Notes on LinkedIn Course "JavaScript Essential Training (2021)" Part 1

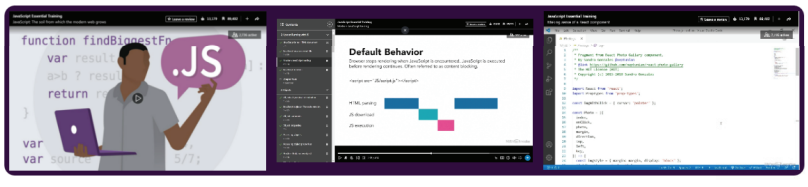
November 26, 2022 / By Eric Hepperle / **COURSE NOTES** / Coding Video , Essential Training , JavaScript , LinkedIn Learning

JavaScript Essential Training (2021)

COURSE INFO

JavaScript Essential Training	
URL:	https://www.linkedin.com/learning/javascript-essential-training
Channel:	LinkedIn Learning
Instructor:	Morten Rand-Hendriksen
Release Date:	2021-01-20
Date Started:	2022-11-21
Date Completed:	–
File Name:	EHD_VIDNOT_20221121_LL_JSEss_2021__WIP__01.md

COURSE PREVIEW



COURSE DETAILS

Course Length:	5h 29m
Level:	Beginner
Tools & Software:	Visual Studio Code
Skills Covered:	JavaScript

DESCRIPTION:

JavaScript is a scripting language of the web. As the web evolves from a static to a dynamic environment, technology focus is shifting from static markup and styling—frequently handled by content management systems or automated scripts—to dynamic interfaces and advanced interaction. Once seen as optional, JavaScript is now becoming an integral part of the web, infusing every layer with its script.

Through practical examples and mini-projects, this course helps you build your understanding of JavaScript piece by piece, from core principles like variables, data types, conditionals, and functions through advanced topics including loops, and DOM scripting. Along the way, instructor Morten Rand-Hendriksen provides challenges that allow you to put your new skills to the test.

Recent Comments

Eric Hepperle on **Contact Form 7 Broken on WordPress – GoDaddy, MS Exchange Email Spam Issues Likely Cause**

Eric Hepperle on **Freedom Signpost**

Gregory V on **Contact Form 7 Broken on WordPress – GoDaddy, MS Exchange Email Spam Issues Likely Cause**

Recent Posts

Notes on LinkedIn Course "JavaScript Essential Training (2021)" Part 1
November 26, 2022

Underdog Devs DevOps Group: Week 1 Homework November 22, 2022

Project Underdog Cohort 4: Week 1 Report November 20, 2022

December 2022

M	T	W	T	F	S	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

DESCRIPTION:

JavaScript is a scripting language of the web. As the web evolves from a static to a dynamic environment, technology focus is shifting from static markup and styling—frequently handled by content management systems or automated scripts—to dynamic interfaces and advanced interaction. Once seen as optional, JavaScript is now becoming an integral part of the web, infusing every layer with its script.

Through practical examples and mini-projects, this course helps you build your understanding of JavaScript piece by piece, from core principles like variables, data types, conditionals, and functions through advanced topics including loops, and DOM scripting. Along the way, instructor Morten Rand-Hendriksen provides challenges that allow you to put your new skills to the test.

LEARNING OBJECTIVES

- Modern JavaScript (ES6/2015)

INSTRUCTOR

Morten Rand-Hendriksen

Senior Staff Instructor, Speaker, Web Designer, and Software Developer

TOC

TABLE OF CONTENTS GOES HERE

VIDEOS

0. Introduction

0.0 JavaScript: The soil from which the modern web grows

- The modern web runs on JavaScript
- Foundational elements of JS
- JAMstack
- HTML is language of web content, CSS is language of web style, JS is language the binds it all together
- This course deep understanding of modern JS
- JS primary lang for everything we do on web

0.1 How to use the exercise files

- Ex files avail from GitHub
- Folder with e suffix represent the end state
- PDF attached
- Link to doc or code refs in many exercise files

1. JavaScript: A Brief Introduction

1.0 JavaScript: First contact

- A new era of web dev
- JS on browser, server, localhost

APPLY PRACTICAL SKILS

- Modern JS & tooling
- JSX & Typescript
- Frameworks
- We are going to start with complex and then break it down
- Typical react component is long and complex

JSX

- A syntax extension of JS created for the React JS framework
- If I'm going to learn React should I just learn JSX?
- NO! JSX is an extension, so you still need basics
- If you undestand the basics you will be able to quickly pickup JSX and other JS flavors

