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

```
let 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

```
let 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

```
let 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

```
let 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;
}
```

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

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

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

→ 56

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

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

ANONYMOUS FUNCTIONS

No need for naming the function a second time

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

ANONYMOUS FUNCTIONS

No need for naming the function a second time

```
let 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

```
let 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

```
let 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

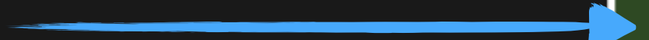
terminal.js

```
let 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

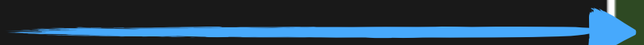
terminal.js

```
let 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

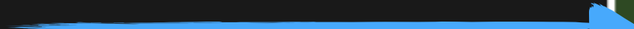
terminal.js

```
let 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

```
let 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

```
let 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

```
let 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();
    ...
}
```

```
let newCustomer = false;
```

NOW, WHAT IF WE HAD MULTIPLE GREETINGS?

Function expressions can give flexibility in choosing which functionality to build

terminal.js

```
let 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();
    ...
}
```

```
let newCustomer = true;
```

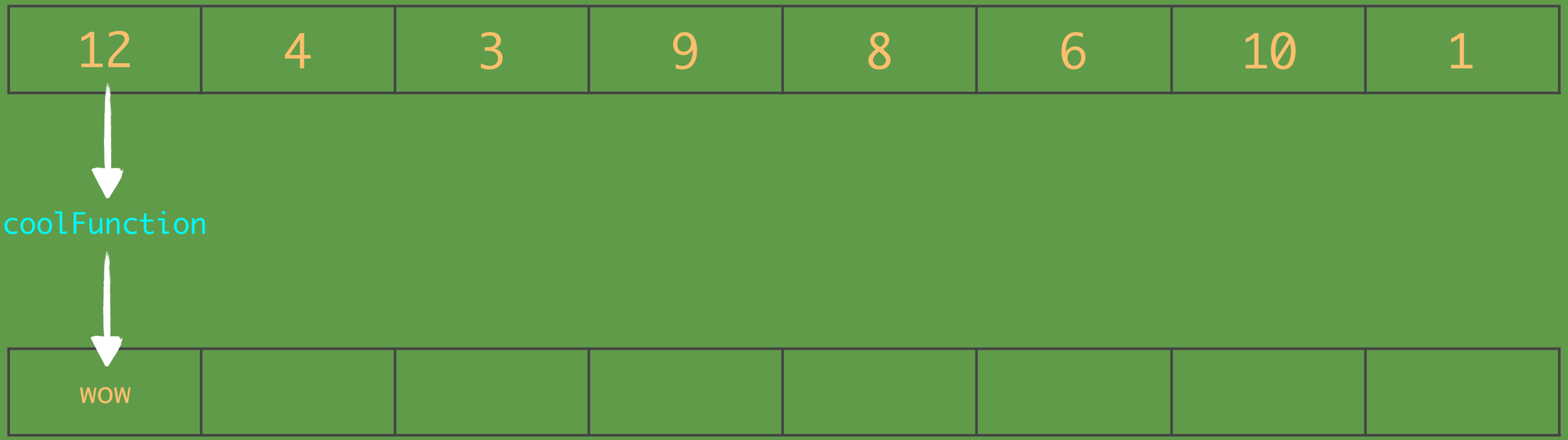
USING FE'S WITH ARRAYS AND MAP()

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

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

```
let 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.

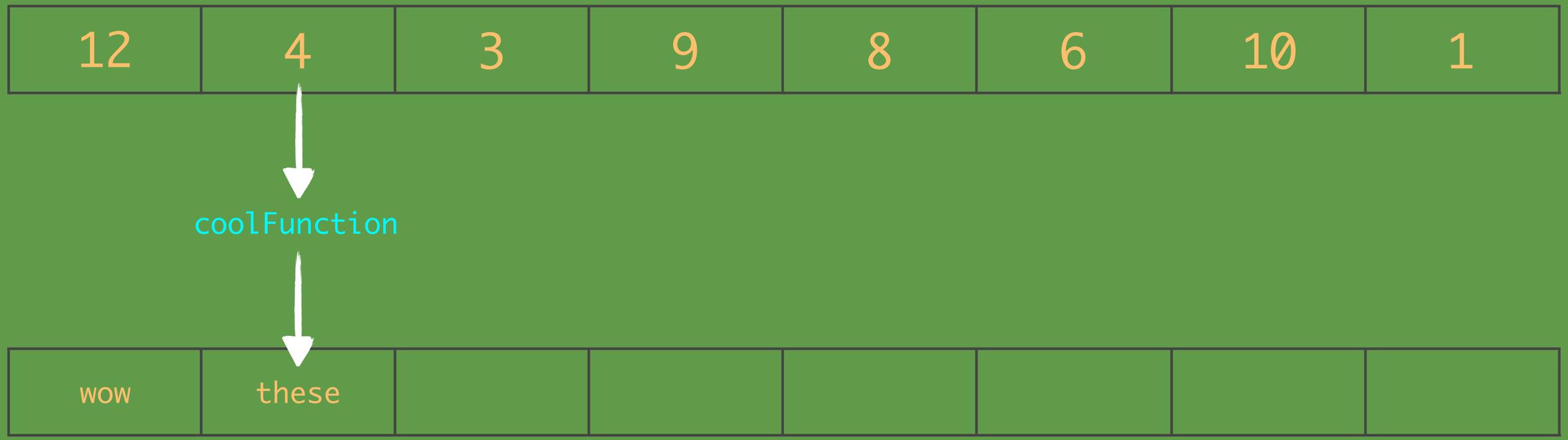


USING FE'S WITH ARRAYS AND MAP()

