Learn to Code

in JavaScript

with Mike Witt

Learn to Code in JavaScript

Introduction

Learn to Code in JavaScript

Course Overview

Course Overview

- Running a JavaScript program.
- HelloWorld
- Primitive data types
- Operators
- Strings
- Dates
- Arrays
- Conditionals
- Looping
- Objects
- Functions
- Classes and Modules
- Putting it all together!

Section 2

The Basics

Learn to Code in JavaScript

Visual Studio Code, Hello World, and Live Server

JavaScript

- Developed by Netscape in 1995 to allow more dynamic websites.
  - Today all browsers support a JavaScript engine.
- Initially embedded code snippets in HTML files to give a more dynamic feel to the pages.
- Jquery added a more structured approach to using JavaScript to manipulate the Document Object Model (DOM).
- JavaScript started emerging in the 2010’s including: Knockout, Ember, Vue, Angular, and React.
- JavaScript’s natural environment is the browser, but it is moving outward with web servers like NodeJS to handle server side events.

Definitions

- JavaScript
- Visual Studio Code (or VS Code)
- index.html
  - DOCTYPE, html, body, h1, a, div, h2, script, button
- Live Server extension

Downloads

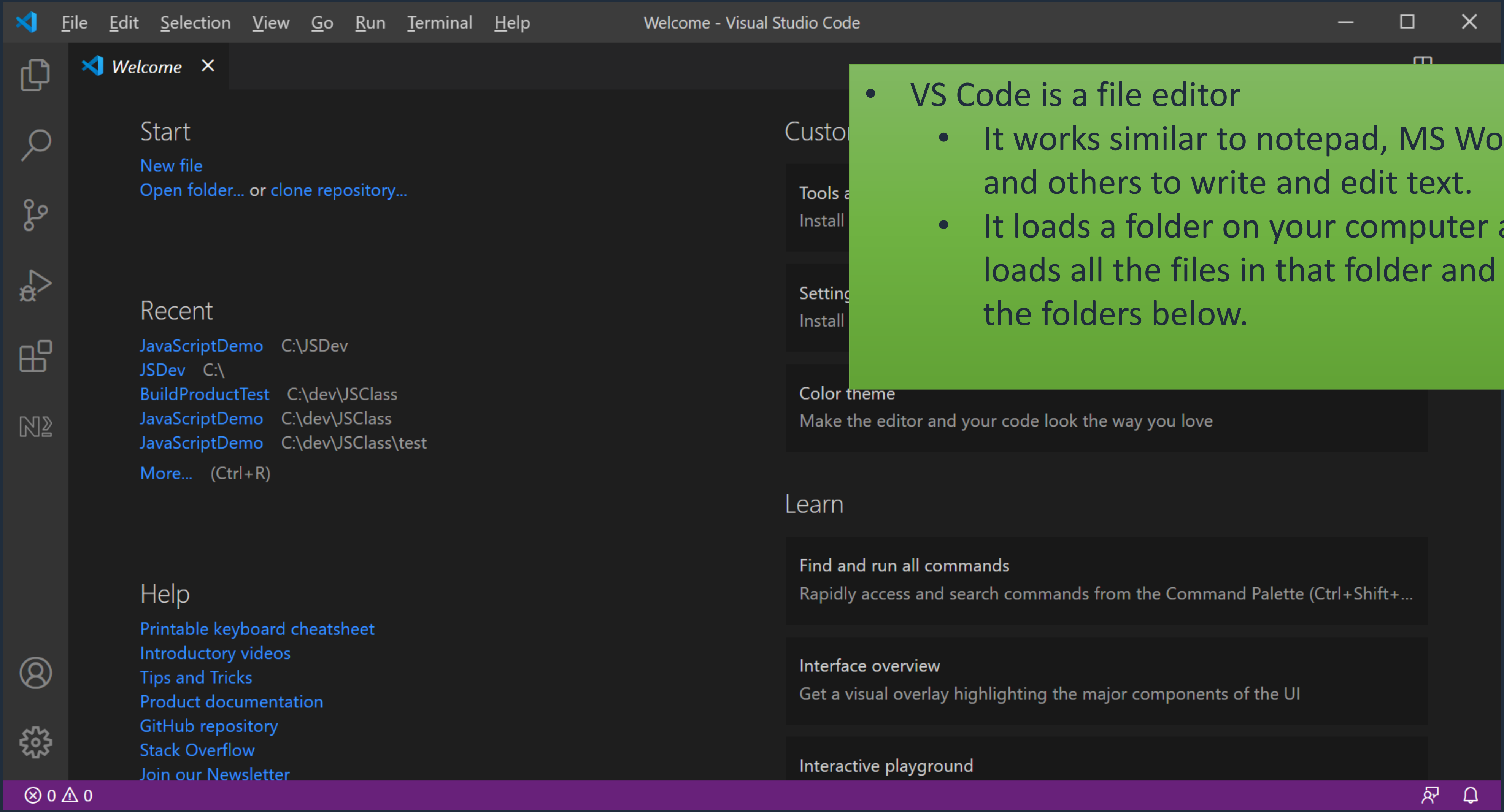
- Visual Studio Code

  https://code.visualstudio.com/?wt.mc_id=vscom_downloads

- Chrome

  https://www.google.com/chrome/

VS Code Overview



Start

New file

Open folder... or clone repository...

