Midterm Exam #3 Review Guide

Curated by: Rishabh Baral
Professor: Nelson Padua-Perez
Course: CMSC335 - Web Application Development with JavaScript
Exam Date: Nov 29, 2023, 5:00 PM

Disclaimer: This guide was made by me, a student. I am not a TA or a professor. If there are any errors in this document they are entirely my own. Please do not take this guide at its total face value unless a TA reviews it. If there are any errors, please contact me either by email at rbaral@terpmail.umd.edu or via any channel of communication used for the course (Canvas, Piazza, and GroupMe).

Further Reference: For further information, please post a question on Piazza or attend office hours at the times presented at https://www.cs.umd.edu/class/fall2023/cmsc335/officeHours.shtml**.**

Contents

1. How to write the HTML that goes in the <body></body> section. You don't need to know the tags associated with the header (only the <title> and <link> tags). 6
   a. <title> tag 6
   b. <link> tag 6
2. You are only responsible for HTML tags that allow us to: 6
   a. Define tables. 6
   b. Define ordered and unordered lists. 7
   c. Define forms using text fields, buttons, reset, and submit buttons. You don't need to know any other form elements (e.g., radio buttons, etc.). 7
   d. Other tags you should be familiar with: 8
   i. <br> 8
   ii. <strong></strong> 8
   iii. <em></em> 8
   iv. <h1></h1> 8
   v. <div></div> 8
   vi. <span></span> 8
   vii. <p></p> 9
   viii. <pre></pre> 9
   ix. <img> 9
   x. <a> 9
3. You are only responsible for the following CSS properties: 10
a. Color 10
b. font-size 10
c. background-color 10
4. You are only responsible for the following CSS selectors: 10
a. type (e.g., h1, h2, p, table) selectors 10
b. class selectors 10
c. id selectors 10
d. descendant selectors 11
e. pseudo-classes for link (a:link, a:visited, a:hover, a:active) 11
f. :hover pseudo-class 11

5. You should know how to write HTML that corresponds to a provided image. For example, we can provide an image with a table and a list, and you are expected to provide the appropriate HTML. 11

6. JavaScript 11

a. Comments 11

b. Variable declarations using let, const, var 11

c. Primitive types 12

d. Reference types 12

e. Type conversions using Number() 12

f. Reading data using prompt() 13

g. Display results using document.writeln()/write() and alert() 13

h. Mathematical, logical and relational expressions 13

i. Conditional statements 13

j. Loop constructs - while, do-while, for loops 13

k. break, switch statements 14

l. "use strict"; 14

m. Only console.log 14

n. isNaN() 15

o. window object 15

p. Function definition 15

q. One-dimensional array 15

r. String methods - Only includes, trim, indexOf, toLowerCase, toUpperCase, charAt, split, localeCompare 16

s. for...of 16

t. for...in 16

u. Null 16

v. Undefined 16

w. NaN 16

x. Math.sqrt() 17

y. Math.random() 17

z. Two-Dimensional Arrays 17

aa. Template literals 17

bb. Sorting 17

cc. Default Parameters 17

dd. Rest Operator 18
ee. Spread Operator 18
ff. Destructuring 18
7. JavaScript Basics: 18
a. What is ECMAScript 18
b. What is a JavaScript engine 18
c. How to include JavaScript in an HTML file 18
d. What is the DOM 19
e. Function execution context 19
f. JavaScript Sets and Maps 19
g. Freezing and Sealing Objects 20
h. Operators 21
8. Set and Clear Interval 21
a. Set Interval 21
b. Clear Interval 21
c. Syntax 21
9. JSON 22
a. JSON Functions 22
b. How to write JSON based on string description 22
10. Array Methods 23
11. Defining classes using **class, extends, super** 23
12. Server-Related Information 24
a. HTTP post and get 24
b. What is localhost and 127.0.0.1 24
13. Event-Driven Programming 24
a. What is Event-Driven Programming? 24
b. On Click Event 24
c. Associating a function with an event 25
14. DOM-related Functions 25
15. Retrieving values from a text field 25
16. Modifying HTML and Attributes 26
a. GetAttribute() 26
b. SetAttribute() 26

c. InnerHTML 26
17. Loading Javascript from file 26
18. What is Ajax?..26
19. What is Axios? 26
20. HTTP Requests 26
21. CORS 27
22. LocalStorage..27
23. Node.js 27
a. Main Initialization Syntax 27
b. Files that any Node Project should have..27
24. Using window.onsubmit to control Form Data 27
25. Event-Loop 28
26. Promises...28
27. Async/Await 28
28. Using Fetch to retrieve JSON Data 28
29. Modules 29
30. User-Defined Errors using Error...29
31. Cookies...29
32. Sessions 30
33. Express...30