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

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

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

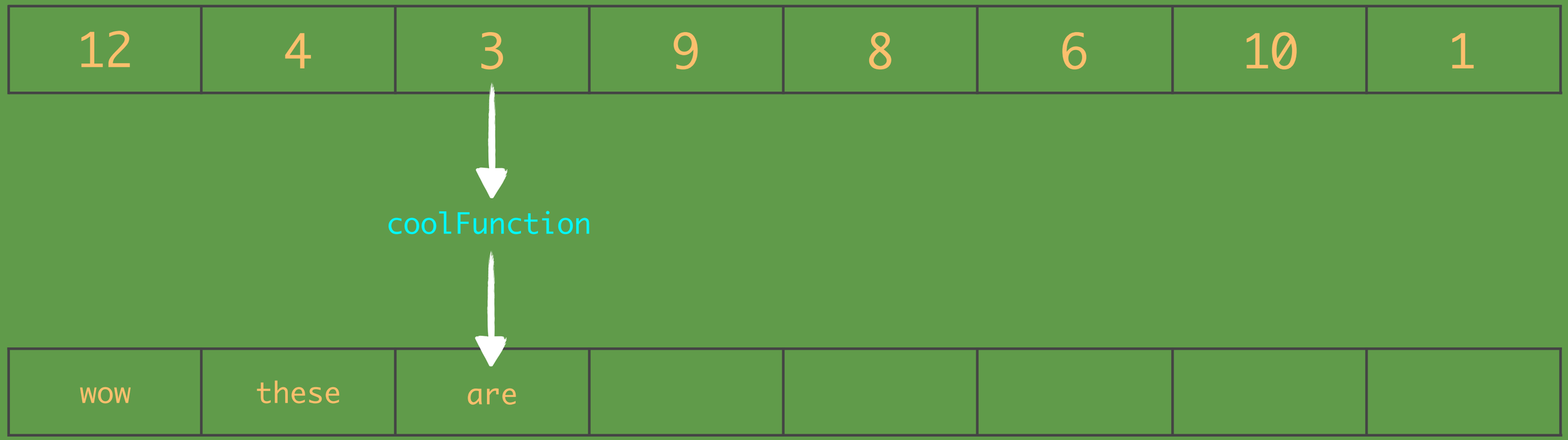


USING FE'S WITH ARRAYS AND MAP()

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

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

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

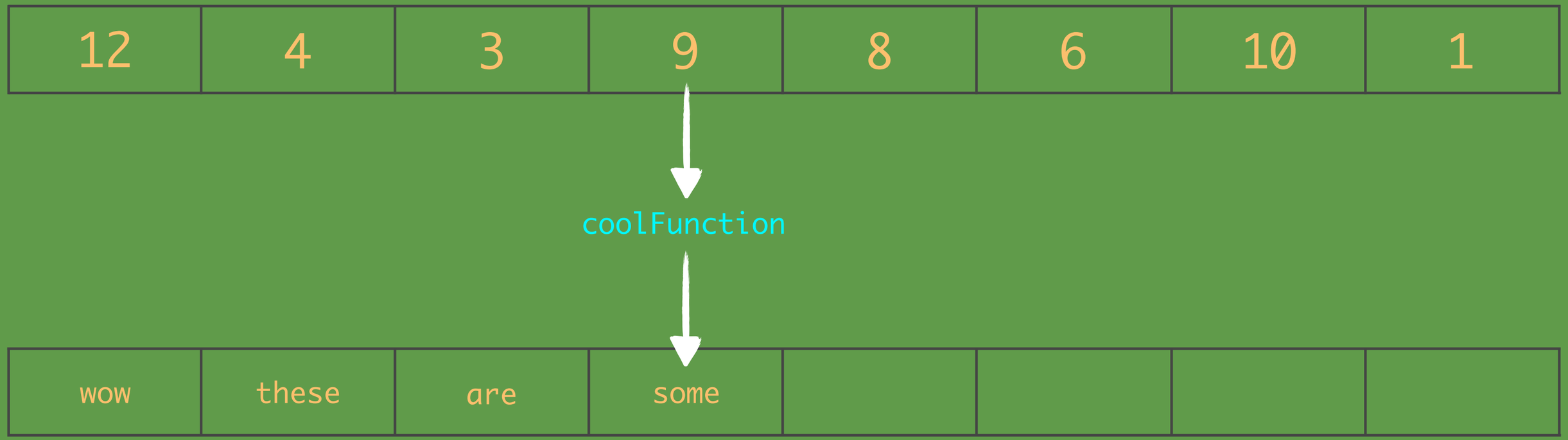


USING FE'S WITH ARRAYS AND MAP()

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

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

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

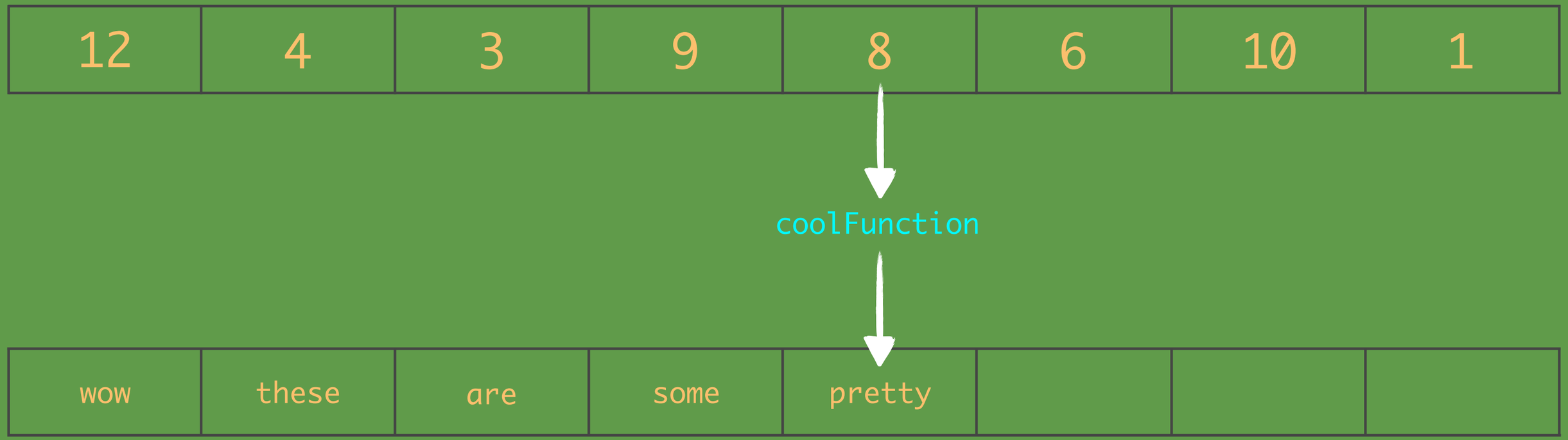


USING FE'S WITH ARRAYS AND MAP()

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

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

```
let 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!

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

```
let 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!

