ThePower business school

Reduce JS 🚫🤖

El método `reduce()` de JavaScript es una función de la clase `Array` que permite aplicar una función a cada elemento de un array y reducirlo a un único valor. Es muy útil para realizar cálculos sobre un array de una forma rápida y sencilla.

Aquí tienes un ejemplo de cómo se usa `reduce()`:

```
const numbers = [1, 2, 3, 4, 5];

const sum = numbers.reduce((accumulator, currentValue) => accumulator + currentValue);

console.log(sum); // 15
```

En este ejemplo, hemos creado un array `numbers` con cinco números. Luego, hemos utilizado `reduce()` para sumar todos los números del array y guardar el resultado en una variable `sum`. Al final, hemos imprimido esta variable en la consola y podemos ver que contiene la suma de todos los números del array.

Otro ejemplo de cómo se puede usar `reduce()` es para calcular el promedio de un array de números:

```
const scores = [90, 75, 60, 100, 85];

const average = scores.reduce((accumulator, currentValue) => accumulator + currentValue) / s
cores.length;

console.log(average); // 81
```

En este caso, tenemos un array `scores` con cinco números que representan las calificaciones de distintos estudiantes. Usamos `reduce()` para sumar todas las calificaciones y luego dividimos el resultado entre la cantidad de calificaciones para

obtener el promedio. Al final, imprimimos el promedio en la consola y podemos ver que es 81.

Avanzando en el uso de reduce

Al igual que ***.map()***, ***.reduce()*** también ejecuta una devolución de llamada para cada elemento de un ***array***. Lo diferente aquí es que **reduce el resultado** de esta devolución de llamada (el acumulador) de un elemento del ***array*** a otro.

> El acumulador puede ser prácticamente cualquier cosa (entero, cadena, objeto, etc.) y debe instanciarse o pasarse al llamar a .reduce().

¡Hora de un ejemplo! Digamos que tienes una ***Array*** con estos pilotos y sus respectivos años de experiencia.

```
var pilots = [
  { id: 10, name: "Poe Dameron", years: 14, },
  { id: 2, name: "Temmin 'Snap' Wexley", years: 30, },
  { id: 41, name: "Tallissan Lintra", years: 16, },
  { id: 99, name: "Ello Asty", years: 22, }
];
```

Necesitamos conocer el **total de años de experiencia** de todos ellos. Con **.reduce()**, es bastante sencillo.

```
var totalYears = pilots.reduce(function (accumulator, pilot) {
return accumulator + pilot.years;}, 0);
```

Tenemos en cuenta que hemos establecido el **valor inicial en 0**. También podría haber usado una variable existente si fuera necesario. Después de ejecutar la devolución de

llamada para cada elemento del array, reduce devolverá el valor final de nuestro acumulador (en nuestro caso: 82).

Veamos cómo se puede acortar esto con las funciones de flecha de ES6.

```
const totalYears = pilots.reduce((acc, pilot) => acc + pilot.years, 0);
```

Ahora digamos que quiero encontrar qué piloto es el más experimentado. Para eso, puedo usar .reduce() también.

```
var mostExpPilot = pilots.reduce(function (oldest, pilot) {
return (oldest.years || 0) > pilot.years ? oldest : pilot;}, {});
```

Llamé a mi acumulador más antiguo. Mi callback de llamada compara el acumulador con cada piloto. Si un piloto tiene más años de experiencia que el más antiguo, entonces ese piloto se convierte en el nuevo más viejo, así que ese es el que esta dentro del return.

Como puedes ver, usar .reduce() es una manera fácil de generar un único valor u objeto a partir de una matriz.

Combinando .map(), .reduce(), and .filter()

Nuestro objetivo: obtener la puntuación total de los usuarios de la fuerza solamente. ¡Hagámoslo paso a paso!

```
var personnel = [
  { id: 5, name: "Luke Skywalker", pilotingScore: 98, shootingScore: 56, isForceUser: true, },
  { id: 82, name: "Sabine Wren", pilotingScore: 73, shootingScore: 99, isForceUser: false, },
  { id: 22, name: "Zeb Orellios", pilotingScore: 20, shootingScore: 59, isForceUser: false, },
  { id: 15, name: "Ezra Bridger", pilotingScore: 43, shootingScore: 67,
  isForceUser: true, },
  { id: 11, name: "Caleb Dume", pilotingScore: 71, shootingScore: 85,
  isForceUser: true, }
];
```

Primero, necesitamos filtrar al personal que no puede usar la fuerza.

```
var jediPersonnel = personnel.filter(
  function (person) {
    return person.isForceUser;
  }
);

// Result: [{...}, {...}, {...}] (Luke, Ezra and Caleb)
```

Con eso nos quedan 3 elementos en nuestra matriz resultante. Ahora necesitamos crear una matriz que contenga la puntuación total de cada Jedi.

```
var jediScores = jediPersonnel.map(
  function (jedi) {
    return jedi.pilotingScore + jedi.shootingScore;
  }
);// Result: [154, 110, 156]
```

Y usemos .reduce() para obtener el total.

```
var totalJediScore = jediScores.reduce(
  function (acc, score) {
    return acc + score;
  }, 0
); // Result: 420
```

Y ahora esta es la parte divertida ... podemos encadenar todo esto para obtener lo que queremos en una sola línea.

```
var totalJediScore = personnel
  .filter(function (person) {
    return person.isForceUser;
  })
  .map(function (jedi) {
    return jedi.pilotingScore + jedi.shootingScore;
  })
```

```
  .reduce(function (acc, score) {
    return acc + score;
  }, 0);
```

Y ahora haciendo uso de las arrow functions.

```
const totalJediScore = personnel.filter(person => person.isForceUser)
  .map(jedi => jedi.pilotingScore + jedi.shootingScore)
  .reduce((acc, score) => acc + score, 0);
```