2.c. Valores Falsy y valores Truthy

Falsy ("Falsillo")

Obervación importate:

Es diferente el valor `Boolean` **true** que valor **primitivo booleano** `true`:

- el valor Boolean **true**: es una propiedad de un objeto `Boolean`, el cual tiene dentro un valor **`[[PrimitiveValue]]: true`**

```
const valorVerdadero = new Boolean(true);
undefined
valorVerdadero
▼Boolean {true} i
  ▼[[Prototype]]: Boolean
    ▶constructor: ƒ Boolean()
    ▶toString: ƒ toString()
    ▼valueOf: ƒ valueOf()
       length: 0
       name: "valueOf"
       arguments: (...)
       caller: (...)
      ▶[[Prototype]]: ƒ ()
      ▶[[Scopes]]: Scopes[0]
    ▶[[Prototype]]: Object
      [[PrimitiveValue]]: false
    [[PrimitiveValue]]: true
```

Es diferente valor Boolean **false** que valor primitivo booleano `false`:

- el valor Boolean **false**: es una propiedad de un objeto `Boolean`

```
const valorFalso = new Boolean(false);
undefined
valorFalso
▼Boolean {false} i
  ▶[[Prototype]]: Boolean
    [[PrimitiveValue]]: false
```

Un **valor falsy** (**valor falsillo**) (a veces escrito **falsey**):

- es un valor que se considera `false` cuando se encuentra en un **contexto booleano** .
- JavaScript utiliza **la conversión de tipos** para **forzar** (convertir) cualquier valor de cualquier tipo a un valor booleano en contextos que lo requieran, como condicionales y bucles.

La siguiente tabla proporciona una lista completa de valores falsy de JavaScript:

Valor	Descripción
`false`	La palabra clave false .
`0`	El Número cero (entonces, también 0.0 , etc., y 0x0).
`-0`	El Número menos cero (entonces, también -0.0 , etc., y -0x0).
`0n`	El BigInt cero (entonces, también 0x0n). Tenga en cuenta que no hay un cero negativo de BigInt : la negación de 0n es 0n .

`""`
`''`
`` `` ``
	Valor de cadena vacío.

`null`	null — la ausencia de referencia a objeto.
`undefined`	**`undefined`** — el valor primitivo.
`NaN`	NaN: no es un número.
`document.all`	Los objetos son falsos si y solo si tienen la ranura interna **`[[IsHTMLDDA]]`**. Dicha ranura solo existe en `document.all` y no se puede configurar mediante JavaScript.

Ejemplos

Ejemplos de **valores *falsy*** en JavaScript (que se **convierten en `false`** en contextos booleanos y, por lo tanto, *eluden* el bloque if):

```
if ( false ) {
   // No alcanzable
}

if ( null ) {
   // No alcanzable
}

if ( undefined ) {
   // No alcanzable
}

if ( 0 ) {
   // No alcanzable
}

if ( -0 ) {
   // No alcanzable
}

if ( 0n ) {
   // No alcanzable
}

if ( NaN ) {
   // No alcanzable
}

if ( "" ) {
   // No alcanzable
}
```

El operador lógico AND, &&

Si el primer objeto es un **falsy**, devuelve ese objeto (y no `false`):

```
consola.log( false && "perro" );
// devuleve:  false

consola.log( 0 && "perro" );
// devuelve: 0
```

Truthy ("Verdaderillo")

En JavaScript, un **valor truthy**

- es un valor que se considera `true` cuando se encuentra en un **contexto booleano** .
- todos los valores son truthy a menos que sean un falsy (o uno u otro): es decir, todos los valores son truthy excepto `false` , 0 , -0 , 0`n` , `""` , `null`, `undefined` y `NaN` .
- JavaScript utiliza **la coerción de tipos** en contextos booleanos.

Ejemplos de valores verdaderos en JavaScript (que serán obligados a ser verdaderos en contextos booleanos y, por lo tanto, ejecutarán el bloque `if`):

```
if ( true )
if ( {} )
if ( [] )
if ( 42 )
if ( "0" )
if ( "false" )
if ( new Date())
if ( -42 )
if ( 12n )
if ( 3.14 )
if ( -3.14 )
if ( Infinity )
if ( -Infinity )
```

El operador lógico AND, &&

Si el primer evaúa a `true`, el operador lógico AND devuelve el segundo operando (y no `true`):

```
true && "perro"
// devuelve "perro"

[] && "perro"
// devuelve "perro"
```

Coerción Boolean

El **objeto** `Boolean` representa un **valor de verdad** : verdadero o falso .

Primitivos booleanos y objetos booleanos

No confundas los **valores booleanos primitivos** `true` y `false` con el valor **true** y el valor **false** del objeto Booleano .

