Visit the Wondrous

FOREST OF FUNCTION EXPRESSIONS

LEVEL 1

FOREST OF FUNCTION EXPRESSIONS

FUNCTIONS ON THE FLY

Building functions within code execution rather than at program load time

```
function diffOfSquares (a, b) {
    return a*a - b*b;
}
```

Builds in memory immediately
when the program loads

FUNCTIONS ON THE FLY

Building functions within code execution rather than at program load time

```
function diffOfSquares (a, b) {
    return a*a - b*b;
}
```

Builds in memory immediately when the program loads

```
var diff =
```

FUNCTIONS ON THE FLY

Building functions within code execution rather than at program load time

```
function diffOfSquares (a, b) {
    return a*a - b*b;
}
```

Builds in memory immediately when the program loads

```
var diff =  function diffOfSquares (a, b) {
    return a*a - b*b;
};
```

The function keyword will now assign the following function to the variable.

FUNCTIONS ON THE FLY

Building functions within code execution rather than at program load time

```
function diffOfSquares (a, b) {
    return a*a - b*b;
}
```

Builds in memory immediately when the program loads

```
var diff =  function diffOfSquares (a, b) {
    return a*a - b*b;
};
```

Needs a semicolon to complete the assignment statement in a file.

FUNCTIONS ON THE FLY

Building functions within code execution rather than at program load time

```
function diffOfSquares (a, b) {
    return a*a - b*b;
}
```

Builds in memory immediately when the program loads

```
var diff =  function diffOfSquares (a, b) {
    return a*a - b*b;
};
```

Now the function builds ONLY when this line of code is reached.

```
diff( 9, 5 );
```

→ 56

FUNCTIONS ON THE FLY

Building functions within code execution rather than at program load time

```
function diffOfSquares (a, b) {
    return a*a - b*b;
}
```

```
var diff =  function diffOfSquares (a, b) {
    return a*a - b*b;
};
```

```
diff( 9, 5 );
```

→ 56

This name is optional in JavaScript since we now use the variable name.

Notice the variable name needs parentheses, parameters and a semicolon to execute the function it contains.

ANONYMOUS FUNCTIONS

No need for naming the function a second time

```
var diff =  function diffOfSquares (a, b) {
    return a*a - b*b;
};
```

ANONYMOUS FUNCTIONS

No need for naming the function a second time

```
var diff =  function                    (a, b) {
    return a*a - b*b;
};
```

ANONYMOUS FUNCTIONS

No need for naming the function a second time

```
var diff =  function (a, b) {
    return a*a - b*b;
};
```

Look Ma, no name!

```
diff( 4, 2 );
```

→ 12

Parameters are still passed to the variable name, which JavaScript believes is a function.

ANONYMOUS FUNCTIONS

No need for naming the function a second time

```
var diff =  function  (a, b) {
    return a*a - b*b;
};
```

```
console.log(diff);
```

```
function (a, b) {
    return a*a - b*b;
}
```

Logging out using the variable name will display the entire function it contains.

STORED FUNCTIONS IN A NATIONAL PARK TERMINAL

A variable that holds a function can be passed into other functions

terminal.js

```
var greeting = function () {
    alert("Thanks for visiting the Badlands!\n" +
          "We hope your stay is...better than most.");
};


...


closeTerminal( greeting );


function closeTerminal( message ){
    ...
    message();
    ...
}
```

The greeting variable is passed in as a parameter to an existing declared function.

STORED FUNCTIONS IN A NATIONAL PARK TERMINAL

A variable that holds a function can be passed into other functions

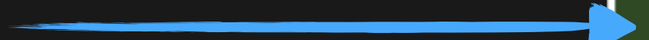
terminal.js

```
var greeting = function () {
    alert("Thanks for visiting the Badlands!\n" +
          "We hope your stay is...better than most.");
};


...


closeTerminal( greeting );


function closeTerminal( message ){
    ...
    message();
    ...
}
```



```
function closeTerminal( message ){
    ...
    message();
    ...
}
```

