UD4 – FUNCIONES

1. Introducción a las Funciones

¿Qué es una función?

Una función es un **grupo de instrucciones** que tiene un **nombre identificativo**, puede recibir **valores de entrada** (argumentos) y devolver un **valor de salida**.

Ventajas de usar funciones:

- ✅ **Dividir programas complejos** en partes más pequeñas
- ✅ **Facilitar la programación y depuración**
- ✅ **Reutilizar código** en distintos programas
- ✅ **Mejorar la eficiencia y legibilidad**
- ✅ **Reducir costes de mantenimiento**

---

2. Tipos de Funciones en JavaScript

A. Funciones Predefinidas

Funciones que vienen incluidas por defecto en JavaScript.

Ejemplos:

```
alert();
document.write();
Number();
prompt();
```

B. Funciones Definidas por el Usuario

Funciones creadas por el programador para adaptarse a sus necesidades.

---

3. Definición de una Función

Estructura básica:

```
function calcularCuadrado(numero) {
    let resultado = numero ** 2;
    return resultado;
}
```

Componentes:

1. **Palabra clave** `function`
2. **Identificador** (nombre de la función)
3. **Parámetros** (datos de entrada)
4. **Cuerpo** (instrucciones entre llaves {})
5. **Valor de retorno** (opcional, con `return`)

---

4. Invocación de una Función

Llamada básica:

```
mensajeAlerta();  // Sin parámetros
raizCuadrada(4); // Con parámetros
```

Funciones con valor de retorno:

```
function raizCuadrada(numero) {
    return Math.sqrt(numero);
}

console.log(raizCuadrada(4));          // 2
console.log(raizCuadrada(4+5));        // 3
console.log(raizCuadrada(4) + raizCuadrada(9)); // 5
```

---

5. Valor de Retorno

Ejemplo correcto:

```
function esPar(numero) {
    if (numero % 2 == 0)
        return true;
    else
        return false;
}

if (esPar(4))  // ✅ Correcto - devuelve booleano
    console.log("El número es par");
```

Ejemplo incorrecto:

```
console.log(esPar(4) + 5);  // ❌ Incorrecto - suma booleano + número
```

Mejor práctica - un solo return:

```
function esPar(numero) {
    let resultado = false;
    if (numero % 2 == 0)
        resultado = true;
    return resultado;
}
```

---

6. Ámbito de las Variables

Variable Local:

Solo visible dentro de la función donde se declara.

Variable Global:

Visible desde cualquier parte del programa.

Ejemplos:

```
var mensaje = "Fuera de la función";
```

```
function mostrarAnuncio() {
    var mensaje = "Dentro de la función";  // Variable local
    console.log(mensaje);  // "Dentro de la función"
}

mostrarAnuncio();
console.log(mensaje);  // "Fuera de la función"
```

⚠️ **Recomendación:** Evitar variables globales porque son difíciles de depurar.

---

7. Parámetros

Parámetros por Valor:

```
var numero1 = 7;
var numero2 = 8;

function menor(primero, segundo) {
    primero = 10;    // Solo modifica la copia local
    segundo = 21;    // No afecta las variables originales
    return primero < segundo ? primero : segundo;
}

console.log(menor(numero1, numero2));  // 10
console.log(numero1);  // 7 (no se modifica)
console.log(numero2);  // 8 (no se modifica)
```

Parámetros por Referencia (con arrays):

```
let sinIVA = [20.45, 39.95, 6.69];
let conIVA = sinIVA;          // Ambas variables apuntan al mismo array
conIVA[0] = 110.25;           // Modifica el array original

console.log(sinIVA);  // [110.25, 39.95, 6.69]
console.log(conIVA);  // [110.25, 39.95, 6.69]
```

Parámetros por Defecto:

```
function dividir(numerador, denominador = 1) {
    return numerador / denominador;
}

console.log(dividir(4));      // 4 (4/1)
console.log(dividir(4, 2));   // 2 (4/2)
console.log(dividir());       // NaN (undefined/1)
```

Parámetros Variables:

Con `arguments`:

```
function sumaTodo() {
    let sum = 0;
    for (let i = 0; i < arguments.length; i++)
        sum += arguments[i];
    return sum;
}

console.log(sumaTodo(11, 22, 33, 44, 55));  // 165
```

Con operador spread (`...`):

```
function sumaTodo(...parameters) {
    let sum = 0;
    for (let i = 0; i < parameters.length; i++)
        sum += parameters[i];
    return sum;
}

console.log(sumaTodo(11, 22, 33, 44, 55));  // 165
```

---

8. Tipos de Funciones

1. Funciones por Declaración:

```
let resultado = multiplicar(7, 5);  // ✅ Se puede llamar antes de definir

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

console.log(resultado);  // 35
```

2. Funciones por Expresión:

```
const bienvenido = function sesionIniciada() {
    console.log("Bienvenido de nuevo");
};

bienvenido();         // ✅ Correcto
sesionIniciada();     // ❌ Error - no definido
```

3. Funciones como Objetos:

```
const bienvenido = new Function("console.log('Bienvenido de nuevo');");
bienvenido();
```

4. Funciones Anónimas:

```
const bienvenido = function() {
    console.log("Bienvenido de nuevo");
};
bienvenido();
```

5. Callbacks:

```
// Opción 1: Con variable
const bienvenido = function() {
    return "Bienvenido de nuevo, ";
};
```

```
const usuario = function(callback) {
    console.log(callback() + "Javier");
};

usuario(bienvenido);

// Opción 2: Directamente
const usuario = function(callback) {
    console.log(callback() + "Javier");
};

usuario(function() {
    return "Bienvenido de nuevo, ";
});
```

