

JavaScript: to ES6 and Beyond

Why are we here today?

Who is Thinkful



Update with thoughtful verbage & styling

Why Learn This?

- JavaScript is constantly improving
- Need to understand new syntax
- Write cleaner and more concise code

ES6 Workshop

bit.ly/tf-es6-start-made-up-link

What is a variable in Javascript

```
var firstVariable = 20
```

let / const

- **let**
 - Now preferred instead of var
 - Block-scoped
 - Var is function-scoped
 - Better memory management
- **const**
 - Stands for constant
 - Used for any values we do not want changed
 - NO_MORE_ALL_CAPS_VARIABLES

var theOldWay = true

let theNewWay = 'I am the new way'

const name = 'Sally Student'

name = 'Susie Student' // error

Variable Examples

codepen.io



The screenshot shows a CodePen editor with the title "Let vs Var && Const" and the author "A PEN BY Chris Bridges". The editor is set to "JS" mode. The code defines two functions: `showMeLet` and `showMeVar`. The `showMeLet` function uses a `for` loop with `let` variables, demonstrating that `let` variables are only visible within their block of code. The `showMeVar` function uses a `for` loop with `var` variables, demonstrating that `var` variables are visible throughout the entire function scope.

```
1 function showMeLet () {  
2  
3   for ( let letExample = 0; letExample < 5; letExample++ ) {  
4     //letExample is only visible in here (and in the for()  
     parentheses)  
5     //and there is a separate letExample variable for each  
     iteration of the loop  
6     console.log(letExample)  
7   }  
8  
9   //letExample is *not* visible out here  
10  console.log(letExample)  
11  return '^^^ Cannot access let outside the block ^^^'  
12 }  
13  
14 function showMeVar () {  
15  
16   for ( var varExample = 0; varExample < 5; varExample++ ) {  
17     //varExample is visible to the whole function  
18     console.log(varExample)
```


Template Strings

- Can now insert variables directly into strings
- No need for cumbersome concatenation

```
let activity1 = 'jump rope'
```

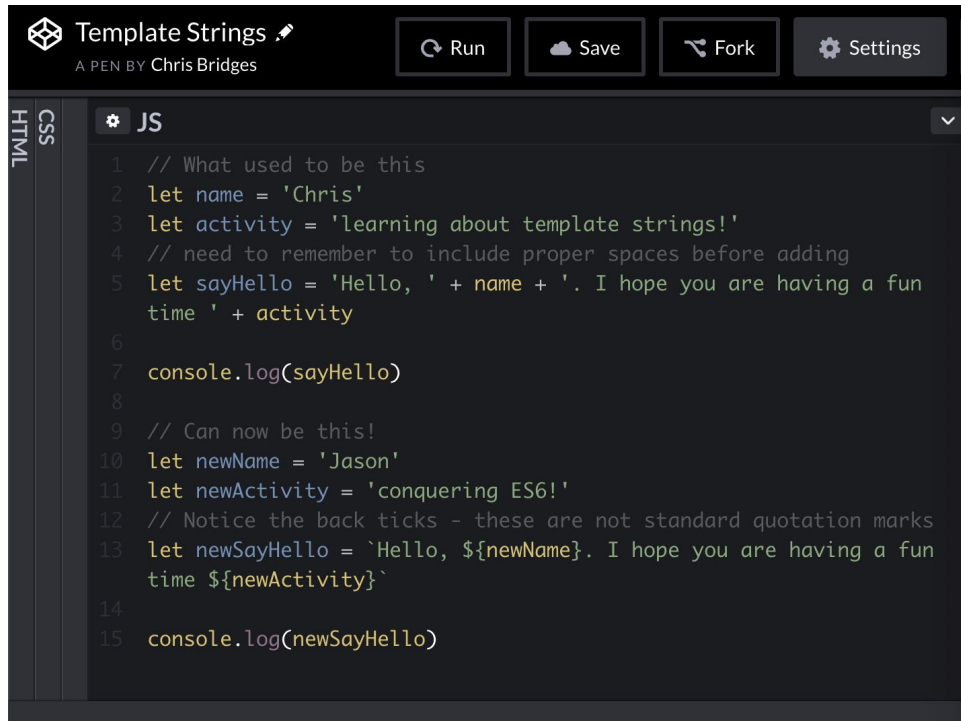
```
let activity 2 = 'eat ice cream'
```

```
let activity3 = 'go swimming'
```

```
let myDay = `Today, I will ${activity1},  
${activity2}, and ${activity3}`
```

```
// 'Today, I will jump rope, eat ice cream, and  
go swimmng'
```

