MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO DE WEB ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Estructura de datos. El array

06

ÍNDICE

OBJETIVOS

Conocer los principios de orientación a objetos en JavaScript.

Saber identificar propiedades y métodos.

Saber utilizar propiedades y métodos.

Conocer el objeto array y sus métodos.

Conocer mecanismo de manipulación de arrays.

/ 1. Introducción y contextualización práctica

JavaScript es un lenguaje orientado a objetos en el cual se intentan modelar las soluciones atendiendo a elementos que existen en la realidad. Al igual que muchos lenguajes de programación, JavaScript ofrece la posibilidad tanto de crear objetos como de utilizar los propios del lenguaje. En este sentido, todos los objetos de los que disponemos, así como los que podemos crear, van a tener propiedades. Dependiendo del lenguaje de programación utilizado, la creación de objetos y el uso de estos puede variar, pues el aspecto sintáctico puede ser diferente en cada lenguaje.

Como norma general, los lenguajes de programación ofrecen la posibilidad de trabajar con objetos del propio lenguaje. Algunos de estos objetos pueden facilitar significativamente el desarrollo de soluciones como, por ejemplo, la utilización de colecciones que permitan almacenar conjuntos de datos. Existen diferentes tipos de colecciones que se pueden categorizar por diferentes criterios: si admiten duplicados o no, si mantienen un orden en los elementos, según el tiempo de respuesta para localizar un elemento, etc.

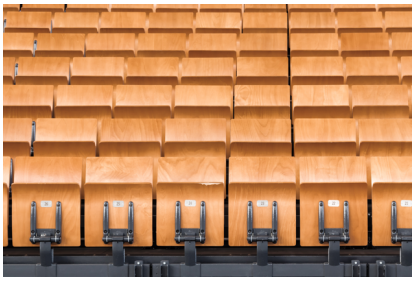

Fig.1. Elementos de un array con el mismo tipo

Escucha el siguiente audio que describe el caso práctico que iremos resolviendo a lo largo de la unidad:

Audio Intro. Ordenación y objetos en JavaScript
https://bit.ly/2Yx2M48

/ 2. Orientación a objetos en JavaScript

La orientación a objetos en JavaScript es más sencilla que en otros lenguajes orientados a objetos. Aunque JavaScript no está basado en el concepto de clase, sí es cierto que reúne algunas características de la orientación a objetos. Tradicionalmente, esta orientación se vincula con la utilización de clases; sin embargo, como hemos dicho anteriormente, JavaScript no se basa en este principio, sino que se centra en el prototipado. Un lenguaje de este estilo se basa en plantillas que permiten definir las propiedades de un nuevo objeto.

Al igual que otros lenguajes, JavaScript permite el acceso a sus atributos a través de la notación punto. Utilizando el prototipado también se permite crear funciones asociadas a los objetos.

Con el objetivo de ser más similar a otros lenguajes de programación orientado a objetos, en 2015 JavaScript incorporó la palabra reservada class. No obstante, no ha abandonado su principio de prototipado.

JavaScript también soporta la herencia entre objetos. Es decir, se puede ampliar la funcionalidad de un objeto heredando del mismo. En JavaScript, mediante el uso del puntero this, se pueden crear nuevos objetos de manera sencilla. Existen diferentes formas de crear objetos:

- Definiendo un constructor. El constructor es la función encargada de crear un objeto determinado.
- Utilizando el objeto genérico «Object».

De manera coloquial, podemos decir que un objeto es una variable que puede contener muchos valores, además de métodos asociados. Por ejemplo:

```
var person = {name:"Fran", surname:"Pérez"}
```

Código 1. Declaración de un objeto con dos atributos

En este ejemplo se muestra la declaración de un objeto que contiene dos atributos: name y surname.