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

```
let 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!

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

```
let 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!

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

```
let 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

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

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

```
let results = [ ];
for(let 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

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

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

```
let results = [ ];
for(let 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.

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

```
let 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.

USING FE'S WITH ARRAYS AND MAP()

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

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

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

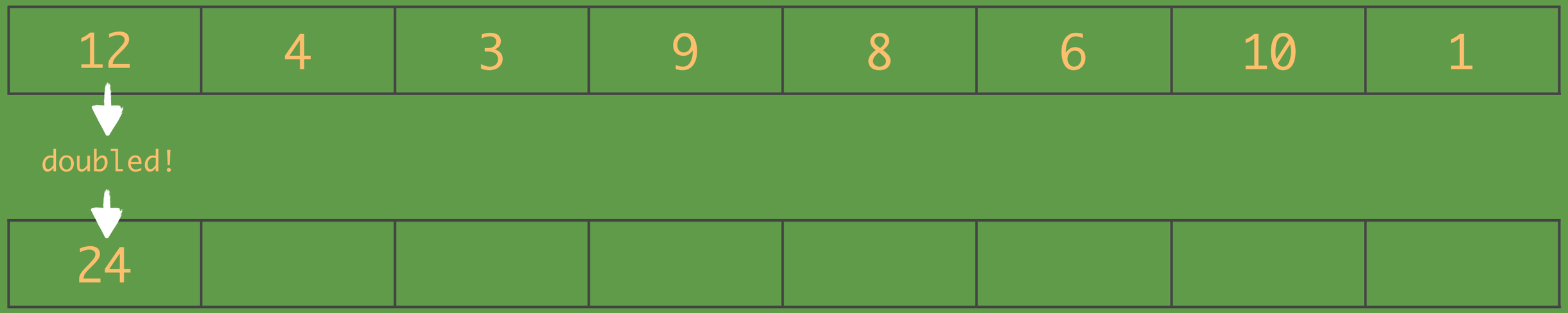
Don't forget to close both your anonymous function with a } and the map method with a), while also adding a semicolon in order to execute the map.

USING FE'S WITH ARRAYS AND MAP()

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

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

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

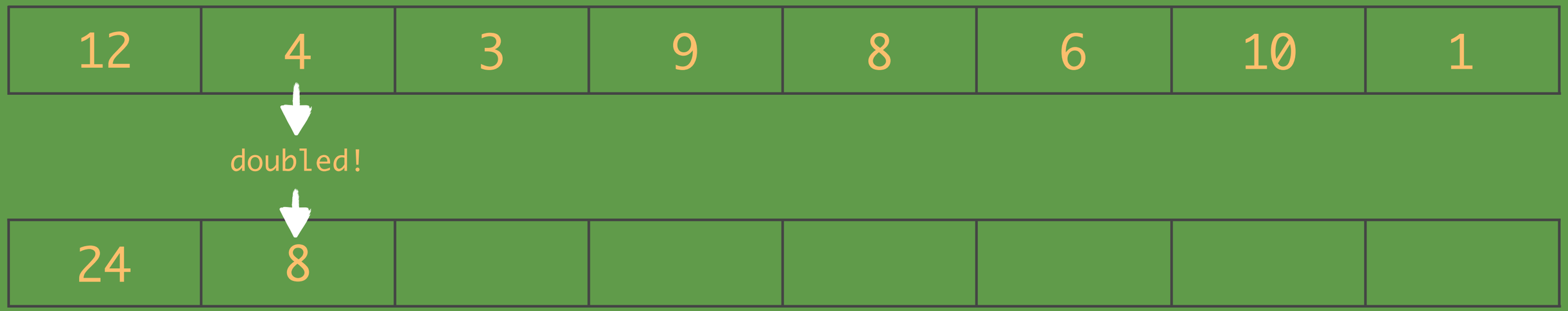


USING FE'S WITH ARRAYS AND MAP()

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

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

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

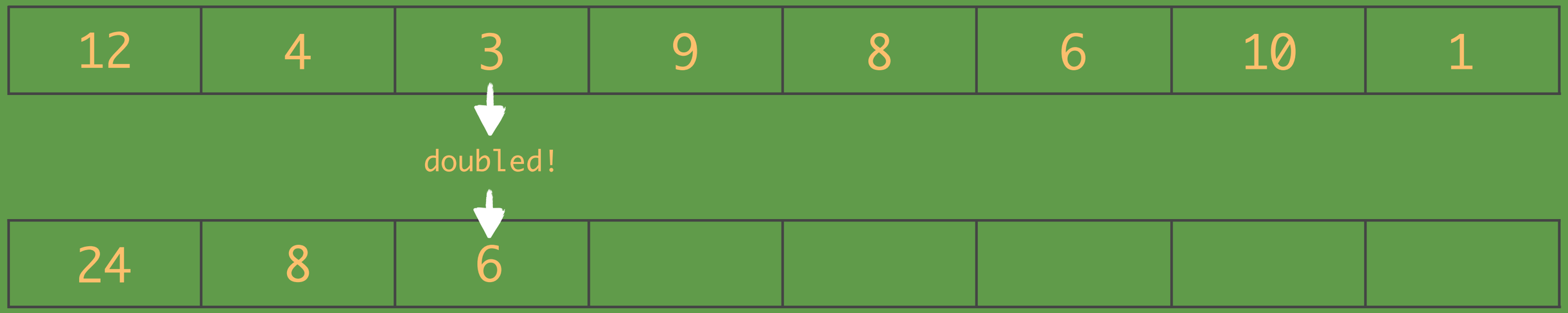


USING FE'S WITH ARRAYS AND MAP()

