**¿Qué es JavaScript?**

JavaScript es un lenguaje de programación dinámico que tiene la capacidad de ser utilizado en muchos dispositivos diferentes. Puede usarse en ordenadores personales, servidores web y teléfonos inteligentes. Es un lenguaje interpretado, orientado a objetos, débilmente tipado y dinámico.

JavaScript se emplea comúnmente para el desarrollo web front-end y más recientemente para algunos desarrollos back-end a través de frameworks como Node.Js. o Next.Js. Tiene características como la programación orientada a objetos, funciones y herencia basada en prototipos.

**¿Cómo nace Javascript?**

Nace con la necesidad de generar dinamismo en las páginas web y que a su vez los usuarios y las empresas pudieran interactuar unos con otros. Fue creado por Brendan Eich en 1995 y se convirtió en un estándar oficial del World Wide Web Consortium (W3C) en 1997.

**¿Por qué decimos que Javascript es un lenguaje dinámico?**

Corre directamente en la etapa de Runtime, sin una etapa de compilación previa. Esto permite probar nuestro código inmediatamente; pero también es lo que hace que los errores no se muestren sino hasta que se ejecuta el programa. Lo que se ve a primera vista, cuando se analiza el código, es muy probable que no sea lo que se va a obtener cuando el programa se ejecute.

JavaScript permite declarar (por ejemplo) variables cuyo valor (y tipo) solo se conocerá al momento de su ejecución en función de las condiciones dadas al momento de correrlo en un entorno real. En cambio, los lenguajes estáticos no compilarán en código ejecutable a menos que todos los valores (o tipos de valores) se conozcan por adelantado.

**¿Por qué es débilmente tipado?**

Porque los tipos de datos no están bien definidos en el lenguaje y permite, por ejemplo, operaciones entre numerosos y letras. Esto sucede porque el lenguaje asume tipos de datos que no necesariamente fueron los que se querían representar. Se pueden hacer operaciones entre tipos distintos de datos (enteros con strings, booleanos con enteros, etc.). Ejemplo:

4 + "7"; // 47

4 \* "7"; // 28

2 + **true**; // 3

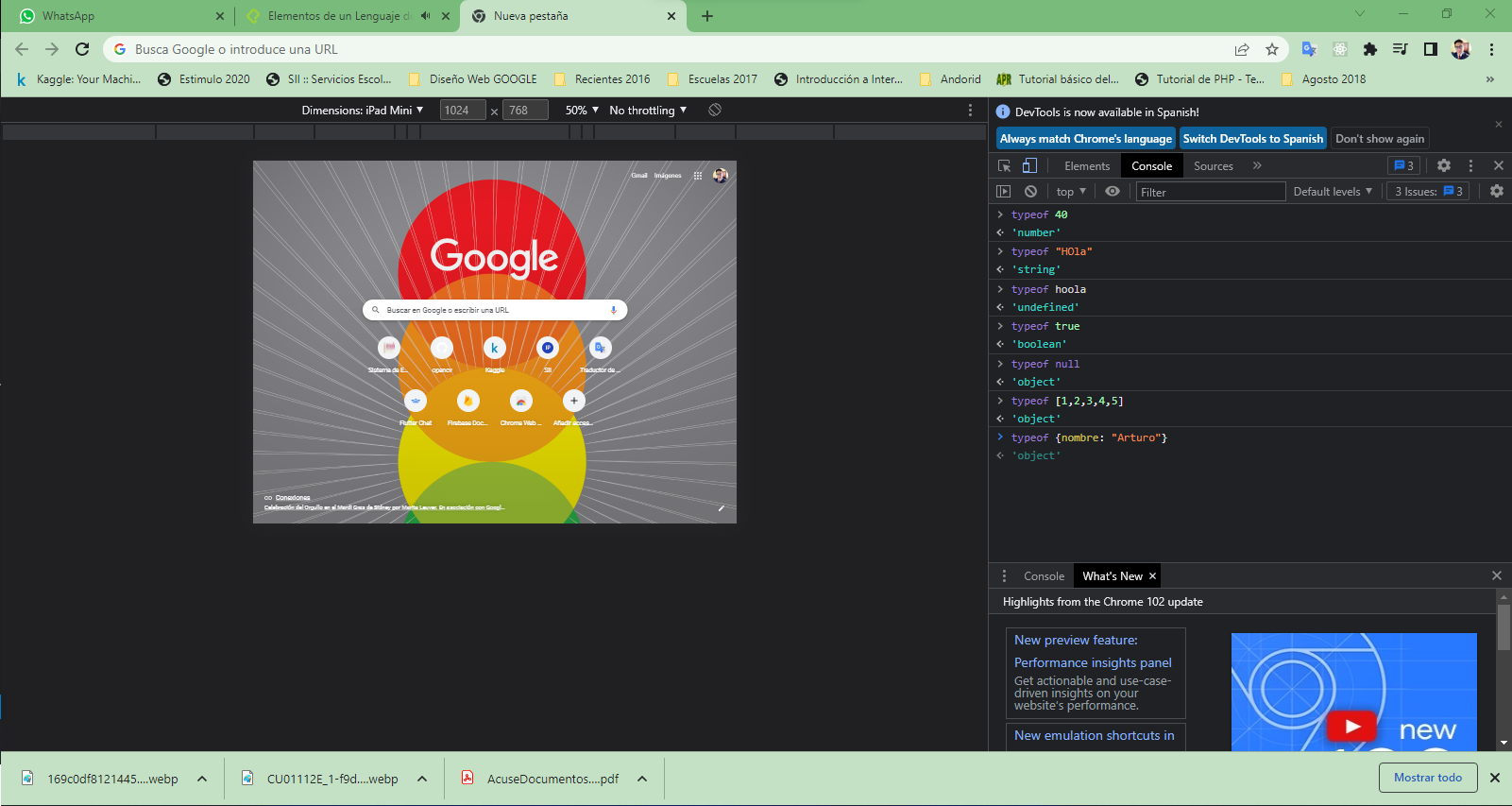
**false** - 3; // -3

**¿Realmente es Javascript un lenguaje interpretado?**

Sí, y la razón es que el navegador lee línea por línea nuestro código, el cual le indica lo que tiene que ir haciendo, sin la necesidad de compilar. Todo esto es controlado por el motor de Javascript V8 del navegador

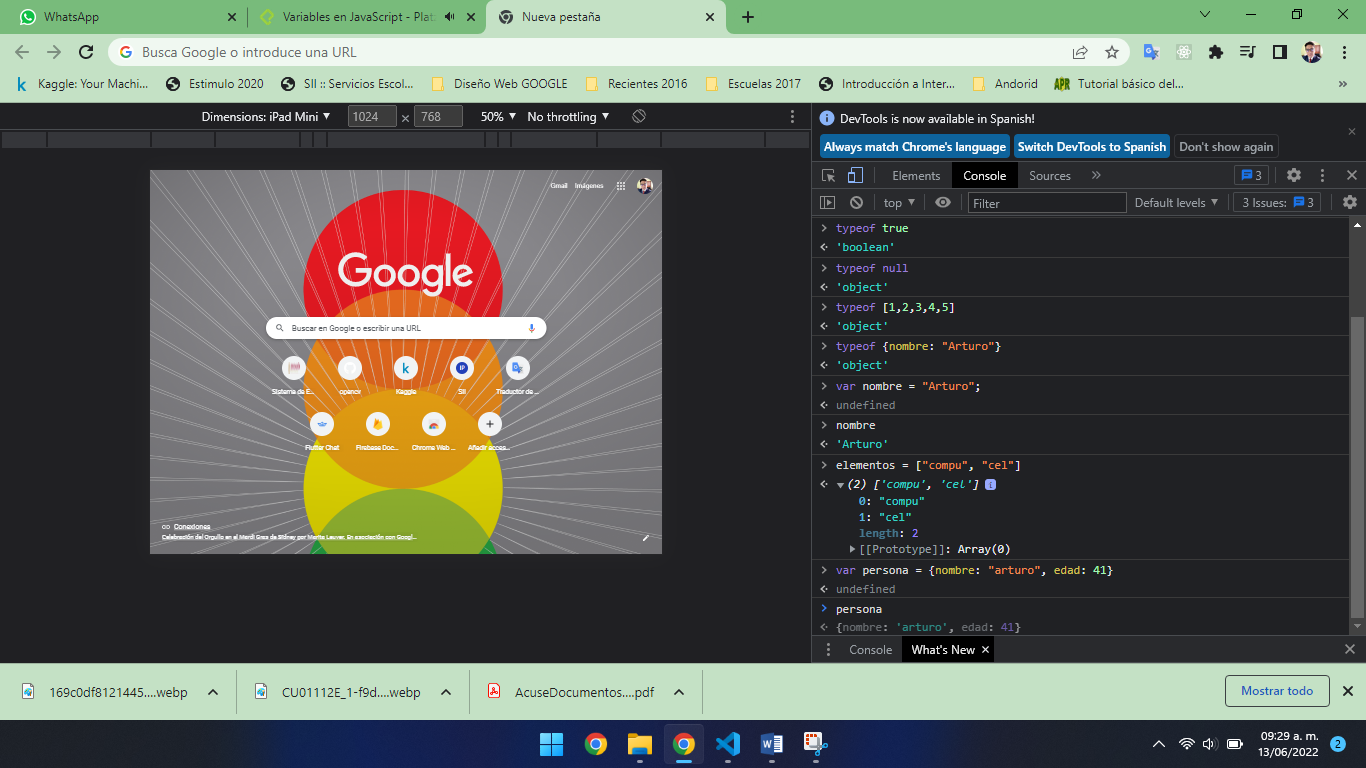
**Qué significa que Javascript es Backwards Compatible**

