Contenido de la clase

Tiempo de lectura
23 min

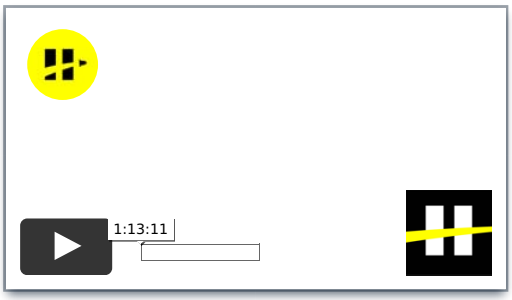
Homework 📝

JavaScript I

Introducción a JavaScript

En esta lección cubriremos:

- **Introducción a Javascript**
- **Variables**
- **Strings, Numbers y Booleanos**
- **Math**
- **Introducción a las Funciones**
- **Control de flujo y operadores de comparación**
- **Introducción a Node y NPM**



JavaScript es un lenguaje de programación que fue creado originalmente para ser usado en el front-end de una página web. La idea original era poder dar dinamismo a las páginas webs, que en un principio eran estáticas. La introducción del "motor V8" de Google ha mejorado la velocidad y el funcionamiento de JS. Haciendo que JS (javascript) sea la lengua franca de la web, llegando inclusive al Back-End a través de NodeJs.

Vamos a aprender los conceptos más básicos de JS:

Variables

Una variable es una forma de almacenar el valor de algo para usar más tarde. (Una nota para aquellos con conocimientos previos de programación: Javascript es un lenguaje de tipado dinámico, una variable se puede configurar (y restablecer) a cualquier tipo, no necesitamos declarar su tipo al iniciar la variable).

Para crear una variable en JavaScript utilizamos la palabra clave `var` , seguida de un espacio y

Dejanos tu feedback!👍

Contenido de la clase

valor usando el signo `=`.

Nota: Las palabras claves o keywords son palabras especiales que utiliza el lenguaje para indicar algo. No podremos usas las palabras claves del lenguaje cómo nombres de variables.

Existen tres formas de declarar una variable:

var

Es la forma declarar una variable en ES5 (ES5 es la versión de JS, hoy en día existe ES6 que es la nueva versión, pero que todavía no es la más usada). Esta es una *palabra clave* genérica para "variable".

Las dos formas siguentes, si bien son válidas, vamos a utilizarlas más adelante en la carrera, cuando tengamos más claros otros conceptos:

let

Es una nueva palabra clave de ES6, esto asignará una variable muy similar a `var`, pero con un comportamiento un poco diferente. Lo más notable es que difiere al crear un "nivel de *scope*" (hablaremos sobre esto más adelante).

const

También es nuevo en ES6. Un `const` es una variable que no se podrá cambiar. Esto es la abreviatura de "constante".

```
var nombre = 'Juan'; // Vamos a usar principaleme
let apellido = 'Perez';
const comidafavorita = 'Pizza';
```

console.log

Otro concepto del que hablaremos de inmediato es

```
console.log();
```

Este método muy simple nos permitirá imprimir en la consola todo lo que pongamos entre paréntesis.

Tipos de Datos

En ciencias de la computación, un tipo de dato informático o simplemente tipo, es un atributo de los datos que indica la clase de datos que se va a manejar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar.

Los tipos de datos aceptados varían de lenguaje en lenguaje.

Los tipos de datos más básicos en Javascript son *Strings*, *Numbers*, and *Booleans*.

Dejanos tu feedback!👍

Contenido de la clase

Las "strings" son bloques de texto, siempre se definirán entre comillas, ya sea simple o doble. Cualquier texto entre comillas es una cadena o string.

```
var nombrePerro = 'firulais';
```

Numbers

Los números son solo eso, números. Los números NO se envuelven en comillas. Pueden ser negativos también. Javascript tiene una limitación en el tamaño de un número (+/- 9007199254740991), pero muy raramente aparecerá esa limitación en nuestro uso diario.

```
var positivo = 27;
var negativo = -40;
```

Boolean

Los booleanos provienen de la **lógica de Boole**. Es un concepto que alimenta el código binario y el núcleo de las computadoras. Es posible que haya visto código binario en el pasado (0001 0110…), esto es lógica booleana. Esencialmente significa que tiene dos opciones, activar o desactivar, 0 o 1, verdadero o falso. En Javascript usamos booleanos para significar verdadero o falso. Esto puede parecer simple al principio, pero puede complicarse más adelante.

```
var meEncantaJavascript = true;
```

Los valores posibles de un dato booleando en JS son: `true` o `false` .

Operadores

Vamos a poder realizar operaciones en JavaScript a través de los *operadores*. Básicamente son símbolos que ya conocemos (`+` , `-` , `/` , `*`) que indican al intérprete de JavaScript las operaciones que debe realizar.