Recent

JavaScriptDemo C:\JSDev
JSDev C:\
BuildProductTest C:\dev\JSClass
JavaScriptDemo C:\dev\JSClass
JavaScriptDemo C:\dev\JSClass\test
More... (Ctrl+R)

Help

Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Join our Newsletter

Custo

Tools a
Install

Setting
Install

Color theme
Make the editor and your code look the way you love

Learn

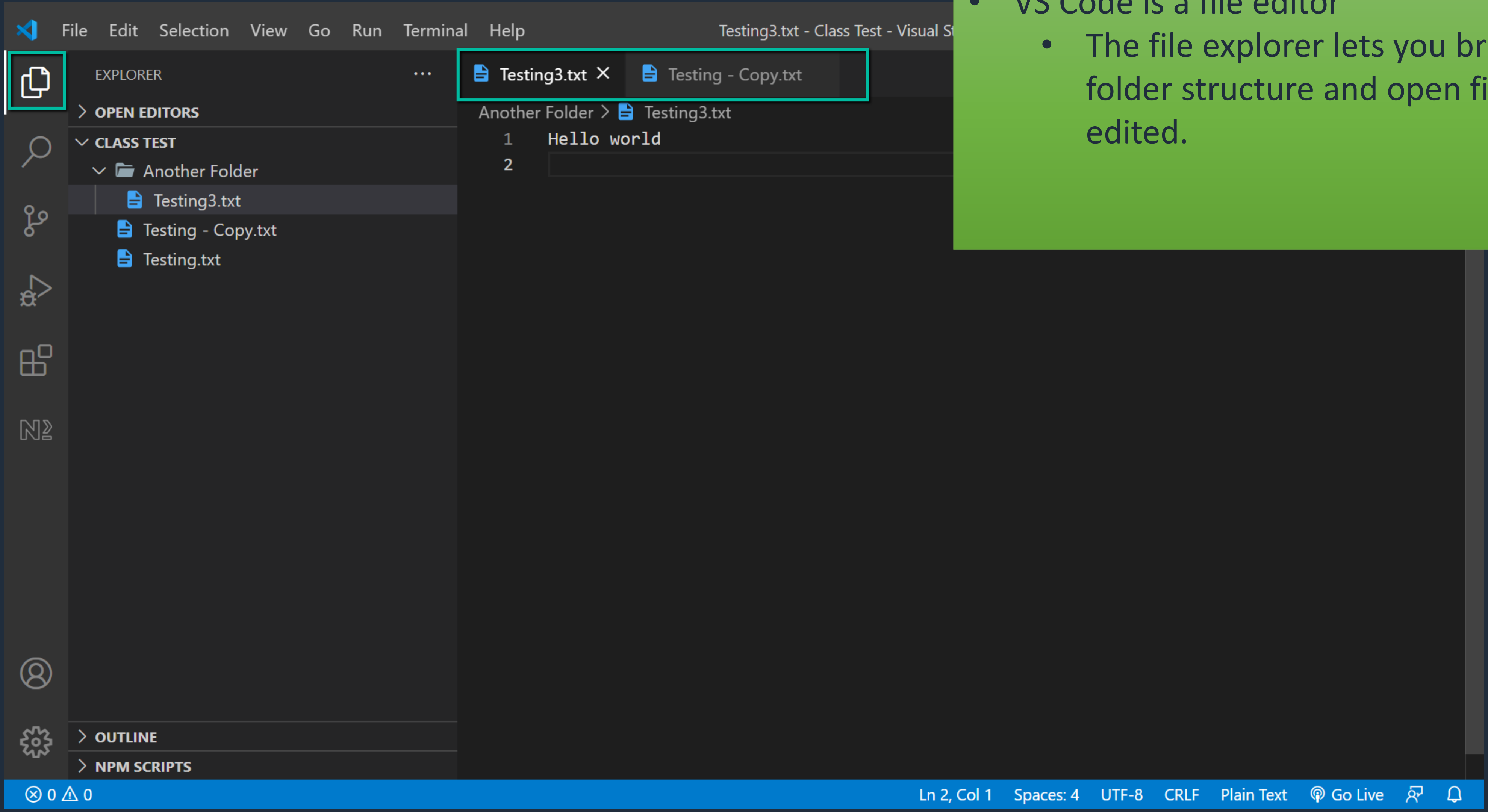
Find and run all commands
Rapidly access and search commands from the Command Palette (Ctrl+Shift+...

Interface overview
Get a visual overlay highlighting the major components of the UI

Interactive playground

- VS Code is a file editor
  - It works similar to notepad, MS Word, and others to write and edit text.
  - It loads a folder on your computer and loads all the files in that folder and all the folders below.

