**Que es JavaScript**

Es un lenguaje de programación que se utiliza principalmente para crear páginas web interactivas y dinámicas. Es un lenguaje de scripting que permite a los desarrolladores agregar funcionalidades complejas a sitios web, como actualizar contenido, mostrar animaciones y mapas interactivos. JavaScript se ejecuta en el navegador del usuario, lo que permite que las páginas web sean más interactivas y respondan a las acciones del usuario sin necesidad de recargar la página constantemente.

JS es un lenguaje interpretado que va ejecutar línea por línea de código desde la primera línea hasta la última de manera secuencial!!

1. aplicaciones web

2. crear presentaciones

3. webserves

Con node js: podemos crear servidores, hacer peticiones, comunicación en tiempo real.

Node js: nos permite correr javascript fuera de nuestro navegador web

4. videos juegos

5. aplicaciones móviles

Podemos crear aplicaciones

Ionic, react etc

Aquí hay más detalles sobre JavaScript:

* **Interacción dinámica:**

JavaScript permite que las páginas web respondan a las acciones del usuario, como clics, movimientos del mouse y entradas de formulario.

* **Manipulación del DOM:**

JavaScript puede modificar el Document Object Model (DOM) de una página web, lo que permite cambiar el contenido, la estructura y el estilo de la página.

* **Validación de formularios:**

JavaScript se puede usar para validar la información ingresada por el usuario en los formularios antes de enviarla al servidor, lo que ayuda a prevenir errores.

* **Desarrollo frontend y backend:**

Si bien se usa comúnmente para el desarrollo frontend (la parte visible de una página web), también se puede utilizar para el desarrollo backend (la parte del servidor) con tecnologías como Node.js.

* **Amplia gama de aplicaciones:**

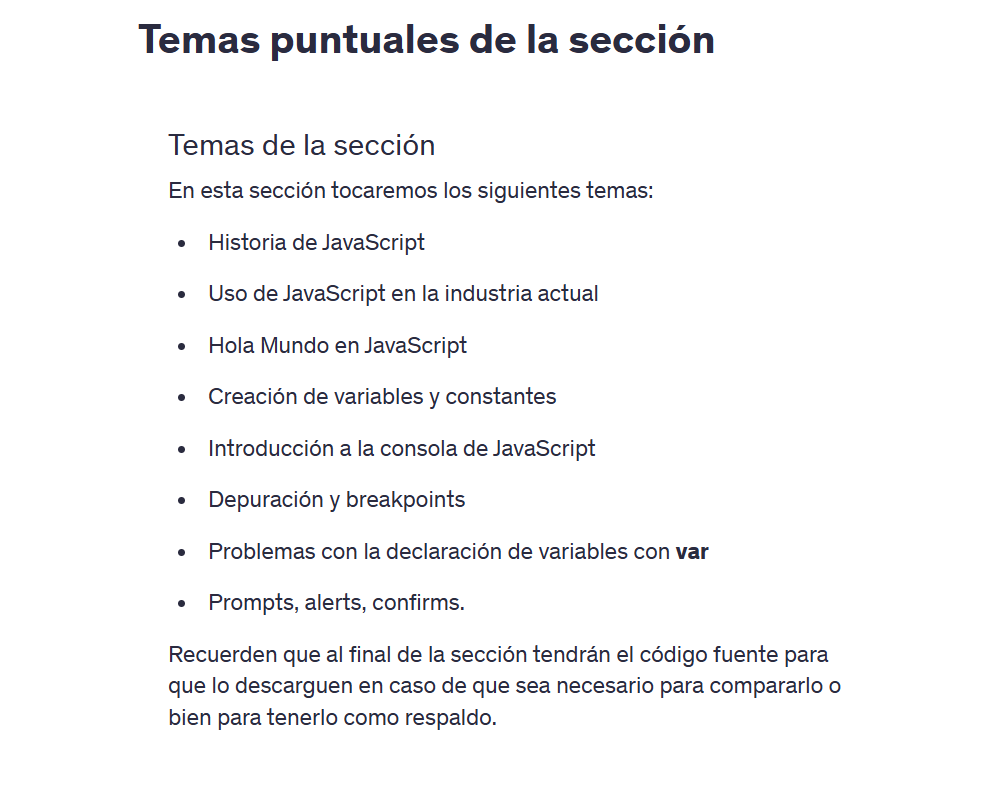
Además de las páginas web, JavaScript se utiliza en aplicaciones web, juegos, aplicaciones móviles (con marcos como React Native) y más.

**Que es un lenguaje scripting (secuencia de comandos)**

Un lenguaje de scripting es un tipo de lenguaje de programación que se utiliza para automatizar tareas y extender la funcionalidad de aplicaciones existentes. Se caracteriza por ser interpretado en tiempo de ejecución, en lugar de ser compilado, lo que permite una ejecución más rápida y flexibilidad en el desarrollo.

**framework, o marco de trabajo:** es una estructura o conjunto de herramientas y componentes que se utilizan para desarrollar aplicaciones o crear páginas web de manera más organizada, eficiente y rápida. Es como una plantilla que proporciona una base sólida y definida, permitiendo a los desarrolladores concentrarse en la lógica específica de su proyecto en lugar de tener que construir todo desde cero.

**Sección 2**



**Variable:** es un contenedor de informacion que apunta a un lugar en memoria, y puede cambiar y actualizar dicha informacion a lo largo de un programa

**Constante:** es un valor que no puede ser modificado durante la ejecución de un programa. Es decir, una vez que se le asigna un valor a una constante, este valor permanece inalterado. Esto contrasta con las variables, que pueden cambiar de valor a lo largo del programa.

**objeto global window**

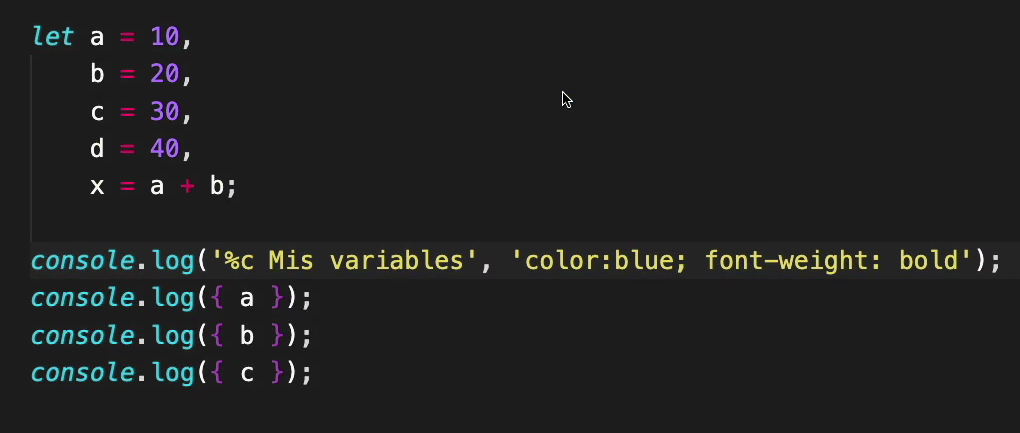
En el contexto de JavaScript en navegadores web, el objeto global window representa la ventana del navegador y actúa como el objeto raíz de todo el entorno del navegador. Es decir, todas las variables y funciones globales se convierten en propiedades o métodos de window. Esto significa que puedes acceder a ellas directamente o a través de **window.nombreDeLaVariable.**

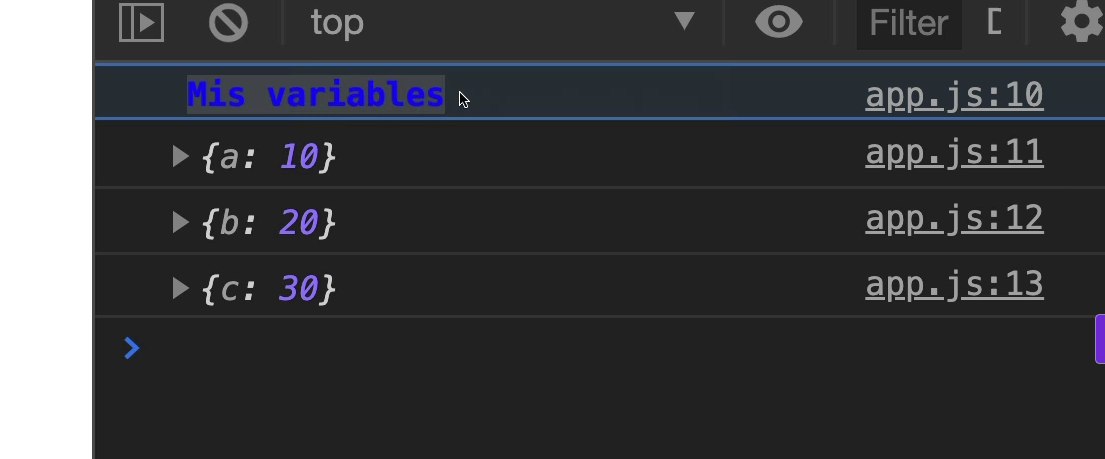
* **Objeto global:**

Un objeto global es aquel que está disponible en todo el ámbito de una aplicación, sin necesidad de ser importado o declarado específicamente.

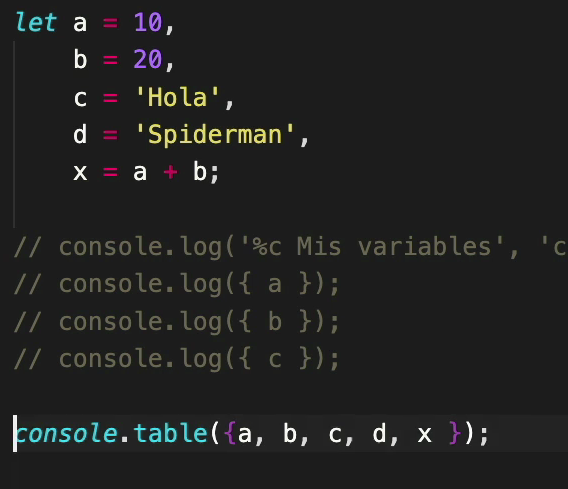
**Polyfill:** es un codigo que provee una funcionalidad de una caracteristica de JavaScript(ES6), en versiones Viejas ES5

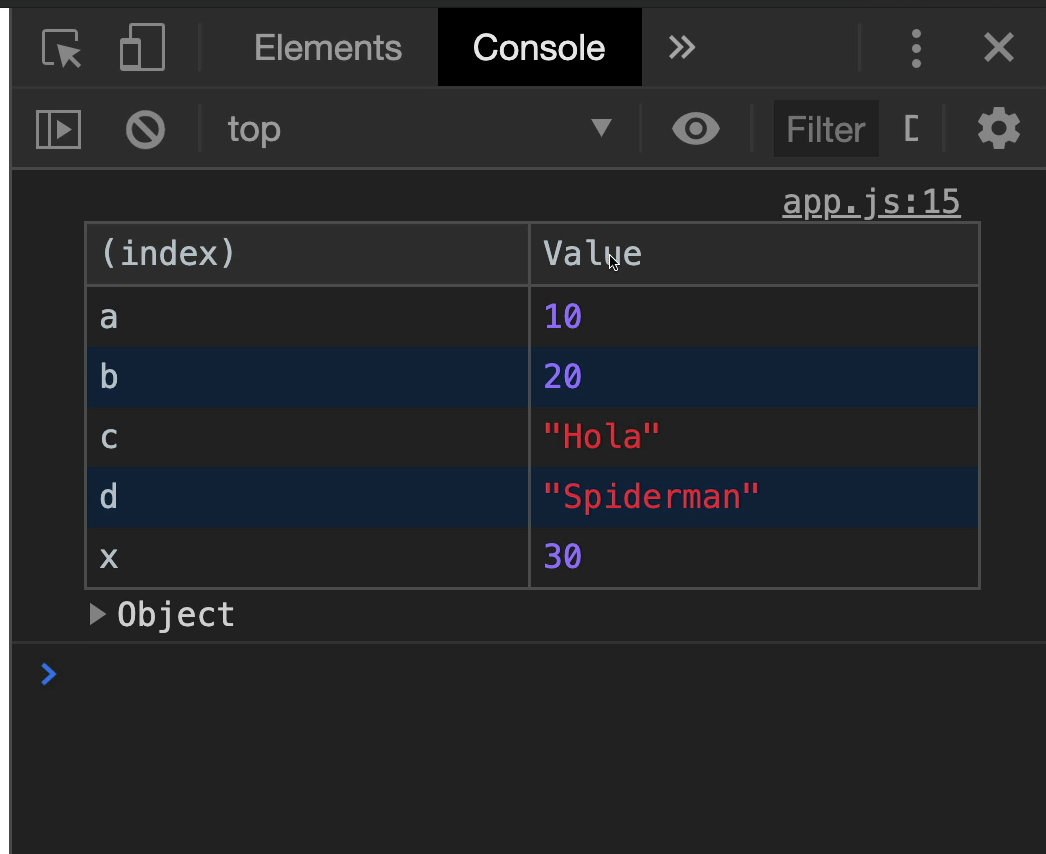
**Depuracion y breakpoint**





Console.table





Console.log(): imprime en consola el mensaje el valor asociado

Console.info():

Console.warn(): imprime en consola mensaje de alerta

Console.error(): imprime en consola mensaje de error

**typeof** es un operador unario que devuelve una cadena que indica el tipo de dato de un operando. Es útil para determinar el tipo de dato de una variable, objeto, función o expresión.

El operador **typeof()** toma un operando (una variable, valor, etc.) y devuelve una cadena que describe su tipo. Los paréntesis son opcionales.

