October 18, 2021

Week 2

**Functions**

**Intro**

**Overview**

* Data Type Conversion
* String Methods
* The What & Why of Functions
* Anatomy of a Function
* Calling a Function
* Return Values
* Function Scope

**Objectives**

* Student can manipulate data types in JavaScript.
* Student can manipulate strings using methods such as split and trim.
* Student can create functions that take in parameters and return values.
* Student can call functions, pass in parameters, and obtain values.
* Student can implement debugging strategies.
* Students understands the control flow of different functions.

**Data Type Conversion**

**Loosely Typed**

JavaScript is loosely typed, which means that we don’t have to tell our code what data type a variable should be.

In JavaScript, we can change a variable from a string to a number and our code doesn’t care.

We can also convert the value of a variable into another data type.

**Converting to a Boolean**

Calling the Boolean function will convert a string or number into true or false.

The number 0 and an empty string '' are the only values that will return false.

**const** myVar = 5

**const** myVarTwo = Boolean(5)

console.log(myVarTwo) *// true*

**const** someVar = ''

**const** someVarTwo = Boolean(someVar)

console.log(someVarTwo) *// false*

**Converting to a Number**

Using the Number function will convert a string or boolean to a number if possible.

true will turn into the number 1 and false will turn into the number 0.

We can use a + as shorthand for Number.

We are just doing JS operations.

**const** bool = **true**

**const** boolTwo = Number(**true**)

console.log(boolTwo) *// 1*

There’s another method in JavaScript called parseInt that is a little more forgiving than Number.

It will search for a number in a string and return that number.

**const** string = '42ft'

**const** withNumber = Number(string)

**const** withParseInt = parseInt(string)

console.log(withNumber) *// NaN*

console.log(withParseInt) *// 42*

**Converting to a String**

We can also convert booleans and numbers into strings. To do this we can use the String function.

**const** evenNumber = 22

**const** stringNumber = String(evenNumber)

console.log(stringNumber) *// '22'*

JavaScript also allows you to concatenate numbers and booleans onto strings, which converts them.

If you just add an empty string to a number or boolean, it will convert to a string.

**const** myBool = **false**

**const** phrase = 'That is ' + myBool (const, the variable is assigned to the phrase equals the string plus the variable ‘myBool’

console.log(phrase) *// 'That is false'*

**const** oddNubmer = 3 (const variable is assigned the value 3)

**const** stringThree = 3 + '' (const string 3 variable is assigned the value 3 plus a string)

console.log(stringThree) *// '3'*

**String Methods**

There are lots of string methods in JavaScript. We will be covering some of the most common methods today.

We’ll talk about methods to use for checking if certain conditions are true or false in regards to a specified string.

We’ll also cover some methods that manipulate strings.

**Checking Conditions**

.includes() checks to see if a specified string is contained in another and returns true or false.

.startsWith() checks to see if a string starts with a certain character and returns true or false.

.endsWith() checks to see if a string ends with a certain character and returns true or false.

**const** myString = 'devmountain'

console.log(myString.includes('ou')) *// true*

console.log(myString.startsWith('v')) *// false*

console.log(myString.endsWith('n')) *// true*

example:

let string = (‘1, 2, 3, $, );

if (string.includes($#%){

console.log(“Algright!)

} else {

Console.log( Nope, try again

Console.log(string.endsWith(2)); {will return a true statement}

**Manipulating Strings**

We can’t actually directly manipulate strings. So running a method like .toUpperCase won’t affect the original string.

If we want to change the value of the original variable, we’ll need to reassign it. Or we can assign the value to a new variable.

**let** lowerString = 'lowercase letters'

lowerString.toUpperCase() <<<not doing a reassignment yet

console.log(lowerString) *// 'lowercase letters'*

lowerString = lowerString.toUpperCase()<<<<<new value

console.log(lowerString) *// 'LOWERCASE LETTERS'*

**let** upperString = lowerString.toUpperCase()<< let is on here to reassign

console.log(upperString) *// 'LOWERCASE LETTERS'*

**Trimming Whitespace**

.trim() takes any whitespace off the beginning and end of a string.