STORED FUNCTIONS IN A NATIONAL PARK TERMINAL

A variable that holds a function can be passed into other functions

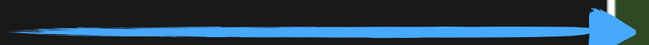
terminal.js

```
var greeting = function () {
    alert("Thanks for visiting the Badlands!\n" +
          "We hope your stay is...better than most.");
};


...


closeTerminal( greeting );


function closeTerminal( message ){
    ...
    message();
    ...
}
```



```
function closeTerminal(          ){
    ...
    message();
    ...
}
```

STORED FUNCTIONS IN A NATIONAL PARK TERMINAL

A variable that holds a function can be passed into other functions

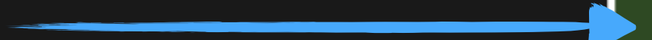
terminal.js

```
var greeting = function () {
    alert("Thanks for visiting the Badlands!\n" +
          "We hope your stay is...better than most.");
};


...


closeTerminal( greeting );


function closeTerminal( message ){
    ...
    message();
    ...
}
```



```
function closeTerminal( greeting ){
    ...
    message();
    ...
}
```

STORED FUNCTIONS IN A NATIONAL PARK TERMINAL

A variable that holds a function can be passed into other functions

terminal.js

```
var greeting = function () {
    alert("Thanks for visiting the Badlands!\n" +
          "We hope your stay is...better than most.");
};


...


closeTerminal( greeting );


function closeTerminal( message ){
    ...
    message();
    ...
}
```


```
function closeTerminal( greeting ){
    ...
    greeting();
    ...
}
```

NOW, WHAT IF WE HAD MULTIPLE GREETINGS?

Function expressions can give flexibility in choosing which functionality to build

terminal.js

```
var greeting = function () {
    alert("Thanks for visiting the Badlands!\n" +
          "We hope your stay is...better than most.");
};


...


closeTerminal( greeting );


function closeTerminal( message ){
    ...
    message();
    ...
}
```


```
function closeTerminal( greeting ){
    ...
    greeting();
    ...
}
```

NOW, WHAT IF WE HAD MULTIPLE GREETINGS?

Function expressions can give flexibility in choosing which functionality to build

terminal.js

```
var greeting;
...some code sets newCustomer to true or false...
if( newCustomer ){
    greeting = function () {
        alert("Thanks for visiting the Badlands!\n" +
              "We hope your stay is...better than most.");
    };
} else {
    greeting = function () {
        alert("Welcome back to the Badlands!\n" +
              "Guess they aren't so bad huh?");
    };
}
closeTerminal( greeting );
function closeTerminal( message ){
    ...
    message();
    ...
}
```

```
var newCustomer = false;
```

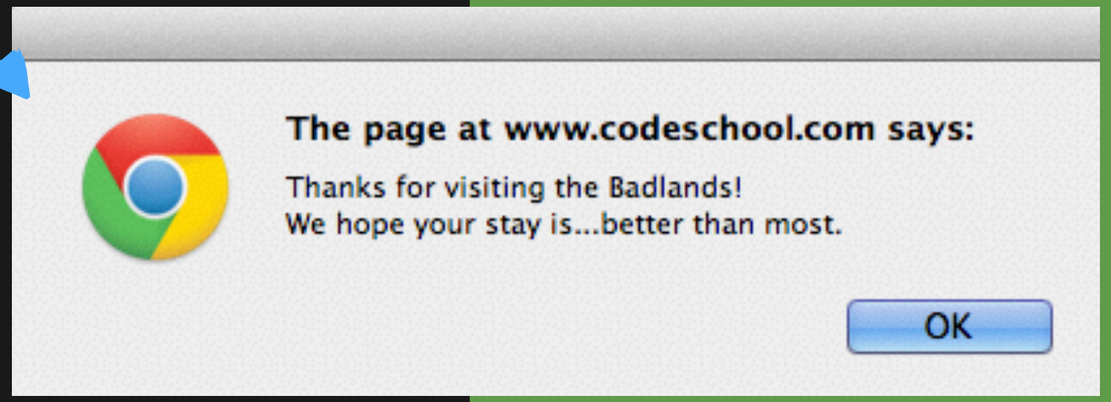
NOW, WHAT IF WE HAD MULTIPLE GREETINGS?

