UT 3 - Programación Avanzada. Objetos Predefinidos del Lenguaje

Marina Navarro Pina

Tipos de datos básicos

Tipo	Ejemplo	Descripción
Cadena	“Hola Mundo”	Caracteres dentro de comillas
Número	9.34	Números con punto para decimales
Boolean	true	true o false
Null	null	Sin valor
Function		Una función es referenciable como una variable
Object		Objetos como arrays o otros

Javascript tiene los tipos *string* y **String** o *number* y **Number**, así como **Boolean**.

Con **mayúsculas** son objetos especiales que pueden ser usados como primitivos, pero también ***tienen métodos***.

Typeof - Valores

Si queremos saber de qué tipo es una variable, podemos preguntar con **typeof()**:

```
let array_mix = [
 "abcdef", 2 , 2.1 , 2.9e3 , 2e-3 ,
 0o234 , 0x23AF , true , [1,2,3] , {'a': 1, 'b': 2}
];
for (let i=0;i<array_mix.length;i++) {
 console.log(typeof(array_mix[i]));
}
```

Además, soporta **valores especiales**:

- **undefined**: Indica que a la variable no se le ha asignado valor.
- **null**: Valor nulo, se comporta como un objeto vacío. (*typeof null* devuelve **object**)
- **NaN**: Not a Number, se obtiene cuando no se puede convertir a número el resultado de una operación. (ej. *'Hola'*2*, aunque *'2'*2* daría 4 ya que se convierte la cadena '2' al número 2)
- **Inifinity** y **-Infinity**: Indica que el resultado es demasiado grande o demasiado pequeño (ej. 1/0 o -1/0)

Casting de variables

Las variables pueden contener cualquier tipo de valor y, en las operaciones, Javascript realiza automáticamente las conversiones necesarias para, si es posible, realizar la operación.

- '4' / 2 devuelve 2 (convierte ‘4’ en 4 y realiza la operación)
- '23' - null devuelve 23 (hace 23 - 0)
- '23' - undefined devuelve NaN (no puede convertir undefined a nada así que no puede hacer la operación)
- '23' * true devuelve 23 (23 * 1)
- '23' * 'Hello' devuelve NaN (no puede convertir ‘Hello’)
- 23 + 'Hello' devuelve ‘23Hello’ (+ es el operador de **concatenación** así que convierte 23 a ‘23’ y los concatena)
- 23 + '23' devuelve 2323 (OJO, convierte 23 a ‘23’, no al revés)

Las funciones constructoras de los objetos Number, String, Boolean invocadas sin new, hacen una conversión de tipos.

String

Además del tipo de dato *string* JavaScript define el objeto **String** que ofrece un conjunto de propiedades muy útiles a la hora de trabajar con cadenas de texto.

Mediante la conversión de tipos se transforma un tipo primitivo en objeto **String** al intentar utilizar una de sus propiedades.

```
var a= 'Hola', // Tipo primitivo
    b= new String('Hola'); // Objeto String

console.log('Tipo dato a:', typeof a);
console.log('Tipo dato b:', typeof b);
console.log('a == b', a == b);
console.log('a === b', a === b);
console.log('Longitud a:', a.length);
```

String - Propiedades y Métodos

El operador **new** crea e inicializa un nuevo objeto.

A las propiedades de un objeto se accede mediante el operador punto . o los corchetes [] y nombre de la propiedad entre comillas (simples o dobles).

Algunos métodos y propiedades de las cadenas son:

Ejemplo 3

Más información:

- https://www.w3schools.com/jsref/jsref_obj_string.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

String - Template Literals

Desde ES2015 también podemos poner una cadena entre ` (acento grave) y en ese caso podemos poner dentro variables y expresiones que serán evaluadas al ponerlas dentro de ${}.

También se respetan los saltos de línea, tabuladores, etc. que haya dentro.

```
let edad=25;
console.log(`El usuario tiene: \n
                              ${edad} años`);
```

Number

Sólo hay 1 tipo de números, **no** existen enteros y decimales: *number* 23,12 se escribe 23.12

Podemos usar los **operadores aritméticos** (+, -, *, / y %), que se pueden usar junto con el **operador de asignación** = (+=, -=, *=, /= y %=). **Operadores unarios** (++, --) y **valores especiales** (Infinity, -Infinity).

Podemos forzar la conversión a número con la función **Number(valor)**.

Ejemplo: Number('23.12') devuelve 23.12

Number - Métodos y Funciones

Algunos **métodos** útiles, son:

- ✓ **.toFixed(num)**: redondea el número a los decimales indicados. *23.2376.toFixed(2) devuelve 23.24*
- ✓ **.toLocaleString()**: devuelve el número convertido a string en función del formato local. *23.76.toLocaleString() devuelve '23,76'* (convierte el *punto decimal en coma*)

Otras **funciones** útiles, son:

- **isNaN(valor)**: nos dice si el valor pasado es un número (*false*) o no (*true*)
- **isFinite(valor)**: devuelve true si el valor es finito (*no es Infinity ni -Infinity*).
- **parseInt(valor)**: convierte el valor pasado a un número entero. Siempre que comience por un número, la conversión se podrá hacer.
- **parseFloat(valor)**: como la anterior pero conserva los decimales.

