# *10 abreviaturas JavaScript para principiantes*

JavaScript es el lenguaje de programación de lado de cliente más utilizado a la hora del desarrollo de aplicaciones web. Este lenguaje se encarga de dar cierta funcionalidad a los portales de Internet y de hacerlos más vistosas. Tener un buen manejo de este lenguaje, es fundamental para poder optimizar el código de la página y mejorar su velocidad de carga. Por este motivo, hoy queremos dedicar este White Paper a todos aquellos que se están iniciando en este lenguaje de programación.

A lo largo de este documento nos vamos a centrar a explicar el uso de algunas de las abreviaturas más útiles que nos ofrece JavaScript. Gracias a este código reducido conseguiremos una mayor velocidad de carga. Lo mejor de todo es que los códigos abreviados son tan válidos como sus versiones largas. Representan la misma cosa, sólo que en un formato más compacto.

JavaScript ofrece varias abreviaturas pero no tiene una guía oficial de referencia. Nos podemos encontrar algunas realmente sencillas y otras complejas de utilizar, incluso para desarrolladores experimentados. A continuación veremos 10 abreviaturas para principiantes que nos ofrece este lenguaje de programación.

## Números decimales

Hay ocasiones en las que tenemos que trabajar con grandes números, algo que puede resultar incómodo en el caso de tener que teclear todos los números. La abreviatura que os vamos a enseñar puede ser una bendición, ya que no será necesario escribir todos los ceros, sólo deberemos reemplazarlos con la notación ***e***. Por ejemplo, 24e8 equivaldría a 2.400.000.000, es decir, la cifra que aparece tras la ***e***, indica el número de ceros que hay que poner. Esto codificado en sintaxis de Java script, quedaría de la siguiente forma:

```
var myVar = 1e8;
```

Si lo pusiéramos sin abreviar, sería así:

```
var myVar = 100000000;
```

## Incrementar o decrementar un valor

En este caso se utiliza el doble signo más ***++*** para aumentar su valor en una unidad y el doble signo menos **--** para reducirlo. Estas dos abreviaturas se pueden utilizar únicamente con datos numéricos. Suelen ser muy utilizadas en el interior de los bucles. Su uso de forma abreviada es la siguiente:

```
i++;
j--;
```

Mientras que en su forma larga sería:

```
i=i+1;
j=j-1;
```

## Sumar, restar, multiplicar y dividir

¿Qué pasa si queremos incrementar o reducir el valor de una variable en más de una unidad? Para esto JavaScript también nos ofrece abreviaturas, al igual que para la multiplicación y la división. Su funcionamiento es similar a los operadores vistos en el punto anterior. Veamos un ejemplo de este código para cada una de las operaciones:

```
i+=125;
j-=21;
k*=25;
l/=4;
```

En el caso del código sin abreviar, el uso seria de esta forma:

```
i=i+125;
j=j-21;
k=k*25;
l=l/4;
```

## Mostrar el caracter que ocupa una posición en una cadena de texto

El método ***charAt()*** se encarga de devolver el caracter dado su posición. Aquí hay que resaltar que la posición del primer elemento no es el 1 sino el 0. Pues bien, además de este método, este lenguaje permite hacer lo mismo agregando la posición del caracter que queremos recuperar entre corchetes después del nombre de la variable. Supongamos que hemos declarado la siguiente variable:

```
var miCadena = "Alojamiento Web";
```

Recuperar la posición 4 en forma abreviada lo haríamos de la siguiente forma:

```
miCadena[4];
```

Mientras que utilizando el método charArt() se haría así:

```
miCadena.charAt(4);
```

# Declarar variables en masa

Si en algún momento del proceso de creación de nuestra página web necesitamos declarar más de una variable, no es necesario escribir una por línea, sino que podemos declarar todas en una única línea. Para esto lo único que debemos hacer es utilizar la palabra clave ***var*** y a continuación indicar las variables separadas por comas. Ejemplo de declaración de variables en masa:

```
var i, j=5, k="Buenos días", l, m=false;
```

Si las tuviéramos que declarar una por línea, sería de la siguiente forma:

```
var i;
var j=5;
var k="Buenos días";
var l;
var m=false;
```

Gracias a esta abreviatura, podemos declarar tanto variables definidas como variables que no tienen ningún valor aún.

# Declarar un array asociativo

Es una acción relativamente sencilla gracias a sintaxis ***var miArray = [“plátano”, “sandía”, “melón”]***. Sin embargo, si queremos declarar una matriz asociativa la cosa se complica un poco, ya que además de indicar el valor tenemos que añadir las claves, a no ser que sea un array regular donde sus claves sean 0, 1, 2….