Function expressions can give flexibility in choosing which functionality to build

terminal.js

```
var greeting;
...some code sets newCustomer to true or false...
if( newCustomer ){
    greeting = function () {
        alert("Thanks for visiting the Badlands!\n" +
              "We hope your stay is...better than most.");
    };
} else {
    greeting = function () {
        alert("Welcome back to the Badlands!\n" +
              "Guess they aren't so bad huh?");
    };
}
closeTerminal( greeting );
function closeTerminal( message ){
    ...
    message();
    ...
}
```

```
var newCustomer = true;
```



Visit the Wondrous

FOREST OF FUNCTION EXPRESSIONS

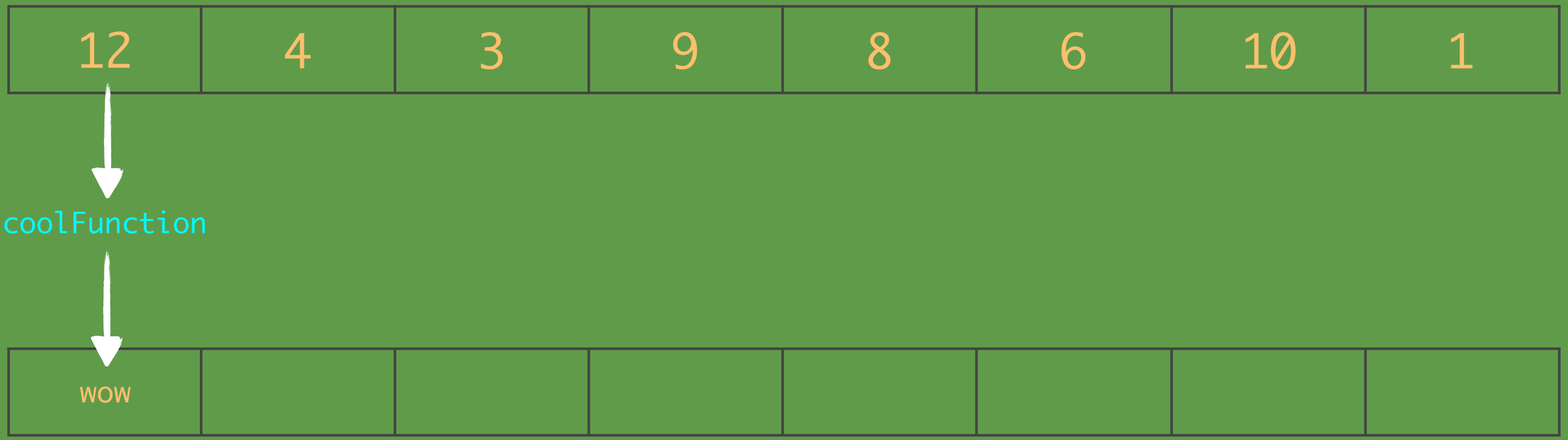
USING FE'S WITH ARRAYS AND MAP()

A function expression is just that...an expression. We can pass them without variables!

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

The map( ) method will always take in a function as a parameter, and return a new array with the results.

