no se encuentre en esta lista es un valor verdadero o “truthy” value

Convirtiendo valores a Booleanos

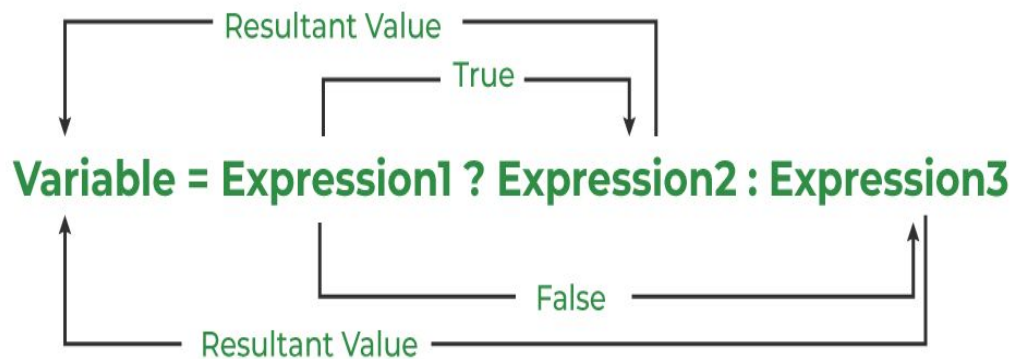
- ▶ Boolean()
- ▶ ! (operador NOT)
- ▶ !! (apilando el operador NOT)

8

Operadores lógicos

Ternarios

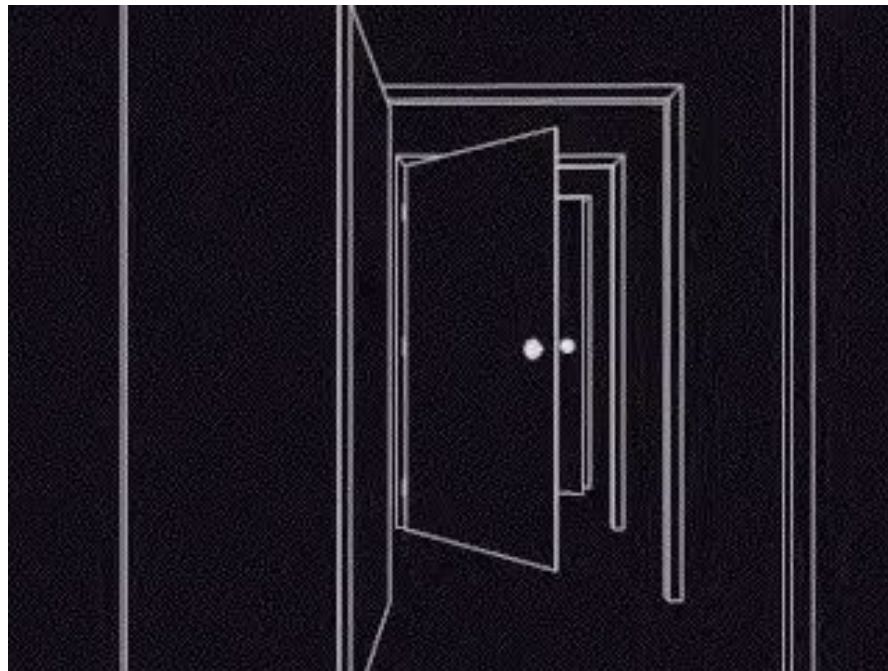
El operador ternario es el único operador en JavaScript que tiene tres operandos. Una condición seguida por un signo de preguntas (?), luego una expresión a ejecutar si la condición es verdadera seguida por dos puntos (:), y finalmente una expresión a ejecutarse si la condición es falsa. Este operador es frecuentemente usado como una alternativa a la sentencia if...else



AND (&&)

El operador lógico AND (&&) será true para un conjunto de operandos booleanos si y solo si todos los operandos son true. En caso contrario, será false.

