Desarrollo Web Frontend

Edición 2021

Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología – Sede Oro Verde

Arreglos

Métodos en los arreglos

Manejo de cadenas (string)

Funciones

Arreglos

Existen dos formas de crear arreglos en JavaScript

- Utilizando la sentencia **new Array();**

let colores = **new** Array(); → declaración de un arreglo sin posiciones.

let colores = **new** Array(10); → declaración de un arreglo de 10 posiciones.

let colores = **new** Array('rojo','verde','azul'); →declaración de un arreglo con valores iniciales rojo, verde y azul.

- Utilizando notación de corchetes **[ ]**

let colores = **[ ];** → definimos un arreglo vacío.

let colores = **[**'rojo', 'verde', 'azul'**]**; → definimos un arreglo con valores iniciales

Arreglos

Acceso a los elementos de un arreglo

let arreglo = [‘primero’,’segundo’,’tercero’];

console.log(arreglo[0]); → ‘primero’

console.log(arreglo[1]); → ‘segundo’

console.log(arreglo[arreglo.length - 1]); → ‘tercero’ (último elemento de la lista)

console.log(arreglo.0) → **error de sintáxis**

Arreglos

Recorrer elementos de un arreglo

```
let arreglo = [‘primero’,’segundo’,’tercero’];

arreglo.foreach( function(valor, indice) {

        //.....
        //.....
        console.log( indice + “=>” + valor );

});

//Devuelve:
        0 => primero
        1 => segundo
        2 => tercero
```

~~Arreglos~~

Métodos en los arreglos

Manejo de cadenas (string)

Funciones

Métodos en los arreglos

JavaScript cuenta con diversos métodos para el manejo de arreglos. Veremos algunos de ellos a continuación:

- **length** → permite conocer el número de elementos del arreglo. (También puede utilizarse con cadenas de texto).

```
let colores = [‘azul’, ‘negro’, ‘rojo’];

colores.length; → 3
```

Otras maneras de utilizar **.length**

```
colores.length = 2;

//colores → [‘azul’, ‘negro’];


colores[colores.length] = ‘verde’;

//colores → [‘azul’,’negro’,’amarillo’]
```

Métodos en los arreglos

- **toString()** → permite mostrar el contenido del arreglo como un string donde los elementos se separan por coma.

```
let colores = [‘Verde’, ‘Azul’, ‘Amarillo’];

colores.toString(); → “azul,verde,amarillo”
```

- **indexOf(‘valor’)** → permite encontrar el índice de un elemento del arreglo (Puede utilizarse con cadenas de texto).

```
let colores = [‘Verde’, Azul’, ‘Amarillo’];

let pos = colores.indexOf(‘Azul’);

//pos → 1
```

Métodos en los arreglos

- **push(valor, [valor2], [valor3], ...)** → Transforma un arreglo añadiendo elementos al final de la lista y devolviendo la nueva longitud del array.

```
let colores = ['azul','rojo','verde'];

colores.push('blanco'); → 4     // ['azul','rojo','verde','blanco']
colores.push('negro','amarillo'); → 6     //['azul','rojo','verde','blanco','negro','amarillo']
```

- **pop()** → Permite eliminar el último elemento de la lista. Devuelve el elemento eliminado.

```
let colores = ['azul','rojo','verde','blanco'];

colores.pop(); → 'blanco';        // ['azul','rojo','verde']
```

Métodos en los arreglos

- **Shift()** → obtiene el primer elemento del arreglo.

```
let animales = [‘perro’, ‘gato’,’tigre’];

animales.shift(); → devuelve “perro”
```

- **unshift()** → permite agregar elementos al inicio del arreglo. Devuelve el tamaño del arreglo.

```
let animales = [‘perro’, ‘gato’,’tigre’];

animales.unshift(‘conejo’,’elefante’);
//devuelve → 5

// [‘conejo’,’elefante’,‘perro’, ‘gato’,’tigre’]
```

Métodos en los arreglos

- **reverse()** → Devuelve un arreglo con los elementos en orden inverso.

```
let animales = [‘conejo’,’elefante’,‘perro’, ‘gato’,’tigre’];

animales.reverse();

devuelve → [‘tigre’,’gato’,’perro’,’elefante’,’conejo’];
```

- **sort()** → permite ordenar elementos de un arreglo alfabéticamente. No es recomendado utilizarse con valores numéricos. Devuelve el arreglo ordenado.

```
let animales = [‘tigre’,’gato’,’perro’,’elefante’,’conejo’];

animales.sort();

devuelve → [’conejo’,’elefante’,’gato’,’perro’,’tigre’];
```

~~Arreglos~~

~~Métodos en los arreglos~~

Manejo de cadenas (string)

Funciones

Manejo de cadenas (string)

JavaScript convierte automáticamente las cadenas literales (definidas con comillas dobles o simples) en objetos **String**. Es por esto que es posible utilizar métodos del objeto **String** sobre cadenas primitivas.

