



códigofacilito

Fundamentos de JavaScript

Uriel - CTO de Código Facilito





- >_ Variables y constantes
- >_ Tipos de datos y coerción de tipos
- >_ Trabajando con números
- >_ Trabajando con cadenas
- >_ Booleanos





>_ undefined y null

>_ Condiciones





Variables y constantes





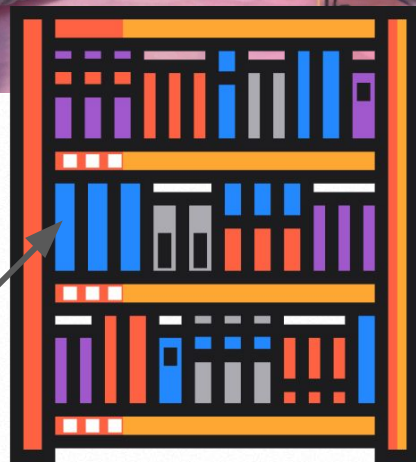
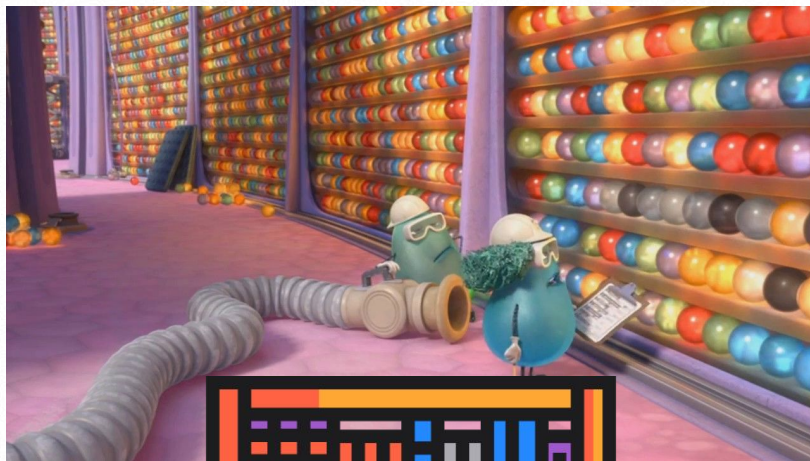
>_

**Las variables son etiquetas
para datos almacenados en
la memoria RAM.**

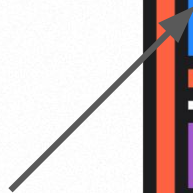




1



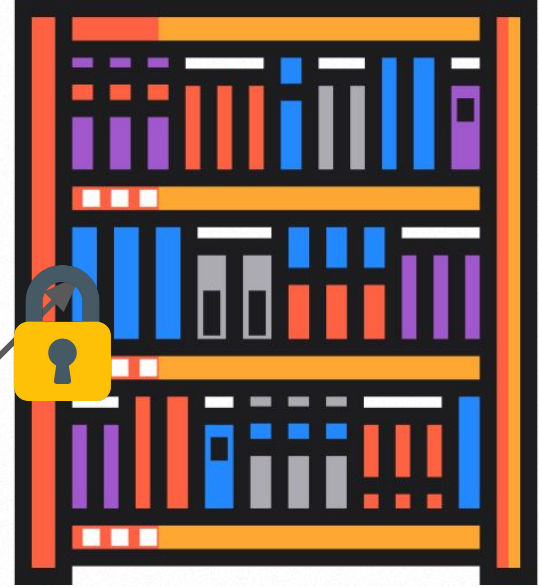
(x)





Las constantes son muy similares, apuntan o etiquetan un dato almacenado en la memoria RAM, pero no pueden reasignarse.

(x)





Variables en JavaScript





Para definir una variable en JavaScript podemos usar 3 sintaxis alternativas:



```
1 numero = 0;
```



```
1 var numero = 0;
```



```
1 let numero = 0;
```





**Palabra
reservada**



**Operador
asignación**



let/var numero = 0;



Identificador



**Valor para
asignar**





La diferencia entre **let** y **var** recae en un concepto muy importante de JavaScript: el scope o el alcance, del que hablaremos más adelante cuando revisemos el tema de funciones.





let numero;

Una variable puede declararse sin un valor inicial cuando usamos let o var

En cuyo caso el valor al que apunta es undefined





numero = 10;

Cuando una variable no utiliza ni var, ni let, esta se convierte en una variable global

No se recomienda que declares variables sin let o var.





