# **Variables**

Es una posición de memoria en la cual se guarda un valor.

Ejemplo:



# **Variables: Strings**

Los strings son cadenas de texto. Para indicar que estamos usando una cadena de texto debemos de colocar las comillas simples.

En este curso utilizaremos los métodos:

* **toUpperCase**, que sirve para transformar un String a mayúsculas.
* **toLowerCase**, que sirve para transformar el string a minúsculas.

También usaremos el atributo:

* **length**, que nos indica la cantidad de caracteres que tiene un string.

Para concatenar dos strings se utiliza el símbolo (+)

var nombreCompleto = nombre + ’ ’ + apellido

# **Variables: Números**

En esta clase aprenderemos variables de tipo números y las operaciones más comunes en este tipo de variables.

En esta clase veremos operadores matemáticos como:

* suma ( + )
* resta ( - )
* multiplicación
* división

Recuerda que en operaciones con decimales debemos realizar operaciones adicionales para conseguir un resultado preciso. Ej. var total = ( precioDeVino \* 100 \* 3) / 100

Para redondear una operación se utiliza la función: **Math.round**

# **Funciones**

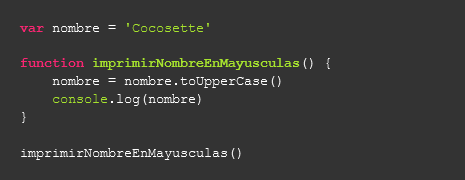
Las funciones son fracciones de código reutilizable. En esta clase aprenderemos a definir e invocar nuestras funciones. Para definir una función utilizaremos la palabra reservada ““function””.

Delimitamos el cuerpo de la función usando llaves { }. Los parámetros de la función son variables que se pasan a la función escribiéndolos entre paréntesis ()

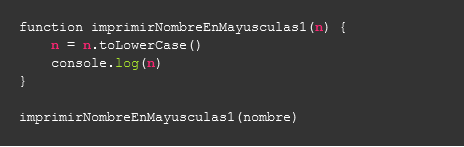
Definir funciones nos sirve para reutilizar código. JavaScript es un lenguaje interpretado, esto quiere decir que intentará ejecutar el código sin importar si los parámetros que le pasemos a la función estén invertidos o incluso incompletos.

# **El alcance de las funciones**

* **Variable global**: Variable que no está definida dentro de una funcion sino por fuera de ella.



* **Variable local**: Es una variable definida dentro de una función.



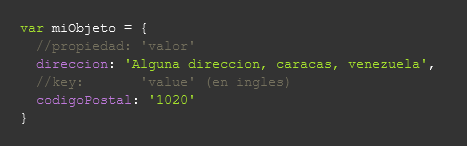
Para que la ejecución de una función no modifique una variable global usamos parámetros en lugar de pasar directamente la variable.

Es posible utilizar el mismo nombre para una variable global y para el parámetro de una función con un alcance local.

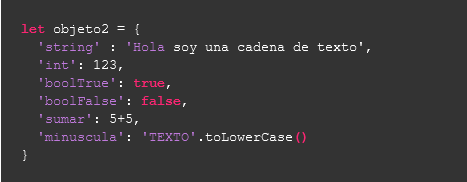
# **Objetos**

Los objetos se definen con **{}** (llaves) para asignarlo a una variable se hace de esta manera **var ejemplo =** **{}**

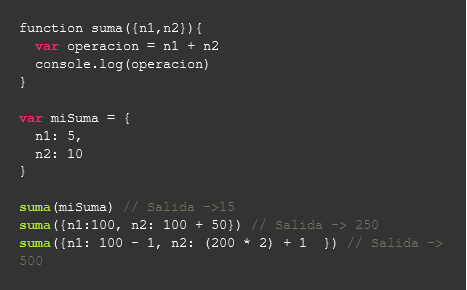
Los objetos se componen por **propiedades** y **valores**, se utiliza ’**,**' para separar cada propiedad.



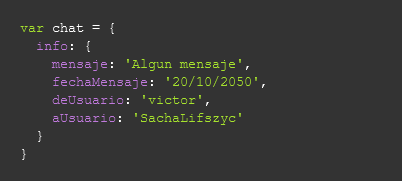
Los objectos pueden recibir como valor en sus propiedades, cadenas de texto (string), valores numéricos (int) y booleanos. También puedes realizar operaciones dentro de los valores.



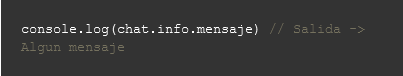
Los objetos se pueden enviar por parámetro a alguna función.



Las propiedades en los objetos pueden recibir objetos.



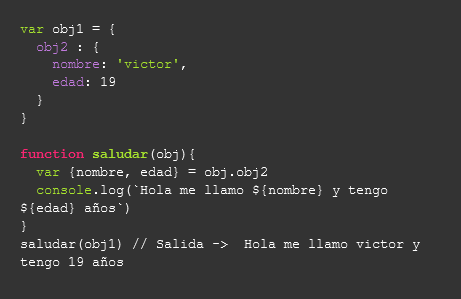
En este caso para acceder a el mensaje del objeto chat seria de esta manera:



# **Desestructurar objetos**

Para no duplicar las variables introducir el nombre de la variable como parámetro de la segunda variable. Ej **var{nombre} = persona**

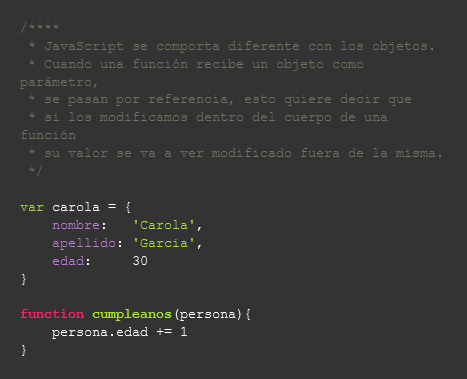
También se puede desestructurar un objeto de un objeto para obtener sus valores.



