

Unidad 4: JAVASCRIPT

Contenido

¿Qué es javascript?.....	2
¿Javascript es igual en todos los navegadores web?.....	2
¿Un mismo código javascript puede tener un comportamiento distinto según el navegador?	2
Escribiendo Código	4
Consola del navegador	4
Bloques de código	9
Variables	9
Tipos de datos primitivos	11
Operadores	15
Aritméticos	15
Asignaciones.....	17
Comparaciones	17
Condicionales.....	19
Funciones	26
Métodos típicos para manipular cadenas.....	27
Métodos típicos para manipular arreglos.....	27
Accediendo al DOM.....	28
Buscando elementos por ID.....	28
Buscando elementos por Atributo nombre	28
Buscando elementos por nombre de clases de CSS.....	28
Buscando elementos usando selectores de CSS	28
Agregando eventos al DOM	28
REFERENCIAS.....	29

¿Qué es javascript?

Javascript es un lenguaje de programación interpretado o de scripting. En sus inicios solo era utilizado en el desarrollo de páginas web pero desde el 2009 aproximadamente, con la salida de una herramienta llamado nodejs, también se comenzó a popularizar su uso en muchos otros ámbitos como servidores o dispositivos IoT.

¿Javascript es igual en todos los navegadores web?

Para entender esto primero debemos entender que cada navegador tiene una pieza de código que se llama “motor de javascript”. Cada organización o empresa que decide crear un navegador puede optar por crear su propio motor de javascript o utilizar uno ya creado, ya sea porque se pagó una licencia o porque es de uso gratuito.

Para entender mejor esto veamos un ejemplo:

Alrededor del año 2008 nació un proyecto financiado por Google llamado “The Chromium Projects”. Básicamente la idea de Google fue crear un navegador web de uso libre que sea más rápido y seguro que los existentes en aquellos años. Junto a este proyecto Google también estuvo creando un nuevo motor de javascript que lo denominó V8.

Teniendo lo anterior en mente, ahora debemos saber que existe una gran cantidad de navegadores web actuales cuyo código fuente está basado en el proyecto Chromium, por ejemplo: Chrome, Opera, Edge, Brave, Vivaldi, etc. Esto quiere decir que su motor de javascript es el mismo, el V8. Existen navegadores que no están basados en Chromium, como Firefox cuyo motor es spidermonkey o Safari cuyo motor es JavascriptCore

Para finalizar entonces podemos decir que el código javascript que uno escribe va a ser el mismo sin importar el navegador en el que se corra, pero la pieza de código que finalmente va a ejecutar el código va a depender del navegador en donde se ejecute.

¿Un mismo código javascript puede tener un comportamiento distinto según el navegador?

No, si bien cada navegador tiene su propio motor existe una organización encargada de estandarizar el comportamiento de los lenguajes de scripting. Esta organización se llama

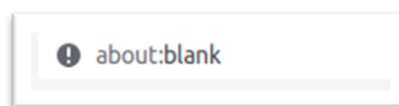
ECMAScript y cada año se reúne un grupo de expertos y analizan qué nuevas características se pueden agregar o mejorar. Una vez que se decide qué características se van a incluir en las versiones nuevas de ECMAScript los fabricantes de los navegadores comienzan a trabajar en ajustar los motores de javascript para soportar las nuevas características.

Escribiendo Código

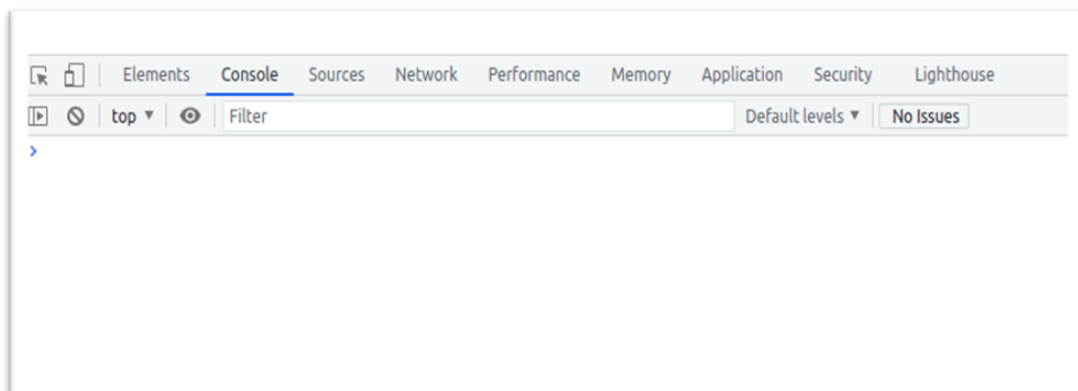
Consola del navegador

El navegador web ya nos provee de una herramienta para realizar pruebas sencillas y ejecutar código javascript. Esta herramienta se denomina consola.

Abrir el navegador y escribir "about:blank", veremos una página totalmente en blanco



Ahora presionamos F12 y veremos que se nos abre una ventana, seleccionamos la pestaña que diga "consola" o "console"

