

# USO AVANZADO DE FUNCIONES

Tema 4

# ÍNDICE

- La pila de funciones
- Recursividad
- Funciones Callback
- Uso de métodos avanzados para manipular estructuras de datos

# LA PILA DE FUNCIONES

- Cuando se invoca a una función en una expresión...
  - esta debe esperar a que la función termine para poder completar la expresión.
- Las funciones utilizan lo que se conoce como **pila de llamadas**.
  - La pila de llamadas tiene un tamaño finito, y puede llegar a llenarse.
    - Desbordamiento de pila → Siguiendo diapositiva
- Ejemplo:

Inicio f1

Inicio f2

En f3

Fin f2

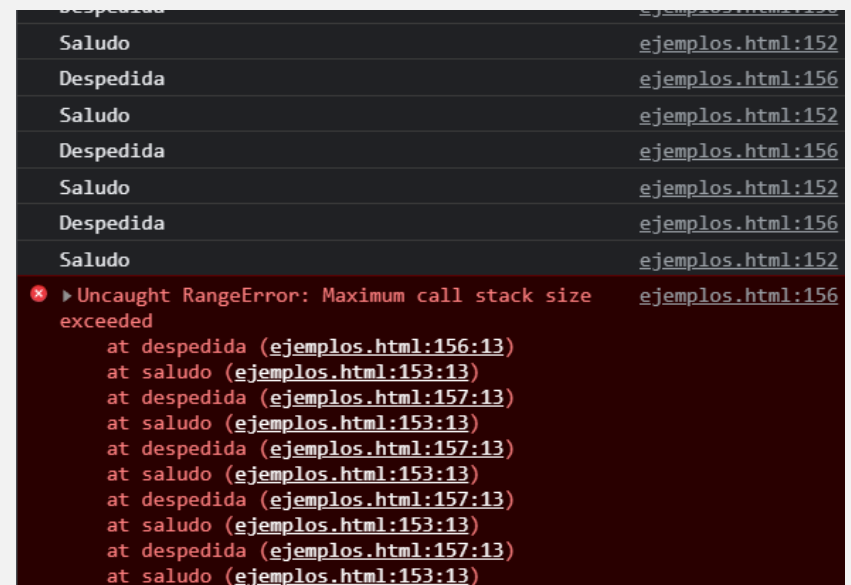
Fin f1

```
function f1() {  
  console.log("Inicio f1");  
  f2();  
  console.log("Fin f1");  
}  
  
function f2() {  
  console.log("Inicio f2");  
  f3();  
  console.log("Fin f2");  
}  
  
function f3() {  
  console.log("En f3");  
}  
  
f1();
```

# DESBORDAMIENTO DE PILA DE LLAMADAS

- Ante una mala gestión de la pila de llamadas, se puede llegar a provocar el famoso **desbordamiento de pila**.
- Tras una serie de llamadas, el propio motor de JavaScript cortará la ejecución de código si detecta que la pila de llamadas se va a llenar.
- Ejemplo:

```
function saludo() {  
    console.log("Saludo");  
    despedida();  
}  
function despedida() {  
    console.log("Despedida");  
    saludo();  
}  
saludo();
```



Despedida	ejemplos.html:156
Saludo	ejemplos.html:152
Despedida	ejemplos.html:156
Saludo	ejemplos.html:152
Despedida	ejemplos.html:156
Saludo	ejemplos.html:152
Despedida	ejemplos.html:156
Saludo	ejemplos.html:152
✖ ▶ Uncaught RangeError: Maximum call stack size exceeded	
at despedida (ejemplos.html:156:13)	
at saludo (ejemplos.html:153:13)	
at despedida (ejemplos.html:157:13)	
at saludo (ejemplos.html:153:13)	
at despedida (ejemplos.html:157:13)	
at saludo (ejemplos.html:153:13)	
at despedida (ejemplos.html:157:13)	
at saludo (ejemplos.html:153:13)	
at despedida (ejemplos.html:157:13)	
at saludo (ejemplos.html:153:13)	

