Txema GG Docs

M4C7 JS Documentación

¿Qué diferencia a JavaScript de cualquier otro lenguaje de programación?

JavaScript se diferencia de otros lenguajes de programación en varios aspectos, como su sintaxis, su ejecución, su orientación a objetos, y su **enfoque** en la **web**; este último quizás sea su gran diferencia, ya que JavaScript es el único lenguaje de programación que un navegador web puede entender, ya que el resto de lenguajes necesitan un servidor.

Algunas de las aplicaciones más importantes del mundo, como por ejemplo *Gmail o Facebook*, están integradas en JavaScript y usan este lenguaje de manera extensiva.

Trabajar en JavaScript nos ofrece características más allá del texto estático. Tiene todas sus funciones ejecutadas en tiempo real en comparación con HTML y CSS. Esto incluye gráficos animados, mapas interactivos, etcétera. No necesitaremos un compilador para la configuración. En su lugar, Java Script crea y conecta su index.html y app.js, obteniendo así un prototipo funcional.

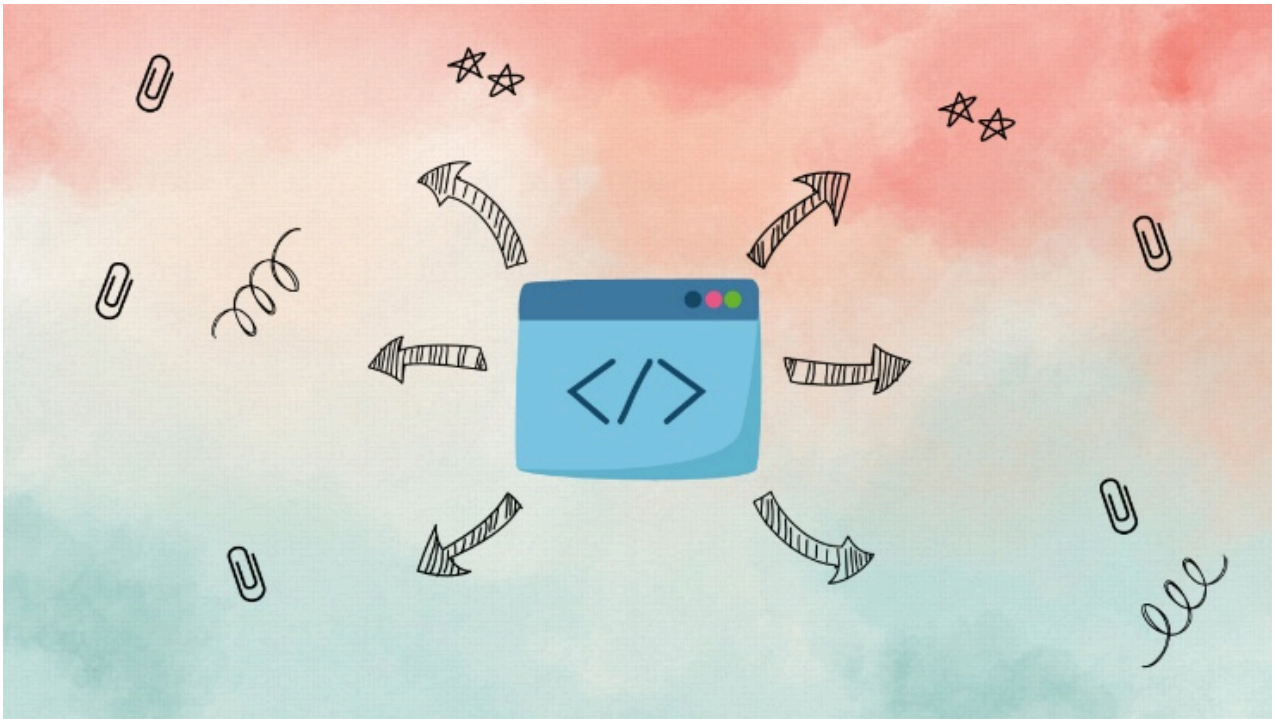
En este sentido, los administradores de paquetes proporcionados en este lenguaje permiten compartir código entre equipos y son simplistas en términos de uso. El punto de venta clave de JavaScript es definitivamente crear elementos para páginas web HTML y obtener acceso a los elementos tanto desde el servidor como desde el cliente.

