## 0. Introduction

### 1. JavaScript - The soil from which the modern web grows

### 2. How to use the exercise files

### 3. Exploring the lab environment

## 1. JavaScript - A Brief Introduction

### 1. JavaScript - First contact

### 2. Navigating the JS landscape

### 3. Tools for working with JavaScript

### 4. Linting and formatting

### 5. Get to know the browser console

### 6. JavaScript language basics

### 7. Learning JavaScript backward

### Chapter Quiz

Question 1 of 7

What does it mean when we say JavaScript is an "object-oriented" language?

JavaScript is uses objects for data storage and manipulation.

JavaScript objects enforce a directional flow of data. Their orientation tells us whether the data flows up or down.

JavaScript is modeled on a Java object, and provides a browser-level scripting language to interact with that Java object.

JavaScript is modeled around objects with properties and methods which can be handled in a modular fashion.

Correct

Question 2 of 7

What happens to the website and the code when you write code in the browser console?

When you write code in the console, that code only exists in the console. It doesn't do anything to the website or the window.

The console allows you to write code to your project directly in the browser. The code is automatically added to your JavaScript file in your project.

Code in the browser console impacts the current browser instance only. It exists in the console for as long as the window is open.

Correct

Code in the browser console impacts the website and is saved to the original files automatically. Once saved, you can recall the code in the console at any time.

Question 3 of 7

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.

Correct

They use only tabs for indentation.

They use only single quotes for strings.

They are a React expert and write only ES6.

Question 4 of 7

What is the natural environment for JavaScript?

The browser, server environments, and your computer.

Correct

The browser.

Incorrect

The code editor.

Server environments.

Question 5 of 7

What is ECMAScript?

A variant of JavaScript used by advanced developers.

Incorrect

The specification describing how browsers should implement and interpret JavaScript.

Correct

A specification describing the JavaScript programming language and how developers use it.

A modern version of JavaScript used by React developers.

Question 6 of 7

Where should you develop and test your JavaScript code?

Develop in a code editor, test in your primary browser.

Incorrect

Develop in the browser, test in the same browser.

Develop in a code editor, test in as many browsers as you can get your hands on.

Correct

Develop in command line tools like the browser console, test on a different computer.

Question 7 of 7

Why have command line and automation tools become popular in JavaScript development?

They simplify complex processes and introduce features to help developers write better code.

Correct

They add complexity to the development setup to ensure only real JavaScript developers are able to do the work and that amateurs give up.

They help developers write better code by transforming the code for browser optimization.

They make developers look like hackers from movies and are great for showing off.

## 2. Get Up and Running with JS

### 1. JavaScript in an HTML document

### 2. JavaScript as an external file

### 3. Modern JavaScript loading

### 4. JavaScript modules

### Chapter Quiz

Question 1 of 4

When does the browser execute JavaScript?

By default: When the script is encountered. If the script is set to "async", asynchronously as the HTML page loads. If the script is set to "defer", when all other script is executed.

Incorrect

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.

Correct

By default: After the HTML page is rendered. If the script is set to "async", asynchronously while the HTML page is rendered. If the script is set to "defer", when the user performs an action.

Incorrect

Always when the script is encountered. The difference between "async" and "defer" is in what order the scripts are executed.

Question 2 of 4

What is the correct markup for adding an external JavaScript file to an HTML document?

<script src="javascript.js" async></script>

Correct

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.

<script src="javascript.js" async="true"></script>

<script type="text/javascript">javascript.js</script>

<script src="javascript.js" type="text/javascript"></script>

Question 3 of 4

What happens when you defer JavaScript?

The browser loads and executes the JavaScript after all HTML is rendered.

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

Correct

The browser loads the JavaScript asynchronously when it is encountered, then executes it when it is fully loaded.

Incorrect

The browser defers loading of the JavaScript until a function tells it to load and execute it.

Question 4 of 4

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.

Correct

Modules are required for modern JavaScript frameworks to work properly.

Incorrect

Modules allow developers to split code into multiple files for easier sharing and maintenance.

Incorrect

Modules enable modern loading of JavaScript to improve performance.

Incorrect

JavaScript modulesReplay

Review this video

JavaScript modules

4m 18s

## 3. Objects

### 1. Objects - A practical introduction

### 2. JavaScript objects - The code version

### 3. Object containers

### 4. Object properties

### 5. Accessing objects

### 6. Accessing object properties

### 7. Practice - Build a new object

### 8. Prototype inheritance

Prototype Inheritence

Consider this simplified version of the backpack object:

const backpack = {

name: "Everyday Backpack",

volume: 30,

};

How does JavaScript know how to turn this code into an object? By using a template, every object in JavaScript is based on a prototype object, a template built into JavaScript itself. That prototype becomes part of every object you make, and you can find it as a hidden property called [[Prototype]]. Inspect the object in the console and you can inspect the prototype:

Screenshot of browser console showing the hidden [[Prototype]] property expanded to the first level.

For an object literal like this ("object literal" meaning it's created with curly braces {} rather than using an object constructor which is covered later in the course), the prototype can be accessed using Object.prototype. This is the core prototype for all objects created with object literals or the Object() constructor and it's the first link of the prototype chain (unless you explicitly set another prototype).

This is what's referred to as "prototype inheritance" in JavaScript. Every object you work with inherits its structure from another prototype object, all the way up to the core Object.prototype. This means every object you work with inherits the built-in properties and methods of its prototypes!

So even though the simplified `backpack` object above only has two properties - name and volume - it also has access to all the methods and properties of Object.prototype. Two of these methods are valueOf() and toString(). Here's an example of what they do:

// Return the primitive value of an object property.

console.log(backpack.volume.valueOf()); // 30

// Return a string representation of the object property.

console.log(backpack.volume.toString()); // "30"

These and other methods are available to every object in JavaScript because all objects inherit the methods of the Object.prototype object. Furthermore, as you start making your own object constructors, you can add new properties and methods for their children to inherit. You can even override the inherited properties and methods like this:

const backpack = {

name: "Everyday Backpack",

volume: 30,

toString: function () {

return `Backpack: ${this.name}, Volume: ${this.volume} liters`;

},

};

console.log(backpack.toString()); // "Backpack: Everyday Backpack, Volume: 30 liters"

As we move through the course, inspect your objects in the console to see the prototype chain. You'll notice objects built from constructors have a prototype chain that includes the constructor as the first prototype, then the constructor's prototype object, and so on. This way you create a chain of objects that inherit from each other, building an expanding list of available properties and methods.

### 9. Object methods

### 10. Practice - Build a new method

### 11. Classes - Object blueprints

### 12. Object constructors

### 13. Extending classes

### 14. Practice - Build a new object with a constructor

### 15. Global objects

### 16. Challenge intro - Create a new object type

### 17. </> Code Challenge - Create a new object type

### Instructions

### **Create a new object constructor**

You are given a set of variables containing different data about a book:

const bookTitle = "Alice's Adventures in Wonderland"

const bookAuthor = "Lewis Carroll"

const bookPubYear = 1865

const bookISBN = 9798369203415

**Your task**:

* Make an object constructor for Book objects.
* The constructor must have properties for title, author, ISBN, and publicationYear.

When used, the constructor produces an object from the following input:

const book = new Book(bookTitle, bookAuthor, bookISBN, bookPubYear);

### **Parameters**

**document**: The `document` property of the project DOM.

### Result

**object**: New book object containing the four properties title, author, ISBN, and publicationYear.

### **Constraints**

* The object constructor can be a function declaration, function expression, or class.

### Want a hint?

Learn about creating new DOM elements in [this course](https://www.linkedin.com/learning/javascript-essential-training/) on LinkedIn Learning.

### Answer

// JavaScript code​​​​​​‌​‌‌​‌​‌​‌​​‌​‌‌​‌​​‌​​​‌ below

// Write your answer here, and then test your code.

// Your job is to implement the findLargest() method.