VS Code Overview

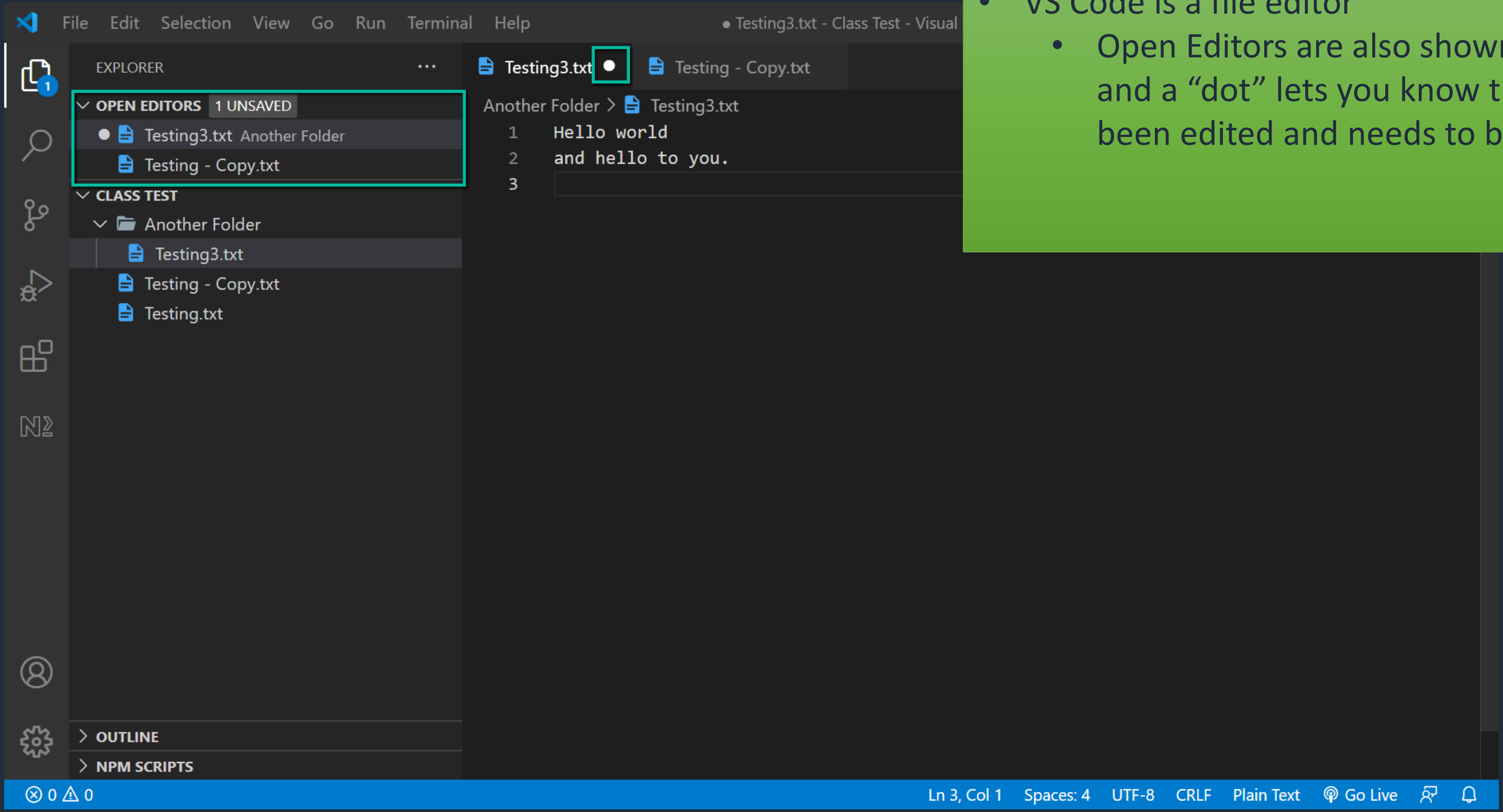


```
Hello world

```

- VS Code is a file editor
  - The file explorer lets you browse your folder structure and open files to be edited.

VS Code Overview

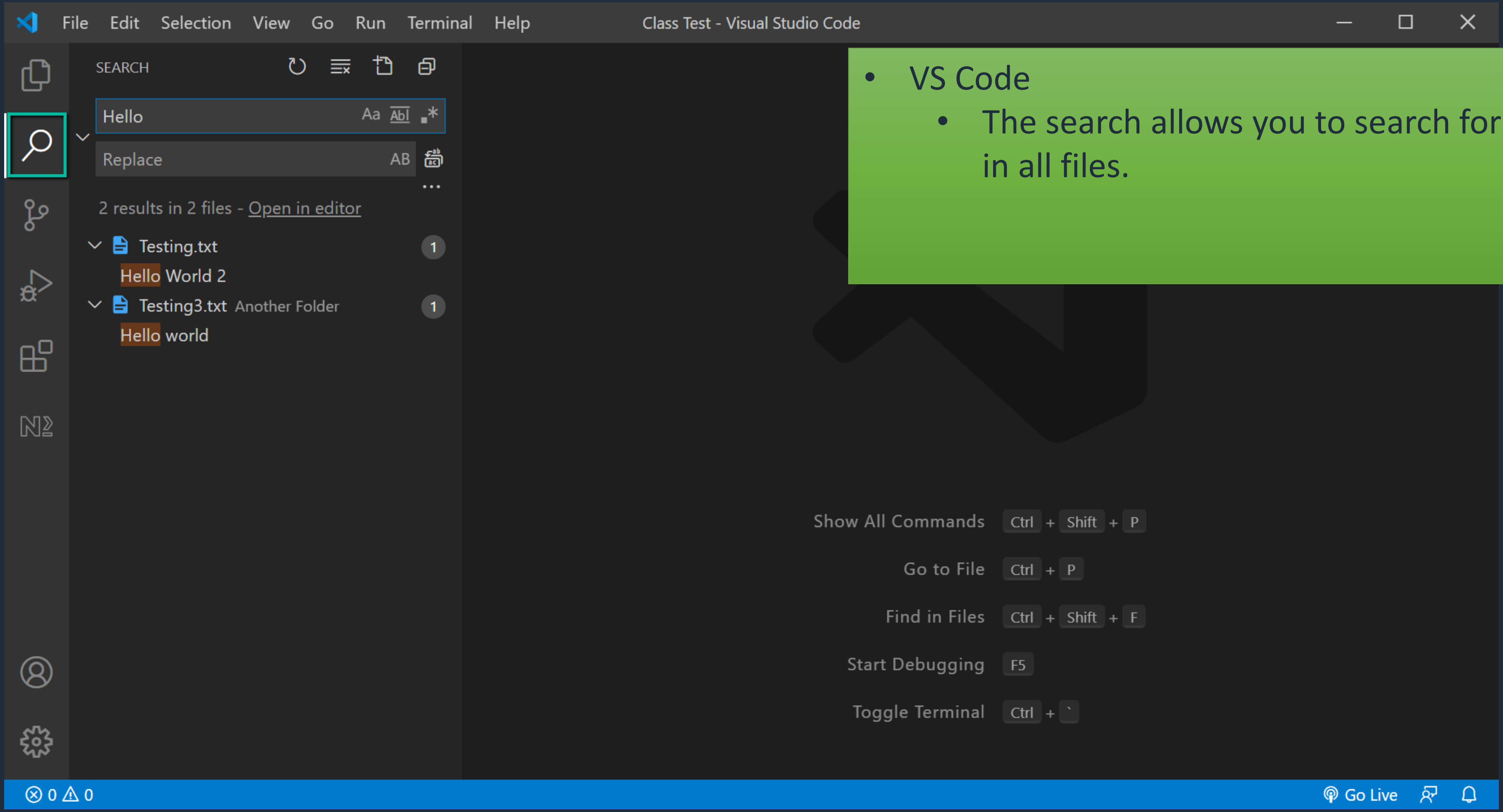


```
Hello world
and hello to you.

