Arrays

An array is a special type of variable that can hold multiple values. So, if you had several related values—like all the courses you are taking this semester—you don't need to create a separate variable to hold each one. Instead, you can store them in an array.

So instead of:

```
let course1 = 'CSE111';
let course2 = 'WDD131';
let course3 = 'ITM220';
```

We could use an array like this:

```
let courses = ['CSE111', 'WDD131', 'ITM220'];
```

There are a few ways to create an array. One way is to use the array constructor, with or without the new keyword, and place the values in parentheses:

```
const cars = new Array("Saab", "Volvo", "BMW");
```

But the easiest way is by using an array literal.

```
const cars = ["Saab", "Volvo", "BMW"];
```

This is how you'll usually see arrays in this course. Notice that with the array literal, the values are inside square brackets. String values should be in quotes, while numeric values don't need quotes—just like regular variables.

Here's an array with different strings representing fruit:

```
const fruit = ['watermelon', 'peach', 'apple', 'banana'];
```

We can interact with array values in various ways. To retrieve a value, we use an index number. Arrays use **zero-based indexing**, so the first item has an index of 0.

```
                    0            1        2         3
const fruit = ['watermelon', 'peach', 'apple', 'banana'];
```

To show the value in the second position, use fruit[1];

```
console.log(fruit[1]);
```

This would result in ‘peach’ showing up on the console.
You can also assign a new value by specifying the array name and index number. For example, to add `'grapes'` as the fifth value:

```
fruit[4] = 'grapes';
```

```
const fruit = ['watermelon', 'peach', 'apple', 'banana', 'grapes'];
```

Another way to add a value to the end of an array is with the `.push()` method, which doesn’t require specifying an index:

```
fruit.push(‘orange’);
```

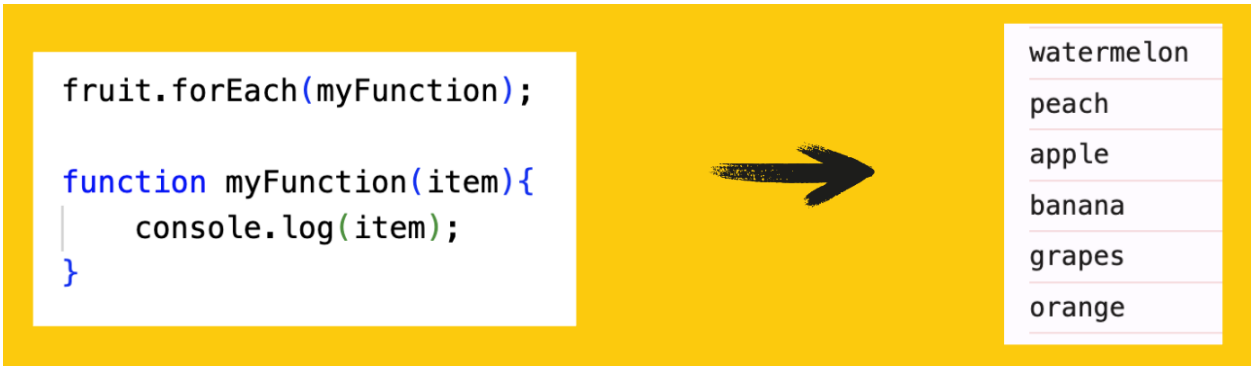
```
const fruit = ['watermelon', 'peach', 'apple', 'banana', 'grapes', 'orange'];
```

Iteration Array Methods

There are several array iteration methods that you need to understand and be able to code.

.forEach( )

The `.forEach()` method runs a function for every item in the array, sending one item to the function each time it runs.




```
fruit.forEach(myFunction);

function myFunction(item) {
     console.log(item);
}
```

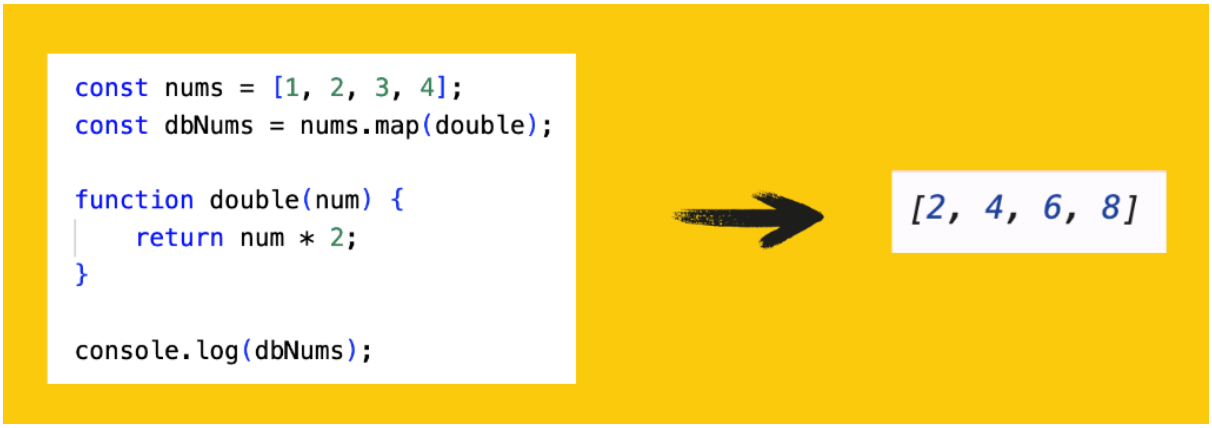
The array name is followed by the .forEach method, and the method’s parameter is the function. This function receives each item of the array—referred to as ‘item’ in this case. So, item represents each value in the array: first ‘watermelon’, then ‘peach’, and so on.

