Desarrollo Web en Entorno Cliente  
UD 02. Sintaxis Javascript ES6

short line

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

**📖 Importante**

**❕ Atención**

**💬 Interesante**

**Índice de contenido**

[1. Funciones 3](#_Toc83656660)

[2. Funciones flecha (arrow functions) 5](#_Toc83656661)

[3. Clases en Javascript 6](#_Toc83656662)

[4. Más cosas interesantes de Javascript ES6 7](#_Toc83656663)

[5. Material adicional 8](#_Toc83656664)

[6. Bibliografía 8](#_Toc83656665)

UD01. Sintaxis Javascript ES6

# Funciones

Una función es un conjunto de instrucciones que se agrupan bajo un nombre de función. Se ejecuta solo cuando es llamada por su nombre en el código del programa. La llamada provoca la ejecución de las órdenes que contiene.

Las funciones son muy importantes por diversos motivos:

* Ayudan a estructurar los programas para hacerlos su código más comprensible y más fácil de modificar.
* Permiten repetir la ejecución de un conjunto de órdenes todas las veces que sea necesario sin necesidad de escribir de nuevo las instrucciones.

Una función consta de las siguientes partes básicas:

* Un nombre de función.
* Los parámetros pasados a la función separados por comas y entre paréntesis.
* Las llaves de inicio y final de la función.
* Desde Javascript ES6, se pueden definir valores por defecto para los parámetros.

**Sintaxis de la definición de una función**

|  |
| --- |
| **function** **nombrefuncion** (parámetro1, parámetro2=valorPorDefecto){ *// instrucciones* *//si la función devuelve algún valor añadimos:* **return** valor; } |

**Sintaxis de la llamada a una función**

|  |
| --- |
| *// La función se ejecuta siempre que se ejecute la sentencia.* valorRetornado=nombrefuncion (parám1, parám2); |

Es importante entender la **diferencia entre definir una función y llamarla:**

* Definir una función es simplemente especificar su nombre y definir qué acciones realizará en el momento en que sea invocada, mediante la palabra reservada function.
* Para llamar a una función es necesario especificar su nombre e introducir los parámetros que queremos que utilice. Esta llamada se puede efectuar en una línea de órdenes o bien a la derecha de una sentencia de asignación en el caso de que la función devuelva algún valor debido al uso de la instrucción return.

La definición de una función se puede realizar en cualquier lugar del programa, pero se recomienda hacerlo al principio del código o en un fichero “js” que contenga funciones.

La llamada a una función se realizará cuando sea necesario, es decir, cuando se demande la ejecución de las instrucciones que hay dentro de ella.

**Ejemplo**: funciones que devuelve la suma de dos valores que se pasan por parámetros y que escriben el nombre del profesor.

|  |
| --- |
| *// Definiciones de las funciones* **function** **suma** (a,b){  *// Esta función devuelve un valor*  **return** a+b; }  *// Esta función muestra un texto, pero no devuelve un valor* **function** **profe** (){   console.log("El profesor es muy bueno");   *// OJO: esto es un ejemplo, pero rara vez se realiza en una función real* } *// Código que se ejecuta* **let** op1=5, op2=25; **let** resultado; *// Llamada a función* resultado=suma(op1,op2);  *// llamada a la función* console.log (op1+"+"+op2+"="+resultado); *// Llamada a función* profe(); |

**❕ Atención:** recordad que dentro de las funciones **rara vez se utilizan funciones de entrada/salida**. El 99.9% de las veces simplemente procesan la entrada por parámetros y devuelven un valor.

**Ejercicio 1**

Se ha implementado una función llamada saludar que recibe un parámetro nombre y debe retornar un saludo que incluya dicho nombre. Sin embargo, hay partes del código que faltan. Tu tarea es completar las partes faltantes para que la función funcione correctamente.

function saludar(nombre) {

return "¡Hola " + \_\_\_\_1\_\_\_\_ + "!";

}

console.log(saludar("Juan")); // Debería imprimir: "¡Hola Juan!"