Por ejemplo: Para el intérprete al ver el signo `+` , sabe que tiene que ejecutar la función suma (que tiene internamente definida), y toma como parámetros los términos que estén a la izquierda y la derecha del operador.

```
var a = 2 + 3; // 5
var b = 3 / 3; // 1
```

De hecho, esa forma de escribir tiene un nombre particular, se llama notación notación `infix` o `infija` , en ella se escribe el operador entre los operandos. Pero también existen otro tipos

Dejanos tu feedback!👍

izquierda respectivamente.

Infix Expression	Prefix Expression	Postfix Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

En fin, lo importante a tener en cuenta es que los operadores *son* funciones.

Precedencia de Operadores y Asociatividad

Esto parece aburrido, pero nos va a ayudar a saber cómo piensa el intérprete y bajo qué reglas actúa.

La *precedencia de operadores* es básicamente el orden en que se van a llamar las funciones de los operadores. Estás funciones son llamadas en *orden de precedencia* (las que tienen mayor precedencia se ejecutan primero). O sea que si tenemos más de un operador, el intérprete va a llamar al operador de mayor precendencia primero y después va a seguir con los demás.

La *Asociatividad de operadores* es el orden en el que se ejecutan los operadores cuando tienen la misma precedencia, es decir, de izquierda a derecha o de derecha a izquierda.

Podemos ver la documentación completa sobre Precedencia y Asociatividad de los operadores de JavaScript **acá**

Por ejemplo: `console.log( 3 + 4 * 5)`
Para resolver esa expresión y saber qué resultado nos va a mostrar el intérprete deberíamos conocer en qué orden ejecuta las operaciones. Al ver la tabla del link de arriba, vemos que la multiplicación tiene una precedencia de 15, y la suma de 14. Por lo tanto el intérprete primero va a ejecutar la multiplicación y luego la suma con el resultado de lo anterior -> `console.log( 3 + 20 )` -> `console.log(23)` .

Cuando invocamos una función en Javascript, los argumentos son evaluados primeros (se conoce como non-lazy evaluation), está definido en la **especificación**.

No confundir el orden de ejecución con asociatividad y precedencia, **ver esta pregunta de StackOverflow**.

Ahora si tuvieramos la misma precedencia entraría en juego la asociatividad, veamos un ejemplo:

```
var a = 1, b = 2, c = 3;

a = b = c;
```


Dejanos tu feedback!👍

Contenido de la clase

Qué veriamos en el console.log? Para eso tenemos que revisar la tabla por la asociatividad del operador de asignación `=`. Este tiene una precedencia de 3 y una asociatividad de `right-to-left`, es decir que las operaciones se realizan primero de derecha a izquierda. En este caso, primero se realiza `b = c` y luego `a = b` (en realidad al resultado de `b = c`, que retorna el valor que se está asignando). Por lo tanto al final de todo, todas las variables van a tener el valor `3`. Si la asociatividad hubiese al revés, todos las variables tendrían el valor `1`.

Math

Los operadores matemáticos trabajan en JavaScript tal como lo harían en su calculadora.

+ - * / =

```
1 + 1 = 2
2 * 2 = 4
2 - 2 = 0
2 / 2 = 1
```

%

Algo que quizás no haya visto antes es el Módulo (`%`), este operador matemático dividirá los dos números y devolverá el resto.

```
21 % 5 = 1;
21 % 6 = 3;
21 % 7 = 0;
```

Objetos Globales y métodos

Javascript tiene una serie de objetos integrados para que los usemos. Ya hemos visto, y hemos estado usando, el objeto de consola y su método `log`. Otro de estos objetos es `Math`. Éste tiene varios métodos, al igual que `console` tiene `log`. Para agregar a esto, algunos de nuestros tipos de datos también tienen métodos incorporados.

Math.pow

Podemos usar el método `pow` en `Math` para devolver un número elevado a un exponente. Tomará dos números.

Dejanos tu feedback!👍

```
Math.pow(3,2) = 9;
Math.pow(3,3) = 27;
```

Math.round, Math.floor, Math.ceil

`Math` también tiene métodos que redondearán los números para nosotros. `.round` redondeará un número al número entero más cercano. `.floor` siempre redondeará un número al número entero más cercano hacia abajo. `.ceil` siempre se redondeará al número entero más cercano hacia arriba.

```
Math.round(6.5) = 7;
Math.round(6.45) = 6;
Math.floor(6.999) = 6;
Math.ceil(6.0001) = 7;
```

.length

El tipo de datos "string" tiene un método incorporado llamado `.length`. Cualquier cadena que llamemos a esto devolverá la cantidad de caracteres en esa cadena.

```
var nombreGato = 'felix';
console.log(nombreGato.length); // 5
```

Veremos muchos otros métodos integrados en otros tipos de datos a lo largo de este curso.

Introducción a las Funciones

Las funciones son una parte muy importante de todo lenguaje de programacion y sobre todo en JavaScript. Són tipos particulares de objetos, llamados *callable objects* u objetos invocables, por lo que tienen las mismas propiedades que cualquier objeto.