COMPONENTS

- Objects
- Methods
- Functions
- Template Literals
- Arrays
- At end of course we will return to this component and you will be able to make sense of it

1.1 Navigating the JS landscape

COMPONENTS

- Objects
- Methods
- Functions
- Template Literals
- Arrays
- At end of course we will return to this component and you will be able to make sense of it

1.1 Navigating the JS landscape

- Overwhelmed by that all the JS flavors
- Vanilla Javascript
- ES2015
- Babel
- React
- Vue.js
- ECMAScript
- CoffeScript
- JSX
- TypeScript
- Angular
- ES6
- WebPack
- Node.js
- Gulp
- JS is constantly evolving

MORTEN'S INCOMPLETE GUIDE TO NAVIGATING THE JS LANDSCAPE:

- Javascript: the core lang; sometimes referred to as vanilla JS
- This is what we will cover in this course
- ECMAScript: browser specification of the JS lang. not the lang itself, but official description of how the lang should be interpreted by browsers
- Cutting edge
- Babel.js: Use to convert modern JS into plain JS the browser can read
- JS is an opinionated coding lang
- TypeScript: variation, dialect, or flavor of JS introducing features like **strong typing**
- Abstracted versions of JS with additional features.
- .ts ext
- CoffeeScript: another dialect of JS
- React, Vue, Angular: JS frameworks allowing us to write JS-based font-end apps. Adds abstraction layer on top of JS
- Introduce new coding convention like JSX and reliance on tools like Beble, WebPack, and Node.js
-
- npm, WebPack, Gulp: Build tools and infrastructure to automate the process of optimizing human-readable JavaScript for the best browser performance
- Node.js: JS server runtime used to run JS everywhere; used to run npm, WebPack, Babel, and more on your computer

TAKEAWAY:

- Learn JS first then optimize and specialize your knowledge

1.2 Tools for working with JavaScript

- Modern browser: ideally all the browsers for testing (Chrome, Firefox)
- Code editor: VS Code is becoming industry standard. Highlight code as you work
- *Live server env: extension for VS Code or similar. Efficieny*
- Browser console: included with every browser
- Live Server (Ritwick Dey)

1.3 Linting and formatting

- Morten is dyslexic
- Code highlighter helps, but other extensions help
- ESLint: helps auto detect coding errors and can do basic cleanup auto
- Prettier: helps auto clean up your formatting
- Both require Node.js (install Node.js via download)

INSTALL NPM

- Navigate to Terminal install
- `npm install` : node package manager goes on the internet and pulls everything into VS code to get prettier and eslint to work
- With prettier you can set what rules to enforce

1.4 Get to know the browser console

- JS doesn't run in the code editor it runs in the browser

EX: 01_05 Script.js

- Click Go Live at bottom bar of VSCode
- Go to current folder 01_05

- Both require Node.js (install Node.js via download)

INSTALL NPM

- Navigate to Terminal install
- `npm install` : node package manager goes on the internet and pulls everything into VS code to get prettier and eslint to work
- With prettier you can set what rules to enforce

1.4 Get to know the browser console

- JS doesn't run in the code editor it runs in the browser

EX: 01_05 Script.js

- Click Go Live at bottom bar of VSCode
- Go to current folder 01_05
- Open console
- `window.document`
- `backpack` hit right key for autocomplete
- `backpack.toggleLid(true)`
- `backpack` : lid is still set to true

1.5 JavaScript language basics

EX: 01_06

- Write JS top to bottom
- Define functions top before you use them
- Code comments: green
- **JS Doc:** Verbose comment:/** [ENTER]:

```
/**
    * function updateBackpack()
    * Outputs HTML
    * @param {string} update
    */</pre >
```

- CTRL + /: quickly comment / uncomment code
- Whitespace is just for humans. #BESTPRACTICE: Use indentation to indicate visual hierarchy
- Change Tab indent size by clicking `Spaces:2` on VSCode bottom bar
- Semicolons: JS doesn't care if semicolons or not. If anyone tells you it is wrong to do one or the other, it's not true. This is purely developer preference.
- #BESTPRACTICE: Be consistent with double quotes and single quotes
- #BESTPRACTICE: Use tools like Prettier and ESLint to automatically enforce that consistency

1.6 Learning JavaScript backward

- When you learn JS for the first time you typically start with the basics.
- But, due to the popularity of JS frameworks like React an Vue, today the first intro many people have to JS is through **advanced objects and methods**.
- For this reason, we'll start with objects / methods, then Data Types and DOM, finally functions, methods, events at end.
- Learning JS requires understing basics and patterns, but also

  Finding ways to make them make sense to you
- With examples first, you will see the context and then ask why questions
- Open-ended practice assigments thorughout the course
- Refer to MDN Web Docs as part of your process
- Make documentation part of your learing journey
- Documentation-based process for robust learning

1.7 Chapter Quiz

What is an indicator of someone being a good JavaScript developer? They follow standards, invest in learning, use formatting and linting tools for consistency, and write accessible code.

