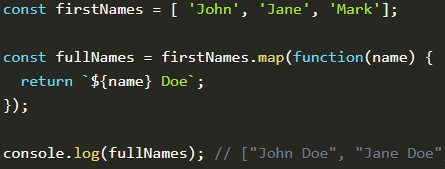
ECMASCRIPT 6

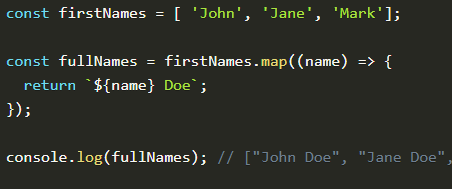
Arrow functions

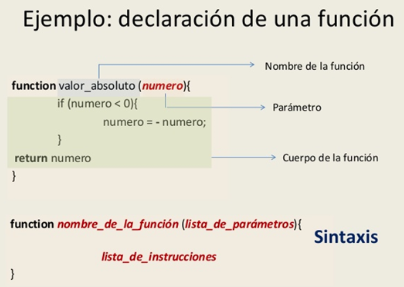
Las funciones de flecha o arrow functions es otra de las novedades de esta versión de JavaScript. Presentan una sintaxis más concisa comparada con las funciones normales, también cuentas con returns implícitos y no cambian el valor de this.

Sintaxis

Para escribir arrow functions solo necesitamos eliminar el keyword function y agregar lo que se conoce como fat arrow (=>) después de los parámetros de la función. La funcionalidad es exactamente la misma.

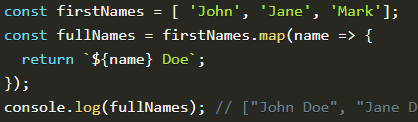






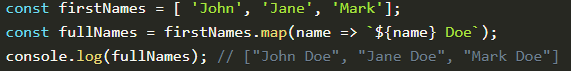
Paréntesis

La sintaxis de los arrow functions puede ser simplificada todavía más. Cuando solamente tenemos un parámetro en la función podemos eliminar los paréntesis. Si tenemos ninguno, dos o más parámetros, los paréntesis son necesarios.

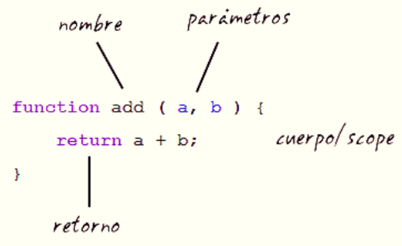


Return implícito

Cuando escribimos return estamos definiendo explícitamente lo que debe retornar la función. Muchas veces las funciones solo retornan un valor sin hacer alguna otra operación antes. Cuando este es el caso podemos omitir el return pues el arrow function asume que se va a retornar lo que sigue después del fat arrow.



De esta manera podemos colocar todo en una sola línea, además de eliminar el return también debemos eliminar las llaves (ya que estos definen un bloque de líneas). No es necesario definir qué queremos retornar ${name} Doe, esto queda implícito.



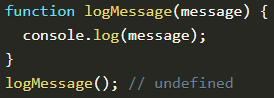
Funciones anónimas

Si usamos este tipo de funciones no podemos asignarle un nombre ya que los arrow functions son funciones anónimas. La forma más común de uso es asignar estas funciones a variables que sí podemos nombrar.

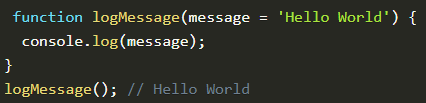


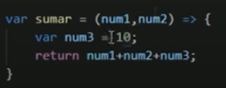
DEFAULT PARAMETERS

Por default todos los parámetros de una función se definen como undefined, este es el valor que tienen cuando no pasamos un argumento al momento de llamar una función.



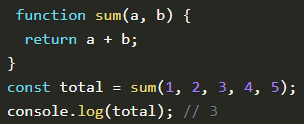
Cuando estamos definiendo los parámetros de una función podemos asignar un valor por default, este será asignado siempre y cuando el valor sea undefined, es decir, se omitió el argumento al momento de ejecutar la función.



