# **JAVASCRIPT**

* **HISTORIA DE JAVASCRIPT**

En los 90 se empezaron a construir sitios web más “bonitos” por lo que las aplicaciones eran más complejas exigiendo formularios más complejos, además el internet era lento (30kbps), por esta razón surgió la necesidad de un lenguaje de programación que se ejecutaran del lado del cliente.

**Páginas antes**

Si faltaba un campo en el formulario tocaba esperar que el servidor dijera que cosas se tenía que llenar



**Creador: Brendan Eich** empleado de Netscape convino varias tecnologías para prevenir este envío de información innecesaria al servidor, este lenguaje se llamó LiveScript, sin embargo, por temas de marketing lo llamaron JavaScript (por que Java era un lenguaje que estaba de moda muy popular)

**Microsoft** también tenía la necesidad de poder ejecutar este mismo código para hacer las validaciones del lado del cliente, por lo que copiaron JavaScript y le pusieron JScript (siendo básicamente lo mismo)

**Netscape** sin animo de entrar a una guerra de tecnologías decide estandarizar el lenguaje de Javascript, por lo que para **1997** envía la versión **1.1** de este lenguaje a la asociación europea de fabricantes de computadoras o mejor conocido como ECMA, la cual creo un comité para estandarizar el lenguaje de script multiplataforma sin importar la empresa que lo usara, el cual se llamó **TC39,** quien a su vez crea la primera especificación llamada **ECMA-262,** la cual se definió por primera vez como el lenguaje de **ECMAScript.**

**ECMAScript** es un estándar para lenguaje de scripting (lenguajes diseñados para integrarse y comunicarse con otros lenguajes, su principal diferencia con los lenguajes de programación es en términos de compilación, estos no necesitan compilarse es decir ser transformados a código binario, sino que se interpretan directamente desde el código fuente, por ende, son más rápidos)

**Nota:** JavaScript es una implementación que realizo la empresa Netscape del estándar ECMAScript.

* **USOS DE JAVASCRIPT**

JavaScript surge por la necesidad de enviar información al servidor la más procesada posible, para que este trabaja únicamente con peticiones válidas.

Hoy en día JavaScript también puede correr de forma independiente en el servidor y no solo en el navegador web.

Utilizado para:

* crear aplicaciones web (desde la caja de búsqueda, animaciones, cálculos, etc). Un ejemplo es Google Maps (todas las animaciones , cálculos lo hace javascript)
* crear presentaciones <https://revealjs.com/>
* servidores web que corren enteramente con JavaScript y eso gracias a **Node ,**  con el conocimiento adquirido de JavaScript se va poder crear servidores , manejar peticiones, crear archivos en el file sytem del servidor, conectarnos a la base de datos, recuperar información, generar información de respuesta a las peticiones, comunicación en tiempo real con sockets.
* juegos <https://js13kgames.com/games/underrun/index.html> (juego que pesa menos de 100kb),
* aplicaciones móviles utilizando frameworks y librerías como Ionic, react Native, NativeScript
* crear aplicaciones nativas para Windows, Linux y OS X utilizando Electron

**Nota:** Existen muchas tecnologías que nos ayudan a trabajar con JavaScript de manera sencilla, como Angular, Vue o React (frameworks y librerías más populares en Javascript). Si nosotros comprendemos JavaScript utilizarlas será sencillo.

* **HOLA MUNDO EN JAVASCRIPT**

Al abrir un navegador web estos ya traen su versión de JavaScript (cada uno tiene una implementación del estándar ECMAScript por lo que no todos son iguales).

Abrir las herramientas de desarrollo – consola

**console.log(“Hola mundo”); 🡪 esto lo escribe en la consola**

Imprime en consola lo que tengas en comillas (cadena de caracteres)

**Console** 🡪 es un objeto

**.log** 🡪 es una funcion dentro del objeto 🡪 es un método

**Nota:** al imprimir sale también un undefined y esto es ya que todas las funciones de JavaScript tienen un return

**Document.write(“Hola mundo”) 🡪 esto lo escribe en el html**

**Document** 🡪 hace referencia a todo el html

**.write 🡪** para escribir en el documento

**Nota:** recordar que **Node** nos permite correr JavaScript fuera del navegador

Al presionar **node** dentro de un terminal entramos en un modo de edición y ya se puede escribir código JavaScript y se tendrá la misma respuesta obtenida en el navegador.

En la terminal de VS Code abrimos la terminal y ejecutamos un archivo Javascript con el comando node nombreArchivo Para que nos aparezca el undefined debemos hacer: **Console.log(console.log(“Hola mundo”)** 🡪 imprime el resultado de la operación dentro de los paréntesis

Crear un archivo index.html (pagina web). Para crear la estructura básica presionamos ! tecla TAB. En el HTMl body podemos ejecutar nuestros Scripts envolviéndolos en la etiqueta **Script**

|  |
| --- |
| <!DOCTYPE html>  <html lang=”es”>  <head>       <meta charset=”UTF-8”> <!-- para que reconozca bien los caracteres -->      <meta http-equiv=”X-UA-Compatible” content=”IE=edge”>      <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>      <title>Document</title>  </head>  <body>      <script>          console.log(“Hola mundo en HTML”)      </script>  </body>  </html> |
|  |

Hacer esto es mala práctica. No es bueno mezclar el HTML con JavaScript porque la idea es que las paginas solo se enfoquen en HTML.

Si queremos importar el app.js en este archivo se haría lo siguiente

|  |
| --- |
| <!DOCTYPE html>  <html lang=”es”>  <head>       <meta charset=”UTF-8”> <!-- para que reconozca bien los caracteres -->      <meta http-equiv=”X-UA-Compatible” content=”IE=edge”>      <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>      <title>Document</title>  </head>  <body>      <script>          console.log(“Hola mundo en HTML”)      </script>      <script src=”app.js”></script>  </body>  </html> |

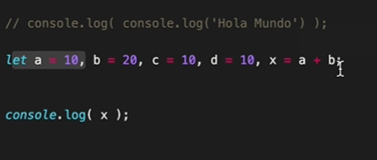
**Nota:** es recomendable que estos scripts vayan dentro de la página (body) porque si se le coloca en otro lado, el navegador cargará primero el archivo JavaScript antes que el HTML, y si este archivo es muy pesado o tiene algún bloqueo el sitio no se renderizara hasta que todo el código de JavaScript termine dando la impresión que el sitio es lento.

**Comentarios:** Son líneas de código que el intérprete de JavaScript ignorará a la hora de ejecutarlo y nos sirven para describir que hace cierto código. Se puede comentar en VSCode con **ctrl + /**

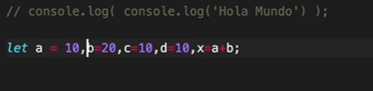
* **• CREACIÓN DE VARIABLES Y CONSTANTES**

**Variables:** Una variable es un contenedor de información que apunta a un lugar en memoria. Dicha información puede cambiar a futuro.