Number - Métodos y Funciones

```
parseInt(3.65)      // Devuelve 3
parseInt('3.65')    // Devuelve 3
parseInt('3 manzanas')    // Devuelve 3, Number devolvería NaN
```

Más información:

- https://www.w3schools.com/jsref/jsref_obj_number.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number

Math - Funciones

JavaScript tiene un objeto incorporado *Math* que contiene una pequeña biblioteca de funciones matemáticas y constantes (**Math.PI**, **Math.E**, etc.) que nos ayudarán a realizar operaciones aritméticas, redondeos, etc.

Algunos de los métodos más importantes son:

- **Redondeo** :
  - floor(numero): redondea hacia abajo un número.
  - ceil(numero): redondea hacia arriba un número.
  - round(numero): redondea al entero mas cercano.
- **Operaciones matemáticas**:
  - abs(numero): devuelve el número en valor absoluto.
  - max / min (x,y): devuelve el mayor / menor de dos números x e y.
  - pow(x,y): devuelve x elevado a y.
  - random(): devuelve un número aleatorio con decimales entre 0 y 1.
  - sqrt(numero): devuelve la raíz cuadrada del número indicado.

Más información:

- https://www.w3schools.com/jsref/jsref_obj_math.asp
- https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math

Eval()

Es una función que recibe una cadena y la interpreta como *código Javascript*.

Se puede usar para calcular el resultado de expresiones numéricas.

Muy útil ya que podemos construir código dinámicamente.

```
let x=3;
let y=2;
let a=eval("2+3");
let b=eval("x*y");
eval("alert('a vale '+a+' b vale '+b)");
```

Date

Date es un objeto predefinido que nos permite trabajar con **fechas**. Para crear un objeto date se admiten múltiples formatos.

Los meses en Date se numeran del 0 al 11 (siendo el 0 Enero y el 11 Diciembre) mientras que los días si se numeran del 1 al 31.

Ejemplo 4

Más información:

- https://www.w3schools.com/js/js_dates.asp
- https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Date

Date - Funciones

Algunos de los métodos más importantes son:

- Métodos **set/get** : son métodos que permite cambiar el valor de alguna parte de la fecha (*set*) o de obtener el valor de alguna parte de la fecha (*get*).

Ejemplos: setMonth(mes), getMonth(); setDate(dia), getDate(), setHours(hora, minuto,segundo), getHours(), etc.

- **getDay()**: devuelve el día de la semana, (el día del mes es con *getDate*). Están numerados del 0 al 6, siendo el 0 domingo y el 6 sábado.
- **toDateString()**: convierte la fecha del objeto a una cadena en objeto fecha.
- **toGMTString()**: convierte la fecha del objeto en una cadena con formato de fecha GMT.
- **toUTCString()**: convierte la fecha del objeto en una cadena con formato de fecha UTC.

Boolean

Los valores **booleanos** son *true* y *false*.

Para convertir algo a booleano se usar **Boolean(valor)** aunque también puede hacerse con la doble negación (**!!**).

Cualquier valor se evaluará a true excepto **0, NaN, null, undefined o una cadena vacía** (**''**) que se evaluarán a false.

Boolean - Operadores

Los operadores lógicos son:

- ✓ **!** (negación), **&&** (and), **||** (or).
- ✓ Para comparar valores tenemos **==** y **===** (devuelve true si son *igual valor* y *mismo tipo*).

Como Javascript hace conversiones de tipos automáticas conviene usar la === (*Ejemplo 5*).

- ✓ 2 operadores de diferente: **!=** y **!==** que se comportan igual que == y ===.
- ✓ Operadores relacionales son **>**, **>=**, **<**, **<=**. Si se compara un número y una cadena, ésta se convierte a número y no al revés (23 > '5' devuelve *true*, aunque '23' > '5' devuelve *false*).

Más información:

- https://www.w3schools.com/js/js_booleans.asp
- https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Boolean

Window-BOM

El objeto *Window* solo está presente al trabajar con Javascript en **navegadores**, no estando presente en entornos como NodeJS.

Posee propiedades y controla elementos relacionado con lo que ocurre en la "*ventana*" del navegador.

Los métodos ***alert***, ***prompt***, ***confirm*** forman parte de este objeto.

Más información:

- https://www.w3schools.com/js/js_window.asp
- https://developer.mozilla.org/es/docs/Web/API/Window

Window - Timers - Métodos

Otros métodos importantes son:

- **setTimeout(cadenaFuncion,tiempo)**: este método ejecuta **la llamada a la función proporcionada** por la cadena (se puede construir una cadena que lleve parámetros) y la ejecuta **pasados los milisegundos** que hay en la variable tiempo. Devuelve un identificador del "*setTimeout*" que nos servirá para referenciar lo si deseamos cancelarlo. SetTimeout solo ejecuta la orden una vez.
- **setInterval(cadenaFunción, tiempo)**: exactamente igual que setTimeout, con la salvedad de que no se ejecuta una vez, sino que **se repite cíclicamente** cada vez que pasa el tiempo proporcionado.
- **clearTimeout / clearInterval (id)**: se le pasa el identificador del timeout/interval y lo anula.

Ejemplo 6

Manejo de errores

Si sucede un error en nuestro código el programa dejará de ejecutarse por lo que el usuario *tendrá la sensación de que no hace nada* (el error sólo se muestra en la consola y el usuario *no suele abrirla nunca*). Para evitarlo es crucial capturar los posibles errores de nuestro código antes de que se produzcan.

En JavaScript el manejo de errores se realiza con las sentencias: (*Ver ejemplo en apuntes de referencia*)

```
try {
    ...
}
catch(error) {
    ...
}
```

Más información:
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/try...catch