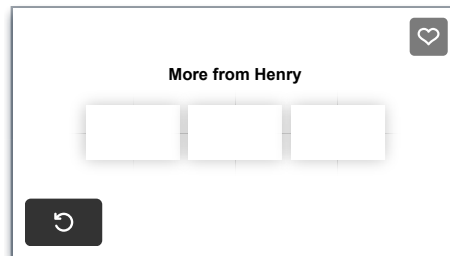


Hecho con  por alumnos de Henry[Intro](#) [Primeros Pasos](#) [Git](#) [Git y GitHub](#) **[Conceptos](#)** [JS I](#) [JS II](#) [JS III](#) [JS IV](#) [JS V](#) [JS VI](#) [HTML](#) [CSS](#) [Calendario](#)[Glosario](#) [Challenge](#)**Contenido de la clase****Expressions**[Expresiones Aritméticas](#)[Expresiones de Strings](#)**Expresiones lógicas**[Expresiones primarias](#)[Expresiones de asignación](#)[Expresiones con efectos](#)[secundarios \(side effects\)](#)**Statements (sentencias)**[Declaration Statements](#)[Conditional Statements](#)[Loops \(bucles\) y Jumps \(saltos\)](#)[Expression Statements](#)Tiempo de lectura
13 min

Conceptos

Expresiones vs Statements



Podemos decir que todo el código que escribimos en JS o “hace algo” o “retorna algo” (o una combinación de los dos). En la terminología de lenguajes de programación esta diferencia está clasificada en la definición de **expressions** (expresiones) y **statements** (sentencias).

Podríamos definir conceptualmente a ambas como:

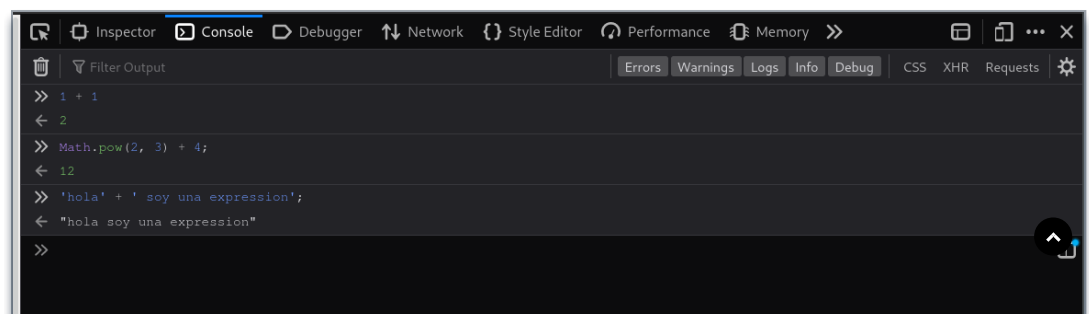

- Una **expression** siempre se convierte (retorna) un valor.
- Un **statement** realiza una acción.

Cuando escribimos código, todo el tiempo mezclamos expresiones y statements para conseguir el resultado final. Por lo tanto, al principio es un poco difícil ver la diferencia entre las dos, pero vamos a intentar ejemplificar lo anterior:

```
// retorna algo
1 + 1
Math.pow(2, 3) + 4;
'hola' + ' soy una expression';
```

1 + 1 intuitivamente se convierte o resuelve a **2** ! eso es una expresión. Es cualquier cosa que escribamos y esperamos que se convierta en otro valor.

Cuando *pegamos* una expression en la consola de Firefox o de Chrome, vamos a poder ver el resultado al que resuelve:

Dejanos tu feedback! 



Contenido de la clase

Expressions

Expresiones Aritméticas

Expresiones de Strings

Expresiones lógicas

Expresiones primarias

Expresiones de asignación

Expresiones con efectos

secundarios (side effects)