**const** whitespace = ' code '

**const** newWhitespace = whitespace.tri

console.log(newWhitespace) *// 'code'*

**Changing Casing**

.toUpperCase() and .toLowerCase() do what they sound like! They’ll make all the letters in a string upper or lower case.

**const** crazyString = 'tHiS is A StrINg'

**const** upperCrazy = crazyString.toUpperCase()

**const** lowerCrazy =crazyString.toLowerCase()

console.log(upperCrazy) *// 'THIS IS A STRING'*

console.log(lowerCrazy) *// 'this is a string'*

**Replacing Characters**

.replace() will replace the first instance of a string with a new one.

.replaceAll() will replace every instance of a string with a new one.

**const** shortStory = 'Once upon a time, there was a dragon. He was awesome. The end.'

**const** noFirstE = shortStory.replace('e', 'x')

console.log(noFirstE) *// Oncx upon a time, there was a dragon. He was awesome. The end.*

**const** noLetterE = shortStory.replaceAll('e', 'x')

console.log(noLetterE) *// Oncx upon a timx, thxrx was a dragon. Hx was awxsomx. Thx xnd.*

**Splitting and Joining Strings**

.split() will split a string into multiple strings in an array according to the character we send in.

.join() will join an array of strings together into one string

**const** statement = 'It is your birthday.'

**const** splitStr = statement.split(' ')

console.log(splitStr) *// [ 'It', 'is', 'your', 'birthday.' ]*

**const** splitOnLetter = statment.split('t')

console.log(splitOnLetter) *// [ 'I', ' is your bir', 'hday.' ]*

Passing an empty string to .split() will create a new string for every character including spaces

**const** statement = 'JS rocks'

**const** splitStr = statement.split('')

console.log(splitStr) *// ['J', 'S', ' ', 'r', 'o', 'c', 'k', 's']*

Joining with no argument will automatically put commas into the new string.

We can pass in an empty string to omit the commas.

Passing in a space or any character will insert that character between each entry in the array.

**const** arrayOfWords = ['North', 'South', 'East', 'West']

**const** joinWithCommas = arrayOfWords.join()

console.log(joinWithCommas) *// 'North,South,East,West'*

**const** joinWithNoCommas = arrayOfWords.join('')

console.log(joinWithNoCommas) *// 'NorthSouthEastWest'*

**const** joinWithSpaces = arrayOfWords.join(' ')

console.log(joinWithSpaces) *// 'North South East West'*

**const** joinWithPlus = arrayOfWords.join(' + ')

console.log(joinWithPlus) *// 'North + South + East + West'*

**The What & Why of Functions**

**Functions Explained**

A function is a block of code that can be reused over and over again without you having to rewrite your code. This can save you a lot of time and energy!

Before we create a function, let’s look at an example of some code we might want to run.

Let’s say a student takes a test worth 50 points. This student scores 44. Now, we want to work out their percentage, but first, we want our students to receive a 5 point curve. To do this, we could write:

**let** hermioneScore = 44

hermioneScore += 5 *// 49*

hermioneScore \*= 2 *// 98*

This is not horrible, it is only 3 lines of code. But what happens if this is a school-wide test and you need to grade 1000 tests?

This is where functions come in!

Look at this:

**function** calculateScore(studentScore) {

studentScore += 5

studentScore \*= 2

**return** studentScore

}

You will notice the body of our function looks almost identical to our last block of code, however, it is wrapped in some curly braces and has some additional info at the top. Don’t worry, we will break this down soon. However, with this function, we can now call one line of code for each student we want to be scored. Ex:

**let** hermioneScore = calculateScore(44)

**let** harryScore = calculateScore(41)

**Anatomy of a Function**

**The Parts**

**function** doSomething() {

...

