Tema 6

Estructuras de datos. El array

MEDAC
Instituto Oficial de Formación Profesional

Estructura de datos: el array

Aprendiendo Javascript

- Arrays
- Arrays avanzados (matrices)

MEDAC

Arrays

- En los lenguajes de programación existen estructuras de datos especiales que nos sirven para guardar información más compleja que la que podemos almacenar en simples variables

- La estructura más típica en todos los lenguajes de programación es el Array. Un array es “una variable” en la que vamos a poder introducir una colección de valores, en lugar de solamente un valor como ocurre con las variables normales.

- Como símil podríamos decir que un array se asemeja a una cajonera. Ese mueble es todo un bloque, el cual tiene varios cajones donde podemos almacenar y sacar cosas. El array sería ese mueble y cada “cajón” del array funcionaría exactamente igual que una variable.

Definición de Arrays

- La sentencia para crear un array en JavaScritpt es:

```
var mueble = new Array(5)
```

Con esto se crea un array llamado mueble el cual va a tener 5 cajones. Cada cajón del array va a estar numerado desde el 0 hasta el tamaño indicado-1, es decir, como en la mayoría de lenguajes de programación, en Javascript los cajones de un array empiezan a numerarse desde el cero.

Sabiendo esto, el array mueble va a tener cinco cajones que son: el 0, el 1,el 2,el 3 y el 4.

Uso de Arrays

- Para usar un cajón del array es necesario referenciarlo por su número. Así pues, para introducir valores en los cajones de un array, tan solo hay que indicar que cajón es el que voy a “abrir” y meter en él lo que queramos (como ya hemos visto en el tema de variables):

```
mueble[0] = 290
mueble[2] = 127
```

- Como vemos, se debe indicar el nombre del array y, entre corchetes, el índice del cajón donde queríamos guardar el dato.

MEDAC

Uso de Arrays

- Para sacar datos de un array lo hacemos igual: poniendo entre corchetes el índice del cajón que queremos abrir.

```
let miArray = new Array(3)

miArray[0] = 155
miArray[1] = 4
miArray[2] = 499

console.log(miArray[0]);
console.log(miArray[1]);
console.log(miArray[2]);
```

MEDAC

Importante: Definición de arrays

- Hoy en día existe una forma mucho más rápida de definir los arrays en Javascript: la notación JSON que veremos más adelante

Tipos de datos en arrays

- En las casillas de los arrays podemos guardar datos de cualquier tipo de los que conocemos

Ejemplo: array donde introducimos datos de tipo cadena

```
miArray[0] = "Hola"
miArray[1] = "a"
miArray[2] = "todos"
```

MEDAC

Tipos de datos en arrays

A diferencia de lenguajes fuertemente tipados (como JAVA o C++), en Javascript podemos guardar distintos tipos de datos en los cajones de un mismo array. Es decir, podemos introducir números en unos cajones, textos en otros, booleanos o cualquier otra cosa que deseemos.

```
miArray[0] = "Que interesante es la clase"
miArray[1] = 1275
miArray[2] = 0.78
miArray[3] = true
```

Longitud de un array: length

- Todos los arrays en javascript, aparte de almacenar el valor de cada una de sus posiciones también almacenan el número de cajones totales que tienen. Para ello utilizan una propiedad del array, la propiedad length.

- Es una variable que se crea automáticamente al crear el array y que almacena un número igual al número de cajones totales del array.

- Esta variable solo puede consultarse (no puede cambiarse). Para ello se escribe el nombre del array del que queremos saber el número de posiciones que tiene, sin corchetes ni paréntesis, seguido de un punto y la palabra length.

Longitud de un array: length

```
var miArray = new Array(3)

miArray[0] = 155
miArray[1] = 499
miArray[2] = 65

console.log("Longitud del array: " + miArray.length);
```

Inicialización de arrays

- Los arrays tienen una manera de inicializar sus valores a la vez que los declaramos, así podemos realizar de una forma más rápida el proceso de introducir valores en cada una de las posiciones del array.