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

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

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

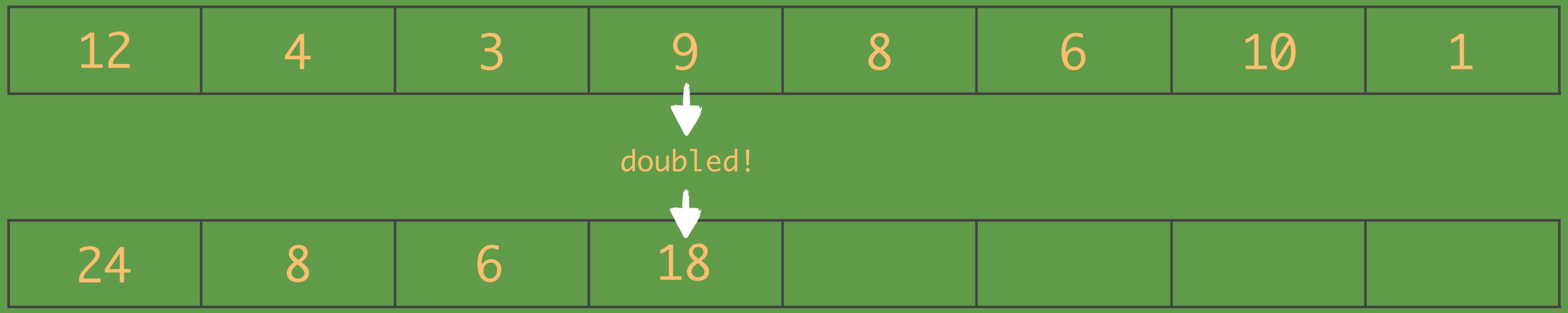


USING FE'S WITH ARRAYS AND MAP()

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

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

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

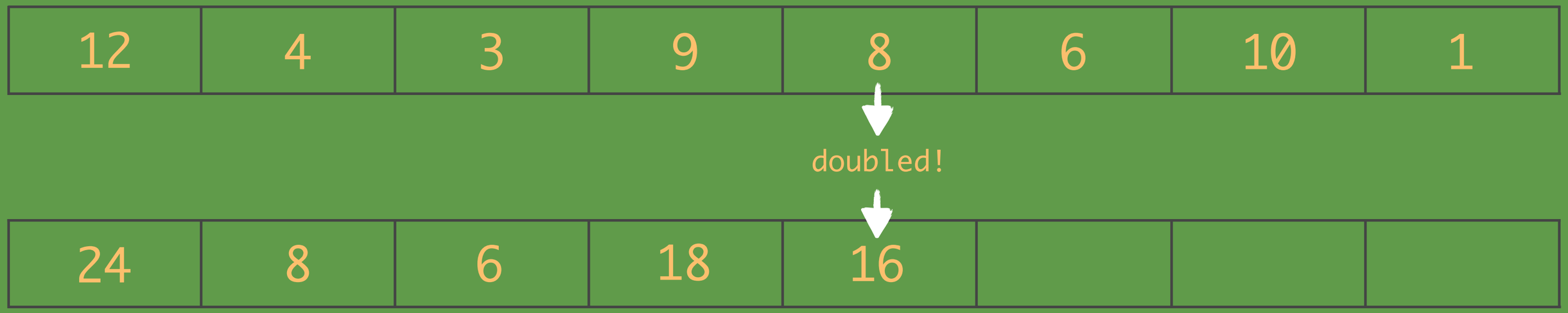


USING FE'S WITH ARRAYS AND MAP()

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

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

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

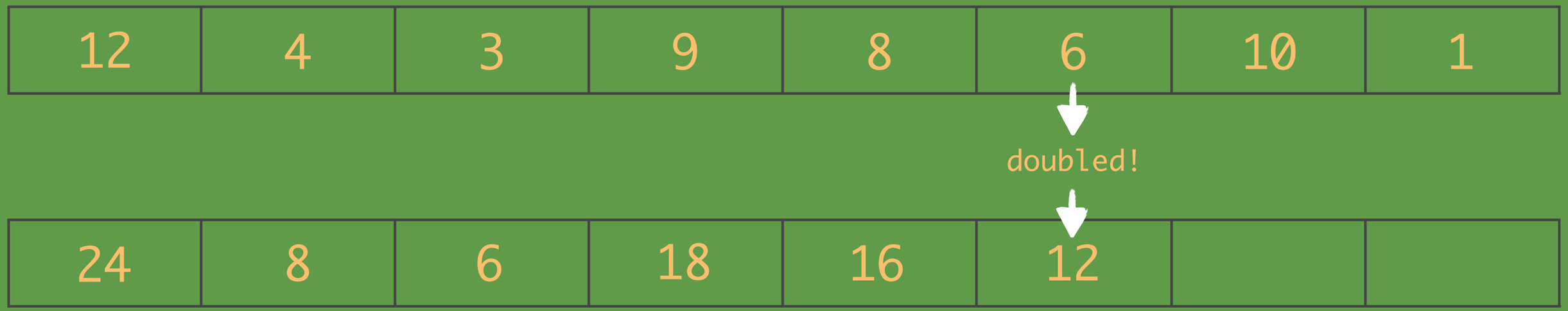


USING FE'S WITH ARRAYS AND MAP()