**Ejemplos:**

* typeof 42 // "number"
* typeof "Hola" // "string"
* typeof true // "boolean"
* typeof undefined // "undefined"
* typeof null // "object"
* typeof {} // "object"
* typeof [] // "object"
* typeof function() {} // "function"
* typeof (function() {}) // "function"

**Tipos de datos devueltos por typeof:**

* "undefined": Si el operando no está definido o no tiene valor.
* "object": Si el operando es un objeto, un array o null.
* "boolean": Si el operando es un valor booleano (true o false).
* "number": Si el operando es un número.
* "string": Si el operando es una cadena de texto.
* "function": Si el operando es una función.
* "symbol": Si el operando es un símbolo (introducido en ES6).
* "bigint": Si el operando es un número entero grande (introducido en ES2020).

Consideraciones:

* typeof null devuelve "object" debido a un error histórico en JavaScript.
* typeof NaN devuelve "number".
* typeof new Date() devuelve "object"

**El operador typeof es útil para:**

* Validar tipos de datos en funciones o lógica.
* Realizar comprobaciones antes de ejecutar ciertas operaciones.

Depurar código, especialmente en JavaScript que es un lenguaje de tipado dinámico.

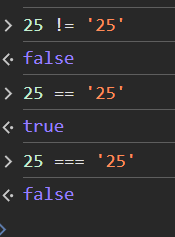
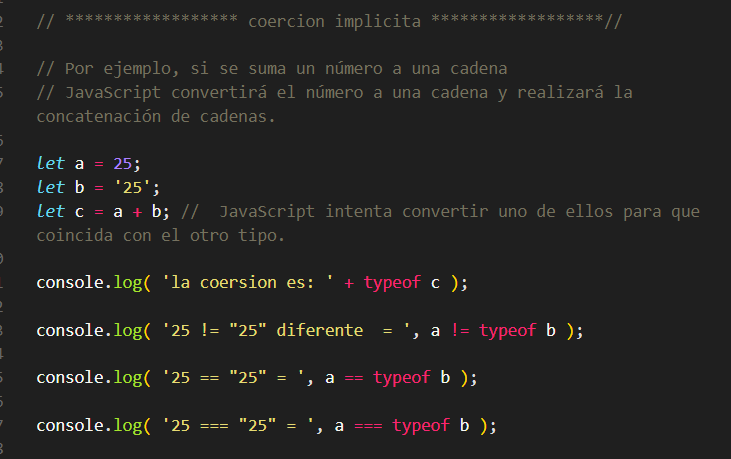
**La coerción**

En JavaScript, la coerción de tipos se refiere a la conversión automática de valores de un tipo de dato a otro. Esto sucede cuando se utilizan diferentes tipos de datos en una operación y JavaScript intenta convertir uno de ellos para que coincida con el otro tipo. La coerción puede ser implícita (automática) o explícita (realizada por el desarrollador).

**Coerción implícita:**

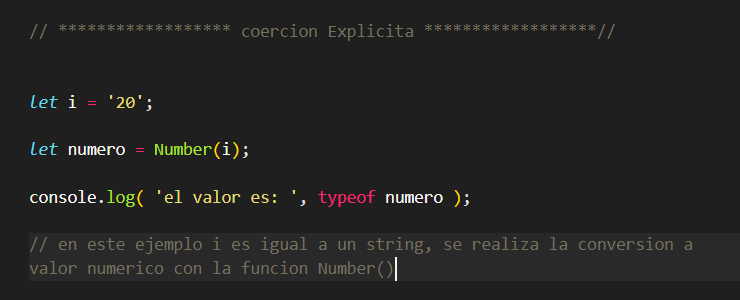
* Se produce automáticamente por JavaScript en ciertas situaciones, como cuando se utilizan operadores como **+, ==,** etc. con diferentes tipos de datos.

Por ejemplo, si se suma un número a una cadena, JavaScript convertirá el número a una cadena y realizará la concatenación de cadenas.

****

**Coerción explícita:**

La coerción explícita se realiza intencionalmente por el desarrollador, **utilizando funciones como Number(), String(), Boolean(), etc.** Por ejemplo:



**Ejemplos comunes de coerción implícita:**

* Suma con el operador +:

Si se utiliza el operador + con una cadena y un número, se produce la concatenación de cadenas.

* Comparación con ==:

El operador == realiza la coerción de tipos antes de comparar, mientras que === realiza una comparación estricta sin coerción.

* **Conversión a booleano:**

En contextos booleanos (como en condicionales), los valores se convierten a booleanos.

* **Coerción implícita:**
  + "5" + 9 -> "59" (el número 9 se convierte a cadena)
  + 5 == "5" -> true (el string "5" se convierte a número)
* **Coerción explícita:**
  + Number("5") + 9 -> 14 (la cadena "5" se convierte a número)
  + String(10) + 20 -> "1020" (el número 10 se convierte a cadena)

**En JavaScript, los símbolos =, == y === representan diferentes tipos de operaciones:**

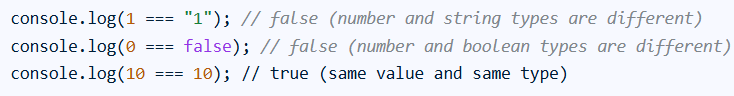
**= (Operador de asignación**): Este operador asigna un valor a una variable.

****

**== (Operador de igualdad flexible):** Este operador compara dos valores para determinar su igualdad, aplicando la conversión de tipos si es necesario. Esto significa que, si los operandos son de tipos diferentes, JavaScript intenta convertir uno o ambos operandos a un tipo común antes de realizar la comparación.

****

**=== (Operador de Igualdad Estricta):** Este operador compara dos valores para determinar su igualdad sin realizar conversión de tipos. Para que la comparación sea verdadera, tanto el valor como el tipo de dato de los operandos deben ser idénticos.



**Recomendación:** Generalmente, se recomienda usar el operador === (igualdad estricta) en JavaScript siempre que sea posible para evitar comportamientos inesperados que pueden surgir de la coerción de tipos con el operador ==. El uso de igualdad estricta promueve un código más predecible y robusto.

**Orden y lugar de las importaciones**

Que es una importación:

En programación, las importaciones se refieren al proceso de utilizar código o funcionalidades que están definidas en otros archivos, módulos o bibliotecas dentro de un programa. Esto permite la reutilización de código, la organización modular de proyectos y la extensión de las capacidades del lenguaje mediante el uso de bibliotecas externas.

**orden**

Es importante a la hora de hacer una importación colocar la etiqueta <script> dentro del body en la parte final, para que así se cargue bien los datos del HTML antes que JS y así evitar cargas innecesarias.

**Principales problemas declaración de variables con VAR**

ES QUE PERMITE REEMPLAZAR PROPIEDADES PROPIAS DEL OBJETO GLOBAL WINDOWS, sobrescribiéndolas si se usan los mismos nombres de esas propiedades creando conflictos.

**En cambio, let y const no sobre escribe las propiedades del objeto global Windows y son compatibles con los navegadores modernos.**

Otro problema que tiene VAR es que si se define de ultimo y primero se llama la variable creada, nos creara un mensaje UNDERFINE(indefinido) pudiendo general conflictos la hora de resolver la ejecución de una función o tarea asignada a esa variable.

**Nota:** utilidad ¡se puede usar para crear compatibilidad con navegadores antiguos!

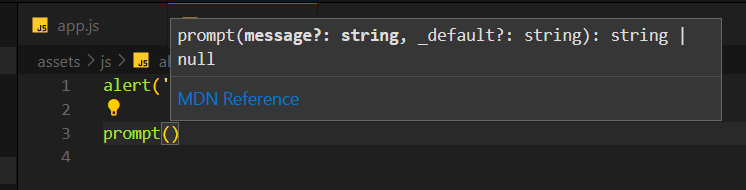
**Alerts “alertas”**

Es un mensaje de alerta en la pantalla, y es bloqueante es decir hasta que el usuario no le de click no sigue la siguiente tarea

**NOTA:** TODO LO QUE ESTA EN PARENTESIS SON ARGUNMENTOS O PARAMETROS DE UNA FUNCION

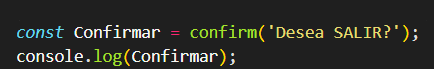


**Prompt() “indicaciones”**

En JavaScript, un prompt es una función integrada que muestra un cuadro de diálogo modal en el navegador, solicitando al usuario que ingrese datos. Este cuadro de diálogo incluye un mensaje opcional y un campo de entrada de texto, donde el usuario puede escribir su respuesta. El resultado del prompt, ya sea el texto ingresado por el usuario o null si se cancela, se devuelve como una cadena. 

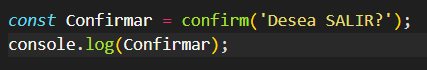
**Confirm() “conformaciones”**

La función confirm() muestra una ventana modal con una pregunta y dos botones: OK y CANCELAR. El resultado es true si se pulsa OK y false en caso contrario.

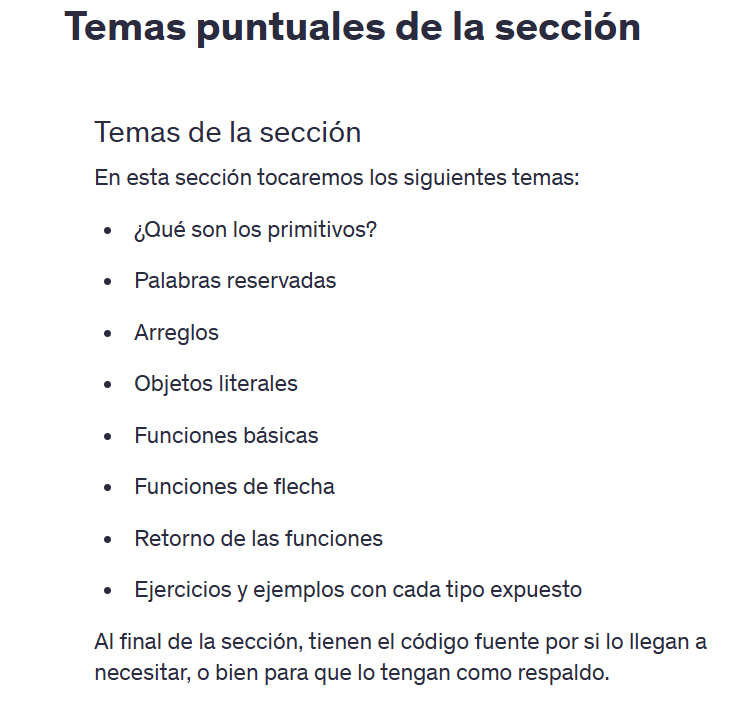


**Notas:**

* **Alerts(), Prompt() y confirm()** son funciones del objeto Windows
* **EL METODO ES UNA FUNCION DENTRO DE UN OBJETO**
* **Con llaves {} llamo objetos**
* **Con corchetes [] llamo a arreglos**
* Se puede recibir un string vacio, un valor o un null
* Null y underfiner no son lo mismo



**Sección 3**

****

**JavaScript:** es un lenguaje débilmente tipado, esto quiere decir que el programador no le dice que contiene la variable, JS lo va inferir, dependiendo del valor que se le asigne a la variable.

**Nota:** en JV todos los datos son objetos exceptuando los primitivos.

**Camelcase:** se acostumbra a trabajar en JV a trabajar declarando variable la primera letra minúscula y las siguientes Palabras empiezan en Mayúscula.

**Tipo de datos de una Variable**

Describe el contenido del valor de que tiene una variable

**En JavaScript, existen dos tipos principales de datos:**

* 1. **Valores Primitivos:** Es una Información que no es un objeto y son inmutables. Los tipos de datos primitivos son:
* **Boolean:** true / false :: Verdadero y Falso.
* **Null:** Sin valor en lo Absoluto.
* **Undefined:** una variable declarada que aún no se le asigna Valor.
* **Number:** integers, floats,etc.
* **String:** una cadena de caracteres, ejemplo: Palabras, nombres etc.
* **Symbol:** Es un valor único que no es igual a ningún otro Valor.
* **BigInt:** es un tipo de dato primitivo que permite representar números enteros con precisión arbitraria, más allá del rango manejable por el tipo Number. Esto significa que puedes trabajar con números enteros muy grandes, que antes no eran posibles en JavaScript debido a las limitaciones del tipo Number.

No se pueden mezclar BigInt y Number en operaciones matemáticas. Para realizar operaciones, es necesario convertir ambos operandos al mismo tipo.