Vídeo 1. Creando objetos en JavaScript
https://bit.ly/3hoTbVA

/ 3. Propiedades y métodos

JavaScript implementa las propiedades siguiendo un esquema clave-valor. En este sentido, sigue una anotación similar a las tablas hash de Java o los diccionarios de Phyton, entre otros. Los métodos del objeto son aquellas tareas que puede realizar el objeto, es decir, sus acciones. En JavaScript estos métodos se definen como propiedades que apuntan a funciones.

La forma más sencilla de crear un objeto JavaScript es de manera literal, ya que de esta forma somos capaces de definir y crearlo en una única sentencia. La declaración literal de un objeto no es más que una lista de parejas de clave-valor. Por ejemplo:

```
//creación de un objecto con 4 propiedades
var person = {name:"Fran", surname:"Pérez", age:36, state:"Spain"}
```

Código 2. Declaración de un objeto de forma literal

También es posible crear el mismo objeto con el operador new. Por ejemplo:

```
var person = new Object();
person.name = "Fran";
person.surname = "Pérez";
person.age = 36;
person.state = "Spain";
```

Código 3. Declaración de un objeto con operador new

Estas dos alternativas de creación de un objeto son prácticamente iguales. No existe ninguna necesidad de utilizar el operador new. Por ello, en este escenario es recomendable utilizar la forma literal. En JavaScript los objetos pueden cambiar, dado que se acceden por referencia y no por valor. La parte más importante de un objeto JavaScript son sus propiedades y existen diferentes formas de acceder a ellas:

```
//Opción 1
person.name;
//Opción 2
person["name"];
//Opción 3
var name = "name";
person[name];
```

Código 4. Formas de acceso a las propiedades de un objeto

3.1. Manejo de propiedades en JavaScript

Las propiedades de un objeto pueden modificarse, no solo en valor, sino también en existencia. Se puede añadir una propiedad simplemente definiéndola y se puede borrar una propiedad utilizando la palabra reservada delete. Borrando una propiedad con esta palabra clave hay que tener en cuenta lo siguiente:

- Se borra tanto el valor como la existencia la propiedad.
- Para poder volver a acceder a esa propiedad, es necesario volver a definirla.
- Este operador solo tiene efectos sobre propiedades de los objetos.

- No debería usarse sobre objetos predefinidos del lenguaje.

Las propiedades poseen algunos atributos, por ejemplo, su valor. Existen otros atributos que se pueden configurar, entre otros: escritura, enumeración, lectura, etc.

3.2. Métodos de los objetos

Un método definido en un objeto es una funcionalidad que este puede llevar acabo. Para acceder a los métodos del objeto se utiliza la notación punto. Véase el siguiente ejemplo:

```
var persona = {
    name:"Fran",
    surname:"Pérez",
    age:36,
    state:"Spain",
    fullInfo: function(){
            return this.name + " "+ surname + " " + age+ " "+ state;
    }
};
```

Código 5. Métodos de los objetos

Audio 1. JavaScript: modelando objetos
https://bit.ly/2Uli4lq

En el ejemplo anterior se crea el objeto persona, que tiene cinco propiedades y una de ellas es un método: fullInfo, que ofrece la funcionalidad de mostrar toda la información del objeto.

/ 4. Caso práctico 1: "Análisis de código"

Planteamiento. Como experto en JavaScript nos piden realizar el análisis del siguiente código que ha fallado en ejecución y que está relacionado con la orientación a objetos:

```
var student = {
name : "Jose",
clazz : "web",
id : 12,
info: return(){this.name+" "+ this.clazz}};
console.log("Nombre del usuario: "+ student[name]);
```

Código 6. Análisis de código con fallo

Nudo. ¿Qué crees que puede estar pasando? ¿Crees que existe algún error?

Desenlace. Como hemos visto en el apartado sobre la creación de objetos en JavaScript, el objeto definido anteriormente parece estar bien creado y JavaScript no dará ningún error hasta que se visualice el contenido en el navegador de la web. Si nos fijamos en la declaración de las claves y los valores, todo es correcto. Sin embargo, a la hora de crear una función que devuelva la información del usuario, se ha cometido un error sintáctico, pues es necesario incluir la palabra reservada function en la clave info.

