JAVASCRIPT

PROGRAMACIÓN AVANZADA

FUNCIONES

- Las estructuras de control, los operadores y todas las utilidades propias de JavaScript, permiten crear scripts sencillos y de mediana complejidad. Sin embargo, para las aplicaciones más complejas son necesarios otros elementos como las funciones.
- Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones. Un script para una tienda de comercio electrónico por ejemplo, tiene que calcular el precio total de los productos varias veces, para añadir los impuestos y los gastos de envío.
- Las funciones son la solución a esto, tanto en JavaScript como en el resto de lenguajes de programación. **Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente**.
- En el siguiente ejemplo, las instrucciones que suman los dos números y muestran un mensaje con el resultado se repiten una y otra vez:

FUNCIONES

```
let resultado;
let numero1 = 3;
let numero2 = 5;
// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
numero1 = 10;
numero2 = 7;
// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
numero1 = 5;
numero2 = 8;
// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
...
```

FUNCIONES

- Es un ejemplo muy sencillo y parece evidente que repetir las mismas instrucciones a lo largo de todo el código no es algo recomendable. La solución que proponen las funciones consiste en extraer las instrucciones que se repiten y sustituirlas por una instrucción del tipo *"en este punto, se ejecutan las instrucciones que se han extraído"*:

```
let resultado;
let numero1 = 3;
let numero2 = 5;
/* En este punto, se llama a la función que suma
2 números y muestra el resultado */
numero1 = 10;
numero2 = 7;
/* En este punto, se llama a la función que suma
2 números y muestra el resultado */
numero1 = 5;
numero2 = 8;
/* En este punto, se llama a la función que suma
2 números y muestra el resultado */
...
```

FUNCIONES

- Para que la solución del ejemplo anterior sea válida, las **instrucciones comunes se tienen que agrupar en una función** a la que se le puedan indicar los números que debe sumar antes de mostrar el mensaje.

1. Crear la función con las instrucciones comunes. Las funciones en JavaScript se definen mediante la palabra reservada `function`, seguida del nombre de la función. Su definición formal es la siguiente:

```
function nombre_funcion() {
...
}
```

- El nombre de la función se utiliza para *llamar* a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las que se les asigna un nombre único para poder utilizarlas dentro del código.
- Después del nombre de la función, se incluyen dos paréntesis y, por último, los símbolos `{` y `}` se utilizan para encerrar todas las instrucciones que pertenecen a la función.

FUNCIONES

```
function suma_y_muestra() {
        resultado = numero1 + numero2;
        alert("El resultado es " + resultado);
}
```

- El nombre de la función se utiliza para *llamar* a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las que se les asigna un nombre único para poder utilizarlas dentro del código.
- Después del nombre de la función, se incluyen dos paréntesis y, por último, los símbolos { y } se utilizan para encerrar todas las instrucciones que pertenecen a la función.
- Una vez creada la función, desde cualquier punto del código se la puede *llamar* para que se ejecuten sus instrucciones (además de *"llamar a la función"*, también se suele utilizar la expresión *"invocar a la función"*).
- La llamada a la función se realiza simplemente indicando su nombre, incluyendo los paréntesis del final y el carácter ; para terminar la instrucción:

FUNCIONES

```
function suma_y_muestra() {
    resultado = numero1 + numero2;
    alert("El resultado es " + resultado);
}
let resultado;
let numero1 = 3;
let numero2 = 5;
suma_y_muestra();
numero1 = 10;
numero2 = 7;
suma_y_muestra();
numero1 = 5;
numero2 = 8;
suma_y_muestra();
...
```

FUNCIONES

- El código del ejemplo anterior es mucho más eficiente que el primer código que se mostró, ya que no existen instrucciones repetidas. Las instrucciones que suman y muestran mensajes se han agrupado bajo una función, lo que permite ejecutarlas en cualquier punto del programa simplemente indicando el nombre de la función.
- Lo único que le falta al ejemplo anterior para funcionar correctamente es poder indicar a la función los números que debe sumar. Cuando se necesitan pasar datos a una función, se utilizan los *"**argumentos**"*, como se explica en la siguiente sección.

