

Hecho con  por alumnos de Henry[Intro](#) [Primeros Pasos](#) [Git](#) [Git y GitHub](#) [Conceptos](#) [JS I](#) **[JS II](#)** [JS III](#)[JS IV](#) [JS V](#) [JS VI](#) [HTML](#) [CSS](#) [Calendario](#) [Glosario](#) [Challenge](#)**Contenido de la clase**Tiempo de lectura
20 min**Homework** 

Undefined y null

Veracidad

Operadores de comparación
(continuación)Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos**Bucles for**

El operador ++

Bucles infinitos

ArgumentsLa mejor herramienta del
programador

Recursos adicionales

Homework

JavaScript II

Flujos de control, operadores lógicos, bucles *for*

More from Henry



Undefined y null

Hay un par de objetos Javascript que realmente no encajan en ningún tipo de dato. Esos son los valores **undefined** y **null**. Obtendrás **undefined** cuando busques *algo* que no existe, como una variable que aún no tiene un valor. **undefined** simplemente significa que lo que estás pidiendo no existe.

Dejanos tu feedback! 



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación
(continuación)

Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

La mejor herramienta del
programador

Recursos adicionales

Homework

`null` es un objeto que nosotros, los desarrolladores, establecemos cuando queremos decirles a otros desarrolladores que el elemento que están buscando existe, pero no hay ningún valor asociado con él. Mientras que `undefined` está configurado por Javascript, `null` está configurado por un desarrollador. Si alguna vez recibes `null`, debes saber que otro desarrollador estableció ese valor en `null`

```
let numeroTelefono = '11-1234-5678';  
numeroTelefono = null;
```

```
numeroTelefono; // null
```

Una última cosa a tener en cuenta, ni `undefined` ni `null` son cadenas, están escritas tal como están sin comillas, como un booleano.

Veracidad

En estas lecciones hemos hablado sobre los valores booleanos, `true` y `false`. Cuando se usa una declaración `if` u otra declaración que espera un valor booleano (como `!`, `NOT`), y la expresión dada no es un valor booleano, Javascript hará algo llamado "coerción de tipo" y transformará lo que sea que se le entregue a un valor booleano. Esto se conoce como "truthy" y "falsey". Cada tipo de datos tiene una veracidad. Aquí hay unos ejemplos:

```
// Datos que son forzados a verdaderos/"true"  
true  
1  
, ,
```



Dejanos tu feedback! 👍



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación
(continuación)

Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

La mejor herramienta del
programador

Recursos adicionales

Homework

```
// Datos que son forzados a falsos/"false"
false
0
undefined
null
'' // Una cadena vacía
```

Operadores de comparación (continuación)

En la última lección usamos operadores de comparación, ahora profundizaremos un poco más sobre cómo funcionan y luego presentaremos un pariente cercano de operadores de comparación, los "operadores lógicos".

En la última lección presentamos nuestros operadores de comparación, (> >= < <= === !==). Estos operadores funcionan como lo harían en una clase de matemáticas, mayor que, menor que, etc. Utilizamos estos operadores para evaluar dos expresiones. A medida que la computadora ejecuta el código, el operador devolverá un **verdadero** (si la declaración es verdadera) o un **falso**.

```
1 > 2;    // false
2 < 3;    // true
10 >= 10; // true
100 <= 1; // false
```

El "triple igual" (===) no debe confundirse con un solo signo igual (que indica asignar un valor a una variable). El triple igual comparará todo sobre los dos elementos, incluido el tipo, y devolverá si son exactamente iguales o no:



Dejanos tu feedback! 👍



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación

(continuación)

Flujos de control

(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores

lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

La mejor herramienta del

programador

Recursos adicionales

Homework

Debido a esto , se considera una mala práctica usar el doble igual. Nos gustaría verte siempre usando el triple, y siempre nos verás usándolo.)

```
1 === 1;           // true
1 === '1';         // false
'perro' === 'perro'; // true
'perro' === 'Perro'; // false
```

El último operador de comparación que nos gustaría presentarle tiene dos partes.

Primero es el “NOT” (!). Cuando veas esto significará que estamos preguntando lo contrario de la expresión (volveremos a visitar el operador NOT más adelante en esta lección).

Con eso en mente, podemos introducir el “no es igual” (!==). Esto devolverá verdadero si los artículos NO son iguales entre sí de alguna manera. Esto, como el triple igual, tiene en cuenta el tipo de dato.

```
1 !== 1;           // false
1 !== '1';         // true
'perro' !== 'perro'; // false
'perro' !== 'Perro'; // true
```

Flujos de control (continuación)

En la última lección aprendimos sobre el operador **if** . Podemos usar **if** para verificar y ver si una expresión es **true** , si es así, ejecute algún código, o si no es así, que omita el código y siga ejecutando el programa.