1. How to write the HTML that goes in the <body></body> section. You don't need to know the tags associated with the header (only the <title> and <link> tags).

a. <title> tag

This tag serves no purpose with respect to content found in the body of the HTML file. However, the <title> tag is found in the head of an HTML document. This placement of the <title> tag indicates that it serves some purpose in the head of the document, which it does. The <title> tag indicates to a web browser (or Live Server) what title to display for the associated HTML page. In fact, the purpose that <title> serves for an HTML page is so important that it is actually a **required** tag when writing any HTML file.

b. <link> tag

The <link> tag, like the <title> tag, is found in the head of an HTML document. The <link> tag can serve many purposes, but the most common one is to serve as a reference parameter to any external resource files that the HTML file itself is dependent on. Just like the <title> tag, the <link> tag serves no purpose to the body of an HTML document but serves a purpose for the HTML document in relation to how it is rendered as the <link> tag most often defines the style via link to external stylesheet(s).

2. You are only responsible for HTML tags that allow us to:

a. Define tables.

Table **Header**: <thead></thead>
    Attributes:
1. **row**span→ number of rows to merge
2. **col**span → number of columns to merge

Table **Heading:** <th></th>
Table Row: <tr></tr>
    Attributes: **Row**span → number of rows to merge
Table Data (Element): <td></td>
    Attributes:
3. **row**span→ number of rows to merge
4. **col**span → number of columns to merge

Table **Footer:** <tfoot></tfoot>
    Attributes:
1. **row**span→ number of rows to merge
2. **col**span → number of columns to merge

b. Define ordered and unordered lists.

Ordered List: <ol></ol>
Ordered List Types:
1. type = ‘1’ → numbered list
2. type = ‘i’ → Numbered List, Roman Numerals
3. type = 'a’ → List with numbers as letters (a = 1, b=2, c=3, etc.)

Unordered List: <ul></ul>
List Item: <li></li>

c. Define forms using text fields, buttons, reset, and submit buttons. You don't need to know any other form elements (e.g., radio buttons, etc.).

Define a form: <form></form>
Text fields: <input></input>
Text Field Types:
1. **Button**
2. **Checkbox**
3. **Color**
4. **Date**
5. **Datetime Local (Local Date/Time)**
6. **Email**
7. **File**
8. **Hidden**
9. **Image**
10. **Month**
11. **Number**
12. **Password**
13. **Radio**
14. **Range**
15. **Search**
16. **tel (telephone number)**
17. **text**
18. **time**
19. **url**
20. **week**

Reset Button: use <input></input> with type = ‘reset’
Submit Button: use <input></input> with type = ‘submit’

d. Other tags you should be familiar with:

i. <br>

The <br> tag is intended to add a line break between the elements immediately preceding and following the <br> tag. This tag is formerly the </br> tag, but the self-enclosing <br> tag is the current version. Please use <br>.

ii. <strong></strong>

The <strong> tag is intended to **bold** text. The <strong> tag is applicable anywhere, but the **boldface** applies to only whatever appears in between the <strong> and </strong> tags.

iii. <em></em>

The <em> tag is intended to *italicize* text. This tag is applicable anywhere and is often used in combination with the <strong> tag to produce **bold, italicized** text. Analogously to the <strong> tag, the effect of *italicization* applies only to whatever appears in between the <em> and </em> tags.

iv. <h1></h1>

The <h1> tag is the **strongest** of a set of six similar tags (h1, h2, h3, h4, h5, h6) that all serve the same purpose. The <h1> tag is used to indicate important headings, such as those for sections or paragraphs within a document. It is, by default, 32px **bold** font. Usually, the h1 tag is the first heading tag used and often precedes tags such as <p>, <pre>, and <article>; but can precede many others.

v. <div></div>

The <div> tag is a key tag when it comes to organization within an HTML document. Often, the <div> tag is used to split sections within an HTML document and is often styled to look like a box using CSS (to be discussed later in this document). The <div> tag does have attributes, but the only ones that are ever used in practice are **name** and **id**. In most uses of the <div> tag, the <div> tag itself will contain other tags within it, such as <p>, <pre>, and <article>.

vi. <span></span>

The <span> tag serves an identical purpose to the <div> tag, serving as a container. Often the <span> tag can take the place of a <div> tag except that the <span> tag displays the container created as an **inline** element as opposed to the **block** display style of the <div> tag. Nowadays, the <div> tag has fallen into disuse because of its block display style in favor of the <span> tag.