🎞 BOOKMARK

2. Up and Running with JS

2.0 JavaScript in an HTML document

- Where does jS live? Where do you actually write the code?
- Inline (using `style` tag)

2. Up and Running with JS

2.0 JavaScript in an HTML document

- Where does jS live? Where do you actually write the code?
- Inline (using `style` tag)

EX: 02_01

- #BESTPRACTICE: Add script tag at end right before closing `body` tag
- Anything inside the script tag will automatically be rendered as javascript
- Can technically place script tag anywhere in the doc

Why is script tag placed at bottom?

- When browser encounters script tag all rendering stops
- But, this is an **antipattern**: we have more better modern ways of loading JS

2.1 JavaScript as an external file

- Writing inline jS is edge case: only applies to current doc and nowhere else
- #BESTPRACTICE: Put script in its own file and reference the stylesheet in whatever file you want to use it in

EX: 02_02

- index.html references script.js with the `src` attribute
- This error is noted in the course:

```
Uncaught TypeError: Cannot read property 'appendChild' of null at script.js:50
```

- #SOLVED: This is caused by the script being run in the head before the page has been rendered

2.2 Modern JavaScript loading

- Browser reads HTML top to bottom line by line and fetch and execute any elements it encounters

> When you open an HTML document in the browser, the browser will read that document line by line from the top down and fetch and execute any element it encounters as necessary.

02_03

- Typical header: references to scripts and external stylesheets
- As browser encouters calls, stop renering, goes and gets external files, executes what external files tell it, then continues rendering
- The error in the console is a result of the JS referenced and run in the browser before the element it is acting upon is rendered.

> It can't do what the JavaScript is trying to do because elements don't yet exist.

- The traditional solution: move script tag to end of doc so browser only encounters it when it has finished rendering the whole doc. Not really a solution though, actually a #HACK / #WORKAROUND
- #GOTCHA: some JS should run at the beginning or while the doc is being loaded
- New tools in JS to tightly control when and how JS is loaded: **async** and **defer** keywords

DEFAULT BEHAVIOR

- Browser stops rendering when JS is encountered. JS is executed before rendering continues. Often referred to a **content blocking** or **render blocking**.

- Blocks rendering of content of page and can cause page to load slower
- **Async**: tells browser to keep parsing HTML while JS is downloaded and only stop rendering once you have downloaded the JS

ASYNC

- Browser downloads JS in parallel while HTML renders. When JS is fully loaded, rendering stops while JS is executed.

DEFAULT BEHAVIOR

- Browser stops rendering when JS is encountered. JS is executed before rendering continues. Often referred to a **content blocking** or **render blocking**.

- Blocks rendering of content of page and can cause page to load slower
- **Async**: tells browser to keep parsing HTML while JS is downloaded and only stop rendering once you have downloaded the JS

ASYNC

- Browser downloads JS in parallel while HTML renders. When JS is fully loaded, rendering stops while JS is executed.

- Dramatically shortens time it takes for browser to execute everything and there is only a short render blocking issue
- **defer**:

DEFER

- Browser downloads JS in parallel while HTML renders, then defers execution of JS until HTML rendering is complete

- At end of script tag in index.html add the `defer` keyword

```
<script scr="script.js" defer></script></pre >
```

- We are deferring the execution of the script until everything esle has been rendered
- #BESTPRACTICE: async/defer should be the standard. Only use render blocking when you have a specific reason. Loading JS in the footer is now an **anti-pattern**.

> async/defer should be the standard. Only use render blocking when you have a specific reason. Loading JS in the footer is now an anti-pattern.