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

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

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

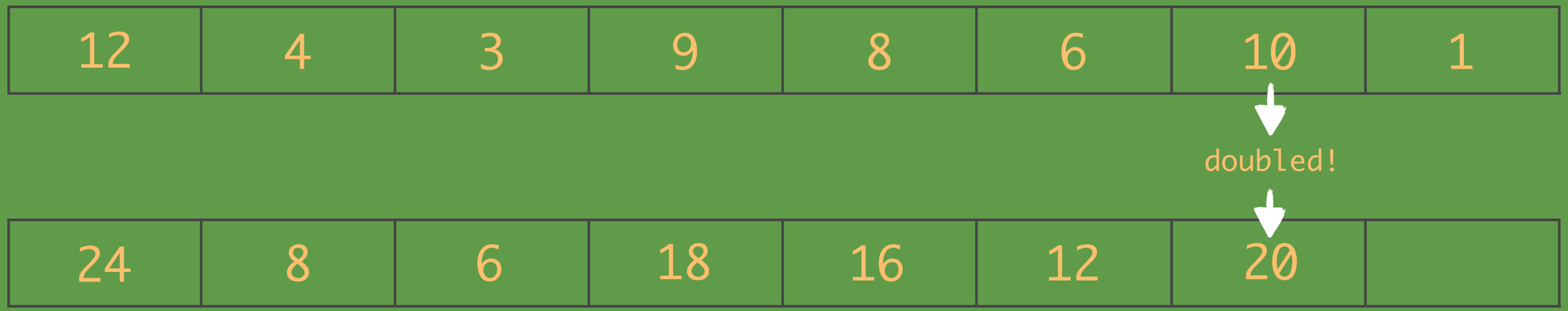


USING FE'S WITH ARRAYS AND MAP()

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

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

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

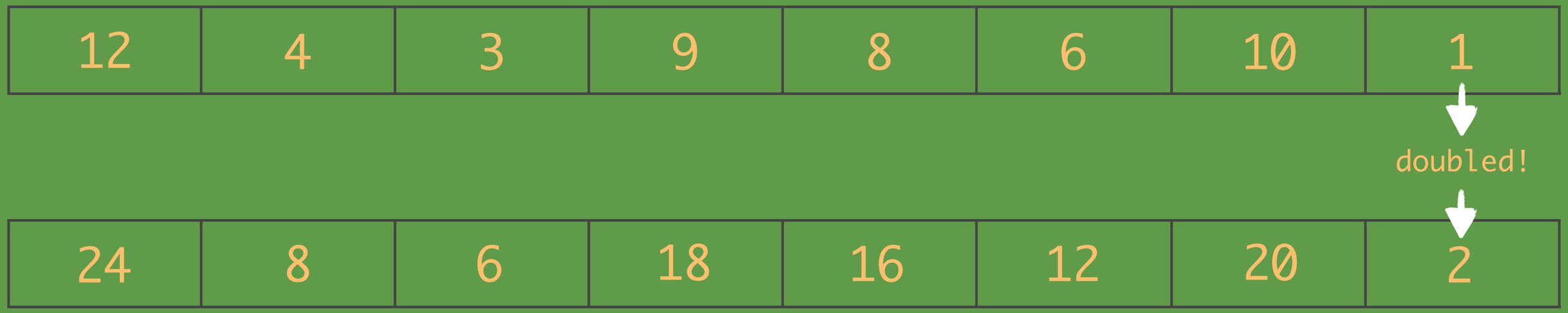


USING FE'S WITH ARRAYS AND MAP()

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

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

```
let 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.

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

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

```
console.log(results);
```

→ `[24, 8, 6, 18, 16, 12, 20, 2]`

USING FE'S WITH ARRAYS AND MAP()

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

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

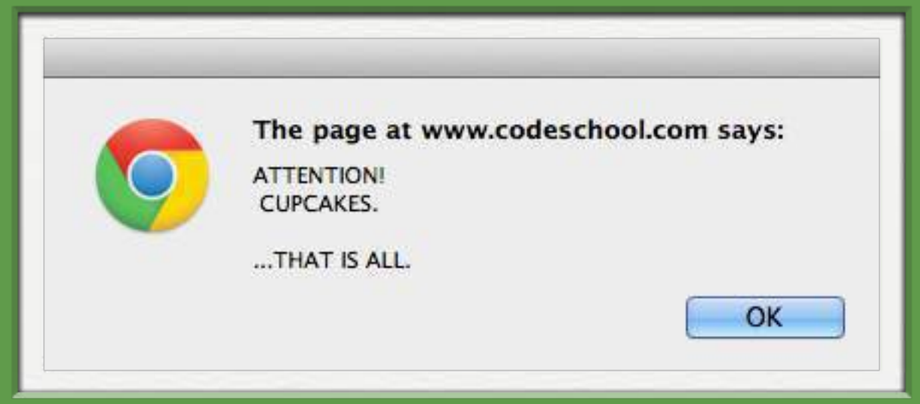
```
let results = numbers.map( function (arrayCell) { return arrayCell * 2; } );
```

Short functions are often built in one line for clarity and simplicity.

```
console.log(results);
```

→ [24, 8, 6, 18, 16, 12, 20, 2]

```
let sweetAnnouncement = function () { alert("ATTENTION!\n CUPCAKES.\n\n...THAT IS ALL.") };
sweetAnnouncement();
```



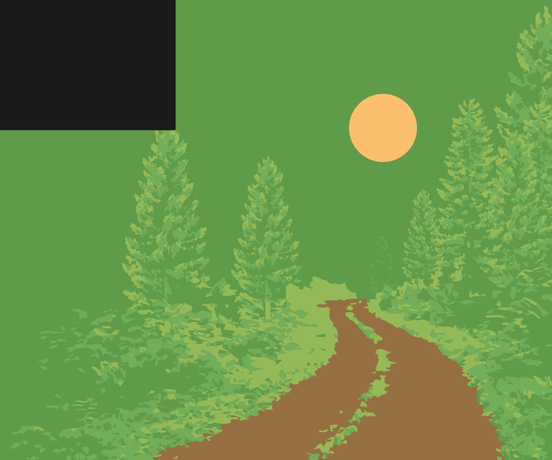