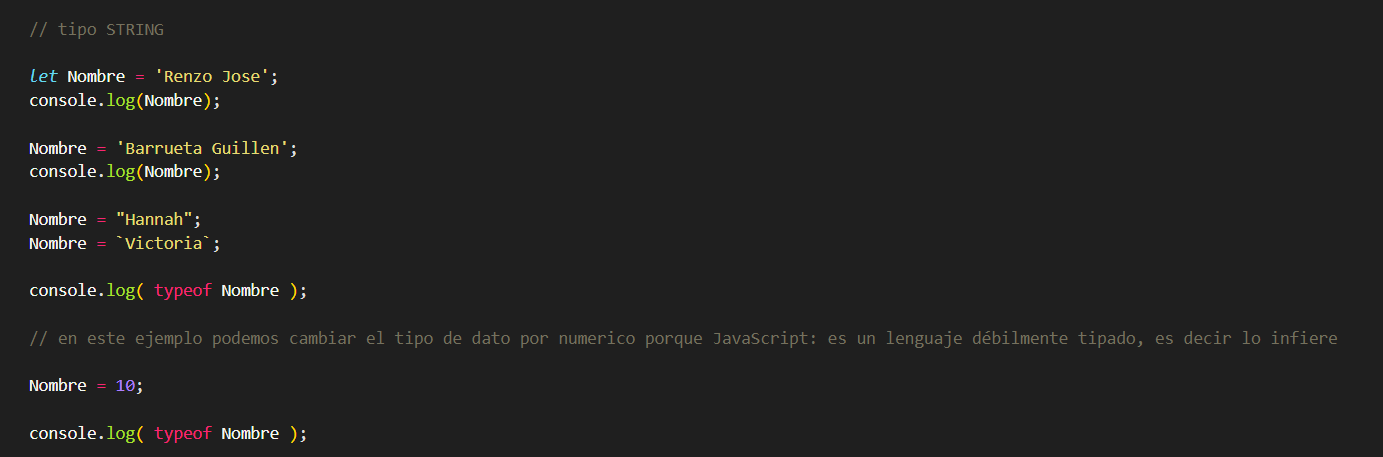
* 1. **Valores Complejos (u Objetos):**

Estos son datos mutables y se asignan por referencia. Los tipos complejos son:

* + **object** (incluyendo arreglos, fechas, etc.)
  + **function**

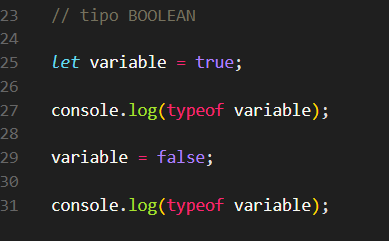
**Ejemplos de string:**

Los string se pueden representar con comillas dobles “ ”, comillas ‘ ’, y comillas simples invertidas back tips``

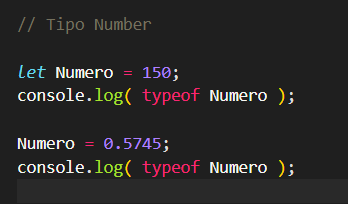


**Typeof:** es un operador que devuelve una cadena que indica el tipo de dato de un operador. Es una herramienta útil para determinar el tipo de una variable o expresión en tiempo de ejecución.

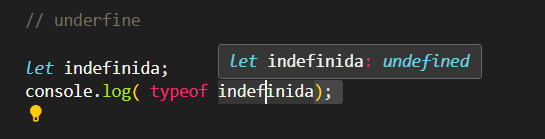
**Ejemplos de boolean:**

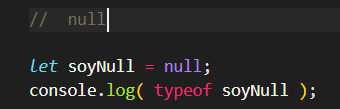
****

**Ejemplo tipo Number**

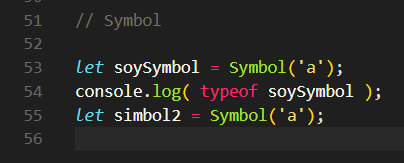


**Ejemplo Tipo undefined**



**Ejemplo de tipo Null**  


**Ejemplo tipo Symbol**



**Palabras reservadas y nombre de variables**

Son las palabras que utiliza el lenguaje de programación y se no se deben usar para hacer declaraciones de variables, constantes, métodos, funciones etc

Se pueden verificar en los siguientes enlaces:

<https://mothereff.in/js-variables#%E0%B2%A0%5f%E0%B2%A0>

algunos caracteres araros que se utilizan pero no es recomendable:

<https://mathiasbynens.be/notes/javascript-identifiers>

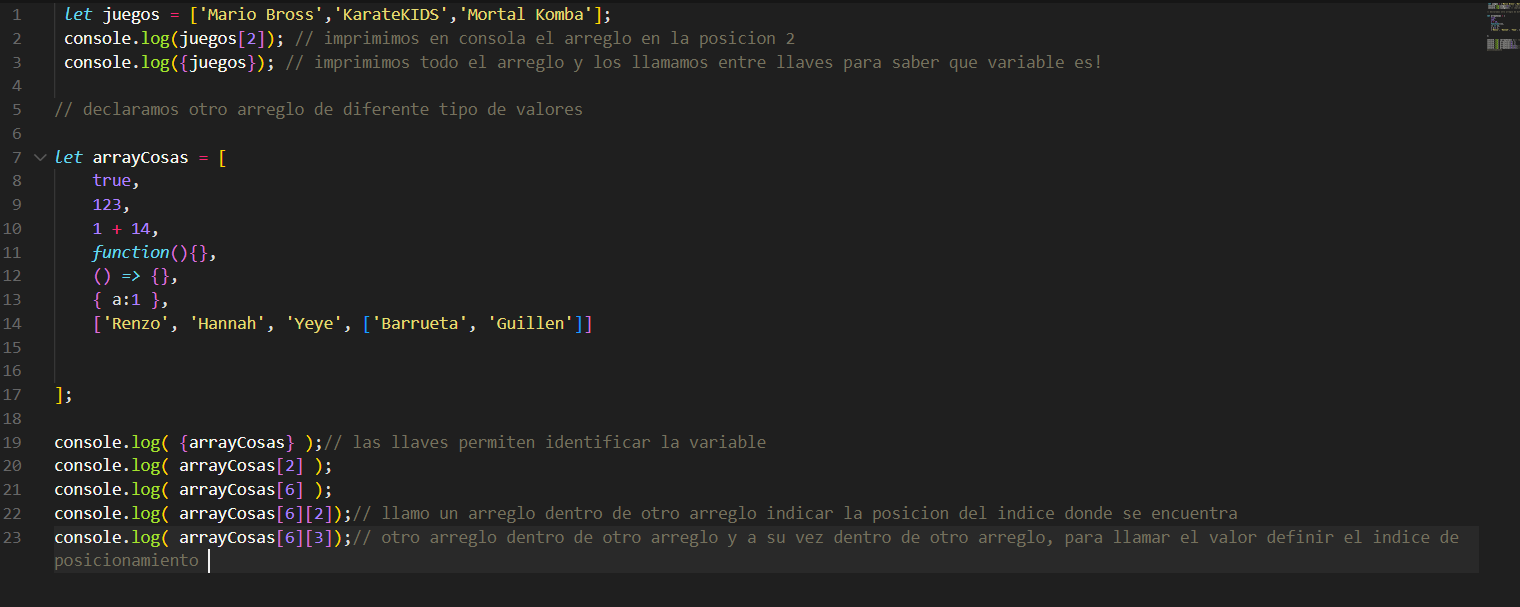
y otra pagina donde si es buena práctica utilizar

<https://underscorejs.org/>



**Arreglos**

Son un objeto muy parecido a una lista de información, que contiene un grupo de elementos

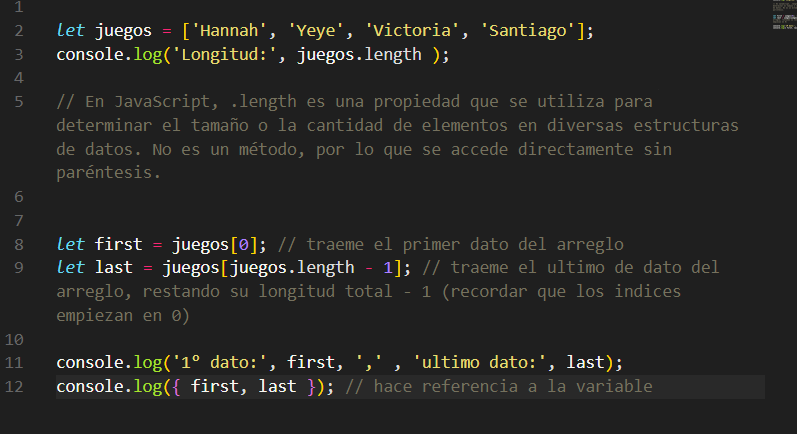


**Detalles de los arreglos**

**Propiedad length:** es una propiedad que se utiliza para determinar el tamaño o la cantidad de elementos en diversas estructuras de datos. No es un método, por lo que se accede directamente sin paréntesis.

**Nota:** para traer el último dato de un arreglo dinámico, podemos usar la propiedad **.length** de esa manera accedemos al total de datos que existen y le restamos uno.

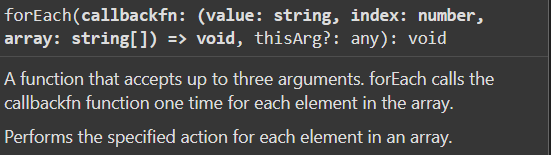
Ejemplo:



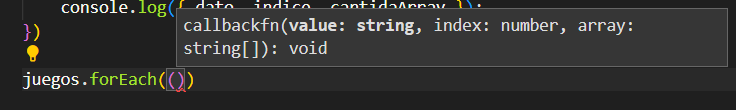
**Metodo forEach:** es un método de los arreglos (arrays) que permite ejecutar una función dada una vez por cada elemento del arreglo. Es una forma de iterar sobre los elementos de un arreglo sin la necesidad de usar un bucle **for** tradicional. Es útil para realizar operaciones o acciones sobre cada elemento del arreglo, como imprimir valores, realizar cálculos o modificar los elementos.

Cabe destacar, para barrer todos los elementos de los arreglos podemos usar un método (ejecuta una instrucción por cada uno de los elementos que se encuentran en el arreglo)

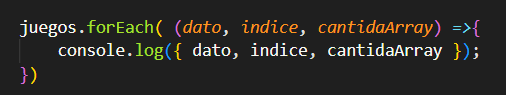
Ejemplo:



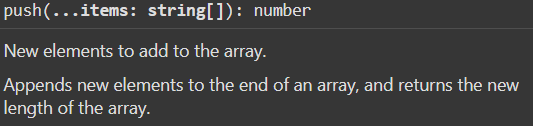
**La sintaxis es la siguiente:** (el primer valor, dato interno) podemos asignarle un nombre en este caso le colocamos **dato,** el segundo valor es el índice donde se encuentra el dato podemos asignarle un nombre en este caso le colocamos **indice,**y el tercer valor muestra la cantidad de nuestro arreglopodemos asignarle un nombre en este caso le colocamos **cantidadArray**

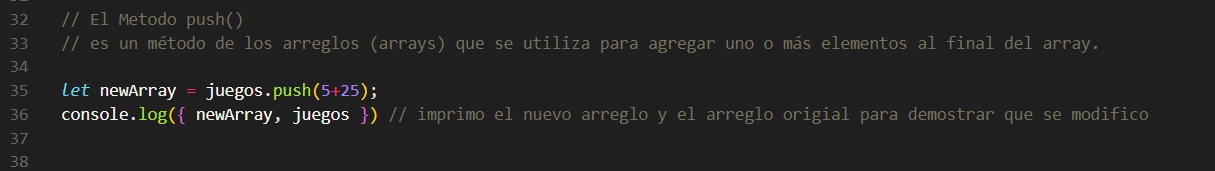


Pasará por cada uno de los elementos que componen nuestro arreglo y los imprimimos en pantalla con llaves para saber a qué variable se refiere de la sintaxis.

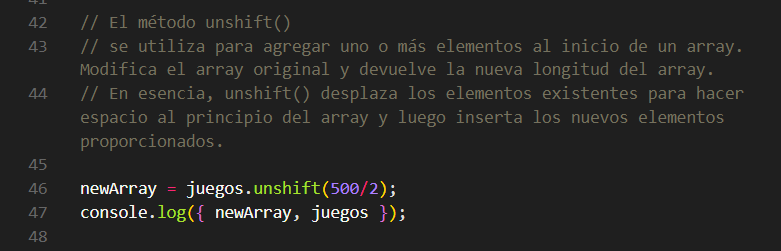


**Metodo push():** es un método de los arreglos (arrays) que se utiliza para agregar uno o más elementos al final del array.





**El método unshift()** en JavaScript se utiliza para agregar uno o más elementos al principio de un array. Modifica el array original y devuelve la nueva longitud del array. En esencia, es como el método push(), pero en lugar de agregar al final, agrega al inicio.

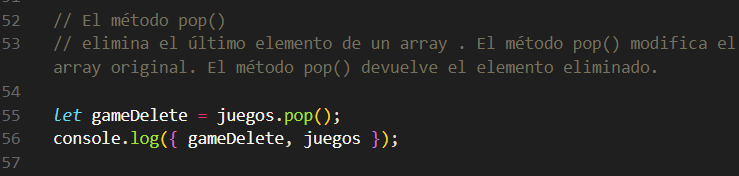


**Método pop():** se utiliza para eliminar el último elemento de un array y devolverlo. Este método modifica el array original, reduciendo su longitud en uno.

 Si el array está vacío, pop() devuelve undefined.

En resumen, **pop()** es un método de mutación que sirve para:

1. **Eliminar el último elemento:** Remueve el último valor agregado al array.
2. **Devolver el elemento eliminado:** El valor que se elimina se retorna como resultado de la llamada al método.
3. **Modificar el array original:** El array se altera directamente, perdiendo su último elemento.



**El método splice():**se utiliza para modificar el contenido de un array eliminando, reemplazando o agregando elementos.

 Es un método mutable, lo que significa que altera directamente el array original y devuelve un nuevo array con los elementos eliminados, si los hay.

¿Para qué sirve?

* **Eliminar elementos:**

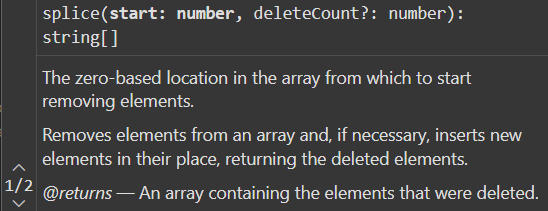
Se puede eliminar uno o más elementos de un array especificando el índice inicial y la cantidad de elementos a eliminar.