En definitiva, según muchas personas especialistas, JavaScript ha logrado mantener su liderazgo durante ocho años consecutivos, y eso dice mucho de este lenguaje. La versatilidad proporcionada sigue siendo incomparable en comparación con otros lenguajes. El desarrollo continuo y la accesibilidad de *back-end* con múltiples crecimientos de dominios son lo que la convierte en la mejor opción entre los desarrolladores. Y en lo que respecta al futuro, parece que seguirá siendo inigualable durante mucho tiempo.

¿Por qué se utiliza JavaScript?

También es importante saber que JavaScript se utiliza mucho es un muy buen lenguaje para crear aplicaciones móviles.

Por otro lado, se utiliza muy a menudo porque es uno de los recursos más poderosos para automatizar el flujo de trabajo diario ya que funciona muy bien con la automatización de procesos y tareas.



¿Para qué se utiliza JavaScript? Una comparación entre lenguajes.

Python VS JavaScript:

- Originalmente, JS fue diseñado para admitir páginas HTML y Python fue diseñado para admitir servidores desde la línea de comandos.
- La flexibilidad de los bloques de parámetros para los métodos la ofrece Python y no JavaScript.
- Donde Python responde a las llamadas a funciones, JavaScript responde a los clics del ratón y las pulsaciones de teclas.
- Se crearon bibliotecas de Python para el análisis y procesamiento de datos y bibliotecas de JavaScript para manipular HTML en los navegadores web

Java VS JavaScript:

- JavaScript es un lenguaje de programación y Java es un lenguaje de programación centrado en la red.
- Java es un lenguaje fuertemente tipado, mientras que JavaScript es un lenguaje débilmente tipado.
- JavaScript es un lenguaje multiplataforma, mientras que Java no lo es.
- Antes de la ejecución en el cliente, Java debe compilarse en el servidor, mientras que JavaScript es interpretado por el cliente.
- JavaScript es un lenguaje dinámico y Java es un lenguaje estático.
- JavaScript tiene como objetivo la creación de páginas web interactivas.

C VS JavaScript:

- Donde JS se compila con el compilador just-in-time, C se compila de antemano.
- JavaScript inicialmente estaba incrustado en páginas web. Más tarde, ayudó a desarrollar aplicaciones de servidor desarrolladas a través de Node.js. Y C generalmente está integrado en sistemas de alto rendimiento.
- JavaScript no funciona directamente con la memoria del ordenador, mientras que C lo hace.
- En JS, los usuarios pueden hacer malabares entre varios trabajos, mientras que C ofrece un control explícito de los hilos.
- JavaScript se escribe dinámicamente en comparación con C que se escribe estáticamente.
- El manejo automático de bloques de memoria es posible con JS y no con C.
- El código debe recompilarse mientras se mueve a un procesador diferente para C, mientras que este no es el caso de JavaScript.
- JavaScript ha logrado ser la mejor opción entre los desarrolladores.

¿Cuál es la principal sintaxis de JavaScript?

La sintaxis de JavaScript es un conjunto de reglas que permiten a los programadores escribir código para desarrollar páginas web. Se basa fundamentalmente en lenguajes de programación como C y Java.

Características de la sintaxis de JavaScript:

- Es sensible a mayúsculas y minúsculas.
- Utiliza palabras clave para realizar diferentes funciones.
- Utiliza llaves para delimitar bloques de código.
- Utiliza punto y coma para finalizar sentencias.

Algunos ejemplos de sintaxis de JavaScript:

- Para definir variables, se pueden usar las palabras clave **const**, **var** y **let**.
- Para definir matrices, se pueden usar corchetes (``).
- Para definir objetos literales, se pueden usar llaves (**{}**).
- Para crear repeticiones, se pueden usar bucles **while** y **for**.
- Para hacer evaluaciones, se pueden usar las palabras clave **if** y **else**.
- Para definir si algo más sucederá, si algo sucede, se puede usar la palabra clave **switch**.