- Se coloca entre los paréntesis los valores con los que deseamos rellenar las casillas separados por coma.

```
var diasSemana = new Array ("Lunes", "Martes", "Miércoles",
"Jueves",      "Viernes",      "Sábado",      "Domingo")
```

MEDAC

Inicialización de arrays (*)

```
var mezcla = new Array ("Lunes", 3.14159, "true", false,
23, "11");
```

MEDAC

Control de arrays con el bucle FOR

- El uso del bucle FOR para recorrer arrays es de lo más habitual en programación. Esto es porque un bucle FOR da siempre el número de vueltas que le indiquemos. Vueltas que se pueden aprovechar para leer/introducir cosas en los cajones de un array de forma rápida y limpia (escribiendo unas pocas líneas de código). Para ello basta con tener en cuenta que el primer cajón de un array es el 0 y el último es el número total de cajones menos 1.

- Para recorrer un array de forma rápida se usa el bucle FOR:
  - Indicamos las vueltas que vamos a dar (desde 0 hasta el número total de cajones menos 1).
  - Para cada vuelta, la variable 'i' va a tomar cada uno de los números correspondientes a los cajones del array. De esta forma, podemos usar esa variable para abrir un cajón distinto en cada vuelta.

Control de arrays con el bucle FOR

```
var  meses = new Array (12).

meses[0] = "Enero";
meses[1] = "Febrero";
meses[2] = "Marzo";
…
meses[11] = "Diciembre";
```

```
for(let i=0;i<=11;i++){ //empiezo en 0 y acabo en 11 (12-1)
  console.log(meses[i]);
}
```

Control de arrays con el bucle FOR (*)

- También, es muy habitual que se utilice la propiedad length para poder recorrer un array, del cual no conocemos su tamaño, por todas sus posiciones.

```
for(let i=0;i<=(miArray.length -1);i++){
  console.log(miArrat[i]);
}
```

MEDAC

Dinamismo en los arrays (*)

- A diferencia de otros lenguajes, En Javascript los arrays son dinámicos, es decir, si intentamos acceder a un cajón que no existe, este cajón se crea automáticamente, así como todos los que “falten” hasta llegar a él. Además, el valor de la propiedad length queda actualizado también.

- Las posiciones que se crean nuevas sin valor quedan como undefined o null

Control de arrays con el bucle WHILE (*)

- El bucle while se usa también habitualmente para recorrer arrays

- Es importante indicar que el bucle while se usa con arrays cuando se realizan búsquedas de valores en el array: sigo en el bucle mientras no lo encuentre. Si lo encuentro, acabo el bucle (aunque no haya acabado el array).

MEDAC

Ejercicios básicos con Arrays (*)

Listado de ejercicios Tema 6 Aula Virtual (Ejercicios 1 Arrays Básicos.pdf)

Arrays Avanzados

- Javascript nos permite un control de la estructura tipo Array mucho más completo
- Vamos a tener a nuestra disposición una serie de funciones ya predefinidas que nos facilitarán el realizar operaciones tanto con Arrays como con cada uno de sus elementos
- Además, vamos a poder crear un par de estructuras complejas basadas en Arrays: los arrays multidimensionales y los arrays asociativos.
- Javascript proporciona un sistema de notación propio llamado JSON (Javascript Object Notation) el cual nos va a permitir, entre otras cosas, definir de manera muy sencilla Arrays y estructuras basadas en arrays tan complejas como queramos.

Definición (*)

- Aunque el uso de new Array es correcto actualmente es mucho más rápido y sencillo definir los arrays usando la nomenclatura JSON:

```
let formaLarga = new Array('a','e','i','o','u');
let formaCorta = ['a','e','i','o','u'];
```

```
let vacioLargo = new Array();
let vacioCorto = [];
```

MEDAC

length

- Indica el número de elementos que posee el Array. En realidad, length no es una función, es una variable asociada a cada array. Por ese motivo no lleva paréntesis ni argumentos.

concat ()

- Esta función nos permite concatenar elementos a un array. Devuelve un array con los elementos nuevos concatenados, es decir, no modifica el array sobre el que se invoca la función.