Constantes





>_

Para definir una constante
utilizamos la palabra reservada
const:



```
1 const PI = 3.1416;
```





>_

La sintaxis es similar a la definición de variables, lo único que cambia entre su contraparte de variables es que si intentas reasignar una constante, recibirás un error

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains two lines of code. The first line is '1 const PI = 3.1416;' with 'const' in yellow, 'PI' in blue, and '3.1416' in red. The second line is '2 PI = 10;' with 'PI' in blue and '10' in red. To the right of the second line is a red circle containing a white 'X', indicating an error.

```
1 const PI = 3.1416;  
2 PI = 10;
```





**Las constantes pueden
representar un beneficio de
rendimiento en la ejecución
de tu código, aunque no uno
crítico.**





Cuándo usar variables y cuándo usar constantes





Existen distintas olas de pensamiento en lo que respecta al uso de constantes y variables, veamos dos de las más populares.





Todo es una constante hasta probar lo contrario.

En esta idea, el programador usará `const` siempre, y sólo cambiaremos una constante a variable si eventualmente en el código identificamos que este dato puede cambiar

Ventajas

Las constantes traen consigo una ventaja en rendimiento. Puede ser más fácil razonar el código.

Desventaja

Si todas las etiquetas son constantes, la ventaja cognitiva de usarlas se pierde, ya no se puede asumir que una constante no puede modificarse





Usa constantes sólo para etiquetas que aseguras nunca se reasignarán

En esta idea, el programador usará comúnmente `let`, y sólo usará `const` cuando se trate de un valor que no espera se modifique

Ventajas

Las constantes sirven como referencia semántica del código, entendemos que este dato es constante, como su nombre lo indica.

Desventaja

El código puede ser más difícil de razonar.





>_

No hay una respuesta definitiva sobre cuál usar, personalmente prefiero la opción 2 donde las constantes tienen un valor semántico que nos indica si este valor puede cambiar o no.





Inmutabilidad





Inmutabilidad es el concepto con que nos referimos a un objeto que no puede modificar su valor. En JavaScript los tipos primitivos son estructuras inmutables, algunos ejemplos son los números o las cadenas





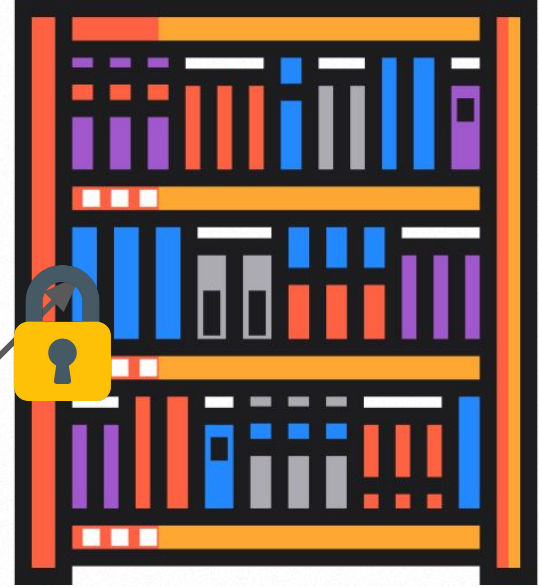
**Las estructuras más
complejas como los objetos o
los arreglos, no son
inmutables (mutables)**





Recordemos que las variables y las constantes son etiquetas que apuntan a un valor, en el caso de las constantes esta dirección de referencia no puede ser modificada.

(x)





Dicho esto, el valor al que apunta la constante puede ser modificado, por lo que no debemos asumir que el valor de una constante siempre será el mismo, sólo podemos asumir que apuntará al mismo valor.