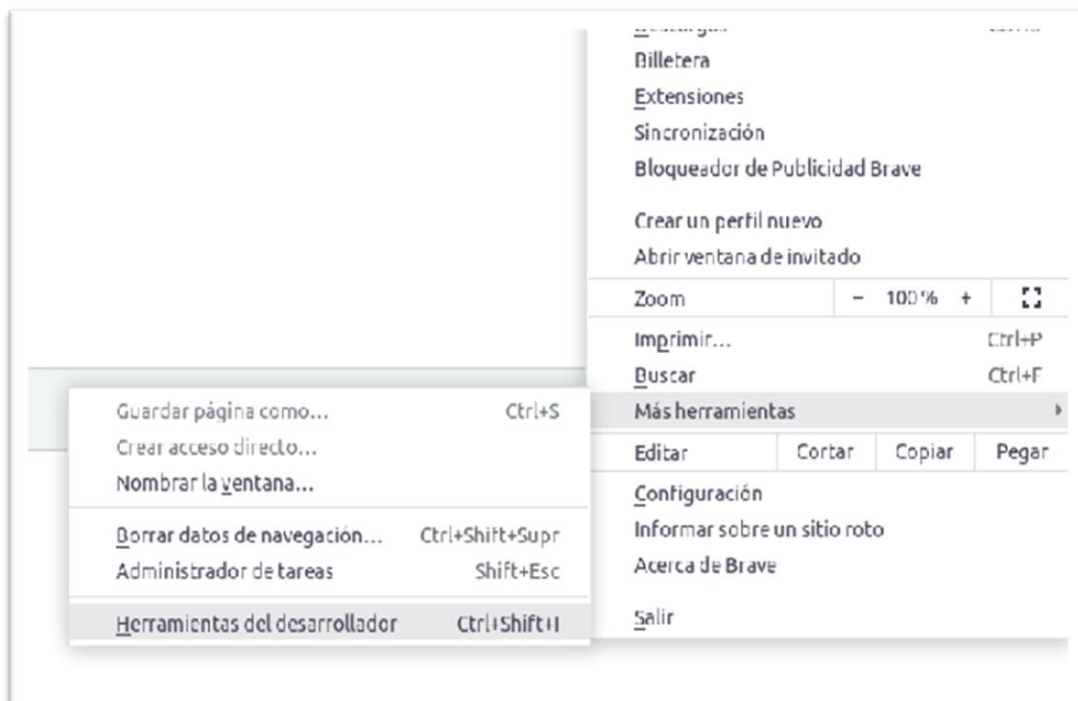


Existen otras maneras de abrir esta ventana como, por ejemplo,

1. Dando click derecho sobre la página en blanco y seleccionando "Inspeccionar elemento" o "Inspeccionar".



- Desde el menú del navegador (suponiendo que usamos Chrome) Buscamos "Herramientas del desarrollador"

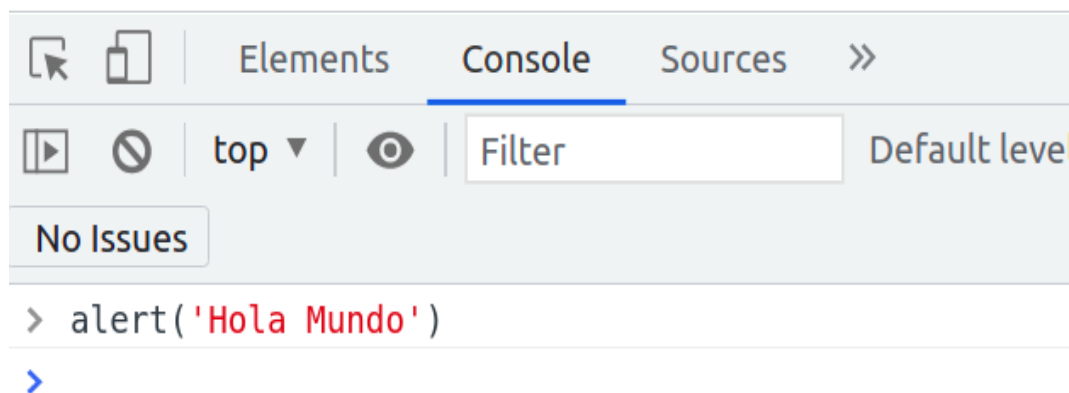
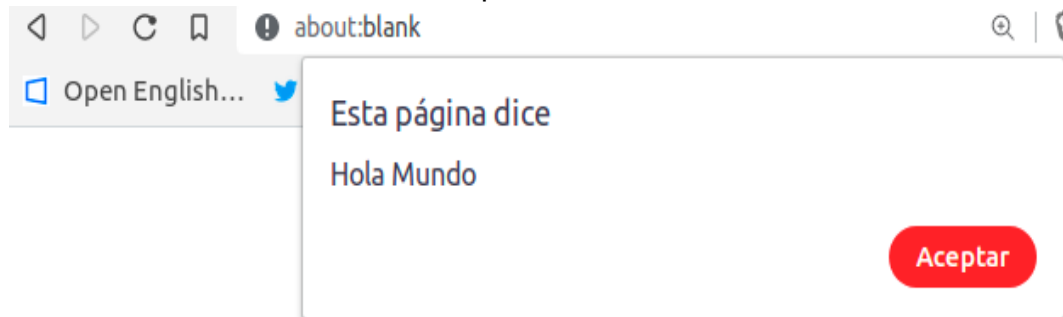


Ahora que ya tenemos la consola abierta comencemos a escribir código.