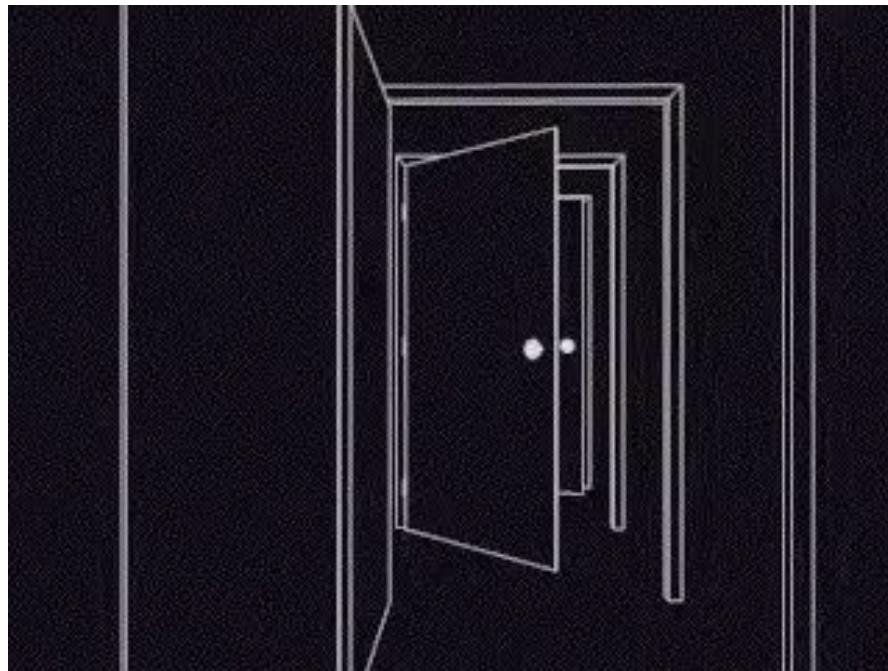
El operador retorna el valor del primer operando falsy encontrado cuando evalúa de izquierda a derecha, o el valor del último operando si todos ellos son truthy.



OR (II)

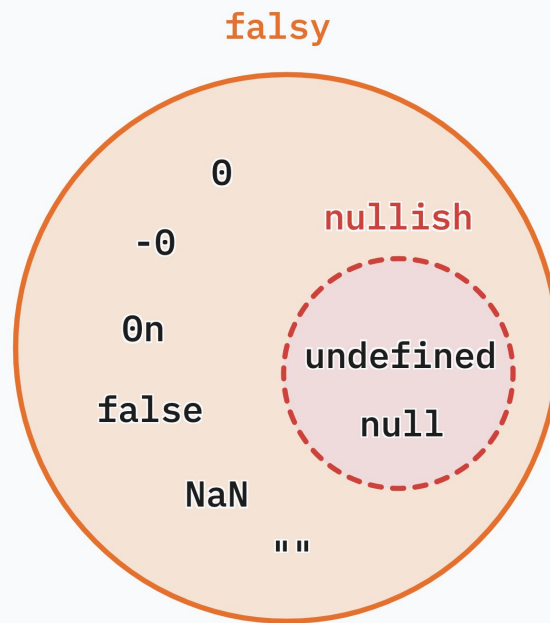
El operador lógico OR (||) es verdadero para un conjunto de operandos si y solo si uno o más de sus operandos es verdadero.

El operador retorna el valor del primer operando truthy encontrado cuando evalúa de izquierda a derecha, o el valor del último operando si todos ellos son falsy.



Nullish coalescing operator (??)

El operador ?? es un operador lógico que retorna el operando de lado derecho cuando el operando de lado izquierdo es null o undefined, y en caso contrario retorna el operando de lado izquierdo.



Optional chaining

El operador Optional Chaining permite leer el valor de una propiedad ubicada dentro de una cadena de objetos conectados sin tener que validar expresamente que cada referencia en la cadena sea válida.

Sirve para que si una referencia es nula, la expresión hace una evaluación de circuito corto con un valor de retorno de undefined.



9

Asignar vs Mutar

Asignar Vs Mutar

Para comprender esta aparente contradicción que vemos a la derecha, es importante aprender sobre la diferencia entre asignar y mutar. Es un concepto clave en JavaScript, y muchos temas del lenguaje van a tener sentido cuando tengamos una distinción clara de qué significa cada una.

```
1  const nombre = 'Juan';
2  nombre = 'Ivan'; // TypeError: Assignment to constant variable.
3  console.log(nombre);
4
5  // Pero con objetos...
6  const persona = {
7    nombre: 'Juan',
8  };
9  persona.nombre = 'Ivan';
10 console.log(persona.nombre);
11 // Cómo es posible si usamos const? 🤔
```

Variables como etiquetas

```
// Lo creamos ahora...  
const frutas = ['pera', 'manzana', 'banana'];  
  
// ...y luego podemos referenciarlo:  
console.log(frutas);  
// -> ['pera', 'manzana', 'banana']
```

Reasignando nuestras etiquetas

Al usar `let`, podemos cambiar a donde apunta nuestra etiqueta.