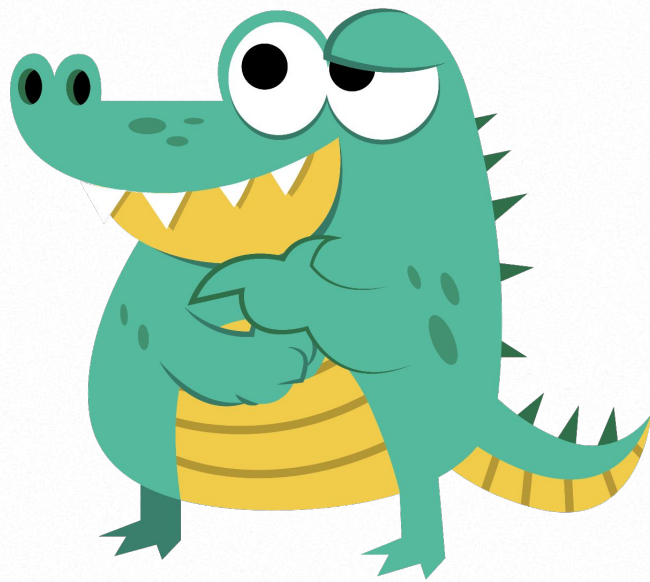


Constantes e inmutabilidad





En resumen esto quiere decir
que en JavaScript las
constantes deben servir como
una referencia semántica, no
como una certeza de que el
valor al que apuntamos no
cambiará





Ejercicio #1

Vamos a realizar un programa de lectura de datos e impresión, solicitaremos que el usuario ingrese su nombre, y luego imprimiremos su nombre





Tipado





Repaso General





Los tipos de datos son anotaciones que agregamos a la información de nuestro programa, para que el intérprete o el compilador usen y manejen esta información.





Los lenguajes de programación comúnmente pueden categorizarse en dos: débilmente y fuertemente tipados.





Fuertemente tipados

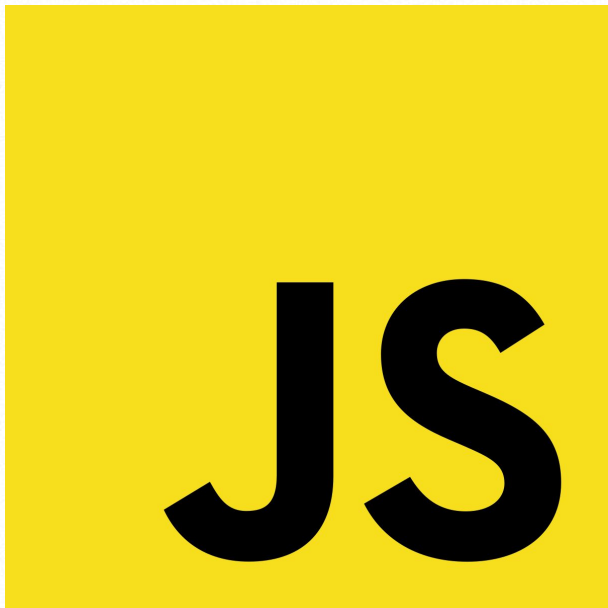
Son muy estrictos con los tipos y generalmente requieren que el tipo de dato se defina al declarar la variable, así como que la variable no se reasigne a un dato de distinto tipo





Débilmente tipados

No son tan estrictos, por lo que una variable puede cambiar y apuntar a valores de distintos tipos.





Algunos lenguajes incluyen tipado dinámico, en cuyos casos no es necesario especificar en el código de qué tipo será una variable, el intérprete puede asignar y definir el tipo de dato de la variable en tiempo de ejecución.



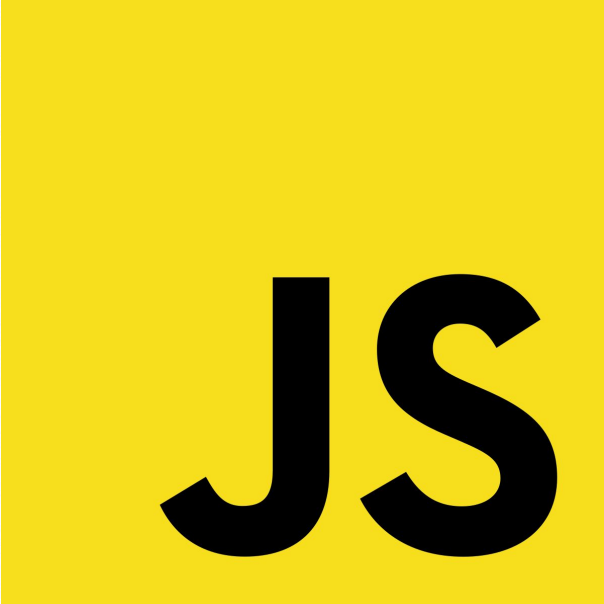


Tipos de datos en JavaScript





En JavaScript existen 6 tipos primitivos: string para el texto, number para los números, boolean para valores verdaderos o falsos, undefined para expresar la ausencia de valor, symbol, y null.