// Change these boolean values to control whether you see

// the expected answer and/or hints.

const showExpectedResult = false;

const showHints = false;

const bookTitle = "Alice's Adventures in Wonderland"

const bookAuthor = "Lewis Carroll"

const bookPubYear = 1865

const bookISBN = 9798369203415

// Your code goes here

function Book(title, author, ISBN, publicationYear) {

this.title = title;

this.author = author;

this.ISBN = ISBN;

this.publicationYear = publicationYear;

}

### Test code

// This is how your code will be called.

// Your answer should be the largest value in the numbers array.

// You can edit this code to try different testing cases.

const **result** = new Book(**bookTitle**, **bookAuthor**, **bookISBN**, **bookPubYear**);

### Console output

✓ Test passed: Title matches data.

✓ Test passed: Author matches data.

✓ Test passed: ISBN matches data.

✓ Test passed: Publication year matches data.

Tests passed: 4 of 4

> All tests passed

That's it! You're getting good at this.

Your code returned:

Book {

title: "Alice's Adventures in Wonderland",

author: 'Lewis Carroll',

ISBN: 9798369203415,

publicationYear: 1865

}

--- -- -- -- -- -- -- -- -- -- -- --

### 18. Solution - Create a new object type

### Chapter Quiz

Question 1 of 15

Given the code below, how do you access the property named in let propName?

let propName = "color"

const myObject = {

ID = 3,<br />

color = "pink",

propLength = 4,

use = false

};

Using bracket notation:

myObject.[propName]

Using the getProperty() method:

myObject.getProperty(propName)

Incorrect

Using dot notation:

myObject.propName

Incorrect

Using bracket notation:

myObject[propName]

Correct

Question 2 of 15

In the following object, what is the code in the second line called?

const myObject = {

color: "pink"

};

An object attribute defined using a key/value pair.

Incorrect

An object argument with an argument name and an argument value.

Incorrect

An object variable defined using a key/value pair.

Incorrect

An object property with a property name and a property value.

Correct

JavaScript objects: The code versionReplay

Review this video

JavaScript objects: The code version

2m 57s

Question 3 of 15

Why is the best-practice to place objects inside constants?

So the object and its properties is unchangeable.

So the object isn't accidentally altered or overwritten.

Correct

So the object can be used anywhere in the code.

So the object remains in a constant place in the code.

Question 4 of 15

Which of the below object property names are not valid?

const myObject = {

propName = "property", // line 1

prop-name = "hyphenated", // line 2

3rdProp = "numbered", // line 3

$prop = "dollar", // line 4

%prop = "percentage", // line 5

prop name = "space" // line 6

};

Lines 3, 4, and 5

Lines 3, 4, 5, and 6

Lines 2, 3, 5, and 6

Correct

Lines 2, 4, 5, and 6

Incorrect

Question 5 of 15

How do you access an object in JavaScript?

Call the object using the getObject([object name]) method

Call the object using bracket notation: [object name]

Call the object using the new keyword.

Incorrect

Call the object by naming its container.

Correct

Question 6 of 15

Can an object created from a class be given the same name as the class?

Yes: The object is a different data type from the class and they can both have the same name.

Incorrect

No: If you give an object the same name, it will be ignored by the browser.

Yes: If the class is defined using a var or a let, you can create an object with the same name without conflict.

No: If the class is a constant, this will cause an error. If the class is not a constant, the new object will overwrite the class.

Correct

Question 7 of 15

How do you define an object in JavaScript?

Create a variable, give it a name, and assign it an object using square brackets:

const myObject = [

// Properties and methods go here.

];

Incorrect

Create a variable, give it a name, and assign it an object using parentheses:

const myObject = (

// Properties and methods go here.

);

Create a variable, give it a name, and assign it an object using the "new" keyword:

const myObject = new Object(

// Properties and methods go here.

);

Incorrect

Create a variable, give it a name, and assign it an object using curly brackets:

const myObject = {

// Properties and methods go here.

};

Correct

Question 8 of 15

What does the this keyword refer to in a class?

this refers to the current class itself.

Incorrect

this refers to the class constructor.

this refers to the current object created from the class.

Correct

this refers to the current property.

Question 9 of 15

Where do you go to find official documentation and code examples for standard built in (global) objects?

Google or your preferred search engine

the W3Schools reference for standard built-in objects

the MDN Web Docs for standard built-in objects

Correct

the W3C reference for standard built-in objects

Question 10 of 15

How do you create a new object from a class?

Using the new keyword, naming the class, and passing the properties as parameters.

Correct

Using the newObject method, naming the class, and passing the properties as parameters.

Using the new Object phrase, naming the class, and passing the properties as parameters.

Creating a new variable and setting it equal to the class name followed by curly brackets wrapped around the properties.

Question 11 of 15

What is one advantage to using a class over an object constructor method?

Classes are easier to understand.

Classes can be extended.

Correct

Classes can be added to their own JavaScript module.

Classes are less likely to cause errors.

Question 12 of 15

What is the established convention for formatting objects?

Properties are listed on one line, methods on their own separate lines.

All properties and methods are listed on one line.

Properties are listed on their own separate line, methods are listed on one line.

All properties and methods are listed on their own separate line.

Correct

Question 13 of 15

What is the difference between a function and a method?

"functions" and "methods" are two words for the same thing.

Incorrect

A function is a function within an object. A method is a stand-alone function.

A function is a stand-alone function. A method is a function within an object.

Correct

A function is declared using the function keyword. A method is declared using the method keyword.

Question 14 of 15

Can you use arrow functions to create object methods?

No, object methods are defined using function declarations.

Incorrect

Yes, arrow functions can be used.

No, object methods must be declared using function expressions or the method definition shorthand.

Correct

Yes, arrow functions can be used, but only when they are called in as an external function.

Question 15 of 15

When creating a class, the prototype methods are added inside the constructor method.

FALSE

Correct

Class prototype methods are added after the constructor method as discrete named methods.

TRUE

## 4. Sidebar - String Output

### 1. Mix text and variables with template literals

### 2. Traditional string output

### 3. Challenge intro - Create a template literal

### 4. </> Code Challenge - Create a template literal

### Instructions

### **Create a string from object data using template literals**

Your starting code contains a constructor for the Camera object capturing information about cameras, and a function cameraAge() to calculate the age of a camera. The test code has a constant myCamera using the Camera object constructor to create a new object.

**Your task**: Add a template literal to the cameraStory constant matching the Result template below, populated with the data from the myCamera object.

### **Parameters**

**myCamera**: An object generated from the Camera object constructor.

### Result

**string**: "My camera is a [brand] [model] made in [year] making it [camera age] years old. It's a [format] camera with a [lens] lens using [film type] film."

### **Constraints**

* The output must match the Result example above exactly.
* Camera age is calculated using the cameraAge() function which receives a number as an argument.

### **Example 1:**

Input: "Hasselblad", "500C/M", 1963, "medium format", "Carl Zeiss 80mm f/2.8 Planar T", "120"

Result:

"My camera is a Hasselblad 500C/M made in 1963 making it 60 years old. It's a medium format camera with a Carl Zeiss 80mm f/2.8 Planar T lens using 120 film."

### Want a hint?

Learn about JavaScript arrays in [this course](https://www.linkedin.com/learning/javascript-essential-training/) on LinkedIn Learning.

### Answer

// JavaScript code below

// Write your answer here, then test your code.

// Your job is to implement the findLargest() method.

// Change these boolean values to control whether you see

// the expected answer and/or hints.

const **showExpectedResult** = false;

const **showHints** = false;

function Camera(brand, model, year, format, lens, filmType) {

    this.brand = brand;

    this.model = model;

    this.year = year;

    this.format = format;

    this.lens = lens;

    this.filmType = filmType;

}

const **getCurrentYear** = () => new Date().getFullYear();

const **cameraAge** = (year) => **getCurrentYear**() - year;

// Goal output:

// My camera is a [brand] [model] made in [year] making it [age] years old. It's a [format] camera with a [lens] lens using [filmtype] film.`

const **cameraHTML** = (myCamera) => {

    // Your code goes between the backticks in `cameraStory` below.

    const **cameraStory** = `My camera is a ${myCamera.brand} ${myCamera.model} made in ${myCamera.year} making it ${**cameraAge**(myCamera.year)} years old. It's a ${myCamera.format} camera with a ${myCamera.lens} lens using ${myCamera.filmType} film.`;

    return **cameraStory**;

}

### Test code

// This is how your code will be called.