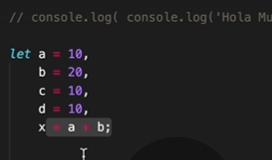
* **Var** forma antigua de crear variables
* **Let** adaptada a nuevas versiones de javascript a partir de 2015, donde se dio la actualización mas grande que ha tenido JavaScript.



JavaScript ignora cualquier enter o espacio. Para JavaScript es como que el código se encuentra así:

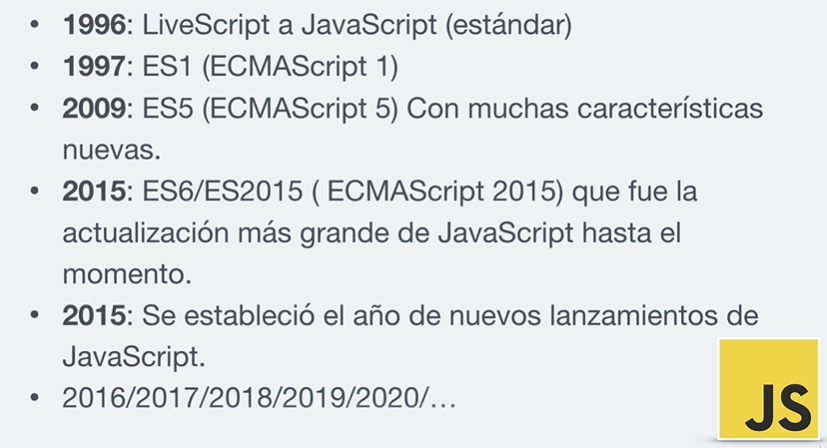


Buena práctica sería:



* **Constante:** hace referencia a un valor de memoria, el cual no podrá cambiar.

**Versiones de JavaScript**



EC5 🡪 soportado por la mayoría de los navegadores Web, tiene la mayor compatibilidad posibible

EC6/ES2015, ES7/ES2016, ES8/ES2018 🡪 son soportados por la mayoría de los navegadores web modernos, perdiendo la compatibilidad con navegadores viejos.

**Nota**: Para poder utilizar estas características se hace uso de **Polyfills**, códigos que proveen el funcionamiento de una nueva característica de JavaScript (ES6), en versiones viejas como ES5. Es decir, es un código que me permite usar características modernas en versiones viejas de JavaScript.

**Las constantes** al no tener propiedades para establecer o cambiar el valor son mas ligeras que un let o un var.

* **INTRODUCCIÓN A LA CONSOLA DE JAVASCRIPT**

JavaScript es un lenguaje interpretado por lo que al cargar el archivo JavaScript se le dice que ejecute línea por línea. Sin embargo, puede haber muchas cosas que alterarían ese orden.

**Los mensajes de consola** son utilizados para no interferir con el flujo normal del programa. Nos permiten conocer como se encuentra una variable en determinado punto.

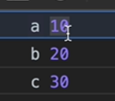
Console.log(x)

Console.warn(x)

Console.error(x)

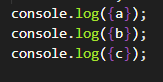
Console.log es un método que recibe varios argumentos, todo lo que se le coloque dentro del paréntesis son los argumentos.

**Los colores** en la consola son importantes. **Gris**  es un string, **morado** es un numero o booleans.



**Nota:** Al ver paréntesis en JavaScript significa que eso es una **función** o un **método** (funcion dentro de un objeto)

Para imprimir el nombre de la variable con su valor se utiliza las **{ }**

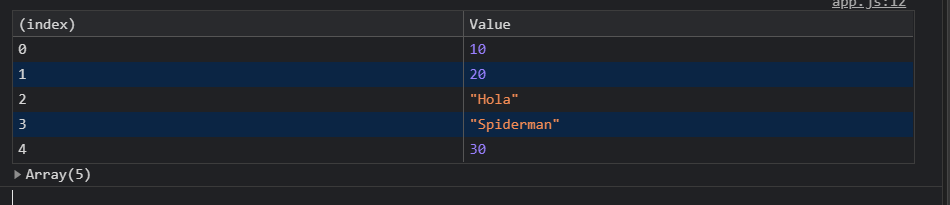


Para cambiar el color del texto que se imprime en la consola, se utiliza **%c** para incrustar un estilo y como segundo argumento alguna **propiedad de CSS**



Cuando se necesita imprimir muchas variables y estas tiende un orden lógico, se utiliza el **console.table** que recibe un arreglo utilizando **[ ]**

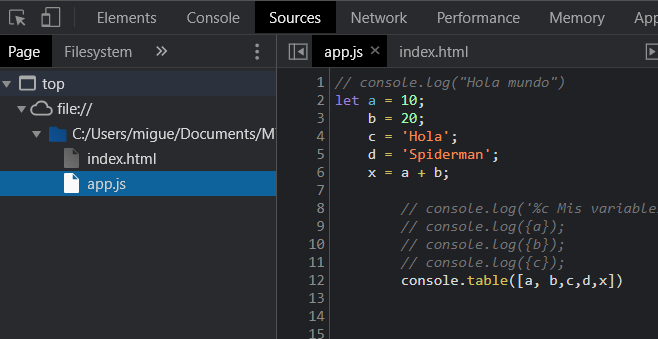




**Nota:** cuando encontremos las constantes en mayúsculas quiere decir que son constantes de Entorno.

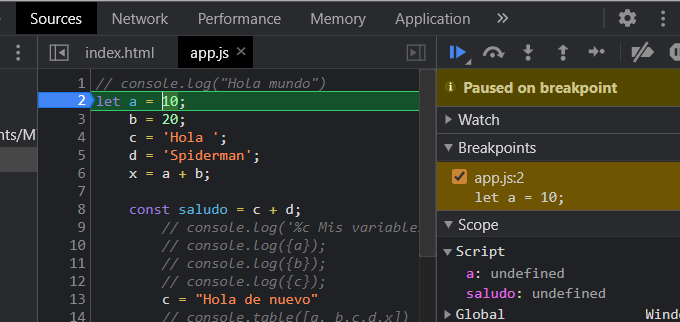
* **Depuración y breakpoints**

Consola de Google Chrome – Sources 🡪 tenemos el archivo index.html y su archivo dependencia (app.js)

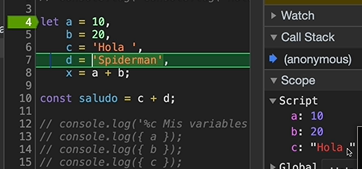


**Hay dos formas para evaluar el código.**

* **1. Tradicional o desde el navegador:** dar clic en una línea de código para establecer un **breakpoint** que básicamente es que cuando la aplicación llegue a ese punto se detendrá ahí, mostrando todo lo definido en ese momento



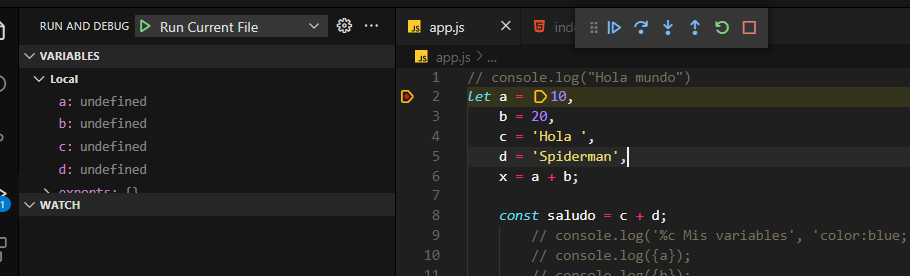
El **intérprete** de JavaScript que se encuentra interno en Google Chrome, el cual es **V8** ayuda a determinar en qué línea se encuentra el código del programa.