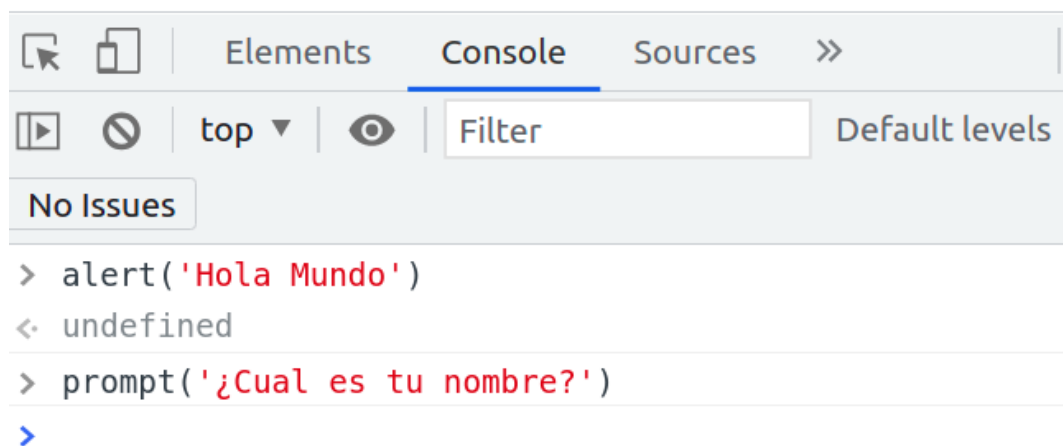
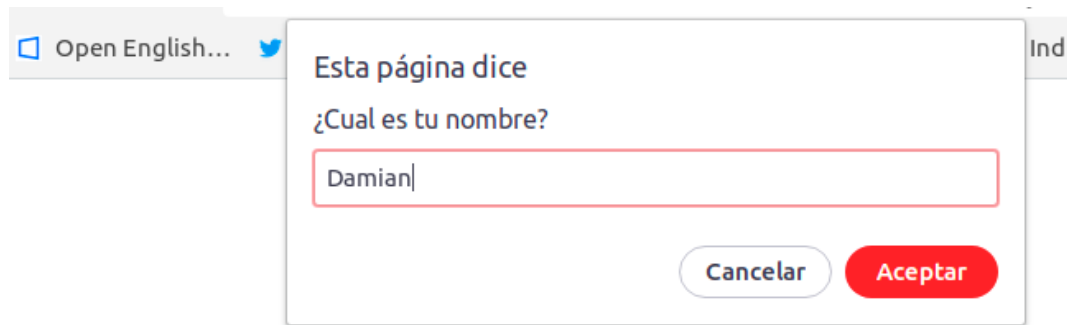
Escribamos en la consola la siguiente línea y luego damos enter "alert('Hola Mundo')"
Veremos que nos aparecerá un popup con el texto que pusimos entre paréntesis.

Javascript por defecto, nos trae algunas funciones predeterminadas, por ejemplo una de ellas es la función alert(). El texto que pusimos entre paréntesis se llama parámetro. Las

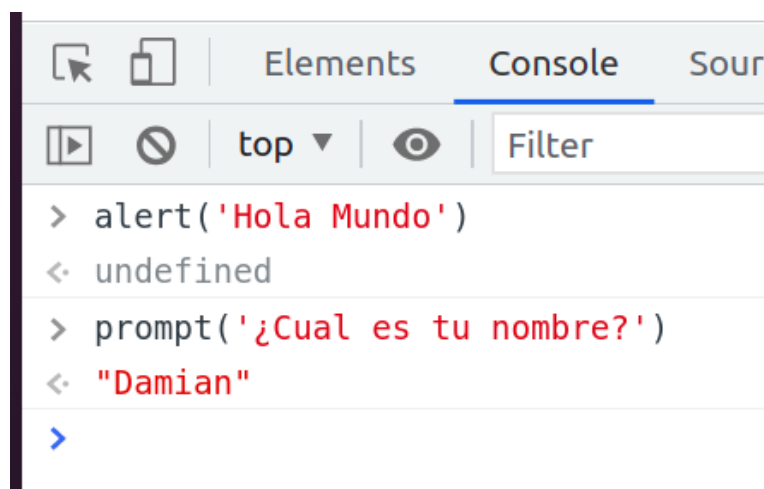
funciones muchas veces suelen recibir parámetros para que podamos personalizar su comportamiento.



Escribamos ahora en la consola la siguiente línea y luego nuevamente damos enter
“prompt(‘¿Cuál es tu nombre?’)”



Luego de dar aceptar veremos lo siguiente



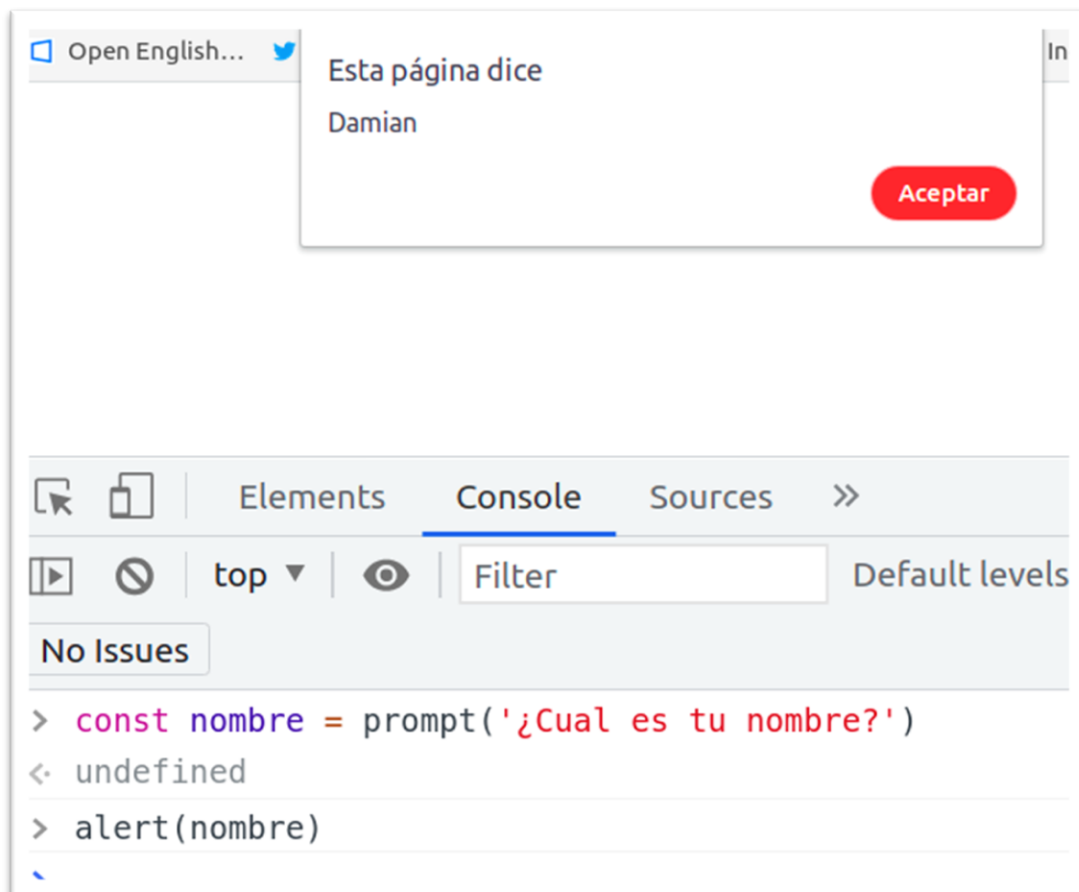
Abajo de la función prompt() vemos el texto "Damian" que fue lo que se ingresó en el ejemplo. Esto es así porque la función nos está avisando que fue lo que se ingresó.

