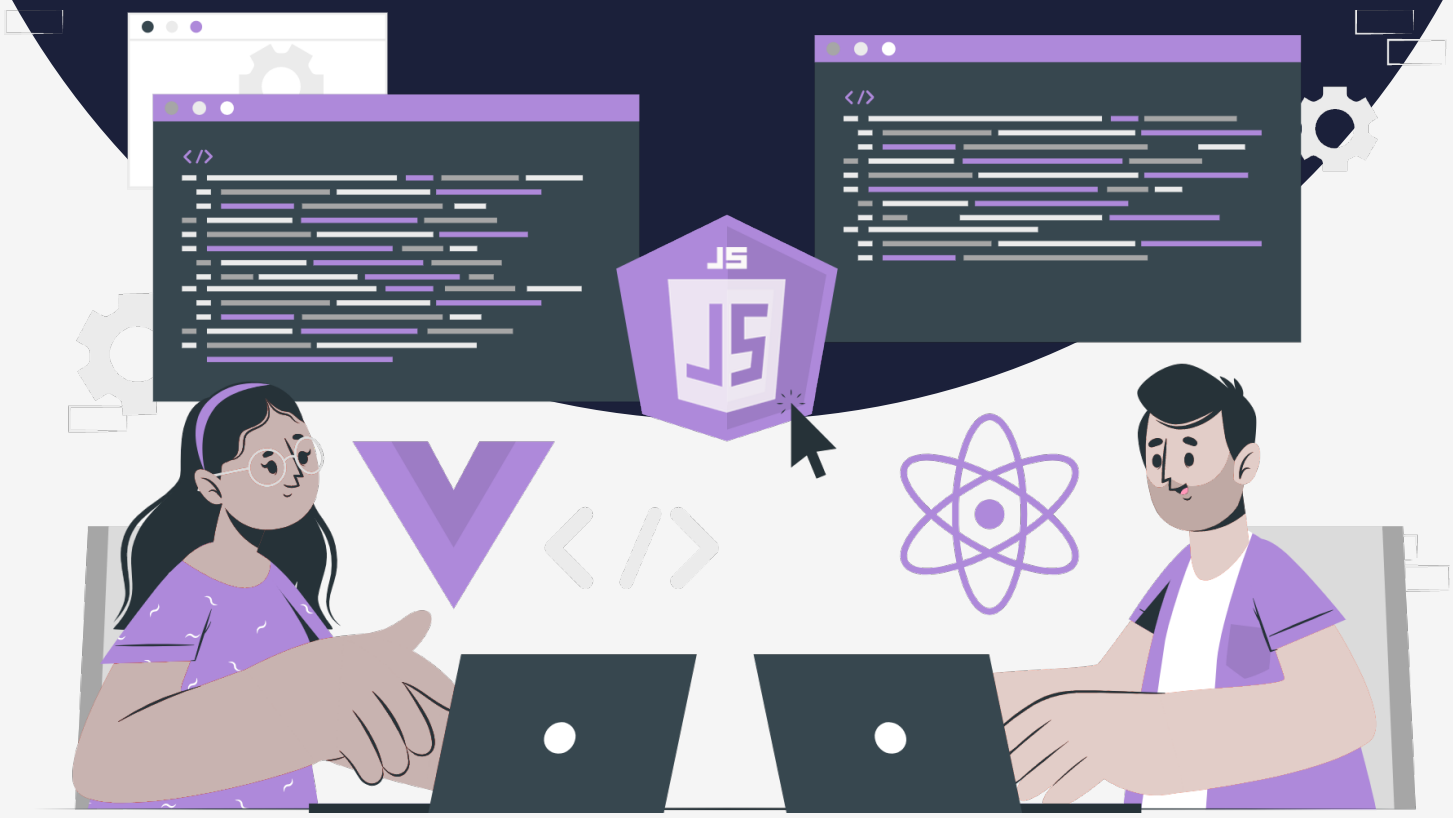
Lesson:

For each, map, filter



PW SKILLS

Topics Covered

1. Introduction to forEach.
2. Syntax.
3. Implementation.
4. Introduction to map method.
5. Syntax.
6. Implementation.
7. Introduction to filter method.
8. Syntax.
9. Implementation.

Earlier whenever there was a necessity of iterating over arrays, developers majority used for loops. But, using loops can sometimes be slow and error-prone when we deal with large complex arrays. So, forEach was standardized with the release of ECMAScript 5 (ES5) in 2009.

The introduction of the forEach() method in ES5 provided a more convenient and efficient way to iterate over the elements of an array. It allowed developers to perform a specific function on each element of an array without having to manually write for a loop. One more thing to note is that within forEach we cannot use the loop control statements such as break, and continue.

Syntax:

```
// Syntax
array.forEach(callback(item, index, array) ⇒ Statements);
```

The following are the parameters of forEach() in Javascript:

- callbackFunction: This argument contains the method that will be called for each array element. It is a mandatory parameter.
- item: the array item on iteration.
- index: This parameter is optional and contains the index number of each element.
- array: This parameter is optional and contains the entire array on which the method is being applied.

The return value of this method is always undefined.