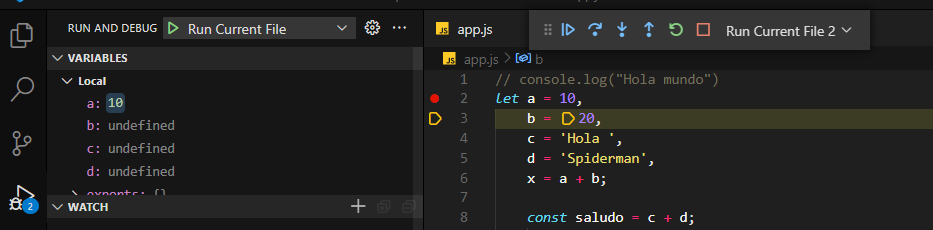
**NOTA:** los comentarios para JavaScript no existen. Estos únicamente nos sirven para conocer como funciona el código.

* **Desde VScode** dar clic en una línea hasta que aparezca el circulo **rojo** ir a **Ejecutar** y seleccionar **empezar depuración** y escoger **node**

JavaScript automáticamente al abrir el archivo hace una revisión o lee todas las definiciones de las variables y funciones que este utiliza, no las ejecuta, sino que solamente graba en un índice donde están dichas referencias para posteriormente ejecutarlas. Es por este que tenemos lo siguiente como **undefined**

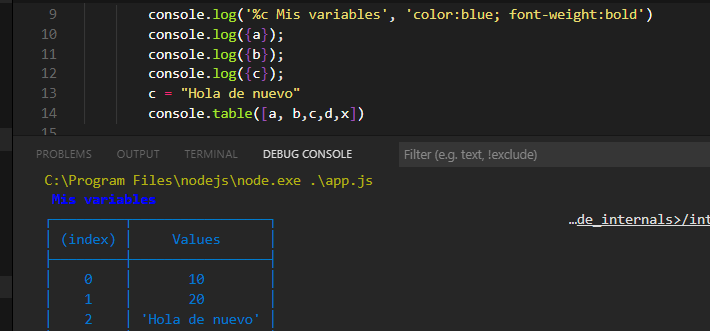


Al ejecutar línea por línea se van a ir asignando los valores a cada variable



**NOTA:** La depuración nos ayuda a ver en qué punto está fallando nuestra aplicación.

Cuando ejecutamos la aplicación **F5** podemos ver los mensajes en la pestaña **Debug Console.**

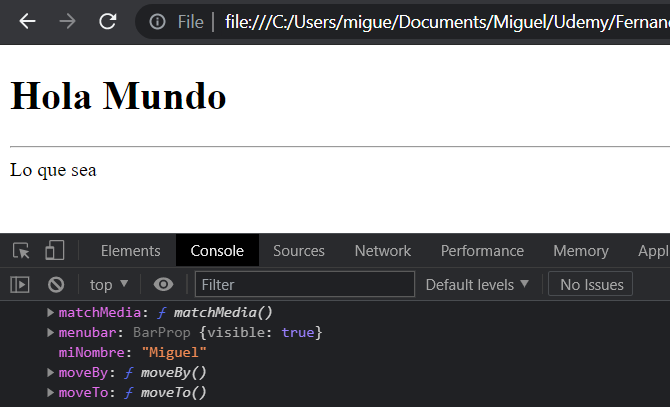


* **ORDEN Y LUGAR DE LAS IMPORTACIONES**

Las importaciones de código javascript es mejor hacerlas antes del cierre del **body** ya que al colocarlas en el **head** estas se ejecutarán primero, y después la parte del html, por lo que darán la impresión de que la página se carga más lento.

**Nota: alert()** es una funcion que se encuentra dentro del objeto global **window. Assets** son los recursos del sitio web. Normalmente se debe colocar una estructura de directorios para tener ordenado el proyecto.

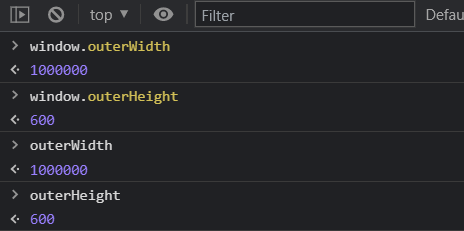
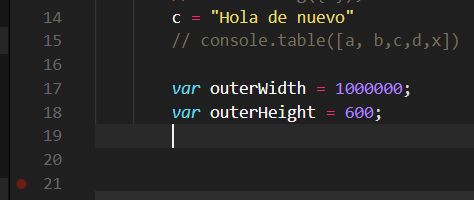
Al declarar una variable con **var** la coloca en un objeto global denominado **window** y esto no es una buena práctica, ya que si colocamos **window** en el navegador web obtendremos toda la información (métodos y atributos) de dicho objeto.



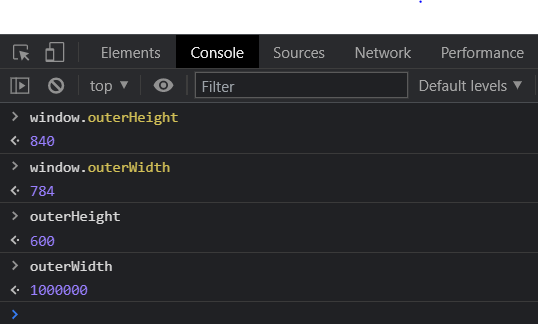
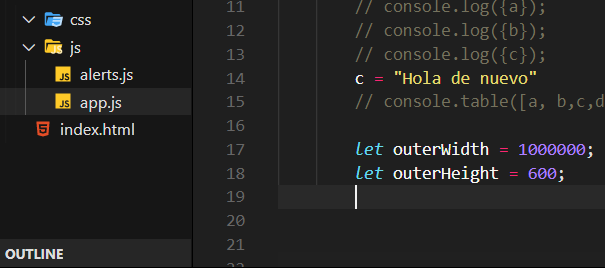
* **PROBLEMAS CON LA DECLARACIÓN DE VARIABLES CON VAR**

**Let y const no sobrescriben las variables que se encuentran en el objeto global window o donde sea que este corriendo JavaScript.**