¿Cuáles son algunos tipos de datos JavaScript?

Los tipos de datos en JavaScript o como este lenguaje los categoriza muestran las diferencias existentes entre por ejemplo una variable que contiene un enunciado o una variable que contiene un número.

- Los **datos lógicos** o **datos boléanos**. Pueden tener dos valores, verdadero (T) o falso (F). Ejemplo:

```

var truthy = true;
```

```
var notTruthy = false;

console.log(truthuy);

console.log(notTruthy);
```

- Los **datos nulos** (*null*). Se refieren a la ausencia de cualquier tipo de valor, en este sentido *null* significa vacío.

```
var nully = null;

console.log(nully);
```

- Los **datos indefinidos** (undefined) nos permiten hacer algo que normalmente se usa más en la depuración, porque es exactamente lo que se obtiene cuando algo simplemente se declara sin asignarle un valor. En este sentido, es importante saber que, al declarar una variable en JavaScript por defecto, lo establece como indefinido.

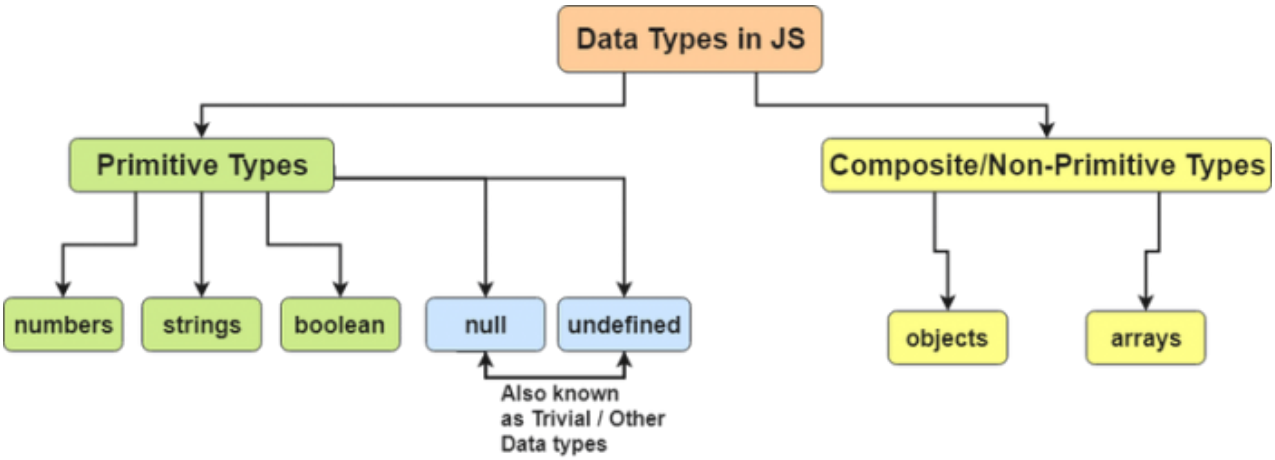
```
var notDefined;

console.log(notDefined);
```

- Los **datos numéricos**, lógicamente hacen referencia a la tipología de datos basada en números.

```
```

```
var age = 12;

console.log(age);
```

Respecto a los datos numéricos es importante saber que el ejemplo funciona porque **JavaScript** es un **tipo de lenguaje dinámico** y no estático como sucede con Java, C o C++. En este sentido, JavaScript es como Ruby o Python, donde el propio lenguaje de manera dinámica interpreta el tipo de dato que es.