/ 5. Arrays

JavaScript permite utilizar elementos de colección que se caracterizan por poder almacenar un número determinado de valores. Además, es posible consultar modificar y borrar estos valores de la colección.

El array o lista es un objeto del tipo colección que nos proporciona JavaScript. Un array no deja de ser una lista que encapsula diferentes funcionalidades y le facilita la vida al programador.

En JavaScript las listas no tienen un tamaño determinado, pues pueden cambiar según se necesite. Las listas pueden ser multidimensionales, es decir, pueden formar matrices.

5.1. Creación

En este apartado estudiaremos los diferentes mecanismos que JavaScript ofrece para crear una lista. La forma más sencilla de crear un array en JavaScript es como un literal:

```
var pets = [“dog”, “cat”, “rabbit”];
```

Código 7. Creación de array como literal

En el ejemplo anterior, se crea un literal con los valores de diferentes mascotas. Los corchetes y la separación con comas nos indican que se está declarando un array. Al crear la lista de esta forma, se ignoran los saltos de línea y los espacios en blanco.

Otra forma de crear un array en JavaScript es mediante el operador new. Con este operador no solo se pueden crear listas, sino que este también permite la creación de todos los objetos de lenguaje. Desde el punto de vista de la creación de arrays, no es recomendable utilizar el operador new, pues puede complicar el código y, en ocasiones, dar lugar a posibles errores. Un ejemplo de ello es cuando creamos un array con dos elementos o con uno solo. El segundo caso puede conducir a error, dado que se está indicando el tamaño del array, es decir, el número de elementos que como máximo puede contener.

```
va var items = new Array(1, 6);  // Crea un array con dos elementos (1 y 6)
var items = new Array(6);  // Crea un array con 6 elementos
```

Código 8. Creación de array con new

5.2. Acceso

Desde el punto de vista del programador, además de crear listas, necesitamos poder acceder a ellas para manipularlas. JavaScript ofrece la posibilidad de acceder a los elementos de una lista con la siguiente sintaxis:

```
var pets = new Array(“dog”, “cat”, “rabbit”);
var pet = pets[0];
console.log (pet);
```

Código 9. Acceso a una lista

En este ejemplo se declara un array de mascotas y se accede al elemento cero, el cual se asigna a la variable pet. Por último, el valor de la variable se muestra por consola. Nótese que, en JavaScript, los arrays comienzan en el índice cero.

Para modificar el contenido de un array, necesitamos acceder a la posición correspondiente y utilizar el operador «asignación».

Un array también puede almacenar objetos de la misma forma que se emplea con tipos básicos.

5.3. Propiedades de los arrays

Para poder trabajar correctamente con un array es indispensable consultar su longitud. JavaScript ofrece esta posibilidad a través del atributo length, mediante el cual podemos consultar tanto el primer elemento como el último. Sabiendo que los arrays en JavaScript comienzan en la posición cero, para acceder al último elemento es necesario conocer la longitud de la lista. Por ello, el último elemento es la longitud-1.

```
var people = ["Jose", "Antonio", "Luis", "Olga"];
people.length;   //La longitud es 4
```

Código 10. Longitud del array

5.4. Iteración sobre arrays

Una de las posibilidades que nos ofrecen los arrays es poder recorrer todos sus elementos. Conociendo el primer elemento de un array en la posición cero y sabiendo cómo acceder a la última, podríamos utilizar un bucle tipo for para acceder a todos los elementos.

Otra posibilidad sería utilizar el método forEach, que nos ofrece JavaScript para recorrer las listas.

5.5. Ejemplo de iteración sobre arrays

```
var people, text, fLen, i;
people = ["Pepe", "Luis", "Antonio", "Nela"];
len = people.length;
text = "<ul>";
for (i = 0; i < len; i++) {
  text += "<li>" + people[i] + "</li>";
}
text += "</ul>";
```

Código 11: Bucle for tradicional

```
var people, text;
people = ["Pepe", "Luis", "Antonio", "Nela"];
text = "<ul>";
fruits.forEach(buildList);
text += "</ul>";

function buildList(value) {
  text += "<li>" + value + "</li>";
}
```

Código 12: Bucle forEach

5.6. Insertar elementos en un array

Una de las operaciones más comunes que se puede realizar sobre una lista es añadir elementos. JavaScript permite realizar esta acción utilizando la función push() sobre la lista en cuestión.