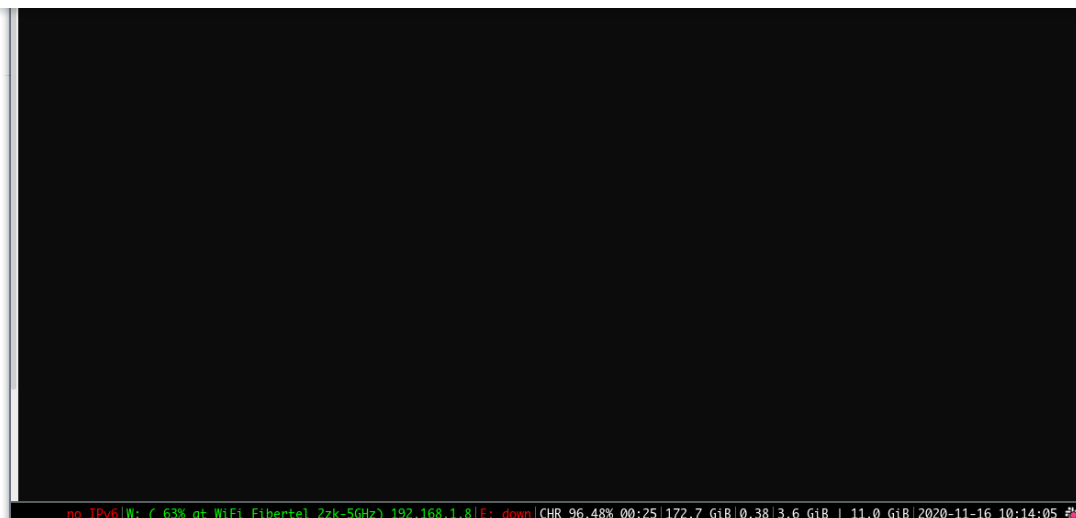
Statements (sentencias)

Declaration Statements

Conditional Statements

Loops (bucles) y Jumps (saltos)

Expression Statements

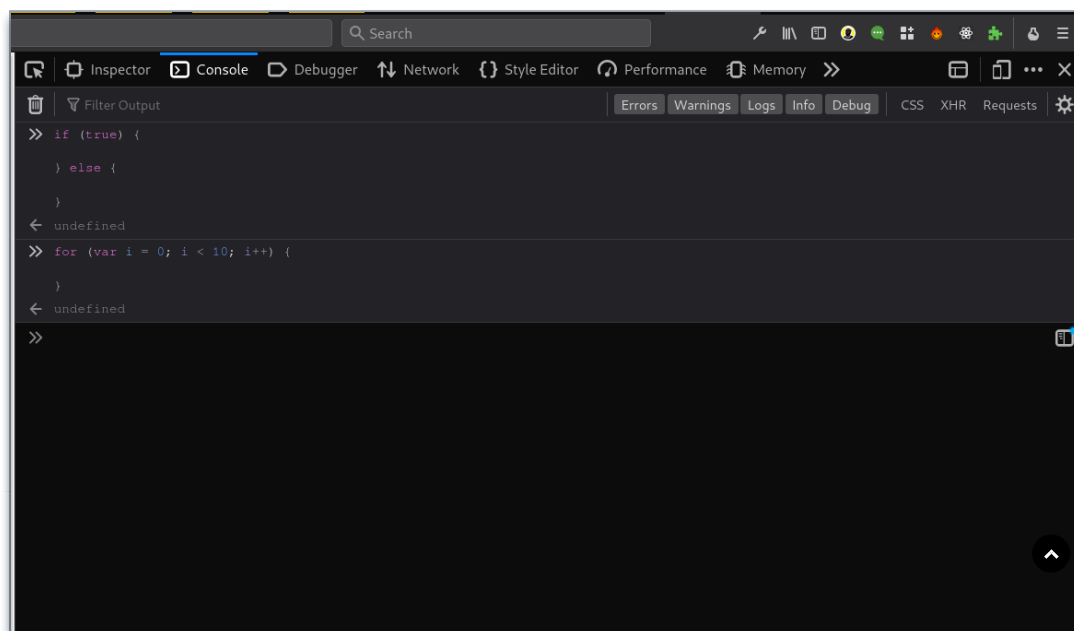


```
// hace algo
if (condicion) {
  // código ejecutado si es true
} else {
  // código ejecutado si es false
}
```

En este ejemplo, vemos que el `if` "hace algo", es decir, escribimos el `if` para que bifurque la ejecución del código según el valor de `condicion`. Como ya sabemos, si `condicion` tiene un valor *verdadero* entonces se ejecutará el bloque de código de arriba, y si no, el de abajo.

