Array Operations

JavaScript offers several array-related methods that favor a functional programming style. Let's go over these methods one by one.

WE'LL COVER THE FOLLOWING

Functional programming is about writing programs by combining functions expressing what the program should do, rather than how to do it.

The `map()` Method

The `map()` method takes an array as a parameter and creates a new array with the results of calling a provided function on every element in this array. A typical use of `map()` is to replace a loop for array traversal. Let's see `map()` in action.

```
const numbers = [1, 5, 10, 15];
// The associated function multiply each array number by 2
const doubles = numbers.map(x => x * 2);

console.log(numbers); // [1, 5, 10, 15] (no change)
console.log(doubles); // [2, 10, 20, 30]
```

Here's how our `titles()` could be rewritten using `map()`. Look how the function code is now more concise and expressive.

```
// Get movie titles
const titles = movies => {
  /* Previous code

  const titles = [];
  for (const movie of movies) {
    titles.push(movie.title);
  }
  return titles;
  */

  // Return a new array containing only movie titles
  return movies.map(movie => movie.title);
};
```

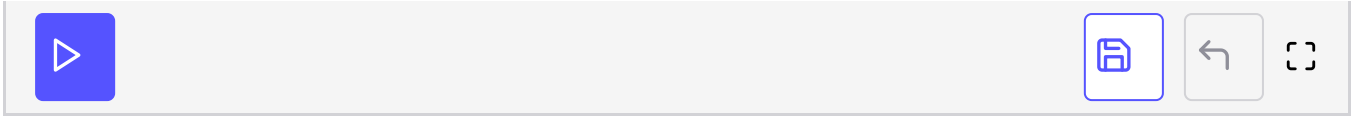
The `filter()` Method

The `filter()` method offers a way to test every element of an array against a provided function. Only elements that pass this test are added to the returned array.

Here's an example of using `filter()`.

```
const numbers = [1, 5, 10, 15];
// Keep only the number greater than or equal to 10
const bigOnes = numbers.filter(x => x >= 10);

console.log(numbers); // [1, 5, 10, 15] (no change)
console.log(bigOnes); // [10, 15]
```



We can use this method in the `nolanMovies()` function.

```
// Get movies by Christopher Nolan
const nolanMovies = movies => {
  /* Previous code
  const nolanMovies = [];
  for (const movie of movies) {
    if (movie.director === "Christopher Nolan") {
      nolanMovies.push(movie);
    }
  }
  return nolanMovies;
  */

  // Return a new array containing only movies by Christopher Nolan
  return movies.filter(movie => movie.director === "Christopher Nolan");
};
```

The `map()` and `filter()` method can be used together to achieve powerful

effects. Look at this new version of the `bestTitles()` function.

```
// Get titles of movies with an IMDB rating greater or equal to 7.5
const bestTitles = movies => {
  /* Previous code
  const bestTitles = [];
  for (const movie of movies) {
    if (movie.imdbRating >= 7.5) {
      bestTitles.push(movie.title);
    }
  }
  return bestTitles;
  */

  // Filter movies by IMDB rating, then creates a movie titles array
  return movies.filter(movie => movie.imdbRating >= 7.5).map(movie => movie.title);
};
```

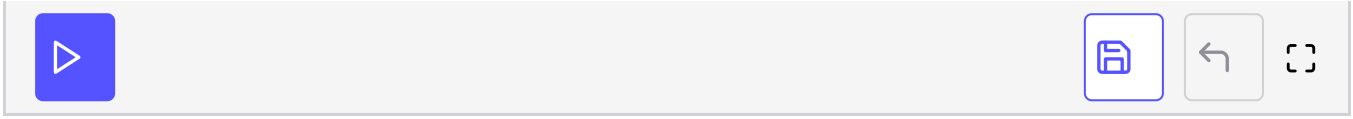
The `reduce()` Method

The `reduce()` method applies a provided function to each array element in order to reduce it to one value. This method is typically used to perform calculations on an array.

Here's an example of reducing an array to the sum of its values.

```
const numbers = [1, 5, 10, 15];
// Compute the sum of array elements
const sum = numbers.reduce((acc, value) => acc + value, 0);

console.log(numbers); // [1, 5, 10, 15] (no change)
console.log(sum);     // 31
```



The `reduce()` method can take several parameters:

- The first one is the function associated to `reduce()` and called for each array element. It takes two parameters: the first is an *accumulator* which contains the accumulated value previously returned by the last invocation of the function. The other function parameter is the array element.
- The second one is the initial value of the accumulator (often 0).

Here's how to apply `reduce()` to calculate the average rating of a movie list.

```
// Compute average rating of a movie list
const averageRating = movies => {
  /* Previous code
  let ratingSum = 0;
  for (const movie of movies) {
    ratingSum += movie.imdbRating;
  }
  return ratingSum / movies.length;
  */

  // Compute the sum of all movie IMDB ratings
  const ratingSum = movies.reduce((acc, movie) => acc + movie.imdbRating, 0);
  return ratingSum / movies.length;
};
```

Another possible solution is to compute the rating sum by using `map()` before reducing an array containing only movie ratings.

```
// ...
// Compute the sum of all movie IMDB ratings
const ratingSum = movies.map(movie => movie.imdbRating).reduce((acc, value) => acc + value, 0
// ...
```