FUNCIONES
ARGUMENTOS Y VALORES DE RETORNO

- Las funciones más sencillas no necesitan ninguna información para producir sus resultados. Sin embargo, la mayoría de funciones de las aplicaciones reales deben acceder al valor de algunas variables para producir sus resultados.
- Las **variables que necesitan las funciones** se llaman ***argumentos***.
  1. Definición de la función: antes de que pueda utilizarlos, la función debe indicar cuántos argumentos necesita y cuál es el nombre de cada argumento.
  2. Llamada a la función: al invocar la función, se deben incluir los valores que se le van a pasar. Los argumentos se indican dentro de los paréntesis que van detrás del nombre de la función y se separan con una coma (,).

FUNCIONES
ARGUMENTOS Y VALORES DE RETORNO

- Siguiendo el ejemplo anterior, la función debe indicar que necesita dos argumentos, correspondientes a los dos números que tiene que sumar:

```
1. function suma_y_muestra(primerNumero, segundoNumero) { ... }
```

- A continuación, para **utilizar el valor de los argumentos dentro de la función**, se debe emplear el mismo nombre con el que se definieron los argumentos:

```
function suma_y_muestra(primerNumero, segundoNumero) { ... }
    let resultado = primerNumero + segundoNumero;
    alert("El resultado es " + resultado);
}
```

- Dentro de la función, el valor de la variable primerNumero será igual al primer valor que se le pase a la función y el valor de la variable segundoNumero será igual al segundo valor que se le pasa. Para pasar valores a la función, se incluyen dentro de los paréntesis utilizados al llamar a la función:

FUNCIONES
ARGUMENTOS Y VALORES DE RETORNO

```
// 1. Definición de la función
function suma_y_muestra(primerNumero, segundoNumero) { ... }
    let resultado = primerNumero + segundoNumero;
    alert("El resultado es " + resultado);
}
// Declaración de las variables
let numero1 = 3;
let numero2 = 5;
// 2. Llamada a la función
suma_y_muestra(numero1, numero2);
```

FUNCIONES
ARGUMENTOS Y VALORES DE RETORNO

- En el código anterior, se debe tener en cuenta que:
  - Aunque casi siempre se utilizan variables para pasar los datos a la función, se podría haber utilizado directamente el valor de esas variables:

    ```
    suma_y_muestra(3, 5);
    ```
  - El número de argumentos que se pasa a una función debería ser el mismo que el número de argumentos que ha indicado la función. No obstante, JavaScript no muestra ningún error si se pasan más o menos argumentos de los necesarios.
  - El **orden de los argumentos es fundamental**.
  - Se puede utilizar un número ilimitado de argumentos, aunque si su número es muy grande, se complica en exceso la llamada a la función.
  - No es obligatorio que coincida el nombre de los argumentos que utiliza la función y el nombre de los argumentos que se le pasan.

FUNCIONES
ARGUMENTOS Y VALORES DE RETORNO

```
// Definición de la función
function calculaPrecioTotal(precio) {
    let impuestos = 1.16;
    let gastosEnvio = 10;
    let precioTotal = ( precio * impuestos ) + gastosEnvio;
}
// Llamada a la función
calculaPrecioTotal(23.34);
```

- La función toma como argumento una variable llamada precio y le suma los impuestos y los gastos de envío para obtener el precio total. Al llamar a la función, se pasa directamente el valor del precio mediante el número 23.34.