- Los **datos "string"** hacen referencia a una cadena de caracteres. Normalmente veremos cadenas para palabras y oraciones. En este punto es importante anotar, que, en el lenguaje de JavaScript, en general no importa si se utiliza comillas dobles (**""**) o comillas (**''**). Así, como se puede ver en el siguiente ejemplo de sintaxis, el intérprete de JavaScript verá que ambos son compatibles, reconocerá las comillas y las establecerá como iguales al tipo de datos de la cadena.

```
var name = "Kristine";

var nameTwo = 'Jordan';

console.log(name);

console.log(nameTwo);
```

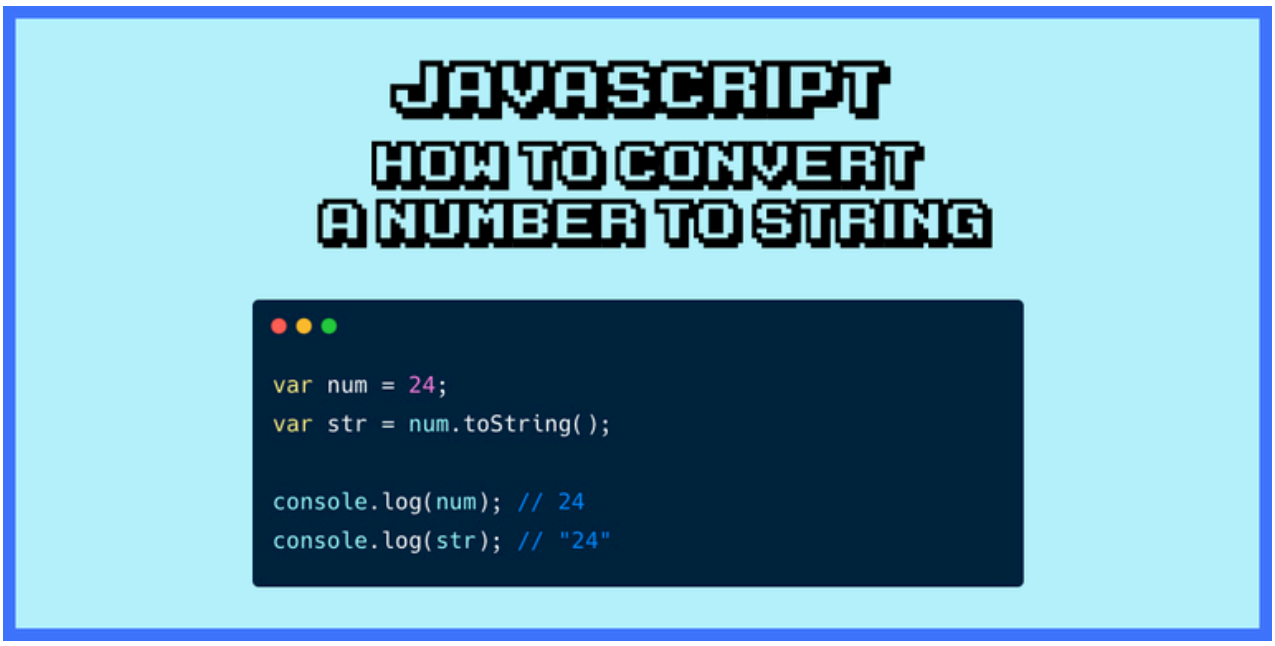
```

- Los **datos simbólicos** (symbol) son muy similares a las cadenas, exceptuando que no pueden ser cambiados, sólo puede haber uno de ellos. En este sentido, se puede decir que, en el lenguaje de JavaScript, esto es lo más parecido a un objeto inmutable.

```

```
var mySym = Symbol();

console.log(mySym);
```

¿Cuáles son las tres funciones de “string” en JavaScript?

En primer lugar, es importante recordar que las cadenas (“string”) son útiles para almacenar datos que se pueden representar en forma de texto. Del mismo modo es importante saber que una función permite “encapsular” un comportamiento. De este modo, la librería básica del lenguaje JavaScript provee de múltiples funciones.

En una función podremos tener un objeto y también poder cambiarlo o hacer búsquedas de valores dentro de él, utilizando menos código. Así, tres ejemplos de operaciones o funciones en cadenas sería verificar su ***length***, para construirlas y concatenarlas usando operadores de cadena + y +=, verificando la existencia o ubicación de subcadenas con ***indexOf()*** o extraer subcadenas con el método ***substring()***.