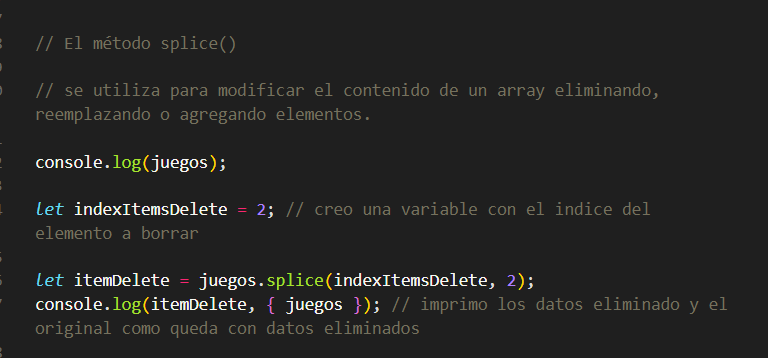
* **Reemplazar elementos:**

Se pueden eliminar elementos existentes y reemplazarlos con nuevos elementos en la misma posición.

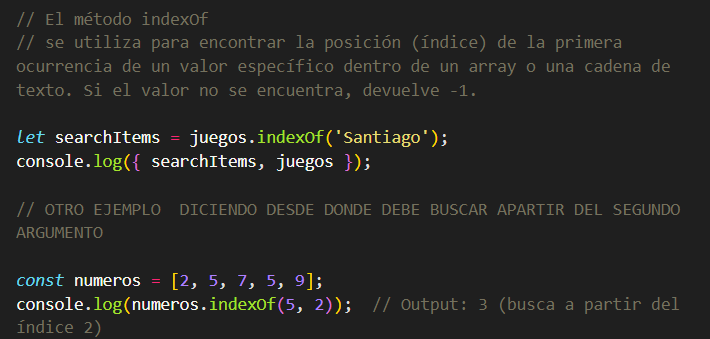
* **Agregar elementos:**

Se pueden insertar nuevos elementos en cualquier posición del array sin eliminar elementos existentes.





**El método indexOf():** se utiliza para encontrar la posición (índice) de la primera ocurrencia de un valor específico dentro de un array o una cadena de texto. Si el valor no se encuentra, devuelve -1.



**Objetos literales**

Un objeto literal es una forma concisa de crear objetos directamente en el código, utilizando pares clave-valor dentro de llaves {}. Estos objetos son útiles para representar datos estructurados de manera organizada y son una alternativa más eficiente a la creación de objetos utilizando el constructor new Object().

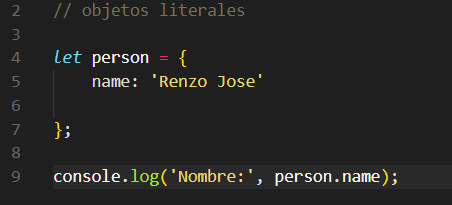
**¿Cómo se crean objetos literales?**

Los objetos literales se crean utilizando llaves {}. Dentro de las llaves, se definen los pares clave-valor separados por comas. La clave se escribe seguida de **dos puntos :** y luego el valor asociado.

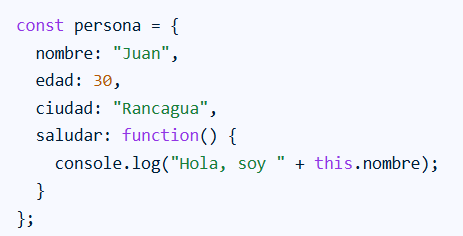
**Nota:** la clave hace referencia al nombre dado al valor, que a la final se convertirá en un índice del objeto literal

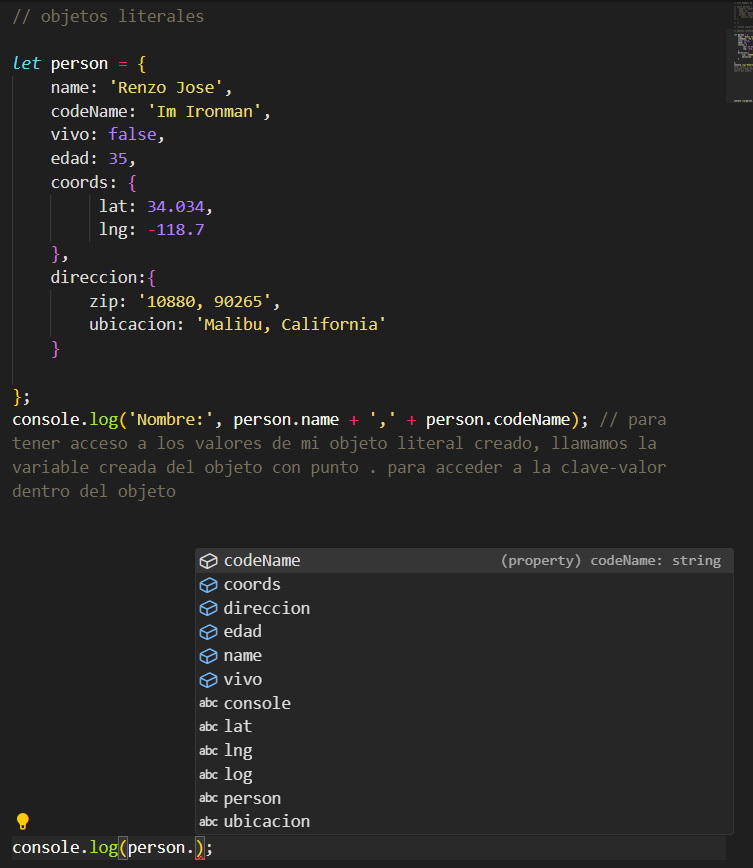
**Otra definición:** son objetos que tienen pares de valores (nombre: valor), van a tener una propiedad creada con el valor dentro de la llave estructurada.

Ejemplo:



**Nota:** cuando se visualizan **llaves {}** se asocia a un objeto y cuando se visualiza **corchetes []** es un arreglo.



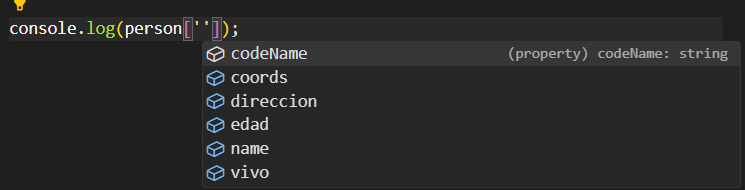
****

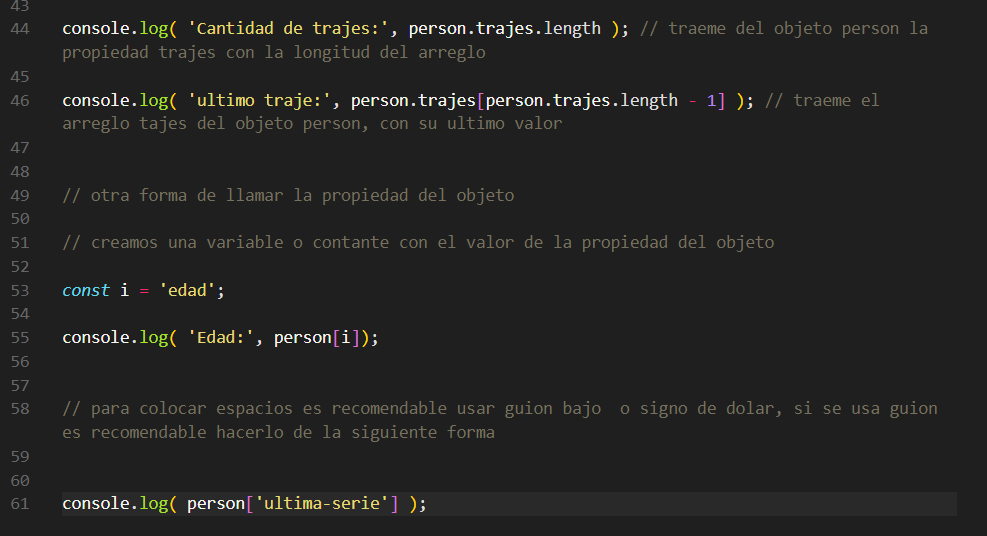
Para acceder a los datos de objeto utilizamos la anotación del punto **.**

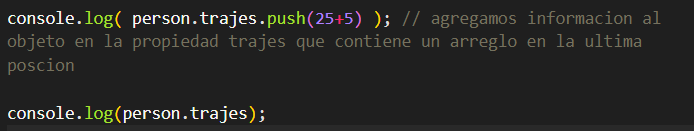
**Nota:** cabe destacar que JV los ordena por orden alfabético los datos creados (clave-valor) dentro del objeto literal

Otra forma de acceder a los datos del objeto es con corchetes con la siguiente sintaxis









**Mas detalles de los objetos literales**

[**https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global\_Objects/Object**](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Object)

**FUNCIONES**

En JavaScript, una función es un bloque de código reutilizable diseñado para realizar una tarea específica.

Las funciones pueden tomar entradas (argumentos), ejecutar instrucciones, y devolver un valor. Son fundamentales para organizar y modularizar el código, permitiendo la reutilización

**Todas las funciones tradicionales que son las que tienen la palabra explicita de Function**, internamente tienen un objeto implícito que son las argument (argumentaciones), esto lo que hace referencia es a todos los elementos enviados a la función.

El objetivo es centralizar la lógica de un procedimiento que podemos utilizar varias veces.

**Nota: Método Es una función que se encuentra dentro de un objeto**

**los parámetros de una función:** son variables que se definen en la declaración de la función y que reciben valores cuando la función es llamada. Estos valores, conocidos como argumentos, se pasan a la función a través de la llamada y se utilizan dentro del cuerpo de la función.

**En resumen:**

* **Parámetros:**Son los nombres de las variables que se listan entre los paréntesis de la declaración de una función.
* **Argumentos:**Son los valores reales que se pasan a la función cuando se llama.

**Argumentos:**

* Las funciones pueden recibir cero o más argumentos.
* Los argumentos se especifican entre paréntesis, separados por comas.
* Si una función recibe menos argumentos de los declarados, los faltantes serán undefined.
* Si una función recibe más argumentos de los declarados, los argumentos sobrantes serán ignorados**.**

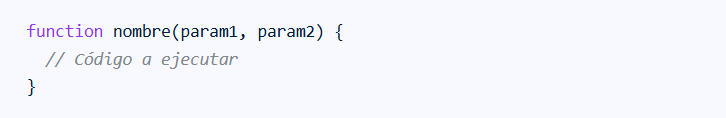
**Llamada a funciones:**

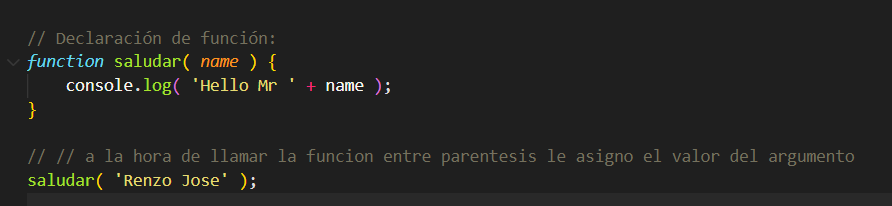
* Para ejecutar una función, se la llama por su nombre seguido de paréntesis, pasando los argumentos si los requiere.

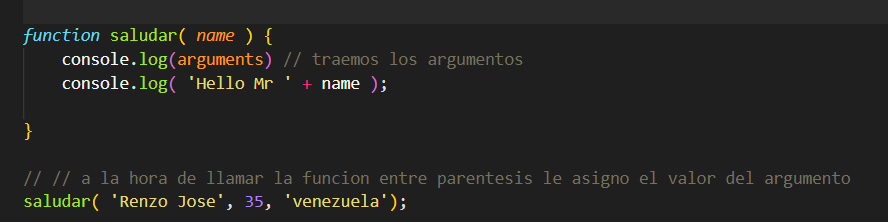
**Ejemplo:  miFuncion(argumento1, argumento2);**