¿Existirá alguna manera de utilizar ese valor que ingresamos para usarlo más tarde?

Si, necesitamos utilizar algo que en programación se llama variable, una porción de memoria que me va a permitir almacenar un valor para usarlo en otro momento.

Veamos el siguiente ejemplo

En este caso estamos ejecutando primero la función `prompt` y guardando su valor en una variable que se llama `nombre`, y luego llamamos a la función `alert` y le pedimos que nos muestre el contenido de la variable `nombre`.



Bloques de código

En javascript, como en muchos otros lenguajes, los bloques de códigos son secciones de código que están entre llaves. Estos bloques de código definen el alcance de las variables (quién las puede leer) y su tiempo de vida

Variables

Existen tres maneras de declarar variable en javascript

- **const** nombre_de_variable
- **let** nombre_de_variable
- **var** nombre_de_variable

const es la mejor opción siempre que la variable no vaya a cambiar de valor. Vive dentro del bloque de código en el que se declaró.

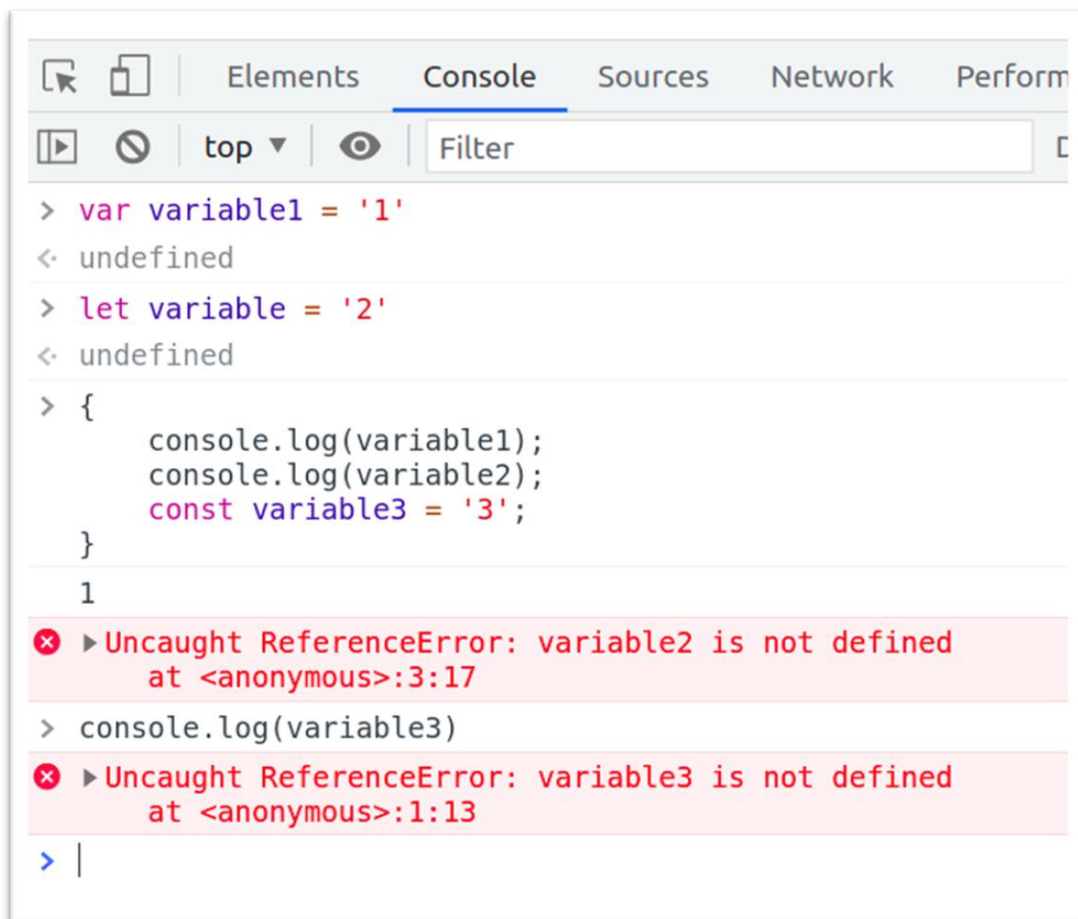
let es la mejor opción si necesitamos una variable que más adelante puede llegar a cambiar de valor. Vive dentro del bloque de código en el que se declaró.

var era la manera antigua de declarar variables (let y const aparecieron con ECMAScript 6) y por retrocompatibilidad se puede seguir usando, pero es una mala práctica. La razón por la que es una mala práctica es que las variables declaradas con var viven por fuera del bloque de código en el que se declaró y además pueden ser re declaradas.

Vemos un ejemplo:

En la siguiente imagen vemos que tenemos dos errores:

- La variable variable1 se imprimió porque fue declarada con var y se tiene acceso a ella desde cualquier parte del código.
- La variable variable2 se declaró con let y afuera del bloque de código que está entre llaves, pero se quiso acceder a ella desde adentro del bloque, como eso no se puede porque es un bloque distinto al de ella se obtuvo un error.
- La variable variable3 se declaró con const y adentro del bloque de código que está entre llaves, pero se la quiso leer afuera del bloque, por esa razón también dió un error.