**MAL**



**BIEN**



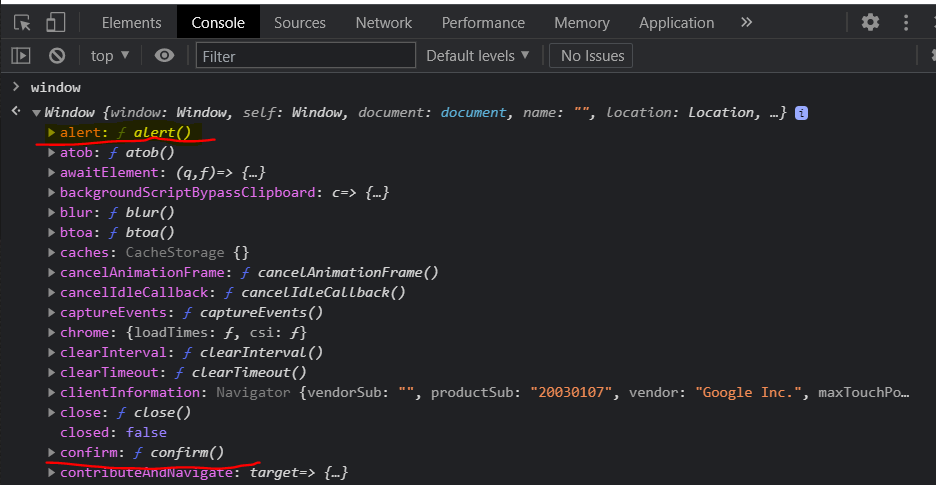
**Nota: outerWidth y outerHeight se utilizan para obtener el ancho y alto de la pantalla.**

**En JavaScript cualquier variable que no este Inicializada tendrá el valor de undefined**

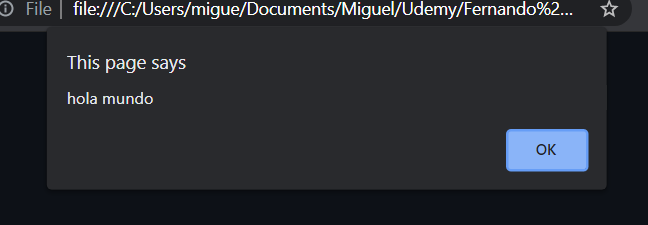
**Var solo se usa para implementar compatibilidad con navegadores por debajo del 2015.**

* **PROMPTS, ALERTS, CONFIRMS.**

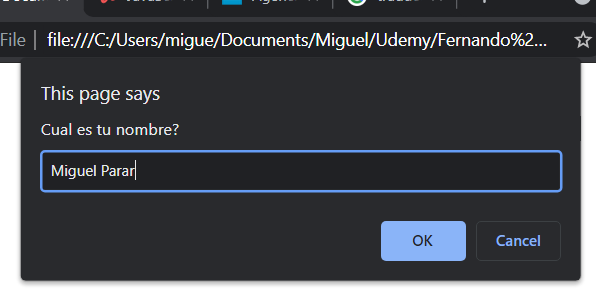
Son formas de ingreso de información que contiene instrucciones bloqueantes, es decir no dejan ejecutar ninguna línea de código hasta q el usuario haga clic. Además son métodos que vienen dentro del objeto **window**



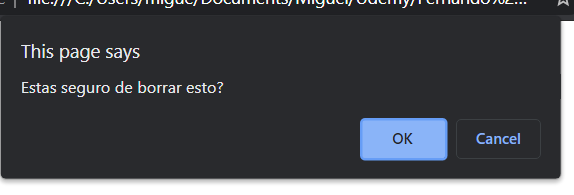
* **alert()**



* **prompt()** recibe argumentos de entrada, este retorna lo que el usuario escriba (string)



* **confirm** retorna **true** si el usuario acepta o **false** si rechaza el mensaje.



**Nota:** string vacio no es lo mismo que null ni que undefined.

Estos 2 métodos solo funcionan en el navegador y no fuera de él. La variable **Global** en cambio solo funciona fuera del navegador.

**SECCIÓN #3**

* **TIPOS DE DATOS PRIMITIVOS**

**Tipo de dato 🡪** describe el contenido del valor que tiene la variable.

**Primitivos 🡪** Es una información que no es un objeto y son inmutables.

**Boolean:** true o false

let esMarvel = true // typeof esMarvel 🡪 boolean

**Null:** sin valor en lo absoluto

let soyNull = null // typeof soyNull🡪 object (ver Nota)

**Undefined:** una variable declarada que aún no se le asigna valor

let superPoder // typeof superpoder 🡪 undefined

**Number:** integers, floats, etc

let edad = 3 o let edad = 3.3 // typeof edad 🡪 number

**String:** cadena de caracteres. Ejemplos: nombres, palabras, frases, etc. Pueden ser declarados con comillas simples, dobles o backticks

**Symbol:** es un valor único que no es igual a ningún otro valor. (al comparar symbols en las mismas circunstancias nunca son iguales). Su principal es uso es para identificar propiedades de manera única. Permiten crear identificadores únicos a JavaScript

let symbol1 = Symbol(‘a’) //typeof symbol1 🡪 symbol

**Notas:** JavaScript es un lenguaje débilmente tipado, es decir que no se le coloca de que tipo va a ser la variable, sino que JavaScript va a inferir esto.

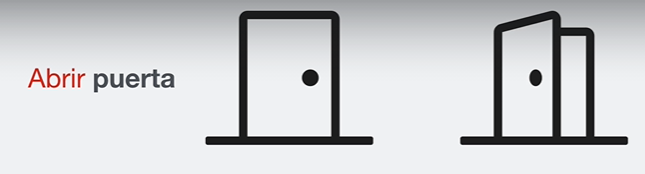
JavaScript en la primera barrida, antes de ejecutar línea por línea lee todas las definiciones de las variables.

**Typeof** nombre\_variable es un operador interno que ya existe en JavaScript y me permite conocer que tipo de dato es el valor al que apunta la variable.

Al declarar una variable como **null,** el tipo de dato que le coloca JavaScript es de un objeto, sin embargo, no es así, ya que este es un tipo de dato primitivo. En JavaScript todo son objetos a excepción de los primitivos.

* **PALABRAS RESERVADAS Y NOMBRES DE VARIABLES**

**Palabras reservadas** son palabras que tienen un uso específico para el cual fueron creadas. Solo se deben usar para lo que fueron creadas.



Las palabras reservadas más comunes en JavaScript son:



Es fácil reconocer una palabra reservada porque estas cambias de color en el editor de código.

**Estándar**

* No se puede crear nombre de variables que empiecen con números, pero si con \_ (underscore)
* Se puede utilizar normalmente el $ símbolo de dólar para dar nombre a la variable
* No se puede colocar . (punto) ya que esto para JavaScript significa que se quiere acceder a una propiedad o método de algún objeto.

**Nota** es aconsejable colocar el nombre de los archivos en minúsculas y separar las palabras con un guion debido a que la mayor parte de servidores corren en Linux y pueden traer problemas al momento de utilizar letra capitalizadas.