JS



Tipado, typeof y conversión de tipo.



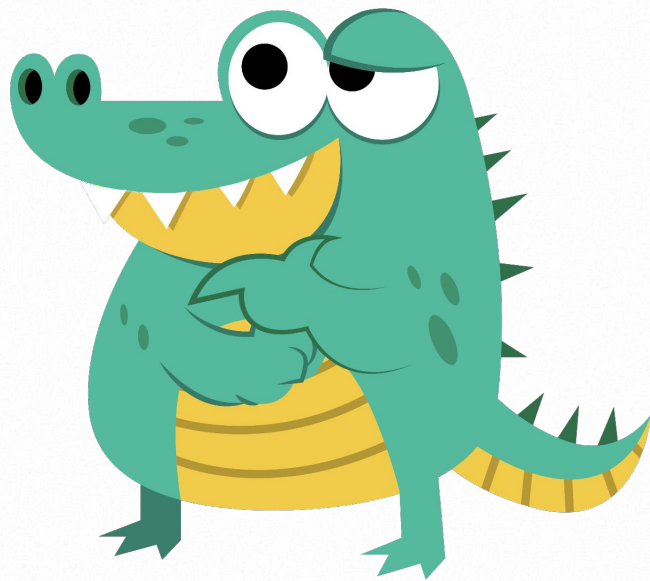


Type coercion





Cuando en nuestro código interactúan dos tipos de datos distintos en una misma operación, JavaScript hará una conversión implícita de datos a la que llamamos **type coercion**.





Type coercion en acción





Trabajando con números





Un número, es un valor que puede usarse en una operación matemática, en JavaScript, estos valores son del tipo Number.





- >_ No son objetos
- >_ No pueden tener propiedades
- >_ Son inmutables

Características de los primitivos





>_

Considerando este ejemplo.

¿Cambió el número?

```
1 let numero = 10;
```

```
2 numero = 20;
```



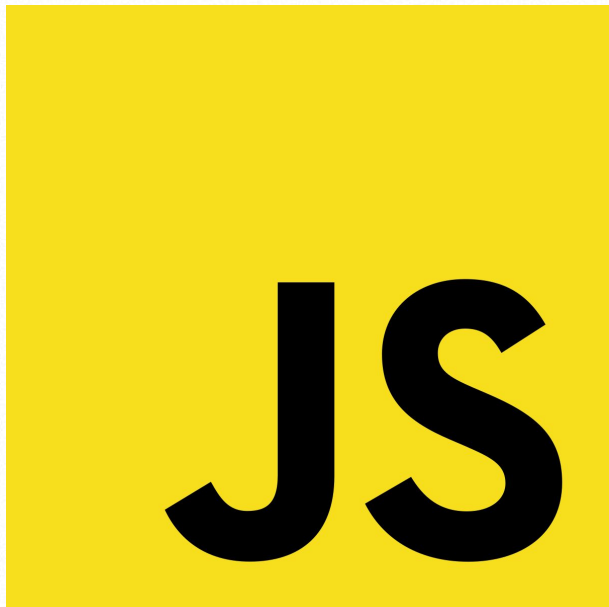


La respuesta es no, el número 18, o el número 20 son inmutables, no pueden modificarse, lo que estamos haciendo aquí es cambiar la referencia de la variable.





Esta característica del lenguaje permite que, si múltiples variables están usando un mismo número, todas apunten al mismo lugar en memoria.





Números primitivos, objetos y operadores aritméticos.