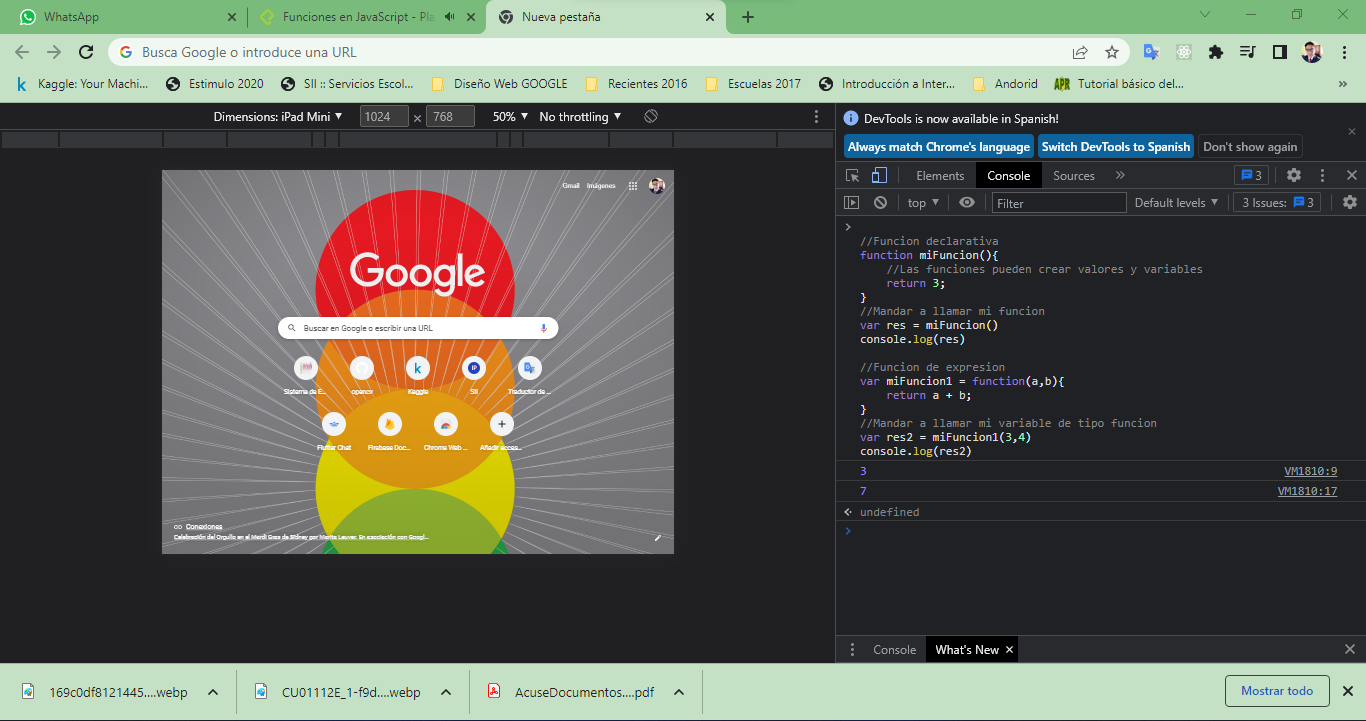
Todas las funciones nuevas que salen para Javascript no dañarán el trabajo ya hecho previamente, pero no se podrá utilizar en nuestro entorno de trabajo inmediatamente. Para solucionar esto está [Babel](https://platzi.com/blog/que-es-babel/), que permite usar las nuevas características del lenguaje, pero lo transforma a una versión que el navegador pueda entender.

