Objetos nativos. Number

- C.F.G.S. DAW
- 6 horas semanales
- Mes aprox. de impartición: Nov

Carmelo José Jaén Díaz

- iPasen - cjaedia071@g.educaand.es

Índice

OBJETIVO

- Aprender cómo se puede manejar el tiempo en JavaScript.
- Saber cómo se programa la ejecución de funciones de forma puntual y periódica en el tiempo.
- Registrar en el front-end información de navegación, usuario, etc., mediante cookies o mediante el almacenamiento local que ofrece el nuevo estándar HTML5.
- Profundizar en el concepto de objeto.
- Conocer y manejar funciones relativas al lenguaje sobre arrays, strings, números.

GLOSARIO

Backtracking. Estrategia utilizada en algoritmos que resuelven problemas que tienen ciertas restricciones. Este término fue creado por primera vez por el matemático D. H. Lehmer en la década de los cincuenta.

BOM (Browser Object Model). Convención específica implementada por los navegadores para que JavaScript pudiese hacer uso de sus métodos y propiedades de forma uniforme.

Expresión regular. Secuencia de caracteres que forman un patrón determinado, generalmente un patrón de búsqueda.

NaN. Propiedad que indica Not a Number (valor no numérico).

Objeto window. Aquel que soportan todos los navegadores y que representa la ventana del navegador. Se estudiará en profundidad en capítulos posteriores.

URI (Uniform Resource Identifier). Cadena de caracteres que identifica un recurso en una red de forma unívoca. Una URI puede ser una URL, una URN o ambas.

GLOSARIO

URN. Localizador de recursos en la web que funciona de forma parecida a una URL, pero su principal diferencia es que no indica exactamente dónde se encuentra dicho objeto.

INTRODUCCIÓN

Los objetos nativos **Number** permiten trabajar con valores numéricos.

A diferencia de otros lenguajes de programación, en Javascript **solo hay un tipo de datos numérico**. Es decir, no hablamos de números enteros, dobles, en coma flotante... **¡todos los números se pueden representar con objetos de tipo *Number*!**

En esta lección vamos a ver varias operaciones con Number muy útiles:

- Cómo **cambiar de base entre números**: pasar a binario, octal o hexadecimal.
- Cómo representar el **-infinito** y el **+infinito**... ¡sí, hay dos infinitos diferentes!
- Cómo **comprobar si un dato es un número** o no.
- Cómo **representar un número como un objeto** y como un **dato primitivo**.

INTRODUCCIÓN

En Javascript todos los números pueden ser representados por el mismo tipo de dato numérico (*Number*) **¡Ni floats, ni doubles, ni integers!**

¡Es un all-in-one!

CARACTERÍSTICAS

- Solo hay un tipo de datos numérico.
- Podemos representar:
    - Enteros: 34
    - Decimales: 34.05
    - Números en notación científica positivos: 123e4 ($123 * 10^4$)
    - Números en notación científica negativos: 123e-4 ($123 * 10^{-4} = 123 / 10^4$)
    - Hexadecimales: 0xFF (255)

Todos los datos de tipo numérico se almacenan como tipo flotante de doble precisión.

CAMBIO DE BASE

Para ello se emplea el método *toString()*.

Ejemplo de uso:

```
let base10 = 128;

        alert (base10+" en base 2 es "+base10.toString(2));
        alert (base10+" en base 8 es "+base10.toString(8));
        alert (base10+" en base 16 es "+base10.toString(16));
```

+ INFINITO y - INFINITO

En programación entendemos un número infinito como el número más grande que podemos almacenar en una variable.

En JavaScript hacemos uso de las palabras reservadas *Infinity* y *-Infinity*.

Una forma rápida de obtener este dato es realizando una división entre cero.

```
alert ("División 2/0="+2/0);
alert ("División -2/0="+(-2/0));
alert ("Typeof: "+(2/0));
```

NOT A NUMBER: *NaN*

Es un dato peculiar que obtenemos cuando intentamos realizar una operación atípica, por ejemplo, dividir un número entre un *String*.