Let's imagine you are assigned a task to display all the items added to the cart in an e-commerce application. You can do it in multiple ways but let's assume we are asked to do it using forEach.

```
let itemsInCart = [
  "apple",
  "comb",
  "mike",
  "keyboard",
  "t-shirt",
  "mobile holder",
];
```

```
let itemsInCart = [
  "apple",
  "comb",
  "mike",
  "keyboard",
  "t-shirt",
  "mobile holder",
];
// Display the items in cart

itemsInCart.forEach((item) => console.log(item));

/* OUTPUT :
        apple
        comb
        mike
        keyboard
        t-shirt
        mobile holder
*/
```

Now let's try to use all the parameters. Let's print the item name, the position in which the item was added to the cart, and the total cart items.

```
itemsInCart.forEach((item, index, arr) =>
  console.log(
    `The item ${item} was added to cart in position ${
      index + 1
    }. The items in cart are ${arr}.`
  )
);

/*

OUTPUT:
The item apple was added to cart in position 1. The items in cart are
apple,comb,mike,keyboard,t-shirt,mobile holder.
The item comb was added to cart in position 2. The items in cart are
apple,comb,mike,keyboard,t-shirt,mobile holder.
The item mike was added to cart in position 3. The items in cart are
apple,comb,mike,keyboard,t-shirt,mobile holder.
The item keyboard was added to cart in position 4. The items in cart are
apple,comb,mike,keyboard,t-shirt,mobile holder.
The item t-shirt was added to cart in position 5. The items in cart are
apple,comb,mike,keyboard,t-shirt,mobile holder.
The item mobile holder was added to cart in position 6. The items in cart are
apple,comb,mike,keyboard,t-shirt,mobile holder.

*/
```

In this case, for each element in the itemsInCart array, the function will log a string to the console. The string includes the current item, its index (incremented by 1 as the indexing starts from 0, to make it more readable), and the full array.

The map method in javascript was introduced to iterate and perform operations on array items in a more concise and readable way.

The map method of the array takes a function as a parameter and returns a new array that contains the result of the function performed on each array item.

The map() method is used over other iterative methods such as for loops or forEach() based on the applications. map() allows us to easily operate on the array items. It returns a new array containing the result of these operations. As it creates a new array, the array on which the map method is applied would be unaltered and less prone to error.

Syntax:
The map method can be applied to an array in the following ways:

1. Function declaration.
2. Arrow Function.
3. Callback Function.

```
// Function Declaration
array.map(function (item, index, array) {/* Function Body */})

// Arrow Function

array.map((item, index, array) => {/* Function Body */})
// Call back Function

array.map(callback)
```

- item: It is a required parameter and it holds the value of the current item in the iteration.
- index: It is an optional parameter and it holds the index of the current item in the iteration.
- arr: It is an optional parameter and it holds the array.

The return value is a new array whose items are the result of the function performed on each array item.

Let's now look at a use case of the map method. Imagine you are working at an e-commerce site. Imagine the website is storing the price as strings and you are assigned a task to convert all the prices in the cart to numbers so the total can be calculated.

```
// Array of item prices as strings
let cartItemsPriceAsStrings = ["100", "58.50", "134", "175", "146", "205"];

// Array of item prices as numbers
let cartItemsPriceAsNumbers = cartItemsPriceAsStrings.map((item) =>  Number(item));

console.log(cartItemsPriceAsNumbers);
```

The above example has an array "cartItemsPriceAsStrings" that contains the prices of items as strings. We will use the map method over that array. Using the "Number()" function to convert each string to a number. The result of each iteration on array items is assigned to a new array called "cartItemsPriceAsNumbers"

filter

As the name suggests the filter method is used to filter the data present in an array. On a daily basis in every application, we use filtering. We filter the products in a shopping site based on criteria, we filter the posts on any social media based on the date posted or based on the keywords. So, filtering the data is one of the important task.

The filter method is used to create a new array whose items are the result of the filtering criteria of the original array. Only those values which satisfy the given criteria are added to a new array, and that array is returned. The original array does not get changed.

Whenever the filter method is applied to an array, we pass a filter function to it. The filter function iterates over all the elements of the given array and passes each element to the callback function. If the callback function returns true, then the element is added to the result array otherwise it is ignored.

Syntax:
The callback function can be passed to the filter function in the following ways:

1. Function declaration.
2. Arrow Function.
3. Callback Function.

```
// Function Declaration
array.filter(function (item, index, array) {/* Function Body */})

// Arrow Function
array.filter((item, index, array) => {/* Function Body */})

// Call back Function
array.filter(callback)
```

- item: It is a required parameter and it holds the value of the current item in the iteration.
- index: It is an optional parameter and it holds the index of the current item in the iteration.
- arr: It is an optional parameter and it holds the array.

The filter function returns an array. If items satisfy the condition the items are returned as an array, if none of the items satisfy the filtering criteria an empty array is returned.

Let's imagine you are working for an e-commerce organization and you are assigned to return the long usernames from the data of the username. Long usernames in this case are usernames whose length is greater than 5 characters.

```
let userNames = ["Mithun", "Anurag", "Alka", "Prabir", "Vinay"];

let longUserNames = userNames.filter((item) ⇒ item.length > 5);

console.log(longUserNames); // OUTPUT : [ 'Mithun', 'Anurag', 'Prabir' ]
```

The filter() method creates a new array “longUserNames” by filtering out the names with a length greater than or equal to 5.

The filter() method iterates through each element in the userNames array, and for each element, it calls the provided function ((item) => item.length > 5) with the element as the argument.

If the function returns true for that element (if the length of the name is greater than 5), it is included in the longUserNames array, otherwise, it is excluded.