Página para conocer que palabras son utilizables para dar nombres a las variables <https://mothereff.in/js-variables>, y esta otra página para ver que nombres raros han dado a la variables y que son permitidos <https://mothereff.in/js-variables>

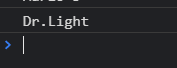
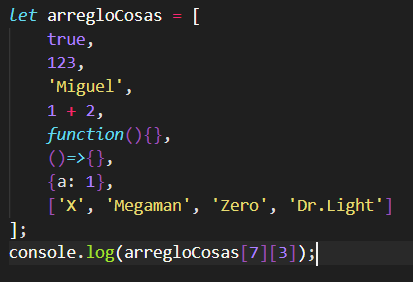
* **ARREGLOS**

**Un arreglo** es un o**bjeto** que contiene un grupo de elementos (usualmente toda su información es del mismo tipo)

const arr = new Array(10) 🡪 no es muy común ver esta forma de definir un arreglo

const arr = [ ] 🡪 forma común de declarar un arreglo

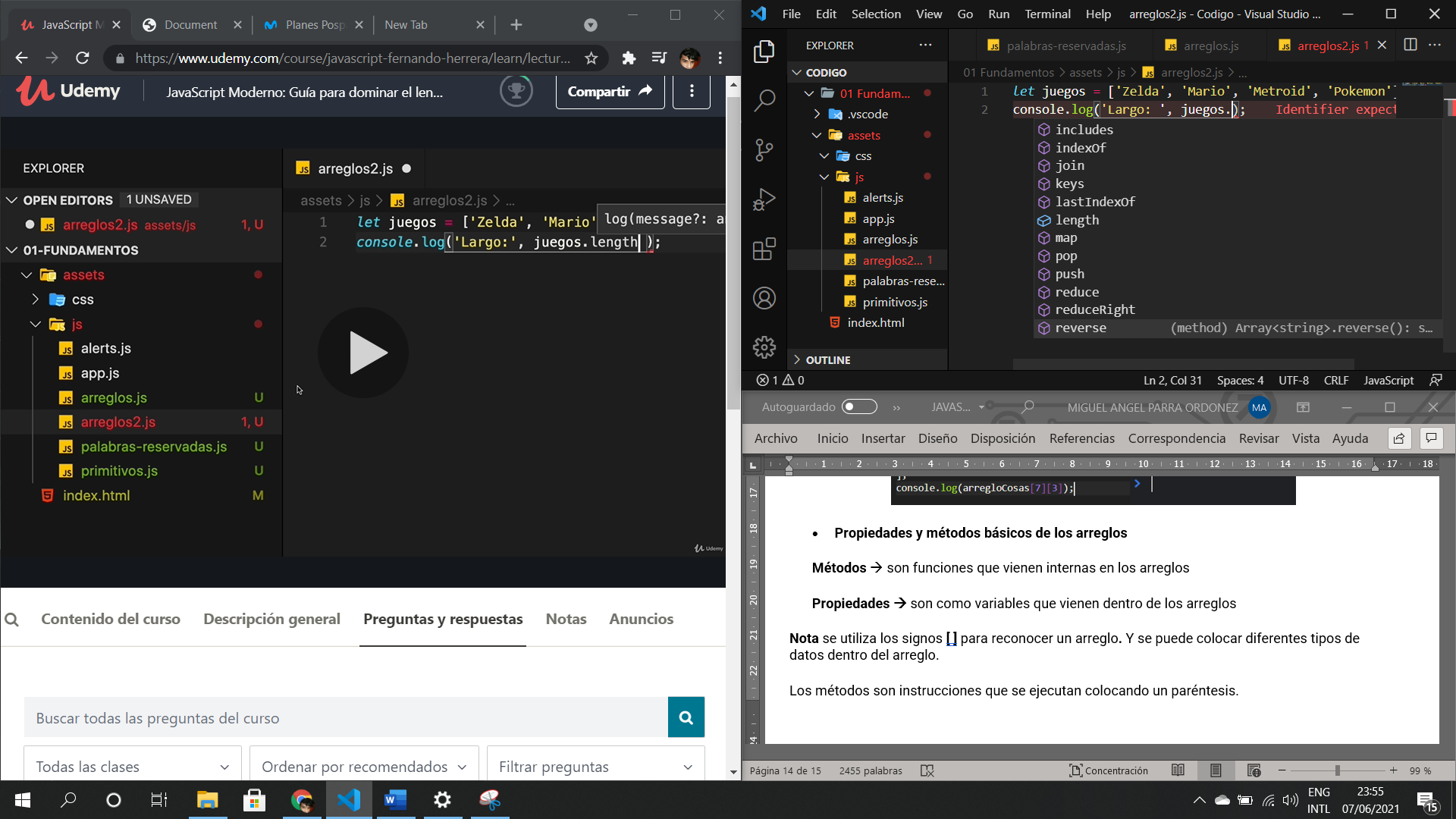
Para obtener una posición especifica del arreglo se utiliza los números junto [ ] 🡪 arreglo[0]



* **Propiedades y métodos básicos de los arreglos**

**Métodos** 🡪 son funciones que vienen internas en los arreglos

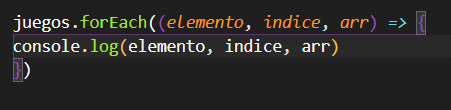
**Propiedades 🡪** son como variables que vienen dentro de los arreglos

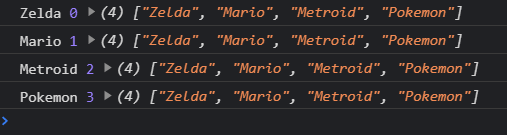


Los métodos con cubo morado se ejecutan colocando un paréntesis y los de color celeste no necesitan el paréntesis porque son propiedades

**forEach**

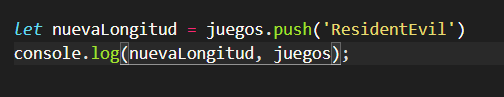
Ejecuta una instrucción por cada elemento presente en el arreglo





**Push**

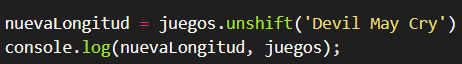
Permite añadir un elemento al final del arreglo y regresa el número de elementos que actualmente contiene el arreglo.





**Unshift**

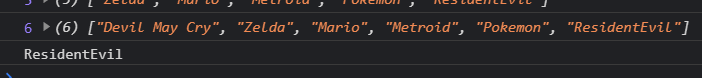
Permite añadir un elemento al inicio del arreglo



**Pop**

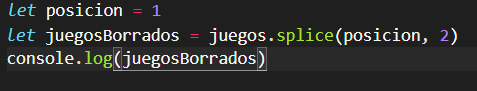
Permite eliminar el último elemento del arreglo. Retorna un string con el elemento eliminado.





**Splice**

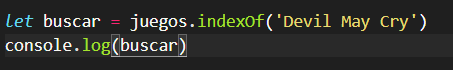
Permite eliminar uno o varios elementos desde alguna posición en específico. Retorna un arreglo con los elementos eliminados.





**IndexOf**

Permite buscar en que posición del arreglo se encuentra determinado elemento. Retorna un número.



Si no encuentra dicho elemento arroja **-1**

**Nota** se utiliza los signos **[ ]** para reconocer un arreglo**.** Y se puede colocar diferentes tipos de datos dentro del arreglo.