- From now on: load JS in the header then use async or defer to control when the js is loaded in the doc

2.3 JavaScript modules

- As you start working with JS you will notice files get large and hard to work with, requires scrolling up and down. #SOLUTION: JS modules
- JS modules: allow us to break pieces out of a JS file into separate files and then import back into the original file again

02_04 script.js

- at top of script.js

```
import backpack from "./backpack.js"</pre >
```

- constant called `backpack` is what is being imported from backpack.js
- at bottom of backpack.js

```
export default backpack</pre >
```

- export tells browser that the entity (const) can be used by any other file it is imported into
- To get this work in HTML, tell browser the files are modules (auto deferred)
- Modules are an advanced and new feature in modern JS. But, is standard practice with React and Vue

```
<!DOCTYPE html>
  <html lang="en">
    <head>
      <meta charset="UTF-8" />
      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
      <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
      <title>Module Demo</title>
      <script type="module" src="backpack.js"></script>
      <script type="module" src="script.js"></script>
    </head>
```

```
/>
        <title>Module Demo</title>
        <script type="module" src="backpack.js"></script>
        <script type="module" src="script.js"></script>
      </head>
      <body></body>
    </html></pre >
        #GOTCHA: Can't call backpack object in console, because it is only
      scoped to the file, not to the browser
```

- As of 2021, modules are cutting edge JS

2.4 Chapter Quiz

WHEN DOES THE BROWSER EXECUTE JAVASCRIPT?

By default: When the script is encountered. If the script is set to “async”, when the script is fully loaded. If the script is set to “defer”, when the entire HTML page is rendered.

WHAT IS THE CORRECT MARKUP FOR ADDING AN EXTERNAL JAVASCRIPT FILE TO AN HTML DOCUMENT?

While `<script src="javascript.js"></script>` is technically correct, it is recommended to always async or defer your script unless you have a specific reason for the script to cause render blocking.

WHAT HAPPENS WHEN YOU DEFER JAVASCRIPT?

The browser loads the JavaScript asynchronously when it is encountered, then waits until all HTML is rendered before executing the script.

JAVASCRIPT MODULES ARE HEAVILY USED IN FRAMEWORKS LIKE REACT AND VUE. WHAT IS THE ADVANTAGE OF USING MODULES?

Modules enable modularization of code where individual functions, components, data objects, and other parts can be separated into individual files.

3. Objects

3.0 Objects: A practical introduction

- JS is prototype based
- At core we are working with objects based on prototypes
- Quickest path to understand JS is understanding what objects are and how they work
- Object in JS represents object in real life

EX:

- Backpack is an object
- Pockets, straps, zippers are object properties
- **object properties:** Define the specifics of each unique object
- Each object is a unique instance of an object prototype.

> Each object is a unique instance of an object prototype.

- A thing is type of object because it shares common properties which define them as belonging to the backpack category
- The particular combination and config of these properties define each individual backpack.
- Changing the property values can create new backpacks
- Prototype based OOP allows us to categorize things by similarities
- Objects have features that allow us to change their properties (state)

EX: lid can open and close

- **method:** property-changing features inside objects
- methods act on current object only
- Objects can contain other objects
- Objects inside other objects are properties of those objects

3.1 JavaScript objects: The code version

- javascript object: collection of data and functionality stored as properties and methods that describe the object and what it can do
- To define object / create it: use variable to hold object
- Modern convention: use `const`

```
const backpack = {
      volume: 30,
```

- methods act on current object only
- Objects can contain other objects
- Objects inside other objects are properties of those objects

3.1 JavaScript objects: The code version

- javascript object: collection of data and functionality stored as properties and methods that describe the object and what it can do
- To define object / create it: use variable to hold object
- Modern convention: use `const`

```
const backpack = {
      volume: 30,
  }</pre >
```

- variables hold data
- curly brackets: say this is a js object
- properties: key-value pair separated by a colon. Can even be another object or array. can nest sub-objects. separate with comma. convention is put each on own like
- object can have as many properties as it needs
- methods: change properties of object. also name-value pairs, but in method the value is a function
- this: "this" keyword refers to the current object. "This object right here"

EX:

```
const backpack = {
    name: "Everyday Backpack",
    volume: 30,
    color: "grey",
    pocketNum: 15,
    strapLength: {
      left: 26,
      right: 26,
    },
    lidOpen: false,
    toggleLid: function (lidStatus) {
      this.lidOpen = lidStatus
    },
    newStrapLength: function (lengthLeft, lengthRight) {
      this.strapLength.left = lengthLeft
      this.strapLength.right = lengthRight
    },
  }</pre >
```

3.2 Object containers

- const: a variable / container

OBJECTS ARE TYPICALLY CONSTANTS

- We can change the properties of the object inside the container. We can't remove or replace the object from the container.

