Tip 24: Apply Consistent Actions with forEach()

In this tip, you'll learn how to apply an action to each member of an array with forEach().

We'll cover the following

- Operations on the array values
  - Example
- forEach() method
  - Example 1: Conversion to uppercase
  - Example 2: Sending email

Operations on the array values

Things are going to get a little different in this tip. The two array methods you've explored so far return a new, altered array. You either changed the shape by pulling out a subset of information on each item, or you changed the size by returning only part of the total number of items.

In this tip, you aren't changing the input array at all. Instead, you're going to perform an action on every member. This is common when you finally get an array to the size and shape you want and then you want to do something with that data.

Example

As an example, say you have a club with a group of members and you want to write a script to send an invitation to every club member when the next meeting is scheduled. You want a function that takes each member individually so that you can use other information—*name, email, and so on*—to customize the message. Here's a list of members:

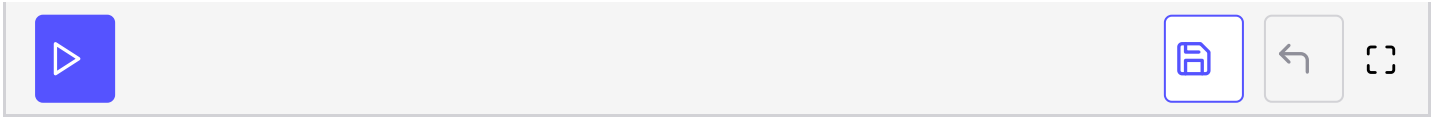
```
const sailingClub = [
    'yi hong',
    'andy',
    'darcy',
    'jessi',
    'alex',
```

```
    'nathan',
];
```

Don't worry about the implementation details of the email function. All you need to know is that it takes a member object. As always, you could easily achieve your goal with a simple `for` loop.

```
const sailingClub = [
    'yi hong',
    'andy',
    'darcy',
    'jessi',
    'alex',
    'nathan',
];

for (let i = 0; i < sailingClub.length; i++) {
    sendEmail(sailingClub[i]);
}
```



You really can't get much simpler than that. Unlike other methods, `forEach()` isn't valuable because it makes your code simpler. It's valuable because it's predictable and because it works like other array methods so it can be chained together (you'll see more about that in the next tip) with other methods.

`forEach()` method

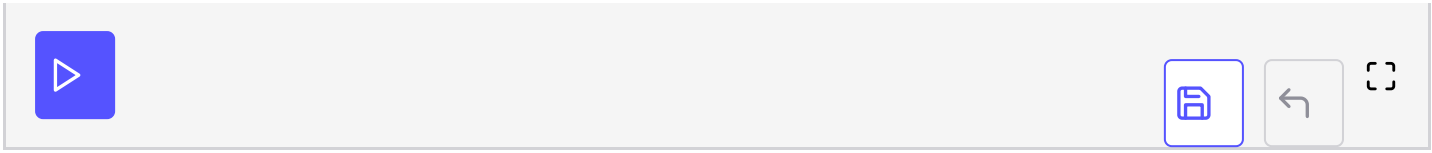
The `forEach()` method, like all you've seen before, takes a function that takes a single argument: *the individual member of the array*. Unlike the other methods, the *return* statement (*whether explicitly or implicitly defined*) does absolutely nothing. Any action you take must affect something outside the function. Changing something outside the scope of the function is called a **side effect**, and though it's not horrible, it should be exercised with caution.

Example 1: Conversion to uppercase

If you use `forEach()` to transform some names to uppercase, you'd get no results. This method does nothing unless you have a side effect of some sort. (*By the way, this is why you should always test your code.*) This code would effectively do nothing:

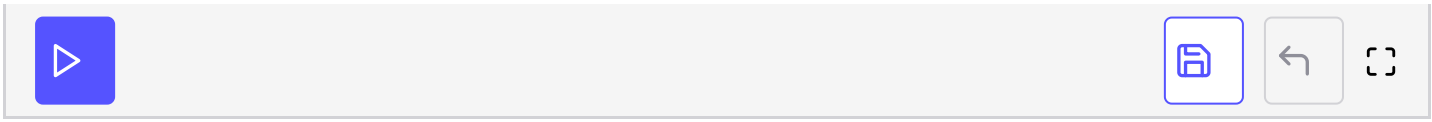
```
const names = ['walter', 'white'];
const capitalized = names.forEach(name => name.toUpperCase());
```

```
console.log(capitalized);
```



You could have a container array to collect the change result, but by now, you know that's bad because it mutates the capitalized array. Besides, that isn't even necessary because `map()` does the same thing.

```
const names = ['walter', 'white'];
let capitalized = [];
names.forEach(name => capitalized.push(name.toUpperCase()));
console.log(capitalized);
```



So when should you use `forEach()`? The best time is precisely when you want to perform an action outside the scope of the function. In other words, when you know you must cause a side effect, you should use `forEach()`.

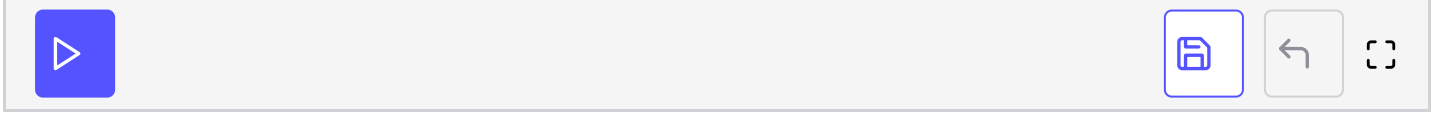
As it happens, that's exactly what you're doing when you send an invitation. You're causing a side effect—sending an email—but you aren't mutating any data (you assume).

Example 2: Sending email

Here's the updated action:

```
const sailingClub = [
    'yi hong',
    'andy',
    'darcy',
    'jessi',
    'alex',
    'nathan',
];

sailingClub.forEach(member => sendEmail(member));
```



Three lines down to one line isn't bad, but it's certainly no cause for celebration.

So what's the point? The point is that you do get some predictability. When you see a `forEach()`, you know there's going to be a side effect. And as you learned in Tip

1, Signal Unchanging Values with const, if you can't be certain of something, the next best option is knowing that there might be instability.

Even with that, the best reason to keep `forEach()` in your toolbox is that you can combine it with other array methods in a process called **chaining**. That means that you can perform multiple actions on the same array without needing to save the output to variables each time.

In the next tip, you'll use chaining to combine several actions into one process.