DESARROLLO WEB EN ENTORNO CLIENTE

UD4 – FUNCIONES

CICLO FORMATIVO DE GRADO SUPERIOR EN DESARROLLO DE APLICACIONES WEB

I.E.S. HERMENEGILDO LANZ – 2022/2023

PROFESORA: VANESA ESPÍN

vespin@ieshlanz.es

Índice

FUNCIONES EN JAVASCRIPT

- Una función en Javascript puede devolver un valor o no hacerlo.
- Sintaxis: Sin devolver valor

```
function nombreFunción ([parámetro1],....,[parámetroN])
{
     // cuerpo de la función
}
```

- Sintaxis: Devolviendo valor

```
function nombreFunción ([parámetro1],....,[parámetroN])
{
     // cuerpo de la función
        return valor;
}
```

FUNCIONES EN JAVASCRIPT

- Usar **nombres de función** que realmente identifiquen lo que hace (ideal: verbo+nombre). Ejemplo de nombre de función: *chequearMail*

- **Llamada a una función:**

```
nombreFunción ( );
```

```
nombreFunción (par1, …, parN);
```

```
variable=nombreFunción ( );
```

- Las funciones en JavaScript también **son objetos**, y como tal tienen métodos y propiedades. Un método, aplicable a cualquier función puede ser *toString()*, que devolverá el código fuente de esa función.

PARÁMETROS

- También conocidos como argumentos

```
function saludar(a,b)
{
     alert("Hola " + a + " y "+ b +".");
}
```

Llamada:

```
saludar ("Martin", "Beatriz");
```

```
function devolverMayor(a,b){
     if (a > b) then
          return a;
     else
          return b;
}
```

Llamada:

```
document.write ("El número mayor entre 35 y
21 es el: " + devolverMayor(35,21) + ".");
```

Otra llamada:

```
let mayor = devolverMayor(35,21);
```

Ámbito de las variables

- **Variables globales**: se definen fuera de la función.
  - Su alcance se limita al documento actualmente cargado en navegador o frame.
  - Todas las instrucciones de nuestro script tendrán podrán leerla o modificarla
- **Variables locales**: de definen dentro de la función (debemos usar var, let o const, porque si no, será global).
  - Su alcance es solo dentro de la función.
  - OJO: no usar nombres de variables locales que ya tengamos definidas como globales puesto que será fuente de errores imprevistos.

ejemplo

```
/* Uso de variables locales y globales no muy recomendable,
 ya que estamos empleando el mismo nombre de variable en global y en local.*/
var  chica  =  "Aurora";                          // variable global
var  perros  =  "Lacky, Samba y Ronda";        // variable global
function  demo()
{
    //  Definimos una variable local (fíjate que es obligatorio para las variables locales usar var, let, const)
    //  Esta variable local tendrá el mismo nombre que otra variable global pero con distinto contenido.
    //  Si no usáramos var estaríamos modificando la variable global chica.
    let  chica  =  "Raquel";    // variable local
    document.write( "<br/>"+ perros +"  pertenecen a "+ chica +".");
}
// Llamamos a la función para que use las variables locales.
demo();
//  Utilizamos las variables globales definidas al comienzo.
document.write( "<br/>"+ perros +"  pertenecen a "+ chica +".");
```

¿Dan el mismo resultado?

Paso por valor y por referencia

- Si pasamos una variable (no objeto) a una función como parámetro, recogemos una copia de su valor. La original no se modifica.
- En JavaScript:
  - los **tipos básicos** (boolean, numero, string) se pasan **por valor** (se envía una copia a los parámetros).
  - los **tipos complejos** (objetos, arrays, conjuntos, mapas) se pasan por referencia.

```
var x=19;
function incrementa(x){
        x++;
}
incrementa(x);
console.log(x); //Devuelve 19

//Lo mismo pasa en:
function incrementa2(y){
        y++;
}
incrementa2(x);
console.log(x); //Devuelve 19

//Aqui modificamos la variable global
function incrementa3(){
        x++;
}
incrementa3(x);
console.log(x); //Devuelve 20
```