# **Parámetros como referencia o como valor**

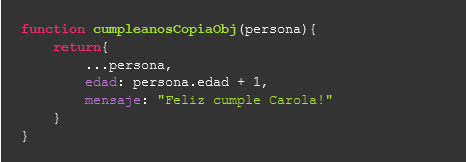
Javascript se comporta de manera distinta cuando le pasamos un **objeto** como **parámetro**.

Cuando los objetos se pasan como una referencia, estos se modifican fuera de la función. Para solucionar esto se puede crear un objeto diferente. Esto lo podemos hacer colocando tres puntos antes del nombre. Ej **…persona**.



Una de las soluciones para que cuando se le pase un **objeto** como **parámetro** y **no lo modifique**, sería correcto **copiar** ese objeto, **modificarlo** y **retornarlo**. También **se puede agregar atributos** al objeto copiado.

Para esto usamos **...Objeto**, **parametrosAModif**:



# **Comparaciones en JavaScript**

Existen varias maneras de comparar variables u objetos dentro de JavaScript.

Existen cinco tipos de datos que son primitivos y es necesario comprender al momento de hacer comparaciones.



**Variables**

En el primer ejemplo le asignamos a ‘**x**’ un valor numérico y a ‘**y**’ un string.

Para poder **compararlos** debemos agregar dos signos de igual “**==**”. Esto los **convierte al mismo tipo de valor** y permite que se puedan comparar.

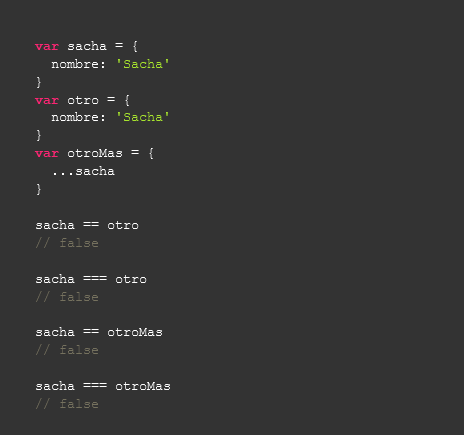


Cuando **realizamos** **operaciones** es recomendable usar tres símbolos de igual (**===**). Esto permite que JavaScript **no iguale las variables** que son de distinto tipo.

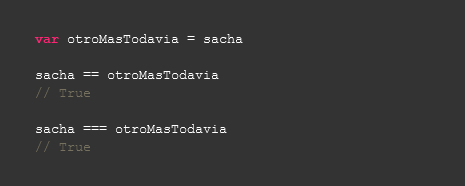


Al **comparar objetos** JS tiene en cuenta también el **nombre del objeto**, por lo tanto, se remite a comparar el **nombre de las variables** además del **valor de los atributos**.

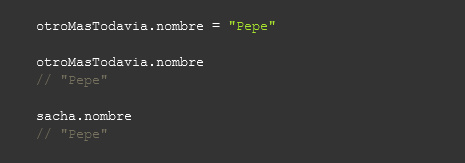
Con **objetos literales desglosados** (**otroMas** en este caso), pasa lo mismo y la comparación da **false** ya que lo que se genera es un nuevo objeto a partir del desglosado.



Si **asignamos el valor** del objeto **a una variable** y los comparamos, el **doble** y el **triple igual** darán como resultado ‘**true’** ya que en este caso las **dos variables** estarían refiriendo **al mismo espacio en la memoria RAM**.



Otra cosa a tener en cuenta es que, si **cambiamos el valor del atributo en la variable**, automáticamente **cambia el valor del objeto también**, por el mismo motivo que los operadores dan ‘**true’**, ambos refieren **al mismo espacio en la memoria RAM**.



# **Condicionales**

Los condicionales nos permiten decidir **si un código se ejecuta o no**. También introducimos un nuevo tipo de datos primitivos: el **booleano**, que determina si un valor es **falso** o **verdadero**. Mediante un condicional (**if**) decidiremos si se ejecuta una parte de nuestro código cuando se **cumpla** o **no** cierta condición.

# **Funciones que retornan valores**

Debemos de tener en cuenta que el número 18 **está** incluido dentro del rango de edad. Para ello utilizamos los símbolos **>=**.

**Return** detiene la ejecución de una función y **devuelve** el valor de esa función.

Las variables definidas con **const** se comportan como las variables, **excepto que no pueden ser reasignadas**. Las constantes pueden ser declaradas en mayúsculas o minúsculas. Pero por convención, para distinguirlas del resto de variables, se escribe todo en **mayúsculas**.

# **Arrow functions**

Permiten una nomenclatura más corta para escribir expresiones de funciones. Este tipo de funciones deben definirse antes de ser utilizadas.

Al escribir las **Arrow Functions** no es necesario escribir la palabra **function**, la palabra **return**, ni las **llaves**.