(*)

pop ()

- Esta función elimina el ultimo elemento del array y devuelve dicho elemento. Es decir, el array sobre el que se invoca esta función se modifica y disminuye su longitud.

push()

- Esta función añade un elemento al final del array. El elemento se le pasa como argumento. El array sobre el que se invoca la función queda afectado y aumenta su longitud. Se usa habitualmente cuando tenemos que añadir elementos a un array del que se desconoce su tamaño

(*)

MEDAC

shift ()

- Esta función es exactamente igual a pop() pero elimina y devuelve el primer elemento del array

unshift ()

- El funcionamiento de esta función es análogo a push() con la diferencia de que unshift() añade el elemento por el principio del array.

(*)

MEDAC

splice ()

- Esta función se utiliza para añadir, eliminar o reemplazar elementos de un array. Hay que tener en cuenta que modifica el array original.
- La sintaxis es la siguiente:

```
array.splice (inicio, numeroDeElementos, elemento1, elemento2, ……. elementoN);
```

- inicio: índice en el array donde comenzar a hacer cambios
- numeroDeElementos: Número de elementos a eliminar desde la posición inicio
- elemento1, elemento2…….elementoN: (Opcional) Nuevos elementos que se agregarán en la posición inicio

(*)

MEDAC

slice ()

- Esta función se usa para extraer una porción de un array, sin modificar el original
- La sintaxis es la siguiente

```
let subarray = array.slice (inicio, fin);
```

- inicio: índice en el que comenzar la extracción (incluido)
- fin: (Opcional) índice en el que terminar la extracción (no incluido). Si no se proporciona, extrae hasta el final del array

(*)

MEDAC

split ()

- Aunque esta función se aplica a cadenas de caracteres nos interesa verla aquí ya que se utiliza para dividir un elemento de tipo String en un array de cadenas mediante la separación de la cadena en subcadenas. A la función split habrá que indicarle el carácter separador

join ()

- Esta función es la contraria a la función de cadenas split(). Se aplica sobre un array y necesita como argumento un carácter separador. Devuelve una cadena con todos los elementos del array unidos por el carácter separador. Transforma por tanto arrays en cadenas

reverse ()

- Esta función toma el array sobre el que se invoca y reordena sus elementos de forma inversa. Es decir, el orden del array original queda modificado.

(*)

MEDAC

Spread Operator en Arrays

- Este operador (operador de propagación en español) es muy versátil y nos permitirá transformar, copiar y concatenar arrays de una forma rápida y sencilla.

- El operador se compone del carácter punto repetido 3 veces y se debe usar siempre como primer elemento del array: [...

Spread Operator - Copiando Arrays

- Tal y como ocurre en otros lenguajes de programación, cuando asignamos una variable que contiene un array a otra, no se realiza una copia de esta, sino que se crea una nueva referencia. Es decir, hay dos variables apuntando al mismo array. Si una variable cambia el contenido del array, la otra variable ve esos cambios:

```
let vocales = ['a','e','i','o','u'];
let copia = vocales; //asigno una copia
copia[0] = 25; //cambio valores de la copia
console.log(vocales) //[ 25, 'e','i','o','u' ]
```

MEDAC

Spread Operator - Copiando Arrays

- Esto es algo que podemos solucionar usando el operador de propagación:

```
let vocales = ['a','e','i','o','u'];
let copia = [...vocales]; //asigno una copia
copia[0] = 25; //cambio valores de la copia
console.log(vocales) //[ 'a', 'e','i','o','u' ]
```

(*)

MEDAC

Spread Operator – Concatenando Arrays

- Con el operador de propagación podemos concatenar arrays de forma sencilla:

```
let vocales = ['a','e','i','o','u'];
let numeros = new Array(1,2,3,4,5);
let mix = [...vocales,...numeros];

console.log(mix); //[ 'a', 'e', 'i', 'o', 'u', 1, 2, 3, 4, 5 ]
```

(*)

Spread Operator – Transformando Arrays

- Si usamos el operador de propagación sobre una cadena de caracteres, vamos a obtener el mismo resultado que si usáramos el método .split sin separador en dicha cadena:

```
let frase = "Martillo va!"
let arrayT = [...frase];
console.log(arrayT);
//[ 'M', 'a', 'r', 't', 'i', 'l', 'l', 'o', ' ', 'v', 'a', '!' ]
```

(*)

Ejercicios Arrays Avanzados

MEDAC

Array multidimensionales

- Un array multidimensional es una estructura compuesta de un array donde cada elemento de ese array es a su vez, otro array. Además, esos arrays pueden contener, a su vez, otros arrays como elementos y seguir así hasta que queramos.