Los métodos son instrucciones que se ejecutan colocando un paréntesis, mientras que las propiedades no.

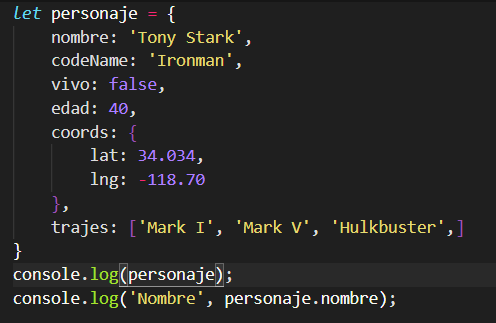
Al colocar sobre un método, este nos dice que parametros recibe y el tipo de dato que retorna

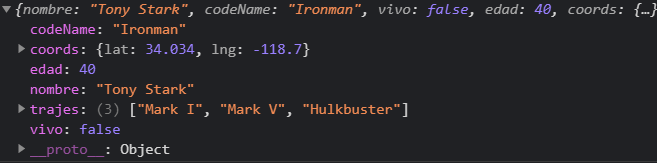
* **OBJETOS LITERALES**

JavaScript tiene dos tipos de datos principales:

* **Primitivos**
* **Objetos** si no es un primitivo entonces es un Objeto.

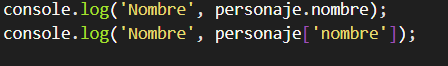
**Los objetos literales** son como **objetos** que tienen pares de valores (clave, valor). Para acceder a las propiedades de los objetos se hace uso de la **notación de punto (.)**



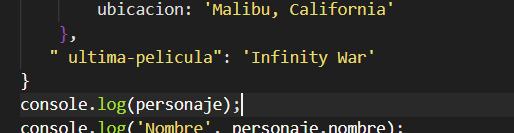


Como se aprecia las propiedades se imprimen en orden alfabético

**Otra forma** de acceder a las propiedades del Objeto es utilizando la **notación de corchetes [clave]**



Si se quiere colocar guion **(-)** en el nombre de las claves se debe utilizar las comillas sencillas o dobles.

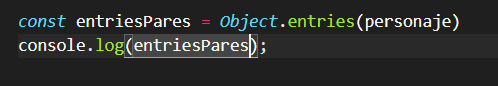


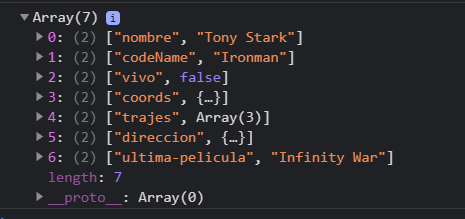
**Para eliminar una propiedad** se utiliza la palabra **delete**

Delete Objecto.clave

Delete Objeto[‘clave’]

**Para transformar un objeto a un array** se utiliza el método **entries** de la **Interfaz Object**





Para **añadir una nueva propiedad** en el Objeto se utiliza la siguiente sintaxis:

Objeto**.**nuevaPropiedad = valor

Para **bloquear tanto el ingreso de nuevas propiedades como modificar las existentes** se utiliza el método **freeze** dentro del **Object.**

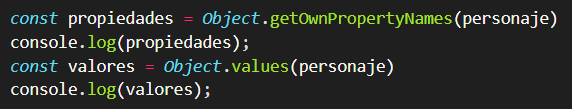
Object.freeze (objeto)

Para **obtener todas las propiedades que tiene el Objeto** se utiliza el método **getOwnPropertyNames** dentro del **Object.** Esto retornara un arreglo con las propiedades

Object.getOwnPropertyNames(objeto)

Para **obtener todos los valores que tiene el Objeto** se utiliza el método **values** dentro del **Object.** Esto retornara un arreglo con las propiedades

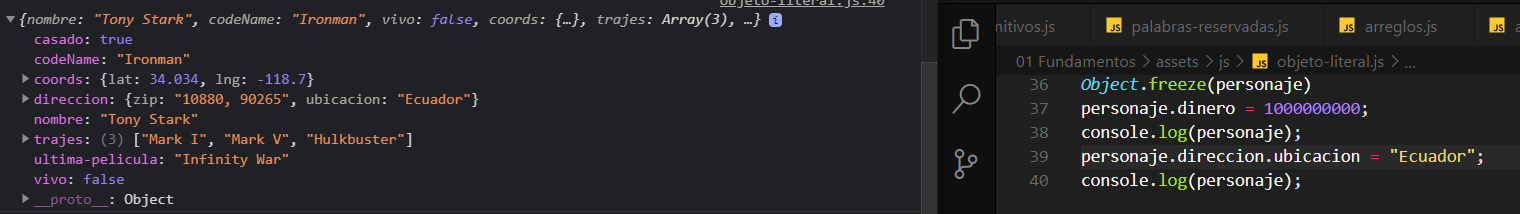
Object.values(objeto)



**Nota:**  las llaves en JavaScript **{ }** se asocian a que se está declarando un **Objeto literal**

Al declarar una variable como **const**  permite bloquear únicamente la asignación (ponerle de otro tipo de dato) mas no aumentar o disminuir datos o propiedades, para bloquear estas se utiliza **freeze**.

Un inconveniente del método **freeze** es que no congela las asignaciones a los Objetos que estén dentro del Objeto. Si se quisiera bloquear a estos Objetos, se llamaría al método **freeze** apuntando a dicho Objeto dentro.



Para obtener más información de las propiedades y métodos que tienen los objetos se puede visitar el siguiente link: <https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Object> (Mozilla MDN)

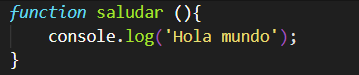
* **FUNCIONES BÁSICAS**

El objetivo de una función es centralizar la lógica de un procedimiento que podríamos necesitar ejecutar varias veces.

Al definir una funcion se utiliza los paréntesis **( )**

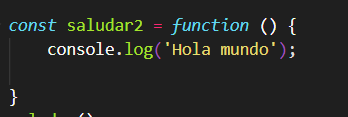
**Formas de crear funciones**

* + **Tradicional**



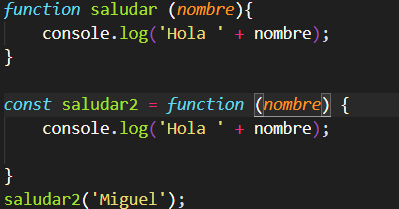
* + **Funciones anónimas**

Funciones que se les puede asignar a una variable (preferible constante) y toman ese nombre



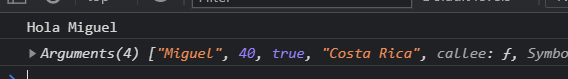
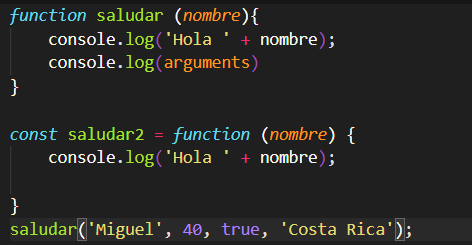
El beneficio de crear funciones así es que al tener la palabra **const** ya no se puede reutilizar el nombre.