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following code and its execution results:

```
> var variable1 = '1'
< undefined
> let variable = '2'
< undefined
> {
  console.log(variable1);
  console.log(variable2);
  const variable3 = '3';
}
1
```

Two error messages are shown below the code:

- Uncaught ReferenceError: variable2 is not defined at <anonymous>:3:17
- Uncaught ReferenceError: variable3 is not defined at <anonymous>:1:13

The console also shows the prompt character > and a cursor | at the bottom.

¿Cómo se podrían corregir estos errores?

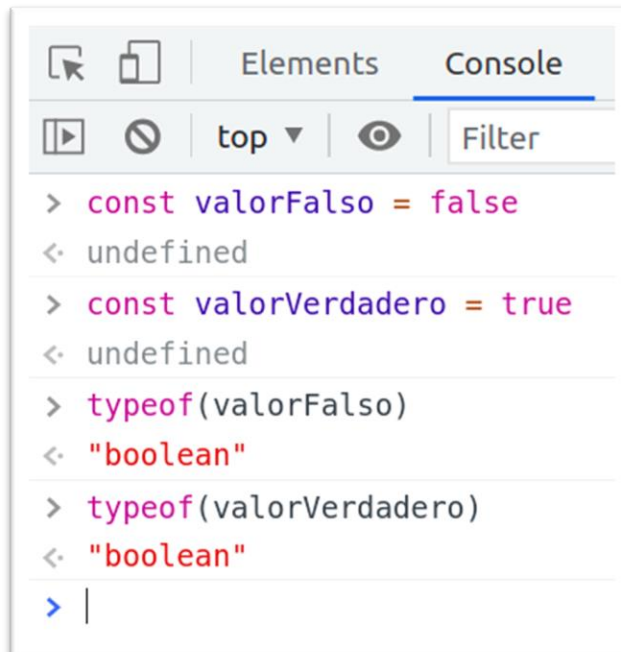
Tipos de datos primitivos

Para poder representar mejor nuestros datos en el código, javascript nos da algunos tipos de datos con los que podemos trabajar.

A continuación, vemos los 7 tipos de datos primitivos.

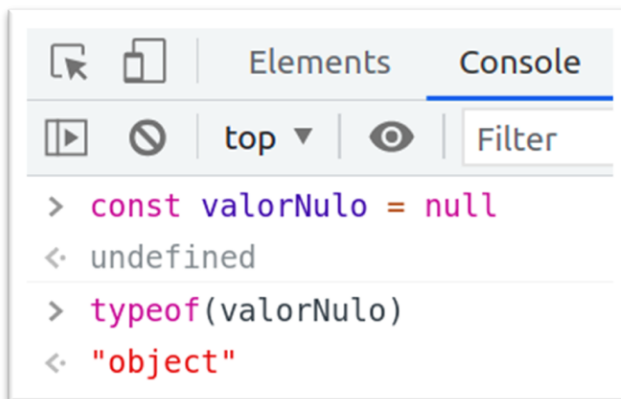
Para conocer el tipo de una variable podemos usar lo que se llama operador `typeof`. Profundizaremos en operadores más adelante.

- Boolean
 - Valores posibles: true o false
 - Ejemplo:



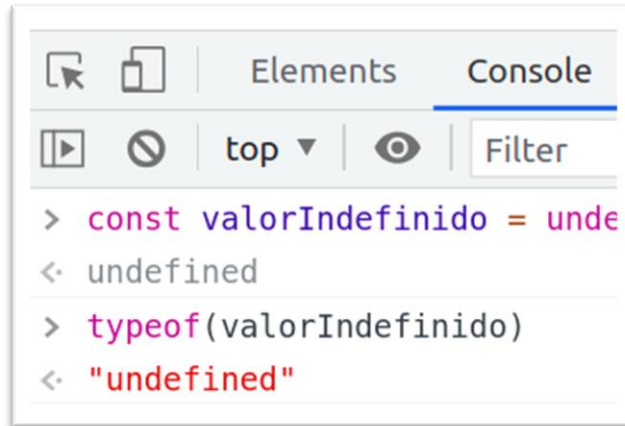
```
> const valorFalso = false
< undefined
> const valorVerdadero = true
< undefined
> typeof(valorFalso)
< "boolean"
> typeof(valorVerdadero)
< "boolean"
> |
```

- Null
 - Valores posibles: null
 - Ejemplo:



```
> const valorNulo = null
< undefined
> typeof(valorNulo)
< "object"
```

- Undefined
 - Valores posibles: undefined
 - Ejemplo



```
> const valorIndefinido = unde  
< undefined  
  
> typeof(valorIndefinido)  
< "undefined"
```

- Number
 - Valores posibles: entre $(2^{53} - 1)$ y $2^{53} - 1$
 - Valores especiales: +Infinity -Infinity NaN (Not A Number)
 - Ejemplo

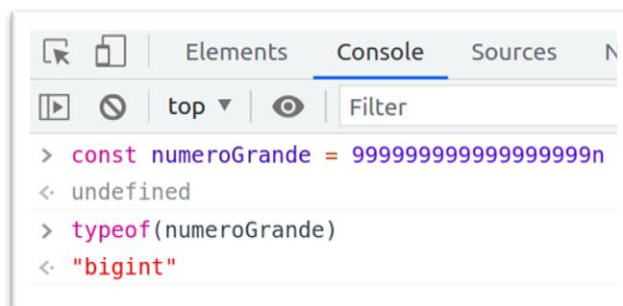


```

> const valorNumerico = 2
< undefined
> typeof(valorNumerico)
< "number"
> const ceroPositivo = +0
< undefined
> const ceroNegativo = -0
< undefined
> typeof(ceroNegativo)
< "number"
> typeof(ceroPositivo)
< "number"
> 2 / ceroNegativo
< -Infinity
> 2 / ceroPositivo
< Infinity
  
```

- BigInt

- Se utiliza para representar números más grandes que los que soporta Number
- Ejemplo (observemos que al final del número debemos agregar una 'n')