6. Funciones Autoejecutables:

```
// Sin parámetros
(function() {
    console.log("Bienvenido de nuevo");
})();

// Con parámetros
(function(usuario) {
    console.log("Bienvenido de nuevo, " + usuario);
})("Javier");

// Asignada a variable
const variable = (function(usuario) {
    return "Bienvenido de nuevo, " + usuario;
})("Javier");

console.log(variable);
```

7. Funciones Flecha:

```
// Forma básica
const mensaje = () => {
   console.log("Hola de nuevo.");
};

// Una sola línea (sin llaves)
const mensaje = () => console.log("Hola de nuevo.");

// Con return implícito
const cuadrado = (x) => x * x;

// Un solo parámetro (sin paréntesis)
const mensaje = nombre => console.log("Hola de nuevo, " + nombre);

// Múltiples parámetros
const sumar = (a, b) => a + b;
```

Ventajas de funciones flecha:

- ✅ Código más conciso
- ✅ `return` implícito en una línea
- ✅ Sin paréntesis para un solo parámetro
- ✅ Mejor legibilidad
- ✅ Mayor productividad

---

9. Recursividad

La capacidad de una función de **llamarse a sí misma**.

⚠️ Precaución:

Puede generar llamadas infinitas y desbordar la pila del sistema.

Estructura:

```
Caso base + Caso recursivo
```

Ejemplo: Factorial

```
function factorial(n) {
    if (n <= 1)
        return 1;                  // Caso base
    else
        return n * factorial(n - 1); // Caso recursivo
}

console.log(factorial(5));  // 120
```

Ejemplo: Serie de Fibonacci

```
function fibo(numero) {
    if (numero < 2)
        return 1;                              // Caso base
    else
        return fibo(numero - 1) + fibo(numero - 2);  // Caso recursivo
}

// Mostrar primeros 10 elementos
for (let i = 0; i < 10; i++)
    console.log(fibo(i));  // 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

⚠️ **Importante:** Las soluciones recursivas son más costosas computacionalmente que las iterativas.

---

10. Otras Funcionalidades Avanzadas

A. Ordenación Avanzada de Arrays

Ordenación básica (problema Unicode):

```
let vector = ["Casado", "casa", "prueba", "zancos", "ñam"];
vector.sort();
// Resultado: ['Casado', 'casa', 'prueba', 'zancos', 'ñam'] (incorrecto)
```

Ordenación por longitud:

```
vector.sort((primera, segunda) ⇒ primera.length - segunda.length);
// Resultado: ['ñam', 'casa', 'Casado', 'prueba', 'zancos']
```

Ordenación en español:

```
vector.sort((primera, segunda) ⇒ primera.localeCompare(segunda, "es"));
// Resultado: ['casa', 'Casado', 'ñam', 'prueba', 'zancos']
```

B. Recorridos con forEach

Con arrays:

```
let vector = [12, 334, 111, 52, 98];
vector.forEach(function(elemento, posicion) {
   console.log(`Posición ${posicion}: ${elemento}`);
});
```

Con conjuntos:

```
let conjunto = new Set();
conjunto.add(12).add(334).add(111).add(52).add(98);
conjunto.forEach(function(elemento) {
   console.log(`Elemento: ${elemento}`);
});
```

Con mapas:

```
let mapa = new Map();
mapa.set('a', 12).set('b', 334).set('c', 111).set('d', 52).set('e', 98);
mapa.forEach(function(valor, clave) {
   console.log(`Clave: ${clave} / Valor: ${valor}`);
});
```

C. Map - Transformar arrays

```
let sinIVA = [12.45, 34.42, 99.90, 49.95];
let conIVA = sinIVA.map(x => (x * 1.21).toFixed(2));
// Resultado: ['15.06', '41.65', '120.88', '60.44']
```

D. Filter - Filtrar arrays

```
let playas = ["Hierbabuena", "Caños", "Zahara", "Carmen", "Palmar"];
let filtradas = playas.filter(elemento => elemento.length != 6);
// Resultado: ['Hierbabuena', 'Caños']
```

📋 Ejemplos Prácticos

1. Función para invertir cadenas

```
let invertida = function(cadena) {
    let resultado = "";
    for (let i = cadena.length - 1; i >= 0; i--) {
        resultado += cadena[i];
    }
    return resultado;
};

let frase = prompt('Introduce una frase para invertirla:');
document.write('La frase inicial es: ' + frase);
document.write('<br>La frase invertida es: ' + invertida(frase));
```

2. Contar vocales con funciones flecha

```
let contarVocales = (cadena) => {
    let numVocales = 0;
    const vocales = ["a", "e", "i", "o", "u"];
    for (let i = 0; i < cadena.length; i++) {
        if (vocales.includes(cadena[i].toLowerCase())) {
            numVocales++;
        }
    }
```

```
    return numVocales;
};

let masVocales = (cadena1, cadena2) => {
    let numVocalesCadena1 = contarVocales(cadena1);
    let numVocalesCadena2 = contarVocales(cadena2);

    // Comparar y mostrar resultados...
};
```

3. Función autoejecutable con fecha

```
(function() {
    let fechaInicio = new Date();
    document.write("La función se inició el " +
                fechaInicio.toLocaleDateString() +
                " a las " +
                fechaInicio.toLocaleTimeString());
})();
```