EX: 03_03

- Type `backpack` in console
- Type `backpack = 5`

Uncaught TypeError: Assignment to constant variable. at <anonymous>1:10

- can't change a constant to something entirely different (object to integer, for instance). Properties CAN be changed
- This way you don't accidentaly destroy your object while working with it

3.3 Object properties

- Defined with colon-separated key-value pair
- key: any string
- value: string in quotes, int, float, boolean, array, object
- prop names: letters, digits, $, _
- #ROT: use **camelCase** for property names

> **#ROT:** use camelCase for property names

3.4 Accessing objects

- To access ojbejcts in JS call it by name

EX: 03_05

3.4 Accessing objects

- To access ojbejcts in JS call it by name

EX: 03_05

- Type in console: `backpack`
- Click arrows on object to view properties

CONSOLE METHODS

- console.log(): tells browser to print the object in the console automatically
- console.log("The backpack object:", backpack) // comma tells browser to add a space

3.5 Accessing object properties

- Two ways of accessing properties: dot notation and bracket notation

EX: 03_06

- Looking at dot notation first.
- At bottom of script.js,

```
1.  console.log("The pocketNum value:", backpack.pocketNum)</pre >
```

- VSCode will automatically code hint the functions available to use
- properties of nested objects

```
1.  console.log("Strap length L:", backpack.strapLength.left)</pre >
```

- Bracket notation: useful if you need more control or have non-standard names

```
1.  console.log("The pocketNum value:", backpack["pocketNum"])</pre >
2.        EX: Pass property value as a variable (
```

variable variable

```
1.  )
```

```
1.  var query = "pocketNum"
2.    console.log("The pocketNum value:", backpack[query])</pre >
```

- **#GOTCHA:** Variable variables can't be done with dot notation
- 3rd use case for bracket notation: nothing prevents you or software from creating prop names that break the naming conventions. Bracket notation will allow you access non-standard names via bracket notation because property is enclosed in quotes.

3.6 Practice: Build a new object

- Pause video to play around with your new-found knowledge

EX: practice > 03_07

PRACTICE

First practice assignment. Take objects within reach and turn them into a JS object.

1. Give each object an identifiable name.
2. Create properties to describe the objects and set their values.
3. Find an object that has another object inside of it to create a nested object.

3.7 Object methods

EX: 03_08

- **methods:** functions inside methods
- Two syntaxes for methods: function expression and shorthand
- **function expression:** explicitly says "function"

toggleLid:

3.7 Object methods

EX: 03_08

- **methods:** functions inside methods
- Two syntaxes for methods: function expression and shorthand
- **function expression:** explicitly says “function”

○

toggleLid:

function (
lidStatus) { this.lidOpen = lidStatus }

- **Shorthand:** harder to read

○

toggleLid(lidStatus) { this.lidOpen =
lidStatus }

- #BESTPRACTICE: Convention is to use function expression for clarity

How Method works:

- **parameter:** a piece of data we can pass to the function.
  AKA arguments
- **function:** a program that does something / changes a
  falue
- make a function run with **function call**
- function call: typing the function name with empty parens or parens
  holding params
- in console,

○

backpack.lidOpen
// false backpack.toggleLid(
true)
// undefined backpack.lidOpen
// true

- The value only changes in the user’s browser at that moment

○

console.log(
“Left before:”,
backpack.strapLength.left)
// 26 backpack.newStrapLength(
10, 15)
console.log(
“Left after:”, backpack.strapLength.left)
// 10

3.8 Practice: Build a new method

EX: Practice > 03_09

- Create new methods in this object, one to change each property

1. Pass value to func inside the parens
2. Refer to current obj as “this”
3. Assign any value to any property

MY PRACTICE RESULTS

○

/** * Practice: Making methods * * – Create a method for each object
property. * – The method receives a value to match the property to be
changed. * – Create a simple function to replace the current property
value with the received value. * – Test the method by sending new values
and checking the properties in the console. */
/* Eric Hepperle 2022-11-23 */
const backpack = {
name:
“Everyday Backpack”,
volume:
30, color:
“grey”,
pocketNum:
15,
strapLength: {
left: 26,
right: 26,

changed. – Create a simple function to replace the current property value with the received value. * – Test the method by sending new values and checking the properties in the console. */
/* Eric Hepperle 2022-11-23 */
const backpack = {
name:
“Everyday Backpack”,
volume:
30, color:
“grey”,
pocketNum:
15,
strapLength: {
left: 26,
right: 26,
}, toggleLid:

function (
lidStatus) { this.lidOpen = lidStatus; },
newStrapLength:

function (
lengthLeft, lengthRight) { this.strapLength.left = lengthLeft;
this.strapLength.right = lengthRight; },
setName:

function (
newName) { this.name = newName },
setVolume:

function (
newVol) { this.volume = newVol },
setColor:

function (
newColor) { this.color = newColor
if (newColor ===
‘red’) {
console.log(
`JACKPOT!!!`) } },
setPocketNum:

function (
numPockets) { this.pocketNum = numPockets }, };

3.9 Classes: Object blueprints

- Classes relatively new to JS
- Classes reduce repetion by acting as templates for object types
- Any time you create a new instance of a class it automatically gets all the same properties and methods of the class

> Classes work as templates for an object type. And any time you create a new object based on a class, that object automatically gets all the properties and the methods from that class.