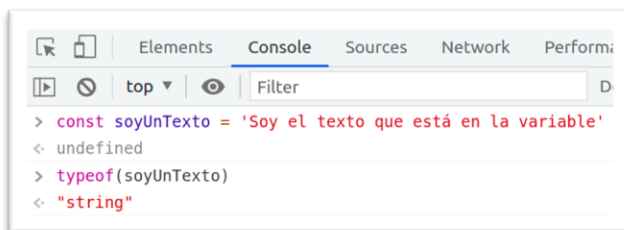


```

> const numeroGrande = 999999999999999999n
< undefined
> typeof(numeroGrande)
< "bigint"
  
```

- String

- Se utiliza para representar texto.
- Ejemplo



```

> const soyUnTexto = 'Soy el texto que está en la variable'
< undefined
> typeof(soyUnTexto)
< "string"

```

- Symbol

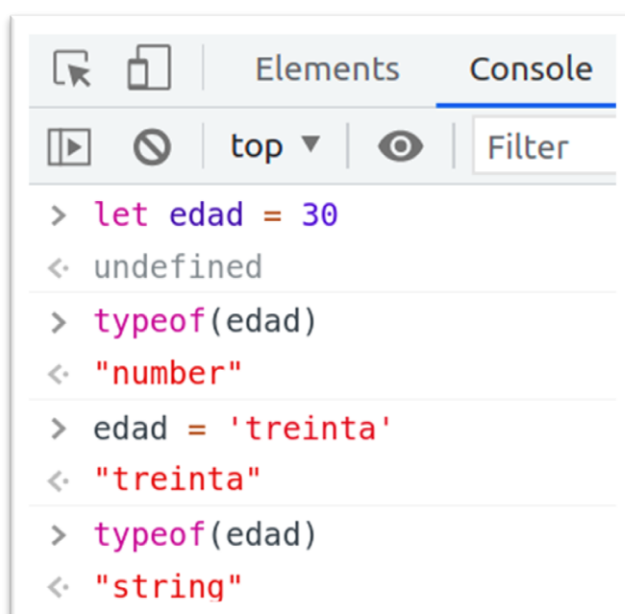
○

Aunque acabamos de ver que existen siete tipos de datos primitivos para una variable en javascript, dicho lenguaje es débilmente tipado, lo que significa que la variable puede ir cambiando su tipo a lo largo del código. Esta es una gran diferencia con respecto a otros lenguajes que son fuertemente tipados como Java, C, C++, C#, GO, etc.

Veamos un ejemplo:

En la siguiente imagen se puede observar como la variable edad primero fue de tipo number, porque guardaba un número, pero luego pasó a ser tipo string.

NOTA: A diferencia de los demás ejemplos, aquí necesitamos usar let porque nuestra variable va a cambiar de valor. No podíamos usar const.



```

> let edad = 30
< undefined
> typeof(edad)
< "number"
> edad = 'treinta'
< "treinta"
> typeof(edad)
< "string"

```

Operadores

Muchas veces nos vamos a encontrar con alguno de los siguientes problemas:

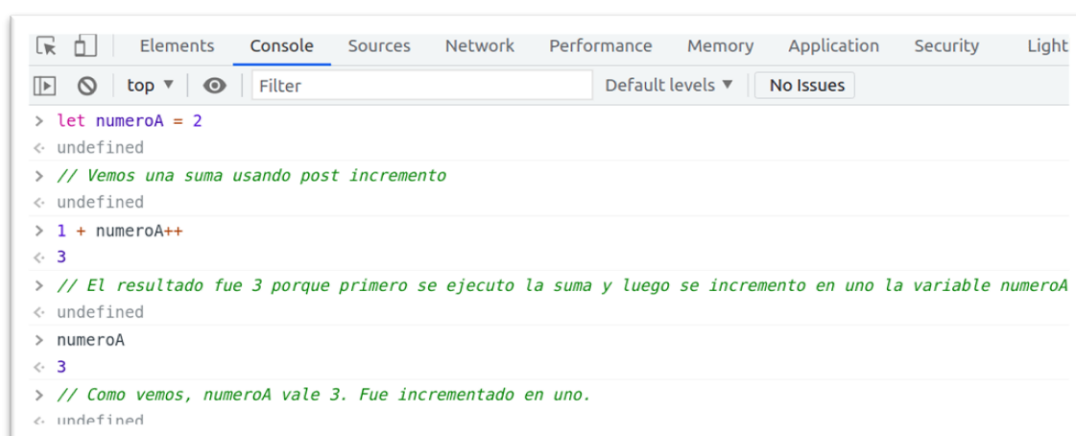
- Querer almacenar nuestros datos en variables
- Querer comparar dos valores
- Querer realizar operaciones con dos o más valores

Para solucionar estos problemas javascript nos provee una serie de operadores que vemos a continuación.

Aritméticos

Nombre	Sintaxis
Suma	$a + b$
Resta	$a - b$
División	a / b
Multiplicación	$a * b$
Exponente	$a ** b$
Resto de división o residuo	$a \% b$
Incremento en 1	$a++$ ó $++a$
Decremento en 1	$a--$ ó $--a$

Ejemplo post incremento

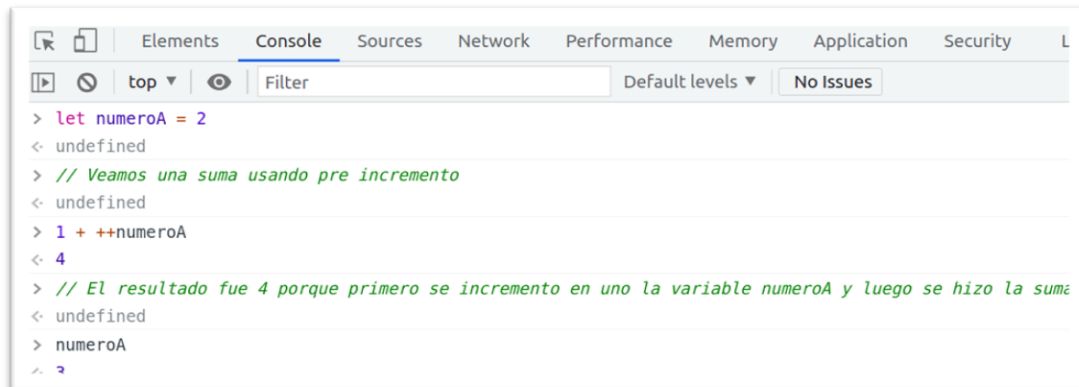


```

> let numeroA = 2
< undefined
> // Vemos una suma usando post incremento
< undefined
> 1 + numeroA++
< 3
> // El resultado fue 3 porque primero se ejecuto la suma y luego se incremento en uno la variable numeroA
< undefined
> numeroA
< 3
> // Como vemos, numeroA vale 3. Fue incrementado en uno.
< undefined