# RECURSIVIDAD

- La pila de funciones de JavaScript soporta la recursividad.
- Técnica de resolución de un problema en la que una función se llama a sí misma para resolver porciones de este.
- En cada llamada el problema debe ser cada vez más sencillo.
- Se acaba llegando a una llamada que devuelve un único valor.
- ES FUNDAMENTAL PREPARAR ESTA ÚLTIMA LLAMADA PARA PODER CERRAR EL BUCLE.
- Ejemplo:



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

# ¿RECURSIVIDAD O ITERACIÓN?

- Hay otra versión del factorial que se podría hacer con un bucle:

```
function factorial(n) {  
  let res=1;  
  while (n>1) {  
    res*=n;  
    n--;  
  }  
  return res;  
}
```

- La pregunta es: ¿cuál es mejor?
  - EN TÉRMINOS DE RENDIMIENTO ES MÁS COSTOSA LA RECURSIVIDAD → MUCHAS LLAMADAS A FUNCIONES → MUCHAS COPIAS DE CÓDIGO DE LA MISMA
  - SI POSEEMOS SOLUCIÓN ITERATIVA → MEJOR NO USAR RECURSIVIDAD
    - SOLO USAR RECURSIVIDAD SI NO ENCONTRAMOS SOLUCIÓN ITERATIVA O EL CÓDIGO ES MUCHO MÁS CLARO QUE USANDO BUCLES

# FUNCIONES CALLBACK

- Las funciones callback son muy usadas en JavaScript.
- Si las funciones se pueden asignar a variables → también se pueden asignar a parámetros de las funciones
- Con esto conseguimos que las funciones ejecuten otras funciones a través de los parámetros
  - Es decir → las funciones pueden recibir datos y acciones a realizar

- En el primer parámetro está el texto a escribir.
  - En el segundo el nombre de la función que se encargará de realizar la escritura.
- Es equivalente a `console.log(dato)`.

Esta página dice

Hola

```
function escribe(dato,funcion) {  
    funcion(dato);  
}  
escribe("Hola", console.log)  
  
escribe("Hola", console.error);  
  
escribe("Hola", alert);
```

ult.net

DevTools is now av

Always match Chrom

Element

top ▾

Hola

✖ ▶ Hola

>

## OTRO EJEMPLO DE FUNCIÓN CALLBACK (CON MÁS SENTIDO)

- En este caso (aunque sigue siendo algo enrevesado) se usa una función callback para calcular el doble de un número:

```
function escribir(x, accion) {  
    console.log(accion(x));  
}  
function doble(y) {  
    return 2*y;  
}  
escribir(12,doble);
```



# FUNCIONES CALLBACK ANÓNIMAS Y FLECHA

- Es común ver las funciones callback como funciones anónimas (en este ejemplo, esa función anónima sería el equivalente en el anterior a lo que asociamos a la función de acción):

```
escribir(12,function(y) {  
    return 2*y;  
});
```

- También es posible definir las funciones de callback como flecha:

```
escribir(12, y=>2*y);
```

- Inicialmente puede parecer complejo, pero muchos objetos las usan, como los temporizadores.

USO DE MÉTODOS AVANZADOS PARA  
MANIPULAR ESTRUCTURAS DE DATOS

# ORDENACIÓN AVANZADA DE ARRAYS

- Cuando usamos el método sort para ordenar arrays, se basa en la tabla Unicode:

```
const palabras=["Ñu", "Águila", "boa", "oso", "marsopa", "Nutria"];  
palabras.sort();  
console.log(palabras); // ["Nutria", "boa", "marsopa", "oso", "Águila", "Ñu"]
```

- La función sort permite incluir una función callback que podemos usar para especificar criterios de ordenación.
  - Número negativo = 1º num < 2º num ;; 0 = son iguales ;; Número positivo = 1º num > 2º num
  - Que aparezcan primero los textos más cortos:

```
function ordenPersonal(a,b) {  
  return a.length-b.length;  
}
```

```
const palabras=["Ñu", "Águila", "boa", "oso", "marsopa", "Nutria"];  
palabras.sort((a,b)=>a.length-b.length);  
console.log(palabras); // ["Ñu","boa","oso","Águila","Nutria","marsopa"]
```