Ahora que tenemos un conjunto de variables, necesitamos funciones para calcularlas, cambiarlas, hacer algo con ellas. Hay tres formas en que podemos construir una función.

```
function miFuncion() {}
var otraFuncion = function () {};
var yOtra = () => {};
```

Usaremos la primera forma en esta lección y hablaremos sobre las otras formas en próximas lecciones.

Anatomía de una Función

Dejanos tu feedback!👍

Contenido de la clase

Una función comenzará con la palabra clave `function` , esto le dice a lo que sea que esté ejecutando tu programa que lo que sigue es una función y que debe tratarse como tal. Después de eso viene el nombre de la función, nos gusta dar nombres de funciones que describan lo que hacen. Luego viene un paréntesis abierto y uno cercano. Y finalmente, abra y cierre los corchetes. Entre estos corchetes es donde irá todo nuestro código a ejecutar.

```
function logHola() {
    console.log('hola!');
}

logHola();
```

En este ejemplo declaramos una función `logHola` y la configuramos en `console.log('hola')` . Entonces podemos ver que para ejecutar esta función, necesitamos escribir el nombre y los paréntesis. Esta es la sintaxis para ejecutar una función. Una función siempre necesita paréntesis para ejecutarse.

Argumentos

Ahora que podemos ejecutar una función básica, vamos a comenzar a pasarle argumentos.

```
function logHola(nombre) {
    console.log('Hola, ' + nombre);
}

logHola('Martin');
```

Si agregamos una variable a los paréntesis cuando declaramos la función, podemos usar esta variable dentro de nuestra función. Iniciamos el valor de esta variable pasándola a la función cuando la llamamos. Entonces en este caso `nombre = 'Martin'` . También podemos pasar otras variables a esto:

```
function logHola(nombre) {
    console.log( `Hola, ${nombre}`);
}

var miNombre = 'Antonio';
logHola(miNombre);
```

Podemos agregar múltiples argumentos colocando una coma entre ellos:

```
function sumarDosNumeros(a, b) {
  var suma = a + b;
  return suma;
```

Dejanos tu feedback!👍

Contenido de la clase

Declaración "return" y Scope

En el ejemplo anterior presentamos la declaración `return`. No vamos a usar `console.log` con todo lo que salga de una función. Lo más probable es que queramos devolver algo. En este caso es la suma de los dos números. Piense en la declaración de retorno ("return") como la única forma en que los datos escapan de una función. No se puede acceder a nada más que a lo que se devuelve fuera de la función. También tenga en cuenta que cuando una función golpea una declaración de retorno, la función detiene inmediatamente lo que está haciendo y "devuelve" lo especificado.

```
function dividirDosNumeros(a, b) {
  var producto = a / b;
  return producto;
}

dividirDosNumeros(6, 3); // 2
console.log(producto); // undefined
```

Si intentamos `console.log` algo que declaramos dentro de la función, devolverá `undefined` porque no tenemos acceso a él fuera de la función. Esto se llama Scope ("alcance"). La única forma de acceder a algo dentro de la función es devolverlo.

También podemos establecer variables para igualar lo que devuelve una función.

```
function restarDosNumeros(a, b) {
  var diferencia = a - b;
  return diferencia;
}

var diferenciaDeResta = restarDosNumeros(10, 9);
console.log(diferenciaDeResta); // 1
console.log(diferencia); // undefined
```

Podemos ver que la diferencia se establece dentro de la función. La variable dentro de la función solo pertenece dentro de la función.

Control de flujo y operadores de comparación

En este ejemplo, vamos a utilizar operadores de control de flujo y comparación. El flujo de control ("control flow") es una forma de que nuestra función verifique si algo es `true`, y ya sea ejecutando el código suministrado si es así o avanzando si no lo es. Para esto usaremos la palabra clave `if`:

Dejanos tu feedback!👍

Contenido de la clase


```
    if (edad > 18) {
        return true;
    }

    return false;
}

puedeManejar(22); // true
```

Aquí estamos tomando un número (`edad`) y verificando si la declaración es `true` (`22>18`), lo es, por lo que devolveremos `true` , y la función se detendrá. Si no es así, omitirá ese código y la función devolverá `false` .

El símbolo "mayor que" (`>`) que ve en el último ejemplo se llama *Operador de comparación*. Los operadores de comparación evalúan dos elementos y devuelven *verdadero* o *falso*. Estos operadores son: `<` , `<=` , `>` , `>=` , `===` , `!==` . Aprenderemos más sobre estos operadores en la próxima lección.

Recursos adicionales

- Codecademy: Learn Javascript
- Udacity: Intro to Javascript
- MDN: Official Javascript Documentation

Homework

Completa la tarea descrita en el archivo README

Si tienes dudas sobre este tema, puedes consultarlas en el canal *02_js-i* de Slack

Dejanos tu feedback!👍