UNIT 07

LESSON 07.04

### array.reduce() method

About the **reduce()** method:

- As with **map()** and **filter()**, **reduce()** is called on an array and runs a function on each array item, with the current item as the argument, **e**.
- The **reduce()** method returns ONE value which could be of any datatype (string, number, object, etc.).
- The **reduce()** method works on the principle of an **accumulator** value, which gets updated with each iteration of the array.
- The **accumulator** should be the first argument of the function.
- The **answer** is saved to the variable: **let answer = array.reduce()**

Let's try an example of **reduce()** to add up an array of numbers. The one value returned will be the **sum**.

```
const nums = [89, 95, 99, 95, 19, 95, 39, 79];
```

Next, call **reduce()** on the **nums** array, setting that equal to a new variable, **sum**.

```
const sum = nums.reduce();
```

Pass the function to the **reduce()** method. The function's arguments are the **accumultor**, which we will call **tot** and the current array item as **e**. The **accumulator** and the **answer** do not have to have different names--they could both be called **tot** or **sum**, but we will use different names to keep them distinct:

```
const sum = nums.reduce(function (tot, e) {
});
```

Inside the function, add the current item **e** to the **tot**, and return it. In this way, **tot**, *accumulates*:

```
const sum = nums.reduce(function(tot, e) {
    return tot + e;
});
```

Log the answer, **sum**, to see if it worked.

```
    const sum = nums.reduce(function (tot, e) {
        return tot + e;
    });
    console.log(sum); // 610
```

Let's try another example of **reduce()**. This time, it's an array of strings, but the answer is still a number:

```
    const veggies = ['beet', 'carrot', 'peas', 'celery', 'cucumber',
'broccoli'];
```

Use **reduce()** to add up the total number of chars of all the items ('beet' = 4, etc). Use the accumulator, **sum** again, since we are adding up a number:

```
    const totChars = veggies.reduce(function (sum, e) {
        return sum + e.length;
    });
    console.log(totChars); // beet64688
```

We get a strange result: **beet64688**. What happend is since the first array item, **beet**, is a string, the data type string was assumed for the accumulator, with the result that the item length values were concatenated--not added. To fix this, declare number as the data type. This is done right after the closing curly brace of the function, with the initial value set to zero:

```
    const totChars = veggies.reduce(function (sum, e) {
        return sum + e.length;
    }, 0);
    console.log(totChars); // 36
```

using an array of objects with reduce()

Given an array of movies, which you may recall from lesson 05.05 (Looping Arrays of Objects), we will use **reduce()** to get the average duration in minutes of all the movies:

```
    const movies = [
        { name: "The Wizard of Oz", year: 1939, mins: 101 },
        { name: "Casablanca", year: 1942, mins: 102 },
        { name: "Roman Holiday", year: 1953, mins: 118 },
        { name: "Lawrence of Arabia", year: 1962, mins: 227 },
        { name: "Lilies of the Field", year: 1963, mins: 94 },
        { name: "The Godfather, Part II", year: 1974, mins: 202 },
        { name: "The Shining", year: 1980, mins: 144 },
        { name: "Beverly Hills Cop", year: 1984, mins: 105 },
        { name: "Ghost", year: 1990, mins: 127 },
        { name: "Forrest Gump", year: 1994, mins: 142 },
        { name: "Ali", year: 2001, mins: 157 },
        { name: "Due Date", year: 2010, mins: 95 },
        { name: "Black Panther", year: 2018, mins: 134 },
        { name: "Avengers: Endgame", year: 2019, mins: 181 },
    ];
```

Set up the **reduce()** method call, saved to a variable:

```
    const avgMovieMins = movies.reduce();
```

Pass the function to reduce, with arguments of **sum** and **e**. We want the average of the items' **mins** property, but to calculate an average, first we need the sum, which we divide by the total number of items:

```
    const avgMovieMins = movies.reduce(function(sum, e) {
    });
```

Set the accumulator to zero, even though it will be able to figure out to do math without it:

```
    const avgMovieMins = movies.reduce(function(sum, e) {
    }, 0);
```

Inside the function, add the current movie's running time as **e.mins** to **sum**.

```
    const avgMovieMins = movies.reduce(function(sum, e) {
        sum += e.mins;
    }, 0);
```