Paso por valor y por referencia

- Si pasamos una variable (no objeto) a una función como parámetro, recogemos una copia de su valor. La original no se modifica.
- En JavaScript:
  - los **tipos básicos** (boolean, numero, string) se pasan **por valor** (se envia una copia a los parámetros).
  - los **tipos complejos** (objetos, arrays, conjuntos, mapas) se pasan por referencia.

```
var array=[1,2,3,4,5];
//El array es pasado por referencia
function cambiar(a){
        a[0]=9;
}
cambiar(array);
console.log(array[0]); //Valdrá 9. Se modifica
```

Funciones anidadas

- Los navegadores más modernos nos permitirán **programar una función dentro de otra función**.
- Las funciones anidadas en otras (llamadas *internas*) solo son **accesibles** desde la función *principal.*

```
function principalA()
{
// instrucciones
    function internaA1()
    {
        // instrucciones
    }
// instrucciones
}
function principalB()
{
    // instrucciones
    function internaB1()
    {
        // instrucciones
    }
    function internaB2()
    {
        // instrucciones
    }
    // instrucciones
}
```

ejemplo

Funciones anidadas

Una buena opción para funciones anidadas, es cuando tenemos una secuencia de instrucciones que necesitan ser **llamadas desde múltiples sitios dentro de una función**, y esas instrucciones **sólo tienen significado dentro del contexto** de esa función principal.

```
function hipotenusa(a, b)
{
  function cuadrado(x)
   {
      return x*x;
   }
  return Math.sqrt(cuadrado(a) + cuadrado(b));
}
document.write("<br/>La hipotenusa de 1 y 2 es: "+hipotenusa(1,2));
// Imprimirá: La hipotenusa de 1 y 2 es: 2.23606797749979
```

Funciones anónimas

- Son funciones sin nombre.

```
const square = function (number) {
  return number * number;
};
var x = square(4); // x obtiene el valor 16
```

Funciones Flecha

https://www.w3schools.com/js/js_arrow_function.asp
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions/Arrow_functions

- Es una **alternativa compacta** a una expresión de función tradicional.
- Se introduce en ES6 pero es limitada y no se puede utilizar en todas las situaciones.

Before Arrow:

```
hello = function() {
  return "Hello World!";
}
```

With Arrow Function:

```
hello = () => {
  return "Hello World!";
}
```

Se suelen usar con funciones sencillas y siempre son anónimas

- ¡Se hace más corto! Si la función tiene **solo una declaración** y la declaración **devuelve un valor**, puedo eliminar las llaves y la palabra clave return:

Arrow Functions Return Value by Default:

```
hello = () => "Hello World!";
```

Funciones Flecha

- Si la **función tiene parámetros**, los paso entre los paréntesis:

Arrow Function With Parameters:

```
hello = (val) => "Hello " + val;
```

- Aunque si **SOLO tengo una parámetro**, puedo quitar los paréntesis

Arrow Function Without Parentheses:

```
hello = val => "Hello " + val;
```

Desglose de la función flecha con 1 argumento.

```
// Función tradicional
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.
(a) => {
  return a + 100;
}

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

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

Función flecha con varios o ningún argumento

En este caso debemos **dejar los paréntesis**

```
// Función tradicional
function (a, b){
  return a + b + 100;
}

// Función flecha
(a, b) => a + b + 100;
```

```
// Función tradicional (sin argumentos)
let a = 4;
let b = 2;
function (){
  return a + b + 100;
}

// Función flecha (sin argumentos)
let a = 4;
let b = 2;
() => a + b + 100;
```

Función flecha con más instrucciones o return

En este caso debemos **dejar las llaves y el return**

```
// Función tradicional
function (a, b){
  let chuck = 42;
  return a + b + chuck;
}

// Función flecha
(a, b) => {
  let chuck = 42;
  return a + b + chuck;
}
```

Función flecha en funciones con nombre

En este caso tratamos las expresiones de flecha **como variables**