Notice we didn't need to pass an argument to the function, it implicitly receives each array value as it iterates. You could use a different parameter name instead of `'item'` if you prefer.

.map( )

The .map() method creates a new array by calling a function for each item in the original array. The function returns a value for each new element.

Here we have an array of numbers, called nums, that gets doubled using .map and the doubled numbers are placed in a new array called dbNums. The original array remains unchanged.

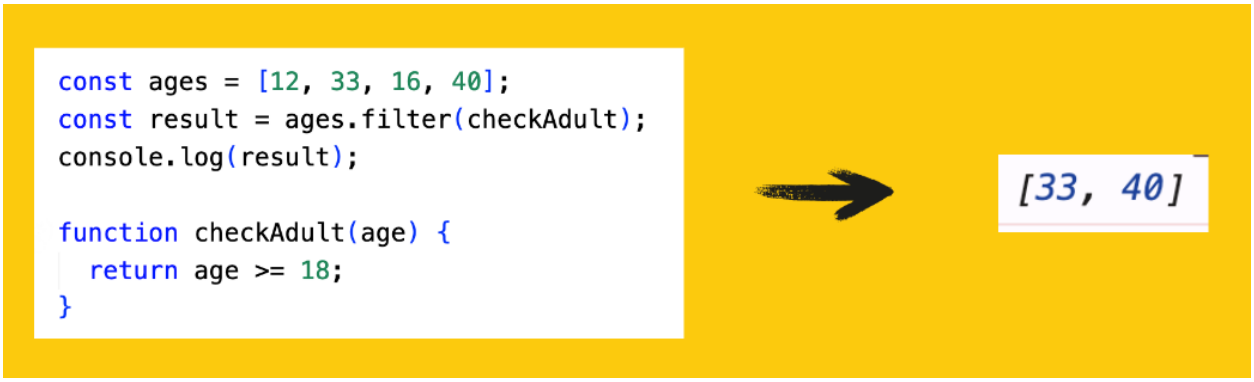



```
const nums = [1, 2, 3, 4];
const dbNums = nums.map(double);
function double(num) {
     return num * 2;
}
console.log(dbNums); // 2, 4, 6, 8
```

Notice how each value in the original array is reference by the term 'num' in this example. The function is called 4 times because there are 4 values in the original array.

.filter( )

The .filter() method creates a new array, but only includes items that meet a certain condition. Each array value is evaluated within the function to see if it passes the condition or not.

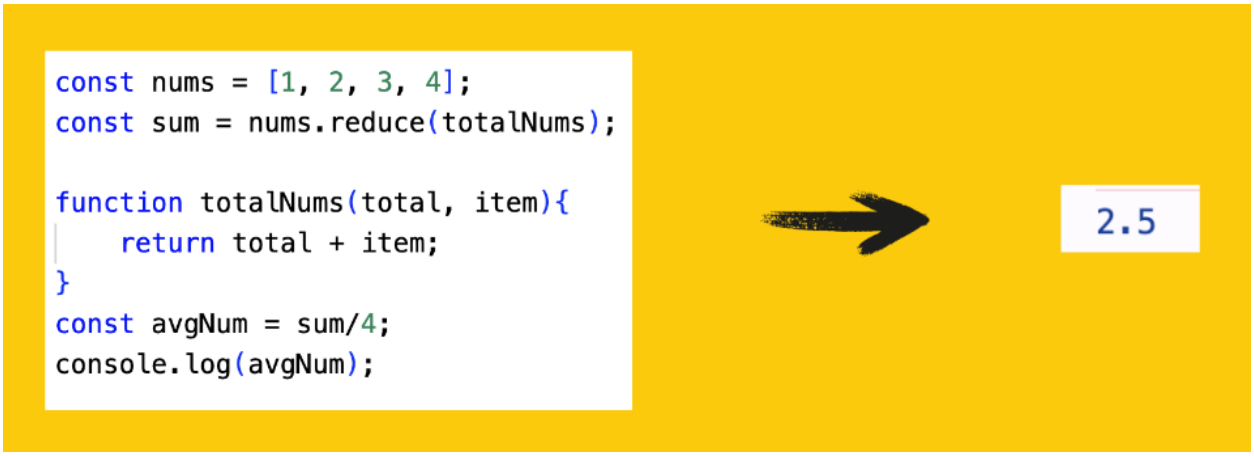



```
const ages = [12, 33, 16, 40];
const result = ages.filter(checkAdult);
console.log(result);
function checkAdult(age) {
     return age >= 18; // 33, 40
}
```

This filter condition is checking the four ages that come into the function to see if they are equal to or greater than 18. If they are, they will be sent back to the new 'result' array.

.reduce( )

The .reduce() method returns a single value. The function it calls accumulates the result. This time the array has two parameters: one for the accumulated result and one for the current value.




```
const nums = [1, 2, 3, 4];
const sum = nums.reduce(totalNums);
function totalNums(total, item) {
     return total + item;
```

```
}
const avgNum = sum/4;
console.log(avgNum); // 2.5
```

Here, all the numbers are added together to get 10, then divided by 4 to get the average.

In all these examples we called a named function. You can also use anonymous functions as the array method parameter with any of these array methods.

```
const sum = nums.reduce(function(total, item) {
 return total + item;
});
console.log(sum);  //results in 10
```

Or use an arrow function to do the same thing.

```
const sum = nums.reduce((total, item)=> total + item);

console.log(sum); //results in 10
```

So, there we have a brief overview of arrays and some array methods you will use to process items in an array.