Objetos. Constructores integrados

- C.F.G.S. DAW
- 6 horas semanales
- Mes aprox. de impartición: Nov
- iPasen - cjaedia071@g.educaand.es

Carmelo José Jaén Díaz

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

En esta lección nos vamos a centrar únicamente en los **objetos nativos y sus constructores integrados.** Un constructor integrado nos permite definir un objeto del tipo que sea.

En este caso, vamos a ver cómo definir los siguientes tipos de objetos nativos:

- String
- Number
- Boolean
- Array
- Function
- Date
- Math
- Object

INTRODUCCIÓN

En general, todos los objetos nativos utilizan la siguiente sintaxis para definirse:

```
new <TipoObjetoNativo>([value]);
```

A excepción del objeto Math que no se puede definir utilizando la palabra reservada *new*.

Aunque es una manera totalmente válida de crear objetos, lo cierto es que **no es recomendable** utilizar la palabra *new* para **definir números, cadenas, booleanos**... Reduce la velocidad de ejecución, complica el código y puede producir resultados inesperados en las comparaciones de objetos. Se aconseja, por tanto, su definición mediante tipos primitivos.

CONSTRUCTOR *String*

El constructor *String* nos permite almacenar cadenas.

No es recomendable su uso, dado que existe un dato primitivo para su declaración.

Por ejemplo:

```
let x1 = new String(); //Objeto vacío. No utilizar
let y1 = "Carmelo Jaen"; //Utilizar
```

CONSTRUCTOR *Number*

El constructor *Number* nos permite almacenar números.

No es recomendable su uso, dado que existe un dato primitivo para su declaración.

Por ejemplo:

```
let x2 = new Number(); //Objeto vacío. No utilizar
let y2 = 3.14; //Utilizar
```

CONSTRUCTOR *Boolean*

El constructor *Boolean* nos permite almacenar booleanos.

No es recomendable su uso, dado que existe un dato primitivo para su declaración.

Por ejemplo:

```
let x3 = new Boolean(); //Objeto vacío. No utilizar
let y3 = true; //Utilizar
```

CONSTRUCTOR *Array*

El constructor *Array* nos permite almacenar arrays o vectores.

No es recomendable su uso, dado que existe una forma más simple para su declaración.

Por ejemplo:

```
var x4 = new Array(); //No utilizar
var y4 = []; //Utilizar
```

CONSTRUCTOR *RegExp*

El constructor *RegExp* nos permite almacenar expresiones regulares, las cuales nos permiten validar cadenas como el DNI o correos electrónicos, y comprobar así que están bien estructuradas.

No es recomendable su uso, dado que existe una forma más simple para su declaración.

Por ejemplo:

```
let x5 = new RegExp(); //No utilizar
let y5 = /<aquí va la expresión regular>/; //Utilizar
```

CONSTRUCTOR *Function*

El constructor *Function* nos permite almacenar funciones, ya estudiadas en el RA2.

No es recomendable su uso, dado que existe una forma más simple para su declaración como vimos en dicho RA.

Por ejemplo:

```
var x6 = new Function(); //No utilizar
var y6 = function (){}; //Utilizar
```

CONSTRUCTOR *Date*

El constructor *Date* nos permite almacenar valores de tipo temporal como fechas u horas.

Por ejemplo:

```
var x7 = new Date();
```

CONSTRUCTOR *Math*

No se puede declarar con *new* porque es un objeto global incluido en JavaScript.

Para usarlo basta con escribir “math.*metodoDeMath*”

Por ejemplo:

```
let x8 = Math.sqrt(a);
```

CONSTRUCTOR *Object*

El constructor *Object* nos permite almacenar objetos personalizados, como hemos estudiado anteriormente.

Por ejemplo:

```
let persona = new Object();
persona.nombre = “Carmelo”;
persona.apellido = “Jaen”;
persona.inicio = 2020;
```

O de una manera más simple (la recomendable):

```
let persona = {}; //Objeto vacío, habría que definir sus pares clave:valor.
```