Fijense que acá es importante discriminar lo que está *adentro* de los bloques de código, ya que ahí adentro podríamos escribir *expresiones* que sí devuelvan algo.

Nos podemos dar cuenta que algo es un statement, porque si lo *pegamos* en la consola del intérprete -por ejemplo, en la consola del Firefox o Chrome- vamos a ver que no produce ningún resultado:



Dejanos tu feedback! 🍌



Contenido de la clase

Expressions

Expresiones Aritméticas

Expresiones de Strings

Expresiones lógicas

Expresiones primarias

Expresiones de asignación

Expresiones con efectos

secundarios (side effects)

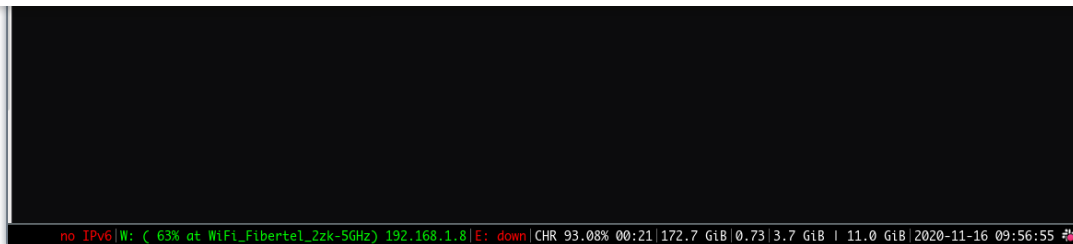
Statements (sentencias)

Declaration Statements

Conditional Statements

Loops (bucles) y Jumps (saltos)

Expression Statements



Una regla fácil para distinguir entre una *expression* y un *statement* en JS es la siguiente:

Si podemos ponerlo dentro de un `console.log`, es una *expression*, si no, es un *statement*. Por ejemplo:

```
// expresiones!  
console.log(1 + 1);  
console.log(Math.pow(2,3) + 22);  
  
// statements  
console.log(if( true) {  
  // código  
});  
// jamás haríamos esto de arriba, no?
```

El **operador ternario**, es una expresión o un statement? ej: (numero > 10 ? 'mayor' : 'menor');

Expressions

Cómo dijimos arriba, una *expression* es cualquier pedazo de código **que pueda ser evaluado a un valor**. Justamente por esto, las vamos a usar en lugares donde JavaScript *espera un valor*. Por ejemplo, cómo cuando pasamos una expresión como argumento de una función.

Según la documentación de MDN, las expresiones se pueden clasificar en las siguientes categorías:

Expresiones Aritméticas

Son las expresiones que resuelven a un valor **númeroico**. Por ejemplo:

```
10;  
1 + 10;  
2 * 16;
```

Expresiones de Strings

Son expresiones que resuelven a una **string**. Por ejemplo:

```
'hola';  
'hola' + ' como va?';
```

Expresiones lógicas

Son expresiones que resuelven a un valor **booleano**. Por ejemplo:

Dejanos tu feedback! 🍌





Contenido de la clase

Expressions

Expresiones Aritméticas

Expresiones de Strings

Expresiones lógicas

Expresiones primarias

Expresiones de asignación

Expresiones con efectos

secundarios (side effects)

Statements (sentencias)

Declaration Statements

Conditional Statements

Loops (bucles) y Jumps (saltos)

Expression Statements

```
11 === 2;  
false;
```

Expresiones primarias

Son expresiones que se escriben por si mismas, y no utilizan ningún operador. Incluyen a valores literales, uso de variables, y algunos keywords de JS. Por ejemplo:

```
'hola';  
23;  
true;  
this; // hace referencia al keyword this  
numero; // hace referencia a la variable numero
```

Expresiones de asignación

Cuando utilizamos el operador `=` hablamos de un *assignment expression*. Está expresión retorna el valor asignado. Por ejemplo:

```
a = 1; // si probamos esto en la consola, vemos  
var c = (a = 2); // vamos a ver que dentro de 1
```

Este es un caso muy particular, nótese que esta expresión retornar un valor, **pero a su vez hace algo!!** Ese algo, es guardar el valor a la derecha del signo `=` en la variable a la izquierda del signo `=`. Otra cosa a notar, es que si usamos el keyword `var` la expresión retorna `undefined`, es decir, no es lo mismo una asignación que una declaración de variables.