```

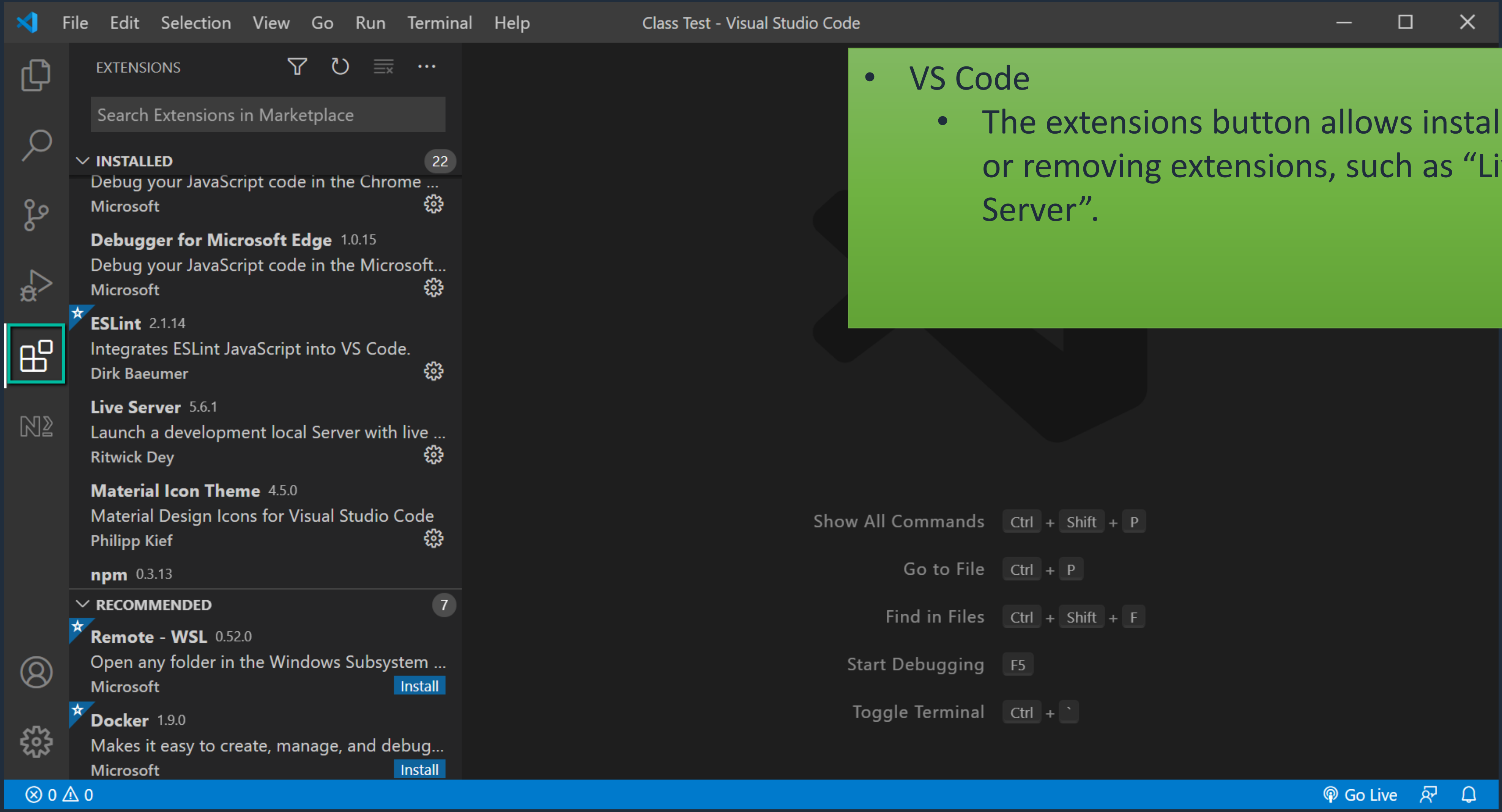
- VS Code is a file editor
  - Open Editors are also shown on the left and a “dot” lets you know this file has been edited and needs to be saved.