```

Ejemplo pre incremento



```
> let numeroA = 2
< undefined
> // Veamos una suma usando pre incremento
< undefined
> 1 + ++numeroA
< 4
> // El resultado fue 4 porque primero se incremento en uno la variable numeroA y luego se hizo la suma
< undefined
> numeroA
< 3
```

El mismo comportamiento observaremos si realizamos prueba con el operador de decremento en uno.

Asignaciones

Nombre	Sintaxis	Abreviación de...
Asignación simple	<code>a = b</code>	
Asignación de adición	<code>a += b</code>	<code>a = a + b</code>
Asignación de resta	<code>a -= b</code>	<code>a = a - b</code>
Asignación de multiplicación	<code>a *= b</code>	<code>a = a * b</code>
Asignación de división	<code>a /= b</code>	<code>a = a / b</code>
Asignación de residuo	<code>a %= b</code>	<code>a = a % b</code>
Asignación de exponente	<code>a **= b</code>	<code>a = a ** b</code>

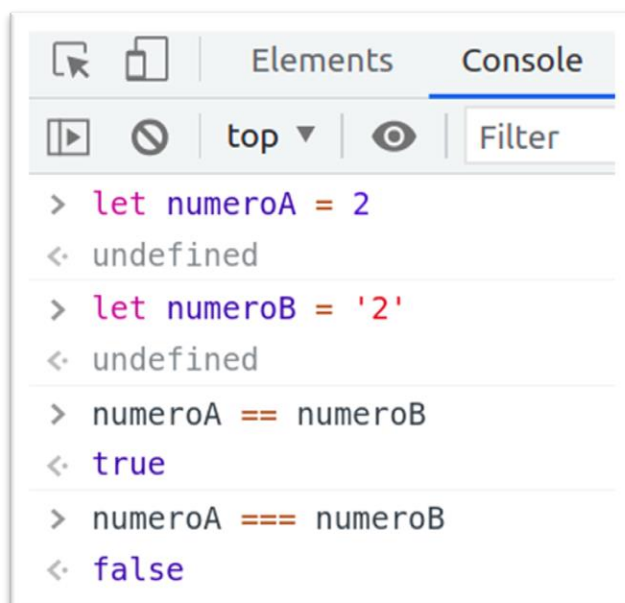
Comparaciones

Cada operador de comparación devolverá true o false dependiendo del resultado de la comparación.

Nombre	sintaxis
Igualdad	<code>a == b</code>
Igualdad estricta	<code>a === b</code>
Distinto	<code>a != b</code>
Distinto estricto	<code>a !== b</code>
Mayor que	<code>a > b</code>
Menor que	<code>a < b</code>
Mayor o igual que	<code>a >= b</code>
Menor o igual que	<code>a <= b</code>

Ejemplo usando `==` y `===`

En el siguiente ejemplo vemos la diferencia entre `==` y `===`. El primero solo valida el valor de una variable sin importar su tipo. En el segundo caso, además de validar que ambos valores sean iguales se validará que sus tipos sean iguales. El mismo comportamiento ocurre con `!=` y `!==`.



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following sequence of commands and results:

```
> let numeroA = 2
< undefined

> let numeroB = '2'
< undefined

> numeroA == numeroB
< true

> numeroA === numeroB
< false
```

Condicionales

Los condicionales son estructuras de código que nos permiten controlar qué líneas de código se deben ejecutar dada una condición en particular.

Sintaxis IF

```
if(condicionAEvaluar){  
    /* Código que se debe ejecutar  
    si la condición es verdadera */  
}
```

Ejemplo

```
const nombre = "pedro"  
if(nombre == "pedro"){  
    console.log("Buenos días ruben")  
}  
// En este caso si ejecutamos el código el mensaje que saldrá será  
"Buenos días ruben"
```

Sintaxis IF ELSE

```
if(condicionAEvaluar){  
    /* Código que se debe ejecutar  
    si la condición es verdadera */  
}else{  
    /* Código que se debe ejecutar  
    si la condición es falsa */  
}
```

Ejemplo

```
const nombre = "pedro"  
if(nombre == "ruben"){  
    console.log("Buenos días ruben")  
}else{  
    console.log("Usted no es ruben")  
}  
  
// En este caso si ejecutamos el código el mensaje que saldrá será  
"Usted no es ruben"
```

Sintaxis FOR

```
// Paso 1: El índice comienza con un valor inicial
// Paso 2: Se evalúa la condición, si da verdadero se ejecuta
// el código dentro del for
// Paso 3: Se cambia el valor del índice

for(let índice=valorInicial; condición; cambiaIndice){
    /* Código a ejecutar mientras la condición
       del for sea verdadera */
}
```

Ejemplo

```
for(let i=0; i<4; i++){
    console.log(`Vuelta numero: ${i}`)
}

// En este caso si ejecutamos el código veremos
Vuelta numero: 0
Vuelta numero: 1
Vuelta numero: 2
Vuelta numero: 3
```

Sintaxis WHILE

```
while(condicion){  
    /*Codigo que se ejecuta mientras la condición sea verdadera */  
}
```