Template Strings Examples



The screenshot shows a code editor interface with a dark theme. At the top, there's a header bar with the title "Template Strings" and a sub-header "A PEN BY Chris Bridges". To the right of the header are four buttons: "Run", "Save", "Fork", and "Settings". Below the header, there's a sidebar on the left with tabs for "HTML", "CSS", and "JS". The "JS" tab is selected. The main area displays JavaScript code with line numbers from 1 to 15. The code demonstrates the use of template strings, comparing an older method with a newer one using backticks and interpolation.

```
1 // What used to be this
2 let name = 'Chris'
3 let activity = 'learning about template strings!'
4 // need to remember to include proper spaces before adding
5 let sayHello = 'Hello, ' + name + '. I hope you are having a fun
  time ' + activity
6
7 console.log(sayHello)
8
9 // Can now be this!
10 let newName = 'Jason'
11 let newActivity = 'conquering ES6!'
12 // Notice the back ticks - these are not standard quotation marks
13 let newSayHello = `Hello, ${newName}. I hope you are having a fun
  time ${newActivity}`
14
15 console.log(newSayHello)
```

What is a function in Javascript

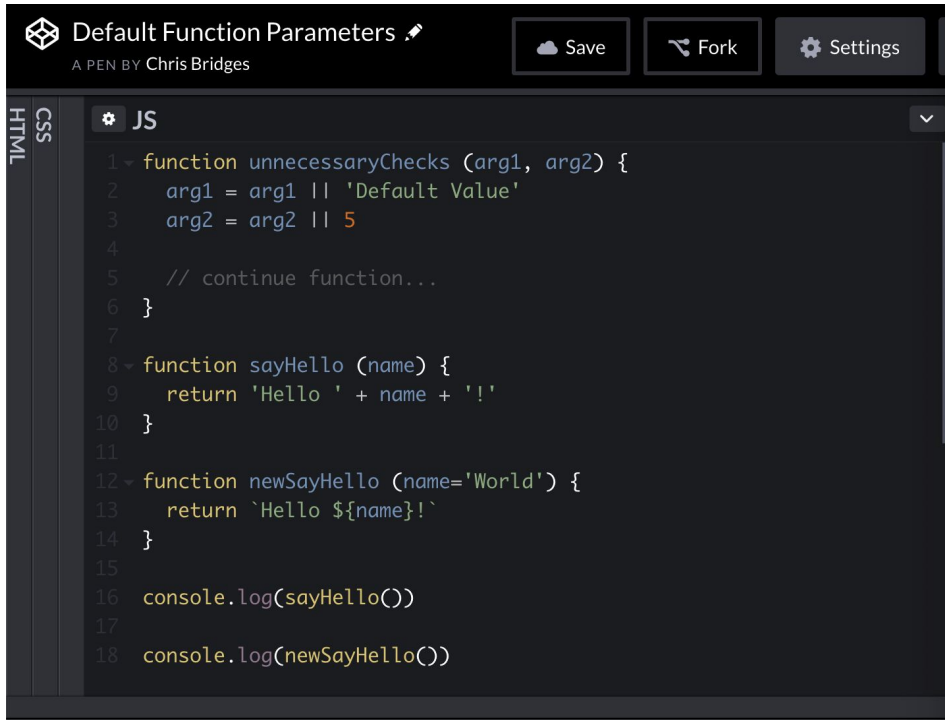
```
function logMe (name) {  
    console.log(name)  
}
```

Default Function Parameters

- Allows us to run functions with preset values if arguments are not provided
- No longer have to write explicit checks within function

```
function myDay (activity='ride my bike') {  
  
    return `Today, I will ${activity}`  
  
}  
  
myDay()  
  
// 'Today, I will ride my bike'
```

Function Params Examples



The screenshot shows a code editor interface with a dark theme. At the top, the title bar reads "Default Function Parameters" with a small icon on the left and "A PEN BY Chris Bridges" on the right. To the right of the title are three buttons: "Save", "Fork", and "Settings". On the left side, there is a vertical sidebar with tabs for "HTML", "CSS", and "JS", with "JS" being the active tab. The main editor area displays the following JavaScript code:

```
1 function unnecessaryChecks (arg1, arg2) {  
2   arg1 = arg1 || 'Default Value'  
3   arg2 = arg2 || 5  
4  
5   // continue function...  
6 }  
7  
8 function sayHello (name) {  
9   return 'Hello ' + name + '!'  
10 }  
11  
12 function newSayHello (name='World') {  
13   return `Hello ${name}!`  
14 }  
15  
16 console.log(sayHello())  
17  
18 console.log(newSayHello())
```