VS Code Overview



- VS Code
  - The search allows you to search for text in all files.

VS Code Overview



Debug your JavaScript code in the Chrome ...
Microsoft

Debugger for Microsoft Edge 1.0.15
Debug your JavaScript code in the Microsoft...
Microsoft

ESLint 2.1.14
Integrates ESLint JavaScript into VS Code.
Dirk Baeumer

Live Server 5.6.1
Launch a development local Server with live ...
Ritwick Dey

Material Icon Theme 4.5.0
Material Design Icons for Visual Studio Code
Philipp Kief

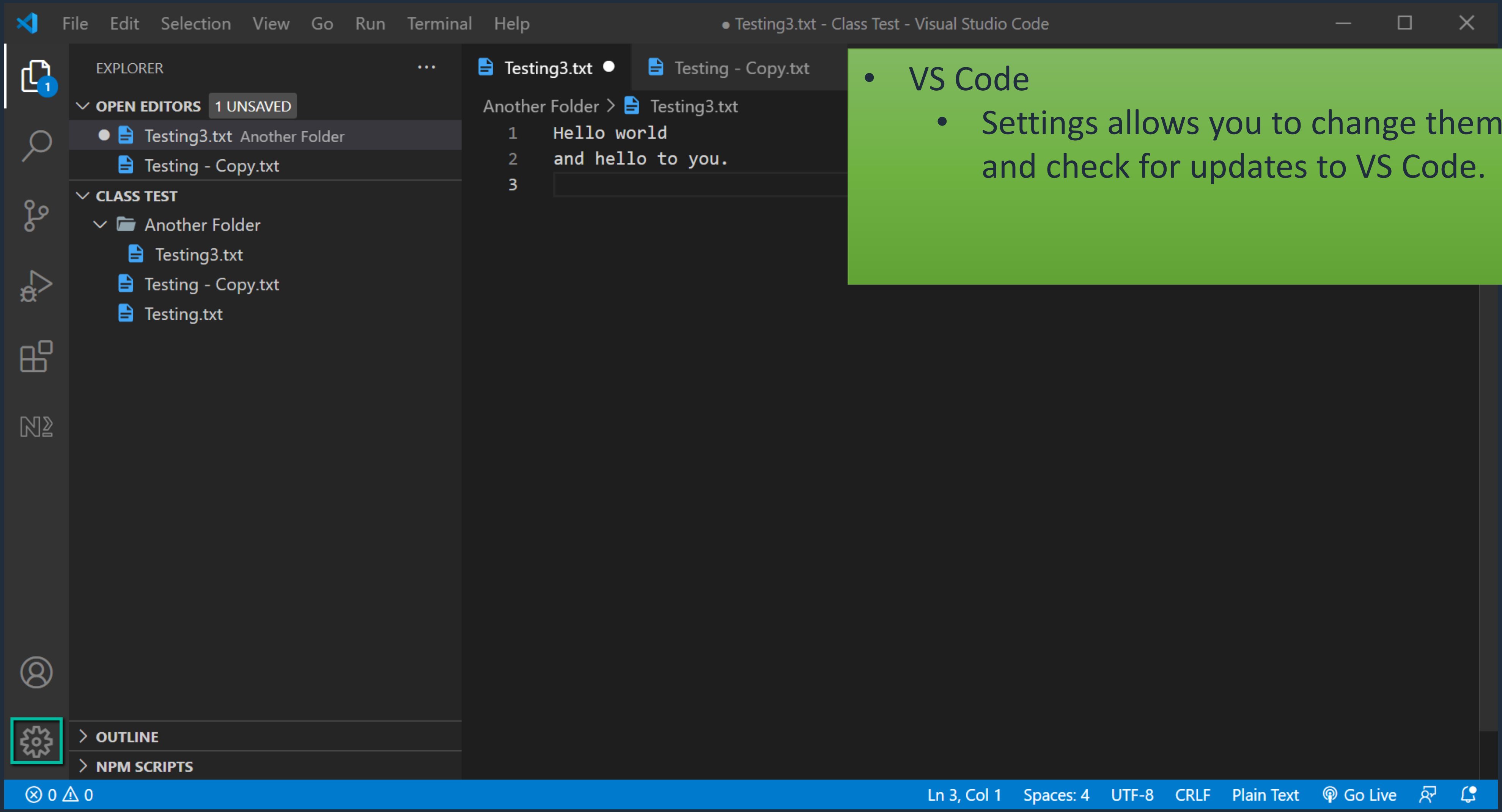
npm 0.3.13

Remote - WSL 0.52.0
Open any folder in the Windows Subsystem ...
Microsoft Install

Docker 1.9.0
Makes it easy to create, manage, and debug...
Microsoft Install

- VS Code
  - The extensions button allows installing or removing extensions, such as “Live Server”.

VS Code Overview



```
Hello world
and hello to you.

```

- VS Code
  - Settings allows you to change themes and check for updates to VS Code.

Hello World

- Display “Hello World”
- Do some variations with other console/prompt/confirm/alert calls.