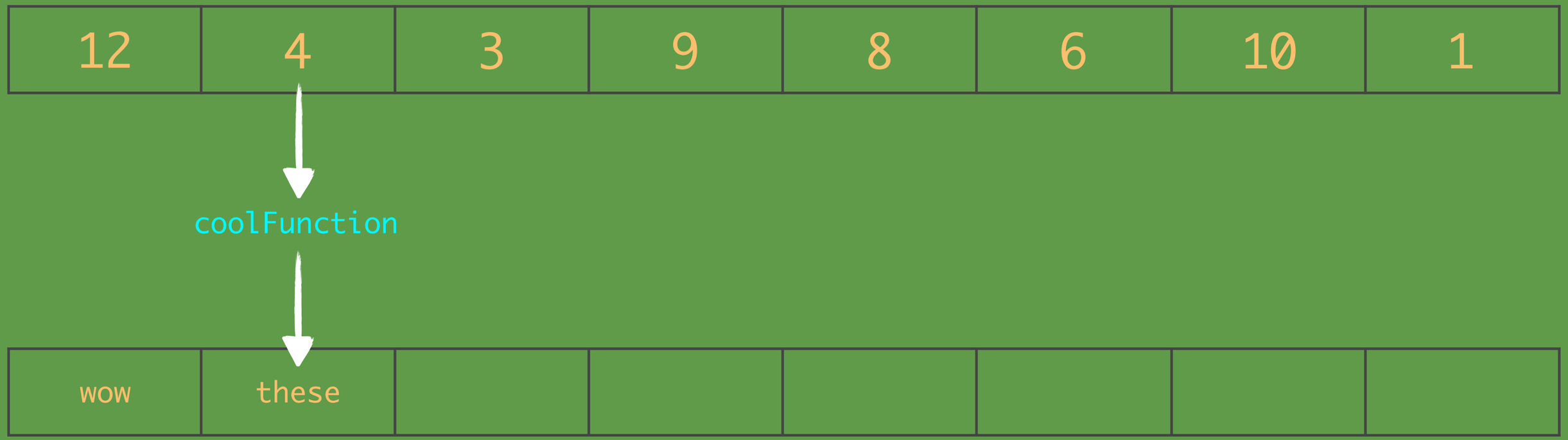


USING FE'S WITH ARRAYS AND MAP()

A function expression is just that...an expression. We can pass them without variables!

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

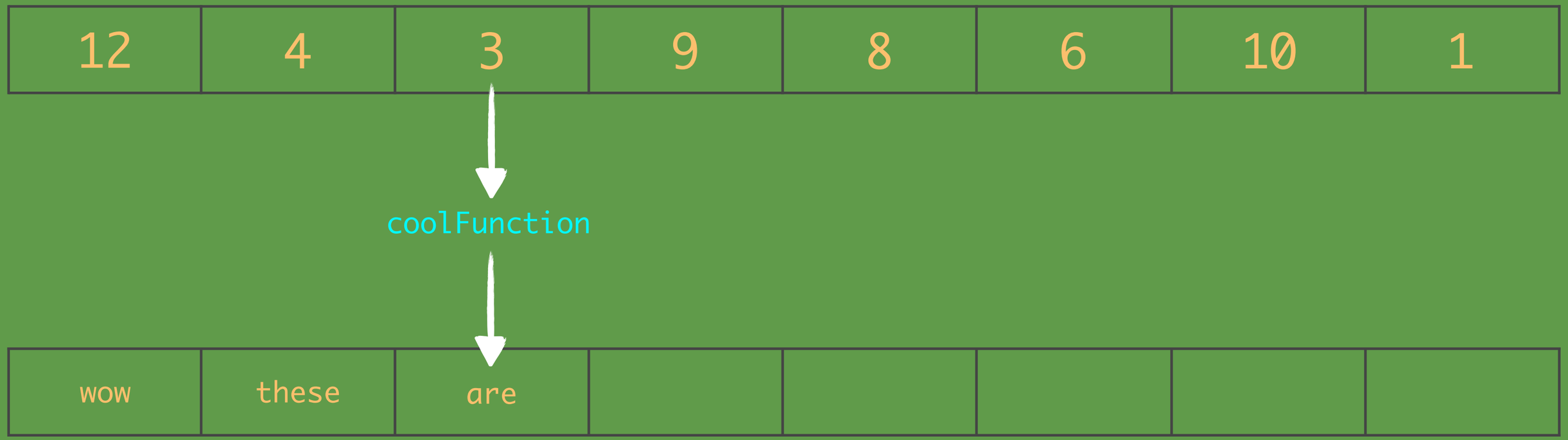


USING FE'S WITH ARRAYS AND MAP()

A function expression is just that...an expression. We can pass them without variables!