**Formas de crear funciones:**

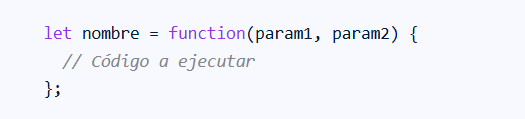
* 1. **Función tradicional**

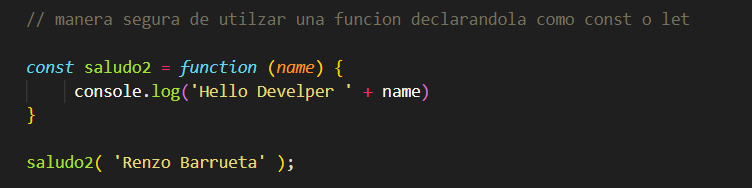
****

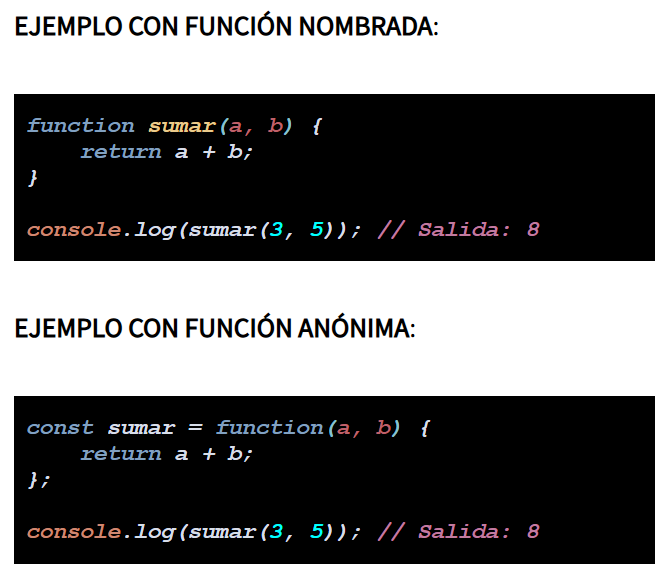
****

****

* 1. **Expresión de un Función**

****

****

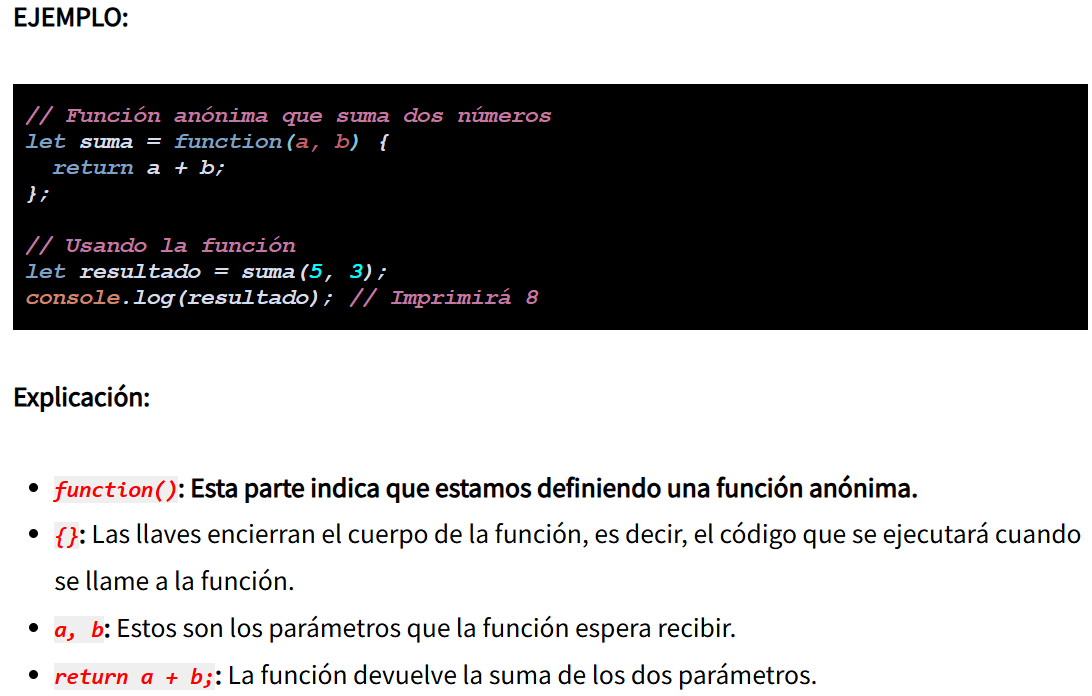
****

**Diferencias clave:**

* **En funciones nombradas, el nombre (*sumar*) está directamente asociado al bloque de la función.**
* **En funciones anónimas, el nombre no está asociado al bloque de código. Sin embargo, puede ser invocada a través de la variable *sumar*. *Básicamente, es una función que se define directamente donde se va a usar, sin necesidad de darle un nombre.***

Las funciones anónimas en JavaScript **son aquellas que no tienen un nombre explícito asociado a ellas.** A diferencia de las [**funciones declarativas**](https://www.apinem.com/que-son-y-para-que-sirven-las-funciones-en-javascript/) (o nombradas), donde se asigna un nombre a la función para poder llamarla posteriormente, las funciones anónimas se declaran directamente dentro de una expresión.

Estas funciones **son usadas comúnmente en situaciones donde no se necesita reutilizar la función en otro lugar del código, como en callbacks o expresiones que serán ejecutadas en un contexto limitado.**

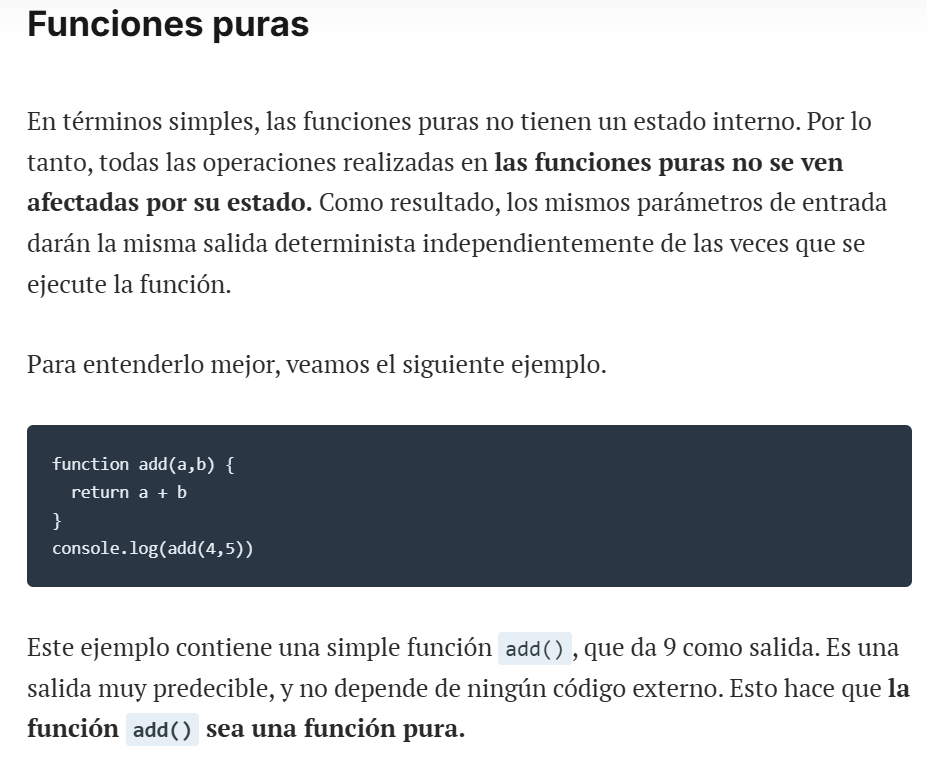


**OTRO DATO A DESTACAR DE LAS FUNCIONES PUEDEN SER PURAS E IMPURAS:**

En JavaScript, una función pura es aquella que, dado los mismos argumentos de entrada, siempre devuelve el mismo resultado y no tiene efectos secundarios. Esto significa que su comportamiento solo depende de sus parámetros de entrada y no interactúa con ningún estado externo ni modifica datos fuera de su ámbito.

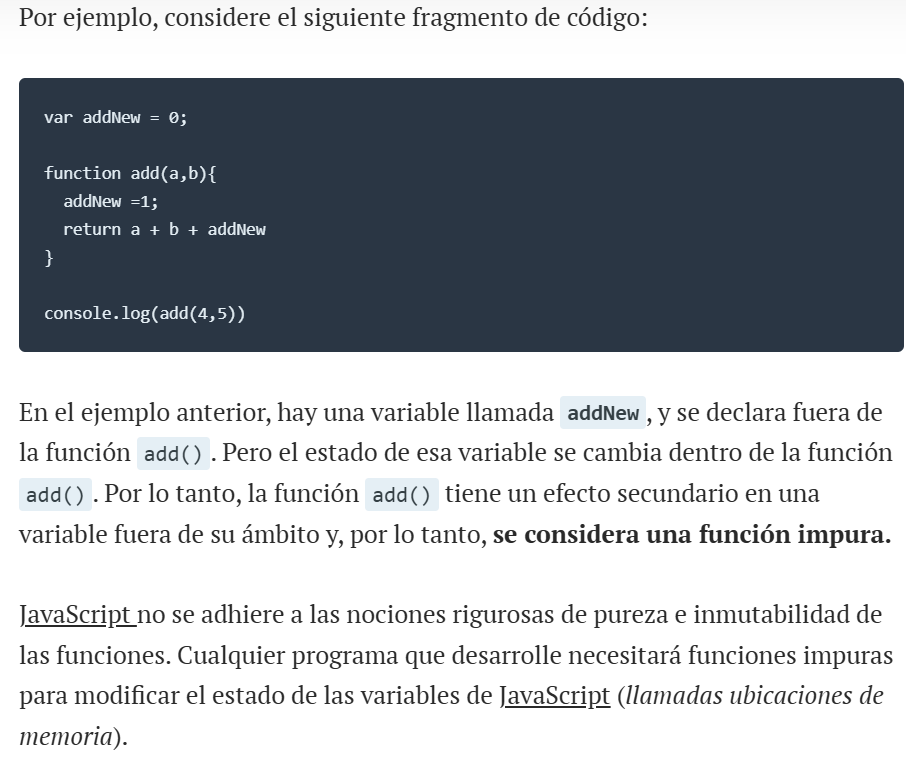
**Características de una función pura:**

* **Determinista:** Para las mismas entradas, siempre produce la misma salida.
* **Sin efectos secundarios:** No modifica variables globales, parámetros de otras funciones ni realiza operaciones de entrada/salida (como lecturas o escrituras en archivos, peticiones a la red, etc.)



**Funciones impuras**

Una función impura es una función que **contiene uno o más efectos secundarios.** Muta los datos fuera de su ámbito léxico y *no produce* previsiblemente la misma salida para la misma entrada.



En general, lo ideal es mantener los elementos impuros de sus programas separados del procesamiento de datos, que suele ser puro. Además, la actualización y el mantenimiento de tus aplicaciones será mucho más fácil si confinas los elementos impuros a sus funciones particulares.

**Math.random()** es una función impura ya que modifica el estado interno del objeto Math y **proporciona resultados diferentes con cada llamada**. Por lo tanto, Math.random() puede contener efectos secundarios.

* 1. **Función flecha (arrow function)**

En JavaScript, una función lambda, también conocida como función de flecha, es una forma concisa de definir funciones anónimas. Se utilizan principalmente para simplificar la sintaxis de funciones pequeñas y para pasarlas como argumentos a otras funciones de orden superior.

**¿Para qué sirve?**

* **Sintaxis más corta:**

Las funciones lambda ofrecen una forma más breve de escribir funciones, especialmente cuando son funciones de una sola línea.

* **Funciones como argumentos:**

Son ideales para pasarlas como argumentos a funciones de orden superior como map, filter, reduce, entre otras.

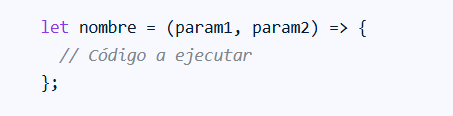
* **Contexto léxico:**

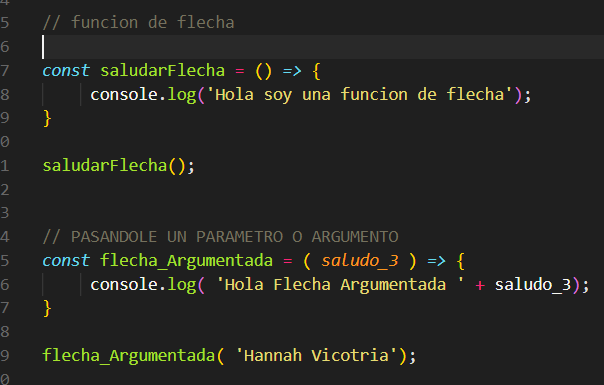
**this** dentro de una función lambda se refiere al contexto léxico **(scope)** donde fue definida la función, lo que puede ser útil para evitar problemas con el **this** en funciones tradicionales.

* **Legibilidad:**

En muchos casos, pueden hacer que el código sea más legible y fácil de entender.

Ejemplo: sintaxis



****

[**https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions**](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions)

**RETURN DE UNA FUNCION**

La sentencia return finaliza la ejecución de la función y especifica un valor para ser devuelto a quien llama a la función.

la sentencia return dentro de una función sirve para dos propósitos principales:

* finalizar la ejecución de la función
* devolver un valor al código que la llamó.

**¿Cómo funciona return?**

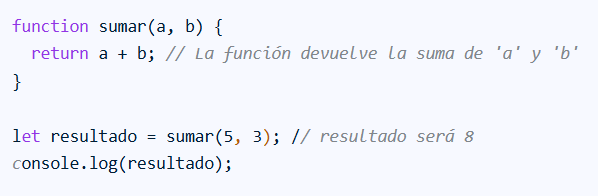
1. **Finalización de la función:**

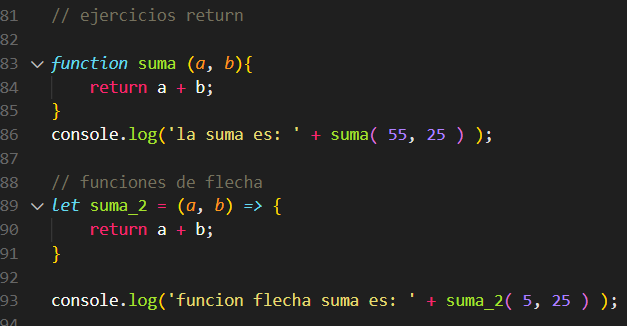
return detiene la ejecución de la función en el punto donde se encuentra. Cualquier código que siga a return dentro de la función no se ejecutará.

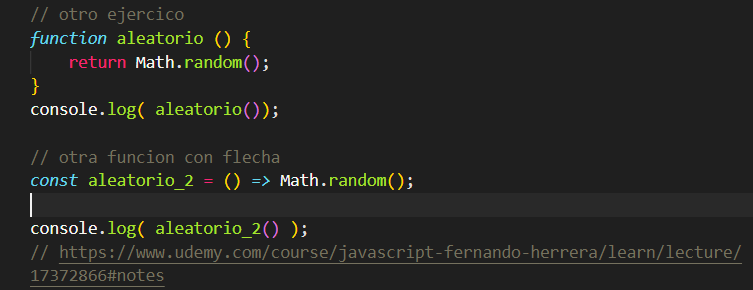
1. **Devolución de un valor:**

return puede ir seguido de una expresión (un valor, una variable, un cálculo, etc.) que se evalúa y se devuelve como el resultado de la función. Si no se especifica ninguna expresión, la función devolverá undefined por defecto.