Simple Input/Output

- console.log()
- console.clear()
- let x = prompt(“Enter a value”);
- alert(“Hello World”);
- let answer = confirm('Is this fun?');

Short HTML Overview

- HTML has matching elements:
  - <html></html>
  - <div></div>
- Elements tell the browser to do/show something:
  - <h1>This is a big header</h1>
  - <b>This will be bolded</b>
- Elements can have attributes:
  - <a href=“https://google.com”>Go to google</a>
- Elements can have style:
  - <div style="color: red; font-size: 2rem;">Hello again</div>

Format of a JavaScript Function

- function – keyword to define a JavaScript function.
- { to begin and } to end. This is referred to as a code block.
- Statements ending with a “;”.

Resources

- Mozilla - https://developer.mozilla.org/en-US/docs/Web/JavaScript
- W3Schools Tutorial - https://www.w3schools.com/js/default.asp
- W3Schools Reference - https://www.w3schools.com/jsref/default.asp

Learn to Code in JavaScript

Primitive Data Types

Primitive Data Types

- undefined – Simply a type that is not defined.
- boolean – Can be true or false.
- number – Holds a numeric value.
- string – A set of 0 or more characters.
- BigInt – Represents integers with arbitrary precision.
- Symbol – Produces a unique value.

We will be focused on undefined, boolean, number, and string in this course.

Literal Data vs Variables

- A literal value can be used only once.
- Examples:
  - Undefined: undefined
  - Boolean: true
  - Number: 1, 0, 22.567, 2.2567E1
  - String: “Hello World”, ‘Goodbye World’, “12.22”, “true”

Literal Data vs Variables

- A variable stores a value and can be used again.
- For primitive data, variables are immutable.
  - This means the value cannot be changed.
- Variables are stored in computer memory as 1 or more bytes where a byte has 8 bits (0s or 1s).
- Variable Scope – A variable is defined in a scope. Avoid global scope!
  - Keywords:
    - var – Declares in global scope, do not use!
    - let – used to declare a reusable variable.
    - const – used to declare a variable that cannot be assigned a new value.
- Dynamic Variables – A variable can store different types:

```
let x = 3;
x = “Hello”;
x = true;
```

Scope & Strict Mode

- ‘use strict’
  - Prevents use of undeclared variables.
- Scope
  - Boundaries using { and } where variables are known and defined.

Learn to Code in JavaScript

Operators

Operators

- An operator “operates” on something.
  - Generally an operator will operate on either 1 thing (unary) or 2 things (binary).
  - An operator will have precedence – more on this coming up.
  - Many operators can work on literals or variables, but some need a variable.
- Types of operators:
  - Arithmetic Operators – Give a numeric response
  - Relational Operators – Give a boolean response
  - Logical Operators – Give a boolean response
  - Assignment Operators – Give a response based on what is assigned

Arithmetic Operators

Name	Symbol	Description	Example
Addition	+	Adds 2 numbers	x + 3
Subtraction	-	Subtracts 1 number from another	x – 3
Multiplication	*	Multiplies 2 numbers together	x * 3
Division	/	Divides 1 number into another	x / 3
Remainder	%	Finds the remainder	x % 3
Exponentiation	**	Raised a number to a power	x ** 3
Unary Plus	+	Creates a positive number	+x
Unary Minus	-	Creates a negative number	-x
Auto Increment	++	Prefixed or Postfixed add 1 to variable.	++x or x++
Auto Decrement	--	Prefixed or Postfixed subtract 1.	--x or x--

Relational Operators

Name	Symbol	Description	Example
Less Than	<	Is value less than?	x < 3
Greater Than	>	Is value greater than?	x > 3
Less than or equal	<=	Is value less or equal?	x <= 3
Greater than or equal	>=	Is value greater or equal?	x >= 3
Equal	==	Is value equal?	x == 3
Not Equal	!=	Is value not equal?	x != 3
Identity Equal	===	Is value equal and of same type?	x === 3
Identity Not Equal	!==	Is value not equal or of different type?	x !== 3

Logical Operators

Name	Symbol	Description	Example
Logical And	&&	True if both values are true	x && y
Logical Or	||	True if one value is true	x || y
Logical Not	!	Switches true to false and false to true	!x

Assignment Operators

Name	Symbol	Description	Example
Assignment	=	Assigns value	let x = 5;
Plus equals	+=	Add and assign	x += 5;
Minus equals	-=	Subtract and assign	x -= 5;
Times equals	*=	Multiply and assign	x *= 5;
Divide equals	/=	Divide and assign	x /= 5;
Remainder equals	%=	Remainder and assign	x %= 5;

Operator Precedence

- Some operators have priority over others. Example:

  2 + 3 * 5 evaluates to 17 not 25 because * has higher priority than +

- Parenthesis can always be used to force a priority. Example

  (2 + 3) * 5 does evaluate to 25.

- JavaScript precedence order is a long list, so see:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

Learn to Code in JavaScript

Strings

Strings

- Strings contain a list of UTF-16 characters.
- Strings support a variety of methods that will operate on the string and produce a new string or give information about a string.
- Some methods are static. These are invoked on the String keyword:
  - String.fromCharCode(65, 66, 67));
- Most useful methods are instance methods. These are invoked on a string:
  - “Hello World”. indexOf("World"));
    - Evaluates to 6.
- There is one useful instance property: length.
  - “Hello World”.length;
    - Evaluates to 11
- The string literal (“Hello World”) could be a variable instead, like:
  - myString.length;
- String Templates: `Hello ${name}`