```
// Función tradicional
function bob (a){
  return a + 100;
}

// Función flecha
let bob = a => a + 100;
```

Funciones Callback

https://www.w3schools.com/js/js_callback.asp
https://developer.mozilla.org/es/docs/Glossary/Callback_function

- Una función de callback es una **función que se pasa a otra función como un argumento**, que luego se invoca dentro de la función externa para completar algún tipo de rutina o acción.

Ejemplo de función callback síncrona:

```
function saludar(nombre) {
  alert('Hola ' + nombre);
}

function procesarEntradaUsuario(callback) {
  var nombre = prompt('Por favor ingresa tu nombre.');
  callback(nombre);
}

procesarEntradaUsuario(saludar);
```

Funciones Callback

https://www.w3schools.com/js/js_callback.asp
https://developer.mozilla.org/es/docs/Glossary/Callback_function

- Los callbacks aseguran que una función no se va a ejecutar antes de que se complete una tarea, sino justo después de que la tarea se haya completado. Nos ayuda a desarrollar **código JavaScript asíncrono** y nos mantiene a salvo de problemas y errores.

```
setTimeout( FUNCTION callback, NUMBER time).
```

- Parámetros del ejemplo:
  - La función *callback* a ejecutar
  - El tiempo *time* que esperará antes de ejecutarla

```
//setTimeout llama a la función callback cuando pasen dos segundos
setTimeout(function() {
  console.log("He ejecutado la función");
}, 2000);

//Podemos acortarlo con arrow function
setTimeout(() => console.log("He ejecutado la función"), 2000);
```

La función SetInterval

- Ofrecido en las interfaces Window y Worker, llama a una función o ejecuta un fragmento de código de forma reiterada, con un retardo de tiempo fijo entre cada llamada.
- El intervalo se puede eliminar con la función **ClearInterval()**;
- Sintaxis

```
setInterval(code);
setInterval(code, delay);


setInterval(func);
setInterval(func, delay, arg0);
setInterval(func, delay, arg0, arg1);
setInterval(func, delay, arg0, arg1, /* ..., */ argN);
```

La función devuelve un identificador de temporizador para poder eliminarlo con ClearInterval (id).

La función SetInterval. Parámetros

- ***code*** → permite incluir una cadena en lugar de una función, la cual es compilada y ejecutada cada *delay* milisegundos. Por seguridad, se recomienda no usar esta sintaxis.
- ***func*** → Una función que se ejecuta cada cierto tiempo. El tiempo lo determina *delay*, estando éste en milisegundos. La primera ejecución ocurre tras el tiempo determinado por *delay*.
- ***delay*** Opcional → tiempo en milisegundos que el temporizador debe retrasar cada ejecución de la función o código especificado. Si no se especifica ninguno, por defecto es 0.
- ***arg0, ..., argN*** Opcional → argumentos adicionales que se pasan a la función especificada por func una vez que el temporizador expira.

La función SetInterval. Ejemplo

```
var intervalID = setInterval(myCallback, 500, 'parámetro 1', 'parámetro 2');

function myCallback(a, b) {
    // Tu código debe ir aquí
    // Los parámetros son totalmente opcionales
    console.log(a);
    console.log(b);
}

console.log("He terminado");