Arrow Functions

- Allow us to write shorter, more succinct functions
- Less boilerplate and implicit returns

```
let numbers = [6, 5, 4, 8, 2]
```

```
let lessThan6 = numbers.filter(function(value) {
```

```
    return value < 6
```

```
})
```

```
let newLessThan6 = numbers.filter( value => value < 6 )
```

```
// [5, 4, 2]
```

Arrow Functions Examples



The screenshot shows a code editor interface with a dark theme. The title bar reads 'Arrow Functions' with a small icon and 'A PEN BY Chris Bridges'. On the right side of the title bar are buttons for 'Run', 'Save', 'Fork', and 'Settings'. The left sidebar shows a file explorer with 'HTML', 'CSS', and 'JS' files. The 'JS' file is selected, and the editor displays the following JavaScript code:

```
1 let groceries = ['apples', 'bananas', 'oranges', 'chocolate']
2
3 let rememberToBuy = groceries.map(function (item) {
4   return 'Remember to buy: ' + item
5 })
6
7 // returns can be implicit if func is kept to one line
8 let newRememberToBuy = groceries.map(item => `Remember to buy:
9   ${item}`)
10
11 console.log(rememberToBuy)
12 console.log(newRememberToBuy)
13
14 // parens are needed around arguments if more than 1 is expected
15 const multipleArguments = (arg1, arg2, arg3) => {
16   let functionIsGreaterThanOneLine = true
17
18   if (functionIsGreaterThanOneLine) {
19     return 'return keyword is still necessary'
```

Spread Operator

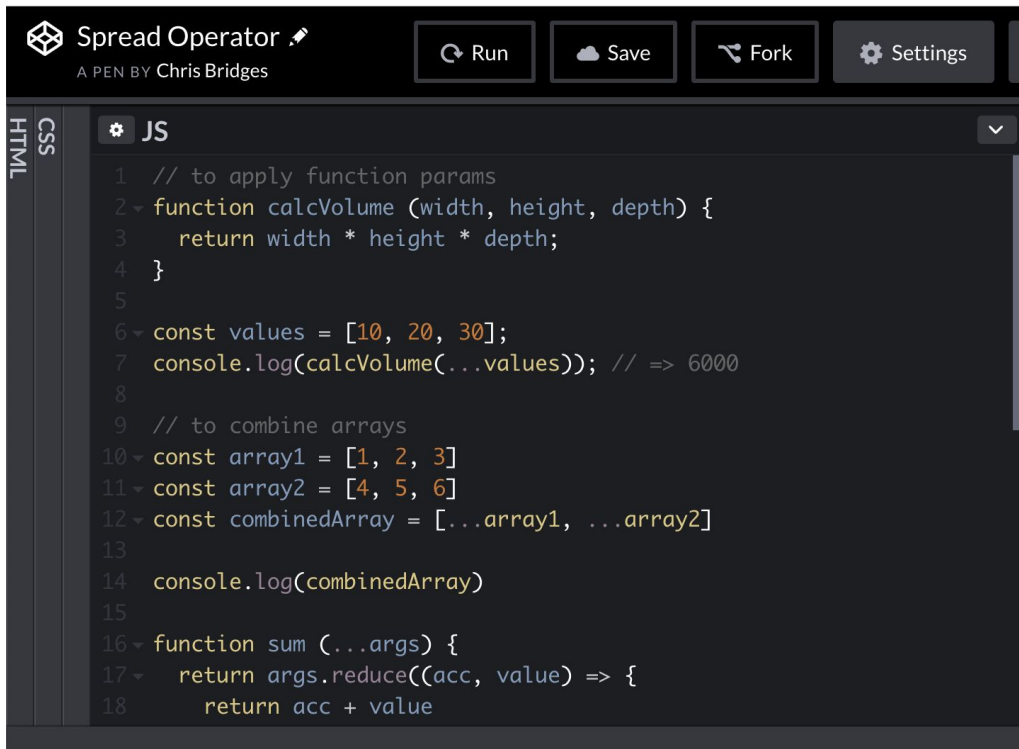
- A more compact way to apply parameters to functions
- Can also be used to combine arrays

```
function sum (...args) {  
  
    return args.reduce((acc, value) => {  
  
        return acc + value  
  
    })  
  
}
```

```
sum(1,2,4,10) // 17
```

```
sum(5, 5, 5, 5, 5, 5, 5, 5, 5) // 45
```