Strings – Selected Methods

Name	Description	Example

concat	Concatenate a list of strings to the target string.	const str = "Hello";
		const str2 = "World";
		console.log(“The string is: " + str.concat(" ", str2);

startsWith/ endsWith	Returns true if the target string starts/ends with the argument.	const str = "Hello World"; console.log("Does string end with World? ", str.endsWith("World"));

includes	Returns true if the string is found starting at position or 0 if not specified.	const str = "Hello World";
		console.log("Does string include Hello World? ", str.includes("Hello World"));

indexOf/ lastIndexOf	Returns the index/lastIndex of a substring located within the string with an optional start position.	const str = "Hello World"; console.log("What is the index of World? ", str.indexOf("World"));

padStart/ padEnd	Returns the string padded to the specified length at the start.	const str = "Hello World";
		console.log(`Here it is: ${str.padStart(25,'*')}`);

Strings – Selected Methods

Name	Description	Example

repeat	Repeats the string the specified number of times.	const str = "Hello World";
		console.log(`Here it is: ${str.repeat(5)}`);

replace/
replaceAll
	Searches for string and replaces 1st/all occurrences.	const str = "Hello World";
		console.log(`After replacing: ${str.replace('o', 'O')}`);

slice	Slices a substring from original string.	const str = "Hello World";
		console.log(`The slice: ${str.slice(3)}`);

split	Splits a string on a separator.	const str = "H.e.l.l.o. .W.o.r.l.d."; console.log(`Get an array of length 5: ${str.split('.')}`);

substring	Pulls a substring from original string.	const str = "Hello World";
		console.log(`Get: ${str.substring(1, 5)}`);

toString	Converts to a string.	123.toString();

toLowerCase/
toUpperCase
	Converts string to all lower/upper case characters.	const str = "Hello World";
		console.log(`Upper: ${str.toUpperCase()}`);

trimStart/
trimEnd/trim
	Trims whitespace from start/end/both of string.	const str = " Hello World ";
		console.log(`Trimmed string: |${str.trim()}|`);

Learn to Code in JavaScript

Dates

Dates

- Javascript date: Stored as milliseconds since January 1, 1970 UTC.
- JavaScript dates are generally not preferred. Most use a library like moment.js. Why?
  - Dates can be a number, an object, or a string.
- Two common ways to create a date:
  - let myDate = new Date(“2021-01-13”);
  - let theDate = new Date(2021, 5, 12, 15, 30, 0, 0);
  - let myDate = Date.now();
- Once you have a Date, there are lot of instance methods.

Dates – Selected Methods

Usage	Description
myDate.getDate()	Gets the day of the month, 1 – 31.
myDate.getFullYear()	Gets the 4 digit year.
myDate.getHours()	Gets the hour, 0 – 23.
myDate.getMilliseconds()	Gets the milliseconds 0 – 999.
myDate.getMinutes()	Gets the minutes, 0 – 59.
myDate.getMonth()	Gets the month, 0 – 11.
myDate.getSeconds()	Gets the seconds, 0 – 59.
UTC variants as well: myDate.getUTCHours()	Gets the hour, 0 – 23 in UTC timezone.
My “set” methods as well: myDate.setMonth(4)	Sets the month to the specified value.

Section 3

The Intermediate

Learn to Code in JavaScript

Arrays

Arrays

- An array is a collection of values. The values do NOT have to be the same type.
- An array literal is defined with square brackets with values separated by commas:

```
let myArray = [1, 3, 5, 7, 9];
let anotherArray = [5, ‘hello’, true];
```

- Array elements are then accessed by a 0 based index:

```
console.log(myArray[1]); // Will output 3
myArray[2] = 55; // Replace 5 with 55
```

- Array length property can give number of elements in array:

```
console.log(myArray.length); // Will output 5
```

- Array length cannot be changed after the array is initialized.

Arrays – Selected Methods

Usage	Description
myArray.push(25);	Pushes a new element onto the end of the array.
let value = myValue.pop();	Pops the last value off the array.
let idx = myArray.indexOf(25);	Returns the 0 based index where the first occurrence of the item appears.
let idx = myArray.lastIndexOf(25);	Returns the 0 based index where the last occurrence of the item appears.
let newArray = myArray.reverse();	Reverses an array.
let newArray = myArray.slice(2,4);	Returns a sub-array starting at index 2 and ending prior to index 4.
myArray.sort();	Will sort primitive items in place. More complicated items (objects) need a callback function.

Arrays – The Spread Operator

- Create an array from its operand:
  - […operand]
  - […array] – Clone an array
  - […array1, …array2, …array3] – Append arrays to make a new (cloned) array.
  - […anyThingYouCanIterate] – Gets turned into an array. For example:
  - […”MyString”] – Becomes: [“M”,”y”,”S”,”t”,”r”,”i”,”n”,”g”]
  - Example with Math

```
let a = [1,2,3,4,5];
let min = Math.min([…a]);
```

Arrays – Destructuring

- Unpacking array elements
  - let [x,y] = [1,2,3,4,5]; - Unpack so x == 1 and y == 2
  - let [x,,y] = [1,2,3,4,5]; - Unpack so x == 1 and y == 3 (Skipping 2)
  - let [x,...more] = [1,2,3,4,5]; - Unpack so x == 1 and more == [2,3,4,5]
  - const [x=3, y=4] = [1]; - Unpack with default value so y==4.

