



JS

UT4. FUNCIONES, ARRAYS Y OBJETOS DEFINIDOS POR EL USUARIO

Desarrollo Web en Entorno
Cliente
2ºDAW

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

JavaScript cuenta con una serie de funciones integradas en el lenguaje.

Dichas funciones se pueden utilizar sin conocer todas las instrucciones que ejecuta.

Simplemente se debe conocer el nombre de la función y el resultado que se obtiene al utilizarla.

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

encodeURI

encodeURIComponent

decodeURI

decodeURIComponent

Cómo utilizarlos: <https://www.freecodecamp.org/espanol/news/url-codificacion-como-utilizar-encodeuricomponent-javascript/>

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

eval() → convierte y evalúa una cadena pasada como si fuese código JS

isFinite() → Verifica si el numero que le pasamos es un numero finito.

Number.isNaN() → Comprueba si el valor pasado es un NaN. Es más robusto que isNaN()

String() → Devuelve un objeto pasado como string.

Number() → Devuelve el objeto pasado como number.

parseInt() → Convierte la cadena que le pasamos a un numérico entero.

parseFloat() → Convierte la cadena que le pasamos como un numérico en punto flotante.

2. FUNCIONES DEL USUARIO

Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

Es posible crear funciones personalizadas diferentes a las funciones predefinidas por el lenguaje.

Hay dos tipos de declaraciones de funciones

- La clásica mediante la palabra reservada `function`
- Mediante las [arrow functions](#) (desde ES6)

2.1 DECLARACIÓN DE FUNCIONES

Creación de funciones mediante declaración.

```
function nombreFuncion ( [parámetros ]  
{  
    instrucciones  
    [return valor]  
}
```

2.1 DECLARACIÓN DE FUNCIONES

De esta manera la función existirá a todo lo largo del código. Podemos ejecutar “saludar” antes de declararla incluso.

```
function saludar() {  
    return "Hola";  
}  
  
saludar(); // 'Hola'  
typeof saludar; // 'function'
```

Fuente <https://lenguajejs.com/javascript/fundamentos/funciones>

2.1 DECLARACIÓN DE FUNCIONES

Declaración de funciones por expresión. Asignamos la función a una variable.

Aquí no se puede invocar la función antes de hacer la asignación

```
// El segundo "saludar" (nombre de la función) se suele omitir
const saludo = function saludar() {
    return "Hola";
};

saludo(); // 'Hola'
```

2.1 DECLARACIÓN DE FUNCIONES

Podemos incluso declarar una función a partir del objeto global Function.

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

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

2.1 DECLARACIÓN DE FUNCIONES

Podemos declarar una **función anónima** o “**lambda**” para asignársela a una variable. El nombre de la función no haría falta es redundante.

```
// Función anónima "saludo"
const saludo = function () {
    return "Hola";
};

saludo; // f () { return 'Hola'; }
saludo(); // 'Hola'
```

2.2 FUNCIONES CALLBACK

A grandes rasgos, un **callback** (llamada hacia atrás) es pasar una **función B por parámetro** a una **función A**, de modo que la función A puede ejecutar esa función B de forma genérica desde su código, y nosotros podemos definirlas desde fuera de dicha función

```
// fB = Función B
const fB = function () {
    console.log("Función B ejecutada.");
};

// fA = Función A
const fA = function (callback) {
    callback();
};

fA(fB);
```

Fuente: <https://lenguajejs.com/javascript/fundamentos/funciones/#callbacks>

2.3 ARROW FUNCTIONS

Son una forma corta de escribir funciones que aparece en Javascript a partir de **ECMAScript 6**.

Diferencias y limitaciones:

- No tiene sus propios enlaces a this o super y no se debe usar como métodos.
- No tiene argumentos o palabras clave new.target.
- No apta para los métodos call, apply y bind, que generalmente se basan en establecer un ámbito o alcance.
- No se puede utilizar como constructor.
- No se puede utilizar yield dentro de su cuerpo.