Ejemplo

```
while(vueltas > 0){  
    console.log(`Cuenta regresiva ${vueltas}`);  
    vueltas--;  
}  
  
// En este caso si ejecutamos el código veremos  
Cuenta regresiva 3  
Cuenta regresiva 2  
Cuenta regresiva 1
```

Sintaxis DO...WHILE

```
do{  
    /* El codigo se ejecuta al menos una vez,  
    luego se evalua la condicion del while para  
    decidir si se sigue vuelve a ejecutar */  
}while(condicion)
```

Ejemplo 1

```
const vueltas = 3;  
do{  
    console.log(`Vuelta ${vueltas}`);  
}while(vueltas > 4)  
// En este caso si ejecutamos el código veremos  
Vuelta 3
```

Ejemplo 2

```
const vueltas = 3
do{
    console.log(`Vuelta ${vueltas}`); vueltas--;
}while(vueltas > 0)
// En este caso si ejecutamos el código veremos
Vuelta 3
Vuelta 2
Vuelta 1
```


Sintaxis SWITCH

```
/* El switch evalua el valor de una variable particular y dependiendo
de su valor ejecuta el "case" correspondiente.
Si el valor no se corresponde con los case definidos se ejecuta el "default"
*/

switch(variableAEvaluar){
case valorA: { /* codigo a ejecutar */ break; }
case valorB: { /* codigo a ejecutar */ break; }
case valorC: { /* codigo a ejecutar */ break; }
default { /* codigo a ejecutar */ break; }
}
```

Ejemplo

```
const opcion = 'a';
switch(opcion){
  case 'b': { console.log("Chocolatada"); break; }
  case 'a': { console.log("Matecocido"); break; }
  case 'c': { console.log("Té"); break; }
}
// En este caso si ejecutamos el código veremos
Chocolatada
```

Funciones

Podemos definir una función como un conjunto de instrucciones que realizan una tarea puntual. Además, este conjunto de instrucciones están englobadas bajo un nombre y puede o no tener un parámetro de entrada y/o una salida.

Existen dos maneras de declarar funciones en javascript

- Usando la palabra reservada function
- Usando => (función flecha o lambda) // No lo veremos en el curso

¿Por qué deberíamos crear funciones?

Por dos razones principales:

- Nos ayudan a no repetir el mismo código en varios lugares al mismo tiempo
- Nos ayudan a que nuestro código sea más legible si le ponemos un nombre descriptivo a las funciones.

Ejemplo:

```
/* Supongamos un programa que le hace las
mismas preguntas a 2 diferentes usuarios
*/
const nombre1 = prompt('¿Cual es tu nombre?')
const edad1 = prompt('¿Cuántos años tienes?')
const localidad1 = prompt('¿Donde vives?')

alert('Hola ', nombre1);

const nombre2 = prompt('¿Cual es tu nombre?')
const edad2 = prompt('¿Cuántos años tienes?')
const localidad2 = prompt('¿Donde vives?')

alert('Hola ', nombre2);
```

Lo que vemos acá es que estamos escribiendo muchas veces la función prompt, con ayuda de las funciones podemos crear un “atajo” para evitar tener que escribir tantas veces a esa función.

```
/*
  Hagamos una primera modificacion para no repetir tanto codigo.
*/

// Llamada a la funcion que solicita los datos al usuario
// Nuestro codigo es mas legible ahora
const usuario1 = solicitarDatosAUsuario()
alert('Hola ', usuario1.nombre);

const usuario2 = solicitarDatosAUsuario()
alert('Hola ', usuario2.nombre);

// Funcion que se encarga de solicitarle los datos al usuario.
// Devuelve un objeto con los datos del usuario
function solicitarDatosAUsuario(){
  const nombre = prompt('¿Cual es tu nombre?')
  const edad = prompt('¿Cuántos años tienes?')
  const localidad = prompt('¿Donde vives?')

  return { nombre, edad, localidad }
}
```

Las funciones que son declaradas con la palabra “function” pueden ser escritas en cualquier parte del código. Pero una buena práctica es tenerlas todas arriba, o todas abajo o en un archivo separado directamente, todo depende de qué tan grande es nuestro proyecto. No es lo mismo tener un proyecto con 2 funciones a uno con 150.

¿Cómo podríamos modificar el código si la cantidad de usuarios fuera aleatoria?

Métodos típicos para manipular cadenas

- length
- indexOf()
- slice()
- toLowerCase() / toUpperCase()
- replace()

Métodos típicos para manipular arreglos

- push()

- pop()
- unshift()

Accediendo al DOM

Buscando elementos por ID

- document.getElementById('<idAbuscar>')

Buscando elementos por Atributo nombre

- document.getElementsByName('<nombreAbuscar>')

Buscando elementos por nombre de clases de CSS

- document.getElementsByClassName('<clasesCSSAbuscar>')

Buscando elementos usando selectores de CSS

- document.querySelector('<selectorCSS>')
- document.querySelectorAll('<selectorCSS>')

Ejemplo

Agregando eventos al DOM

- elementoHTML.addEventListener('<tipoDeEvento>', functionAEjecutar)

Tipos de evento más usados

- click
- mouseup
- mousedown
- keyup
- keydown

REFERENCIAS

- <https://www.chromium.org/Home>
- <https://tc39.es/ecma262/>
- <https://developer.mozilla.org/es/docs/Web/JavaScript>
- <https://v8.dev/>
- <https://spidermonkey.dev/>
- <https://trac.webkit.org/wiki/JavaScriptCore>
- <https://www.w3.org/standards/webdesign/script.html>
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions and Operators](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators)
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Data structures](https://developer.mozilla.org/es/docs/Web/JavaScript/Data_structures)
- <https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions>
- [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First steps/Useful string methods](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Useful_string_methods)
- <https://developer.mozilla.org/es/docs/Web/API/Document>
- <https://developer.mozilla.org/es/docs/Web/API/EventTarget/addEventListener>
- <https://developer.mozilla.org/en-US/docs/Web/API/Event/type>