vii. <p></p>

The <p> tag is one of the most basic HTML tags. The purpose of this tag is to set aside space to write a paragraph in the document. The <p> tag is one of the most used tags but is slowly being phased out in favor of other more robust tags such as <pre>. This is because any text that is placed in a <p> tag is done so in a linear manner, meaning that spacing between lines is neither preserved nor respected.

viii. <pre></pre>

The <pre> tag is the most used alternative to the <p> tag. The usage of this tag is mostly the same as the <p> tag in that the <pre> tag is used to write paragraphs in an HTML document, but the one difference that makes the <pre> tag more desirable as compared to the <p> tag is that the <pre> tag does respect spacing within the text input to the <pre> tag.

ix. <img>

The <img> tag is one of three special tags used for media inclusion in HTML files. As the abbreviated name suggests, the <img> tag is used to include images into an HTML file. Often, images that are included in HTML files are elsewhere in the directory or even online. As such, some of the commonly used attributes of the <img> tag are:

1. src →image source
2. width → image width
3. height → image height
4. alt → Alternate Text to display in the event that the image cannot be rendered.

x. <a>

The <a> tag is a tag that is used for more than one purpose. However, the most common use of the <a> tag is to include links using the ‘href’ keyword within the tag. Just like the image tag, the <a> tag has many attributes, but the most used ones are:

- href → link source (URL)
- alt → text to display when link cannot render

Unlike the <img> tag however, the <a> tag does have a closing tag </a> and usually it is the text appearing between the <a> and </a> tags that is seen on the screen as the “display value” for a particular link.

3. You are only responsible for the following CSS properties:

a. Color

The color property in any CSS document specifies the color of the text for the element that the color property is applied to.

b. font-size

The font-size property in any CSS document specifies the font size of the text for the element that the font-size property is applied to.

c. background-color

The backround-color property in any CSS document specifies the background-color of the element to which the background-color property is applied.

4. You are only responsible for the following CSS selectors:

a. type (e.g., h1, h2, p, table) selectors

These are the most basic type of selector. These are sets of CSS rules that are applied to a document, but only to elements of a specific type (specified by the tag prior to the declaration). This type of selector can be created with the following syntax: **<tag>{CSS}**

b. class selectors

These are selectors that allow us to apply a set of CSS rules across various different elements without having to redeclare the set of rules individually. These are created with the following syntax: **.[name_of_class]{CSS}**

c. id selectors

These are selectors that allow us to define a set of rules and then use it in multiple places just as class selectors do. However, Class selectors can appear multiple times in the same CSS document, but ID selectors appear only once. Usually they are used if a style only needs to be applied once or needs to be applied in a particular manner to a particular ID. This kind of selector can be created with the following syntax: **#[ID]{CSS}**

d. descendant selectors

These selectors are designed to specify styles for elements within other elements (paragraph in a div, link in a footer, etc.). These are designed to override the default type, class, and ID selector styles that have already been defined. These selectors can be created as follows:
<tag> <subtag>{CSS}

e. pseudo-classes for link (a:link, a:visited, a:hover, a:active)

A pseudo class selector is one that serves as a "phantom class" of sorts. It allows style to be assigned to any element not based on the type of element, but rather based on the state of the element. The 'link' pseudo classes are designed to be able to apply a style based on if a link is in one of the 4 common states: unclicked (link), clicked (visited), checking (hover), open (active)

f. :hover pseudo-class

The hover pseudo-class is applicable to any element. It serves the purpose of allowing us to define style if the element in question is being hovered over. Often we see this pseudo-class associated with the <a> tag as links are most often hovered over, but this can be used for various operations on text content as well.

5. You should know how to write HTML that corresponds to a provided image. For example, we can provide an image with a table and a list, and you are expected to provide the appropriate HTML.

For this, please keep one tip in mind: **The HTML is far more important than the CSS. Try to get the HTML first and figure out the CSS later. For the CSS, colors do not likely need to be exact unless mentioned in the problem itself.**

6. JavaScript

a. Comments

Single Line: //	Multiline: /* */

b. Variable declarations using let, const, var

Let	Const	Var
This declaration is used for declaring variables.	This declaration is used for declaring constants	This is deprecated, please **avoid.**

Things to know about declaring variables in JS:

1. There is no type
2. Variables must begin with:
    a. Letter
    b. Underscore
    c. Dollar Sign
3. Variables can only contain:
    a. Letters
    b. Numbers
    c. Underscores
    d. Dollar Signs

c. Primitive types

Primitive types are **static** data types that are not objects and they also have no methods. Furthermore, **all primitive types are immutable.**
There are 7 primitive types in JavaScript:

1. Null
2. Boolean
3. Number
4. String
5. Undefined
6. Bigint
7. Symbol

d. Reference types

Reference Types are those that represent addresses of objects. All reference types in JavaScript refer to Objects, which are a collection of properties.