// Your answer should be the largest value in the numbers array.

// You can edit this code to try different testing cases.

const **myCamera** = new Camera("Hasselblad", "500C/M", 1963, "medium format", "Carl Zeiss 80mm f/2.8 Planar T", "120")

const **result** = **cameraHTML**(**myCamera**);

### Console output

✓ Test passed: Output matches test case.

Tests passed: 1 of 1

> All tests passed

Well done! You reached the expected result.

Your code returned:

My camera is a Hasselblad 500C/M made in 1963 making it 61 years old. It's a medium format camera with a Carl Zeiss 80mm f/2.8 Planar T lens using 120 film.

--- -- -- -- -- -- -- -- -- -- -- --

### 5. Solution - Create a template literal

### Chapter Quiz

Question 1 of 2

How do you declare a JavaScript expression inside a template literal?

Using curly brackets:

const myString = `Some text and an {expression}.`;

Using a dollar symbol followed by curly brackets:

const myString = `Some text and an ${expression}.`;

Correct

Using a dollar symbol followed by square brackets:

const myString = `Some text and an $[expression].`;

Using an ampersand symbol followed by curly brackets:

const myString = `Some text and an &{expression}.`;

Question 2 of 2

What does the following code output in the console?

let number = 5;

let subtracted = 5-1;

console.log("The number before" + number + "is" + subtracted + ".");

The number before 5 is 4.

The number before 5 is 5-1.

The number before [number] is [subtracted].

The number before5is4.

Correct

## 5. DOM

### 1. DOM - The Document Object Model

### 2. Access elements with querySelector methods

### 3. Access elements using older methods

### 4. Practice - Find an element

### 5. Modifying element classes

### 6. Attributes

### 7. Inline style

### 8. Working with element attributes

**Working with data-\* Attributes**

The cool thing about HTML elements is you can add as many attributes as you like to any one of them, and then use JavaScript to create, read, update, modify, and delete them at will. It's useful to think of these attributes as feature lists for the element. Each attribute can change the element's behavior or appearance or even relationship with other elements.

As we've already covered, for a standard HTML element, its attributes can be manipulated using methods like [getAttribute()](https://developer.mozilla.org/en-US/docs/Web/API/Element/getAttribute" \t "_blank), [setAttribute()](https://developer.mozilla.org/en-US/docs/Web/API/Element/setAttribute" \t "_blank), and [removeAttribute()](https://developer.mozilla.org/en-US/docs/Web/API/Element/removeAttribute" \t "_blank). These methods allow you to interact directly with the element's attributes by dynamically editing the HTML source code without reloading the page.

Given the following HTML element...

<button id="myButton" class="btn btn-primary" type="button">Click Me!</button>

...you can manipulate its attributes like this:

const button = document.querySelector("#myButton");

// Get the value of the 'class' attribute

console.log(button.getAttribute("class")); // "btn btn-primary"

// Overwrite the value of the 'class' attribute

button.setAttribute("class", "btn btn-secondary");

// Remove the 'type' attribute

button.removeAttribute("type");

These manipulations serve as straightforward examples of a much broader and more powerful concept. In addition to describing the element by adding basic attributes like classes, attributes can be used to store custom data and even define roles for accessibility. One common use of this feature is to create custom data attributes with a data-\* prefix and retrieve them using the [dataset](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dataset) property.

As an example, in the below example a custom data-sku attribute is appended to each available product, and the SKUs of the selected products is passed on when the submit button is pressed:

<form id="productForm" onsubmit="submitProducts(event)">

<fieldset>

<legend>Select products:</legend>

<label for="product1">

<input type="checkbox" id="product1" class="product" data-sku="SKU1234" />

Product 1

</label>

<label for="product2">

<input type="checkbox" id="product2" class="product" data-sku="SKU5678" />

Product 2

</label>

<label for="product3">

<input

type="checkbox"

id="product3"

class="product"

data-sku="SKU91011"

/>

Product 3

</label>

<button type="submit">Submit Selection</button>

</fieldset>

</form>

<form id="productForm" onsubmit="submitProducts(event)">

<fieldset>

<legend>Select products:</legend>

<label for="product1">

<input type="checkbox" id="product1" class="product" data-sku="SKU1234" />

Product 1

</label>

<label for="product2">

<input type="checkbox" id="product2" class="product" data-sku="SKU5678" />

Product 2

</label>

<label for="product3">

<input

type="checkbox"

id="product3"

class="product"

data-sku="SKU91011"

/>

Product 3

</label>

<button type="submit">Submit Selection</button>

</fieldset>

</form>

Custom data attributes are frequently used in JavaScript frameworks and libraries to store additional information about an element. They are also used to define roles for accessibility, such as [aria-\*](https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Attributes) attributes. These attributes are used to provide additional information to assistive technologies like screen readers, making the web more accessible to users with disabilities.

### 9. Practice - Modify classes and attributes, and styles

### 10. Add DOM elements

### 11. Challenge intro - Create elements

### 12. </> Code Challenge - Create elements

### Instructions

### **Create new elements to populate a navigation menu**

You are given an existing DOM containing a `<header class="siteheader">` element, and a JavaScript template literal containing list items to be used in a navigation menu.

**Your task**:

* Add the list items into an unordered list
* Add the unordered list into a <nav> container
* Give the <nav> container the class .main-navigation
* Append the new <nav> container to the existing element with the class .siteheader

The final HTML for your `<nav>` element should follow the structure below:

<nav class="main-navigation">

<ul>

<li><a href="#">Home</a></li>

(...)

</ul>

</nav>

### **Parameters**

**document**: The `document` property of the project DOM.

### Result

Serialized HTML fragment describing the `<header class="siteheader">` element and its descendants.

### **Constraints**

* The **navContent** variable contains a template literal containing HTML list items and should be added as innerHTML to a `<ul>` element.

### Want a hint?

Learn about creating new DOM elements in [this course](https://www.linkedin.com/learning/javascript-essential-training/) on LinkedIn Learning.

### Answer

// JavaScript code below

// Change these values to control whether you see

// the expected answer and/or hints.

const **showExpectedResult** = false

const **showHints** = false

// Setup data

const **navContent** = `

      <li><a href="#">Home</a></li>

      <li><a href="#">About</a></li>

      <li><a href="#">Backpacks</a></li>

      <li><a href="#">Other things</a></li>

      <li><a href="#">Contact</a></li>

`;

function createNavMenu(document) {

      // Your code goes here

      const **mainNav** = document.createElement("nav")

**mainNav**.classList.add("main-navigation")

      const **navList** = document.createElement("ul")

**navList**.innerHTML = **navContent**

**mainNav**.append(**navList**)

      document.querySelector(".siteheader").append(**mainNav**)

}

### Test code

// This is how your code will be called.

// Your answer should be the HTML output described in the challenge text.

// You can edit this code to try different testing cases.

const **result** = document.querySelector(".siteheader").outerHTML;

### Console output

✓ Test passed: Use document.createElement() method.

✓ Test passed: Create mainNav.

✓ Test passed: Create <nav> element.

✓ Test passed: .main-navigation class appended to <nav> element.

✓ Test passed: Create <ul> element.

✓ Test passed: Content in <ul> matches original.

✓ Test passed: Output matches test case.

Tests passed: 7 of 7

> All tests passed

You did it! This result is exactly right.