setTimeout(() => {
    clearInterval(intervalID);
    console.log("FINAL DEL PROGRAMA");
  }, "5000");
```

Probar este código y ver su funcionamiento

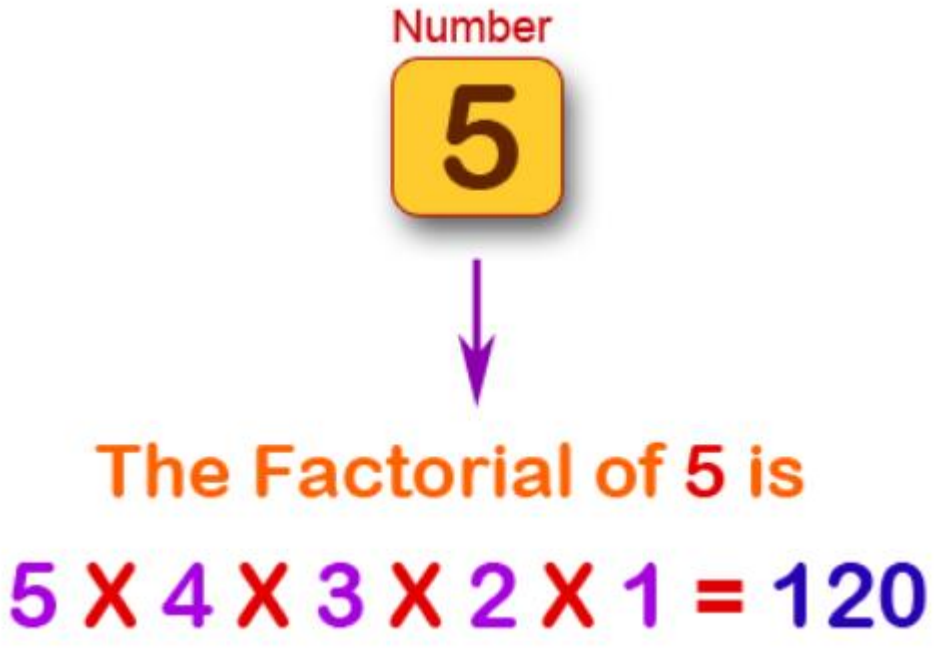
Volveremos a estas funciones más adelante, cuando veamos la programación ASÍNCRONA

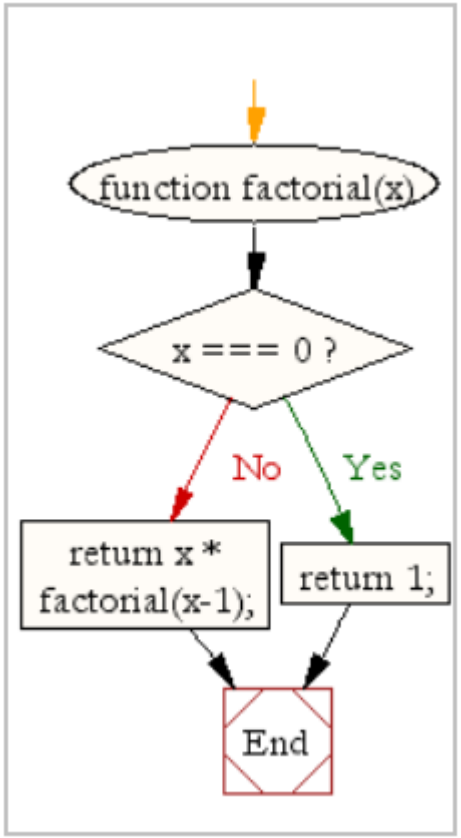
Recursividad

- Javascript también permite las llamadas recursivas

```
function factorial(x)
{
 if (x === 0)
 {
   return 1;
 }
 return x * factorial(x-1);

}
console.log(factorial(5));
```





Documentación de funciones. Estilo JSDoc

- Las funciones se documentan justo sobre su definición.
- La descripción debe ir entre /** **/
- Primero se indica qué hace la función y después se describen los parámetros
- Cada parámetro debe incluir una línea como la siguiente

```
* @param {param-type} param-name -description
```

```
* @param {String} nombre -Nombre de la persona a saludar
```

- Si la función devuelve algún elemento se añade la siguiente línea

```
@returns {return-type} -description
```

```
@returns {Object} –Lista de usuarios
```

Documentación de funciones

- Podemos describir al autor. `@autor Vanesa Espín`
- Ejemplo de documentación de función

```
/**
 * It is used to recommend related products for a specific product.
 * @author Rafael Granados <rafael.granados@gradiweb.com>
 * @param {String} id - Product ID.
 * @param {Number} limit - Limits the number of results.
 * @returns {Object} Product Recommendations.
 */
async getRecommendedProducts(id, limit) {
```

- Al escribir /** y pulsar INTRO en VS Code te genera una plantilla de descripción de tu función.