e. Type conversions using Number()

This conversion is used to go from String to Number. It is much stricter than parseInt or parseFloat because the input to Number must be a number and nothing else.
Here are the three conversions:

```
let number = Number(stringValue)
let number = parseInt(stringValue)
let number = parseFloat(stringValue)
```

f. Reading data using prompt()

The prompt() function is one that displays a dialog box, often with a specified title. It can be used to read any data and the default, should nothing be entered, can be specified after the title of the aforementioned prompt.

g. Display results using document.writeln()/write() and alert()

Document.writeln()	Document.write()	Window.alert()
Generates text output, but **automatically adds new line at end of output**	Generates text output, but **does not** add new line at the end of the output	Generates text output, does not add new line, but **displays in window** as opposed to on the document.

h. Mathematical, logical and relational expressions

Relational	Mathematical	Logical
=== → true if equal, false otherwise	+ → Addition	! → not
!== → true if different, false otherwise	- → Subtraction	&& → and
< → less than	* → Multiplication	|| → or
> → greater than	/ → Division	?: → Ternary
<= → Less than or Equal To	** → Exponentiation	++ → Increment
>= → Greater than or Equal to	% → Modulus (remainder)	-- → Decrement

Notes:

1. == checks for value only, === checks for value and type
2. != checks for value only, !== checks for value and type

i. Conditional statements

Just as with any other language, JavaScript has conditional statements. These are often found in conjunction with Control Structures and are often used to provide a stopping condition. Conditional statements are designed to serve a particular purpose if the condition is true, and a different purpose should the condition be false.

j. Loop constructs - while, do-while, for loops

While	Do-while	For

`while (expression 1) {`
`  // code block to be executed`
`}`
	`do{`
	`  // code block to be executed`
	`} while (expression 1)`
		`for (expression 1; expression 2; expression 3) {`
		`  // code block to be executed`
		`}`

Expression 1: Conditional The loop will run until this becomes false	Expression 1: Conditional The loop will run until this becomes false	Expression 1: initialization The variable for the loop is declared here Expression 2: Conditional The loop will run until this becomes false Expression 3: Increment/Decrement The loop will modify the variable by +1/-1 each loop

k. break, switch statements

Break statements allow us to stop execution of a loop immediately when a condition becomes true. These are often used in conjunction with continue statements, which help to skip certain iterations of a loop where a condition is met.

Switch statements allow us to perform a specific task based on the value of an input received. Usually there are various "cases" and one "default" in the event that the input does not match any one case.

```
switch(expression) {
  case x:
    // code block
    break;
  case x:
    // code block
    break;
  default:
    // code block
}
```

l. "use strict";

1. Allows users to opt-in to a strict mode of JS in which
    a. "Silent" errors (warnings in standard JS) are thrown
    b. Prohibits some common syntax that is allowed in more "loose" JS

m. Only console.log

Shows output of functions in JavaScript to the console of the web browser running the file. console.log is used to show general messages to the user whilst different versions of the console object are used to show different messages.

n. isNaN()

The isNan() method does what it suggests. It checks whether whatever is passed as a parameter to the method is or isn't a number. If the value isn't a number, it returns true and returns false if the value is a number.

o. window object

1. Represents the window in which a page is displayed
2. Provides access to other global objects, namely the document being displayed
3. Keeps track of the user's global variables
4. Allows JS to access the API of the browser on which it is running

p. Function definition

Functions are:

1. Invoked using the () operator
2. **Do not use var, let,** or **const** for parameters
3. There is **no mandatory main function**
4. Return values using the **return** keyword
5. Recursion is supported.

Functions can be declared:

1. In the standard manner using the **function** keyword and the **()** operator
2. As a function expression (**anonymous** function)
3. Using the **function** constructor
4. Using the **arrow** method (**lambda expressions**)

q. One-dimensional array

An array is a special collection of objects that can either be treated as a unit or individually. To access elements of an array, use bracket indexing (**[ ]**).
Arrays can be declared:

1. By using the literal form → [<elements>]
2. By using the array constructor → new Array()

r. String methods - Only includes, trim, indexOf, toLowerCase, toUpperCase, charAt, split, localeCompare

Methods from the String Class
includes → Boolean method to check if a character is present in a string
trim →remove any whitespace from both sides of a string
toLowerCase→convert string to be entirely in lowercase
toUpperCase→convert string to be entirely in uppercase
charAt→ Character at a particular index within a string
split→convert a string to an array of characters
localeCompare→compares two strings, in the current locale, and returns their sorted order

s. for...of

for…of is used to create a loop that iterates over iterable objects such as strings, arrays, array-like objects, and other iterable objects.