FUNCIONES
ARGUMENTOS Y VALORES DE RETORNO

- Sería más útil si la función devolviera el resultado obtenido para guardarlo en otra variable y poder seguir trabajando con este precio total:
- Las funciones, además de recibir variables y datos, también pueden devolver los valores que han calculado. Para devolver valores dentro de una función, se utiliza la palabra reservada **return**.
- Aunque las funciones pueden devolver valores de cualquier tipo, **solamente pueden devolver un valor cada vez que se ejecutan**.

```
function calculaPrecioTotal(precio) {
    let impuestos = 1.16;
    let gastosEnvio = 10;
    let precioTotal = ( precio * impuestos ) + gastosEnvio;
    return precioTotal;
}
let precioTotal = calculaPrecioTotal(23.34);
// Seguir trabajando con la variable "precioTotal"
```

FUNCIONES
ARGUMENTOS Y VALORES DE RETORNO

- Para que la función devuelva un valor, solamente es necesario escribir la palabra reservada **return junto con el nombre de la variable que se quiere devolver**.
- Como la función devuelve un valor, **en el punto en el que se realiza la llamada, debe indicarse el nombre de una variable en el que se guarda el valor devuelto**.
- Si no se indica el nombre de ninguna variable, JavaScript no muestra ningún error y el valor devuelto por la función se pierde.
- Tampoco es obligatorio que el nombre de la variable devuelta por la función coincida con el nombre de la variable en la que se va a almacenar ese valor.
- Si la función llega a una instrucción de tipo return, se devuelve el valor indicado y finaliza la ejecución de la función. Por tanto, **todas las instrucciones que se incluyen después de un return se ignoran** y por ese motivo la instrucción return suele ser la última de la mayoría de funciones.

FUNCIONES
ARGUMENTOS Y VALORES DE RETORNO

```
function calculaPrecioTotal(precio, porcentajeImpuestos) {
    let gastosEnvio = 10;
    let precioConImpuestos = (1 + porcentajeImpuestos/100) * precio;
    let precioTotal = precioConImpuestos + gastosEnvio;
    return precioTotal;
}
let precioTotal = calculaPrecioTotal(23.34, 16);
let otroPrecioTotal = calculaPrecioTotal(15.20, 4);
```

- Para terminar de completar el ejercicio, se puede redondear a dos decimales el precio total devuelto por la función:

```
function calculaPrecioTotal(precio, porcentajeImpuestos) {
    let gastosEnvio = 10;
    let precioConImpuestos = (1 + porcentajeImpuestos/100) * precio;
    let precioTotal = precioConImpuestos + gastosEnvio;
    return precioTotal.toFixed(2);
}
```

Ejercicios Programación Avanzada 1, 2 y 3.

FUNCIONES
TIPOS

- Las funciones tienen multitud de variantes:
  1. Se pueden escribir **funciones sin parámetros.**
  2. Las **funciones** pueden devolver un dato o no.
  3. Funciones por expresión.
  4. Se puede escribir una función **sin paréntesis** y guardarla en una variable.
  5. Se puede escribir una **función sin nombre** (función anónima).

FUNCIONES TIPOS

3. Funciones por expresión

```
const saludo = function saludar() {
  return "Hola";
};

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

- Código donde se «guardan funciones» dentro de variables, para posteriormente «ejecutar las variables». Se crea una función **en el interior de una variable**, lo que permite posteriormente ejecutar la variable (*como si fuera una función*).

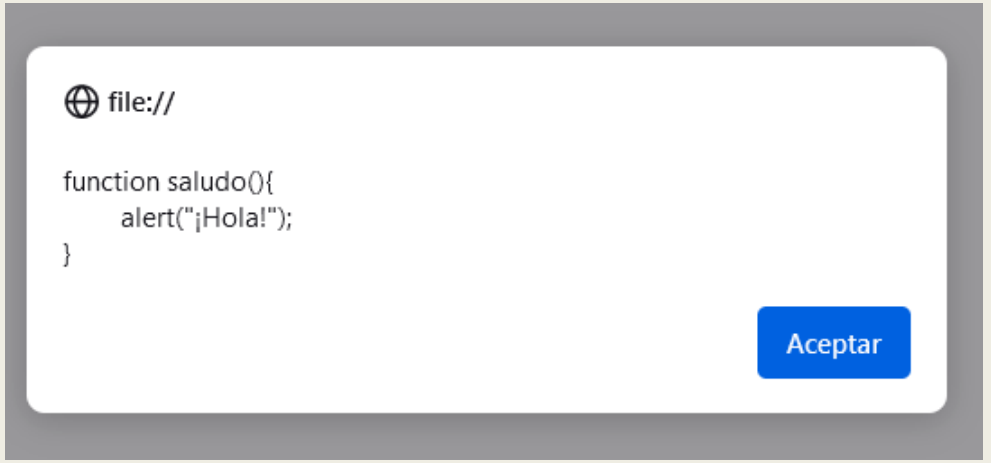
- El nombre de la función (en este ejemplo: saludar) pasa a ser inútil, ya que si intentamos ejecutar saludar() nos dirá que no existe y si intentamos ejecutar saludo() funciona correctamente. Ahora el nombre de la función pasa a ser el nombre de la variable, mientras que el nombre de la función desaparece y se omite, dando paso a lo que se llaman las **funciones anónimas** (*o funciones lambda*).

FUNCIONES TIPOS

4. Escribir una función sin paréntesis y guardarla en una variable.