**Ejemplo:**



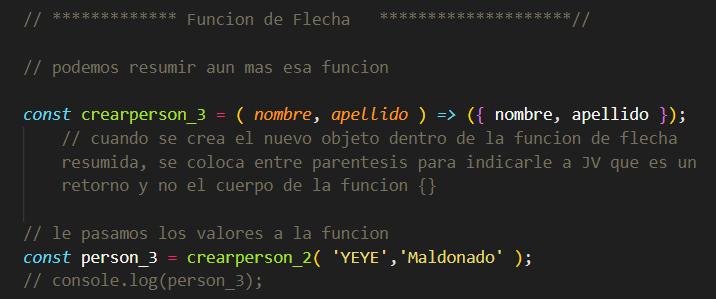




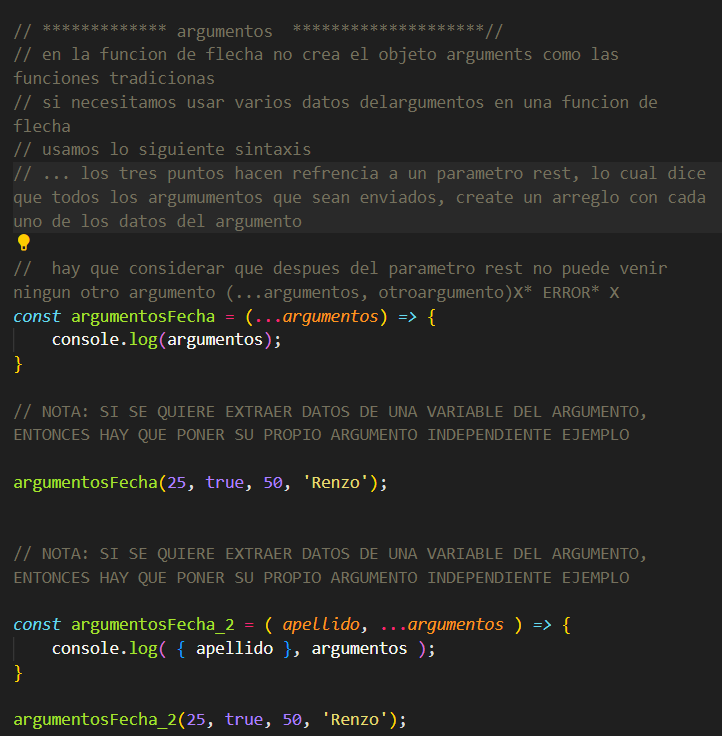
**Protips funciones**

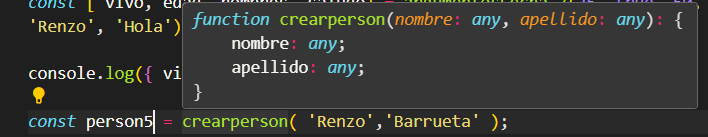
Cuando el nombre de la propiedad es el mismo al nombre del valor de esa propiedad(clave-valor), no hace falta que escriba el valor de esa propiedad simple se escribe el nombre de la propiedad (esto es posible desde ECMASCRIPT6)

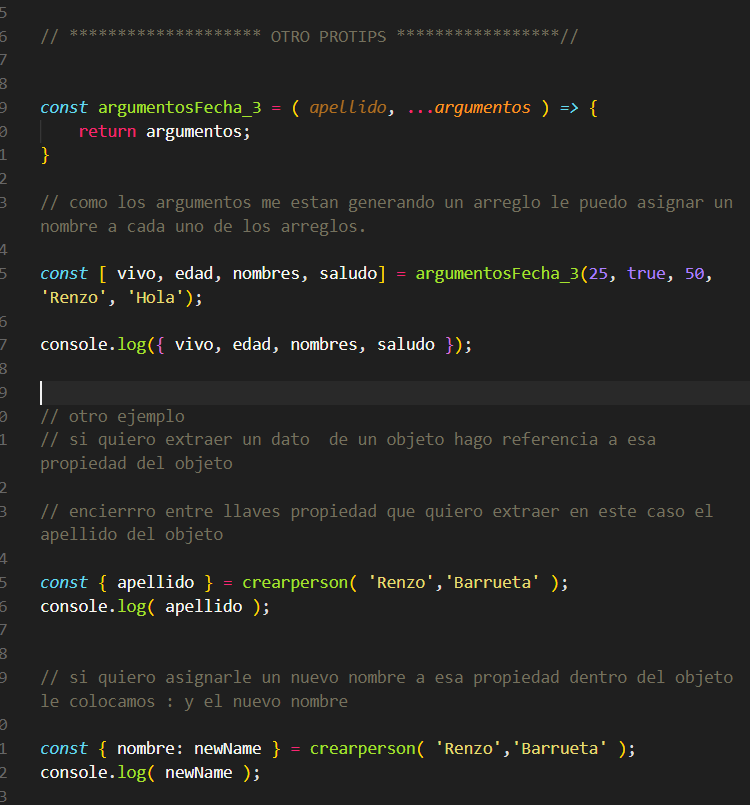




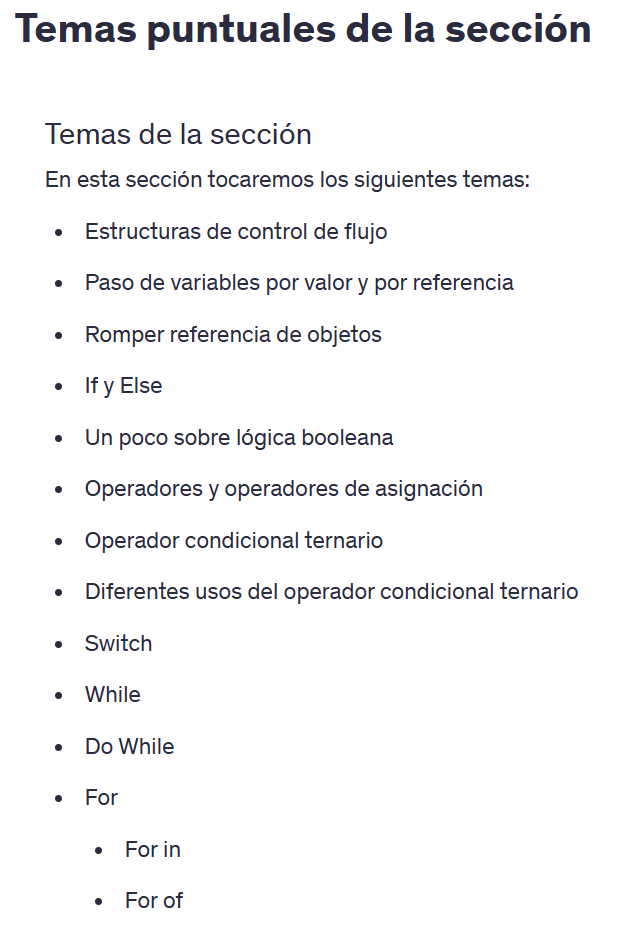
En JavaScript, el operador ... puede actuar como operador spread o como operador rest, dependiendo del contexto. El operador spread se utiliza para expandir elementos de un array u objeto, mientras que el operador rest se usa para agrupar múltiples elementos en un solo array.









**Sección 4**

**Estructura de control de flujo**

En JavaScript, una estructura de control es una construcción que permite al programador dirigir el flujo de ejecución de un programa. Estas estructuras permiten tomar decisiones basadas en condiciones, repetir bloques de código y alterar el orden normal de ejecución del código.

**Tipos de estructuras de control en JavaScript:**

* **Estructuras condicionales:**

Permiten ejecutar diferentes bloques de código dependiendo de si una condición es verdadera o falsa. Incluyen if, else if, else y switch.

* **Bucles (loops):**

Permiten repetir un bloque de código varias veces. Ejemplos son for, while y do...while.

* **Estructuras de control de flujo:**

Permiten modificar el orden normal de ejecución del código, como break para salir de un bucle y continue para saltar a la siguiente iteración.

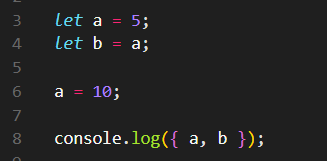
En resumen, las estructuras de control son fundamentales para la lógica de cualquier programa, permitiendo que el código tome decisiones y realice acciones basadas en condiciones específicas y repeticiones.

**Paso de variable por valor o por referencia**

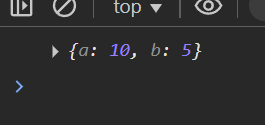
Cuando se trabaja con primitivos cualquier tipo de asignación se enviará como valor, (sin afectar el mismo espacio en memoria), al igual que cuando lo mandamos por valor a una función

**Ejemplo:**

En este ejemplo la variable a se le asigna el valor de 5, y la variable b se le iguala el valor de a

****

Pero cuando le asignamos un nuevo valor a igualándola a 10 (a = 10), entonces b no toma el nuevo valor de a porque ya tiene un valor guardado en memoria

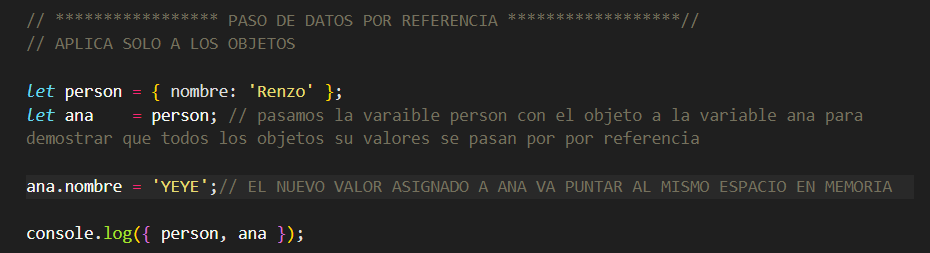
****

Si imprimimos los resultados que a vale 10 y b vale 5, porque a toma el valor por asignación al igual que b que esta previamente guardado en memoria

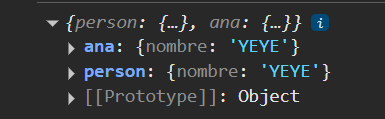
**NOTA:** TODOS LOS PRIMITIVOS SON PASADOS POR VALOR Y TODOS LOS OBJETOS SON PASADO POR REFRENCIA

A diferencia de los objetos que se pasan los valores por referencia, PORQUE APUNTAMOS AL MISMO LUGAR EN MEMORIA --🡪 OJO

**EJEMPLO**



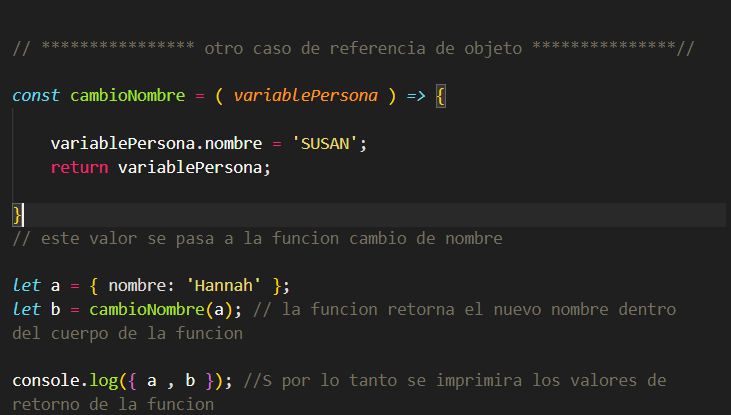
**Imprimimos en consola y demostramos el valor que tiene la variable person va ser igual al nuevo valor asignado a la variable ana**



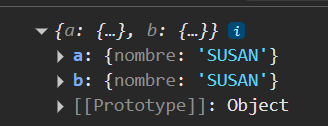
**Otro caso cuando utilizamos una función y creamos un objeto**

**En este caso usamos la función cambioNombre que retorne el valor de la variablePersona.**

**Como b es igual al nombre de la funcion CambioNombre y como argumento el valor de a, entonces el resultado que retorne la función quedara asignado a ambas variable porque apuntan al mismo lugar en memoria**

****

Imprimo en consola el resultado de a y b



**¿Como romper la relación con los objetos?**

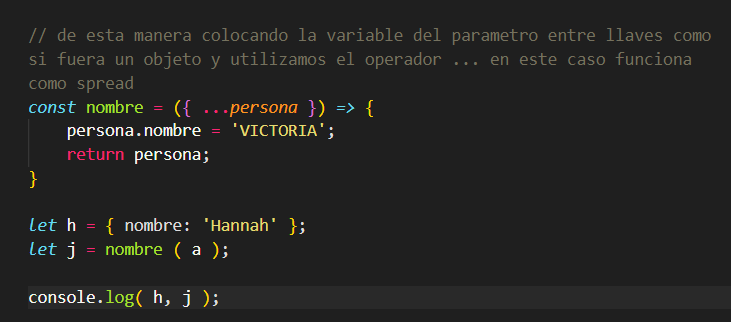
En JavaScript, el operador **...** puede actuar como operador spread o como operador rest, dependiendo del contexto. **El operador spread** se utiliza para expandir elementos de un array u objeto, mientras que **el operador rest** se usa para agrupar múltiples elementos en un solo array.

**NOTA:** Si queremos crear una copia del objeto y romper la relación que existe por referencia, lo que hacemos es:

* colocamos entre llaves y utilizamos el operador spread { ...objeto } para que no se vea alterado esa copia
* el operador spread separa los elementos y Crea una copia superficial de un objeto. evitando que apunte al mismo lugar en memoria al hacer la igualdad.

