JavaScript 02 – Arrays (I)

Introducción

Objetos JavaScript

Los objetos son uno de los tipos de datos de JavaScript.

Los objetos se utilizan para almacenar colecciones de clave/valor (nombre/valor).

Se puede decit que un objeto JavaScript es una colección de valores con nombre.

En Javascript podemos definir cualquier variable como un objeto declarándola con **new** (NO se recomienda) o creando un *literal object* (usando notación **JSON**).

El siguiente ejemplo crea un objeto JavaScript con tres propiedades clave/valor.

Creando con *new* (no recomendado):

```
let alumno = new Object()
alumno.nombre = 'Carlos'     // se crea la propiedad 'nombre' y se le asigna un valor
alumno['apellidos'] = 'Pérez Ortiz'    // se crea la propiedad 'apellidos'
alumno.edad = 19
```

Creando un *literal object* (es la forma **recomendada**). El ejemplo anterior quedaría:

```
let alumno = {
    nombre: 'Carlos',
    apellidos: 'Pérez Ortiz',
```

```
    edad: 19,
};
```

Propiedades de un objeto

Podemos acceder a las propiedades con `.` (punto) o `[ ]`:

```
console.log(alumno.nombre)       // imprime 'Carlos'
console.log(alumno['nombre'])    // imprime 'Carlos'
let prop = 'nombre'
console.log(alumno[prop])        // imprime 'Carlos'
```

Si intentamos acceder a propiedades que no existen, no se produce un error: se devuelve *undefined*:

```
console.log(alumno.ciclo)      // muestra undefined
```

Sin embargo, se genera un error si intentamos acceder a propiedades de algo que no es un objeto:

```
console.log(alumno.ciclo)          // muestra undefined
console.log(alumno.ciclo.descrip)      // se genera un ERROR
```

Para evitar ese error antes había que comprobar que existían las propiedades previas:

```
console.log(alumno.ciclo && alumno.ciclo.descrip)
// si alumno.ciclo es un objeto muestra el valor de
// alumno.ciclo.descrip. Si no muestra undefined
```

Con ES2020 (ES11) se ha incluido el operador **?.** para evitar tener que comprobar esto nosotros:

```
console.log(alumno.ciclo?.descrip)
// si alumno.ciclo es un objeto muestra el valor de
// alumno.ciclo.descrip. Si no muestra undefined
```

Podremos recorrer las propiedades de un objecto con `for..in`:

```
for (let prop in alumno) {
    console.log(prop + ': ' + alumno[prop])
}
```

Nota: En este caso no se puede aplicar el bloque de código `for..of` porque un objeto de este tipo no es iterable, sí son iterables los arrays y strings

Si el valor de una propiedad es el valor de una variable que se llama como la propiedad, no es necesario ponerlo:

```
let nombre = 'Carlos'

let alumno = {
    nombre,           // es equivalente a nombre: nombre
    apellidos: 'Pérez Ortiz',
    ...
```

Métodos de un objeto

Llamamos **método de un objeto** cuando una **propiedad** de ese objeto es una **función**:

```
alumno.getInfo = function() {
    return 'El alumno ' + this.nombre + ' ' + this.apellidos + ' tiene ' + this.edad + ' años'
}
```

Nota: La palabra clave **this** dentro de un método hace referencia al objeto en cuestión al que se aplica dicho método.