**Los parámetros** son variables que reciben las funciones, mientras los **argumentos** son los datos que pasamos al llamar a estas



**Nota:** Se debe recordar que JavaScript hace un barrido previo y saca las referencias de donde están las inicializaciones de variables y funciones por lo que no importa donde se coloquen las funciones ni el llamado. Sin embargo, es buena práctica que estas se definan arriba y abajo se haga el llamado de las mismas.

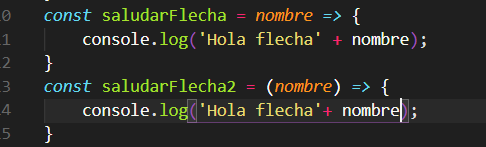
Las funciones tradicionales (las que tienen la palabra **function**) tienen un Objeto implícito llamado **arguments,** el cual obtiene todos los argumentos que han sido enviados a la funcion indistintamente si estos no han sido colocados como parametros



* **FUNCIONES DE FLECHA**

También conocidad como **Lambda functions.** Se introdujeron en el ECMAScript 6 (JavaScript del 2015)

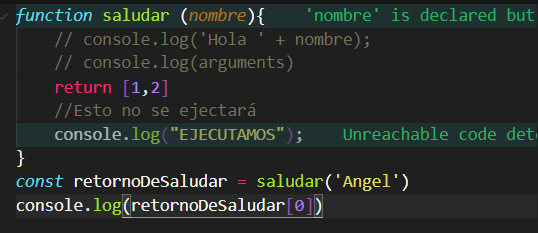
**Nota:** en las funciones flecha los paréntesis son opcionales cuando se envia un solo argumento. Sin embargo, se recomienda que se coloque con paréntesis ya que permite aclarar que esa es una funcion flecha.



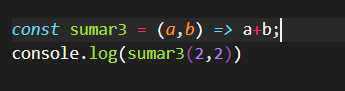
* **RETORNO DE LAS FUNCIONES**

Cada funcion retorna un valor. Si no tiene la palabra **return** explicita, entonces retorna **undefined**

Si queremos retornar 2 cosas, es preferible colocarlas en un arreglo y escoger la posición de retorno que se quiera utilizar desde la llamada a la funcion



**Las funciones de flecha** tienen una característica importante, y es que si lo único que se tiene en la funcion es un **return,** todo puede ser escrito en una línea de código.

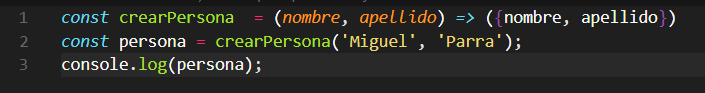


**Nota** todo lo que se encuentra después del **return** no se ejecutará

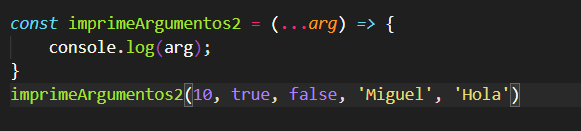
El Objeto “Math” no es necesario importarlo porque ya viene incluido en JavaScript. Este objeto tiene varias funciones matemáticas.

* **PRO TIP: FUNCIONES, ARGUMENTOS Y DESESTRUCTURACION DE OBJETOS**

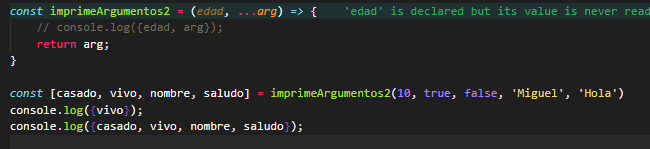
Cuando se quiere retornar un objeto en una funcion flecha (en el caso que no se tenga nada más en el cuerpo de la funcion), sin utilizar el **return** se debe utilizar los paréntesis y llaves ({ })

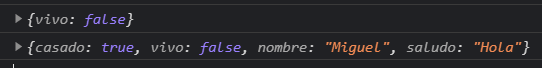


Para poder hacer uso del **Objeto arguments** en una funcion flecha es escribir en la parte de argumentos los tres puntos (…) seguido del nombre que se le quiera asignar. Esto hace referencia a un parámetro **REST** que le dice que todos los argumentos enviados se ingresen a un arreglo



Si se quiere colocar directamente el arreglo que retorna la funcion en distintas variables se puede colocar sus nombres en la declaración como un arreglo [ ]





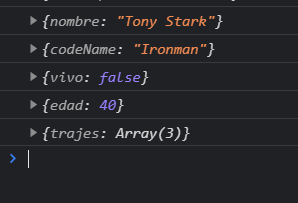
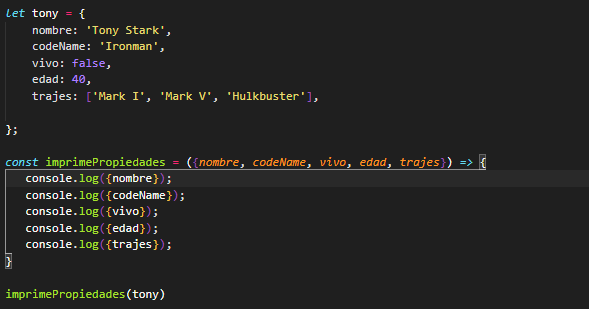
Lo mismo se puede hacer cuando se retorna un **Objeto,** solo que en este caso se utilizan las llaves **{ }** y la clave. Si se quisiera guardar con otro nombre el valor se debería hacer lo siguiente { clave: nuevaClave}



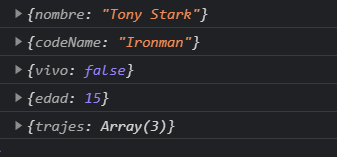
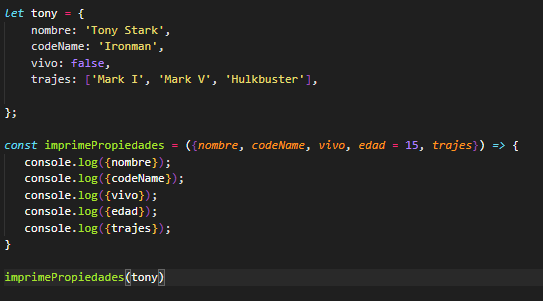


**Destructuración de argumentos**

En la funcion se reciben las **claves** del objeto pero en el llamado enviamos únicamente el **Objeto**



En el caso de que no exista la clave se podría colocar un valor por defecto en los parametros de la funcion



**Nota:** No puede venir ningún argumento después del parámetro REST pero si se puede declarar un parámetro antes del REST.