t. for...in

for…in is used to iterate over the properties of an object.

u. Null

In JavaScript, Null is one of the seven primitive data types. In practice, null is used to represent the absence of some data.

v. Undefined

In JavaScript, undefined and null are different in that though both are primitive data types, Null solely indicates the absence of some data whereas undefined indicates that the data being looked for either was never defined or was never given a value.

w. NaN

NaN (Not a Number) can be generated when:

1. Arithmetic Computation results in an undefined or unrepresentable value
2. Coercion of non-numeric values to numeric values occurs with no predefined numeric equivalent of the value being coerced.

x. Math.sqrt()

Returns the square root of the numerical value provided as the parameter

y. Math.random()

This method returns a random number between 0 (inclusive) and 1 (exclusive). The range can be modified through multiplication (change the end) and addition (change the start).

z. Two-Dimensional Arrays

aa. Template literals

Template Literals:

1. Are String Literals that allow embedded expressions
2. Can replace placeholders in text with values of variables and expressions
3. Are defined using the backtick (`) character
    a. To find the backtick, look underneath the tilde (~) key on the keyboard at the left end of the number row.
    b. To escape a backtick (`) within a template literal, precede the backtick with a backslash (\)
        i. The backslash is nestled between the backspace/delete and enter/return keys on the keyboard

bb. Sorting

In JavaScript, sorting is done using the Array sort() function. However, this method of sorting is flawed since it sorts inherently using strings. For this reason, sorting can be tricky when using numbers. Inherently, “2” is bigger than “1”, meaning that “25” is bigger than “100” as a string, but we know that it isn’t based on basic logic. For this reason, we can provide a comparison function to streamline sorting and make sure that we are sorting correctly.

cc. Default Parameters

In a function, the definition contains a list of parameters. If the function is then called with less than the number of parameters it was declared with, the missing parameters are filled in with undefined as per ECMAScript. If, instead, parameters are defined with a default value, then in a function call, the missing parameter or parameters are replaced with the default value instead of undefined.

dd. Rest Operator

The rest operator (…) is an improved way to handle function parameters. It allows us to circumvent the issues of calling functions with either too many or too few parameters by creating an indefinite number of arguments, whereby the function can now be called with any number of parameters.

ee. Spread Operator

The spread operator (…) appears to be identical to the rest operator, but **do not confuse the two.** The spread operator allows us to copy all or some of an array from one object to another. The spread operator is often used in combination with destructuring of objects.

ff. Destructuring

Think of destructuring kind of like making a sandwich. When you make a sandwich, you don't take out the entire loaf of bread or the entire tray of lunchmeat, you only take out the amount you need. In an analogous manner, destructuring does the same for arrays, whereby we extract certain elements from an array in order to use them.

7. JavaScript Basics:

a. What is ECMAScript

ECMAScript is the style guide for Scripting Languages. It is produced by ECMA international and is the specifications and set of rules that a scripting language must observe to be considered ECMAScript compliant

b. What is a JavaScript engine

JavaScript is a general scripting language that is considered to be ECMAScript compliant. The JavaScript Engine is the utility that processes JavaScript code in different browsers.
This is extra information, but I am including a table of the browsers and their JS Engines below.

Browser	**JavaScript Engine**
Safari	JavaScriptCore
Chrome	V8
Firefox	Spidermonkey
Edge	Chakra

c. How to include JavaScript in an HTML file

Internal: Use the <script> tag for any JS code	External: Use the <link> tag to link to a .js file

d. What is the DOM

The DOM is the **D**ocument **O**bject **M**odel. It is designed to represent a document as a tree where each node is an object that represents a part of the document. Every branch of the DOM tree ends in a node and each of these nodes contain objects. The DOM model also allows us "programmatic access" to the tree, allowing us to change the structure, style, or content of a document. Every node (portion of a document) has an event handler associated with it, and these handlers are executed if an event is triggered.

e. Function execution context

Just as was seen in Java (remember the long red stack trace from eclipse?), JS functions also generate frames on the "stack". Every function call creates what is known as a "**function execution context"** which is just a fancy way of saying **"stack frame".**

f. JavaScript Sets and Maps

Maps are sets of keyed values, just like an object. The only difference is that the keys in an object can only be of the object type, but a map is different in that the keys within a map can be of any type. Common methods include set (assign a value to a key), get (retrieve the value associated with a key), has (check if a map contains a specific key), delete (remove a key-value pair from the map), clear (empty the map), and size (get the number of elements in the map).

