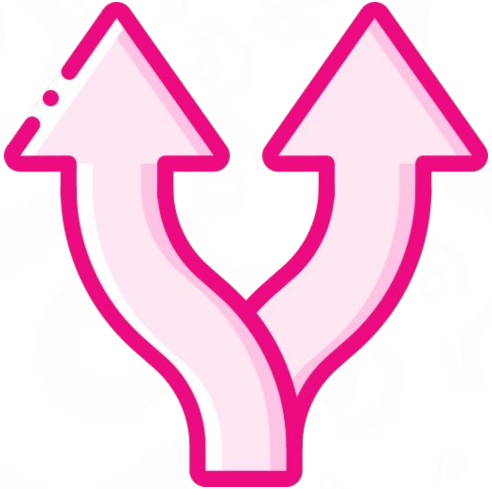
CodingLab Powered by RBK

Bootstrap



Map

An introduction to Map

Duration: 30 minutes
Q&A: 5 minutes by the end of the lecture

Map

There are many situations where we need to apply a **transformation** to every element in an array. This is exactly what the map abstraction is for.

Map

```
var nums = [1, 9, 5, 10, 3];
```

Let’s consider two ways to transform all the elements in an array of numbers: **squaring** and **doubling** all elements. First we’ll write these functions using **each**, and then look for a pattern that we can extract to remove repetition.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {




}
```

We’ll start by writing a function to square all the elements in an array...

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];



  return acc;
}
```

...we need an accumulator to store the numbers as we square them that we will return at the end. We'll call this variable acc for short.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {

  });
  return acc;
}
```

We can use **each** to iterate over all the elements in the **numbers** array...

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
             number * number ;
  });
  return acc;
}
```

...we can compute the square of each number by multiplying it by itself...

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}
```

...and finally, we can **push** each squared number into our acc variable. Let’s verify that it works:

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];
```

Hooray! Now, let’s write a similar function that *doubles* all elements in its parameter.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {



}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];
```

We’ll start by writing the function signature as usual, and we’ll name it **`doubleAll`**.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];
```

Our steps are exactly the same as before, but instead of **squaring** each element, we **double** each element.

CodingLab
Powered by RBK

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]
```

Great! It works exactly as we’d expect. Now, take a look at the two functions -- can you spot the difference?

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]
```

All that changes is what is being pushed into the acc variable! What can we do to remove this repetition?

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]



function map(        ) {



}
```

Now that we notice the repetition, we can begin writing a function to abstract the **map** pattern. We will write it off to the side in the blue box.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]




function map(        ) {
  var acc = [];

  return acc;
}
```

We can start by extracting the acc variable.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]



function map(        ) {
  var acc = [];
  each(    , function(          ) {

  });
  return acc;
}
```

We'll use **each** to perform iteration.

CodingLab Powered by RBK

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]




function map(array   ) {
  var acc = [];
  each(array, function(          ) {

  });
  return acc;
}
```

Using each requires that we have an **array** to iterate over. Since that array will be different each time, we can make it a parameter.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]




function map(array   ) {
  var acc = [];
  each(array, function(element   ) {

  });
  return acc;
}
```

And **each** will be given every **element** in the array -- since these elements could be *anything*, we'll name them something generic like `element`.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]




function map(array, f) {
  var acc = [];
  each(array, function(element   ) {
           f(element   )
  });
  return acc;
}
```

And **each** will be given every **element** in the array -- since these elements could be *anything*, we'll name them something generic like `element`.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]



function map(array, f) {
  var acc = [];
  each(array, function(element   ) {
             f(element   )
  });
  return acc;
}
```

We are going to need to **transform** each element -- the way that we will do this is by invoking a **function** that returns the transformed element. This function will be received as a parameter called **f**.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]




function map(array, f) {
  var acc = [];
  each(array, function(element, i) {
             f(element, i)
  });
  return acc;
}
```

Because map *may* need access to the index of each element at one point, we might as well pass the element’s index from each to the function supplied to map as well.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
  each      (numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]
```

```
function map(array, f) {
  var acc = [];
  each(array, function(element, i) {
    acc.push(f(element, i));
  });
  return acc;
}
```

Finally, we need to ensure that we **push** each transformed element into our accumulator, `acc`. Now we can express `squareAll` and `doubleAll` in terms of `map`!

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
         map(numbers, function(number) {
    acc.push(number * number);
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
         map(numbers, function(number) {
    acc.push(number * 2);
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]
```

```
function map(array, f) {
  var acc = [];
  each(array, function(element, i) {
    acc.push(f(element, i));
  });
  return acc;
}
```

Instead of using each, now we’ll be using map instead.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
         map(numbers, function(number) {
    return   number * number ;
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
         map(numbers, function(number) {
    return   number * 2 ;
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]
```

```
function map(array, f) {
  var acc = [];
  each(array, function(element, i) {
    acc.push(f(element, i));
  });
  return acc;
}
```

The **function parameter** to `map` is slightly different -- because the **result** of invoking it is supposed to **`return`** a value, let's change it to `return` the transformation.

CodingLab
Powered by RBK

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {
  var acc = [];
         map(numbers, function(number) {
    return   number * number ;
  });
  return acc;
}

function doubleAll(numbers) {
  var acc = [];
         map(numbers, function(number) {
    return   number * 2 ;
  });
  return acc;
}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]
```

```
function map(array, f) {
  var acc = [];
  each(array, function(element, i) {
    acc.push(f(element, i));
  });
  return acc;
}
```

We no longer need the **acc** variable, because map does the accumulation step for us...

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {

         map(numbers, function(number) {
    return   number * number ;
  });

}

function doubleAll(numbers) {

         map(numbers, function(number) {
    return   number * 2 ;
  });

}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]




function map(array, f) {
  var acc = [];
  each(array, function(element, i) {
    acc.push(f(element, i));
  });
  return acc;
}
```

...so we can remove it completely! Our last step is to make `squareAll` and `doubleAll` return a value.

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {

         map(numbers, function(number) {
    return   number * number ;
  });

}

function doubleAll(numbers) {

         map(numbers, function(number) {
    return   number * 2 ;
  });

}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]




function map(array, f) {
  var acc = [];
  each(array, function(element, i) {
    acc.push(f(element, i));
  });
  return acc;
}
```

Because map does the accumulation for us and **returns** the accumulator, what should we return?

Map

```
var nums = [1, 9, 5, 10, 3];

function squareAll(numbers) {

  return map(numbers, function(number) {
    return   number * number ;
  });

}

function doubleAll(numbers) {

  return map(numbers, function(number) {
    return   number * 2 ;
  });

}
```

```
squareAll(nums);
// => [1, 81, 25, 100, 9];

doubleAll(nums);
// => [2, 18, 10, 20, 6]




function map(array, f) {
  var acc = [];
  each(array, function(element, i) {
    acc.push(f(element, i));
  });
  return acc;
}
```

The result of invoking map itself!

CodingLab
Powered by RBK

That’s it

Map