Tengan en cuenta como al reasignar lo que se mueve es la flecha, esto se debe a que no estamos cambiando el valor de nuestra variable. Lo que estamos haciendo es cambiar hacia donde apunta (la referencia)

let
frutas → ['manzana', 'banana', 'anana']
 ['manzana', 'banana']
 88
 undefined

Reasignando nuestras etiquetas

Al usar `const`, no tenemos permitida esta acción



Mutando

Pero acá esta la clave, si bien no podemos cambiar a donde apunta nuestra etiqueta, si podemos cambiar la data en si.

Por ejemplo en un array, podemos agregar/eliminar elementos.

En un objeto podemos editar las claves o valores.

A esto se lo conoce como mutación



Asignar vs Mutar

Asignar	Mutar
Apuntar el nombre de una variable a un valor	<p>Editar la data que existe dentro (solo en valores no primitivos como objetos y arrays)</p> <p>Los valores primitivos son inmutables, es decir no se pueden cambiar</p>

Esto explica el ejemplo que vimos al principio ;)

10

Rest / Spread

Rest

La sintaxis de los parámetros rest nos permiten representar un número indefinido de argumentos como un array.

```
function sum(...rest) {  
  let total = 0;  
  for (const arg of rest) {  
    total += arg;  
  }  
  return total;  
}
```

Spread

La sintaxis spread permite a un elemento iterable tal como un arreglo o cadena ser expandido en lugares, para llamadas de función o elementos para arrays, o a un objeto ser expandido en lugares donde cero o más pares de valores clave son esperados.

```
function sum(x, y, z) {  
  return x + y + z;  
}  
  
const numbers = [1, 2, 3];  
console.log(sum(...numbers)); // Expected output: 6  
// El equivalente seria hacer esto  
sum(numbers[0], numbers[1], numbers[2]);
```

Spread casos de uso

- ▶ Hacer la copia de un array u objeto
- ▶ Extender un objeto existente en un nuevo objeto con nuevas propiedades
- ▶ Combinar diferentes arrays u objetos

11

Métodos de Arrays

[🍎, 🍎, 🍎, 🍎].map(🍎 -> 🍌) → [🍌, 🍌, 🍌, 🍌]

[🍌, 🍎, 🍌, 🍎].filter(🍎) → [🍎, 🍎]

[🍌, 🍎, 🍌, 🍎].find(🍎) → 🍎

[🍌, 🍎, 🍌, 🍎].includes(🍍) → false

[🍌, 🍎, 🍌, 🍎].some(🍎) → true




[🍌, 🍎, 🍌, 🍎].every(🍎) → false

12

Modules

Modules

Cada archivo se convierte en un “module”. Un module es un archivo de JavaScript que exporta cierto código para ser utilizado en otros archivos (otros modules).

Module Cheatsheet	
Name Export	Name Import
<pre>export const name = 'value'</pre>	<pre>import { name } from '...'</pre>
Default Export	Default Import
<pre>export default 'value'</pre>	<pre>import anyName from '...'</pre>
Rename Export	Name Import
<pre>export { name as newName }</pre>	<pre>import { newName } from '...'</pre>
Export List + Rename	Import List + Rename
<pre>export { name1, name2 as newName2 }</pre>	<pre>import { name1 as newName1, newName2 } from '...'</pre>
<div> samanthaming</div> <div> samanthaming.com</div> <div> samantha_ming</div>	

Modules

- ▶ La única forma de compartir lo que fuese entre modules es mediante import/export
- ▶ También podemos importar modules de terceros, cómo React

Modules

Named exports	Default exports
<p>Cada archivo puede definir uno o más “named exports”</p> <p>Es posible renombrarlos al momento de importarlos</p>	<p>Solo puede existir un default export por archivo</p> <p>A la hora de importarlos no usamos llaves</p> <p>Al momento de importarlos, podemos nombrarlos</p>

La regla a recordar: cada module puede tener multiples named exports, pero un solo default export