Ejercicio #2

Calcula la edad de una persona, solicita el año de nacimiento e imprime la edad





Trabajando con cadenas



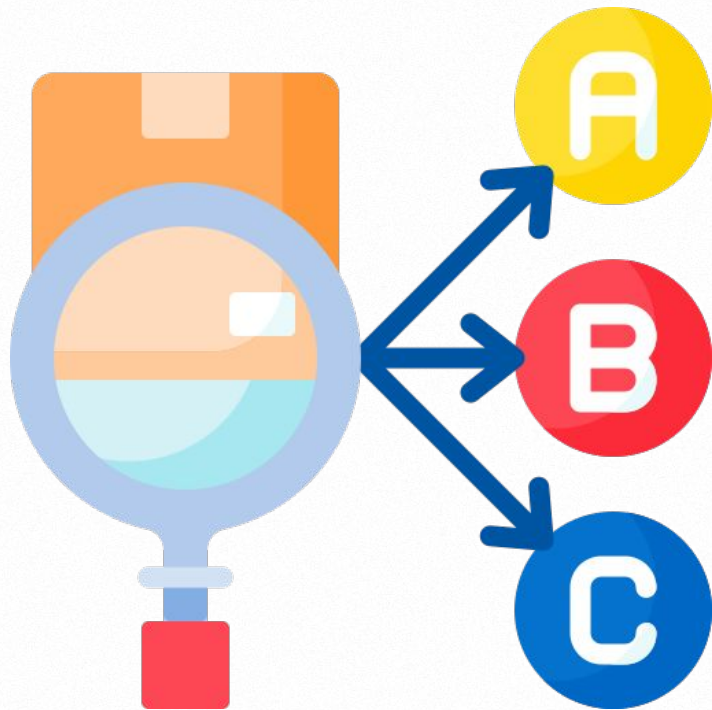


Repaso general



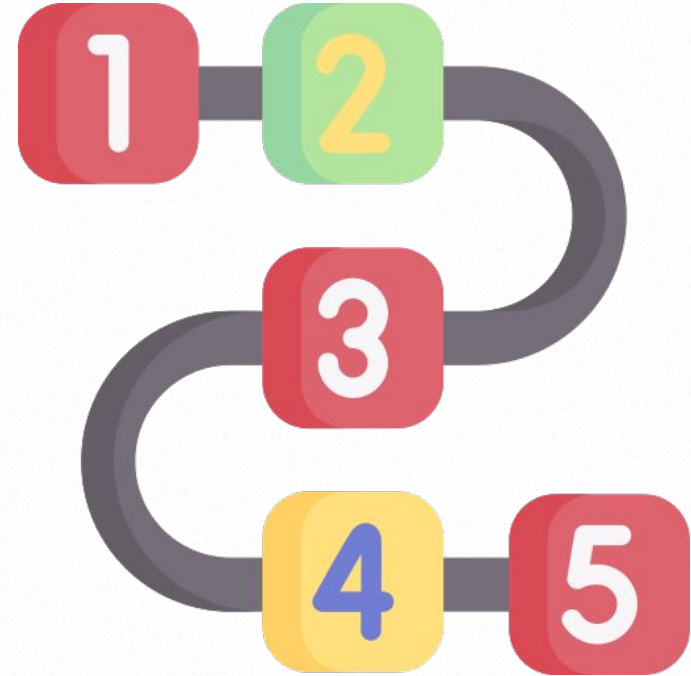


En términos técnicos, las cadenas según la definición del lenguaje, son secuencias ordenadas de 0 más valores unsigned int de 16bits, usados principalmente para representar texto.



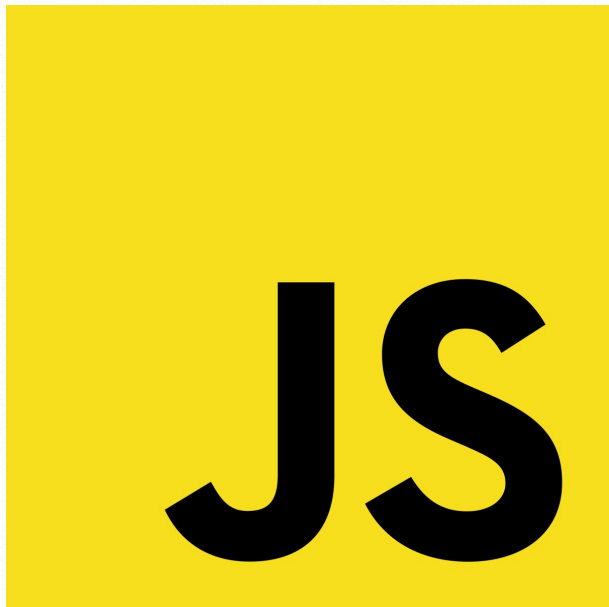


Históricamente, en ciencias de la computación nos hemos referido a las representaciones de texto como cadenas o strings, porque estas son una colección de elementos o caracteres, en un orden específico.