Del mismo modo, si tuviera que destacar tres de las funciones de las más comunes haría referencia a estas con sus correspondientes ejemplos y sintaxis:

1. La propiedad “**length**” se usa para obtener la longitud de una cadena de texto, o lo que es lo mismo, el número de caracteres que contiene.

```

var str = ‘Hola mundo’;

str.length;

console.log(str.length);

```

1. La propiedad “**toUpperCase()**” nos devuelve la cadena en mayúsculas; por el contrario, la propiedad “**toLowerCase()**” nos devuelve la cadena en minúsculas.

```

var str = ‘Hola mundo’;

str.toUpperCase();
```

console.log(str.toUpperCase());

```

1. La propiedad “**repeat**” permite repetir de manera consecutiva la cadena original, las veces que se indiquen. Pero, hay que tener en cuenta, que no modifica la cadena original, sólo la repite.

```

var str = ‘Hola mundo’;

str.repeat(5);

console.log(str.repeat(5));

```

**¿Qué es una condicional en JavaScript?**

Las condicionales son uno de los elementos básicos fundamentales para todo tipo de lenguaje de programación. Porque permiten un comportamiento dinámico en las aplicaciones. Así, debemos saber que hay condicionales simples y condicionales compuestas; aunque en el 95% de las ocasiones usaremos la condicional IF/ELSE.

Las condicionales en JavaScript son estructuras de control que permiten evaluar expresiones y ejecutar acciones. Son una herramienta fundamental para tomar decisiones dinámicas durante la ejecución de un programa.

Tres características fundamentales de las condicionales:

- Permiten crear ramificaciones en la lógica del software.
- Dividen el flujo del programa en diferentes caminos lógicos.
- Permiten adaptar el comportamiento del programa en función de las circunstancias.
```

En definitiva, en programación, las condicionales son una herramienta esencial que nos permite tomar decisiones dinámicas a lo largo de la ejecución de nuestro código y creación de nuestro programa.

Sintaxis básica para usar condicionales en JavaScript.

Las condicionales nos da la posibilidad de observar un par de valores o múltiples valores y poder ver como se relacionan entre sí.

Viendo un poco la sintaxis básica esta sería:

```

Igualdad: ==

Igualdad estricta (para igualar un dato numérico con número en cadena): ===

var age = 10;

var ageTwo = '12';

if (age === ageTwo) {

console.log('They are equal');

}

if (age !== ageTwo) {

console.log('Not equal');

}

if (age >= 25) {

console.log('Old enough to rent a car');

}
```

```
if (age <= 10) {

console.log('You can eat from the kid menu');

}
```

Estos serían los operadores lógicos en JavaScript:

OPERADORES LÓGICOS Y RELACIONALES	DESCRIPCIÓN	EJEMPLO
==	Es igual	a == b
===	Es estrictamente igual	a === b
!=	Es distinto	a != b
!==	Es estrictamente distinto	a !== b
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual	a <=b
&&	Operador and (y)	a && b
||	Operador or (o)	a || b
!	Operador not (no)	!a

Las condicionales IF/ELSE en JavaScript.

El objetivo principal es poder lograr un comportamiento más dinámico con las declaraciones: “If” y “Else”.

Un ejemplo sencillo para poder aprender la sintaxis sería este:

```
var age = 18;

if (age <= 8) {

console.log('Puedes comer el menú infalti');

} else {
```

```
console.log('Puedes comer el menú de adultos');

}
```

Los condicionales compuestos en JavaScript.

El lenguaje JavaScript también tiene la capacidad de configurar escenarios múltiples denominados condicionales compuestos en los que la sintaxis básica sería: if-else-if.

```
var age = 32;

if (age <= 10) {

console.log("You can eat from the kid's menu");

console.log("You are not old enough to drive");

console.log("You are not old enough to rent a car");

} else if (age >= 16 && age < 25) {

console.log("You can not eat from the kid's menu");

console.log("You are old enough to drive");

console.log("You are not old enough to rent a car");

} else if (age >= 25) {

console.log("You can not eat from the kid's menu");

console.log("You are old enough to drive");
```

```
console.log("You are old enough to rent a car");

}
```

¿Qué es un operador ternario en JavaScript?