Your code returned:

<header class="siteheader">

<div class="site-title">Experimental Site</div>

<div class="site-description">Description of experimental site</div>

<nav class="main-navigation"><ul>

<li><a href="#">Home</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Backpacks</a></li>

<li><a href="#">Other things</a></li>

<li><a href="#">Contact</a></li>

</ul></nav></header>

--- -- -- -- -- -- -- -- -- -- -- --

### 13. Solution - Create elements

### Chapter Quiz

Question 1 of 11

What method(s) would you use to check if an element has a specific ID and if so replace it with a different ID?

The element.setAttribute() method.

The element.attribute property.

The element.hasAttribute() method.

the element.hasAttribute() and element.setAttribute() methods.

Correct

Question 2 of 11

What is the difference in the return from the element.className and element.classList properties?

element.className returns a string containing the first class appended to the element. element.classList returns a string containing all classes appended to the element.

Incorrect

element.className returns the HTML content of an element with the specified class. element.classList returns a list of all classes used in the document.

Incorrect

element.className returns an array with each class appended to the element as an array item. element.classList returns a DOMTokenList with each class appended to the element.

Incorrect

element.className returns a string containing all classes appended to the element. element.classList returns a DOMTokenList with each class appended to the element.

Correct

Modifying element classesReplay

Review this video

Modifying element classes

4m 53s

Question 3 of 11

What does the HTML markup of this image element look like after the following script has executed?

const newImage = document.createElement("img");

newImage.classList.add("feat-img");

newImage.setAttribute("src", "logo.svg");

newImage.setAttribute("alt", "The company logo");

newImage.style.cssText = "display: block";

<img style="display: block;" src="logo.svg" alt="The company logo">

Incorrect

<img class="feat-img" src="logo.svg" title="The company logo" style="display: block;">

Incorrect

<img class="feat-img" src="logo.svg" alt="The company logo" style="display: block;">

Correct

<img class="feat-img display: block;" src="logo.svg" alt="The company logo">

Question 4 of 11

What is the value of const target after this code has executed?

const target = document.querySelectorAll("a");

A single element object of the first element matching the query.

Incorrect

An array containing each element object matching the query.

Incorrect

A node list containing each element object matching the query if there is more than one match, a single element object if there is only one match.

A node list containing each element object matching the query.

Correct

Question 5 of 11

What is the value of const target after this code has executed (assuming there are elements in the DOM with the class "note")?

const target = document.getElementsByClassName(".note");

An array-like HTMLCollection object containing each element with the class "note".

Incorrect

The getElementsByClassName() method receives a string of classnames to be found as its parameter Adding the . in front of the classname invalidates the query and no match is produced.

An empty array-like HTMLCollection object.

Correct

The getElementsByClassName() method receives a string of classnames to be found as its parameter Adding the . in front of the classname invalidates the query and no match is produced.

A node list containing each element with the class "note".

A value of "undefined"

Question 6 of 11

The querySelector() and querySelectorAll() methods use what kind of selectors as their parameter?

Any HTML element or classname.

An HTML selector string.

A CSS selector string.

Correct

A custom function.

Question 7 of 11

What happens if you run this code:

const target = document.querySelector(".first-paragraph");

target.style.font-family = "sans-serif";

"Uncaught SyntaxError: Invalid left-hand side in assignment"

Correct

JavaScript does not allow for hyphens in property names. When targeting CSS properties, use camelCase, so the "font-family" property becomes "fontFamily".

The inline style for the target element is set to "font-family: sans-serif;"

Incorrect

JavaScript does not allow for hyphens in property names. When targeting CSS properties, use camelCase, so the "font-family" property becomes "fontFamily".

Nothing. The syntax is wrong so the browser ignores this line of code.

"Uncaught SyntaxError: Invalid CSS selector"

Question 8 of 11

What does the element.classList.toggle() method do?

Adds the a class named "toggle" to the element.

Incorrect

Adds the specified class if it is not appended to the element, removes the specified class if it is appended to the element.

Correct

Adds the specified class if it is not appended to the element, leaves the specified class in place if it is appended to the element.

Toggles the display property from "none" to "block" to hide or show the element.

Question 9 of 11

Where in the HTML document does the new element appear when you use the document.createElement() method?

Nowhere: The element is created, but has not been added to the DOM.

Correct

At the bottom of the HTML document.

Incorrect

Inside the specified element container.

At the top of the HTML document.

Incorrect

Question 10 of 11

What is the "DOM"?

DOM is short for "Document Object Model", the document object the browser creates when it renders an HTML document.

Correct

DOM is the JavaScript environment the browser creates for each HTML document.

DOM is the name of a 2012 album by German singer Joachim Witt

DOM is short for "Document Object Master", the original document the browser renders.

Question 11 of 11

How do you add an element created using createElement() to the DOM?

Using the insertAdjacentElement() method.

All these options and more.

Correct

Using the prepend() method.

Using the append() or appendChild() methods.

## 6. Sidebar - Variables and Data Types

### 1. Variables - Containers for everything

### 2. Var

### 3. Scope

### 4. Let

### 5. Const

### 6. Data types

### 7. Typing in JavaScript

### Stricter Typing in JavaScript

JavaScript is a weakly typed language, meaning variables can hold values of any type without declaring what type of data they hold.

This can produce unexpected behaviours if we're not careful about the data types we use.

Consider this basic addition function:

function add(a, b) {

return a + b;

}

console.log(add(5, 10)); // Output: 15

console.log(add("5", 10)); // Output: "510"

Without strong typing, we can't tell the function to only accept numbers as arguments. As a result, if we accidentally pass in a string instead of a number, the function will concatenate the values together instead of adding them, and suddenly "5" + "10" becomes "510."

This stands in sharp contrast to many other languages including Java, C++, and Python which are more strongly typed. In strongly typed languages, you declare the type of a variable when you define it, and if the data type does not match the declaration, a warning or error is triggered. This can help prevent errors and bugs in code which is why strongly typed languages are often preferred for large applications.

But as I explained, JavaScript is a weakly typed language, yet many of the biggest applications in the world are built using JavaScript. Does that mean those applications are less secure?

There are several connected answers here:

First, to introduce strong typing to JavaScript applications, many developers use [TypeScript](https://www.typescriptlang.org/), a strongly typed variant of JavaScript that adds static typing to solve this exact problem (thus the name). TypeScript is then compiled into JavaScript before being run in the browser.

Second, we can mimic stronger typing in JavaScript using some basic strategies. Let's break them down.

First, we can use the [typeof](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof" \t "_blank) operator to identify the data type of any value, and then test to see if the data type matches what is expected. You manually verify data types at runtime to ensure they match expectations.

Implementing this strategy for our add function would look like this:

function add(a, b) {

if (typeof a !== "number" || typeof b !== "number") {

throw new TypeError("Both arguments must be numbers");

}

return a + b;

}

This approach could be expanded further to identify the type of each argument and tell the developer what data is being passed in. It can also be expanded to include more complex data types like objects or arrays.

We can expand on this approach and make it DRY by creating a generic assert function that throws an error when data does not meet a specified type. This function can then be reused any time we need type checking.

Here's what the add function would look like using an assert function:

function assertType(value, type) {

if (typeof value !== type) {

throw new TypeError(`Expected ${type}, but got ${typeof value}`);

}

}

function add(a, b) {

assertType(a, "number");

assertType(b, "number");

return a + b;

}

For larger applications, this DRY approach is quicker to implement and easier to understand and maintain.

For more complex data types, we can integrate type checks directly into constructor functions or classes to ensure objects are created with valid types.

