

# Writing an Array Extension

Write a function that creates a palindrome of any array.

Suppose an array of numbers is given. Create a method `toPalindrome` that creates a palindrome out of your array in the following way:

```
const arr = [1,2,3];  
//[1, 2, 3]  
const arr2 = arr.toPalindrome()  
//[1, 2, 3, 2, 1]  
const arr3 = arr2.toPalindrome()  
//[1, 2, 3, 2, 1, 2, 3, 2, 1]  
console.log( arr, arr2, arr3 );  
//[1, 2, 3] [1, 2, 3, 2, 1] [1, 2, 3, 2, 1, 2, 3, 2, 1]  
//undefined
```

```
Array.prototype.toPalindrome = function(){  
  //write your code here  
}
```



gives an error

`toPalindrome()` returns a new array. It keeps the element `arr[arr.length - 1]` the same, and concatenates all the other elements of the array after the end in reverse order.

## Solution:

This exercise is straightforward, it only requires basic JavaScript knowledge, including JavaScript prototypes. We can go on the safe route, and only use basic ES5 constructs.

```

Array.prototype.toPalindrome = function() {
  const result = this.slice();
  for ( var i = this.length - 2; i >= 0; --i ) {

    result.push( this[i] );
  }
  return result;
}

const arr = [1,2,3];
console.log(arr.toPalindrome())
//[1, 2, 3, 2, 1]
console.log(arr.toPalindrome().toPalindrome())
//[1, 2, 3, 2, 1, 2, 3, 2, 1]

```



To solve this task, you need to know the following about JavaScript:

- To create an array method, we need to extend the prototype of the `Array` object
- The `slice` method without arguments *clones* an array of integers. In reality, this is a *shallow copy*, which is fine in case of atomic values like integers. To read more about cloning, check out my blog posts [Cloning Objects in JavaScript](#) and [Understanding Value and Reference Types in JavaScript](#).
- We use a simple `for` loop to iterate on the elements of the original array that we want to `push` to the end of the array. Push modifies the original `result` array.

This is a safe and straightforward solution. You can use some more native array methods to make the solution more compact:

```

Array.prototype.toPalindrome = function() {
  return this.slice().concat( this.slice( 0, this.length - 1 ).reverse() )
}

const arr = [1,2,3];
console.log(arr.toPalindrome())
//[1, 2, 3, 2, 1]

console.log(arr.toPalindrome().toPalindrome())
//[1, 2, 3, 2, 1, 2, 3, 2, 1]

```



The solution can be explained as follows:

- `slice` still makes a shallow copy of the original array
- `this.slice( 0, this.length - 1 )` makes a shallow copy of the original array, excluding the last element
- `reverse` reverses the elements of the array. Although `reverse` mutates the original array, we are mutating the shallow copy `this.slice( 0, this.length - 1 )`.
- `concat` concatenates two arrays

```
const arr = [1,2,3];

// slice: shallow copy
const arr2 = arr.slice();
//[1, 2, 3]
arr2[ 1 ] = 5;
console.log( arr, arr2 );
//[1, 2, 3] [1, 5, 3]
const arr3 = arr.slice( 0, arr.length - 1 );
//[1, 2]

// reverse
arr2.reverse();
console.log( arr, arr2 );
//[1, 2, 3] [3, 5, 1]

// concat
console.log(arr.concat( arr3 ));
//[1, 2, 3, 1, 2]

console.log( arr, arr3 )
//[1, 2, 3] [1, 2]

console.log(arr.concat( arr3.reverse() ))
//[1, 2, 3, 2, 1]

console.log(arr3)
//[2, 1]
```



If we use ES6, we can replace the `slice` and the `concat` methods with the spread operator:

```
Array.prototype.toPalindrome = function() {
  return [...this, ...this.slice( 0, this.length - 1 ).reverse() ];
}

const arr = [1,2,3];
console.log(arr.toPalindrome())
```



If you would like to read more about the spread operator, sign up for my ES6 minicourse or check out my article on the [Spread Operator and Rest Parameters](#).

As you can see, this simple exercise is linked to a lot of JavaScript knowledge that you need to be aware of.