Y para llamarlo se hace como con cualquier otra propiedad:

```
console.log(alumno.getInfo())    // imprime 'El alumno Carlos Pérez Ortíz tiene 19 años'
```

Ojo!! No se puede utilizar *this* con sintaxis *arrow function*. Está eplicado en el documento **Anexo – Uso de this en contexto**

Algunos métodos y propiedades genéricas de objetos de JavaScript

- constructor Returns the function that created an object's prototype
- keys() Returns an Array Iterator object with the keys of an object
- prototype Let you to add properties and methods to JavaScript objects
- toString() Converts an object to a string and returns the result. Este método se usa sobrescrito.
- valueOf() Returns the primitive value of an object

Prueba las siguientes sentencias:

```
console.log(alumno.constructor);
console.log(Object.keys(alumno));
console.log(Object.values(alumno));
console.log(Object.entries(alumno));
console.log(alumno.toString());
alumno.edasd=33;
console.log(alumno.valueOf());
console.log(Object.prototype);
console.log(Object.constructor);
```

EJERCICIO: Crea un objeto llamado **tvSamsung** con las propiedades **nombre** (“TV Samsung 42”), **categoría** (“Televisores”), **unidades** (4), **precio** (345.95) y con un método llamado **importe** que devuelve el valor total de las unidades (nº de unidades * precio).

Los arrays en JavaScript

Los arrays, también llamados arreglos, vectores, matrices, listas…, son un tipo de objeto que no tiene tamaño fijo, podemos añadirle elementos en cualquier momento.

Para hacer referencia a (referenciar) los elementos, se hace con un índice numérico. A diferencia de los objetos que se referencian con un nombre.

Podemos crearlos como instancias del objeto Array (No se recomienda):

```
let a = new Array()         // a = []
let b = new Array(2,4,6)    // b = [2, 4, 6]
```

Lo recomendable es crearlos usando notación JSON:

```
let a = []
```

```
let b = [2,4,6]
```

Sus elementos pueden ser de cualquier tipo, incluso podemos tener elementos de tipos distintos en un mismo array.

Si no está definido un elemento, su valor será *undefined*. Ej.:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
console.log(a[0])  // imprime 'Lunes'
console.log(a[4])  // imprime 6
a[7] = 'Juan'        // ahora a = ['Lunes', 'Martes', 2, 4, 6, , , 'Juan']
console.log(a[7])  // imprime 'Juan'
console.log(a[6])  // imprime undefined
```

Acceder a un elemento de un array que no existe no provoca un error (devuelve *undefined*), pero sí lo provoca acceder a un elemento de algo que no es un array. Con ES2020 (ES11) se ha incluido el operador **?.** para evitar tener que comprobar nosotros que sea un array:

```
console.log(alumnos?.[0])
// si alumnos es un array muestra el valor de su primer
// elemento y si no muestra undefined pero no lanza un error
```

Arrays de objetos

Es habitual almacenar datos en arrays en forma de objetos, por ejemplo:

```
let alumnos = [
    {
      id: 1,
      name: 'Carlos Pérez',
      course: '2DAW',
      age: 21
    },
    {
      id: 2,
      name: 'Ana García',
      course: '2DAW',
      age: 23
    },
  ];
```

Operaciones con Arrays

Los arrays tienen las mismas propiedades y métodos que los objetos, y muchos más que son propios de los arrays.

Vamos a ver los principales métodos y propiedades de los arrays.

length

Esta propiedad devuelve la longitud de un array:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
```

```
console.log(a.length)  // imprime 5
```

Podemos reducir el tamaño de un array cambiando esta propiedad:

```
a.length = 3  // ahora a = ['Lunes', 'Martes', 2]
```

Añadir elementos

Podemos añadir elementos al final de un array con el método `push`, o al principio con `unshift`:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
a.push('Juan')   // ahora a = ['Lunes', 'Martes', 2, 4, 6, 'Juan']
a.unshift(7)     // ahora a = [7, 'Lunes', 'Martes', 2, 4, 6, 'Juan']
```

Eliminar elementos

Podemos borrar el elemento del final de un array con `pop`, o del del principio con `shift`. Ambos métodos devuelven el elemento que hemos borrado:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
let ultimo = a.pop()          // ahora a = ['Lunes', 'Martes', 2, 4] y ultimo = 6
let primero = a.shift()       // ahora a = ['Martes', 2, 4] y primero = 'Lunes'
```

splice

Permite eliminar elementos de cualquier posición del array y/o insertar otros en su lugar. Devuelve un array con los elementos eliminados. Sintaxis:

```
Array.splice(posicion, num. de elementos a eliminar, 1º elemento a insertar, 2º elemento a insertar, 3º...)
```