The example below shows the familiar Backpack class with type checks added to the constructor using an expanded assertType() function:

function assertType(value, type, paramName) {

if (typeof value !== type) {

throw new TypeError(`${paramName} must be a ${type}`);

}

}

class Backpack {

constructor(

name,

volume,

color,

pocketNum,

strapLengthL,

strapLengthR,

lidOpen

) {

assertType(name, "string", "name");

assertType(volume, "number", "volume");

assertType(color, "string", "color");

assertType(pocketNum, "number", "pocketNum");

assertType(strapLengthL, "number", "strapLengthL");

assertType(strapLengthR, "number", "strapLengthR");

assertType(lidOpen, "boolean", "lidOpen");

this.name = name;

this.volume = volume;

this.color = color;

this.pocketNum = pocketNum;

this.strapLength = {

left: strapLengthL,

right: strapLengthR,

};

this.lidOpen = lidOpen;

}

toggleLid(lidStatus) {

assertType(lidStatus, "boolean", "lidStatus");

this.lidOpen = lidStatus;

}

newStrapLength(lengthLeft, lengthRight) {

assertType(lengthLeft, "number", "lengthLeft");

assertType(lengthRight, "number", "lengthRight");

this.strapLength.left = lengthLeft;

this.strapLength.right = lengthRight;

}

}

This last example hints at why type checking is so important. When we start working with larger functions and complex data—especially if that data comes from an outside source like an API—it is vital to ensure the data going into different functionality is the type we expect it to be.

The techniques showcased here give you more control over data types in your code, and for many applications they will be sufficient. However, if you find yourself writing a lot of type checking code, it might be time to switch your project to TypeScript. That way, you get all the goodness of JavaScript with strong typing built right in!

### 8. Assignment vs. comparison

### 9. Math operators

### 10. Challenge intro - Calculate sales tax

### 11. </> Code Challenge - Calculate sales tax

### Instructions

### **Calculate taxes and totals**

You are given an almost completed setup to calculate taxes and totals for a store in British Columbia, Canada. There are two taxes, GST and PST, and both need to be applied independently to the before-tax total to get the final price.

**Your task**: Complete the calculateTotal() function by adding GST, PST, and sum properties to the totals object and performing the necessary math to get the right numbers in each property. You can see how these properties are displayed in the showTotals function below.

### **Parameters**

**prices**: A random array of prices, defined in the test code.

### Result

**string**: A text display of the Before tax price, GST and PST values, and the Sum total.

### **Constraints**

* The prices array always contains **at least one number**.
* The prices array may contain **negative numbers**.

### **Example 1:**

Input: [2.5, 9.99, 3.99, 18.59, 49.96]

Result:

Before tax: $85.03

GST: $4.25

PST: $6.80

-----------------

Sum total: $96.08

### Want a hint?

Learn about JavaScript arrays in [this course](https://www.linkedin.com/learning/javascript-essential-training/) on LinkedIn Learning.

### Answer

// JavaScript code below

// Write your answer here, then test your code.

// Change these boolean values to control whether you see

// the expected answer and/or hints

const **showExpectedResult** = false;

const **showHints** = false;

const **taxGST** = 5;

const **taxPST** = 8;

const **calculateTotal** = (prices) => {

    const **totals** = {};

    // Add together all values from the prices array

**totals**.beforeTax = prices.reduce((a,b) => a + b, 0);

    // Your code starts here

**totals**.GST = **totals**.beforeTax \* (**taxGST** / 100)

**totals**.PST = **totals**.beforeTax \* (**taxPST** / 100)

**totals**.sum = **totals**.beforeTax + **totals**.GST + **totals**.PST

    // Your code ends here

    return **totals**;

}

const **showTotals** = (prices) => {

    const **totals** = **calculateTotal**(prices);

    return `

        Before tax: $${**totals**.beforeTax.toFixed(2)}

        GST: $${**totals**.GST.toFixed(2)}

        PST: $${**totals**.PST.toFixed(2)}

        -----------------

        Sum total: $${**totals**.sum.toFixed(2)}

    `

}

### Test code

// This is how your code will be called.

// Your answer should be the largest value in the numbers array.

// You can edit this code to try different testing cases.

// Nested array of random prices

const **pricesArr** = [

  [2.5, 9.99, 3.99, 18.59, 49.96],

  [2.99, 3.99, 4.99, 5.99, 6.99],

  [100.99, 200.99, 300.99, 400.99, 500.99],

  [1.99, -2.99, 3.99, -4.99, 5.99]

];

// Pick random array of prices

const **randomPrices** = Math.floor(Math.random() \* **pricesArr**.length);

const **result** = **showTotals**(**pricesArr**[**randomPrices**]);

### Console output

✓ Test passed: Before tax is correct.

✓ Test passed: GST is correct.

✓ Test passed: PST is correct.

✓ Test passed: Sum is correct.

Tests passed: 4 of 4

> All tests passed

That's it! You're getting good at this.

Your code returned:

Before tax: $24.95

GST: $1.25

PST: $2.00

-----------------

Sum total: $28.19

--- -- -- -- -- -- -- -- -- -- -- --

### 12. Solution - Calculate sales tax

### Chapter Quiz

Question 1 of 8

What does a single equals symbol in JavaScript indicate?

The values to the left and right are compared.

A value is matched with the value of a named variable.

The values to the left and right are equal.

A value is assigned to the named variable.

Correct

Question 2 of 8

What happens if you use a named variable without first declaring it using the var, let, or const keywords?

A const is created with the name and value assignment.

A global var is created with the name and value assignment.

Correct

JavaScript asks what type of variable this is.

An error is triggered and JavaScript stops rendering.

Question 3 of 8

What is the value of defaultColor when it is logged in the console?

var defaultColor = "purple";

function setColor() {

if ( defaultColor === "purple" ) {

let defaultColor = "orange";

}

}

setColor();

console.log(defaultColor);

Undefined

"purple"

Correct

The console.log statement happens in the global scope while the color is changed in a local scope where a separate let with the same name is created. let defaultColor only exists inside the function scope of setColor().

defaultColor

"orange"

Incorrect

The console.log statement happens in the global scope while the color is changed in a local scope where a separate let with the same name is created. let defaultColor only exists inside the function scope of setColor().

Question 4 of 8

How do you capture the result of a math equation like 42 \* 38 in JavaScript?

Create a variable to hold the math equation and then capture the result in another variable.

Create a variable and set it equal to the math equation.

Correct

Use the math() method.

Use the sum() method.

Question 5 of 8

In what scenario should you use var instead of let to define a variable?

When you need a variable to be available in other modules than the one it is defined (thus a "global" variable).

When you need a locally scoped reassignable variable only available within the current function scope.

When you need a globally scoped reassignable variable available to all functions and statements.

Correct

When you need a globally scoped unassignable variable available to all functions and statements.

Question 6 of 8

Which statement is true?

Array items within a constant cannot be changed.

Incorrect

A var or let variable containing an object cannot be reassigned to an array because it is a different data type.

Only const variables can be exported from a JavaScript module for import in another module.

Object properties within a constant can be changed.

Correct

Question 7 of 8

What is logged in the console after this code is executed?

let sum = 23.95;

let tip = "3";

console.log("The total is $" + sum + tip + ".");

Syntax error.

"The total is $26.95 + 3"

"The total is $26.95."

"The total is $23.953."

Correct

Question 8 of 8

What do the operators (equals symbols) in these three lines of code signify?

a = b

a == b

a === b

a = b assigns the value of b to a. a == b tests for identical equality between a and b. a === b is a misspelling of a == b and works the same way.

a = b tests for equality between a and b. a == b tests for identical equality between a and b. a === b tests for absolute equality between a and b.

a = b means b holds the result of whatever equation happens on the side of a. a == b means the value of b is assigned to a. a === b tests for equality between a and b.

a = b assigns the value of b to a. a == b tests for equality between a and b. a === b tests for identical equality between a and b.

Correct

## 7. Arrays

### 1. Arrays explained

### 2. Arrays in code

### 3. Array methods

### 4. Set - Arrays with unique items

Set: Arrays with Unique Items

The backpack featured in this course is the backpack I use any time I go on a trip and need to bring my laptop and other equipment.

Every time our family travels, my wife and I both make lists: arrays of items we need to pack. Those arrays ensure we don't forget anything, but if we're not careful, we end up adding the same item multiple times. To avoid this, we could combine the arrays into one, and then manually remove any duplicates. Or, we could use Set.

The Set object in JavaScript stores unique values of any type, with an emphasis on unique. You can't have the same value more than once in a Set. This makes it perfect for storing a collection of unique items, like the items in our packing list. Unlike an array where items are ordered and you access them by their index number, the items in a Set are unordered and cannot be accessed randomly.