- Cada vez que hacemos esto (definir arrays cuyos elementos son arrays) se dice que estamos añadiendo una dimensión a la estructura.

Array multidimensionales

Dependiendo de las veces que cumplamos la condición anterior a la hora de construir la estructura, podemos tener por ejemplo:

- Estructuras de 2 dimensiones (bidimensionales): son aquellas donde los elementos del array son a su vez arrays de elementos (números, cadenas, booleanos...). Son estructuras muy comunes que se usan para representar mapas, matrices de elementos, tableros, coordenadas...

- Estructuras de 3 dimensiones (tridimensionales): los elementos del array son arrays que a su vez tienen arrays de elementos. Se suelen usar para representar ubicaciones o instancias en entornos 3D.

- Estructuras de más de 3 dimensiones (multidimensionales): se sigue la regla de colocar arrays de arrays tantas veces como haga falta. Son estructuras complejas.

Array multidimensionales

Ejemplo de Array de 2 dimensiones: gestión de una tabla de calificaciones para alumnos de diferentes asignaturas. Imagina que tienes 5 alumnos y cada uno de ellos tiene 3 calificaciones correspondientes a 3 asignaturas (Matemáticas, Lengua, Ciencias).

```
Tabla de calificaciones:
Alumno  Matemáticas Lengua  Ciencias
Alumno 1    8       7       9
Alumno 2    5       6       8
Alumno 3    9       8       7
Alumno 4    7       6       9
Alumno 5    10      9       8
```

MEDAC

Array multidimensionales

Ejemplo de Array de 3 dimensiones: gestión de ventas diarias en diferentes tiendas de una cadena. Supongamos que tienes una cadena de 3 tiendas, cada una de ellas vende 4 productos, y quieres registrar las ventas durante una semana (7 días).

En este caso, puedes usar un array de tres dimensiones, donde:

- La primera dimensión representa la tienda (3 tiendas).
- La segunda dimensión representa los productos (4 productos).
- La tercera dimensión representa los días de la semana (7 días).

Definición de arrays multidimensionales

Notación new Array ()

Vamos a crear un array de dos dimensiones donde almacenaremos las temperaturas medias de 3 ciudades durante los meses de invierno (3 meses).

Definición de arrays multidimensionales

Notación new Array ()

- Primero creamos 3 arrays, correspondientes a las temperaturas medias de cada una de las ciudades, cuyos elementos serán las temperaturas medias de los 3 meses de invierno.

```
var temperaturas_medias_ciudad0 = new Array(12,10,11);

var temperaturas_medias_ciudad1 = new Array (5,0,2);

var temperaturas_medias_ciudad2 = new Array (10,8,10);
```

Definición de arrays multidimensionales

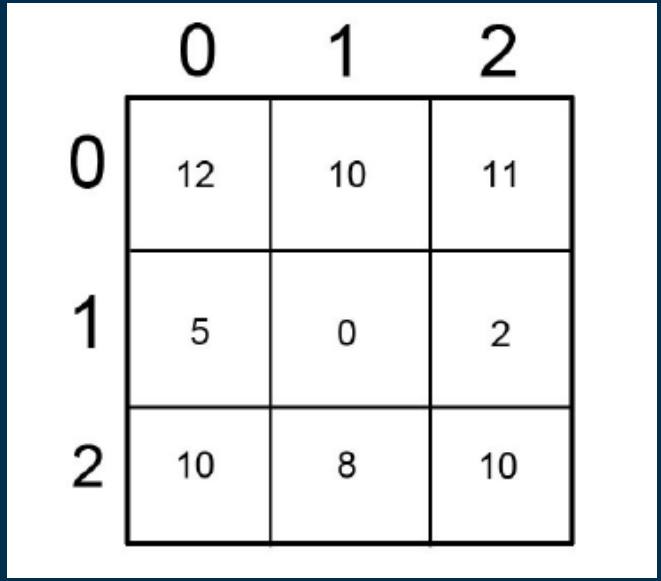
Notación new Array ()

