

مختصر كتابي: Reduce

تحويل المصفوفة إلى قيمة واحدة

Reduce

reduce



سلطة فواكه

filter



اختيار

map



تقطيع

```
numbers.reduce((acc, cur) => acc + cur, 0)
```

المجمع

القيمة الحالية

القيمة الابتدائية

المجمع يحتفظ بالنتيجة



لا تنس كتابة return !

مصفوفة كاملة تصبح قيمة واحدة

مختصر كتابي: Reduce

كيف تحوّل reduce المصفوفة إلى قيمة واحدة؟

ال **reduce** مختلفة قليلًا عن methods التكرار التي رأيناها: فهي تمر على كل عناصر المصفوفة وتجمعها بشكل ما، ثم **ترجع في النهاية نتيجة واحدة فقط**، مثل مجموع الأرقام أو أطول كلمة. وسر عملها متغير اسمه **المُجمّع (Accumulator)** يحتفظ بالنتيجة من دورة إلى التي بعدها.

الفرق بين reduce وأخواتها

تخيل أن عندك ثلاث فواكه: ال map تقطّع كل واحدة (تحوّلها من حالة لأخرى)، وال filter تختار البرتقال فقط، أما ال reduce فتجمع الثلاث معًا في **سلطة فواكه** واحدة.

ال method	ماذا ترجع؟
map	مصفوفة جديدة بنفس الطول بعد تحويل العناصر
find	عنصرًا واحدًا فقط يحقق الشرط
filter	كل العناصر التي تحقق الشرط
forEach	لا ترجع شيئًا، تمر على العناصر فقط
reduce	قيمة واحدة نهائية

جمع كل الأرقام

دالة ال callback تستقبل القيمة السابقة (previous value) والقيمة الحالية (current value)، ويمكنها أيضًا أن تستقبل ال index والمصفوفة الأصلية. لكننا نكتفي بأول اثنين. وما ترجعه الدالة في كل دورة **يُحفظ في ال accumulator** ليُستخدم في الدورة التالية. لذلك لو لم نكتب return ستكون قيمته `undefined` :

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

numbers.reduce((accumulator, currentValue) => {
  console.log(accumulator, currentValue);
});
// 1 2
// undefined 3
// undefined 4
// undefined 5
```

والآن نرجع المجموع، ونعطي ال accumulator **قيمة ابتدائية (Initial value)** هي 0 بعد الدالة:

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

const sum = numbers.reduce(
  (accumulator, currentValue) => accumulator + currentValue,
  0
);

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

الدورة	accumulator	currentValue	القيمة المرجعة
1	0	1	1
2	1	2	3
3	3	3	6
4	6	4	10
5	10	5	15

- آخر قيمة مرجعة هي النتيجة النهائية التي تخرج من reduce.
- لو استخدمت الطرح accumulator - currentValue مع البداية 0 تكون النتيجة -15.
- القيمة الابتدائية تغيّر النتيجة: مع الجمع والبداية 5 تكون النتيجة 20، ومع البداية 10 تكون 25.

مثال بالكلمات: أطول كلمة

هنا القيمة الابتدائية **string فارغ ""**، وفي كل دورة نقارن طول الكلمة المحفوظة في ال accumulator بطول الكلمة الحالية، ونرجع الأطول باستخدام ال ternary:

```
const fruits = ["apple", "banana", "avocado", "orange"];

const longestWord = fruits.reduce(
  (accumulator, currentValue) =>
    accumulator.length > currentValue.length ? accumulator : currentValue,
  ""
);

console.log(longestWord); // avocado
```

المقارنات بالترتيب: "" مع apple فتفوز apple، ثم apple مع banana فتفوز banana، ثم banana مع avocado فتفوز avocado، ثم avocado مع orange فتبقى avocado. ويمكنك كتابة نفس الشرط بـ `if` و `else` لو وجدت ال ternary طويلة، والطريقتان صحيحتان.

نفس المثال بـ `forEach` ولماذا `reduce` أفضل؟

```
const numbers = [1, 2, 3, 4, 5];
let count = 0;

numbers.forEach((number) => {
  count += number;
});

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

النتيجة نفسها والكود يبدو أقصر، لكن الدالة هنا **ليست Pure function**، لأنها تغيّر المتغير `count` الموجود خارجها. أما في `reduce` فالقيمة الابتدائية والمعاملات كلها داخلها، فتستطيع نقلها إلى أي صفحة أو مشروع آخر وتعمل كما هي. الحلّان صحيحان، لكن `reduce` هي ال **best practice** هنا.

الخلاصة

- `reduce` تمر على كل العناصر وترجع قيمة واحدة نهائية.
- ما ترجعه ال `callback` يُحفظ في ال `accumulator` للدورة التالية، فلا تنسَ `return`.
- القيمة الابتدائية تُكتب بعد الدالة، مثل `0` للأرقام أو "" للكلمات.
- تفيد في حالات محددة: الجمع والطرح وإيجاد أكبر أو أصغر قيمة.

مصطلحات الدرس

المصطلح	بالإنجليزية	المعنى
الاختزال	reduce	تحويل المصفوفة كلها إلى قيمة واحدة
المجمّع	Accumulator	يحتفظ بما ترجعه الدالة من دورة إلى أخرى
القيمة الحالية	Current value	العنصر الذي تمر عليه الحلقة الآن
القيمة الابتدائية	Initial value	القيمة التي يبدأ بها ال <code>accumulator</code>
الدالة النقية	Pure function	دالة لا تعتمد على شيء خارجها ولا تغيّره