EX: 03_10 (significant changes to backback class)

TO CREATE A CLASS

- Start with class keyword followed by capitalized name
- convention shows we are looking at class as opposed to a regular object
- **Backpack.js**: new file that exports a Backpack class
- export default Backpack
- in script.js import Backpack at top
- in index.html import both JS files as modules (to have files depend on each other)
- Class is only used in the JS, not accessible from browser
- Two ways to declare a class: class declaration and class expression
- class declaration: class Name {}
- class expression: const Name = class {}
- The choice is user preference, they do the same thing
- #BESTPRACTICE: Current trend is use class expression
- constructor method: defines params for each prop in parens, then inside curly brackets defines all properties and sets values from params
- add class methods after constructor method

Backpack.js

○

/** * Creating classes: * * Class declaration: class Name {} * Class expression: const Name = class {} */

class
Backpack { constructor(// Define parameters
name, volume, color, pocketNum, strapLengthL, strapLengthR, lidOpen) {
// Define properties:
this.name = name
this.volume = volume
this.color = color
this.pocketNum = pocketNum
this.strapLength = {

○

/** * Creating classes: * * Class declaration: class Name {} * Class
expression: const Name = class {} */

class
Backpack { constructor(// Define parameters
name, volume, color, pocketNum, strapLengthL, strapLengthR, lidOpen) {
// Define properties:
this.name = name
this.volume = volume
this.color = color
this.pocketNum = pocketNum
this.strapLength = {
left: strapLengthL,
right: strapLengthR, }
this.lidOpen = lidOpen }
// Add methods like normal functions:
toggleLid(lidStatus) { this.lidOpen =
lidStatus } newStrapLength(lengthLeft, lengthRight) {
this.strapLength.left = lengthLeft
this.strapLength.right = lengthRight } }
export
default Backpack

HOW TO USE A CLASS

- Create a new variable like this,

script.js

○

/** * Create a Backpack object. */
import Backpack
from
"./Backpack.js"
const everydayPack =
new Backpack(
"Everyday Backpack",
30,
"grey",
15, 26,
26,
false)
console.log(
"The everydayPack object:", everydayPack)
// obj
console.log(
"The pocketNum value:",
everydayPack.pocketNum) // 15

- #GOTCHA: JavaScript modules must be run in server context or you will get CORS error like this,

> Access to script at 'file:///Z:/sb/junk.js' from origin 'null' has been blocked by CORS policy: Cross origin requests are only supported for protocol schemes: http, data, isolated-app, chrome-extension, chrome, https, chrome-untrusted.

- Can you use **Five Server** VSCode extension in 2022
- Finally, index.htm is the final piece,

index.html

○

<!DOCTYPE html>

< html
lang=
"en">

< head>

< meta
charset=
"UTF-8" />

< meta
http-equiv=
"X-UA-Compatible"
content=

“IE=edge” />

< meta
name=
“viewport”
content=
“width=device-width, initial-scale=1.0”
/>

< title> Classes
</ title>

< script
type=
“module”
src=
“Backpack.js”>

</ script>

< script
type=
“module”
src=
“script.js”>

</ script>

</ head>

< body>

</ body>

</ html>

- All imports need to happen at the top of the main js file
- Classes must be declared (via import) before they are used

3.10 Object constructors

EX: 03_11

- Shorter lesser advanced alternative to class: object constructor function
- function keyword instead of class
- no explicit “constructor” keywords
- methods live inside the constructor area and use “this”

script.js (*with object constructor*)

○

/* Object constructor example */

function
Backpack(

name, volume, color, pocketNum, strapLengthL, strapLengthR, lidOpen) { this.name = name
this.volume = volume
this.color = color
this.pocketNum = pocketNum
this.strapLength = {
left: strapLengthL,
right: strapLengthR, }
this.toggleLid =

function (
lidStatus) { this.lidOpen = lidStatus }
this.newStrapLength =

function (
lengthLeft, lengthRight) { this.strapLength.left = lengthLeft
this.strapLength.right = lengthRight } }
const everydayPack =
new Backpack(
“Everyday Backpack”,
30,
“grey”,
15, 26,
26,
false)
console.log(
“The everydayPack object:”, everydayPack)
console.log(
“The pocketNum value:”,
everydayPack.pocketNum)