Here's what our packing list looks like as a Set:

const itemsToPack = new Set();

itemsToPack.add("water bottle");

itemsToPack.add("laptop");

itemsToPack.add("headphones");

itemsToPack.add("snacks");

itemsToPack.add("book");

itemsToPack.add("sunscreen");

The Set is created using the new Set() constructor. We then add items to the Set using the add() method. The Set object has several methods to help you manage the collection of items:

- add(): Adds a new item to the Set

- has(): Checks if an item exists in the Set

- delete(): Removes an item from the Set

- size(): Returns the number of items in the Set

Once a Set has been created, we can use the above methods to modify it. For example, we can check if an item is in the Set and add it if it's not:

if (!itemsToPack.has("water bottle")) {

itemsToPack.add("water bottle");

} else {

console.log("Water bottle is already on the list!");

}

Accessing specific items within a Set

Set is often referred to as arrays with unique items (as in the title of this article), but this mental model may lead you astray until you get a full grasp on when to use a Set over an array. Specifically, one of the first things you probably want to know is how to access specific items within a Set.

When you feel this urge, ask yourself: Why do I need access to that specific item beyond knowing it's there (for which there is the has() method described above)? Chances are, if you're asking that question, you shouldn't be using a Set:

- If you need to access items by index, you should probably be using an array

- If you need to access items by key, you should probably be using an Object

- If you need to access items by value, you should probably be using a Map

Set is specifically for when you need to store a collection of unique items and then perform operations on that collection.

So what are the uses of Set then? Let's return to our packing list example. Once we have a list of all the items we need to pack, we can use the forEach() method to loop through the items and check to make sure they're actually packed:

const backpack1.items = ["water bottle", "laptop"];

const backpack2.items = [ "headphones", "snacks", "book"];

itemsToPack.forEach(item => {

if (backpack1.items.includes(item)) {

console.log(`${item} is packed in backpack 1`);

} else if (backpack2.items.includes(item)) {

console.log(`${item} is packed in backpack 2`);

} else {

console.log(`${item} is not packed`);

}

});

Run this code and you'll see which backpack contains which item from the list (and that we once again forgot to pack the sunscreen).

Set is one of those features you don't see a use for until you suddenly need it. It's a powerful tool for managing collections of unique items, and provides a quick and straightforward way of ensuring you don't have duplicates in your data, and removing them if you do.

### 5. Challenge intro - Array manipulation

### 6. </> Code Challenge - Array manipulation

### Instructions

### **Make changes to an array**

You are given an array of random objects. Using array methods, make changes to the array based on the following instructions:

**Your task**:

1. Remove the last item from the array.

2. Sort the remaining array items alphabetically.

3. Move the last item in the array to the first position.

4. Find the "USB drive" item in the array and move it to the last position.

5. Find the "laptop" item in the array and remove it.

### **Parameters**

**deskArray**: An array of random objects.

### Result

**array**: An array reformatted based on the above instructions.

### **Constraints**

* To pass the tests, the items in the original  deskArray  **cannot be changed.**

### **Example:**

Input:

[

"pen",

"camera",

"phone",

"notebook",

"headphones",

"laptop",

"light bulb",

"USB drive",

"elephant"

]

Result:

[

'phone',

'camera',

'headphones',

'light bulb',

'notebook',

'pen'

]

### Want a hint?

Learn about JavaScript arrays in [this course](https://www.linkedin.com/learning/javascript-essential-training/) on LinkedIn Learning.

### Answer

// JavaScript code below

// Write your answer here, then test your code.

// Change these boolean values to control whether you see

// the expected answer and/or hints

const **showExpectedResult** = false;

const **showHints** = false;

const **deskArray** = [

    "pen",

    "camera",

    "phone",

    "notebook",

    "headphones",

    "laptop",

    "light bulb",

    "USB drive",

    "elephant"

];

const **processArray** = deskArray => {

    let newDeskArr = [...deskArray];

    // Your code goes here

    newDeskArr.pop()

    newDeskArr.sort()

    newDeskArr.unshift(newDeskArr.pop())

    const **usblndex** = newDeskArr.indexOf("USB drive")

    newDeskArr.push(newDeskArr.splice(**usblndex**,1))

    const **laptoplndex** = newDeskArr.indexOf("laptop")

    newDeskArr. splice(**laptoplndex**,1)

    // Your code ends here

    return newDeskArr;

};

### Test code

// This is how your code will be called.

// Your answer should be the largest value in the numbers array.

// You can edit this code to try different testing cases.

const **result** = **processArray**(**deskArray**);

### Console output

✓ Test passed: Output matches test case.

Tests passed: 1 of 1

> All tests passed

Great work! You got the right answer.

Your code returned:

[

'phone',

'camera',

'headphones',

'light bulb',

'notebook',

'pen',

[ 'USB drive' ]

]

### 7. Solution - Array manipulation

### Chapter Quiz

Question 1 of 5

Given the array below, how would you access the item whose value is 7?

var numbers=[4, 7, 3, 5, 2];

numbers(2)

numbers[1]

Correct

numbers[2]

numbers(7)

Question 2 of 5

What will the numbers array look like when the following code finishes executing?

var numbers=[1, 2, 3, 4, 5];

numbers.push(6);

numbers.unshift(7);

numbers.pop();

numbers.shift();

1, 2, 3, 4, 5

Correct

7, 2, 3, 4, 6

1, 2, 3, 4, 6

Incorrect

44018

Question 3 of 5

What happens when you add a new array item to a previously undefined slot?

You can only add items in already defined slots.

A new slot is added to the beginning of the array.

Incorrect

A new slot is added to the end of the array.

A new slot is added corresponding to the slot number provided.

Correct

Question 4 of 5

The first item of an array has the index position 1.

FALSE

Correct

TRUE

Question 5 of 5

Arrays can hold a mix of any data type.

TRUE

Correct

FALSE

## 8. Functions and Methods

### 1. The real-world function

### 2. Functions and methods

### 3. A standard function

### 4. The arrow function

### 5. Arrow functions and "this"

### 6. Practice - Build a function

### 7. Pass data to a function with parameters

### 8. Return values from a function

### 9. Return - Deeper dive

Return: A Deeper Dive

The return keyword in JavaScript is far more versatile than both its name and general examples suggest. So let's dive a little bit deeper and see how simply returning from a function can open up a large possibility space.

For reference, here's the standard use of return in a function:

function sum(a, b) {

let result = a + b;

return result;

}

console.log(sum(5, 3)); // Outputs: 8

Here, return is doing its primary job: literally returning a value to the caller.

Early exit

Hidden by the standard example above is how return terminates the execution of any further parts of the function. This feature is crucial for controlling when complex functions run and for how long they run.

Consider this example where return is used to stop a function early:

function calculateDiscount(items) {

// Check if there is more than one item

if (items.length <= 1) {

console.log("Discount is only applied when there are more than one item.");

return; // Exits the function if not more than one item

}

// Calculate total price of all items

let totalPrice = items.reduce((sum, item) => sum + item.price, 0);

// Calculate 15% discount

let discount = totalPrice \* 0.15;

// Calculate final price after discount

let discountedPrice = totalPrice - discount;

// Return the discounted price

return discountedPrice;

}

// Example usage:

const items = [

{ name: "Book", price: 20 },

{ name: "Pen", price: 5 },

];

const discountedPrice = calculateDiscount(items);

console.log("Discounted price: $" + discountedPrice.toFixed(2));

Here the return statement is used to exit the function early if the condition is met, preventing the rest of the function from running. For large applications, this approach is essential for speed and efficiency.

Returning functions

return doesn't just terminate a process and return a value. It can also return functions, also called higher-order functions. This is a powerful feature that can be used to create configurable functions or for currying.