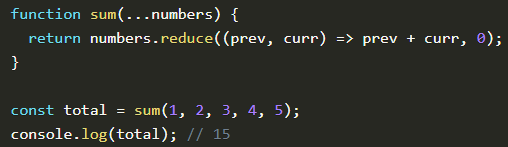


REST PARAMETER O PARAMETROS AGRUPADOS

Todas las funciones pueden ser llamadas con cualquier número de argumentos.



Esto no arroja ningún error ya que los argumentos excedentes son ignorados, solo se toman en cuenta los dos primeros. Usando la sintaxis del spread operator (...) podemos asignar cualquier cantidad de argumentos dentro de un arreglo.

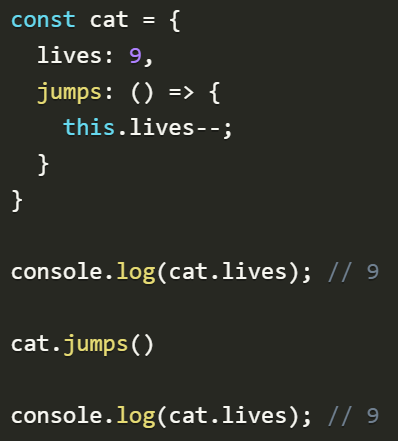


Spread operator y rest parameter tienen la misma sintaxis pero hacen lo opuesto. El primero nos permite extraer todos los elementos de un arreglo, mientras que el segundo crea un arreglo con todos los argumentos que recibe.

CUANDO NO USAR ARROW FUNCTIONS

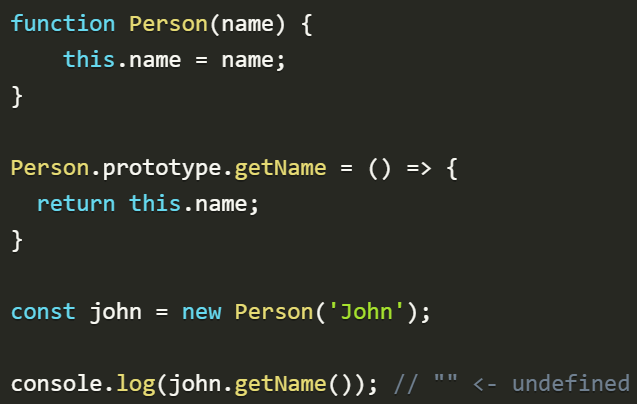
Métodos de un objeto

Después de llamar cat.jumps esperaríamos que lives disminuya su valor en uno. Este no es el caso porque this no está ligado a nada dentro de la función por lo que hereda el valor de this del scope padre, que en este caso viene siendo el objeto global window.



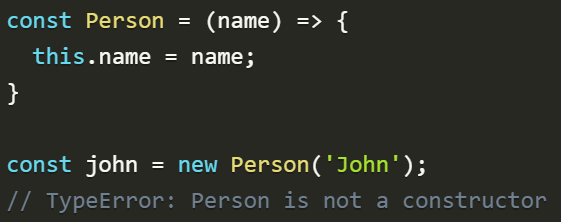
Object prototype

Similar al caso anterior, this está haciendo referencia a window por lo que obtenemos undefined ya que name no se encuentra en window.



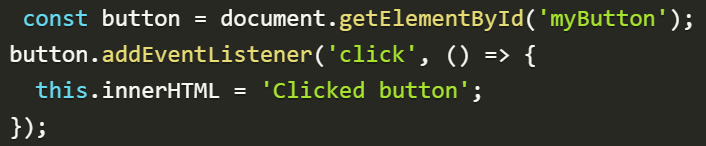
Function Constructor

El operador new crea un nuevo objeto vacío, después se ejecuta la función Person y en este contexto this permite que el constructor asigne las propiedades ya que apunta al objeto vacío. Como los arrow functions no tienen un this propio no pueden usarse para crear function constructors.



Callback con un contexto dinámico

Este es el caso cuando estamos trabajando con event listeners en elementos del DOM. Aquí el contexto this no cambia cuando se declara el arrow function por lo que sigue apuntando a window.



DEEP COMPARISON

El operador == compara objetos por identidad. En ocasiones lo que necesitamos es comparar los valores de sus propiedades.