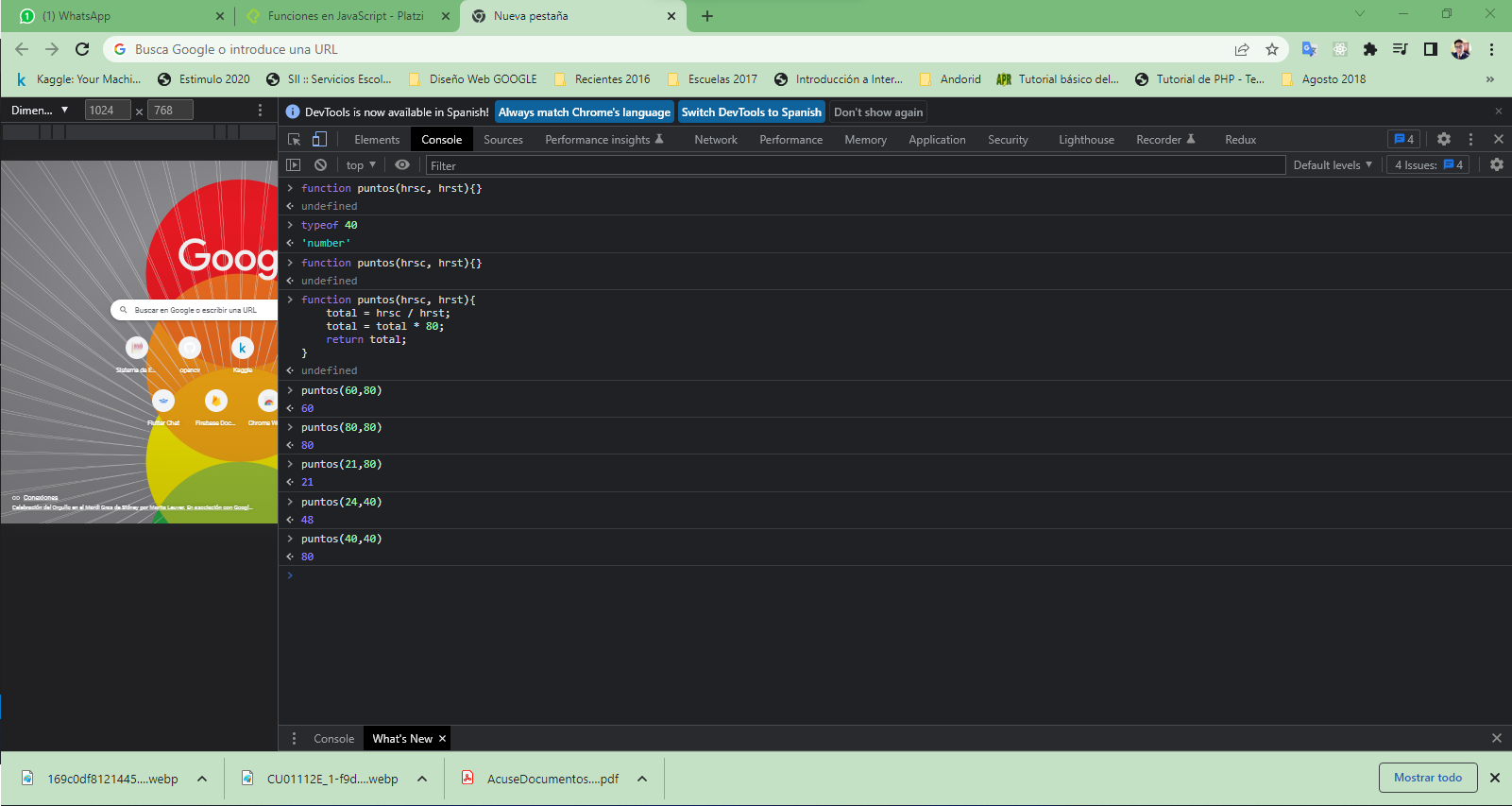


Las funciones son las tareas que va a llevar a cabo el navegador. Existen 2 tipos de funciones  
1) Declarativas  
2) De expresión  
Ambas pueden llevar parámetros, que son los datos que necesitan para ejecutarse.  
Cada parámetro va separado por una coma.  
Cada instrucción que tenga la función debe terminar con ; .  
Si queremos que una función nos dé un numero o dato tenemos que usar la siguiente sintaxis:

return El dato que queremos que nos dé;







Cuando hablamos de funciones en JavaScript, tenemos dos tipos de funciones: Funciones Declarativas (function declaration / function statement) y Expresiones de función (function expression / funciones anónimas).

### Funciones Declarativas:

En las funciones declarativas, utilizamos la palabra reservada function al inicio para poder declarar la función:

**function** **saludar**(nombre) {

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

}

saludar('Diego');

#### Expresión de función:

En la expresión de función, la declaración se inicia con la palabra reservada var, donde se generará una variable que guardará una función anónima.

**var** nombre = **function**(nombre){

console.log(`Hola ${nombre}`)

}

nombre(‘Diego’);

En la expresión de función, la función podría o no llevar nombre, aunque es más común que se hagan anónimas.

### Diferencias:

A las funciones declarativas se les aplica hoisting, y a la expresión de función, no. Ya que el hoisting solo se aplica en las palabras reservadas var y function.