El operador ternario en JavaScript, también conocido como operador condicional, es una herramienta que nos permite escribir declaraciones condicionales de una manera más concisa y elegante. Este operador es una forma abreviada de la estructura “if...else” y es útil muy cuando queremos tomar decisiones basadas en una condición.

Se compone de tres partes: una expresión condicional, una expresión que se evalúa si la condición es verdadera y otra expresión que se evalúa si la condición es falsa.

La sintaxis básica del operador ternario es la siguiente:

condicion ? expresion_verdadera : expresion_falsa

- Condición: una expresión que se evalúa como verdadera o falsa.
- Expresion_verdadera: la expresión que se ejecutará si la condición es verdadera.
- Expresion_falsa: la expresión que se ejecutará si la condición es falsa.

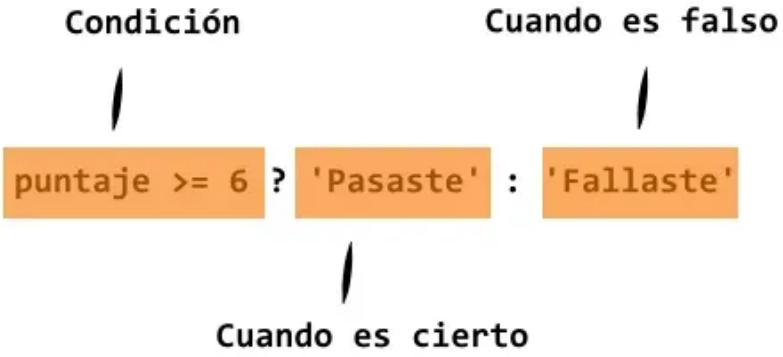
Un ejemplo básico sería este, supongamos que queremos escribir un mensaje que sea “Eres mayor de edad” si edad es mayor o igual a 18, y “Eres menor de edad” si edad es menor de 18. Podríamos escribir el código de la siguiente manera:

```
const edad = 18;

let mensaje;
```

```
if (edad >= 18) {

mensaje = "Eres mayor de edad";

} else {

mensaje = "Eres menor de edad";

}

console.log(mensaje);
```

Sin embargo, utilizando un condicional ternario, podemos simplificar este código de una manera mucho más compacta:

```
const edad = 18;

const mensaje = edad >= 18 ? 'Eres mayor de edad' : 'Eres menor de edad';

console.log(mensaje);
```



Buenas prácticas cuando utilizamos operadores ternarios.

El uso eficiente del operador ternario puede mejorar significativamente la legibilidad y la concisión del código. Por eso mismo es importante conocer las mejores prácticas clave para utilizar el operador ternario de manera efectiva:

- **Mantenlo simple y legible**: escribe expresiones concisas que sean fáciles de entender de un vistazo. Evita anidar demasiados operadores ternarios o escribir condiciones demasiado complejas.
- **Úsalo para asignaciones simples**: los operadores ternarios son ideales para asignaciones simples donde solo hay dos resultados posibles según una condición. Para escenarios más complejos, considera usar declaraciones if/else.
- **Sepa cuándo usarlo**: usa el operador ternario cuando necesitas realizar una verificación condicional simple y asignar un valor según el resultado. Es particularmente útil para asignar valores predeterminados o determinar el valor de una variable en función de una condición.
- **Pruebe minuciosamente**: prueba tu código minuciosamente para asegurarse de que el operador ternario se comporte como se espera en diferentes condiciones. Verifica casos extremos y valida la exactitud de los valores asignados.
- **Evite los ternarios anidados**: si bien es posible encadenar ternarios, el anidamiento excesivo puede generar código difícil de leer. Elige claridad y considera usar if/else para condiciones complejas.
- **Mantén los ternarios breves**: intenta mantener las expresiones ternarias breves y concisas. Los ternarios largos pueden ser difíciles de leer y comprender, lo que genera desafíos de mantenimiento del código.