Learn to Code in JavaScript

Conditionals

Conditionals

- Make decisions about code paths
- Truthy/Falsy – Anything without a value in JavaScript is considered to be false: null, undefined, false, 0
- Do something if a condition is true/truthy.
  - If/Else statement
  - Switch statement
  - Ternary Operator (AKA Conditional Operator)

If/Else Examples

```
if (x > 0) {
  // Do something
}
```

```
if (x > 0 && x < 25) {
  // Do something
}
else {
  // Do something else
}
```

```
if (x < 5 || x > 20) {
  if (y < 10) {
    // Do something
  }
}
```

```
if (x > 2) {
  // Do something
}
else if (y > 5) {
  // Do something else
}
```

Switch Examples

```
switch (x) {
  case 1:
    // do something
    break;
  case 2:
    // do something
    break;
  default:
    // do a default thing
}
```

```
switch (x) {
  case 1:
    // do something
    break;
  case 2:
  case 3:
  case 4:
    // do something
    break;
  default:
    // do a default thing
}
```

Ternary/Conditional Operator Example

```
let y = x > 0 ? 1 : 2;
```

Learn to Code in JavaScript

Looping

Looping

- Make decisions about repeating code paths.
- Do something repeatedly while a condition is true.
    - For loop
    - While loop
    - Do/While loop

For Loop Examples

```
for (let i=0; i<10; i++) {
  // Do something
}
```

```
for (let i=0; i < 10; i++) {
  for (j = 0; j < 10; j++) {
    // Do something
  }
}
```

```
for (let i=0; i<10; i++) {
  // Do something
  if (xyz) continue;
  // Do something
}
```

```
for (let i=0; i<10; i++) {
  // Do something
  if (xyz) break;
  // Do something
}
```

```
let a = [2,4,6,8,10];
for (let v of a) {
  console.log(v);
}
```

While/Do-While Examples

```
while (x > 0) {
  x--;
}
```

```
do {
  x--;
while (x > 0);
```

Day 4

Functions and Objects

Learn to Code in JavaScript

Objects

Objects

- Grouping data together into one package.
- An object literal is defined with curly braces with values separated by commas:

  ```
  let myObject = {name: ‘Bob’, age: 12, signedUp: true};
  ```

- Object elements are then accessed by name:

  ```
  console.log(myObject.name, myObject.age);
  ```

- Objects can also be accessed using a string name with square brackets:

  ```
  console.log(myObject[“name”], myObject[“age”]);
  ```

- Null object

Objects

- Objects can store any type including other objects, arrays, and functions.

```
let myObject = {
        name: ‘Sue’,
        list: [22, 44, 66, 88],
        subObject: otherObject,
};
```

JSON

- JavaScript Object Notation
  - Serialize and Deserialize objects (Convert object to string and back again.)
  - JSON.parse
  - JSON.stringify

Objects – The Spread Operator

- let obj = {…myObj}; - Clone of myObj
- let obj = {…obj1, …obj2}; - Merge 2 objects

Objects – Destructuring

- const {id, name, age} = {name: 'Bill', id: 12, age: 55}; - Pulls obj apart
- const {id: myId, name: myName} = obj; - Renames the values
- const {id=12345, name='Sue'} = obj; - Default values
- const {id: myId=123, name: myName='Mary'} = obj; - Renames/Defaults

Learn to Code in JavaScript

Functions

Functions

- Functions allow complex systems to be decomposed into smaller parts.
- Functions are defined.
  - Functions receive parameters.
  - Functions can return a single value.
- Functions are called in order to invoke.
  - Parameters are sent into function.
  - A return value may be received.
- Functions can call functions.
- Functions can be stored in a variable.

Function Examples

```
function myFunction() {
  console.log(‘Here I am’);
}

myFunction();
```

```
function adder(a, b) {
  return a + b;
}

let total = adder(10, 20);
```

```
function adder(a, b, c=0) {
  return a + b + c;
}

let total = adder(10, 20);
```

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

f(10, 5);
```

Functions – The Rest Syntax

- Allows function to accept an unlimited number of parameters
- Example

```
function myFunction(…args) {
  for (let a of args) console.log(a);
}
```

Arrow Functions – Often used for callbacks

- Callback – Function is used as a parameter to another function and that function makes the call.

Arrow Functions – Often used for callbacks

- Callback – Function is used as a parameter to another function and that function makes the call.

```
function runit(f) {
  f();
}


runit(() => {
  console.log(‘Hello World’);
});
```

Built in array functions:

```
let a = [1,2,3,4,5,6,7,8];
let b = a.filter(x => x < 5);
```

Function Recursion

- Recursion occurs when a function calls itself.

```
function goForever() {
  goForever();
}

goForever();
```

Section 4

The Advanced

Learn to Code in JavaScript

Classes

Classes

- Classes let you provide a template for your objects.
- Class must be declared before it is used (it is not hoisted).
- Classes can have:
  - Super-classes (where class is inherited from another class)
  - Constructor
  - Methods (known as prototype methods)
  - Static Methods and Properties
  - Instance Properties

Learn to Code in JavaScript

Modules

Modules

- Divides JavaScript application into pieces
- Depends on:
    - Export – To make contents of module available.
    - Import – To pull in contents from another module.

Learn to Code in JavaScript

Putting it all together

Putting it all together

- Build out our app
  - Shape drawing app
  - Draw shapes with low resolution (using “*” as dots).
  - Draw into a matrix using logical operators to overwrite what’s there.
  - Display the matrix in an HTML TextArea element.

Section 5

Wrapping It Up

Learn to Code in JavaScript

You Made It!

Learn to Code in JavaScript

Thank You!

Learn to Code in JavaScript

Bonus Lecture