Lo que quiere decir que con las funciones declarativas, podemos mandar llamar la función antes de que ésta sea declarada, y con la expresión de función, no, tendríamos que declararla primero, y después mandarla llamar.

#### **Es la palabra clave que necesita una función para devolver una salida:**

a

return

b

output

c

stdcout

d

resolve

#### **Además de la sintaxis ¿cuál es la principal diferencia entre una función declarativa y una expresión de función?**

a

Ninguna, solo cambia la sintaxis.

b

Podemos llamar una función declarativa antes de ser declarada.

#### **En JavaScript es posible declarar e inicializar una variable en la misma línea:**

a

Verdadero.

b

Falso.

#### **Son los tipos de datos primitivos que maneja JavaScript:**

a

Character, Number, Float, Double y Boolean.

b

String, Number, Boolean, Null y Undefined

c

Array, objeto, número y string.

d

De tipo primitivo y tipo objeto.

#### **¿En qué tipo de proyectos podemos utilizar JavaScript?**

a

Aplicaciones web en fronted y backend solamente.

b

Aplicaciones web frontend únicamente y aplicaciones móviles.

c

Frontend, backend, aplicaciones móviles, aplicaciones de escritorio y IoT.

d

Aplicaciones web frontend únicamente.

#### **¿Por qué nace JavaScript?**

a

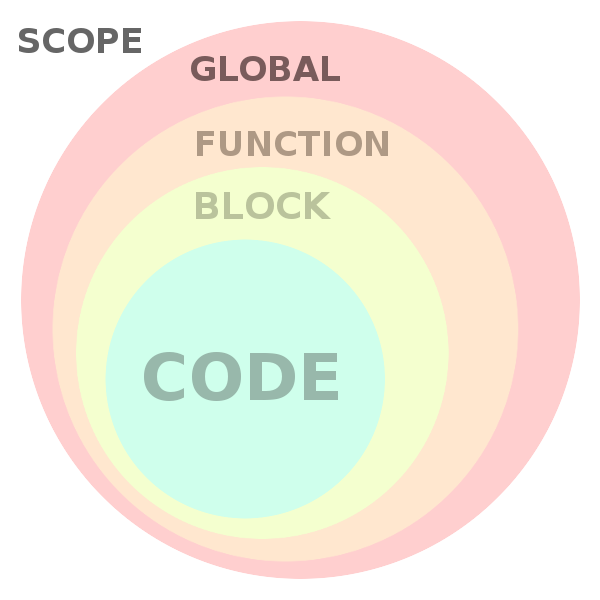
Para darle interacción y dinamismo a los sitios web.