Here's an example of returning a function:

function createMultiplier(multiplier) {

return function (value) {

return value \* multiplier;

};

}

const double = createMultiplier(2);

const quintuple = createMultiplier(5);

console.log(double(5)); // Outputs: 10

console.log(quintuple(5)); // Outputs: 25

In this example, createMultiplier returns a function that multiplies a value by the multiplier parameter. We then use this returned function to create new functions that multiply by 2 and 5 respectively. Finally, we call these new functions with a value to see the results.

Modern JavaScript applications often rely on asynchronous operations to fetch data from servers through APIs or perform other tasks that take time. In these cases, return is used to handle the asynchronous results.

async function fetchData(url) {

let response = await fetch(url);

let data = await response.json();

return data; // Returns a promise that resolves to data

}

Here, return is used to return the data fetched from the server, and once the data is returned, we can output it to the console or use it in other parts of the application.

Inspecting code, you'll notice arrow functions often omit the return keyword altogether. This is because these functions have a special syntax that allows for implicit returns.

const add = (a, b) => a + b; // Implicit return

This feature of arrow functions simplifies function declarations significantly when only a single expression is involved.

### 10. Practice - Pass values between functions

### 11. Callbacks

### 12. Conditional if...else statement

### 13. Logical operators

### 14. Conditional switch statement

### 15. Looping through content

### 16. Using the map() array method

### 17. Challenge intro - Create a content factory

### 18. </> Code Challenge - Create a content factory

### 19. Solution - Create a content factory

### Chapter Quiz

Question 1 of 17

What will be printed to the console as the following script is running?

var sqrt = (function(x) {

console.log(x\*x);

})(my\_number)

my\_number = 5;

x\*x

Incorrect

25

Incorrect

5

Incorrect

This code will error out.

Correct

Replay

Review this video

NaNs

Question 2 of 17

In a switch statement, what happens when several of the switch cases resolve to true?

The switch statement returns all cases that resolves to true, then stops.

Incorrect

The switch statement returns the last case that resolves to true, then stops.

Incorrect

The switch statement returns the first case that resolves to true, then stops.

Correct

The switch statement returns an error. Each switch case should be unique.

Question 3 of 17

What is the new value of myArray after this script executes?

const myArray = [1, 2, 3, 4]

myArray.forEach( (item, index) => {

myArray[index] = ++item;

});

[1,2,3,4,2,3,4,5]

Incorrect

In addition to the current item, the callback function in the array.forEach() method has optional properties for the current item index number and the array itself. In this example, the index number is captured and used to transform the original array.

[1,2,3,4]

Incorrect

In addition to the current item, the callback function in the array.forEach() method has optional properties for the current item index number and the array itself. In this example, the index number is captured and used to transform the original array.

Syntax error: index not defined.

Incorrect

In addition to the current item, the callback function in the array.forEach() method has optional properties for the current item index number and the array itself. In this example, the index number is captured and used to transform the original array.

[2,3,4,5]

Correct

In addition to the current item, the callback function in the array.forEach() method has optional properties for the current item index number and the array itself. In this example, the index number is captured and used to transform the original array.

Looping through content

Replay

Review this video

Looping through content

5m 19s

Question 4 of 17

What is the difference between the array.forEach() and array.map() methods?

array.forEach() executes a provided callback function once for each item in the array.

array.map() creates an object map of the original array to be placed in a new variable.

Incorrect

array.forEach() executes a provided callback function once for each item in the array and returns it to the original array.

array.map() creates a new array with the results of executing a provided callback function once for each item in the original array.

Incorrect

array.forEach() creates a new array for the original data and executes a provided callback function once for each item in the array.

array.map() creates a new array with the results of executing a provided callback function once for each item in the original array.

Incorrect

array.forEach() executes a provided callback function once for each item in the array.

array.map() creates a new array with the results of executing a provided callback function once for each item in the original array.

Correct

Using the map() array method

Replay

Review this video

Using the map() array method

4m 52s

Question 5 of 17

Anonymous functions are dangerous because they can be triggered by accident.

TRUE

Incorrect

FALSE

Correct

Replay

Review this video

NaNs

Question 6 of 17

What logical operator signifies negation (not)?

≠

Incorrect

!

Correct

\*

NOT

Question 7 of 17

How do you capture a value returned from a function?

The value is returned to where the function was called, effectively replacing the function call.

Correct

The return keyword returns the function itself.

Use the catch() method to capture the value as it is returned.

Assign a variable to the function call and the returned value is placed in that variable.

Question 8 of 17

What is the purpose of the return keyword in this function?

const myFunction = (data, modifier) => {

if (data >= 3) {

return;

} else {

console.log(data + modifier);

}

}

The return keyword does nothing because it doesn't have a value to return.

Incorrect

The return keyword returns the value of the function if data is bigger than or equal to 3.

Incorrect

The return keyword stops the function at line 3.

Incorrect

The return keyword stops the function from executing if the conditional statement is true.

Correct

Practice: Pass values between functionsReplay

Review this video

Practice: Pass values between functions

2m 33s

Question 9 of 17

What is meant when we say a function has a callback?

A callback function literally calls back to another function informing it of its status.

Incorrect

A callback function is any function called back from inside another function.

Incorrect

A callback function is a function called back from another function and invoked inside the first function to complete some kind of routine or action.

Incorrect

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

Correct

CallbacksReplay

Review this video

Callbacks

5m 29s

Question 10 of 17

What is logged in the console when the following script executes:

const number = 10;

if (number <= 9) {

console.log("<=9");

} else if ( number >= 10 ){

console.log(">=10");

}

Nothing. The second if statement never executes.

Incorrect

">=10"

Correct

=9

Syntax error. You can't do else if like this.

Question 11 of 17

Which of the following options is correct syntax for an arrow function?

const myFunction => (data) = {

console.log(data);

};

Incorrect

function myFunction(data) => {

console.log(data);

};

Incorrect

myFunction(data) => {

console.log(data);

};

Incorrect

const myFunction = (data) => {

console.log(data);

};

Correct

The arrow functionReplay

Review this video

The arrow function

4m 49s

Question 12 of 17

True or false: An arrow function can be used to define an object method.

TRUE

FALSE

Correct

Question 13 of 17

What is logged in the console when this script executes?

const myFunction = (data) => {

return;

let newData = data + 1;

return newData;

}

console.log(myFunction(5));

6

Incorrect

When the `return` statement is encountered, the function ends. In this example, the `return` statement is the first entry in the function, so nothing else executes.

Nothing (undefined)

Correct

When the `return` statement is encountered, the function ends. In this example, the `return` statement is the first entry in the function, so nothing else executes.

51

Syntax error: Return used twice.

Question 14 of 17

What happens when you assign values to parameters in a function declaration as exemplified below:

const myFunction = (data = 5, color = "red") => {}

The values are added to those passed to the function.

Incorrect

Default values are created for each parameter. If no value is passed to the function, these values are used.

Correct

You get an error. This is incorrect syntax.

The parameter values are set. These values are used by the function even when other vallues are passed from the function call.

Question 15 of 17

Which answer is a valid example of calling the function expression defined below?

const myFunction = function(a) {};

myFunction(1);

Correct

a(1);

function(1);

function(myFunction)(1);

Question 16 of 17

What is the difference between a function expression and a function declaration?

A function expression defines a function with the specified parameters starting with the function keyword. A function declaration expresses a function inside a variable by assigning the function to the variable.

Incorrect

Function expressions can be executed directly. Function declarations need to be called as functions before they can be executed.

Incorrect

Function expressions are a more robust way of declaring functions than function expressions because they preserve this.

Incorrect

A function declaration defines a function with the specified parameters starting with the function keyword. A function expression expresses a function inside a variable by assigning the function to the variable.

Correct

Functions and methods

Replay

Review this video

Functions and methods

6m 23s

Question 17 of 17

What will the following script print?

function myfunc(x,y) {

return(x+y);

}

console.log(myfunc(2,myfunc(5,-2)));

2

Incorrect

7

Incorrect

NaN

5

Correct

## 9. Events

### 1. DOM events explained

### 2. Typical DOM events

### 3. Event listeners

### 4. Practice - Experiment with event listeners

### 5. Advanced event listeners and "this"

### 6. This - A Deeper dive

This: A Deeper Dive

The [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this) keyword in JavaScript has far broader uses than what its name and general examples suggest.