}

All functions have these 4 parts:

* function The function key word. - Indicates that the following block of code is a function.
* doSomething The name of the function. - You can name a function whatever you want. Function names should be lowerCamelCase.
* () Function parameters go inside parentheses. - This function has no parameters. We will explain parameters momentarily.
* { } Opening and closing curly braces, which represent the body of a function. - Inside of the body is where you will put your code.

Let’s look at another example:

**function** doSomethingElse(withThis) {

console.log(withThis)

}

In this above example, we have the function declaration function, the name doSomethingElse, the parameters (just one in this case (withThis), and the body { console.log(withThis) }.

**function** doSomethingElse(withThis) {

console.log(withThis)

}

So what is happening in this above example? We have a function called doSomethingElse that, when it runs, will require 1 argument (piece of info) in order for it to run properly. The body of the function, in this case, will then console.log that piece of information.

Let’s take one more look at the example we started with.

**function** calculateScore(studentScore) {

studentScore += 5

studentScore \*= 2

**return** studentScore

}

In this example, we have a function called calculateScore that requires one piece of information, a student’s score. From there, the body of our function will add 5 points to the score, then it will multiply the score by 2, and return it to the location where the function was called (we will go into returning values and calling functions shortly).

**Calling a Function**

**You need to call a function**

It is important to know, that creating a function does not make it run. It only runs at the time in which it is called. To call a function, simply type the function name, followed by (). If the function has any parameters, you need to pass those values into the (). Ex.

**function** logMyName(name) {

console.log(name)

}

logMyName('Cameron') *// Output: Cameron*

One more example…

**function** sumNumbers(num1, num2) {

console.log(num1 + num2)

}

sumNumbers(3, 9) *// Output: 12*

**JavaScript has it’s dangers!**

Let’s say we have a function which takes in 2 parameters (num1, num2), and in the body of the function we subtract num2 from num1. If we call this function and pass in 7 for num1, and ‘Hello’ for num2, our app will crash!

**Calling a Function multiple times**

The great thing about functions is you can call them as many times as you want. If the function takes arguments, you can pass in different values each time. For example:

**function** sumNumbers(num1, num2) {

console.log(num1 + num2)

}

sumNumbers(3, 9) *// Output: 12*

sumNumbers(12, 33) *// Output: 45*

sumNumbers(-15, 15) *// Output: 0*

**Return Values**

**Return Required?**

Do all functions have to have a return value? Absolutely not. Some functions do something, some functions do something and give a value back to the call location. Ex.

**function** calculateTotal(item1, item2) {

console.log(item1 + item2)

}

calculateTotal(6.59, 9.99) *// Logs 16.58*

In this example, this function simply logs the total of item1 and item2.

**let** purse = 20

**function** calculateTotal(item1, item2) {

**return** item1 + item2

}

purse -= calculateTotal(6.59, 9.99) *// updates purse to equal 20 - 16.58*

In this example, this function returns 16.58 to the call site, which then makes that line of code equivalent to saying purse -= 16.58.

**Scope**

**What is scope?**

Scope is a variables accessibility. Prior to today, most of what we have dealt with has been on the outer scope, however, with functions (as well as if-statements and for loops), an inner scope is created. Let’s look at some examples of this.

**let** age = 21

**function** logDetails() {

**let** name = 'Tyler'

console.log(`My name is *${*name*}* and I am *${*age*}*.`)

}

This works just fine. By nature, we always have access to our current scope and outer scope. In this case, our console.log has access to name, because it is in the current scope, and it has access to age, because it is in the outer scope.

**let** age = 21

**function** logDetails() {

**let** name = 'Tyler'

}

console.log(`My name is *${*name*}* and I am *${*age*}*.`)

This does not work. At the point, our console.log has access to age because it is on the same level, however, name is now in the inner scope of our function (one level lower).

**The End**

© 2021 Devmountain