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

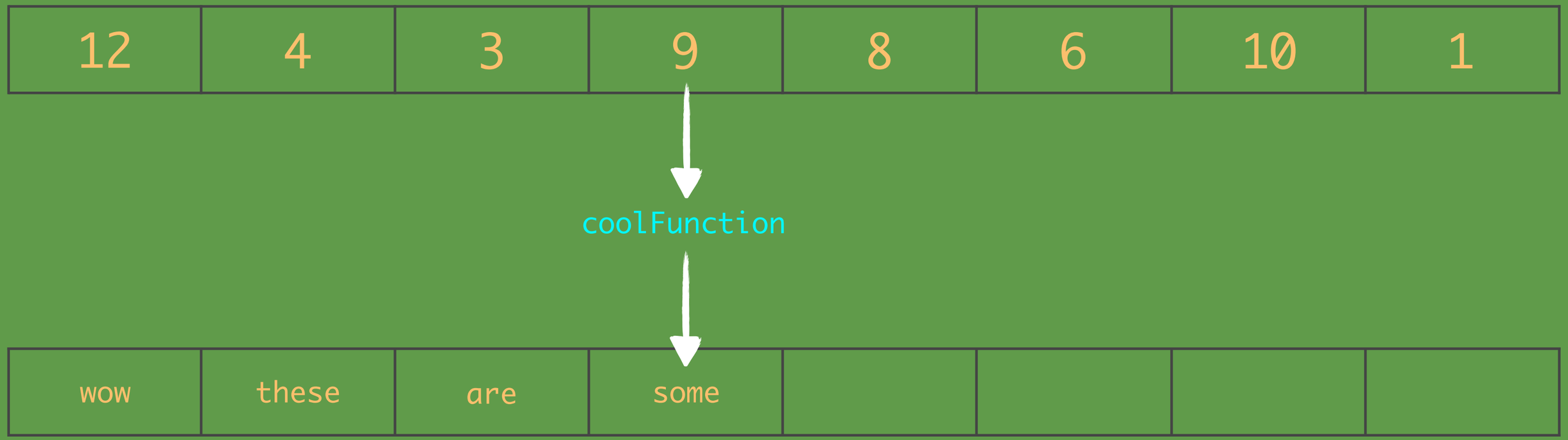


USING FE'S WITH ARRAYS AND MAP()

A function expression is just that...an expression. We can pass them without variables!

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

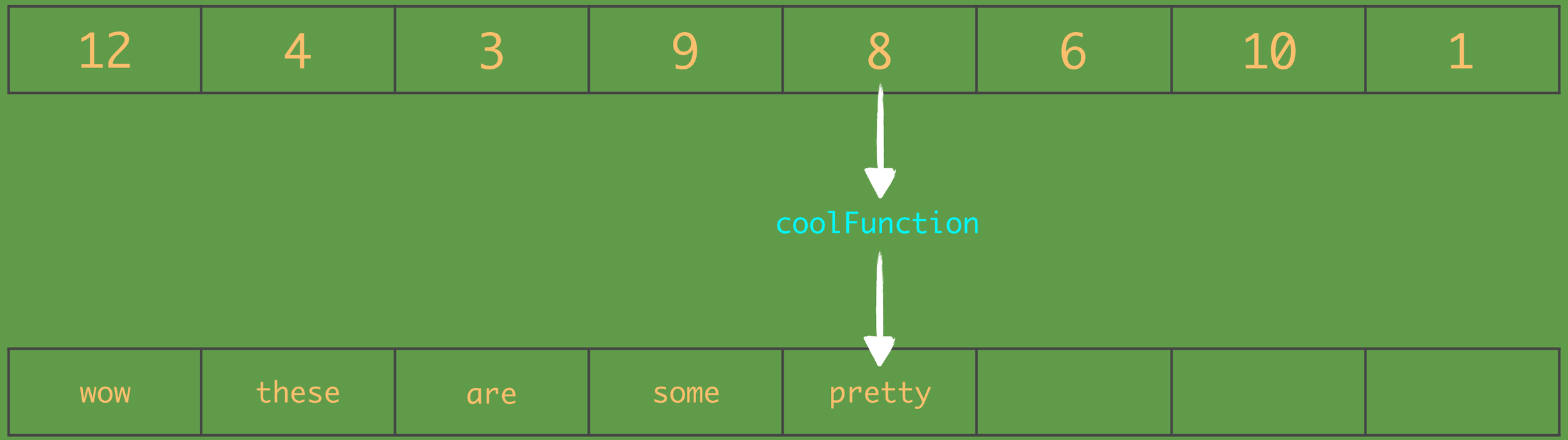


USING FE'S WITH ARRAYS AND MAP()

A function expression is just that...an expression. We can pass them without variables!

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```



USING FE'S WITH ARRAYS AND MAP()

A function expression is just that...an expression. We can pass them without variables!

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

USING FE'S WITH ARRAYS AND MAP()

A function expression is just that...an expression. We can pass them without variables!

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

USING FE'S WITH ARRAYS AND MAP()

A function expression is just that...an expression. We can pass them without variables!

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

USING FE'S WITH ARRAYS AND MAP()

A function expression is just that...an expression. We can pass them without variables!