# ORDENACIÓN AVANZADA DETECTANDO LA EÑE

- Para poder ordenar detectando el idioma español, podemos usar el método `localeCompare`:

```
const palabras=["Ñu", "Águila", "boa", "oso", "marsopa", "Nutria"];  
palabras.sort((a,b)=>a.localeCompare(b));  
console.log(palabras); // ["Águila", "boa", "marsopa", "Nutria", "Ñu", "oso"]
```

- Que es recomendable que indiquemos en su segundo parámetro que estamos trabajando con el español:

```
const palabras=["Ñu", "Águila", "boa", "oso", "marsopa", "Nutria"];  
palabras.sort((a,b)=>a.localeCompare(b,"es"));  
console.log(palabras); // ["Águila", "boa", "marsopa", "Nutria", "Ñu", "oso"]
```

# MÉTODO FOREACH

- **Foreach** → Método sofisticado para recorrer arrays, mapas, y conjuntos.
  - A partir de ES2015.

```
nombreArray.forEach(function(elemento, indice) {  
    instrucciones que se repiten por cada elemento del array  
});
```

- **Foreach** requiere indicar una función que necesita dos parámetros:
  - Uno que irá almacenando los **valores** de cada elemento del array
  - Otro que irá almacenando los **índices** → OPCIONAL

# EJEMPLOS FOREACH PARA ARRAY, CONJUNTO, Y MAPA

```
const notas=[5,6,,,9,,8,,9,,7,8];
notas.forEach(function(nota,i) {
  console.log(`La nota ${i} es ${nota}`);
});
```

La nota 0 es 5
La nota 1 es 6
La nota 5 es 9
La nota 7 es 8
La nota 9 es 9
La nota 11 es 7
La nota 12 es 8

```
let conjunto=new Set();
conjunto.add("Paul").add("Ringo").add("George").add("John");
conjunto.forEach(function(valor) {
  console.log(valor);
});
```

Paul
Ringo
George
John

```
const provincias = new Map();
provincias.set(1,"Álava").set(28,"Madrid").set(34,"Palencia").set(41,"Sevilla");
provincias.forEach(function(valor,clave) {
  console.log(`Clave: ${clave}, Valor: ${valor}`);
});
```

Clave: 1, Valor: Álava
Clave: 28, Valor: Madrid
Clave: 34, Valor: Palencia
Clave: 41, Valor: Sevilla

# MÉTODO MAP

- El método map nos permitirá recorrer arrays y...
  - En cada elemento, usando una función callback → establecer un cálculo a este
- Map NO modifica el array → devuelve otro con los mismos elementos
- Ejemplo: queremos doblar el valor de cada elemento de un array

```
const notas=[5,6,,,9,,8,,9,,7,8];  
const doble=notas.map(x=>2*x);  
console.log(doble);
```

```
ejemplos.html:288  
▶ (13) [10, 12, empty × 3, 18, empty, 16, empty, 18, empty, 14, 16]
```

# MÉTODO REDUCE

- Método que también requiere de función callback
- Permite recorrer cada elemento del array
- **Devuelve un cálculo que se realiza en base a todos los elementos.**
- Tiene un segundo parámetro (el primero es la función callback)
  - Que sirve para indicar el valor inicial que tendrá la variable para acumular el resultado final.
- La función callback recibe dos parámetros:
  - 1º acumulador donde se coloca el resultado deseado
  - 2º recoge el valor del elemento del array que se va recorriendo

```
const array=[1,2,3,4,5];  
let suma=array.reduce((acu,valor)=>acu+valor,0);  
console.log(suma);
```

15



# MÉTODO FILTER

- Se utiliza mucho.
- Usa una función callback que recibe un único parámetro.
  - Ese parámetro recoge cada valor del array.
- La función callback retorna una condición que debe cumplir cada elemento.
- Este método obtiene un nuevo array que tendrá como elementos aquellos que cumplan la condición de la función callback.

```
const array=[4,9,2,6,5,7,8,1,10,3];  
const arrayFiltrado=array.filter(x=>x>5);  
console.log(arrayFiltrado); // [9,6,7,8,10]
```

```
► (5) [9, 6, 7, 8, 10]
```

¿PREGUNTAS?

# USO AVANZADO DE FUNCIONES

Tema 4