Estas mejores prácticas describen pautas para utilizar eficazmente el operador ternario. Si bien no son reglas estrictas, ofrecen información valiosa para mejorar la claridad y legibilidad de nuestro código.

¿Cuál es la diferencia entre una declaración de función y una expresión de función en JavaScript?

En JavaScript, una **declaración de función** define una función como una sentencia independiente, mientras que una **expresión de función** define una función como parte de una expresión.

Las expresiones de funciones también se conocen como funciones anónimas o expresiones de función anónimas con nombre. Aunque normalmente se denominan expresiones de función, y así es como se sueles escuchar en el mundo del desarrollo de código.

Fijándonos en la sintaxis, el flujo es ligeramente diferente y existen diferencias muy sutiles entre las declaraciones de función, como la que tenemos aquí, y las expresiones de función:

```

var greeting = function () {

return "Hi there!";

};

var age = 4;
```

```
if (age <= 10) {

var buildMenu = function () {

return "Kids' Menu";

};

function wrongMenuBuilder () {

return "Wrong Kids' Menu";

}

console.log(buildMenu());

console.log(wrongMenuBuilder());

}
```

Como conclusión y a modo de resumen, es muy útil fijarse en las diferencias entre ellas. Así, aunque como hemos visto anteriormente una expresión de función es muy similar a una declaración de función y tiene prácticamente la misma sintaxis. Aun con todo, la principal diferencia entre una expresión de función y una declaración de función es el nombre de la función, que puede omitirse en las expresiones de función para crear funciones anónimas.

¿Qué es la palabra clave "this" en JavaScript?

En el lenguaje JavaScript, la palabra clave "this" se refiere a un objeto que depende del contexto en el que se utilice. Por ejemplo, "this" puede referirse al objeto de la ventana global, al objeto que recibe un evento, o al objeto al que está vinculado un método específico.

El concepto “this” significa “esto” en castellano y, como su nombre indica, hace referencia al objeto en cuestión. Es decir, si estamos creando cualquier función, la

palabra clave “this” se usará para representar o llamar al objeto que dicha función está modificando. Veamos para entenderlo mejor algunos ejemplos con su sintaxis correspondiente.

“This” en el contexto global, decimos que el contexto global es todo lo que se encuentra fuera de cualquier bloque de código. En este caso, “this” siempre hace referencia al objeto global:

```

console.log(this === window)

this.awesomeNumber = 37

console.log(window.awesomeNumber)

```

“This” en el contexto de una función; si invocamos “this” dentro de una función, su valor cambia dependiendo de cómo ejecutamos la función. Si es una llamada simple y no está en modo estricto, “this” devuelve el objeto global:

```

function whoIsThis() {

return this

}

console.log(whoIsThis() === window)

```

Siguiendo en el contexto de una función; si es una llamada simple y está en modo estricto, “this” conserva el valor que haya recibido antes de la ejecución de la función, y devuelve undefined si no ha recibido ninguno:

```

function whoIsThis() {

'use strict'

return this

}

console.log(whoIsThis())

```

Por otro lado, si la función es el método de un objeto y se invoca como tal, “this” es el objeto en sí mismo:

```

const me = {

name: 'Txema Gonzalez',

sayMyName() {

return this.name

},

}

console.log(me.sayMyName())

```

Para concluir, el último ejemplo se refiere a la función asignada como método de un objeto; así, cuando se define una función y luego se asigna como método de un

objeto, “this” dentro de la función se refiere al objeto al que se ha asignado la función:

```

function sayHello() {

console.log(`Hola, soy ${this.name}.`)

}

const person = {

name: 'Carlos',

greet: sayHello, // asignamos 'sayHello' a la propiedad 'greet'

}

person.greet()

```

En definitiva, estos son sólo algunos ejemplos del denominado multiverso “this”, existen algunos más, te invito a investigarlos para poder conocer en profundidad su funcionamiento dentro del lenguaje de JavaScript que hemos estado viendo en esta documentación.