- Después creamos un nuevo array que representarán las temperaturas medias de 3 ciudades y dentro de cada casilla del array introduciremos las temperaturas creadas anteriormente

```
var temperaturas_cuidades = new Array (3);
temperaturas_cuidades[0] = temperaturas_medias_ciudad0;
temperaturas_cuidades[1] = temperaturas_medias_ciudad1;
temperaturas_cuidades[2] = temperaturas_medias_ciudad2;
```

Definición de arrays multidimensionales

Notación new Array ()



	0	1	2
0	12	10	11
1	5	0	2
2	10	8	10

(*)

MEDAC

Definición de arrays multidimensionales

Notación JSON

Vamos a crear un array de dos dimensiones donde almacenaremos las temperaturas medias de 3 ciudades durante los meses de invierno (3 meses).

Definición de arrays multidimensionales

Notación JSON

```
//Definición con notación JSON

var tempCiudad0 = [12,10,11];
var tempCiudad1 = [5,0,2];
var tempCiudad2 = [10,8,10];

var tempCiudadJson = [tempCiudad0,tempCiudad1,tempCiudad2]
```

MEDAC

Definición de arrays multidimensionales

Notación JSON

```
var temp_ciudadesJson = [
    [12,10,11],
    [5,0,2],
    [10,8,10]
]
```

(*)

MEDAC

Recorriendo Arrays bidimensionales

- Cuando trabajamos con arrays multidimensionales, para acceder a sus elementos vamos a necesitar tantos índices como dimensiones tenga el array..
- Para los arrays bidimensionales vamos a necesitar dos índices. Estos arrays pueden verse como una tabla con filas y columnas. El primer índice serviría para movernos por las filas, y el segundo para las celdas de cada fila (las columnas).

Recorriendo Arrays bidimensionales

- Si quisiéramos mostrar elementos del array anterior

```
alert(temperaturas_cuidades[0][0]); //12
alert(temperaturas_cuidades[0][2]); //11
alert(temperaturas_cuidades[2][1]); //8
```

Recorriendo Arrays bidimensionales

- Si quisiéramos recorrer toda la estructura, debemos recorrer el array original y, a su vez, ir recorriendo el array que corresponde a cada elemento del array original. Es decir, siguiendo la analogía antes comentada, vamos recorriendo las filas y dentro de cada fila, recorremos las celdas.

- El método para hacer un recorrido dentro de otro es colocar un bucle dentro de otro, lo que se llama un bucle anidado. Así que necesitamos meter un bucle FOR dentro de otro

Recorriendo Arrays bidimensionales

```
//Bucle para recorrer las filas
for (let i=0;i<temperaturas_cuidades.length; i++){
 //bucle para recorrer las celdas de cada fila
  for (j=0;j<temperaturas_cuidades[i].length; j++){
      console.log(temperaturas_cuidades[i][j]);
  }
}
```

(*)