Sets are a specific type of map where there are only values, but the values can only appear once in the set. Some common methods include add (add a value to the set), delete (remove a value from the set), has (check if the set contains a value), clear (empty the set), and size (check the number of elements in the set). The advantage of a set is that a set can replace an array and can provide better performance in situations where verification of duplicates would require a full pass through the array since that would be considerably slower.

g. Freezing and Sealing Objects

Freezing an Object	Sealing an Object	Preventing Objects from being Extendable
Freezing an object does not allow new properties to be added to the object and prevents removing or altering the existing properties. Object.freeze() preserves the enumerability, configurability, writability, and prototype of the object. It returns the passed object and does not create a frozen copy.	An object is sealed if all of the below-mentioned conditions hold true : 1. If it is not extensible. 2. If all of its properties are non-configurable. 3. If it is not removable (but not necessarily non-writable).	The Object.preventExtensions() static method prevents new properties from ever being added to an object (i.e. prevents future extensions to the object). It also prevents the object's prototype from being re-assigned.

h. Operators

Chaining Operator (?.)	Logical And (&&)	Logical Or (||)	Coalescing Operator (??)
This is an operator designed based on short-circuiting. If the property for which it is called with regards to the object used is **null** or **undefined,** this operator returns **undefined.**	L | R | L&&R 1 | 1 | 1 1 | 0 | 0 0 | 1 | 0 0 | 0 | 0	L | R | L||R 1 | 1 | 1 1 | 0 | 1 0 | 1 | 1 0 | 0 | 0	This is an operator based on the value **null**. If the left hand side of this operator is **null** or **undefined,** then the right-hand side value is returned.

8. Set and Clear Interval

a. Set Interval

The setInterval() method, which is available with respect to functions and global objects such as the window, repeatedly executes a function with a fixed time delay between calls of the function.

b. Clear Interval

This is the opposite of the setInterval() method. In essence, it stops the timed repetition that was started by setInterval().

c. Syntax

```
function changeColor() {
  // check if an interval has already been set up
  if (!nIntervId) {
    nIntervId = setInterval(flashText, 1000);
  }
}
function stopTextColor() {
  clearInterval(nIntervId);
  // release our intervalID from the variable
  nIntervId = null;
}
```

9. JSON

a. JSON Functions

JSON.parse()	JSON.stringify()
JSON parse is a **static** method that parses a JSON string, constructing the Javascript object as described.	This does what JSON parse does, but in the reverse direction. JSON stringify essentially takes a Javascript value and creates a JSON string from it.

```javascript
const json = '{"result":true, "count":42}';
const obj = JSON.parse(json);

console.log(obj.count);
// Expected output: 42

console.log(obj.result);
// Expected output: true

```

```javascript
console.log(JSON.stringify({ x: 5, y: 6 }));
// Expected output: '{"x":5,"y":6}'

console.log(JSON.stringify([new Number(3), new String('false'), new Boolean(false)]));
// Expected output: '[3,"false",false]'

console.log(JSON.stringify({ x: [10, undefined, function () {}, Symbol('')] }));
// Expected output: '{"x":[10,null,null,null]}'

console.log(JSON.stringify(new Date(2006, 0, 2, 15, 4, 5)));
// Expected output: '"2006-01-02T15:04:05.000Z"'

```

b. How to write JSON based on string description

1. Array of object →[{…}]
2. String→“…”
3. Integers/numbers are written exactly as they would
4. Append a colon (:) after any variable declaration within the JSON string

10. Array Methods

Name of Method	Description of Method and Function
Join	Combines the elements of the array into a single string
Pop	Removes and returns the last element of the array
Push	Add the specified element or elements to the **end** of the specified array
Reverse	Reverses the order of the elements in an array, **in place**, and then returns a reference to the **same** array.
forEach	Executes a function for **every element** of a particular array
find	Returns the **first** instance of a provided element in an array.
findIndex	Returns the **index** of the element that would be returned by a call to Array.find()
Map	Creates a **new** array populated with the result of applying a function to every element of an array.
Every	Tests **all** elements of an array to see whether they satisfy a function.
Some	Tests to see if **at least one** element of an array satisfies a function.
reduce	The reduce() method of Array instances executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The result of running the reducer across all elements of the array is a single value.