También es posible agregar elementos a un array utilizando la propiedad length. Asignando un nuevo valor a esta posición se crea un nuevo elemento y la longitud aumenta. Extrapolando este funcionamiento, se podría utilizar un número entero para añadir un elemento en cualquier índice. El mayor problema de esta alternativa es que la lista puede quedar fragmentada, apareciendo lo que se conoce como «agujeros».

```
var people = [“Pepe”, “Luis”, “Antonio”, “Nela”];
people.push(“Fran”); //Nuevo elemento con método push()
people[people.length-1] = “Olga”; //Nuevo elemento a partir de
la longitud del array
```

Código 13: Inserción de nuevos elementos

5.7. Eliminar elementos de un array

La operación contraria de añadir un elemento en una lista es justamente extraer un elemento de una lista. JavaScript proporciona métodos para poder sacar elementos de una lista de manera sencilla. En esencia, se trata la lista como si fuese una pila y se extrae siempre el elemento que se encuentre en la cima de la pila (último elemento insertado). Esta operación se implementa con el método pop(). Si queremos sacar elementos por el principio de la lista, tenemos que usar el método shift(). Nótese que, con este método, el resto de los índices del array deben ser recalculados.

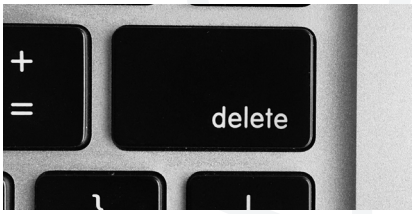

Fig.2. Borrar elementos de una lista

```
var people = [“Pepe”, “Luis”, “Antonio”, “Nela”];
people.push(“Fran”); //Nuevo elemento con método push()
var pop = people.pop(); //Sacamos el último elemento insertado
var shift = people.shift(); //Sacamos el primer elemento insertado
```

Código 14: Uso de pop y shift

5.8. Porciones de arrays

En el caso de que contemos con diferentes listas y necesitemos tenerlas todas en una, JavaScript nos permite realizarlo utilizando el método concat().

```
var people1 = [“Pepe”, “Luis”];
var people2 = [“Antonio”, “Nela”];
var guys = people1.concat(people2); //La concatenación genera un nuevo array sin modificar los originales
```

Código 15: Uso de concat()

Al trabajar con arrays puede surgir la necesidad de unir parte de ellos. JavaScript proporciona el método splice() para poder unir a una lista ya existente otro contenido a partir de una posición determinada. En ocasiones, puede que sea necesario quedarse con una parte de una lista. A través del método slice() de JavaScript podemos indicar qué elementos necesitamos seleccionar.

5.9. Ordenación de arrays

JavaScript proporciona dos métodos para poder ordenar las listas. Por un lado, tenemos el método sort(), que permite ordenar una lista alfabéticamente en orden creciente. Por otro lado, tenemos el método reverse(), que permite dar la vuelta al contenido de una lista. Utilizando ambos métodos, podemos ordenar de manera ascendente o descendente el contenido de una lista.

```
var people = ["Pepe", "Luis","Antonio", "Nela"];
people.sort(); //Permite ordenar alfabéticamente los elementos del array
people.reverse() //Permite dar la vuelta al array ordenado
```

Código 16. Ordenación de listas

Estos métodos de ordenación funcionan correctamente para valores alfanuméricos. Sin embargo, cuando queremos ordenar valores numéricos en formato cadena, el resultado puede no ser el esperado.

En JavaScript es posible personalizar la función de comparación con el fin de cambiar el comportamiento de los métodos de ordenación. Cuando definimos una función de comparación, esta puede devolver valores positivos, negativos o cero, dependiendo de los parámetros utilizados en la invocación.