Expresiones con efectos secundarios (side effects)

Son expresiones que al ser evaluadas retornan algo, pero a su vez tienen un *efecto secundario* (incrementar un valor, etc...). Por ejemplo:

```
contador++; // retorna el valor de contador e i  
++contador; // incrementa el valor de contador  
  
mult *= 2; // multiplica mult por dos, asigna es
```

Statements (sentencias)

Los *Statements* son instrucciones que le damos al intérprete de JS para que **haga algo**, ese algo puede ser: crear una variable, ejecutar un bloque de código N veces, ejecutar un bloque de código o no según una condición de verdad, declarar una función, etc...

Podemos clasificar a los Statements en las siguientes categorías:

Dejanos tu feedback! 🍌



Contenido de la clase

Expressions

Expresiones Aritméticas

Expresiones de Strings

Expresiones lógicas

Expresiones primarias

Expresiones de asignación

Expresiones con efectos

secundarios (side effects)

Statements (sentencias)

Declaration Statements

Conditional Statements

Loops (bucles) y Jumps (saltos)

Expression Statements

Este tipo de statements indican al intérprete que declare variables o funciones, se utiliza el keyword `function` y `var`. Por ejemplo:

```
var prueba; // declaro la variable prueba
var toni; // declaro la variable toni

function suma(a, b) { // declaro la función su
  // bloque de código
}
```

Habíamos dicho que por regla general lo que podamos pasarle a una función (por ejemplo, `console.log`) por argumento era una expresión... y muchas veces pasamos una declaración de una función por argumento. Esto sucede porque en JS existen también las `function expressions`.

Function expressions vs function declarations

Cuando declaramos una función el intérprete puede *interpretarla* como un statement o cómo una expresión, dependiendo del contexto. Por ejemplo:

```
//function declaration

function resta(a, b) {
  // bloque de código
}

// function expression

var resta = function (a, b) {
  // bloque de código
}

array.map(function() {
  // código;
});
// el argumento de la función espera una expres

// Immediately Invoked Function Expression
(function () {
  console.log('IIFE');
})();
```

Cómo vemos en el ejemplo de arriba, el intérprete *hace algo*: declara la función. Por lo tanto es un statement. En cambio, en el segundo ejemplo, estamos haciendo una asignación, y la asignación espera una *expresión* en la parte de la derecha, así que le estamos pasando un function expression.

Nótese que un function expression puede no tener nombre. Estas son las llamadas `funciones anónimas`.

Conditional Statements

Estos statements sirven para controlar el flujo de ejecución de código según si se cumple o no una condición. Por ejemplo:

```
if (condicion) { // condicion puede ser cualquier
  // ejecuta este bloque si condicion es true
} else if (condicion2) {
  // ejecuta este bloque de código si condicion
```

Dejanos tu feedback! 🍌





Contenido de la clase

Expressions

Expresiones Aritméticas

Expresiones de Strings

Expresiones lógicas

Expresiones primarias

Expresiones de asignación

Expresiones con efectos

secundarios (side effects)

Statements (sentencias)

Declaration Statements

Conditional Statements

Loops (bucles) y Jumps (saltos)

Expression Statements

Loops (bucles) y Jumps (saltos)

Estos statements también controlan el flujo de ejecución del código, pero hacen que un bloque se ejecute N veces (ej: **for**), o que la ejecución salte a otro contexto (ej: **return**). Por ejemplo:

```
// loops
while(condicion) { // condicion es una expresió
  // ejecuta este código mientras condicion sea
}

for (var i = 1; i < 10; i++) {
  // ejecuta este bloque de código 9 veces;
}

// jumps

function () {
  // bloque de código
  return; // cuando llegue acá, sale de la eje
  // bloque de código
}

for (var i = 1; i < 10; i++) {
  // ejecuta este bloque de código N veces;
  continue; // salta a la siguiente iteración d
  // desde acá no se ejecuta;
}

throw new Error('hubo un error, se termina la e
```

Expression Statements

JS tiene la particularidad qué en donde sea que el intérprete espera un *statement*, nosotros podemos pasarle una *expresión*. Esto da lugar a los llamados *expression statements*.

Esto no funciona en sentido inverso, donde se espera una expresión **NO** podemos pasar una *statement*.