**Ejercicio 2**

Dada la siguiente función que verifica si un número es primo o no, completa las partes faltantes para que la función funcione correctamente.

function esPrimo(n) {

if (n <= 1) return false;

for (let i = 2; i < n; i++) {

if (n % i === 0) {

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ false;

}

}

return true;

}

console.log(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(11)); // Debería imprimir: true

**Ejercicio 3**

Completa el siguiente código para que calcule la serie de Fibonacci

function fibonacci(n) {

let a = 0, b = 1, temp;

while (n > 0){

temp = a+b

a = b;

b = temp;

n--;

}

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

console.log(fibonacci(7)); // Debería imprimir: 13

Desde Javascript ES6, las funciones soportan los llamados parámetros REST.

Los parámetros REST son un conjunto de parámetros que se almacenan como array en un “parámetro final” nomenclado con **...nombreParametro**. Esto nos permite manejar la función sin tener que controlar el número de parámetros con los que esta es llamada.

**Importante: sólo el último parámetro puede ser REST.**

**Ejemplo:**

|  |
| --- |
| **function** **pruebaParREST**(a, b, ...masParametros) {  console.log("a: "+a+" b: "+ b + " otros: " + masParametros); } pruebaParREST("param1", "param2", "param3", "param4", "param5"); |

Para saber más de los parámetros REST

<https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/parametros_rest>

**Ejercicio 4.** Realiza un programa en JS que calcule la letra del NIF de un DNI que pediremos por consola utilizando una función que la devuelva la letra.

# Funciones flecha (arrow functions)

Una función flecha (arrow function) es una alternativa compacta al uso de funciones tradicionales.

* Este tipo de funciones tienen sus limitaciones y deben ser utilizadas solo en algunos contextos donde sean útiles.
* No son adecuadas para ser utilizadas como métodos.

Soporta varias formas de sintaxis, a continuación indicamos las más típicas:

* (parametro1, parametro2, ...) => {sentencias}
* () => {sentencias}
* parámetro => sentencia

A veces, son utilizados con la **función “map” de los arrays**. Esta función crea un nuevo array formado por la aplicación de una función a cada uno de sus elementos:

<https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/map>

También es muy útil con la **función “reduce” de los arrays**. Esta función ejecuta una “función reductora” sobre cada elemento del array, devolviendo un único valor.

<https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/reduce>

**Ejemplo:**

|  |
| --- |
| ***let*** *nombres = ['Pedro', 'Juan', 'Elena']; console.log(nombres.map(nom => nom.length)); // Muestra el array con los valores [5, 4, 5]* |

Para saber más:

<https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/Arrow_functions>

# Clases en Javascript

Javascript ES6 permite una mejor definición de clases que en anteriores versiones de Javascript.

Mediante la palabra reservada “class”, permite definir clases, métodos, atributos, etc…

Recordad que los objetos en Javascript se guardan como referencias de memoria.

**Ejemplo:**

|  |
| --- |
| class Rectangulo {   / constructor de la clase    constructor(alto, ancho){      this.alto = alto; //atributo clase referido al rectángulo      this.ancho = ancho; //atributo de la clase    };   / método de la clase    area(){      return this.alto \* this.ancho;    };    };  //. objeto de la clase  let r = new Rectangulo(2,3);  // llamada al método de la clase  console.log(r.area()); *//6* |

Para saber más: <https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Classes>

**Ejercicio 5**

Se ha comenzado a definir una clase llamada Círculo que debe tener un radio y un método para calcular su área. Completa el código de la clase para que funcione correctamente.

class Circulo {

constructor(\_\_\_\_1\_\_\_\_){

this.radio = \_\_\_\_2\_\_\_\_;

};

area(){

return \_\_\_\_3\_\_\_\_;

};