Dejanos tu feedback! 👍



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación
(continuación)

Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

La mejor herramienta del
programador

Recursos adicionales

Homework

```
    console.log('La expresión es verdadera');  
  }
```

Para complementar a `if`, también podemos usar las declaraciones `else if` y `else`. Estas declaraciones deben usarse con `if` y deben venir después de él. Estas declaraciones serán evaluadas si el inicial `if` devuelve `false`. Podemos pensar en el `else if` como otra declaración `if` que se ha encadenado (podemos tener tantas otras declaraciones `if` que queramos). Solo se ejecutará un bloque de código de instrucción `if` o `else if`. Si en algún momento una declaración devuelve `true`, ese código se ejecutará y el resto se omitirá:

```
if (false) {  
    console.log('Este código será omitido');  
} else if (true) {  
    console.log('Este código correrá');  
} else if (true) {  
    console.log('Este código NO correrá');  
}
```

La declaración `else` siempre aparecerá al final de una cadena `if-else` o `if`, y actuará de manera predeterminada. Si ninguna de las expresiones devuelve `true`, el bloque de código `else` se ejecutará sin importar qué. Si alguna de las expresiones anteriores `if` o `else if` son `true`, el bloque de código de instrucción `else` no se ejecutará.

```
if (false) {  
    console.log('Este código será omitido');  
} else if (false) {
```



Dejanos tu feedback! 👍



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación
(continuación)

Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

La mejor herramienta del
programador

Recursos adicionales

Homework

Operadores lógicos

También podemos combinar dos expresiones de igualdad y preguntar si alguna de las dos es verdadera, si ambas son verdaderas o si ninguna de ellas es verdadera. Para hacer esto, utilizaremos operadores lógicos.

&&

El primer operador lógico que veremos es el operador "Y" ("AND"). Está escrito con dos símbolos (`&&`). Esto evaluará ambas expresiones y devolverá verdadero si AMBAS expresiones son `true` . Si uno (o ambos) de ellos es falso, este operador devolverá `false` :

```
if (100 > 10 && 10 === 10) {  
    console.log('Ambas declaraciones son ciertas')  
}  
  
if (10 === 9 && 10 > 9) {  
    console.log('Una de las declaraciones es falsa')  
}
```

||

El siguiente es el operador "Ó" ("OR"). Está escrito con dos barras verticales (`||`). Determinará si una de las expresiones es `true` . Devolverá `true` si una (o ambas) de las expresiones es `true` . Devolverá `false` si AMBAS expresiones son `false` :



Dejanos tu feedback! 👍



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación
(continuación)

Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

La mejor herramienta del
programador

Recursos adicionales

Homework

```
console.log('Ambas declaraciones son ciertas')
}

if (10 === 9 || 10 > 9) {
  console.log('Una de las declaraciones es verdadera')
}

if (10 === 9 || 1 > 9) {
  console.log('Ambas declaraciones son falsas')
}
```

!

El último operador lógico es el operador "NOT" ("NO"). Está escrito como un solo signo de exclamación (!). Vimos este operador antes al determinar la igualdad (!==). Como antes, el operador NOT devolverá el valor booleano opuesto de lo que se le pasa:

```
if (!false) {
  console.log('El ! devolverá true, porque es falso')
}

if (!(1 === 1)) {
  console.log('1 es igual a 1, de modo que la negación es true')
}
```

Notas sobre operadores lógicos

Un par de cosas a tener en cuenta sobre los operadores lógicos.

- Las expresiones se evalúan en orden, y la computadora omitirá cualquier expresión redundante. En una declaración &&, si la primera expresión es false, la segunda expresión no se evaluará porque AMBAS



Dejanos tu feedback! 👍



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación
(continuación)

Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

**La mejor herramienta del
programador**

Recursos adicionales

Homework

porque solo debe haber una declaración **verdadero** para cumplir con los requisitos del operador.

- Usá paréntesis. Como vimos en el segundo ejemplo de operador **!**, usamos paréntesis para evaluar PRIMERO lo que estaba dentro de los paréntesis, luego aplicamos el operador **!**. Podemos ajustar cualquier expresión entre paréntesis y se evaluará antes de evaluar la expresión como un todo.

Bucles for

La mayoría del software se ejecuta en bucles, evaluando expresiones una y otra vez hasta que devuelve lo que estamos buscando o se detiene después de cierto tiempo. Javascript tiene dos expresiones de bucle incorporadas y hoy veremos la primera, el bucle "for".