11. Defining classes using **class, extends, super**.

This is the standard way of defining classes as seen in java. The only difference with regards to JavaScript is that the syntax differs from java, but the idea is essentially the same.

```
// Declaration
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

12. Server-Related Information

a. HTTP post and get

	Post Request	Get Request
Form Information	Form information is in the **message body**	Form information is provided in the **URL** itself
Bookmarked	A HTTP Post **cannot** be bookmarked	An HTTP Get **can** be bookmarked.
Limit of Data	There is **no data limit** since a post request isn't associated with a URL.	There is a **data limit** because Get requests modify the URL, meaning that **the URL size is the limit of the data**.
Security	Better at security since information is **hidden**.	Poor for security since the information is **visible in the URL.**
Example	Issues **warning** prior to execution	Intended for search operations that **do not change the server's state.**

b. What is localhost and 127.0.0.1

Localhost and 127.0.0.1 are both the same thing. In the context of XAMPP and web servers, localhost and 127.0.0.1 both represent the standard server address for the **local machine**. When a web server is started, the address of the server, should one wish to access it, is either localhost/ or 127.0.0.1/ depending on which one is working at the time.
Advice: If one isn't working, try the other. If neither is working, check that the server application (XAMPP→Apache) is running.

13. Event-Driven Programming

a. What is Event-Driven Programming?

In computer programming, event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions from mice, keyboards, touchpads, and touchscreens.

b. On Click Event

The onclick event is just as the name suggests. This event listener, if present, is triggered when the user clicks on an html element such as a button or a field in a form.

c. Associating a function with an event

The addEventListener() method attaches an event handler to the specified element. It attaches an event handler to an element without overwriting existing event handlers. You can add many event handlers to one element including those of the same type to one element, i.e two "click" events. You can add event listeners to any DOM object not only HTML elements. i.e the window object. The addEventListener() method makes it easier to control how the event reacts to bubbling. When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

14. DOM-related Functions

Function Name	Function Purpose	Function Syntax
QuerySelector()	Returns the **first** document element that matches the selector or group of selectors.	`Document. querySelector(selectors)`
GetElementByID()	Returns the element within the document that matches the given ID.	`Document.getElementById(id)`

15. Retrieving values from a text field

```
document.getElementById('textbox_id').value
```

16. Modifying HTML and Attributes

a. GetAttribute()

This method, when called on an object, returns the value associated with the requested attribute for that object. If the requested attribute does not exist, the call to getAttribute() will return either ***null*** or ***“” (The Empty String)***.

b. SetAttribute()

Sets the value of an attribute on the specified element. If the attribute already exists, the value is updated; otherwise, a new attribute is added with the specified name and value.

c. InnerHTML

The InnerHTML property gets or sets the HTML or XML markup contained within the element upon which the InnerHTML query is performed.

17. Loading Javascript from file

Use the following line of code
(filename is the file or URL where the javascript is to be loaded from):

```
<script src = “filename”>
```

18. What is Ajax?

Ajax stands for **A**synchronous **J**avaScript **A**nd **X**ML. Asynchronous JavaScript and XML (AJAX) is a combination of web application development technologies that make web applications more responsive to user interaction.

19. What is Axios?

Axios is a JavaScript library used to make HTTP requests from node.js or XMLHttpRequests from the browser and it supports the Promise API that is native to JS ES6. It can be used intercept HTTP requests and responses and enables client-side protection against XSRF. It also has the ability to cancel requests.

20. HTTP Requests

Type	Ajax	Axios	Fetch

Syntax	`xhttp.open("GET", "ajax_info.txt", true);`
	`xhttp.send();`
		`axios.get`	`fetch()`

21. CORS

CORS is an important security feature that is designed to prevent JavaScript clients from accessing data from other domains without authorization. Modern web browsers implement CORS to block cross-domain JavaScript requests by default. CORS stands for **C**ross-**O**rigin **R**equest **S**haring, and refers to the use of a protocol to access restricted resources.

22. LocalStorage

The localStorage read-only property of the window interface allows you to access a Storage object for the Document's origin; the stored data is saved across browser sessions. localStorage is similar to sessionStorage, except that while localStorage data has no expiration time, sessionStorage data gets cleared when the page session ends — that is, when the page is closed. (localStorage data for a document loaded in a "private browsing" or "incognito" session is cleared when the last "private" tab is closed.)

23. Node.js

a. Main Initialization Syntax

To start a Node.js Project: `npm init`	To install necessary dependencies: `npm i`

b. Files that any Node Project should have

1. package.json → this is the file that lists the dependencies required for the project to function
2. node-modules folder: You can think of the node_modules folder like a cache for the external modules that your project depends upon. When you **npm install** them, they are downloaded from the web and copied into the node_modules folder and Node.js is trained to look for them there when you import them (without a specific path). I refer to it as a cache because the node_modules folder can be entirely recreated from scratch at any time by just reinstalling all the dependent modules (that should be listed in your project folders).

24. Using window.onsubmit to control Form Data

To trigger data validation before sending the request to the server, the submit event is utilized. If the user forgets to submit the required fields in the form, they can cancel the submission within the onsubmit event.

25. Event-Loop

The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded — by offloading operations to the system kernel whenever possible.

Since most modern kernels are multi-threaded, they can handle multiple operations executing in the background. When one of these operations completes, the kernel tells Node.js so that the appropriate callback may be added to the poll queue to eventually be executed. We'll explain this in further detail later in this topic.

26. Promises

A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future. A Promise is in one of these states:

1. pending: initial state, neither fulfilled nor rejected.
2. fulfilled: meaning that the operation was completed successfully.
3. rejected: meaning that the operation failed.

The eventual state of a pending promise can either be fulfilled with a value or rejected with a reason (error). When either of these options occur, the associated handlers queued up by a promise's then method are called. If the promise has already been fulfilled or rejected when a corresponding handler is attached, the handler will be called, so there is no race condition between an asynchronous operation completing and its handlers being attached.

A promise is said to be settled if it is either fulfilled or rejected, but not pending.

27. Async/Await

The async function declaration creates a binding of a new async function to a given name. The await keyword is permitted within the function body, enabling asynchronous, promise-based behavior to be written in a cleaner style and avoiding the need to explicitly configure promise chains.

28. Using Fetch to retrieve JSON Data