- Differences between class and object constructor: we can extend classes, add new features, and class is now preferred
- #BESTPRACTICE: Use classes, avoid object constructors
- #GOTCHA: In old code and tutorials you will still see a lot of object constructors because that’s all JS developers used to have
- #ROT: Use a class unless you are required to use an object constructor

26,
false)
console.log(
“The everydayPack object:”, everydayPack)
console.log(
“The pocketNum value:”,
everydayPack.pocketNum)

- Differences between class and object constructor: we can extend classes, add new features, and class is now preferred
- #BESTPRACTICE: Use classes, avoid object constructors
- #GOTCHA: In old code and tutorials you will still see a lot of object constructors because that’s all JS developers used to have
- #ROT: Use a class unless you are required to use an object constructor function because classes give you more capability

3.11 Practice: Build a new object with a constructor

- Classes feature heavily in modern JS frameworks
- Getting comfortable with writing and using classes will be advantageous working with advanced JS and applications

EX: Practice > 03_12

PRACTICE

- Use Backpack class as reference.
- Create new classes based on one or more of the objects you created in the previous lessons.
- Create separate files for each class and import them as modules to have clear **separation of concerns**

Easiest approach:

- Start with a fully built object then migrate classes and properties over one-by-one

MY PRACTICE RESULTS:

Version 1

index.htm

○

<!DOCTYPE html>

< html
lang=
“en”>

< head>

< meta
charset=
“UTF-8” />

< meta
name=
“viewport”
content=
“width=device-width, initial-scale=1.0”
/>

< title> Practice: Making classes and objects
</ title>

< script
type=
“module”
src=
“Backpack.js”>

</ script>

< script
type=
“module”
src=
“Rag.js”>

</ script>

< script
type=
“module”
src=
“script.js”>

</ script>

</ head>

src=
“Rag.js”>

</ script>

< script
type=
“module“
src=
“script.js”>

</ script>

</ head>

< body>

</ body>

</ html>

Rag.js

○

/** * Rag.js * * Programmer: Eric Hepperle * Date: 2022-11-23 * *
Purpose: Class to represent rags for OOP in JavaScript */

class
Rag { constructor(name, color, height,
width) { this.name = name
this.color = color
this.height = height
this.width = width } setName(newName) {
this.name = newName } setColor(newColor)
{ this.color = newColor } logRagInfo() {
let styles = [
`font-weight: bold`,
`padding: 1em`,
`color:
${ this.color} ` ,] let style = styles.join(
‘; ’) +
‘;’
if (
this.color ===
‘yellow’) { style +=
`background: #777;` }
else { style +=
`background: #EEE; ` }
let ragInfo =
`\t* Color:
${ this.color}

\n`

+
`\t* Size: W:
${ this.width} , H:
${ this.height} `

console.log(
“%cRAG INFO for: %s”, style,
this.name) } }
export
default Rag

script.js

○

/** * Practice: Making classes and objects * * – Find a type of object
you have more than one of in your house (eg. clothing, writing tools,
etc). * – Create a class describing this object type – its properties
and methods. * – Create several objects using the class. * – Test the
objecs by calling their properties and using their methods in the
console. */
/* Eric Hepperle 2022-11-23 */
import Rag
from
‘./Rag.js’
// const rag = {
// color: ‘yellow’,
// height: 8,
// width: 8,
// }
const rag1 =
new Rag(
“rag1 – The First!”,
“Red”,
8, 8,)
const rag2 =
new Rag(
“Rag2”,
“blue”,

```
// const rag = {
// color: 'yellow',
// height: 8,
// width: 8,
// }
const rag1 =
new Rag(
"rag1 – The First!",
"Red",
8, 8, )
const rag2 =
new Rag(
"Rag2",
"blue",
32, 32,
) const rag3 =
new Rag(
"Rag3",
'yellow',
9, 14, )
const rags = [rag1, rag2, rag3]
rags.forEach(

rag =>

{ rag.logRagInfo() })
```

The cool thing about this practice was creating the **logRagInfo** method. This method has the following features:

- declares an array of CSS styles **styles**
- joins styles in a string
- uses **style** string to format console.log() with '%c%s'
- Logs out instance info / rag properties in the rag.color
- Uses conditional to use a darker background if the color is yellow

---

Version 2

index.htm

○

```
<!DOCTYPE html>

< html
lang=
"en">

< head>

< meta
charset=
"UTF-8" />

< meta
name=
"viewport"
content=
"width=device-width, initial-scale=1.0"
/>

< title> </span >Practice: Making classes and objects
</ title>

< script
type=
"module"
src=
"Backpack.js">

</ script>

< script
type=
"module"
src=
"Rag.js">

</ script>

< script
type=
"module"
src=
"Basket.js">

</ script>

< script
type=
"module"
src=
"script.js">
```

< script
type=
“module”
src=
“Basket.js”>

</ script>

< script
type=
“module”
src=
“script.js”>

</ script>

</ head>

< body>

</ body>

</ html>

Basket.js Basket is class is made to contain other class objects.