Gracias a este comportamiento con las funciones de comparación, podemos implementar el método de ordenación que estimemos necesario para nuestra solución.

```
var people = ["Pepe", "Luis","Antonio", "Nela"];
people.sort(function(a,b){return Math.floor(0.5 - Math.random())});
```

Código 17. Ejemplo de una ordenación aleatoria

Fig. 3.Ordenación de elementos

Vídeo 2. Arrays en JavaScript
https://bit.ly/2UOGIki

/ 6. Caso práctico 2: "Mejora de rendimiento"

Planteamiento. En nuestra empresa surge la necesidad de mejorar el rendimiento de los algoritmos de ordenación de JavaScript, pues el tiempo empleado en los algoritmos, por defecto, es muy elevado. En este contexto, nuestro cliente nos pide ayuda.

Nudo. ¿Cómo podríamos ayudarle? ¿Es posible cambiar el comportamiento de los algoritmos de ordenación?

Desenlace. Como hemos visto en el apartado sobre la ordenación en JavaScript, el lenguaje nos permite modificar los algoritmos que se tengan por defecto implementados e integrar nuestras propias funciones de ordenación. Para ello podemos definir una función en nuestro código y pasarlas por parámetro a la función sort() de JavaScript. Tenemos que recordar que la función que implementemos solo puede devolver tres posibles valores: <0, 0 o >0.

Lo vemos en el siguiente ejemplo:

```
var lista = [
  { name: “Jose”, value: 30 },
  { name: “Juan”, value: 25 },
  { name: “Fran”, value: 24 },
  { name: “Lucas”, value: 50 }
];
items.sort(function (a, b) {
   if (a.name > b.name)
     return 1;
   if (a.name < b.name)
     return -1;
   return 0;
});
```

Código 18. Resolución caso práctico 2

/ 7. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema hemos presentado las principales características de JavaScript para poder crear objetos y utilizar los que ya existen en el lenguaje. Hemos hecho un recorrido por las diferentes formas que ofrece JavaScript para crear un objeto, observando que, aunque JavaScript sea un lenguaje orientado, sigue una filosofía diferente a la de otros lenguajes de programación, lo cual nos permite crear objetos de una manera sencilla a través de parejas clave-valor. La definición de los métodos se realiza de la misma forma, es decir, mediante parejas clave-valor.

También hemos visto que uno de los objetos más destacados de JavaScript es el array. Este objeto permite almacenar una colección de elementos sin ningún orden preestablecido. Gracias a los métodos que incorpora JavaScript en la clase array, este objeto se convierte en uno de los más potentes del lenguaje. A lo largo del tema hemos aprendido a cómo crear, modificar y borrar elementos de un array.

Resolución del caso práctico de la unidad

Como hemos visto en este tema, JavaScript permite la creación de objetos por parte del usuario. Teniendo en cuenta la situación que se nos plantea, al venir los datos desordenados, es necesario crear una estructura de objetos que nos permita ordenarlos. Como se dice en el planteamiento, es necesario crear una estructura de objeto que permita almacenar, al menos, el nombre y la fecha de contratación del trabajador. A continuación, cada entidad que provenga del servidor creará un objeto trabajador y lo insertará en una lista. Para poder realizar la ordenación tenemos que definir un método que acepte un empleado y compare su fecha de contratación:

```
var fromHost = JSON.stringfy(response);
var list = [];
fromHost.forEach(item){
  var employee = {
    name:item.name,
    hire_date:item.date,
    salary:item.salary
  };
  list.push(employee);
}
```

Código 19. Resolución caso práctico inicial

/ 8. Bibliografía

Flanagan, D. (2020). *JavaScript: The Definitive Guide. (7th edition).* Sebastopol: O'Reilly Media, Inc.

Haverbeke, M. (2019). *Eloquent JavaScript: a modern introduction to programming.* San Francisco: No Starch Press.