Spread Operator Examples



The screenshot shows a code editor interface with a dark theme. At the top, there's a header bar with the title "Spread Operator" and a pencil icon, followed by "A PEN BY Chris Bridges". To the right of the header are four buttons: "Run", "Save", "Fork", and "Settings". Below the header, on the left, is a sidebar with tabs for "HTML", "CSS", and "JS", with "JS" being the active tab. The main editor area displays the following JavaScript code:

```
1 // to apply function params
2 function calcVolume (width, height, depth) {
3   return width * height * depth;
4 }
5
6 const values = [10, 20, 30];
7 console.log(calcVolume(...values)); // => 6000
8
9 // to combine arrays
10 const array1 = [1, 2, 3]
11 const array2 = [4, 5, 6]
12 const combinedArray = [...array1, ...array2]
13
14 console.log(combinedArray)
15
16 function sum (...args) {
17   return args.reduce((acc, value) => {
18     return acc + value
```

Object.assign()

- Combine objects without the need to loop over it or use `.hasOwnProperty`
- Best way to clone and merge objects

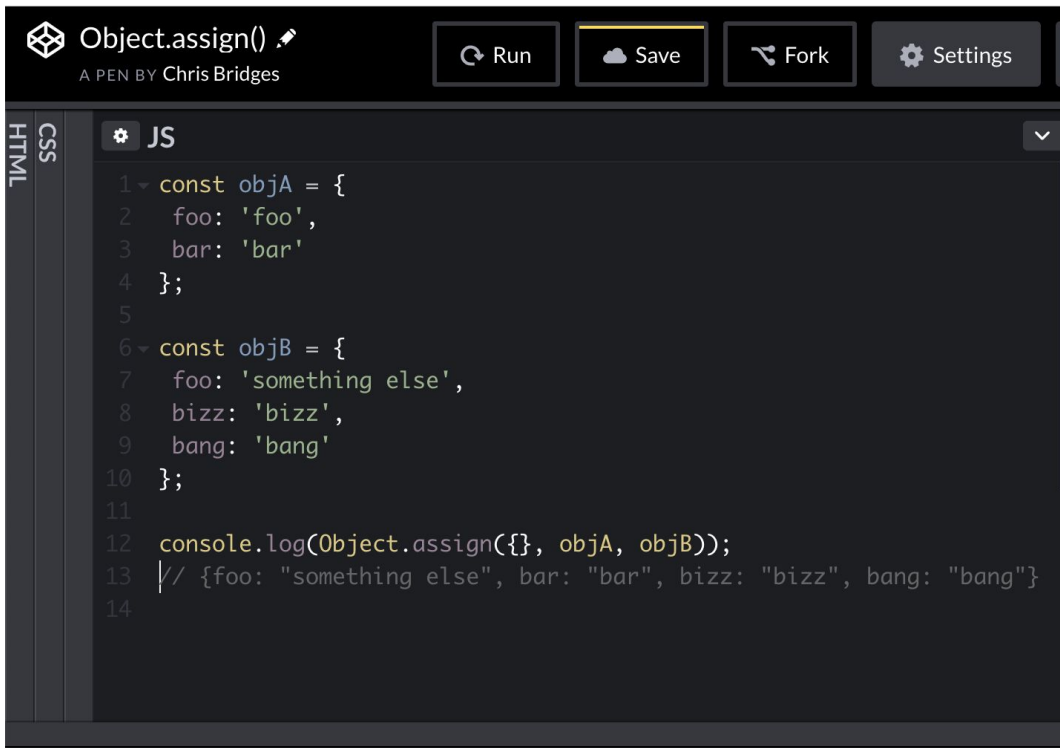
```
const obj1 = { a: 1, b: 2 }
```

```
const obj2 = { b: 'overwrite', c: 3 }
```

```
const combinedObj = Object.assign( {}, obj1,  
obj2 )
```

```
// { a: 1, b: 'overwrite', c: 3 }
```