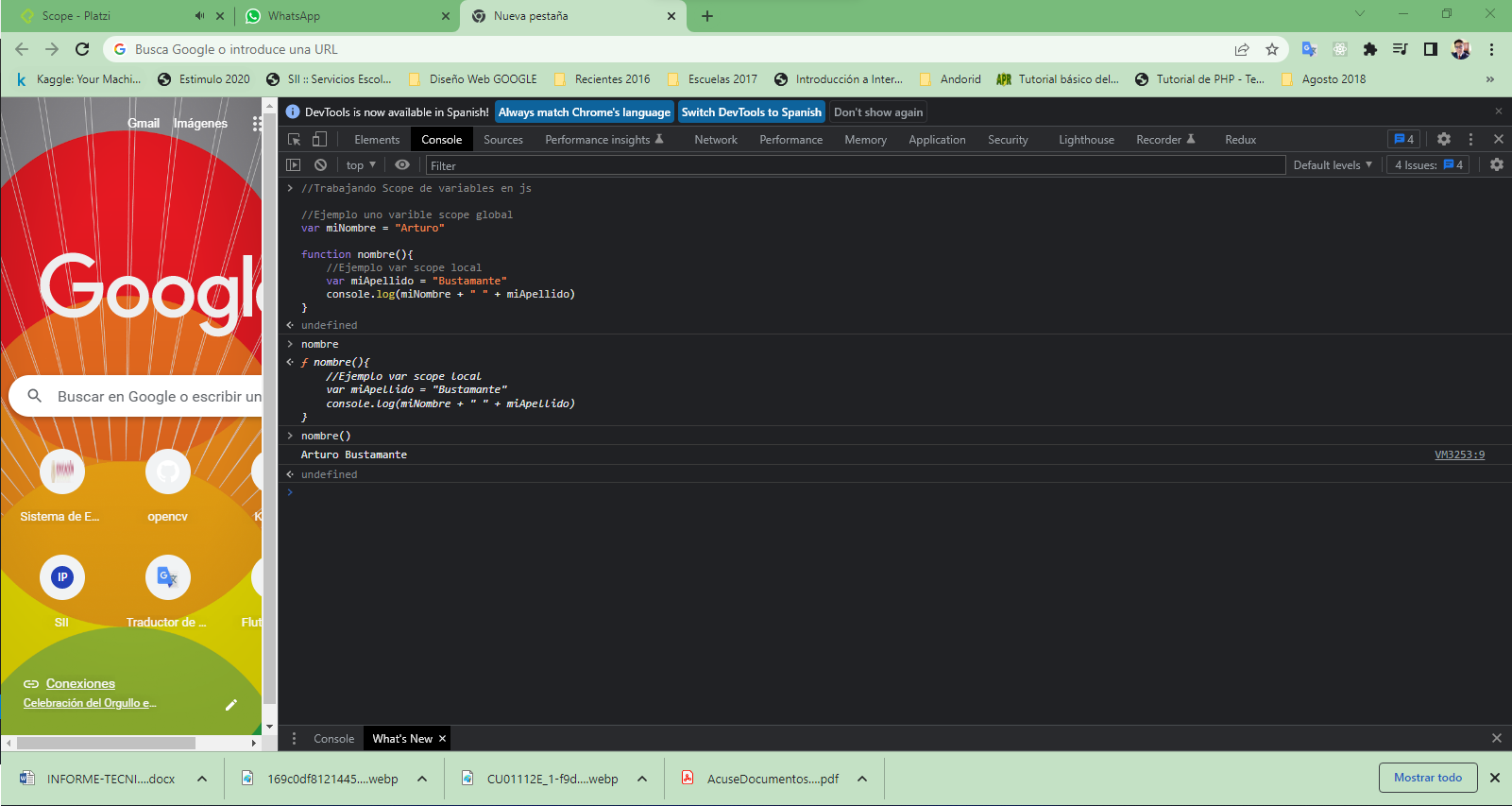
b

Para darle estilo a las etiquetas HTML de los sitios web.

c

Como una competencia directa a Java como lenguaje de programación.