- **cualquier** objeto, incluido un objeto Booleano cuyo valor es **false**, se evalúa como verdadero cuando se pasa a una sentencia condicional.
- por ejemplo, la condición en la siguiente instrucción if se evalúa como verdadera :

```
const x = new Boolean(false);
if (x) {
   // este codigo se ejecuta
}
```

Este comportamiento no se aplica a las **primitivas Boolean** : por ejemplo, la condición en la siguiente instrucción if se evalúa como falsa :

```
const x = false;
if (x) {
  // este codigo no se ejecuta
}
```

No uses el constructor `Boolean()` con `new` para convertir un valor no booleano en un valor booleano, usa más bien:

- la función constructora: `Boolean()`
- una operador doble NO (`!!`)

```
const correcto = Boolean( expresión );          // utilizar este
const correcto2 = !! ( expresión );             // o esto

const incorrecto = new Boolean( expresión );    // ¡No uses esto!
```

Si especificas cualquier objeto, incluido un objeto booleano cuyo valor es `false` , como valor inicial de un objeto Boolean, el nuevo objeto Boolean tiene un valor de **true** .

```
const miFalso = new Boolean(false);        // valor inicial de false
const g = Boolean(miFalso);                // valor inicial true
const miCadena = new String('Hola');       // objeto de cadena
const s = Boolean(miCadena);               // valor inicial de verdadero
```

Advertencia: rara vez deberías usar Boolean como constructor.

Coerción booleana

Muchas **operaciones integradas** que esperan booleanos primero fuerzan sus argumentos que no son booleanos a booleanos y se pueden resumir de la siguiente manera:

- los valores booleanos se devuelven tal cual.
- `undefined` se convierte en `false` .
- `null` se convierte en `false` .
- `0` , `-0` y `NaN` se vuelven `false`; otros números se convierten en `true` .
- `0n` se convierte en `false`; otros `BigInt` se convierten en `true`
- Los `Symbol` se convierten en `true`.
- todos los objetos se vuelven `true`.

Nota: un comportamiento heredado hace que el objeto `document.all` devuelva `false` cuando se usa como valor booleano, a pesar de ser un objeto. Esta propiedad es heredada y no estándar y no debe usarse.

Nota: A diferencia de otras conversiones de tipo, como la **coerción de cadenas** o la coerción de **números**, la coerción booleana no intenta convertir objetos en primitivos: es decir, si es un objeto, pasa a ser `true`.

En otras palabras

- sólo hay un puñado de valores que se convierten en `false`: estos se denominan **valores falsy** .
- todos los demás valores se denominan **valores truthy** .
- la calidad de verdad de un valor es importante cuando se usa con
  - **operadores lógicos**
  - **declaraciones condicionales**
  - cualquier **contexto booleano** .

Hay dos formas de lograr el mismo efecto en JavaScript.

- Doble NO: `!! x` niega x dos veces, lo que convierte `x` en un valor booleano utilizando el mismo algoritmo que el anterior.
- La función constructora `Boolean()`: `Boolean(x)` usa el mismo algoritmo que el anterior para convertir `x` .

Interacción entre falsy, truthy y operador de igualdad no estricta con `false` (`== false`)

Ten en cuenta que la veracidad no es lo mismo que igualdad no estricta (`==`) a `true` o `false` .

```
if ( [] ) {
  console.log("[] es truthy");
}
// [] es truthy

if ( [] == false ) {
  console.log("[] == false");
}
// [] == false
```

- `[]` es un **valor truthy** y también es igual no estrictamente a `false`.
  - es truthy, porque todos los objetos son truthy.
- sin embargo, cuando se compara con `false`, que es un tipo primitivo:
  - 1) `[]` también se **convierte en un primitivo**, que es el string vacío `""` a través de `Array.prototype.toString()`:
  - 2) comparar strings y booleanos da como resultado que ambos se conviertan en números, y ambos se convierten en `0`, por lo que `[] == false` es `true`

En general, ser **falsy** y `== false` difieren en los siguientes casos:

- `NaN`, `undefined` y `null`
  - son **falsy**
  - pero: NO son `== false`
- `"0"` (y otros literales de cadena que no son `""` pero obtener coaccionado a 0):
  - son **truthy**
  - pero: SÍ son `== false`
- los objetos
  - son siempre **truthy**
  - pero: su **representación primitiva** puede ser `== false`.

Interacción entre falsy, truthy y operador de igualdad no estricta con `true` (`== true`)

Es aún más improbable que los **valores truthy** sean iguales no estrictamente a `true` .

- todos los valores son o bien truthy o bien falsy
- pero: la mayoría de los valores son iguales no estrictamente ni verdadero ni falso .

Ejemplos

Crear objetos `Boolean` con un valor inicial `false`

```
const bNoParam     = new Boolean();
const bZero        = new Boolean(0);
const bNull        = new Boolean(null);
const bCadenaVacía = new Boolean('');
const bfalse       = new Boolean(false);
```

Crear objetos `Boolean` con un valor inicial `true`

```
const btrue        = new Boolean(true);
const btrueString  = new Boolean('true');
const bfalseString = new Boolean('false');
const bSuLin       = new Boolean('Su Lin');
const bArrayProto  = new Boolean( [] );
const bObjProto    = new Boolean( {} );
```