El que una cadena se haga muy grande no es un problema. JavaScript no impone límite práctico alguno sobre el tamaño de las cadenas, así que no hay ninguna razón para preocuparse sobre las cadenas largas.

Manejo de cadenas (string)

Acceder a un carácter

Existe dos formas de acceder a un carácter individual en una cadena de texto.

- Utilizando el método **charAt*(posicion)***

'perro'**.charAt(1)** → devuelve "e"

- Tratando la cadena como un arreglo

'perro'**[1]** → devuelve "e"

Manejo de cadenas (string)

Convertir números en string y viceversa

- El objeto **Number** convertirá cualquier valor que se le pase en un número, si puede.

```
let texto = '456'; → // typeof String

let numero = Number(texto); → //typeof Number
```

De manera inversa, cada número tiene un método llamado **toString()** que convertirá el equivalente en un string.

```
let miNumero = 123; → // typeof Number

let miString = miNumero.toString(); // typeof String
```

Manejo de cadenas (string)

- **toLowerCase()** → permite transformar toda la cadena de texto a minúscula

```
let texto = 'HoLa MunDo';
texto.toLowerCase(); → // devuelve 'hola mundo'
```

- **toUpperCase()** → transforma toda la cadena de texto a mayúscula.

```
let texto = 'hola mundo';
texto.toUpperCase(); → // devuelve "HOLA MUNDO"
```

Manejo de cadenas (string)

- **replace('texto_a_reemplazar', 'nueva_Cadena')** → permite reemplazar una subcadena de texto por otra. Devuelve el nuevo String.

```
let texto = "El perro vuela";

let otroTexto = texto.replace('perro', 'pajaro'); → // "El pajaro vuela";

let nuevoTexto = texto.replace('vuela','ladra'); → // "El perro ladra"
```

Manejo de cadenas (string)

- **substring(inicio, fin)** → devuelve una subcadena de texto que comienza en el carácter de inicio y termina en el carácter de **fin - 1**.

```
let texto = “El perro vuela”;
let subCadena = texto.substring(3,8); → // “perro”
```

~~Arreglos~~

~~Métodos en los arreglos~~

~~Manejo de cadenas (string)~~

Funciones

Funciones

Formas de definir una función en JavaScript:

- **Método 1:** Funciones por declaración

```
function nombre_funcion(parámetros){
    //sentencias…
    return valor
}
```

```
function saludar() {
    return "Hola";
}

saludar(); // 'Hola';
```

→ **nombre:** nombre de la función.

→ **parámetros:** variables y/o constantes que se tratarán como variables locales dentro de la función.

→ **return:** valor que devuelve la función.

Funciones

Formas de definir una función en JavaScript:

- **Método 2:** Funciones anónimas o funciones lambda

```
const miConstante = function(parametros){
        //sentencias
        return valor
}
```

```
const multiplica = function(a,b) {
    return a*b;
}

multiplica(4,5); // devuelve 20
```

→ **miConstante**: variable o constante que contiene la función. Se ejecuta dicha variable.

Funciones

Formas de definir una función en JavaScript:

- **Método 3:** Funciones flecha

```
const f  = () => {
        //sentencias
        return valor
}
```

```
const f = function () {
    return "Función tradicional";
}

const f = () => {
    return "Función flecha"
}
```

Funciones

Funciones flecha

- Si el cuerpo de la función sólo tiene una línea, se puede omitir el ámbito de las llaves { }.

```
const f = () => alert("Función flecha");
```

- Si la función posee un solo parámetro, se puede indicar simplemente el nombre del mismo. Si los parámetros son 2 o más deben indicarse entre paréntesis separados por coma.

```
const f = e => e + 1;
f(2); //devuelve 3

const a = (x,y) => x*y;
a(5,3); //devuelve 15;
```

Funciones

Formas de definir una función en JavaScript:

- **Método 4:** Funciones como objetos

```
const miConstante = new Function("return 'Hola'");

miConstante();  //devuelve 'Hola';
```

Las funciones pueden declararse incluso como objetos. Pero este enfoque no suele implementarse realmente.

Permite comprender que en JavaScript todo puede ser un objeto.

Parámetros rest

Los parámetros rest permiten representar un número indefinido de argumentos como un arreglo.

```
function multiplicar(multiplo, ...valores) {
    return valores.map(unValor => multiplo * unValor);
}

let arr = multiplicar(2,4,5,6);
console.log(arr); // Devuelve [8,10,12]
```

En este ejemplo vemos que el parámetro rest **...valores** recopila los argumentos desde el segundo al último en la llamada a la función **multiplicar()** .

¿Consultas?