Ejemplo:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
let borrado = a.splice(1, 3)       // ahora a = ['Lunes', 6] y borrado = ['Martes', 2, 4]
a = ['Lunes', 'Martes', 2, 4, 6]
borrado = a.splice(1, 0, 45, 56)   // ahora a = ['Lunes', 45, 56, 'Martes', 2, 4, 6] y borrado = []
a = ['Lunes', 'Martes', 2, 4, 6]
borrado = a.splice(1, 3, 45, 56)   // ahora a = ['Lunes', 45, 56, 6] y borrado = ['Martes', 2, 4]
```

EJERCICIO: Guarda en un array la lista de la compra con Peras, Manzanas, Kiwis, Plátanos y Mandarinas. Haz lo siguiente con splice:

- Elimina las manzanas (debe quedar Peras, Kiwis, Plátanos y Mandarinas)

- Añade detrás de los Plátanos, Naranjas y Sandía. (Debe quedar: Peras, Kiwis, Plátanos, Naranjas, Sandía y Mandarinas)

- Quita los Kiwis y pon en su lugar Cerezas y Nísperos. (Debe quedar: Peras, Cerezas, Nísperos, Plátanos, Naranjas, Sandía y Mandarinas)

slice

Devuelve un subarray con los elementos indicados, pero sin modificar el array original. Sería como hacer un `substr` pero de un array en vez de una cadena.

Sintaxis: `Array.slice(posicion, num. de elementos a devolver)`

Ejemplo:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
let subArray = a.slice(1, 3) // ahora a=['Lunes', 'Martes', 2, 4, 6] y subArray=['Martes', 2, 4]
```

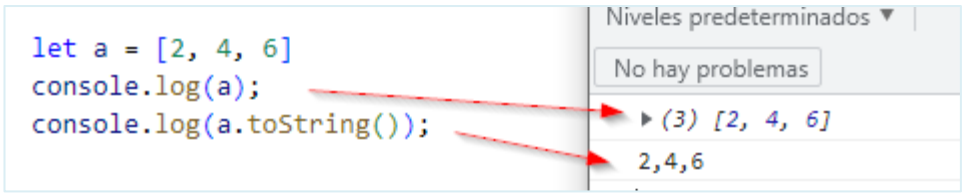
Es muy útil para hacer una copia de un array:

```
let a = [2, 4, 6]
let copiaDeA = a.slice() // ahora ambos arrays contienen lo mismo pero son diferentes arrays
```

Arrays y Strings

Cada objeto, y los arrays son un tipo de objeto, tienen definido el método **`.toString()`** que lo convierte en una cadena. Este método es llamado automáticamente cuando, por ejemplo, queremos mostrar un array por la consola.

En el caso de los arrays esta función devuelve una cadena con los elementos del array separados por coma.



Además, podemos convertir los elementos de un array a una cadena con **`.join()`** especificando el carácter separador de los elementos. Ej.:

```
let a = ['Lunes', 'Martes', 2, 4, 6]
let cadena = a.join('-')       // cadena = 'Lunes-Martes-2-4-6'
```

El método **`.join()`** es el contrario del **`.split()`** que convierte una cadena en un array. Ej.:

```
let notas = '5-3.9-6-9.75-7.5-3'
let arrayNotas = notas.split('-')        // arrayNotas = [5, 3.9, 6, 9.75, 7.5, 3]
let cadena = 'Que tal estás'
let arrayPalabras = cadena.split(' ')    // arrayPalabras = ['Que`, 'tal', 'estás']
let arrayLetras = cadena.split('')       // arrayLetras = ['Q','u','e`,' ','t',a',l','
','e',s',t',á',s']
```