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

USING FE'S WITH ARRAYS AND MAP()

Map works like a loop that applies a function to each array index

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

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

    results[i] =

}
```

USING FE'S WITH ARRAYS AND MAP()

Map works like a loop that applies a function to each array index

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( *some coolFunction goes here* );
```

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

    results[i] = coolFunction(numbers[i]);

}
```

The array's map conveniently takes this entire loop format and consolidates it to one nice line of code.

USING FE'S WITH ARRAYS AND MAP()

Let's pass in function that will double each cell's value in our numbers array.

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( function (arrayCell) {
                              return arrayCell * 2;
                           }
);
```

We build an anonymous function for map's parameter, which takes in the contents of each cell of numbers and returns a doubled value to results.

12	4	3	9	8	6	10	1

USING FE'S WITH ARRAYS AND MAP()

Let's pass in function that will double each cell's value in our numbers array.

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( function (arrayCell) {
                                return arrayCell * 2;
                            }
);
```

Don't forget to close both your anonymous function with a } and the map method with a), while also adding a semicolon in order to execute the map.

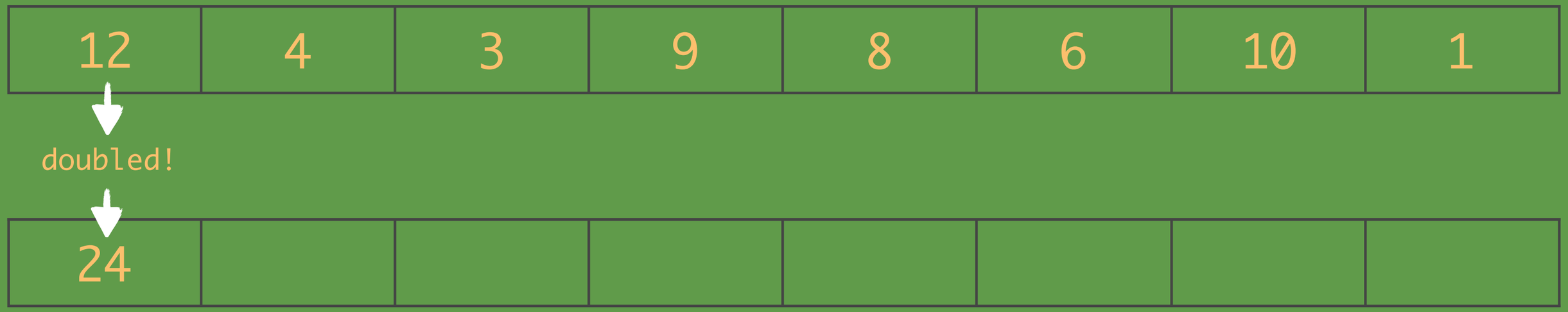
12	4	3	9	8	6	10	1

USING FE'S WITH ARRAYS AND MAP()

Let's pass in function that will double each cell's value in our numbers array.

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( function (arrayCell) {
                              return arrayCell * 2;
                           }
);
```

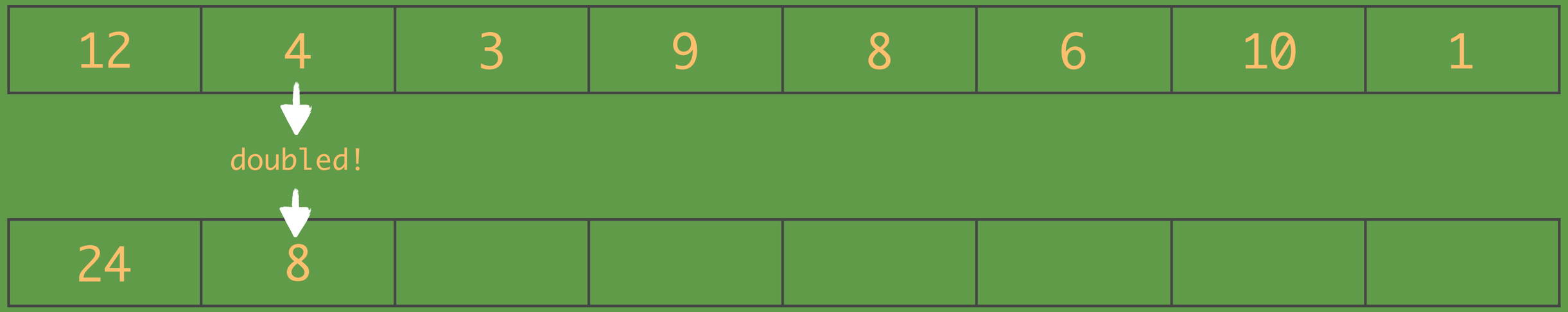


USING FE'S WITH ARRAYS AND MAP()

Let's pass in function that will double each cell's value in our numbers array.