Hoisting solo para versiones antes de la 6 despues se ocupa let y const

Cuando las variables y funciones se procesan antes de que se ejecute cualquier tipo de codigo

¿Qué es Hoisting?  
En JavaScript, las declaraciones (por ejemplo, de variables o funciones) se mueven al principio de su scope o ámbito. Este comportamiento se conoce como hoisting y es muy importante tenerlo en cuenta a la hora de programar para prevenir posibles errores.

* Las funciones siempre se mueven arriba del scope. Por lo tanto, podemos elegir donde declararlas y usarlas.
* La declaración de las variables se mueven arriba del scope, pero no la asignación. Antes de usar una variable, habrá que crearla y asignarla.

Realice un resumen sobre lo que yo entendí de lo que es el *hoisting*, aunque no sé que tan correcto sea lo que yo aprendi, igual se los dejo esperando les sea de ayuda a entender el concepto de mejor manera:

El **Hoisting** es un proceso del compilador de JavaScript, que consiste en que la *declaracion de las variables* y las *funciones* son llevadas al inicio del codigo, sin importar su posicion, para su procesamiento, sin embargo, la inicializacion de las variables no es llevada al inicio del codigo para su compilacion, sino solo su declaracion, lo cual suele dar espacio a errores cuando se declara una variable sin inicializarla y se procesa en el codigo antes de haber llegado a su inicializacion, lo cual nos suele dar una variable con valor ***undefined***, ya que la variable sí fue almacenada en memoria, pero no se le asigno un valor hasta despues de su ejecución.

Aqui un ejemplo de esto:

saludo();

**function** **saludo**() {

console.log("Hola " + nombre);

}

**var** nombre = "Aaron";

Coerción es la forma en la que podemos cambiar un tipo de valor a otro, existen dos tipos de coerción:  
Coerción implícita = es cuando el lenguaje nos ayuda a cambiar el tipo de valor.  
Coerción explicita = es cuando obligamos a que cambie el tipo de valor.

## Operador de asignacion

| **Simbolo** | **Descripcion** |
| --- | --- |
| = | operador de asignacion |

## Operadores de comparacion

| **Simbolo** | **Descripcion** |
| --- | --- |
| == | igual que |
| === | estrictamente igual que |
| > or >= or >== | mayor o mayor igual que |
| < or <= or <== | menor o menor igual que |
| != or !== | diferente que |

## Operadores aritmeticos

| **Simbolo** | **Descripcion** |
| --- | --- |
| + | operador suma este se utiliza para concatener dos cadenas de texto. |
| - | operador resta |
| \* | operador de multicacion |
| / | operador de division |
| % | operador de modulo |
| \*\* | operador de potenciacion |

tambien se les conoce como operadores binarios. por que toman dos valores y generan un resultado.

## Operadores logicos

| **Simbolo** | **Descripcion** |
| --- | --- |
| ! | NOT niega un valor |
| && | AND |
| || | OR |



#### **¿Qué diferencia hay entre comparar con "'==" y "==="?**

a

Uno asigna valores y el otro los compara.

b

No hay diferencia, funcionan igual.

c

"==" compara valores. "===" compara valores y tipo de dato.

#### **¿Cuál es la naturaleza de los valores booleanos?**

a

true y false.

b

empty y null.

c

NaN y undefined.

d

null y none.

#### **¿En qué consiste la coerción explícita?**

a

En obligar cambiar el tipo de una variable a otra.

b

En que una variable cambia su tipo a otra sin que forcemos el cambio.

c

En declarar el tipo de dato de una variable.

SALTAR PREGUNTA

#### **¿Qué valor se asigna por defecto a las variables cuando caen en hoisting?**

a

string.

b

constant.

c

undefined.

d

integer.

#### **En el scope local de una función puedo usar variables:**

a

Solo variables del scope global.

b

Variables locales de esa y de otras funciones.

c

Variables del scope global y locales de esa función.

d

Solo variables del scope local de esa función.

# Condicionales: If, Else, else if

Switch

# Condicionales

#### **¿Qué pasa si no coloco la sentencia `break` en un `switch`?**

a

Se evalúan todos los casos a excepción de aquel que carece de “break”.

b

No pasa nada, la sentencia es solo una formalidad y buena práctica.

c

Se validan todos los casos como verdaderos y se realizan las acciones asignadas a cada uno.

d

La estructura “switch” arroja un error y no se ejecuta el código.