Los bucles **for** tienen una sintaxis única, similar a la instrucción **if**, pero un poco más compleja. Primero tenemos la palabra clave **for**, seguida de paréntesis y luego abrir y cerrar llaves. Dentro de los paréntesis necesitaremos tres cosas. Primero, debemos declarar una variable, esto es sobre lo que se repetirá el bucle. Entonces tendremos una expresión condicional, el ciclo continuará sucediendo hasta que esta declaración sea **false**. Tercero, incrementaremos nuestra variable. Las tres declaraciones están separadas por un punto y coma.

```
for (let i = 0 ; i < 10  
// | Declaramos una variable | Expresión condici  
    console.log(i);  
)
```



Dejanos tu feedback! 👍



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación

(continuación)

Flujos de control

(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores

lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

La mejor herramienta del

programador

Recursos adicionales

Homework

el contador en uno. El bucle `for` evaluará la expresión condicional. Si es `true`, se ejecutará nuevamente, si es `false` dejará de funcionar.

El operador ++

Vimos en el último ejemplo el operador `++`. Esta es la abreviatura de Javascript para "Establecer el valor de la variable a su valor actual más uno". Hay algunas más de estas expresiones abreviadas de matemática / asignación variable, las visitaremos en las próximas lecciones.

Bucles infinitos

Es posible que un bucle se atasque en lo que llamamos un "bucle infinito". Debes asegurarte de que haya una forma de finalizar el bucle. Ejemplo de un bucle infinito:

```
for (let i = 0; i >= 0; i++) {
  console.log(i);
}
```

Debido a que nuestra expresión condicional SIEMPRE será `true` (`i` nunca será menor que 0), este ciclo se ejecutará esencialmente para siempre. Esto interrumpirá su programa y puede bloquear su navegador web o computadora.

Arguments

Como vimos anteriormente, las funciones son objetos invocables, y podemos hacerlo pasándoles argumentos que varíen el comportamiento de estas.



Dejanos tu feedback! 👍

```
> function log(str) {
```



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación
(continuación)

Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

La mejor herramienta del
programador

Recursos adicionales

Homework

```
> log('hola!')
< 'hola!'
```

Si sabemos las variables a tomar, como en el ejemplo **str**, podemos darle nombre a este parámetro. Sino hay una propiedad **arguments**, propia de todas las funciones, que contiene los parámetros pasados como argumento.

```
> function args() {
    console.log(arguments)
}

> args('hola!', 'otro parametro', 3)
< ["hola!", "otro parametro", 3, callee: 'funct
```

arguments nos da acceso a la **n** cantidad como parámetros, pero tengamos en cuenta que **no es un Arreglo**.

```
> function args() {
    return Array.isArray(arguments)
}

> args(1,2,3)
< false
```

Si queremos saber cuantos parámetros puede recibir una función podemos usar la propiedad **length**.

```
> args.length
< 0 // porque en la función `args` definimos 0
```



Dejanos tu feedback! 👍



Contenido de la clase

Undefined y null

Veracidad

Operadores de comparación
(continuación)

Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos

Bucles for

El operador ++
Bucles infinitos

Arguments

La mejor herramienta del
programador

Recursos adicionales

Homework

La mejor herramienta del programador

Día a día nos encontramos con diversos problemas y, como sabemos, cada problema puede tener distintas soluciones. Una buena forma para adquirir las herramientas que nos permitan resolverlos y aprender su correcto uso es leer documentación oficial o "respaldada". Para ello, nuestro mejor amigo es Google!

En el homework de este módulo nos vamos a encontrar con dos temas que no están explicados en este readme: switch y do while.

Te invitamos a buscar en Google información de estos conceptos para poder desarrollar la homework. A continuación, algunos tips de búsqueda.

- **Recomendado buscar en inglés: ¡Aparecen mejores y mayor cantidad de resultados! Ej: "switch statement javascript".**
- **Recordemos aclarar el lenguaje en el que estamos buscando el tema, como en el ejemplo de arriba donde aclaramos javascript, ya que un mismo tema puede existir en distintos lenguajes de programación y funcionar de manera distinta en cada uno de ellos.**
- **Uno de los primeros resultados que vas a encontrar será MDN (Mozilla Developer Network): es una web muy completa que incluye tanto documentación como ejemplos.**
- **Spoiler, resultado de búsqueda en Google!**
[MDN : Switch](#)
[MDN : Do While](#)

Recursos adicionales



Dejanos tu feedback! 👍



Contenido de la clase

- [MDN: for Loops](#)

Undefined y null

Veracidad

Operadores de comparación
(continuación)

Flujos de control
(continuación)

Operadores lógicos

&&

||

!

Notas sobre operadores
lógicos

Bucles for

El operador ++

Bucles infinitos

Arguments

La mejor herramienta del
programador

Recursos adicionales

Homework

Homework

Abre la carpeta "homework" y completa la tarea descrita en el archivo [README](#)

**Si tienes dudas sobre este tema,
puedes consultarlas en el canal *03_js-ii*
de Slack**



Dejanos tu feedback! 👍