Amplíane la explicación

Claro, aquí tienes una explicación ampliada y detallada sobre el método `filter` en JavaScript, basada en la nota que mostraste:

¿Qué es `filter` en JavaScript?

El método `filter` es una función propia de los arrays (arreglos) en JavaScript que sirve para crear un nuevo array, incluyendo solo los elementos que cumplen una condición específica. Esa condición se define en una función llamada *callback*.

Sintaxis ampliada

```
var nuevoArray = arr.filter(function(currentValue, index, array) {
    // Debe devolver true o false por cada elemento
});
```

- **arr:** El array original sobre el que queremos filtrar los datos.
- **nuevoArray:** Es el nuevo array que resulta de aplicar el filtro; contendrá únicamente los elementos que "pasen" la prueba definida en la función callback.

Parámetros de la función callback

1. **currentValue**
   Es el elemento actual del array que está siendo procesado en ese momento. Por cada elemento, la función callback se ejecuta una vez, dándole como primer argumento ese elemento.
2. **index**
   Es el índice o la posición del elemento actual dentro del array original. Este parámetro es útil si necesitas filtrar elementos según su posición.
3. **array**
   Es el array completo sobre el que se está realizando la operación de filter. Normalmente no se usa mucho, pero permite comparar el elemento actual con otros del array, si se necesita.

¿Cómo funciona el callback?

La función callback tiene que devolver **true** o **false**:

- Si devuelve **true** para un elemento, ese elemento será incluido en el nuevo array.
- Si devuelve **false**, el elemento es excluido del resultado.

Ejemplo práctico

```
let numeros = [5, 10, 15, 20];
let mayoresQueDiez = numeros.filter(function(numero) {
    return numero > 10;
});
// mayoresQueDiez = [15, 20]
```

En este ejemplo:

- La condición es `numero > 10`
- Solo los números mayores que 10 pasan el filtro y se incluyen en el nuevo array.

Resumiendo

- El método `filter` te permite seleccionar (filtrar) solamente los elementos de un array que cumplen cierta condición.
- No modifica el array original, sino que crea uno nuevo con los elementos filtrados.
- Es muy útil para buscar, seleccionar o reducir información en arrays de forma clara y funcional.

¿Te gustaría un ejemplo avanzado o con arrow functions?

⁂

Pero cómo funciona la function? Donde lo hemos declarado?

La función que usa `filter` se declara directamente dentro del método, como argumento. Esta función puede ser anónima (no tiene nombre) o una función que ya esté definida previamente.

¿Dónde se declara la función?

La función se declara en el lugar donde llamas a `.filter()`. Puedes hacerlo así:

Función anónima directamente

```
let numeros = [5, 10, 15, 20];
let mayoresQueDiez = numeros.filter(function(numero) {
    return numero > 10;
});
```

En este caso, la función que va dentro (llamada **callback**) es anónima y se declara justo dentro de `filter`. Esa función se ejecuta automáticamente por cada elemento del array.

Función flecha (arrow function)

```
let mayoresQueDiez = numeros.filter(numero => numero > 10);
```

Igual que antes, pero usando una sintaxis más corta, típica de JavaScript moderno.

Función declarada aparte

También puedes declarar la función previamente y luego pasarla por nombre:

```
function esMayorQueDiez(numero) {
    return numero > 10;
}

let mayoresQueDiez = numeros.filter(esMayorQueDiez);
```

La función `esMayorQueDiez` está declarada aparte, y `filter` la usa para decidir qué elementos incluir.

¿Cómo funciona internamente?

- El método `.filter()` recorre automáticamente cada elemento del array.
- Por cada elemento llama a la función callback y le pasa el elemento, su índice y el array original.
- Todo esto lo hace por sí mismo; solo necesitas pasarle la función (dentro o fuera del método).

En resumen, la función se escribe justo dentro del `.filter()` (como función anónima o flecha) o puede ser una función creada fuera y pasada por nombre[2].

⁂

No sería mejor un if, en vez de un return

En realidad, para la función callback del método `filter` es más común y claro usar `return` porque la función debe devolver un valor booleano (`true` o `false`) para decidir si cada elemento se incluye o no en el nuevo array.

Sin embargo, sí es posible usar un `if` dentro de la función, pero al final tienes que retornar un valor booleano. Por ejemplo:

```
let mayoresQueDiez = numeros.filter(function(numero) {
    if (numero > 10) {
        return true;  // Incluye el número en el nuevo array
    } else {
        return false; // Excluye el número
```

```
    }
});
```

Este código es equivalente a escribirlo con solo el `return`:

```
let mayoresQueDiez = numeros.filter(numero => numero > 10);
```

¿Por qué usar solo return?

- Es más corto y sencillo.
- La expresión `numero > 10` se evalúa directamente a `true` o `false`.
- Evita código innecesario y mejora la legibilidad.

Conclusión

Se puede usar `if`, pero es redundante cuando solo necesitas devolver el resultado de la condición. Para funciones simples como las usadas en `filter`, es mejor usar solo `return` con la condición directamente[3].

⁂

1. image-1.jpg
2. image-1.jpg
3. image-1.jpg