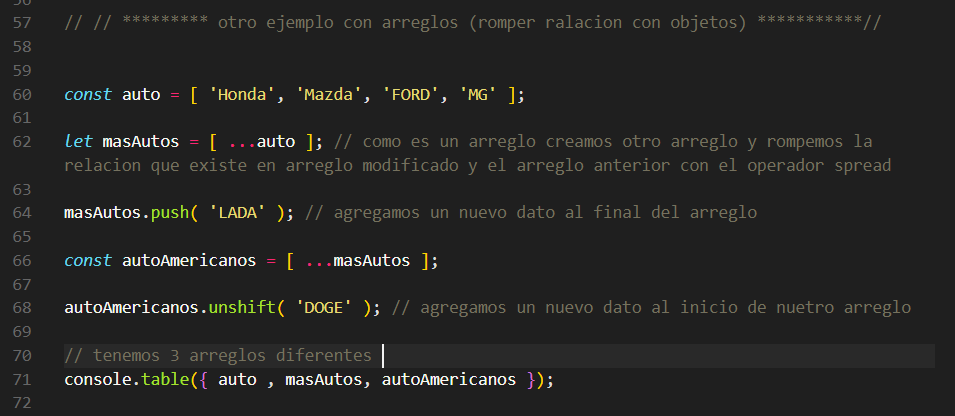
Cabe destacar, que cuando se coloca el operador **… entre paréntesis de un parámetro o argumento (…variable) es un operador spread, quiere decir en javascript, une todos los elementos y tranformalos como un arreglo.**

**Y cuando lo usamos en cualquier otro lugar fuera de lo paréntesis se convierte en el operador rest, y simboliza separa los elementos, la ventaja de hacerlo de esta forma es que rompe la referencia en javascript.**

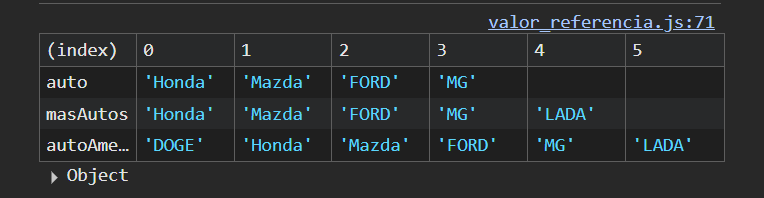


**Otro ejemplo aplicado a arreglos**

En este caso creamos 3 arreglos diferentes **el primer arreglo auto, el segundo arreglo masAutos igualando al arreglo auto, y el tercer arreglo autoAmericanos igualando a masAutos**



Y imprimimos en consola la 3 modificaciones que hicimos en los 3 arreglos diferentes, rompiendo la relación que existe entre ellos con el operador spread



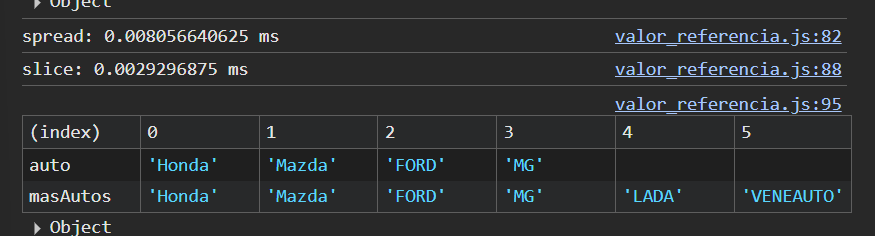
**Otra forma de romper la relacion es utilizando el método slice()**

El método slice() en JavaScript se utiliza para extraer una parte de un array o una cadena y devolverla como un nuevo array o cadena, respectivamente. No modifica el array o la cadena original.

Ejemplo:



**Imprimimos los resultados**

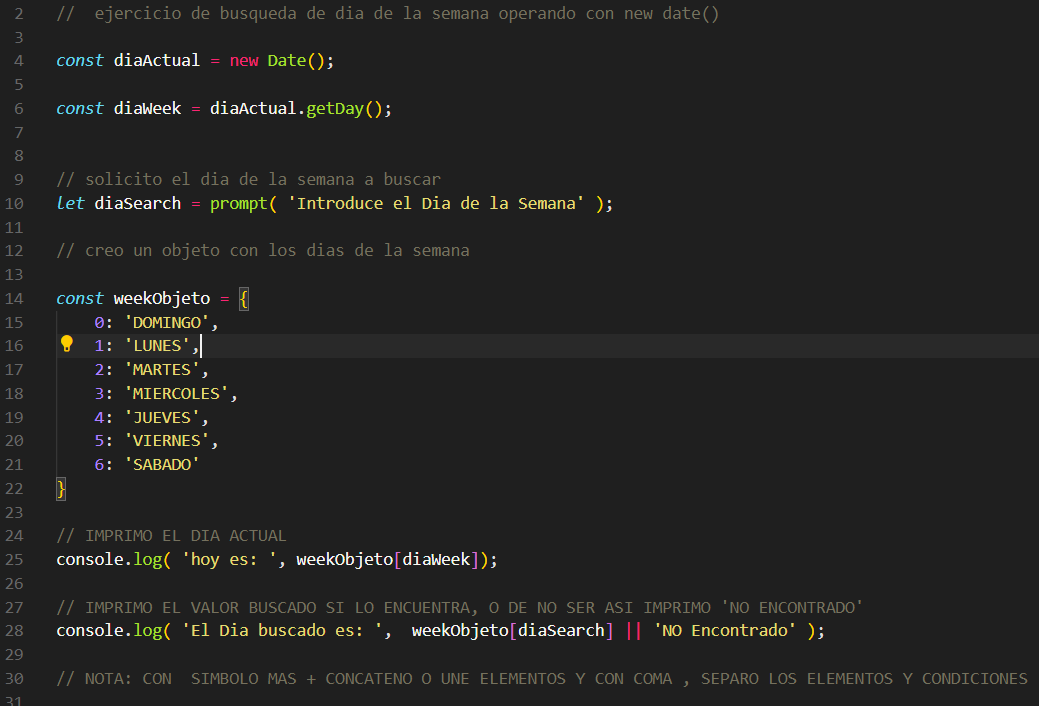


**IF Y ELSE**

**NOTA:** CUANDO SE USEN CONDICIONES NUNCA, **PERO NUNCA** SE DEBE USAR EL IGUAL =, PORQUE BASICAMENTE EL IGUAL = ASIGNA UN VALOR NO COMPARA --🡪**OJO**

EJERCICIO DEL DIA DE LA SEMANA QUE IMPRIMA EL DIA DE LA SEMANA ASIGNADO POR EL USUARIO SIN USAR IF Y ELSE

**CON UN OBJETO:**

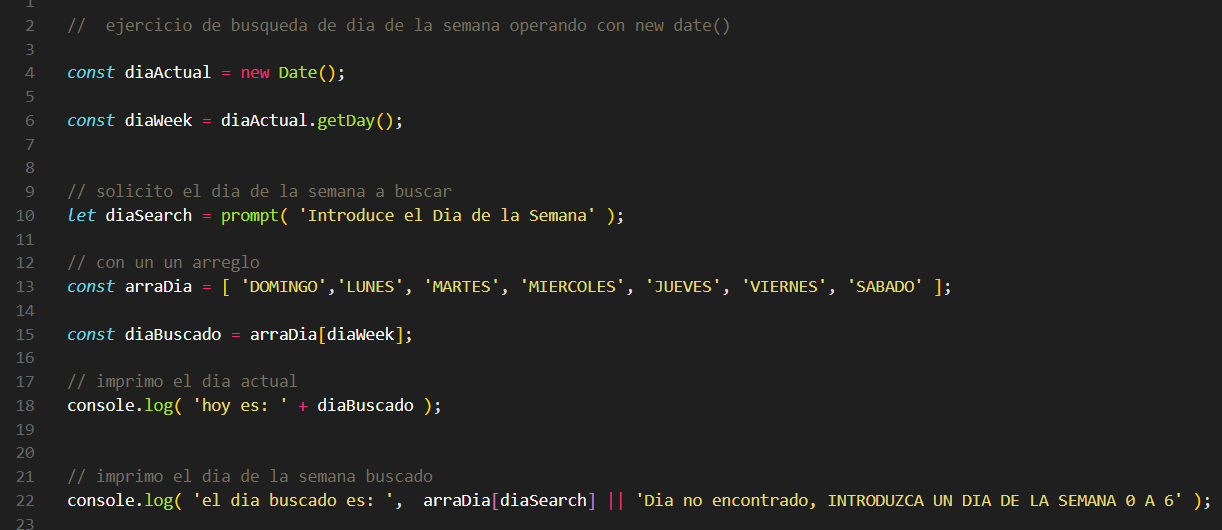


En JavaScript **new**se utiliza para crear instancias de objetos basados en prototipos definidos por funciones constructoras. Al usar new, se crea un nuevo objeto, se enlaza este objeto con la función constructora especificada.

Cabe destacar, que, el **new Date ()** se utiliza para crear un objeto Date (), que representa un punto específico en el tiempo (hora, mes, año, semanas, etc.). Y se pueden extraer esos datos.

Por ejemplo, **getDay()** es un método de la clase Date que devuelve el día de la semana de una fecha específica como un valor numérico, donde 0 representa el domingo, 1 el lunes, y así sucesivamente hasta 6 que representa el sábado.

**Con un arreglo**



**LÓGICA BOLEANA**

**&& AND**

Funciona si las condiciones se cumplen, cabe destacar que si la primera instrucción es FALSA no ejecuta la siguiente, a diferencia que si la primera instrucción es verdadera ejecuta la siguiente y así sucesivamente hasta que todas sean verdaderas o hasta encontrarse con un false.

**Nota:** Un uso común del AND &&, es que regresa los últimos valores si se encuentra con un false

**OR ||**

Es lo opuesto de AND &&, basta que una sola condición se cumpla para que se ejecute

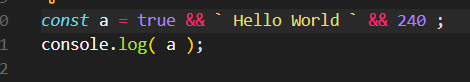
! NOT : negación, niega el valor original, es decir, si es false lo cambia a verdaderp-o y si es verdadero lo cambia a falso

**Protips asignaciones con operadores**

**Asignaciones con operadores**

**&& AND:**  cuando se asigna una comparación a una variable y se utiliza el operador AND, esté toma el ultimo valor de la comparación si son verdaderos, si se encuentra con el false termina la ejecución y retorna ese último valor.

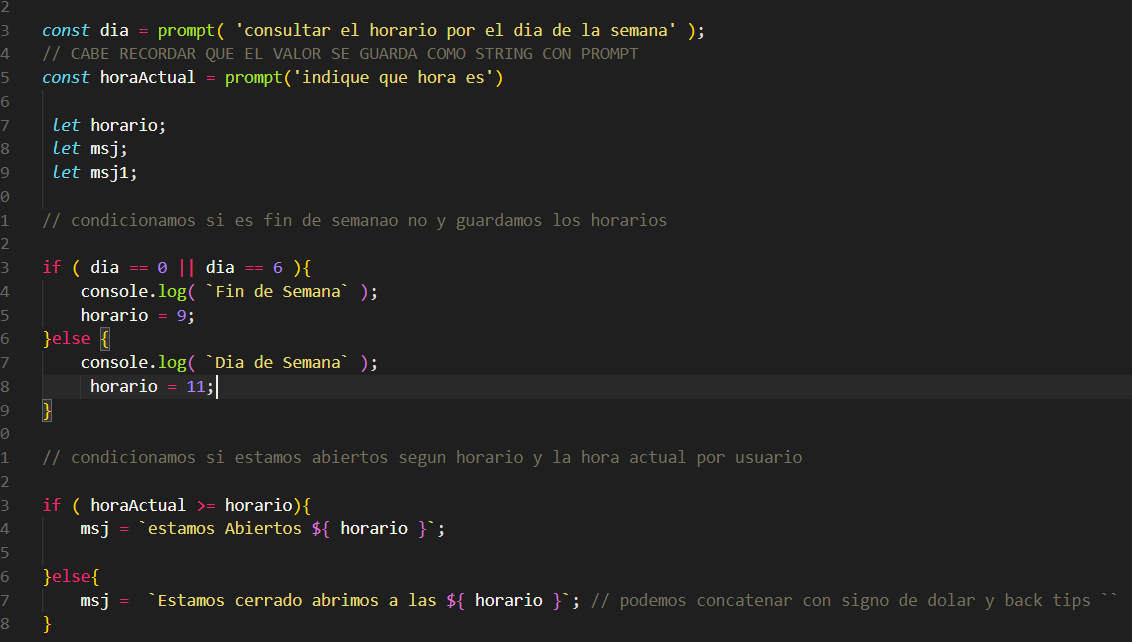
**Ejemplo:** en JavaScript, cuando se utiliza la asignación todos son valores





**OR ||:** Toma el primer valor que encuentre verdadero

**Ejercicio:** realizar una evaluación si es Fin de Semana o Dia de Semana, y el horario de apertura de una tienda



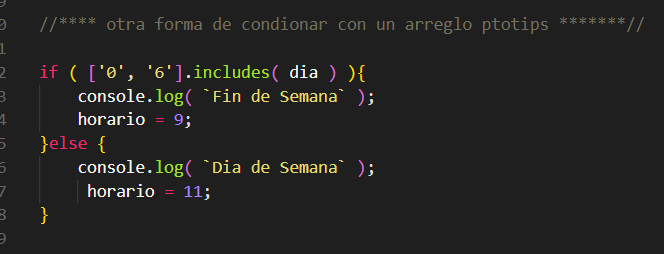
Usamos la estructura de control if y else tradicional y condicionamos si el dia es 0 or 6 ( cabe destacar que se usa == para que no evalué el tipo de dato (porque día es string)).