Fuente:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions/Arrow_functions

2.3 ARROW FUNCTIONS

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

const func = () => {
    return "Función flecha.";
};
```

2.3 ARROW FUNCTIONS

```
// Función tradicional
const f1 = function (a){
  return a + 100;
}

// Desglose de la función flecha

// 1. Elimina la palabra "function" y coloca la flecha entre el argumento y el corchete de apertura.
const f2 = (a) => {
  return a + 100;
}

// 2. Quita los corchetes del cuerpo y la palabra "return" – el return está implícito.
const f3 = (a) => a + 100;

// 3. Suprime los paréntesis de los argumentos
const f4 = a => a + 100;
```

3. ARRAYS

Un Array es un conjunto ordenado de valores relacionados.

Cada uno de estos valores se denomina elemento y cada elemento tiene un índice que indica su posición numérica en el Array.

Es necesario declarar un Array antes de poder usarlo.

3.1 DECLARACIÓN DE ARRAYS

```
const a1 = new Array(); // nuevo array vacío.  
  
const a2 = new Array(4); //Nuevo array de 4 elementos  
  
//nuevo array con tres elementos definidos  
const apellidos = new Array('Perez', 'Martínez', 'González');  
  
//así podemos inicializar un array con valores:  
for (let i = 0; i < 10; i++) {  
    a1[i] = Math.random();  
}
```

3.2 RECORRER ARRAYS

Podemos recorrerlo de tres modos:

- Con un bucle for
- Con un bucle for of (desde ES6)
- Con el método forEach

```
//Podemos recorrerlo con el tradicional bucle for

for (let i = 0; i < a1.length; i++) {
    console.log(a1[i]);
}

//ahora vamos a recorrerlo pero con un bucle for_of (ES6)

for (num of a1) {
    console.log(num);
}

//También podemos recorrerlo con un forEach

a1.forEach(function(elemento, indice, array) {
    console.log(elemento, indice);
})
```

3.3 PROPIEDADES Y MÉTODOS DE LOS ARRAYS

El objeto Array tiene dos propiedades:

1.length:

```
for (let i=0; i< codigos_productos.length; i++) {  
    document.write(codigos_productos[i] + "<br>");  
}
```

2.prototype:

```
Array.prototype.nueva_propiedad= valor;  
Array.prototype.nuevo_metodo= nombre_de_la_funcion;
```

3.3 PROPIEDADES Y MÉTODOS DE LOS ARRAYS

Métodos	
push ()	shift ()
concat ()	pop ()
join ()	slice ()
reverse ()	sort ()
unshift ()	splice ()

Documentación:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array

4. OBJETOS

Un objeto es una colección de propiedades, y una propiedad es una asociación entre un nombre (o clave) y un valor.

El valor de una propiedad puede ser una función, en cuyo caso la propiedad es conocida como un método.

Además de los objetos que están predefinidos en el navegador, puedes definir tus propios objetos

Fuente:

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Working_with_Objects

4.1 DECLARACIÓN DE OBJETOS

Podemos declararlos de dos modos:

```
//podemos declarar un objeto vacío e ir creando sus propiedades
const myCar = new Object();
myCar.make = 'Ford';
myCar.model = 'Mustang';
myCar.year = 1969;

//podemos crear directamente un objeto dando sus propiedades y métodos
|
const myCar2 = {
    make: 'Seat',
    model: '127',
    year: '1975'
}
```

4.1 DECLARACIÓN DE OBJETOS

Así podemos declarar métodos: Es una propiedad que a su vez es una función.

La palabra reservada `this` nos permite acceder a las propiedades del propio objeto:

```
//declaración de un objeto con propiedades y métodos
const persona = {
    nombre: 'juan',
    apellido: 'sin miedo',
    edad: 30,
    saludar: function () {
        //this es la palabra reservada para acceder a un valor del objeto
        console.log(`Me llamo ${this.nombre}`);
        console.log('Hola!');
    }
};
```

4.2 FUNCIÓN CONSTRUCTORA

Podemos crear una función constructora y luego crear instancias de un objeto.

```
function Persona(nombre, apellido, fechaNac) {  
    this.nombre = nombre;  
    this.apellido = apellido;  
    this.anioNac = fechaNac;  
    this.getFechaNac = function () {  
        return this.anioNac;  
    }  
}  
  
//Creamos una instancia del objeto Persona  
const p1 = new Persona('Pepe', 'Pérez', 1975);
```

4.3 DECLARACIÓN DE CLASES

Desde ES6 podemos declarar clases. La clase tendrá un método “constructor”.

Podemos declarar métodos setters y getters.

```
class Persona {  
    //método constructor  
    constructor(nombre, apellidos, anioNac) {  
        this._nombre = nombre;  
        this._apellidos = apellidos;  
        this._anioNac = anioNac  
    }  
    //podemos declarar métodos getters, setters...  
    get anioNac() {  
        return this._anioNac;  
    }  
}
```