};

let c = new Circulo(4);

console.log(c.area()); // Debería imprimir: 78.53981633974483 (pi \* r^2)

**Ejercicio 6 – Modelo clases anidadas**

Dado el siguiente fragmento de código, completa los huecos para que la clase Libro tenga atributos que representen el título, autor y año de publicación del libro. Además, se debe crear una clase Biblioteca que almacene libros.

class Libro {

constructor(\_\_\_\_1\_\_\_\_, \_\_\_\_2\_\_\_\_, \_\_\_\_3\_\_\_\_) {

this.\_\_\_\_1\_\_\_\_ = \_\_\_\_1\_\_\_\_;

this.\_\_\_\_2\_\_\_\_ = \_\_\_\_2\_\_\_\_;

this.\_\_\_\_3\_\_\_\_ = \_\_\_\_3\_\_\_\_;

}

info() {

return this.titulo + " escrito por "' + this.autor + " en el año " + this.anioPublicacion;

}

}

class Biblioteca {

constructor() {

this.libros = [];

}

añadirLibro(libro) {

this.libros[this.libros.lenght+1] = libro;

}

}

const libro1 = new Libro("El principito", "Antoine de Saint-Exupéry", 1943);

const libro2 = new Libro("1984", "George Orwell", 1949);

const biblioteca = new Biblioteca();

biblioteca.añadirLibro(libro1);

biblioteca.añadirLibro(libro2);

Ejercicio 7

Diseña una clase “Colegio”. Dicha clase tendrá como atributos “nombre”, numeroAulas” y “numeroAlumnos”. Cada alumno se representará como un objeto de la clase “Alumno”. En ella se guardaran los atributos “DNI”, “nombre” y “notaMedia”.

Implementa métodos en Colegio y Alumno para modificar la nota media.

Ejercicio 8

Diseña una clase “Aeropuerto”. Tendrá como atributos “nombre”, “ciudad” y “numeroVuelosDiarios”. Cada vuelo diario se representará como una objeto de la clase “Vuelo”. En ella se guardaran los atributos “codigo”, “hora\_llegada” y “hora\_salida”.

Implementa métodos en aeropuerto y vuelo para modificar la hora de llegada, para modificar la hora de salida y para comprobar si la hora de salida es posterior a la hora de llegada.

Ejercicio 9

Diseña una clase “Hospital”. Tendrá como atributos “nombre”, “ciudad” y “numPacientes” (número de pacientes ingresados). Cada paciente se representará como un objeto de la clase “Paciente”. En ella se guardaran los atributos “DNI”, “nombre”, “enfermedad”.

Implementa un método en “Hospital” que reciba el código de paciente y se le de alta a dicho paciente (equivale a eliminar al paciente).

# Más cosas interesantes sobre la sintaxis de Javascript ES6

Este documento pretende ser una simple guía y no podemos incluir todas las novedades de Javascript ES6, pero desde aquí enlazamos las más interesantes por si queréis ampliar:

* Iteradores y generadores
  + <https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Iterators_and_Generators>
* Operadores de propagación (spread)
  + <https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Operadores/Spread_operator>
* Plantillas de cadenas de texto (template strings)
  + <https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/template_strings>
* For … of
  + <https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/for...of>
* Tail Call Optimization (hace más seguras funciones recursivas)
  + <https://hackernoon.com/es6-tail-call-optimization-43f545d2f68b>

Aquí tenéis otras referencias a nuevas características de Javascript ES6

<https://github.com/lukehoban/es6features>

# 

# Material adicional

[1] Curso de Git en Udacity

<https://www.udacity.com/course/how-to-use-git-and-github--ud775>

[2] Uso de la “Consola Web”

<https://developer.mozilla.org/en-US/docs/Tools/Web_Console>

# Bibliografía

[1] Javascript Mozilla Developer <https://developer.mozilla.org/es/docs/Web/JavaScript>

[2] Javascript ES6 W3C <https://www.w3schools.com/js/js_es6.asp>