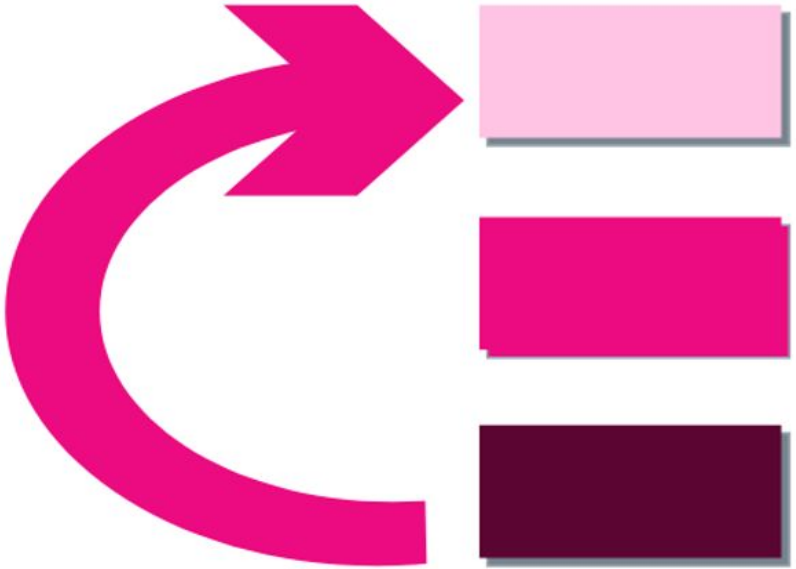

Higher Order Functions

```
var numbers = [1, 9, 27, 5, 10, 3];
```

Let's imagine that we need to compute the **sum** of the `numbers` array.

CodingLab Powered by RBK

Bootstrap

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
```

We'll use a `sum` variable to store our sum in.

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;



for (var i = 0; i < numbers.length; i = i + 1) {
  sum = sum + numbers[i];
}
```

Using a `for` loop, we can compute the sum by iterating over **each number** in the `numbers` array. Now, how about computing the **product** of the `numbers` array?

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
var product = 1;


for (var i = 0; i < numbers.length; i++) {
  sum = sum + numbers[i];
}

for (var i = 0; i < numbers.length; i++) {
  product = product * numbers[i];
}
```

As with computing the sum, we can compute the product by iterating over **each number** in the `numbers` array. How about computing the **maximum number** in the array?

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
var product = 1;
var maximum = numbers[0]; // pick first value to start with

for (var i = 0; i < numbers.length; i++) {
  sum = sum + numbers[i];
}

for (var i = 0; i < numbers.length; i++) {
  product = product * numbers[i];
}

for (var i = 0; i < numbers.length; i++) {
  if (numbers[i] > maximum) {
    maximum = numbers[i];
  }
}
```

Like before, our solution involves iterating over each element in the array; however, we have managed to introduce quite a bit of **repetition**.

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
var product = 1;
var maximum = numbers[0];

for (var i = 0; i < numbers.length; i++) {
  sum = sum + numbers[i];
}

for (var i = 0; i < numbers.length; i++) {
  product = product * numbers[i];
}

for (var i = 0; i < numbers.length; i++) {
  if (numbers[i] > maximum) {
    maximum = numbers[i];
  }
}
```

Notice how we are iterating over the array in exactly the same way in each problem? We start `i = 0`, keep going as long as `i` is less than the length of the array, and increment by one.

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
var product = 1;
var maximum = numbers[0];

for (var i = 0; i < numbers.length; i++) {
  sum = sum + numbers[i];
}

for (var i = 0; i < numbers.length; i++) {
  product = product * numbers[i];
}

for (var i = 0; i < numbers.length; i++) {
  if (numbers[i] > maximum) {
    maximum = numbers[i];
  }
}
```

```
for (var i = 0; i < array.length; i++) {
  // do body
}
```

The blue box shows the formula for iterating over each element of an array. The only parts that change are the **array** being iterated over and the **body** of the loop. What can we use to extract this pattern?

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;



for (var i = 0; i < numbers.length; i++) {
  sum = sum + numbers[i];
}
```

Let's use one of our examples to experiment upon. Our goal is to pull out the **details of iterating over arrays** into its own function, and then use that function to compute the sum.

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(           ) {
for (var i = 0; i < numbers.length; i++) {
  sum = sum + numbers[i];
}
}



each(                 );
```

We'll start by wrapping our `for` loop in a function that we'll call `each`...

CodingLab Powered by RBK

Bootstrap

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(             ) {
  for (var i = 0; i < numbers.length; i++) {
    sum = sum + numbers[i];
  }
}




each(                  );
```

...and then we'll fix our indentation. Now that our loop is in a function, what can we turn into a **parameter**?

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array       ) {
  for (var i = 0; i < array.length; i++) {
    sum = sum + array[i];
  }
}




each(                  );
```

The **array** that we are iterating over! each should work on all arrays, so we'll use a generic parameter name, like array.