En JavaScript, a diferencia de otros lenguajes, no existe un tipo de dato para los caracteres que conforman una cadena. Un carácter es una cadena de 1 elemento





Cadenas en JavaScript





Concatenación e interpolación.





Concatenar es el término que usamos para la operación de unir dos cadenas una tras la otra, en el proceso de concatenación el inicio de una cadena se añade al final de otra.





Interpolar es el proceso de evaluar un string que contiene uno o varios marcadores que serán sustituidos por otro valor.





Ejercicio #3

Solicita del usuario:

- Su nombre
- 3 calificaciones

Calcula el promedio de las 3 calificaciones.

Imprime un mensaje final: "Hola <nombre> tu promedio calculado es <promedio>".





Booleanos



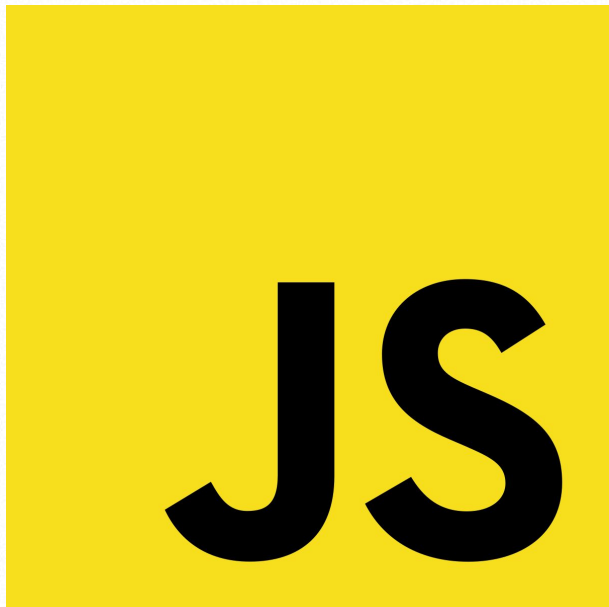


Repaso general



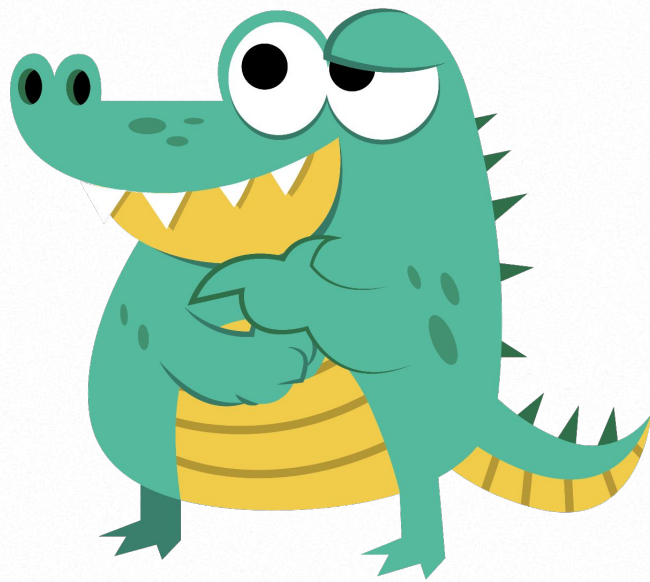


Así como existen las cadenas y los números, también existen los tipos booleanos, cuando un dato es del tipo booleano, significa que su valor únicamente puede ser True o False, Verdadero o Falso.






Decimos que una expresión es booleana cuando su resultado es True o False, y se componen de operadores de comparación, operadores lógicos, y otros tipos booleanos.