13

DOM

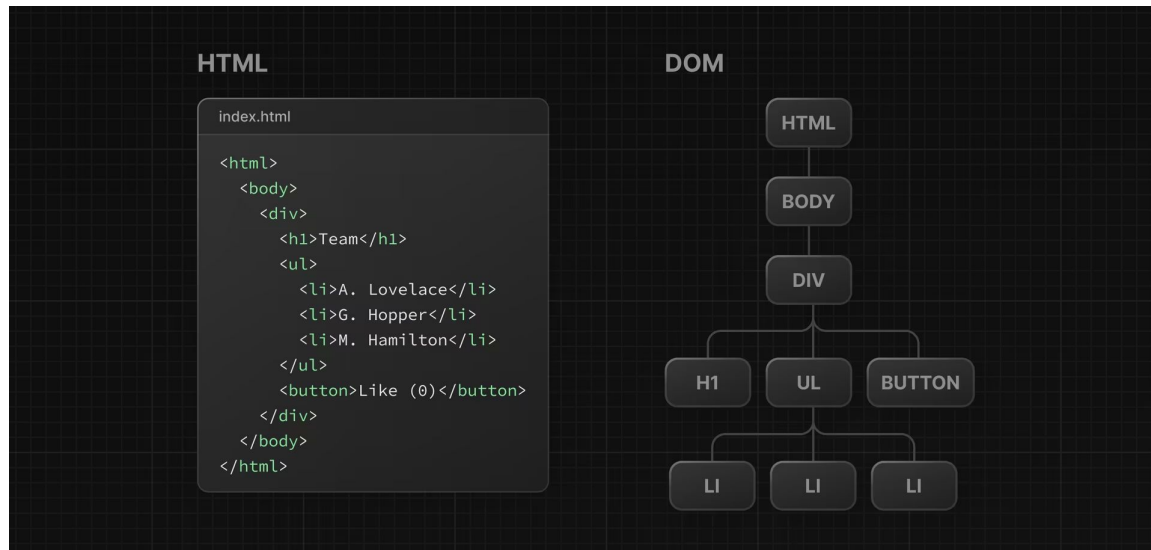
Document Object Model

DOM

Cuando un usuario visita una página web, el servidor retorna un archivo HTML. El navegador lee ese archivo y a través de un proceso (Critical Rendering Path) construye entre otras cosas el Document Object Model (DOM)

El DOM, es una representación de nuestro HTML, con una estructura de árbol.

Actúa como “puente” entre nuestro código y la interfaz del usuario

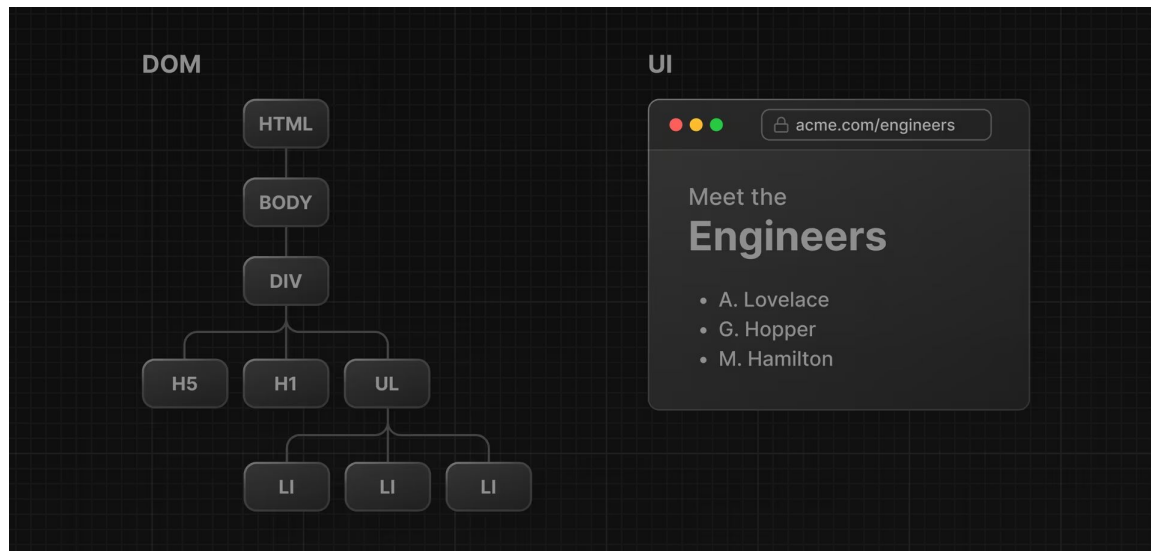


DOM

El DOM es representado con una lógica de árbol. Cada rama del árbol compuesta por nodos, y cada nodo no es más que un objeto.

Los métodos del DOM, nos permite acceder al “árbol” desde nuestro código y de esta forma podemos cambiar la estructura, estilo o contenido.

También podemos agregar eventos para que cuando ocurran realicemos cierta acción.



Acciones que podemos realizar

- ▶ Seleccionar nodos.
- ▶ Editar nodos.
- ▶ Crear y agregar nodos.
- ▶ Destruir nodos.