Object.assign() Examples



The screenshot shows a code editor interface with a dark theme. At the top, there's a header bar with the title "Object.assign()" and a subtitle "A PEN BY Chris Bridges". To the right of the header are four buttons: "Run", "Save", "Fork", and "Settings". Below the header, there's a sidebar on the left with tabs for "HTML", "CSS", and "JS". The "JS" tab is selected. The main editor area displays the following JavaScript code:

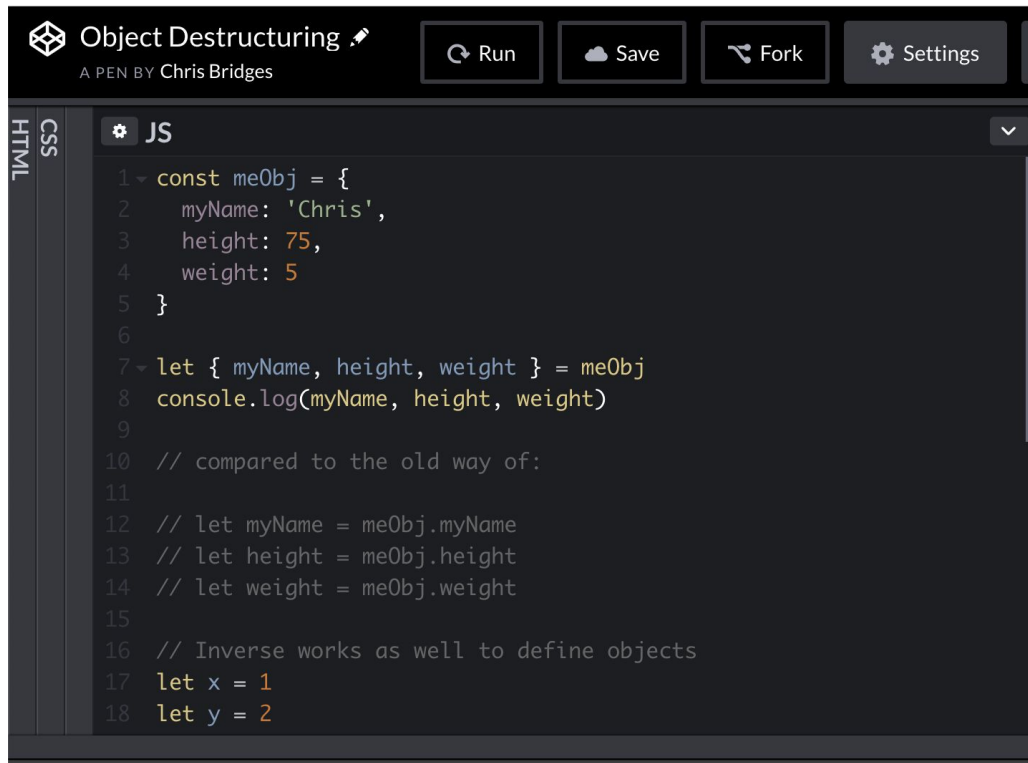
```
1 const objA = {  
2   foo: 'foo',  
3   bar: 'bar'  
4 };  
5  
6 const objB = {  
7   foo: 'something else',  
8   bizz: 'bizz',  
9   bang: 'bang'  
10};  
11  
12 console.log(Object.assign({}, objA, objB));  
13 // {foo: "something else", bar: "bar", bizz: "bizz", bang: "bang"}  
14
```

Object Destructuring

- No longer need to repeatedly define path if pulling values from same object
- Can also simplify defining object literals

```
const car = {  
  
    wheels: 4,  
  
    brakes: true  
  
}  
  
let { wheels, brakes } = car  
  
console.log(wheels, brakes)  
  
// 4, true
```

Object Destructuring Examples



The screenshot shows a code editor interface for a Pen titled "Object Destructuring" by Chris Bridges. The editor has tabs for HTML, CSS, and JS, with the JS tab selected. The code in the JS tab demonstrates object destructuring with a constant object, destructuring assignment, logging, and alternative assignment methods.

```
1 const meObj = {
2   myName: 'Chris',
3   height: 75,
4   weight: 5
5 }
6
7 let { myName, height, weight } = meObj
8 console.log(myName, height, weight)
9
10 // compared to the old way of:
11
12 // let myName = meObj.myName
13 // let height = meObj.height
14 // let weight = meObj.weight
15
16 // Inverse works as well to define objects
17 let x = 1
18 let y = 2
```

Summary

We covered a lot!