Calculate the average and return it. This we obtain by dividing the current item's mins property, **e.mins**, by the total number of items in the array, **a.length**. This requires us to pass **a**, representing the array, to the function. To do this, we also need to pass in **i**. The calculated average is added to the **sum**. When it is done, we have the total average running time of the movies.

```
    const avgMovieMins = movies.reduce(function(sum, e, i, a) {
        return sum += e.mins / a.length;
    }, 0);
    console.log(avgMovieMins); // 137.78571428571428
    console.log(Math.round(avgMovieMins)); // 138
```

reduce() instead of map-into-filter chaining

- **reduce()** returns a single value of any data type--including an array, if you so choose.
- unlike **map()**, **reduce()** is not constrained to returning an arraay of the same number of items as in the original array.
- unlike **filter()**, which cannot transform any of the items in the array it is called on, **reduce()** CAN change the items.
- **reduce()** can--all by itself--make what otherwise would require a combination of **filter()** and **map()** to produce: a filtered array of transformed items.

In this example, if the vegetable starts with "c", make vegetable juice, and save the juices to a new array. We will first implement this with filter AND map, before achiveing the same result with just reduce:

Copy-paste our **veggies** array, so that we have it close at hand, but change the name to **vegetables**:

```
    const vegetables = ['beet', 'carrot', 'peas', 'celery', 'cucumber',
'broccoli'];
```

Call the **filter()** method on the **vegetables** array, setting that equal to a new array, **cVegetables**:

```
    const cVegetables = vegetables.filter();
```

Pass in the function with argument **e** for the current array item:

```
    const cVegetables = vegetables.filter(function(e) {
    });
```

The first character of a string is at index 0, so **e[0]** is the first letter of the current item. If **e[0]** is "c", we want to save **e** to the new array by returning it:

```
    const cVegetables = vegetables.filter(function(e) {
        return e[0] == 'c';
    });
```

Log the result:

```
    console.log('cVegetables:', cVegetables);
    // ['carrot', 'celery', 'cucumber']
```

Copy-paste the filter code, and rename the new array **cVegJuices**:

```
    const cVegJuices = vegetables.filter(function(e) {
        return e[0] == 'c';
    });
```

With **filter()** we have obtained the vegetables that start with "c", namely 'carrot', 'celery' and 'cucumber'. So now, it is time to make the juices. For this. we chain **map()** to the end of **filter()**:

```
    const cVegJuices = vegetables.filter(function(e) {
        return e[0] == 'c';
    }).map();
```

Pass **map()** its required function, with argument **e**:

```
    const cVegJuices = vegetables.filter(function(e) {
        return e[0] == 'c';
    }).map(function(e) {
    });
```

Concatenate 'juice' onto the current item, **e**, and **return** that:

```
    const cVegJuices = vegetables.filter(function(e) {
        return e[0] == 'c';
    }).map(function(e) {
        return e + ' juice';
    });
```

Log the result. It should be 'carrot juice', 'celery juice' and 'cucumber juice':

```
    console.log('cVegJuices:', cVegJuices);
    // ['carrot juice', 'celery juice', 'cucumber juice']
```

So that worked just fine, but now to get the same result using only the reduce method.

Call **reduce()** on the **vegetables** array, saving it to a new array, **cJuices**:

```
    const cJuices = vegetables.reduce();
```

Pass in the required function with argument **e**:

```
    const cJuices = vegetables.reduce(function(e) {
    });
```

We need to tell **reduce()** to make an array, so pass a pair of empty square brackets **[]** to the method. This goes *after* the closing curly brace of the function:

```
const cJuices = vegetables.reduce(function(e) {
}, []);
```

Check to see **if** the first letter of the current item is "c", and if it is, add 'juice' to the item and **return** it:

```
const cJuices = vegetables.reduce(function(e) {
    if(e[0] == 'c') {
        return e + ' juice';
    }
}, []);
```

Log the result:

```
console.log('cJuices', cJuices);
// ['carrot juice', 'celery juice', 'cucumber juice']
```

We see that it worked, so we see that with **reduce()**, we can achieve what otherwise would require both map *and* filter. Also, notice that since we were not adding up a total, there was no **accumulator** argument.

END Lesson 07.04
NEXT: Lab 07.04
Lesson 07.05