Nos indica que el tipo de dato no es un número, como su propio nombre indica.

```
let atipico = 100/"casa";
alert ("100/'casa'="+atipico); //NaN
```

Existe una particularidad en JavaScript al dividir un número entre una cadena que representa un número.

```
let atipico2 = 100/"10";
alert ("100/'10'="+atipico2); // 10
```

Esto es debido a que JavaScript hace una conversión interna de la cadena a número.

NOT A NUMBER: *NaN*

Para saber si un tipo de dato es *NaN*, utilizamos el método *isNaN()*.

```
alert ("¿Es un NaN atipico? "+isNaN(atipico)); //true
```

Si queremos saber qué tipo de dato es NaN, podemos hacer:

```
alert ("¿De qué tipo es NaN? "+typeof(atipico)); //Number
```

Aunque paradójicamente, NaN hace referencia a un tipo de dato no numérico, pertenece al objeto *Number*.

INSTANCIACIÓN DE *Number*

Para definir un número se puede hacer uso de la palabra *new* o definiendo una variable con un dato numérico.

DIFERENTES FORMAS DE INSTANCIACIÓN

```
//Con un dato numérico
        let num = 123;
//Haciendo uso del constructor de objetos
        let num2 = new Number(123);

alert ("num: "+typeof(num)); //Devuelve number
alert ("num2: "+typeof(num2)); //Devuelve object
```

Como se ha comentado anteriormente, es aconsejable declarar los datos numéricos mediante tipos primitivos, ya que el constructor de objetos es más lento y puede dar lugar a errores.

INSTANCIACIÓN DE *Number*

De lo anteriormente estudiado resulta inmediato deducir:

```
let num = 123;
let num2 = new Number(123);
let num3 = new Number(123);

alert ("num == num2: "+(num==num2)); //Devuelve true ya que tienen el
mismo valor.
alert ("num === num2: "+(num===num2)); //Devuelve false ya que aunque
tienen el mismo valor, no son el mismo tipo de dato (Number vs Object).
```

Aquí podemos ver uno de los errores de emplear el constructor de objetos:

```
alert ("num2 === num3: "+(num2===num3)); //Devuelve false, aún tratándose
del mismo valor y tipo de dato
```

PROPIEDADES Y MÉTODOS

A continuación, se estudian:

- Las propiedades del objeto: **MAX_VALUE**, **MIN_VALUE**, **NEGATIVE_INFINITY**, **POSITIVE_INFINITY** y, por supuesto, **NaN**.
- Aprenderemos cómo definir el número máximo de cifras y de decimales con **.toFixed** y **.toPrecision** y cómo representar en exponencial con **.toExponential**.
- Veremos cómo devolver un valor primitivo y una cadena a partir de un número con **.valueOf** y **.toString** y cómo devolver el valor numérico de una variable con el propio **Number()**.
- Y cómo no, uno de los métodos estrella que **utilizaremos cientos de veces** al capturar un número por teclado: el método que nos permite parsear una cadena a entero, **.parseInt** (o a decimal, con **.parseFloat**).

PROPIEDADES Y MÉTODOS

¡Recuerda! ¡Si queremos capturar un número por teclado en #Javascript mediante .prompt o un formulario, es necesario parsearlo a entero o a decimal con .parseInt o .parseFloat!

PROPIEDADES DE *Number*.
MAX_VALUE

Podemos encontrar todos los métodos de *Number* en W3Schools.com>JavaScript>JS References.

Recuerda que al trabajar con propiedades no hace falta poner los paréntesis al final.

Propiedad: MAX_VALUE

Finalidad: devuelve el valor máximo que puede tener un número en JavaScript. No hace falta declarar una variable y aplicar la propiedad, directamente podemos definirlo así:

Ejemplo de uso:

```
let maximo = Number.MAX_VALUE;
alert("Maximo="+maximo);
```

PROPIEDADES DE *Number*.
MIN_VALUE

Propiedad: MIN_VALUE

Finalidad: devuelve el valor mínimo que puede tener un número en JavaScript. No hace falta declarar una variable y aplicar la propiedad, directamente podemos definirlo así:

Ejemplo de uso:

```
let minimo = Number.MIN_VALUE;
alert("Mínimo="+minimo);
```

PROPIEDADES DE *Number*.
NEGATIVE_INFINITY

Propiedad: NEGATIVE_INFINITY

Finalidad: devuelve el infinito negativo. No hace falta declarar una variable y aplicar la propiedad, directamente podemos definirlo así:

Ejemplo de uso:

```
let neginf = Number.NEGATIVE_INFINITY;
alert("-Inf="+neginf);
```

PROPIEDADES DE *Number*.
POSITIVE_INFINITY

Propiedad: POSITIVE_INFINITY

Finalidad: devuelve el infinito positivo. No hace falta declarar una variable y aplicar la propiedad, directamente podemos definirlo así:

Ejemplo de uso:

```
let posinf = Number.POSITIVE_INFINITY;
alert("+Inf="+posinf);
```

PROPIEDADES DE *Number*.

NaN

Propiedad: NaN

Finalidad: devuelve un objeto de tipo no numérico. No hace falta declarar una variable y aplicar la propiedad, directamente podemos definirlo así:

Ejemplo de uso:

```
let numNan = Number.NaN;
alert("NaN="+numNan);
```

MÉTODOS DE *Number*.
toFixed

Método: toFixed(<num decimales>)

Finalidad: devuelve una cadena con el número específico de decimales indicado, realizando redondeo.

Ejemplo de uso:

```
let x = 9.8765;
alert (x.toFixed(0));
alert (x.toFixed(2));
alert (x.toFixed(6)); //Añade tantos ceros como se le especifique
```

MÉTODOS DE *Number*.
toPrecision

Método: toPrecision(<num cifras>)

Finalidad: devuelve una cadena con el número específico de cifras indicado, realizando redondeo.

Ejemplo de uso:

```
let y = 9.8765;
alert (y.toPrecision()); //Al no pasarle ningún argumento, nos
devuelve el número tal cual.
alert (y.toPrecision(2));
alert (y.toPrecision(6));
```

MÉTODOS DE *Number*.
toExponential

Método: toExponential(<num decimales>)

Finalidad: devuelve una cadena con el número redondeado a notación científica.

Ejemplo de uso:

```
let y = 9.8765;
alert (y.toExponential()); //9.8765e+0
alert (y.toExponential(2)); //9.88e+0
let z = 123456789;
alert (z.toExponential(2)); //1.23e+8
```

MÉTODOS DE *Number*.
valueOf

Método: valueOf()

Finalidad: devuelve el valor primitivo de un objeto. Es decir si tenemos un objeto de tipo number nos va a devolver un número.

Ejemplo de uso:

```
let num1 = new Number (123);
alert (num1.valueOf()); //123 como valor primitivo
```

MÉTODOS DE *Number*.
toString

Método: toString()

Finalidad: devuelve la cadena de un número.

Ejemplo de uso:

```
let j = 123;
alert(j.toString());
alert((123).toString()); //Sin necesidad de crear la variable
alert((123+7).toString()); //Pasándole una expresión
```

MÉTODOS GLOBALES.
Number

Los siguientes métodos globales no son sólo del objeto *Number*. Su finalidad es la de convertir variables en números.

Método: Number()

Finalidad: devuelve el valor numérico de una variable.

Ejemplo de uso:

```
alert (Number(true)); // Devuelve 1
alert (Number(false)); // Devuelve 0
alert (Number(new Date())); // Devuelve el número de
milisegundos desde el día 1/1/1970

alert (Number("10")); // Devuelve el número de la cadena
alert (Number("casa")); // Devuelve NaN
```

MÉTODOS GLOBALES.
parseInt y parseFloat

Los siguientes métodos globales no son sólo del objeto *Number*. Su finalidad es la de convertir variables en números.

Método: parseInt() / parseFloat()

Finalidad: devuelve el valor numérico (entero para *parseInt* y decimal para *parseFloat*) de una variable.

Ejemplo de uso:

```
alert(parseInt("10.6"));      //Devuelve el número
alert(parseInt("10 20"));     //Devuelve el primer número
alert(parseInt("10 casa"));   //Devuelve el primer número
alert(parseInt("casa 10"));   //Devuelve NaN
```