```
function saludo(){
    alert(“¡Hola!”);
}

let reciboSaludo = saludo;
console.log(reciboSaludo);
//sin paréntesis se puede guardar una función en una variable
```



FUNCIONES
TIPOS

5. Funciones anónimas: se invocan usando el nombre de la variable a la que se asignan:

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

let producto = function (a,b) { return (a*b); }
let resultado = producto(3,6);
alert (resultado);
/*así, la variable producto  puede usarse directamente como si fuera una función*/
```

FUNCIONES
TIPOS

5. Funciones anónimas: callbacks.

- Una vez que se conocen las funciones anónimas, se puede comprender más fácilmente cómo utilizar ***callbacks*** (también llamadas funciones callback o retrollamadas).
- 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 funcionB = function () {
  console.log("Función B ejecutada.");
  };

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

  funcionA(funcionB);
```

FUNCIONES
PARÁMETROS POR DEFECTO

- Desde la versión de ES6, es posible definir **parámetros por defecto** en una función. Los **parámetros por defecto** permiten que los parámetros formales (los que ponemos en la definición de la función) sean inicializados con valores por defecto en caso de que, cuando llamemos a la función, no le pasemos esos valores o estos sean no definidos (undefined).
- La sintaxis de una **función con parámetros por defecto** es muy sencilla: solo hay que indicar el valor del parámetro entre paréntesis:

```
function [nombre]([param1[ = valorPorDefecto1 ][, ..., paramN[ = valorPorDefectoN ]]])
 { sentencias; }
```

FUNCIONES PARÁMETROS POR DEFECTO

```
/* PARÁMETROS POR DEFECTO */

/* Los parámetros por defecto de una función permiten que los parámetros formales sean inicializados con valores por defecto. Si no se pasan valores o los valores pasados son undefined se utilizan los de por defecto. */

function multiplicacion2 (a, b=1){

        return a * b;

}

console.log(multiplicacion2(9));
```

FUNCIONES PARÁMETROS REST

- La sintaxis de los **parámetros rest** permite representar un **número indefinido de parámetros**. Dentro de la función se tratan como un array.

```
function miLista2(a, b, ...otrosElementos){
  console.log("REST");
  console.log("a="+a);
  console.log("b="+b);
  console.log("Resto de elementos de la lista: ", otrosElementos);
  console.log("Tamaño: "+otrosElementos.length);
  console.log(otrosElementos[2]);
}
miLista2("pera", "manzana", "zanahoria", "melón", "limón", "lima");
```

FUNCIONES PARÁMETROS REST

```
function sum(...theArgs) {
        let total = 0;
        for (let arg of theArgs) {
            total += arg;
        }
        return total;
}
console.log(sum(1, 2, 3));// Expected output: 6
console.log(sum(1, 2, 3, 4));// Expected output: 10
```