```
let fruits = [ "Apple", "Orange", "Pineapple", "Cranberry", "Pomegranate" ];
let fruitJuice = fruits.map( function (fruit) { return "\n" + fruit + " juice"; } );
alert(fruitJuice);
```

```
function mystery () {

    *some mystery code...ooh, spooky.*

    return *a function expression*

}
```

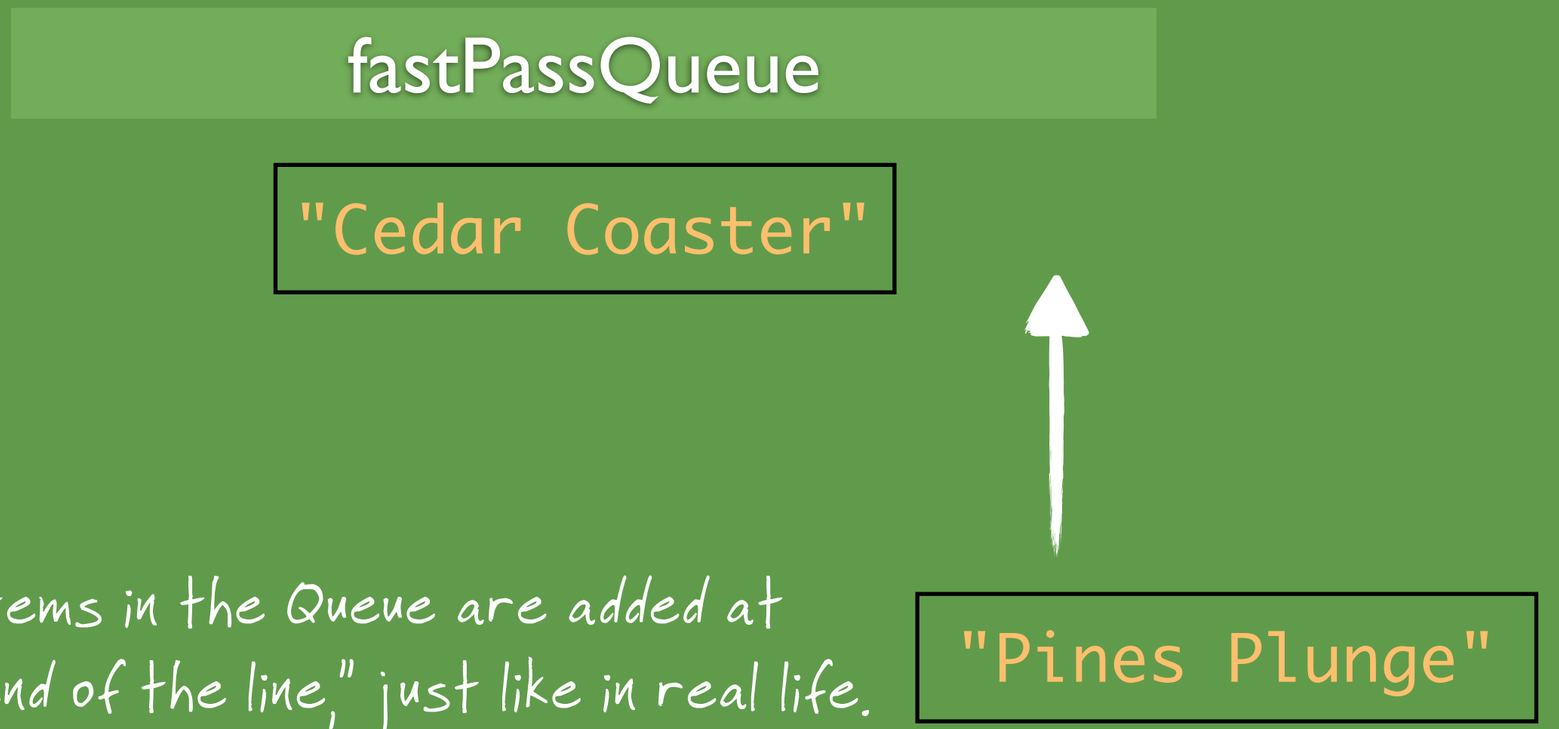


A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

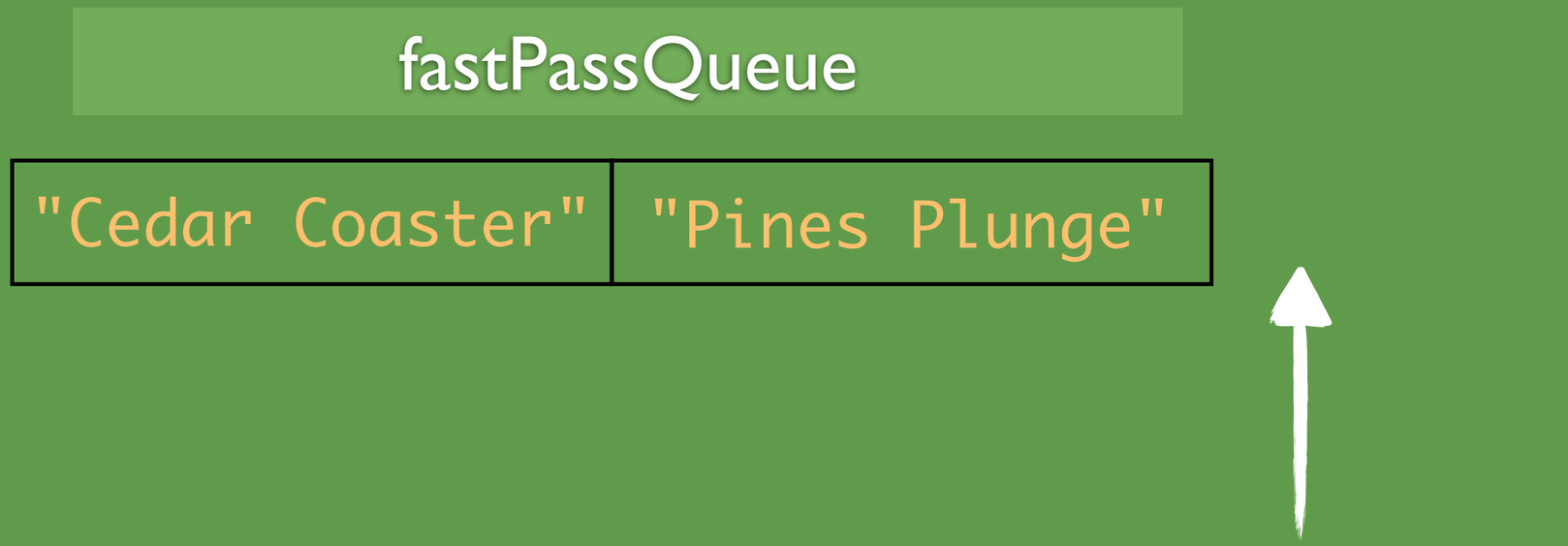


A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```



New items in the Queue are added at "the end of the line," just like in real life.

A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

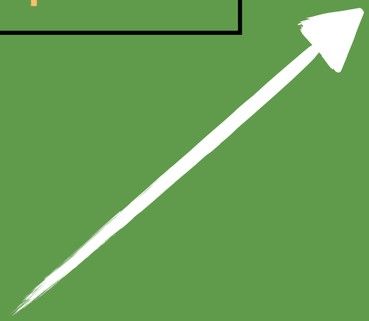
```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue

"Cedar Coaster"	"Pines Plunge"	"Birch Bumpers"

We know a method that adds cells to the back of arrays, right??

```
fastPassQueue.push("Pines Plunge");
```



"Pines Plunge"

A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue

"Cedar Coaster"	"Pines Plunge"	"Birch Bumpers"	"Pines Plunge"

We know a method that adds cells to the back of arrays, right??

```
fastPassQueue.push("Pines Plunge");
```

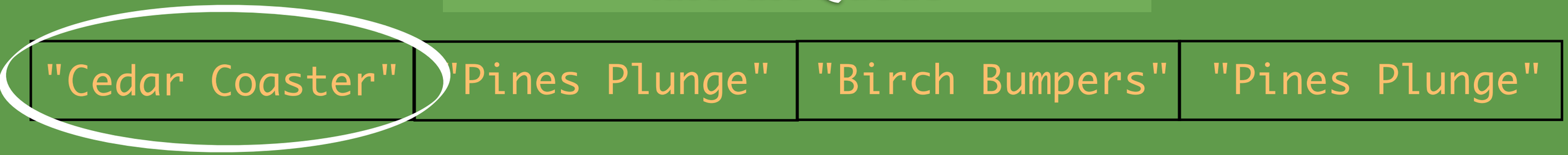
A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue



To empty the queue, however, we need a new method that removes cells from the front of the array!

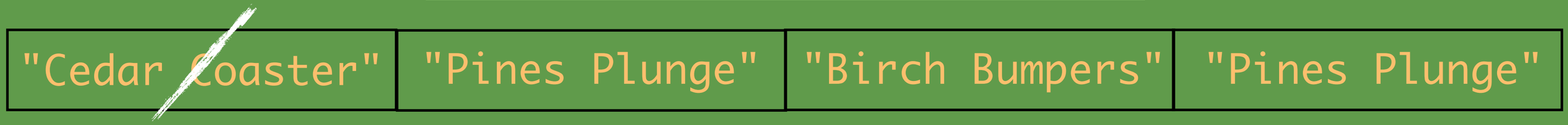
A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue




```
fastPassQueue.shift();
```

A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue

"Pines Plunge"	"Birch Bumpers"	"Pines Plunge"

```
fastPassQueue.shift();
```

→ "Cedar Coaster"

The shift method will return the cell that it removes from the front of the array.

A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue


```
fastPassQueue.shift();
```

→ "Cedar Coaster"

The shift method will return the cell that it removes from the front of the array.

A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue


```
fastPassQueue.shift();
```

→ "Cedar Coaster"

```
fastPassQueue.length;
```

→ 3

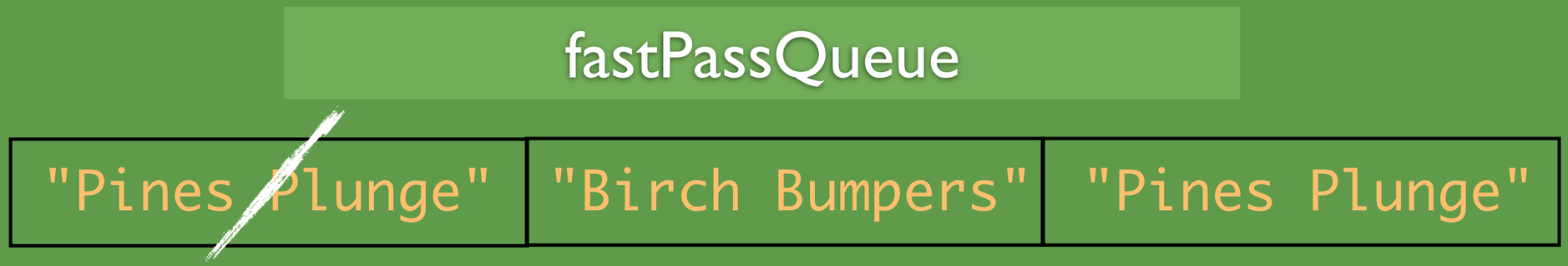
Like pop and push, shift will automatically modify the array's length.