Operador	Significado	Ejemplo
===	Igual a	5 == 10 -> false
!==	Diferente de	5 !== 10 -> true
>	Mayor que	5 > 10 -> false
<	Menor que	5 < 10 -> true
>=	Mayor o igual	5 >= 10 -> false
<=	Menor o igual	5 <= 10 -> true





Operador	Significado	Ejemplo
&&	true si ambas expresiones son verdaderas, si no: false	5 == 10 -> false
	true si al menos una de las expresiones es verdadera, si no false	5 !== 10 -> true
!	Negación, invertimos la expresión booleana	5 > 10 -> false
??	Evalúa el valor o la expresión de la izquierda, si es nula o undefined, retorna el valor de la derecha:	5 < 10 -> true





Truthy y Falsy





>_

Decimos que un valor es Falsy cuando su representación booleana es falso.

Los valores truthy por su parte, son todos aquellos que no sean falsy, es decir que su representación booleana sea verdadero.





>_ NaN

>_ null

>_ 0, y -0

>_ ""

>_ false



Los valores Falsy en JavaScript son:





Cuando usamos un dato que no es booleano, donde se espera uno, JavaScript utilizará type coercion para convertirlo a su valor truthy o falsy.





Ejercicio #4

Determina si las siguientes expresiones son falsas o verdaderas.

Comenta en el chat, o anótalo para ti.





```
1 true
```





```
1 false
```





```
1 !true
```



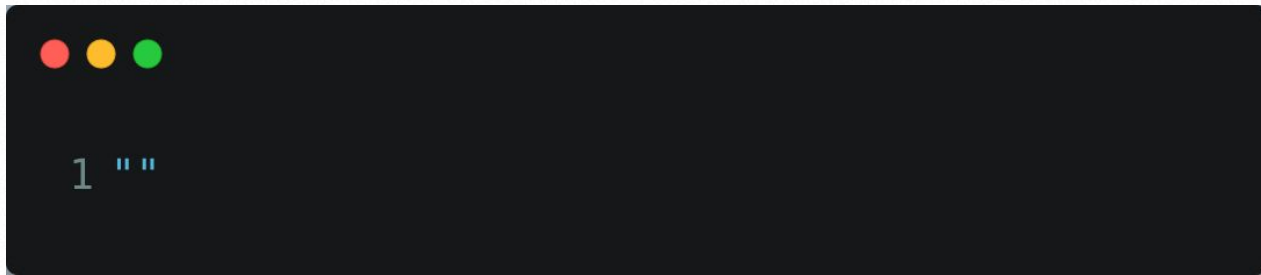


>_



1 0







>_



```
1 !!!""
```





```
1 null || "null"
```





```
1 [0]
```





>_

undefined y null





Ejercicio #5

Contesta las siguientes preguntas:

¿Por qué `undefined == null` es `true`?





Ejercicio #5

¿Por qué `undefined !== null` retorna `false`?





Resumen:

- **undefined** es un tipo de dato y un valor.
- El valor **undefined** es el único del tipo **undefined**.
- **null** es un objeto





Condiciones





**Palabra
reservada**

**Expresión que
condiciona**

```
if(<expresion-booleana>){  
  // Código a ejecutar  
}
```

**Definición de
un bloque**

**Instrucciones
a realizar**





```
if(<expresion-booleana>){  
  // Código a ejecutar  
}else {  
  // Código si la condición  
  // es falsa  
}
```



Esta primer condición puede acompañarse de una declaración else que nos permite indicar instrucciones a ejecutar si esta condición es evaluada como falsa



```
if(<expresion-booleana>){  
    // Código a ejecutar  
} else if(<expresion-booleana>){  
    // Código a ejecutar si la 1a expresión  
    // es falsa, y la segunda verdadera  
} else {  
    // Código si la condición es falsa  
}
```



Después de un if también podemos colocar una declaración else if, que nos permitirá colocar condiciones a evaluar si la primera fue falsa:



Operador ternario





El operador ternario es el único operador del lenguaje que trabaja con 3 elementos:

- **Una condición a evaluar**
- **Una expresión a ejecutar si la condición es verdadera**
- **Una expresión a ejecutar si la condición es falsa.**





Operador ternario





Ejercicio #5

Has consumido X cantidad de megas en wikipedia, y Y cantidad de megas en memes. EL costo de visitar wikipedia es de \$0.10 y el de los memes es de \$0.05.

Si el total consumido es mayor a \$100 imprime: "Consumo demasiado alto", si el consumo de ver memes es mayor al de leer wikipedia, imprime "wow, muchos memes"





¡Muchas gracias!