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( function (arrayCell) {
                              return arrayCell * 2;
                           }
);
```

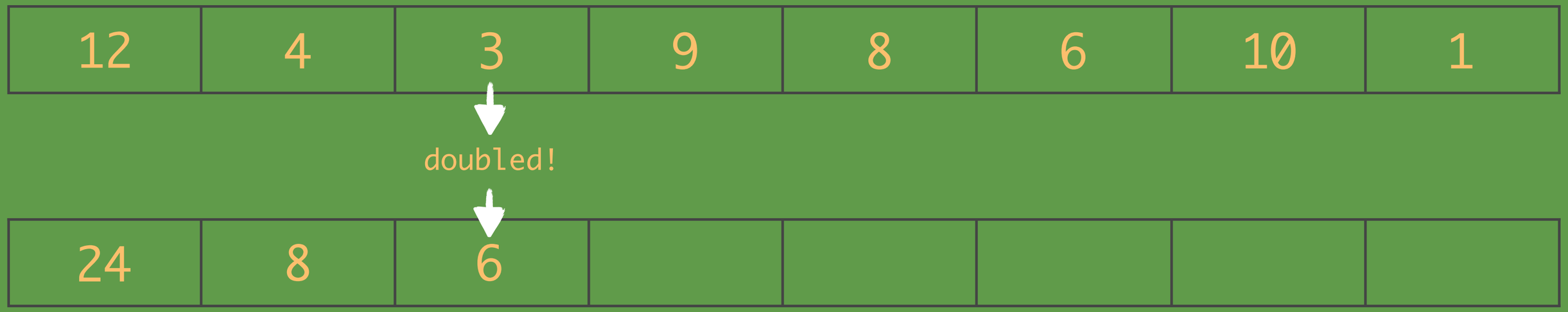


USING FE'S WITH ARRAYS AND MAP()

Let's pass in function that will double each cell's value in our numbers array.

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( function (arrayCell) {
                              return arrayCell * 2;
                           }
);
```

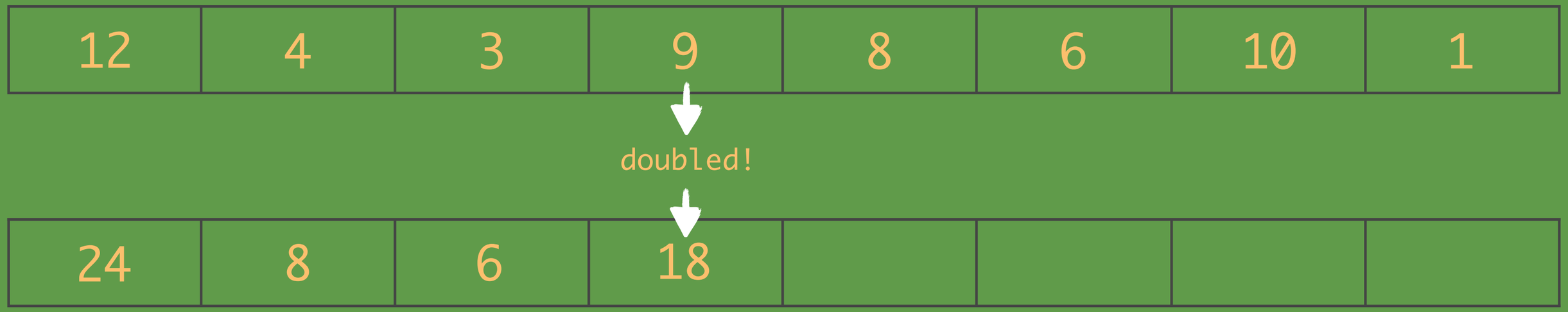


USING FE'S WITH ARRAYS AND MAP()

Let's pass in function that will double each cell's value in our numbers array.