```
async function logMovies() {
  const response = await fetch("http://example.com/movies.json");
  const movies = await response.json();
  console.log(movies);
}
```

29. Modules

JavaScript modules allow you to break up your code into separate files. This makes it easier to maintain a code-base. Modules are imported from external files with the import statement. Modules also rely on type="module" in the <script> tag.

The export declaration is used to export values from a JavaScript module. Exported values can then be imported into other programs with the import declaration or dynamic import. The value of an imported binding is subject to change in the module that exports it — when a module updates the value of a binding that it exports, the update will be visible in its imported value.

30. User-Defined Errors using Error

```
//with new keyword
new Error()
new Error(message)
new Error(message, options)
new Error(message, fileName)
new Error(message, fileName, lineNumber)

//without new keyword
Error()
Error(message)
Error(message, options)
Error(message, fileName)
Error(message, fileName, lineNumber)
```

31. Cookies

Web Browsers and Servers use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website, it is required to maintain session information among different pages. For example, one user registration ends after completing many pages. But how to maintain users' session information across all the web pages. In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields:

1. Expires → the date on which the cookie will expire
2. Domain → the DNS address for the site that the cookie is for
3. Path → The path or directory address for the webpage that created the cookie
4. Secure → Was the cookie generated in **HTTP** or **HTTPS**
5. Name-Value → Cookies are set and retrieved as key-value pairs

32. Sessions

In the context of web development, a session refers to a way of maintaining state information about a user's interactions with a website or web application. When a user visits a website, the server can create a session for that user. Additionally, a session allows the server to keep track of information such as the user's login status, preferences, and any data entered into forms.

The server typically initiates a session when a user logs in to a website. Furthermore, we can identify a session by a unique session ID. Generally, we pass the session IDs as a parameter in URLs or store them in the cookies. The session ID allows the server to associate the user's requests with their specific session. Additionally, it also helps to retrieve and update the session data as needed.

A web session is a period of interaction between a user and a website. Furthermore, the website maintains state information about the user's actions and preferences during a session. The server can initiate a session for a user when they browse through a website. The session remains active until the user logs out.

33. Express

Know the functionality of each and every line of the code below!

```
app.get("/getHTML", (req, res) => {
  res.send("<h1>Returning HTML </h1>");
});

app.get("/getURLParameters", (req, res) => {
  res.send(`Received age: ${req.query.age} salary: ${req.query.salary}`);
});

app.get("/getJSON", (req, res) => {
  const student = { name: "Mary", age: 15 };
  res.json(student);
});

app.get("/getRender", (req, res) => {
  const variables = { semester: "Summer" };
  res.render("welcome", variables);
});

app.get("/getRouteParameters/:semester/CMSC:course", (req, res) => {
  res.send(
    `Received semester: ${req.params.semester}, course: ${req.params.course}`
  );
});


app.get("/getRedirect", (req, res) => {
  res.redirect("http://www.cs.umd.edu");
});

app.get("/getResourceNotFound", (req, res) => {
  const student = { name: "Mary", age: 15 };
  const http_page_file_not_found = 404;
  res.status(http_page_file_not_found);
  res.send("Cannot find resource (e.g., web page or file)");
});
```

```



app.post("/postSendingEmailAddress", (req, res) => {
  res.send(`<h2>Received via post email address ${req.body.email}</h2>`);
});

app.put("/putRequest", (req, res) => {
  res.send("<h2>Received put request</h2>");
});

app.delete("/deleteRequest", (req, res) => {
  res.send("<h2>Received delete request</h2>");
});

app.listen(portNumber);
console.log(`main URL http://localhost:${portNumber}/`);
```