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array       ) {
  for (var i = 0; i < array.length; i++) {
    sum = sum + array[i];
  }
}




each(numbers           );
```

Now we need to invoke each with the **numbers** array as an argument.

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array      ) {
  for (var i = 0; i < array.length; i++) {
    sum = sum + array[i];
  }
}

function addToSum      (number) {
  sum = sum + number;
}

each(numbers          );
```

The next part that we want to extract is the loop's **body.** We can start by introducing a function, `addToSum`, that can be used to add a number to the `sum` variable...

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array       ) {
  for (var i = 0; i < array.length; i++) {
    addToSum(   array[i]);
  }
}

function addToSum       (number) {
  sum = sum + number;
}

each(numbers           );
```

...and then we can replace the loop's body with an invocation of addToSum for each element in the array -- but we're not done yet.

CodingLab Powered by RBK

Bootstrap

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func    (   array[i]);
  }
}

function addToSum      (number) {
  sum = sum + number;
}

each(numbers          );
```

Functions, like data, can be parameters to other functions! Instead of hard-coding `addToSum` into our `each` function, we'll extract it into a parameter...

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func    (   array[i]);
  }
}

function addToSum      (number) {
  sum = sum + number;
}

each(numbers, addToSum);
```

...and pass addToSum as an *argument* to each (since functions are just values).

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func    (   array[i]);
  }
}

var addToSum = function(number) {
  sum = sum + number;
}

each(numbers, addToSum);
```

We can rewrite addToSum using the alternative notation that we've learned today in order to highlight something important.

CodingLab Powered by RBK

Bootstrap

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func    (   array[i]);
  }
}

var addToSum = function(number) {
  sum = sum + number;
}

each(numbers, addToSum);
```

Do we really need to provide a label for something like addToSum?

CodingLab Powered by RBK

Bootstrap

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func    (   array[i]);
  }
}

                function(number) {
  sum = sum + number;
}

each(numbers          );
```

We can remove the label addToSum, and with a little rearranging...

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;


function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func    (   array[i]);
  }
}

each(numbers,  function(number) {
  sum = sum + number;
});
```

We arrive at a much more succinct use of each!

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;



each(numbers, function(number) {
  sum = sum + number;
});
```

```
function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func(array[i]);
  }
}
```

Now, we can rewrite the rest of our problems using each. We have already **summed** the numbers -- how can we use each to compute the product?

CodingLab Powered by RBK

Bootstrap

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
var product = 1;


each(numbers, function(number) {
  sum = sum + number;
});
```

```
function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func(array[i]);
  }
}
```

We'll start by reintroducing the variable to store our product in.

CodingLab Powered by RBK


```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
var product = 1;


each(numbers, function(number) {
  sum = sum + number;
});

each(numbers, function(number) {

});
```

```
function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func(array[i]);
  }
}
```

For **each** number in the **numbers** array...

CodingLab Powered by RBK

Bootstrap

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
var product = 1;


each(numbers, function(number) {
  sum = sum + number;
});

each(numbers, function(number) {
  product = product * number;
});
```

```
function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func(array[i]);
  }
}
```

Update the **product** variable by multiplying **number** times the current **product.** How about computing the maximum number?

CodingLab Powered by RBK

Bootstrap

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
var product = 1;
var maximum = numbers[0];

each(numbers, function(number) {
  sum = sum + number;
});

each(numbers, function(number) {
  product = product * number;
});

each(numbers, function(number) {


});
```

```
function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func(array[i]);
  }
}
```

As before, we'll reintroduce the variable that we can use to store the **maximum**, and begin **each** in the same way: for **each** number in **numbers**...

```
var numbers = [1, 9, 27, 5, 10, 3];
var sum = 0;
var product = 1;
var maximum = numbers[0];

each(numbers, function(number) {
  sum = sum + number;
});

each(numbers, function(number) {
  product = product * number;
});

each(numbers, function(number) {
  if (number > maximum) {
    maximum = number;
  }
});
```

```
function each(array, func) {
  for (var i = 0; i < array.length; i++) {
    func(array[i]);
  }
}
```

And update the **maximum** if the **number** is larger than the current **maximum.**

Why do we care?

With a function like each, our code describes **what** is happening (*iteration*) rather than **how** it is happening.

This is the difference between **declarative** and **imperative** code.

CodingLab Powered by RBK

Bootstrap

Imperative v. Declarative

Imperative code emphasizes **how to do something** over what is being done.

To iterate over an array with `for`:

1. Start with `var i = 0`
2. Check if `i < array.length`
3. Execute loop body
4. Increment `i`
5. Go to step (2)

```
var array = [...]

// Imperative
for(var i=0; i < array.length;i++) {
  // do stuff
}
```

CodingLab Powered by RBK

Bootstrap

Imperative v. Declarative

Declarative code emphasizes **what is being done** over how to do something.

<u>To iterate over an array with each:</u>

1. Pass the array to each as the first argument, and
2. a function as the second argument that will be invoked on each element.

```
var array = [...]

// Imperative
for(var i=0; i < array.length;i++) {
  // do stuff
}

// Declarative
each(array, function(element) {
 // do stuff
});
```

CodingLab Powered by RBK

That's it!

for Higher Order Functions