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( function (arrayCell) {
                              return arrayCell * 2;
                           }
);
```

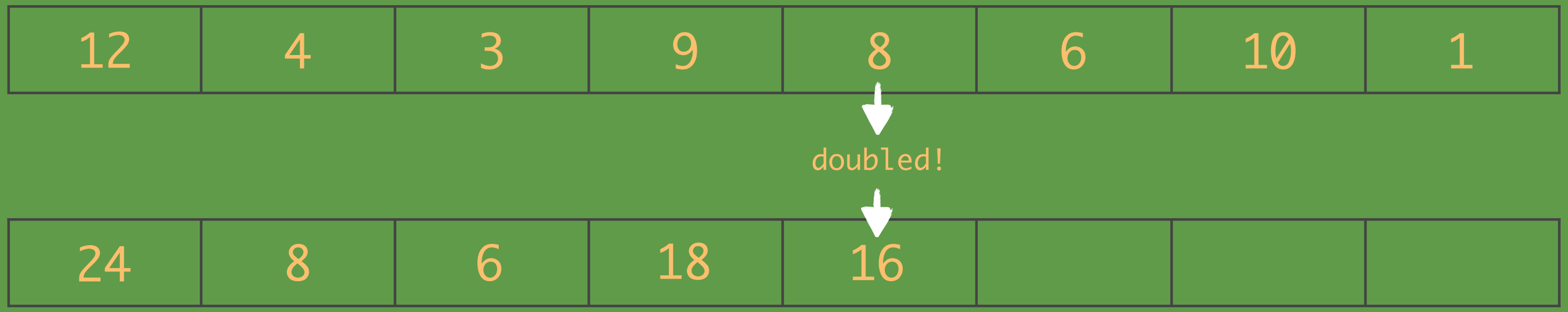


USING FE'S WITH ARRAYS AND MAP()

Let's pass in function that will double each cell's value in our numbers array.

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( function (arrayCell) {
                              return arrayCell * 2;
                           }
);
```

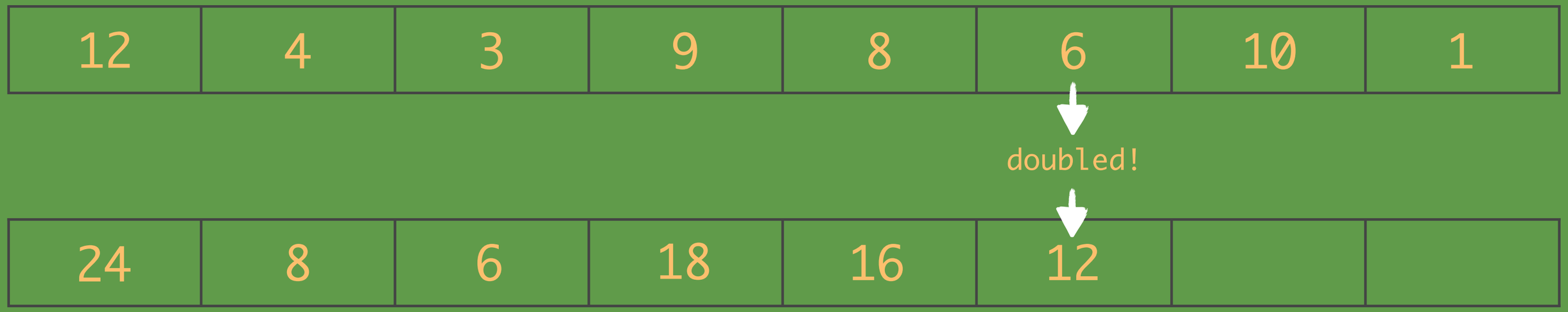


USING FE'S WITH ARRAYS AND MAP()

Let's pass in function that will double each cell's value in our numbers array.

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( function (arrayCell) {
                              return arrayCell * 2;
                           }
);
```



USING FE'S WITH ARRAYS AND MAP()

Let's pass in function that will double each cell's value in our numbers array.

```
var numbers = [12, 4, 3, 9, 8, 6, 10, 1];
```

```
var results = numbers.map( function (arrayCell) {
                              return arrayCell * 2;
                           }
);
```