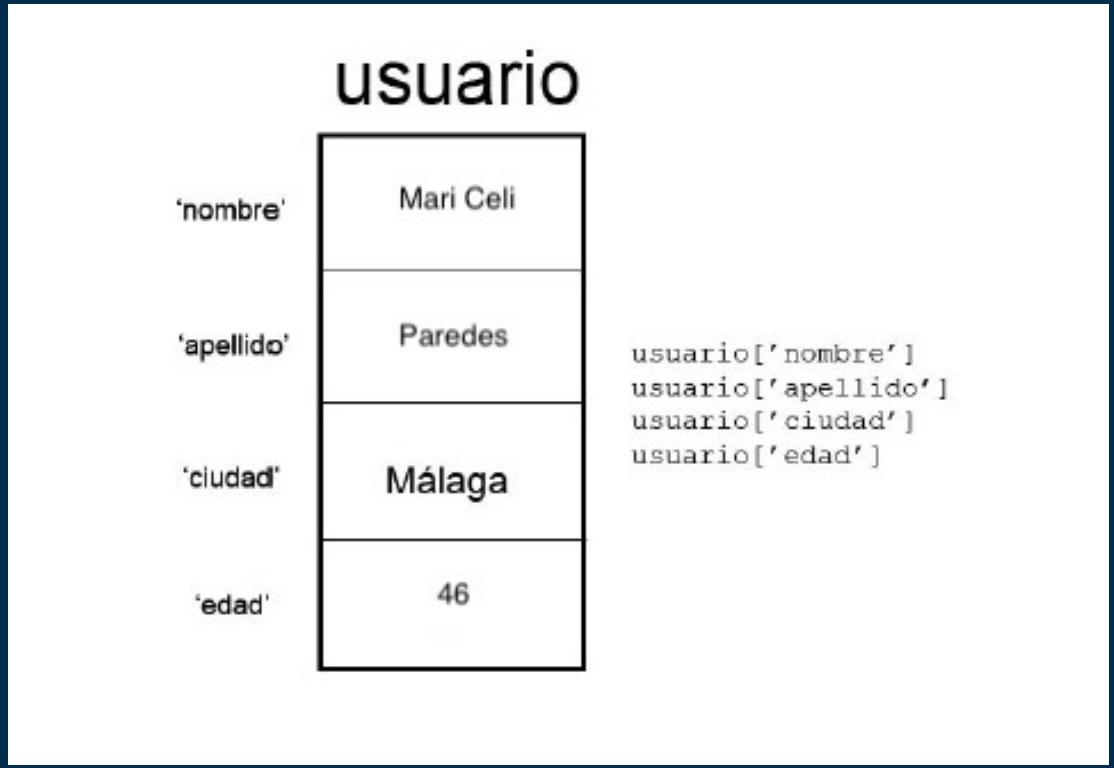
MEDAC

Array asociativos (mapas)

- Los arrays asociativos son un tipo de array muy útiles que consisten en arrays cuyos índices no son números, sino cadenas de texto (en otros lenguajes se llaman mapas o diccionarios). Es decir, para acceder a una celda de un array asociativo vamos a usar su cadena índice en lugar del número de su posición.

- Esto es especialmente útil porque, por un lado, somos nosotros los que decidimos que nombre (cadena) va a tener cada celda del array, y por otro lado, ya no es importante el orden de las celdas puesto que no voy a usar índices numéricos para acceder a ellas.

Array asociativos (mapas)



Definición de arrays asociativos

- A diferencia de otros lenguajes, Javascript no posee el elemento 'array asociativo' como tal. Lo que se hace en realidad es definir un objeto cuyo comportamiento imita el funcionamiento de los arrays asociativos.

Notación new Array ()

```
var usuario = new Array ();
//definimos los elementos
usuario['nombre']="Mariceli";
usuario['apellido'] = "Paredes";
usuario['ciudad'] = "Málaga"
usuario['edad'] = 46;

console.log(usuario);
```

(*)

Definición de arrays asociativos

Notación JSON

```
//Definimos con notación JSON

let persona = {
    "nombre": "Mariceli",
    "apellido": "Paredes",
    "ciudad": "Malaga",
    "edad": 46
}
```

MEDAC

Acceso a elementos de arrays asociativos

```
//Acceso a un elemento
console.log (persona["apellido"]);
```

(*)

MEDAC

Consideraciones para arrays asociativos

Dada la naturaleza especial de estos arrays en Javascript, al crear un array asociativo nos encontramos con una serie de problemas

- Los arrays asociativos no disponen de un método que nos devuelva su longitud (length devuelve 0 o undefined).
- Al no poder controlar la longitud del array, no podemos usar bucles FOR normales para recorrerlos
- Muchas de las funciones vistas para arrays normales no funcionan con este tipo de arrays.

MEDAC

Instituto Oficial de Formación Profesional