We encountered this when we discussed object constructors and classes. Here's our Backpack class for reference:

class Backpack {

constructor(

// Defines 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;

}

}

Here, this is used to refer to the object the method is called on, so when the class is used to create a new object the code literally says "this object's name property is equal to the value of the name argument". This is a common use of this in JavaScript, but it's not the only one.

To simplify code, JavaScript allows you to chain multiple methods together when working with data. In the below example, the concat(), toUpperCase(), and trim() methods are chained together to modify a string:

let greeting = "Hello";

greeting = greeting.concat(" World").toUpperCase().trim();

console.log(greeting); // Outputs: "HELLO WORLD"

Using the this keyword, you can enable method chaining in your own classes.

Here’s the Backpack class with two small modifications:

class Backpack {

constructor(

// Defines parameters:

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.lidOpen = lidOpen;

}

toggleLid(lidStatus) {

this.lidOpen = lidStatus;

return this; // Return the current object to enable chaining

}

newStrapLength(lengthLeft, lengthRight) {

this.strapLength.left = lengthLeft;

this.strapLength.right = lengthRight;

return this; // Return the current object to enable chaining

}

}

// Usage

const myBackpack = new Backpack("Hiker", 30, "blue", 5, 35, 35, false);

myBackpack.toggleLid(true).newStrapLength(25, 25);

Now both toggleLid and newStrapLength use this to return the instance after performing their actions, allowing for the methods to be called in a continuous chain.

The example above barely scratches the surface of what this can do, and also illustrates how quickly JavaScript can become complex. A good rule of thumb is to remember this always refers to the object the method is called on. And when you're ready to dive deeper (or this doesn't work as expected), refer back to the [official documentation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this) for help.

### 7. Pass arguments through event listeners

### 8. Automatically triggering events

### 9. Challenge intro - Create an event listener

### 10. </> Code Challenge - Create an event listener

### Instructions

### **Add event listeners to elements in a document**

You are given an existing DOM displaying the values of Name, Description, Color, and Volume of an item. Under each value is a form field and an "Update" button. Below is the HTML structure:

<div id="info">

<p class="info-item" id="name">Name: <span class="value">-</span></p>

<form id="update-name">

<label for="name-input">Update Name:</label>

<input type="text" id="name-input">

<button type="submit" data-target="name">Update</button>

</form>

<p class="info-item" id="description">Description: <span class="value">-</span></p>

<form id="update-description">

<label for="description-input">Update Description:</label>

<input type="text" id="description-input">

<button type="submit" data-target="description">Update</button>

</form>

<p class="info-item" id="color">Color: <span class="value">-</span></p>

<form id="update-color">

<label for="color-input">Update Color:</label>

<input type="text" id="color-input">

<button type="submit" data-target="color">Update</button>

</form>

<p class="info-item" id="volume">Volume: <span class="value">-</span></p>

<form id="update-volume">

<label for="volume-input">Update Volume:</label>

<input type="text" id="volume-input">

<button type="submit" data-target="volume">Update</button>

</form>

</div>

**Your task**:

* In the updatePage() function, add an event listener to each of the four forms in the HTML.
* Each event listener should listen for a submit event on the form the button belongs to.
* When the button is clicked, the event listener should update the contents inside <span class="value"> for the relevant value.

### **Parameters**

* **document**: The `document` property of the project DOM.
* **inputs**: An array of input objects with an id and a value property. This array populates the fields with data.

### Result

The final HTML the "Name" value should be as follows:

<p class="info-item" id="name">Name: <span class="value">Adventure Backpack</span></p>

### **Constraints**

* The test code automatically tries to fire a "submit" event for each item in the inputs array.
* Do not attempt to change the HTML of the reference document.

### Want a hint?

Learn about creating new DOM elements in [this course](https://www.linkedin.com/learning/javascript-essential-training/) on LinkedIn Learning.

### Test code

// This is how your code will be called.

// Your answer should be the HTML output described in the challenge text.

// You can edit this code to try different testing cases.

const inputs = [

    { id: "name-input", value: "Adventure Backpack" },

    { id: "description-input", value: "A durable backpack for outdoor adventures" },

    { id: "color-input", value: "Green" },

    { id: "volume-input", value: "35L" },

];

### 11. Solution - Create an event listener

### Chapter Quiz

Question 1 of 8

An event listener can be appended to the window object.

TRUE

Correct

Event listeners are often used to listen for when the document is fully loaded in the browser. For this and other window events, we append an event listener to the window object.

FALSE

Incorrect

Event listeners are often used to listen for when the document is fully loaded in the browser. For this and other window events, we append an event listener to the window object.

Question 2 of 8

What goes in place of a and b in this standard event listener?

element.addEventListener("\_\_a\_\_", "\_\_b\_\_");

a: the event name

b: the function to call when the event fires

Correct

a: the actions to perform when the event fires

b: the name of the event

Incorrect

a: the count of how many such event listeners are defined

b: an array of functions to call when the event fires

Incorrect

a: the function to call when the event fires

b: the object ID of the element

Incorrect

Question 3 of 8

What does 'this' refer to in the following event listener definition?

myElement.addEventListener(eventName, function(e) {

eventFunction(e,this);

});

The event itself

The event in eventName

Incorrect

The eventFunction() function

Incorrect

The element in myElement

Correct

Question 4 of 8

Why does this return the window object in the following event listener definition?

myElement.addEventListener(eventName, () => {

console.log(this);

});

Arrow functions do not have their own this, so they refer to the closest defined this which is the window object.

Correct

myElement is not properly defined.

Arrow functions need to be explicitly told what this refers to using the setThis() method.

Incorrect

The arrow function is not capturing the event object as a parameter.

Incorrect

Question 5 of 8

Can a value be passed through an event listener to its callback function?

Yes, by capturing the value as a parameter in the callback function.

Incorrect

Yes, by setting the value inside the callback function.

Incorrect

Yes, by capturing the value in a new function inside the callback function.

This was the correct answer

No, this is a limitation of event listeners.

Incorrect

Question 6 of 8

When is the resize event triggered?

When the document view is resized.

Correct

When the browser zoom ratio is changed.

When the specified element is resized.

When the browser window is resized.

Question 7 of 8

How do you capture the event object in an event listener?

The event object is captured in a variable with the name "e" or "event". If one of these names are used in the callback function, it refer to the event.

The event object is captured in the third parameter of the addEventListener() method.

Incorrect

The event object is captured using the catchEventObject() method.

Incorrect

The event object is automatically passed as a parameter to the callback function. Simply name and use the parameter.

Correct

Question 8 of 8

If you create several event listeners listening to the same event, only the last one in the script will work.

TRUE

Incorrect

FALSE

Correct

## 10. Troubleshooting and Validating JS

### 1. Troubleshooting JavaScript in the browser

### 2. Making sense of a React component

### Chapter Quiz

Question 1 of 2

What is the first troubleshooting step when your JavaScript is not working as expected?

Delete the last change you made and try the code again.

Incorrect

Search for an answer on the web.

Incorrect

Look for errors in the browser console.

Correct

Look for errors in the code editor.

Question 2 of 2

What is the keyboard shortcut to comment out one or more highlighted lines of code in the code editor?

On Windows: Ctrl+C

On Mac: Cmd+C

On Windows: Windows + .

On Mac: Ctrl+Cmd+Space

Incorrect

On Windows: Ctrl+/

On Mac: Cmd+/

Correct

On Windows: Ctrl+Shift+/

On Mac: Cmd+Shift+/

### 0. 11. Next steps on your learning journey

## 11. Next steps on your learning journey

### Chapter Quiz

Question 1 of 1

Java is to JavaScript as \_\_\_\_\_.

metal is to Metallica

Incorrect

car is to cartography

Incorrect

gas is to gastronomy

ham is to hamster

Correct