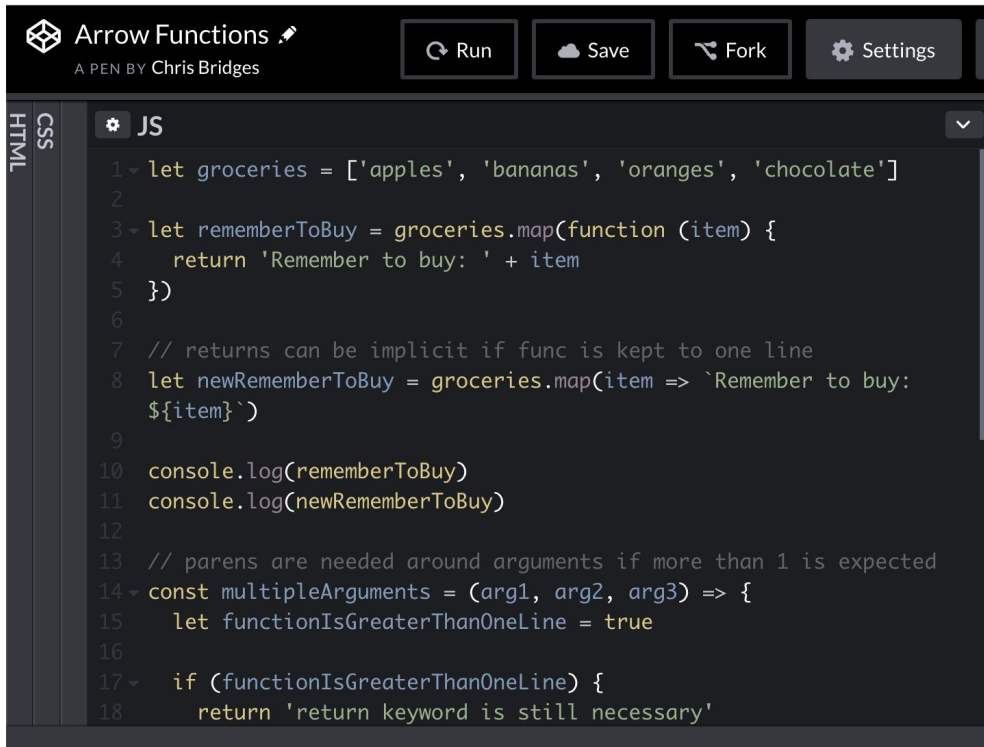
- `let / const`
- Template Strings
- Default Function Parameters
- Arrow Functions
- Spread Operator
- `Object.assign()`
- Object Destructuring

So Much More!

- Promises
- Classes
- Async / Await
- Multi-line strings
- Modules
- Import / Export
- And it's always growing!

Assignments for tonight

bit.ly/tf-es6-classroom



The screenshot shows a web-based code editor interface. At the top, the title is "Arrow Functions" with a small icon and the text "A PEN BY Chris Bridges". To the right of the title are four buttons: "Run", "Save", "Fork", and "Settings". On the left side, there is a vertical sidebar with tabs for "HTML", "CSS", and "JS". The "JS" tab is selected. The main area displays JavaScript code with line numbers from 1 to 18. The code demonstrates various uses of arrow functions, including basic function declarations, implicit return, and function arguments.

```
1 let groceries = ['apples', 'bananas', 'oranges', 'chocolate']
2
3 let rememberToBuy = groceries.map(function (item) {
4   return 'Remember to buy: ' + item
5 })
6
7 // returns can be implicit if func is kept to one line
8 let newRememberToBuy = groceries.map(item => `Remember to buy:
9   ${item}`)
10
11 console.log(rememberToBuy)
12 console.log(newRememberToBuy)
13
14 // parens are needed around arguments if more than 1 is expected
15 const multipleArguments = (arg1, arg2, arg3) => {
16   let functionIsGreaterThanOneLine = true
17
18   if (functionIsGreaterThanOneLine) {
19     return 'return keyword is still necessary'
```




jquery changing background color



jquery changing **css**

jquery changing **text**

jquery changing **background color**

jquery changing **class**



Ways to learn to code

[w3schools.com](https://www.w3schools.com)



meetup



[codecademy](https://www.codecademy.com)

[freeCodeCamp \(🔥\)](https://www.freecodecamp.org)

Two Week Free Trial

bit.ly/tf-free-ff-trail-replacewithrealone

- Free trial of Full Stack Flex online program
- Start with HTML, CSS & JS
- Personal Program Manager
- Unlimited Q&A Sessions
- Student Slack Community