Práctica: funciones callback

1. sin callback

Programa cuatro funciones para realizar las **operaciones** básicas de:

```
let s = suma(x, y);
let r = resta(x, y);
let m = multiplica(x, y);
let d = divide(x, y);
```

Por ejemplo:

```
function suma(x, y) {
  let resultado = x + y;
  return resultado;
}
```

Por practicar, programa al menos **dos de ellas** como función flecha.

EJERCICIO

- **En una sóla línea**, realiza cada uno de los siguientes cálculos:

```
5 + 4                           // resultado: 9
5 + 4 - 1                       // resultado: 8
( 3 + (2 * 2) + 7 ) / 2         // resultado: 7
( (8 + 6) / (9 - 2) ) * 3       // resultado: 6
```

y muestre los resultados en distintos `<h1>`.

2. Función que recibe 2 operandos y la operación

Crea una función calculadora para realizar las **operaciones** anteriores pasándolas como *callback*: `c = calcula(x, y, operacion)`
por ejemplo:

```
let s = calcula(5, 7, suma);  // s = 12
```

> Las *funciones* (callback) son otros argumentos más, indeferenciados de *números* u *objetos*.
> Ya las has programado en funciones como `addEventListener(evento, callback)`, `setTimeout(ms, callback)`, `setInterval(ms, callback)`, ...
>
> Si lo prefieres, puedes pasar la callback como primer argumento: `calcula(suma, 5, 4)`

EJERCICIO

- Haciendo uso de la función '`calcula`', realiza los siguientes cálculos:

```
5 + 4                           // resultado: 9
5 + 4 - 1                       // resultado: 8
( 3 + (2 * 2) + 7 ) / 2         // resultado: 7
( (8 + 6) / (9 - 2) ) * 3       // resultado: 6
```

muestre los resultados en `<h1>`.

Debes agruparlas para obtener cada resultado **en una sola línea**.
- Por ejemplo, el cálculo de 1 + 2 + 3 sería:

```
calcula(3, calcula(1,2, suma), suma);
```

3. con callback en cada función

Si estos cálculos aritméticos fueran realizados por un servidor que tarda un tiempo en resolver (código asíncrono), en estas situaciones no funcionaría el `return` para devolver un resultado, pues no sabemos cuándo el servidor nos entregará el dato.

Para que podamos hacer sus cálculos el servidor nos proporciona una API como la siguiente:

```
let s = suma(x, y, fc);  `// fc es una función callback`
let r = resta(x, y ,fc);
let m = multiplica(x, y, fc);
let d = divide(x, y, fc);
```

En esta API, cada función recibe un tercer argumento que es una función (callback), y se ejecutará cuando el servidor entregue el resultado.

Esa callback la debemos programar nosotros con lo que deseamos hacer con los datos que nos entregue el servidor, y se la pasaremos a la función con los operandos necesarios.

Por ejemplo

La función "suma" de la API podría ser algo como:

```
function suma(x, y, fc) {
  let resultado = x + y;  // esto debería ser asíncrono y podría tardar en resolverse
  fc(resultado);          // la callback debe recibir como argumento el resultado final
}
```

► Y para mostrar la suma de dos números en consola, podría programar algo como:

```
// función que pasaré como callback: muestra el "resultado" en consola.
function showEnConsola( res ) { console.log(res); }

// Uso la función de la API, paso 2 operandos y la callback que opera sobre el resultado fi
suma(1, 2, showEnConsola);   // observar que pasamos la referencia a la función, NO la ejec
```

Esto puedo simplificarlo programando la callback en el interior, como hacemos con los eventos:

```
// con function
suma(1, 2, function(res) {
    console.log(res);
});
```

```
// con función flecha flecha
suma(1, 2, res => console.log(res) );
```

► Si ahora quisiéramos sumar tres números, podríamos hacer algo como:

```
let showEnConsola = dato => console.log(dato);
suma(1, 2, res => suma(3, res, showEnConsola));
```

Que también podemos simplificar programando las callback en el interior de la función suma:

```
// con function
suma(1, 2, function(res) {
        suma(3, res, function(res02){
```

```
// con función flecha flecha
suma(1, 2, res => suma(3, res, res02 => console.log(
```

```
        console.log(res02)
    })
})
```

EJERCICIO

- Realiza los siguientes cálculos:

```
5 + 4                               // resultado: 9
5 + 4 - 1                           // resultado: 8
( 3 + (2 * 2) + 7 ) / 2             // resultado: 7
( (8 + 6) / (9 - 2) ) * 3           // resultado: 6
```

y muestre los resultados en `<h1>`.

a) Hazlo usando solo function. **b)** Hazlo usando solo funciones flecha.

4. Función asíncrona

Vamos a simular la asincronía, voy a tardar en generar el resultado un tiempo aleatorio entre 0 y 2 segundos.

Modifica todas las funciones para que tengan un código como el siguiente:

```
function suma(x, y, fc) {
  setTimeout( function() {
    let resultado = x + y;
    fc(resultado);
  }, 2000*Math.random())
}
```

Y comprueba que funcionan perfectamente los cálculos que hiciste en el ejercicio anterior.

¿Serviría de algo poner un `return resultado;` en estas funciones?

5. API con callbacks

Hemos descargado una API de un servidor que nos proporciona palabras.

Al ser funciones asíncronas, carecen de `return <valor>`, en su lugar, nos ofrecen invocar una función callback cuando termine de ejecutarse la función: `fc(<valor>)`.

Invócalas correctamente, de manera que en la página web se escriba la frase "*En un lugar de la Mancha*".

```
/* tiempo de respuesta del servidor (simulación) */
const t = () => [0,1,2,4,8,16,32,64,128,256,512,1024,2048,4096][parseInt(14*Math.random())]

/* === API proporcionada por el servidor, para uso del cliente === */
function en(fc) {
   setTimeout(() => fc("En"), t());
}

function un(fc) {
   setTimeout(() => fc("un"), t());
}
```

```
function lugar(fc) {
   setTimeout(() => fc("lugar"), t());
}

function de(fc) {
   setTimeout(() => fc("de"), t());
}

function la(fc) {
   setTimeout(() => fc("la"), t());
}

function mancha(fc) {
   setTimeout(       // Esto no debería existir aquí (en el cliente) es simplemente para
      function() {                                                    //  simular la asincroní
          // ...        Aquí el servidor estaría haciendo cosas...
          fc("Mancha")  // callback que se ejecutará cuando expire el temporizador t()
      },
      t()               // tiempo aleatorio, entre 0 y 4 segundos
   );
}
/* === fin API === */
```

Para solucionarlo, debes invocar la función "`en( fcallback_1 )`", de manera que en la callback se invoque "`un( callback_2 )`", and so on...

Observa que la función callback de la API (`fc`) recibe un argumento, podrían ser:

```
en( function(palabro) {
   alert(palabro);
}
un( palabro => alert(palabro) )
```