Una matriz asociativa la podemos definir como una colección de pares clave-valor. La forma larga de definirla es la siguiente:

```
var miArray = new Array();
myArray["Cristiano Ronaldo"] = "Real Madrid";
myArray["Messi"] = "Barcelona";
myArray["Neymar"] = "PSG";
myArray["Griezman"] = "Atlético de Madrid";
myArray["Pogba"] = "Manchester United";
```

Sin embargo, lo podemos crear de una forma abreviada tal y como vemos en el ejemplo siguiente:

```
var miArray  = {
  "Cristiano Ronaldo": "Real Madrid",
  "Messi": "Barcelona",
  "Neymar": "PSG",
  "Griezman": "Atlético de Madrid",
  "Pogba": "Manchester United"
}
```

En este segundo caso no es necesario escribir línea por línea, sino que los pares clave-valor se van colocando separados por comas.

## Declarar un objeto

JavaScript es un lenguaje orientado a objetos. Declarar este tipo de elementos es muy similar al caso visto anteriormente con los arrays asociativos, pero con la diferencia de que en vez de haber pares clave-valor lo que nos encontramos son claves propiedad-valor. Si no conocemos la forma abreviada de crear un objeto en JavaScript, el mecanismo habitual es el siguiente:

```
var objPersona = new Object();
myObj.nombre = "Antonio Pérez";
myObj.lugarNacimiento = "Madrid";
myObj.edad = 58;
```

Es decir, al principio nos declaramos el objeto y luego mediante el operador punto **.** vamos añadiendo valores a las propiedades que forman parte del objeto, uno por línea.

En el caso del método abreviado no hace falta poner cada propiedad en una línea, sino que se puede poner toda en una única dentro de llaves. Veamos el ejemplo anterior de forma abreviada:

```
var objPersona = { nombre: "Antonio Pérez", lugarNacimiento: "Madrid", edad: 58
};
```

## Utilizar el operador condicional

El conjunto de instrucciones ***if-else*** es muy utilizado en cualquier lenguaje de programación para controlar el flujo del programa. Si se cumple la condición que se pone dentro del ***if***, se ejecutarán las instrucciones que haya a continuación. Si no se cumple, entonces se ejecutarán las que aparezcan dentro del ***else***. Un ejemplo de uso es el siguiente:

```
if( edad >= 18) {
  var mensaje = "Acceso Permitido";
} else {
  var mensaje = "Acceso Denegado";
}
```

Como alternativo a esta estructura, tenemos el operador ternario (?) y que consta de tres partes:

- La condición.
- Qué ocurre si la condición es verdadera.
- Qué sucede si la condición es falsa.

El ejemplo anterior realizado con este operador quedaría de la siguiente forma:

```
var mensaje = edad >= 18 ? "Acceso Permitido" : "Acceso Denegado";
```

# Revisar si una variable ha sido definida

Hay ocasiones en las que necesitamos comprobar si una variable está definida o no, es decir, si le hemos asignado algún valor que no sea ***null***, ***0***, ***valor vacío*** o ***false***. Como hemos venido viendo a lo largo de todo nuestro White Paper, tenemos una forma larga de hacerlo pero que puede resultad muy compleja, ya que debemos preguntar por todos los posibles casos. El código sería el siguiente:

```
var miVariable = 99;

if( typeof miVariable !== "undefined" && miVariable !==  "" && miVariable !==
null && miVariable !== 0 && miVariable !== false  ) {
  console.log("La variable miVariable está definida, tiene un valor asignado y
no es null ni false.");
}
```

Sin embargo, todo esto se puede hacer de una forma mucho más sencilla, tal y como podéis ver en el código siguiente:

```
var miVariable = 99;

if( miVariable ) {
  console.log("La variable miVariable está definida, tiene un valor asignado y
no es null ni false.");
}
```

El simple hecho de utilizar ***if( miVariable)*** hace que se compruebe si la variable está definida y si su valor es diferente de ***null***, ***0***, ***false*** o ***vacío***. Mucho más sencillo, ¿verdad?

# Comprobar que no está definida una variable

También podemos encontrarnos situaciones donde sea necesario comprobar que una variable no está definida. Para ello, podemos utilizar el método anterior pero acompañándolo de un signo de exclamación ***!*** antes de ella. Este signo es conocido como el operador lógico de negación, por lo que al anteponerlo en una expresión hace que esa expresión sea verdadera siempre y cuando no se cumpla. El código largo para determinar si una variable no ha sido definida, sería el siguiente:

```
var miVariable;

if( typeof miVariable === "undefined" || miVariable === "" || miVariable ===
null || miVariable === 0 || miVariable === false  ) {
  console.warn("La variable miVariable no está definida, está vacía, tiene valor
null o valor false.");
}
```

Por el contrario, si queremos utilizar el código abreviado sería el siguiente:

```
var miVariable;

if( !miVariable ) {
  console.warn("La variable miVariable no está definida, está vacía, tiene valor null o valor false.");
}
```

Como se puede observar el código es muy parecido al caso anterior, pero le hemos puesto delante del nombre de la variable el signo de exclamación.

A lo largo de este White Paper hemos visto una serie de abreviaturas que podemos utilizar en JavaScript y que nos permitirán crear código más reducido y simple.