Así mismo determinamos si es domingo o sábado y guardamos el valor del horario para condicionar luego en otra estructura de control si estamos abiertos o no

**PROTIPS**

Otra forma de reducir a un más esa expresión condicional si es fin de semana o no, es con un arreglo ejemplo

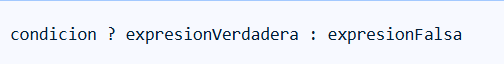
el método **includes()** se utiliza para determinar si una cadena o un array contiene un valor específico. Devuelve true si el valor se encuentra, y false en caso contrario.



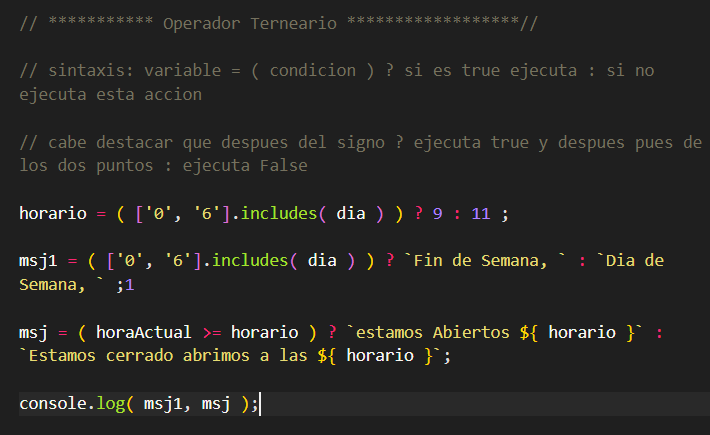
**Cabe destacar que el valor de día es string porque eso lo que arroja prompts y por eso los valores dentro del arreglo se colocan entre comillas simples**

**Operador ternario**

En JavaScript, el operador ternario es una forma abreviada de escribir una estructura condicional if-else en una sola línea. Es el único operador en JavaScript que toma tres operandos: una condición, una expresión que se ejecuta si la condición es verdadera y una expresión que se ejecuta si la condición es falsa

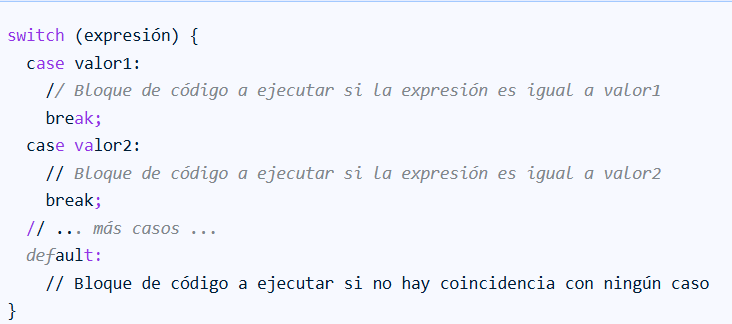


Siguiendo con el ejercicio anterior aplicamos el operador ternario



**switch javascript**

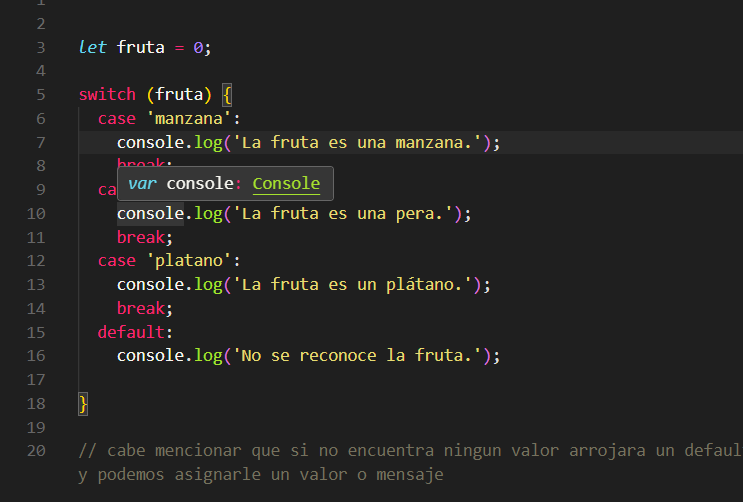
se utiliza para evaluar una expresión y ejecutar diferentes bloques de código según el valor de la expresión. Es una alternativa a las estructuras if-else largas, especialmente cuando se compara una variable con múltiples valores posibles.



Cabe destacar que switch evalúa estrictamente el valor de la condición es semejante al operador ===, es importante el tipo de valor en la condición--- > ojo

Cabe mencionar, que switch en JavaScript itera cada uno de los valores a partir de la condición si no se le coloca un break, para que no siga ejecutando las siguientes líneas de código y se salga del switch.

**la sentencia break** se utiliza para salir prematuramente de un bucle (for, while, do-while) o de una sentencia switch. Cuando se encuentra un break, la ejecución del bucle o switch se detiene y continúa con la siguiente instrucción después del bucle o switch que lo contiene.

Ejemplo 

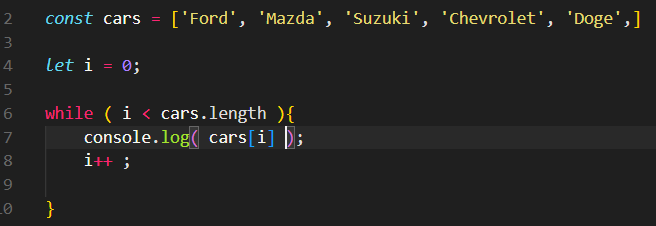
Es importante no confundir break con la sentencia continue. continue omite la iteración actual de un bucle y pasa a la siguiente iteración, mientras que break sale completamente del bucle.

**Ciclos o bucles**

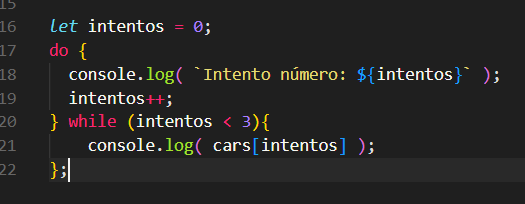
 los ciclos (o bucles) son estructuras que permiten ejecutar un bloque de código repetidamente, ya sea un número específico de veces o hasta que se cumpla una condición. Los tipos principales de ciclos en JavaScript son for, while, y do...while.

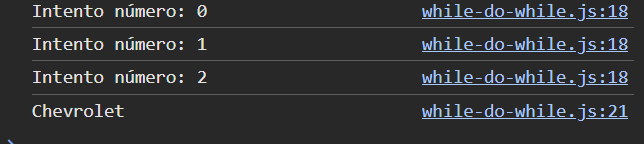
**1. Ciclo while:** Se utiliza cuando la cantidad de iteraciones no se conoce de antemano, y depende de una condición que se evalúa antes de cada iteración.

* El bloque de código se ejecuta mientras la condición sea verdadera.



**2. Ciclo do while:** Similar al ciclo while, pero la diferencia es que el bloque de código se ejecuta al menos una vez, porque la condición se evalúa después de la primera iteración. Es decir hacer lo del bloque do{ } mientras el bloque while {}



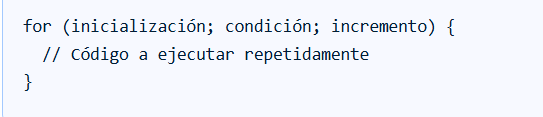


1. **El ciclo for:** es una estructura de control en programación que permite ejecutar un bloque de código repetidamente mientras se cumpla una condición específica, generalmente usando un contador. Es útil cuando se sabe de antemano cuántas veces se necesita repetir el código.

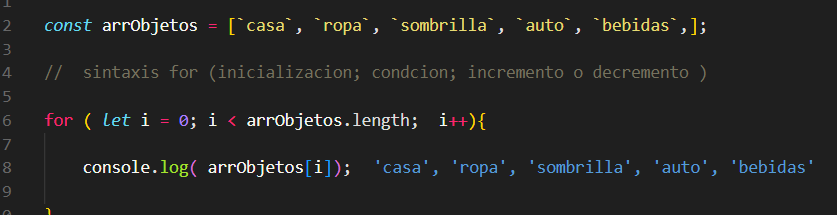
**¿Cómo funciona el ciclo for?**

Un ciclo for típicamente tiene tres partes principales:

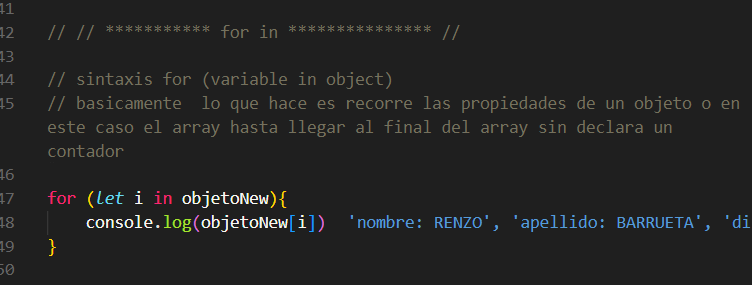
1. **1. Inicialización:** Se define e inicializa una variable de control (el contador). Esta inicialización solo ocurre una vez al inicio del ciclo.
2. **2. Condición:** Se evalúa una condición. Si la condición es verdadera, el código dentro del ciclo se ejecuta. Si es falsa, el ciclo termina.
3. **3. Incremento/Decremento:** Se actualiza el valor de la variable de control (normalmente incrementándola o decrementándola) después de cada iteración.



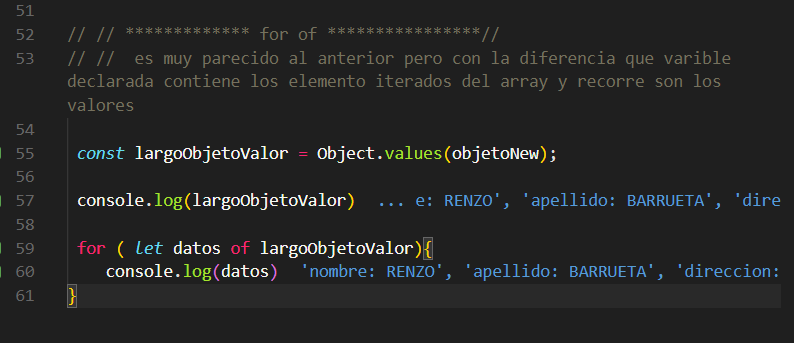
* **for- recorre un bloque de código varias veces**

****

* **for/in- recorre las propiedades de un objeto o datos de un array, en esencia cuenta la cantidad de datos que hay**

****

* **for/of- recorre los valores de un objeto iterable, en esencia guarda la cantidad de valores en la variable que recorrerá esos valores**

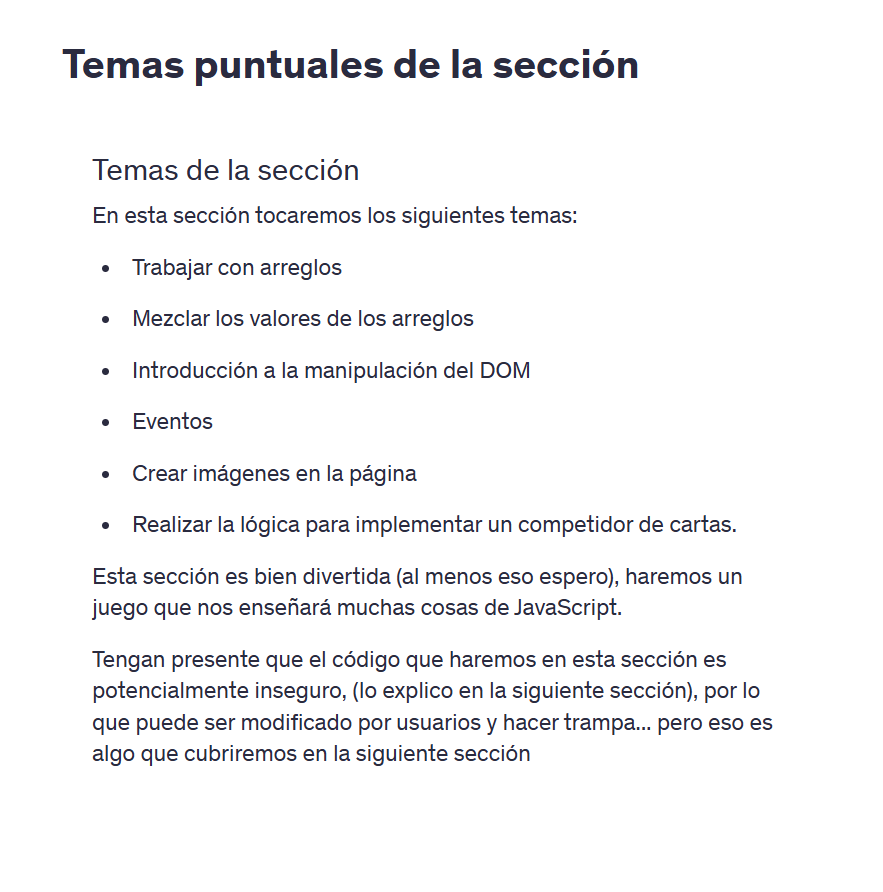
****

**Nota:** Cada uno de estos métodos devuelve una matriz que puede ser iterada con un objeto **Object.entries():** es un método se utiliza para convertir un objeto en un array de arrays, donde cada array interno contiene un par clave-valor del objeto original. En otras palabras, transforma un objeto en una lista de sus entradas, donde cada entrada es un array con la clave y su valor correspondiente.

**Object.keys():** Este método devuelve una matriz con las claves del objeto.

**Object.values():** Este método devuelve una matriz con los valores del objeto.

**Sección 5**



Valores foltis