A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```




```
fastPassQueue.shift();
```

→ `"Cedar Coaster"`

```
let firstFastPass = fastPassQueue.shift();
```

A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue

"Birch Bumpers"	"Pines Plunge"

```
fastPassQueue.shift();
```

→ "Cedar Coaster"

```
let firstFastPass = fastPassQueue.shift();
```

A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue

"Birch Bumpers"	"Pines Plunge"

```
fastPassQueue.shift();
```

→ "Cedar Coaster"

```
let firstFastPass = fastPassQueue.shift();
console.log(firstFastPass);
```

→ "Pines Plunge"

shift always returns the first cell, whether you use it in an expression or store it in a variable.

A TICKET SYSTEM FOR THE FOREST THEME PARK

Using an array as a “queue” for Fast Pass delivery

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

fastPassQueue

"Birch Bumpers"	"Pines Plunge"

```
fastPassQueue.shift();
```

→ "Cedar Coaster"

```
let firstFastPass = fastPassQueue.shift();
console.log(firstFastPass);
```

→ "Pines Plunge"

Now, we'll build a simple ticket system using our queue!

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
function buildTicket ( allRides, passRides, pick ) {



}
```

This parameter will be the array of the rides and their wait times.

...and this will be the array of the next available Fast Pass rides.

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
function buildTicket ( allRides, passRides, pick ) {




}
```

This will be the actual ride for which our customer would like a ticket!

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
function buildTicket ( allRides, passRides, pick ) {
  if(passRides[0] == pick){


  }


}
```

If the next available Fast Pass is for the ride that the customer seeks, we'll give them that pass.

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
function buildTicket ( allRides, passRides, pick ) {
  if(passRides[0] == pick){
     let pass = passRides.shift();


  }



}
```

Here's our shift! It takes the front cell off the array and stores it in the pass variable.

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
function buildTicket ( allRides, passRides, pick ) {
  if(passRides[0] == pick){
     let pass = passRides.shift();
     return function ( ) { alert("Quick! You've got a Fast Pass to " + pass + "!");
           };
  }


}
```

Notice we are treating the function as an expression and returning it directly. No extra storage variable needed!

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
function buildTicket ( allRides, passRides, pick ) {
  if(passRides[0] == pick){
     let pass = passRides.shift();
     return function ( ) { alert("Quick! You've got a Fast Pass to " + pass + "!");
            };
  } else {
     for(let i = 0; i<allRides.length; i++){




     }
  }
}
```

To search for the ride the customer wants, we'll loop over the entire array of rides.

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
function buildTicket ( allRides, passRides, pick ) {
  if(passRides[0] == pick){
     let pass = passRides.shift();
     return function ( ) { alert("Quick! You've got a Fast Pass to " + pass + "!");
            };
  } else {
     for(let i = 0; i<allRides.length; i++){
        if(allRides[i][0] == pick){

        }
     }
  }
}
```

The ride names are contained within the first index of each subarray, or [0].

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
function buildTicket ( allRides, passRides, pick ) {
  if(passRides[0] == pick){
     let pass = passRides.shift();
     return function ( ) { alert("Quick! You've got a Fast Pass to " + pass + "!");
            };
  } else {
     for(let i = 0; i<allRides.length; i++){
        if(allRides[i][0] == pick){
           return function () { alert("This is your ticket to " + pick + "!\n" +
                                      "Your wait time is about " + allRides[i][1] + " minutes.");
           };
        }
     }
  }
}
```

The wait times are in the second index of the subarrays, or [1].

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
function buildTicket ( allRides, passRides, pick ) {
  if(passRides[0] == pick){
     let pass = passRides.shift();
     return function ( ) { alert("Quick! You've got a Fast Pass to " + pass + "!");
             };
  } else {
      for(let i = 0; i<allRides.length; i++){
         if(allRides[i][0] == pick){
            return function () { alert("This is your ticket to " + pick + "!\n" +
                                       "Your wait time is about " + allRides[i][1] + " minutes.");
            };

         }
     }
  }
}
```

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Birch Bumpers";
```

```
let ticket = buildTicket( parkRides, fastPassQueue, wantsRide );
```

When `buildTicket` returns the correct ticket function, we'll store it in a `ticket` variable for later use!

```
function buildTicket ( allRides, passRides, pick ) {
    ...
}
```

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Birch Bumpers";
```

```
let ticket = buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
ticket();
```

To call the function contained in the `ticket` variable, we need a set of parentheses and a semicolon.

```
function buildTicket ( allRides, passRides, pick ) {
    ...
}
```

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Birch Bumpers";
```

```
let ticket = buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
ticket();
```


```
function buildTicket ( allRides, passRides, pick ) {
    ...
}
```

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Cedar Coaster", "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Cedar Coaster";
```

Now, the desired ride matches the first Fast Pass!

```
let ticket = buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
ticket();
```


```
function buildTicket ( allRides, passRides, pick ) {
    ...
}
```

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [                "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Cedar Coaster";
```

```
let ticket = buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
ticket();
```


```
function buildTicket ( allRides, passRides, pick ) {
    ...
}
```

LET'S MAKE SOME TICKETS!

Since functions can be treated as expressions, they can also be returned like values!

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Cedar Coaster";
```

```
let ticket = buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
ticket();
```


```
function buildTicket ( allRides, passRides, pick ) {
    ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
let ticket = buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
function buildTicket ( allRides, passRides, pick ) {
    ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
function buildTicket ( allRides, passRides, pick ) {
    ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
function buildTicket ( allRides, passRides, pick ) {
    ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )
```

So far, all we get back is a function expression. Need more in order to call it!

The contents of pass are saved in a process called "closure," which we'll explore in the next level. For now, we know it's filled with the first available Fast Pass.

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide );
```

```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )
```

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )()                ;
```

```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )()
```

Okay, now we've given our expression some parameter parentheses. We're on the right track!

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )()                ;
```

```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )()
```

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )();
```

```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )();
```

Yep, a semicolon gives the instruction to execute the function!

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )();
```


```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )();
```

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [                    "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )();
```


```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )();
```

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
let parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55] ,
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
let fastPassQueue = [ "Birch Bumpers", "Pines Plunge" ];
```

```
let wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )();
```


```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )();
```

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```