* **CODIGO FUENTE DE LA SECCION**

<https://github.com/Klerith/javascript-basics/releases/tag/v0.0.2>

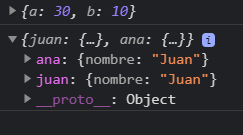
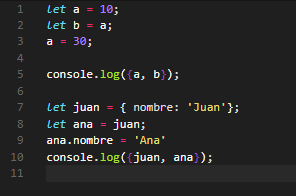
##### SECCION 4

* **VALOR, REFERENCIA y ROMPER LA REFERENCIA**

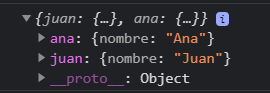
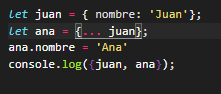
Cuando **se trabaja con primitivos** cualquier tipo de asignación o cuando se envía a una funcion como argumento estamos enviando por **valor,** es decir no afectamos un lugar en memoria.

En JavaScript **todos los objetos** son pasados por **referencia**

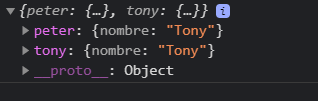
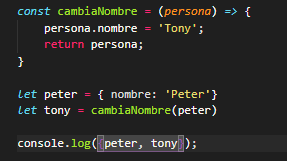
Juan está apuntando a un lugar en memoria y al momento de hacerle la asignación a Ana estamos apuntando al mismo lugar en memoria que Juan por ende cuando cambiamos a Ana también cambiamos a Juan



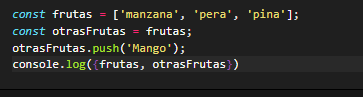
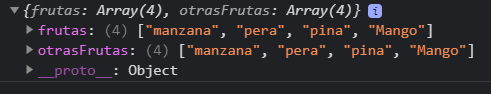
**Una forma de evitar lo anterior** es colocar el **objeto** entre llaves **{ }** y dentro colocar el **operador spread (…)** que nos permite separar las propiedades y valores de un **objeto.**



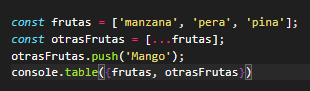
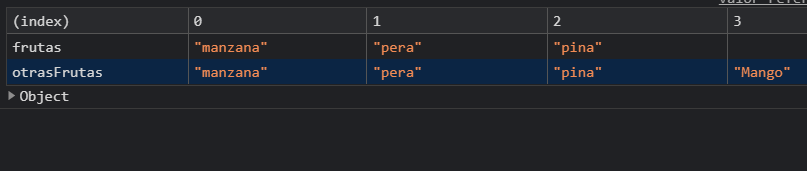
Como se aprecia en el siguiente ejemplo, al momento de llamar a la funcion estamos pasando como referencia a Peter y es ese Peter el que se está afectando dentro de la funcion y asignándose el nombre ‘Tony’.



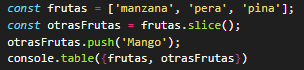
En el caso de los **Objetos** usamos dentro de la parte de parametros de la funcion el **paréntesis { }** junto con los **tres puntos (…).** En este caso no es el parámetro REST sino el operador **SPREAD**

En el caso de los **arreglos** se utiliza el operador **spread** dentro de los corchetes **[ ].**

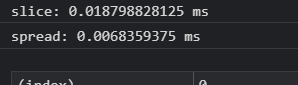
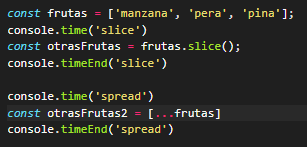
Otra forma es utilizando el método **slice** sin ningún argumento, este método corta el arreglo y manda los elementos especificados, pero sino se especifica algún argumento entonces crea un nuevo arreglo rompiendo la relación.



**Nota** el parámetro **REST** se utiliza en la parte de los parámetros de las funciones y quiere decir que una todos los argumentos en una sola variable y transfórmala en un arreglo. En cambio, cuando se lo utilizan los tres puntos (**…)**  fuera en cualquier otro lugar se denomina el operador

**SPREAD** y quiere decir separa los elementos, pero a la vez también rompe la referencia que hay en JavaScript y con esta manera poder trabajar Objetos haciendo igualaciones que apunten a diferentes espacios en memoria.

Para evaluar si el método **slice()** o el operador **spread** es más eficiente se utiliza el **console.time(‘bandera’)** para iniciar el timer y el **console.timeEnd(‘bandera’)** para finalizar el timer.



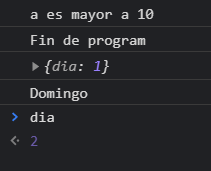
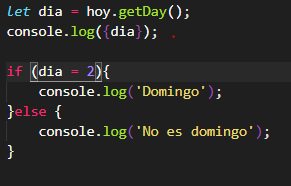
* **ESTRUCTURA DE CONTROL**

Nos permiten alterar la forma en cómo se ejecuta el código

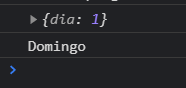
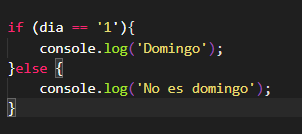
* **IF Y ELSE**

El **if** va a esperar un valor booleano, sin embargo, puede tener excepciones, por ejemplo: undefined, null, una asignación

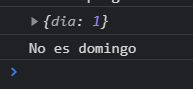
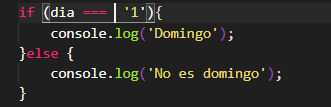
**Cuando se coloca un solo igual (=)** no esta regresando un booleano sino regresa una asignación



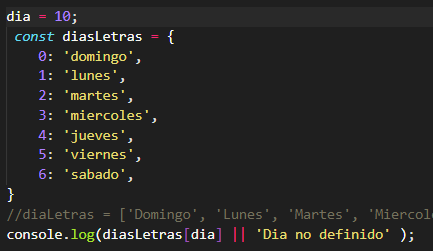
**El doble igual** está diciendo es que quiere confirmar si el día es igual al valor 2 pero no le importa el tipo.



**El triple igual** evalúa que no solo los valores internos sean los mismos, sino que también el tipado sea el mismo

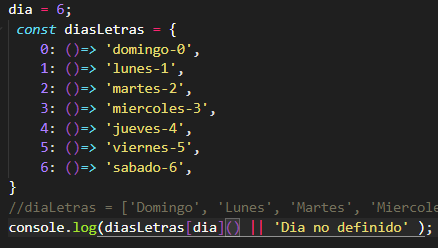


**Imprimir los días de la semana usando objetos**



En el caso de que se envíe un valor que no se encuentre en el Objeto se puede colocar el **operador or** **( || )** y poner la impresión por defecto ya que se obtendría **undefined**  en su lugar.

**Si se quiere ejecutar** una funcion se le puede también colocar dentro del objeto y utilizar los paréntesis desde donde se la llama.



**Nota:** cuando colocamos la palabra reservada **new** estamos creando una **nueva instancia** de un **objeto.**

**Date** es un objeto que nos permite obtener la afirmación del momento actual

**Nunca** usar el igual (=) al hacer condiciones.

* **UN POCO SOBRE LÓGICA BOOLEANA**
* **OPERADORES Y OPERADORES DE ASIGNACIÓN**
* **OPERADOR CONDICIONAL TERNARIO**
* **DIFERENTES USOS DEL OPERADOR CONDICIONAL TERNARIO**