¿Qué retorna el siguiente código? if (false === !true) { console.log(false == true) } else { console.log(true === !false) }

a

false

b

true

c

undefined

#### **¿En qué casos es conveniente utilizar `switch`?**

a

Únicamente para un par de casos de condiciones.

b

Cuando tenemos múltiples casos a evaluar y evitar el uso de múltiples “if else”.

c

Siempre que queramos evaluar pocos o muchos casos de condiciones.

d

Exclusivamente para manejar menos de 10 “if else”.

#### **¿Para qué nos sirve el operador ternario "?" ?**

a

Para concatenar las salidas de if y else en un solo resultado.

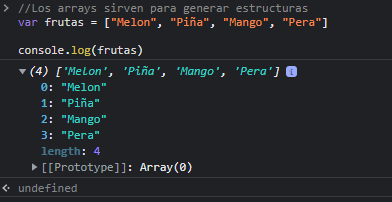
b

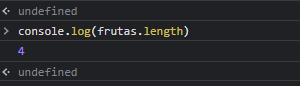
Para definir una condición, el resultado cuando se cumple y cuando no se cumple, en una sola línea.

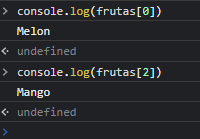
c

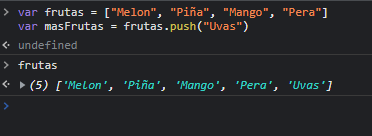
Para anidar varios "if else" en una sola línea de código.

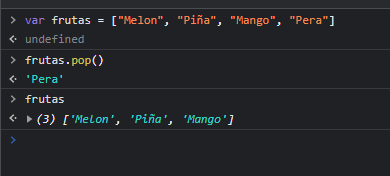
# Arrays

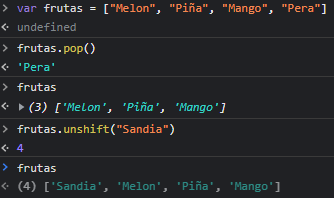


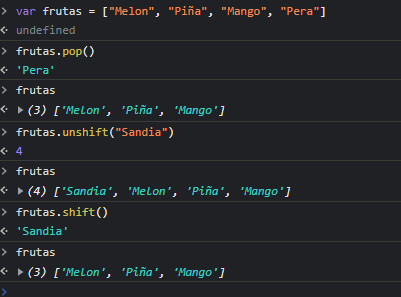














#### **Son los elementos principales de un array:**

a

Elementos y tipo de dato.

b

Métodos y funciones.

c

Índice y elemento.

#### **Un array se mantiene inmutable, es decir, no puede aumentar o disminuir su tamaño. Esta afirmación es:**

a

Falsa.

b

Verdadera.

SALTAR PREGUNTA

#### **Los arrays pueden contener más arrays, esto es:**

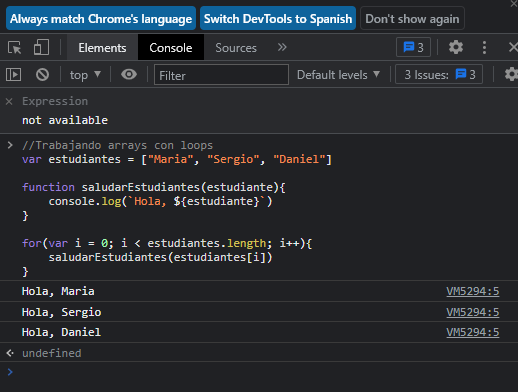
a

Verdadero.

b

Falso.

# Loops: For y For...of



Ejemplos de formas de tipear un for

//Trabajando arrays con loops

var estudiantes = ["Maria", "Sergio", "Daniel"]

function saludarEstudiantes(estudiante){

    console.log(`Hola, ${estudiante}`)

}

//For clasico

for(var i = 0; i < estudiantes.length; i++){

    saludarEstudiantes(estudiantes[i])

}

//For simplificado

for(var estudiante of estudiantes){

    saludarEstudiantes(estudiante)

}