○

/** * Basket.js * * Programmer: Eric Hepperle * Date 2022-11-23 * * Purpose: Basket is a class that can hold other objects */

class
Basket { constructor(contents, units, length, width, depth, area, volume, weight) {
this.contents = contents
this.units = units
this.length = length
this.width = width
this.depth = depth
this.area = area
this.volume = volume
this.weight = weight }
setContents(newContents) { this.contents = newContents } addContents(newContents) {
this.contents.push(newContents) }
calcArea() { let area =
this.width *
this.length
return
` ${area}
${ this.units} `

} getURL() { return
window.document.URL } getContentNames()
{ let out =
‘’
this.contents.forEach(

item =>

{ out += item.name + ‘, ‘ })
return out } }
export
default Basket

script.js

○

/** * Practice: Making classes and objects * * – Find a type of object you have more than one of in your house (eg. clothing, writing tools, etc). * – Create a class describing this object type – its properties and methods. * – Create several objects using the class. * – Test the objecs by calling their properties and using their methods in the console. */
/* Eric Hepperle 2022-11-23 */
import Rag
from
‘./Rag.js’
import Basket
from
‘./Basket.js’
const rag1 =
new Rag(
“rag1 – The First!”,
“Red”,
8, 8,)
const rag2 =
new Rag(
“Rag2”,
“blue”,
32, 32,
) const rag3 =

```
'./Rag.js'
import Basket
from
'./Basket.js'
const rag1 =
new Rag(
"rag1 – The First!",
"Red",
8, 8, )
const rag2 =
new Rag(
"Rag2",
"blue",
32, 32,
) const rag3 =
new Rag(
"Rag3",
'yellow',
9, 14, )
const rags = [rag1, rag2, rag3]
rags.forEach(

rag =>

{ rag.logRagInfo() }) const basket1 =
new Basket( [
'some stuff'],
'cm',
17, 13,
9, '',
'', '',
) // console.log("basket1:", basket1)
console.log(
"basket1.contents START:",
basket1.contents)
// ['some stuff']
basket1.contents.push(...rags)
console.log(
"basket1.contents:", basket1.contents)
// ['some stuff', Rag, Rag, Rag]
basket1.setContents(rags) console.log(
"basket1.contents:", basket1.contents)
// [Rag, Rag, Rag]
console.log(
"basket1 area:", basket1.calcArea())
console.log(
"URL:", basket1.getURL())
const basket2 =
new Basket( [rag1, rag3, {
name:
'apple',
type:
'granny smith',
rating:
'yum!' }] )
// basket2: 2 rags and an apple
console.log(
"basket2 START:", basket2)
basket2.addContents({ name:
'Chocolate Bar',
brand:
"Hershey",
color:
"brown" })
console.log(
"basket2:")
// console.log(basket2.contents.toString())</span >
let myVal = basket2.contents
console.log({ myVal })
console.log(
"Content items names:",
basket2.getContentNames())
/* Here we create 3 rag objects and 2 baskets, each with different
items. Each basket has a "contents" property which is and array can
contain any type. The basket class has several methods including: –
setContents: replaces the contents array completely – addContents:
pushes objects on the the contents array – calArea: calculates the area
from length and width – getURL: returns the url is in the address bar –
getContentNames: prints a list of names of every item in contents array
*/</span >
```

CONTINUE HERE

3.12 Global objects

RESOURCES & REFERENCE

- https://www.linkedin.com/learning/instructors/nigel-french)

(To Be Continued ...)

CONTINUE HERE

3.12 Global objects

RESOURCES & REFERENCE

- https://www.linkedin.com/learning/instructors/nigel-french)

(To Be Continued ...)

NEXT: 2.02 Adding page numbers and running heads

Date Published: 2022-11-26
Date Updated: —

Eric Hepperle

Eric loves to write code, play guitar, and help businesses solve challenges with code and design.

← Previous Post

Subscribe

Be the First to Comment!

0 COMMENTS