Module 11 Day 2 Lesson 1

1. Forms and Form Validation

- Form: A form in web development is a user interface component that collects data from the user. They are primarily used for activities such as registration, login, data submission, etc.
- Form Validation: The process of ensuring that the user has provided valid information in the expected format before the data is sent to the server. Validation enhances user experience and protects your system from malicious or erroneous data.

2. Review of Form Elements and Their Events

Form Elements Include:

- `<input type="text">`: For single-line text input.
- `<textarea>`: For multi-line text input.
- `<input type="checkbox">`: For allowing the user to select zero or more options of a limited number of choices.
- `<input type="radio">`: For allowing the user to select exactly one of a predefined set of mutually exclusive options.
- `<select>`: Creates a drop-down list. `<option>` elements nested inside define the available options.
- `<button>`: Defines a clickable button.

Form Event Handlers Include:

- `input`: Triggers when the value of an input element changes.
- `change`: Similar to `input`, but doesn't trigger until the control loses focus.
- `focus`: Triggers when the input field gets focus.
- `blur`: Triggers when the input field loses focus.

3. The Form Object and Its Attributes

The Form object represents an HTML form element. It provides properties and methods to manipulate the form:

- `action`: Specifies where to send the form-data when a form is submitted.

- `method`: Specifies the HTTP method (GET/POST) to be used when submitting the form-data.
- `target`: Specifies where to display the response after submitting the form.
- `enctype`: Specifies how the form-data should be encoded when submitting it to the server.
- `autocomplete`: Specifies whether a form should have autocomplete on or off.
- `checkValidity()`: Returns true if an element's value satisfies its constraints; false otherwise.
- `validity`: Represents the validity state of an input element.
- `setCustomValidity()`: Sets the validationMessage property of an input element, or the element it is contained within.
- `validationMessage`: Holds the message that the browser will display when the validity is false.
- `valid`: Returns true if an input element contains data that matches its data type; false otherwise.
- `invalid`: Returns true if an input element contains data that does not match its data type; false otherwise.

4. HTML Form Validation Attributes

These are HTML attributes used for validation:

https://www.w3schools.com/html/html_form_attributes.asp

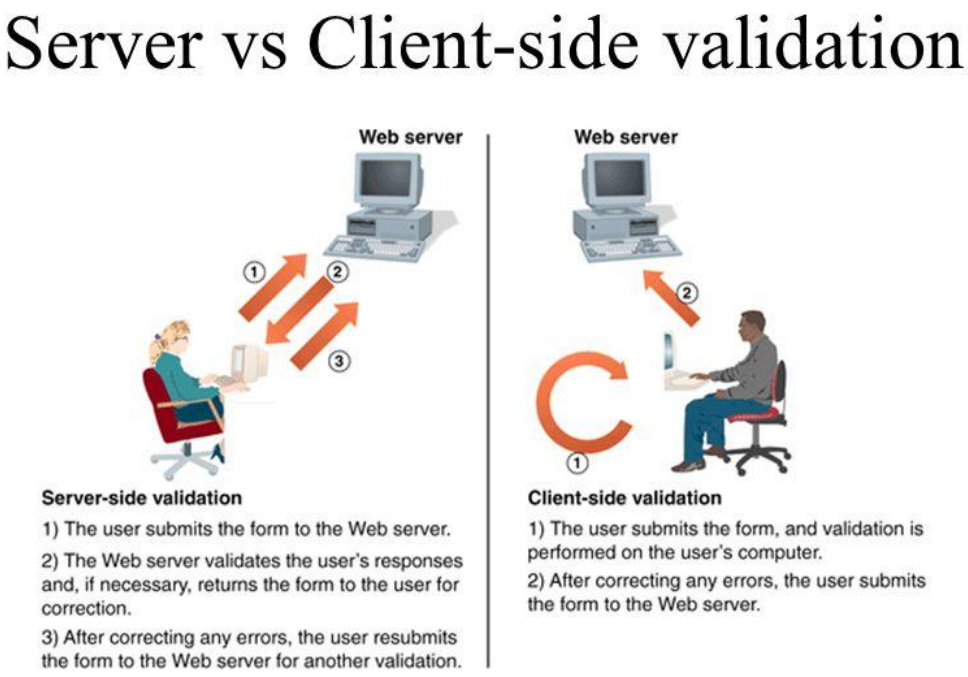
- `required`: Specifies that an input field must be filled out before submitting the form.
- `pattern`: Specifies a regular expression that an `<input>` element's value is checked against.
- `min` and `max`: Define the minimum and maximum values for an `<input>` element.
- `minlength` and `maxlength`: Define the minimum and maximum number of characters for an `<input>` element.

5. Client-Side Form Validation

Client-side form validation is performed in the user's browser before data is sent to a server. Validation can be done using HTML5 validation attributes, JavaScript-based validation for more complex requirements, or custom validation.

6. Difference Between Client-Side and Server-Side Validation

- Client-Side Validation: Provides immediate feedback to users and helps catch errors early. It should not be used alone as users can easily bypass it.
- Server-Side Validation: Takes place on the server, is more secure, and should always be performed because it can't be bypassed by users.
- Always use both client-side and server-side validation to ensure both user friendliness and data integrity. Client-side validation provides immediate feedback, while server-side validation ensures data integrity even if a user bypasses the client-side checks.



GET and POST Methods

- GET: Appends form-data to the URL in name/value pairs. It's best for non-sensitive data and when you want to bookmark the result.

- POST: Form-data is sent as HTTP message body. It's used when dealing with sensitive data or large amounts of data.

1. GET Method

- The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the `?` character.
- Data sent by GET method can be accessed using `$_GET` array in PHP.
- It's the default method for sending form data.
- GET requests can be cached, bookmarked, and remain in the browser history, and have length restrictions.
- GET is less secure compared to POST because data sent is part of the URL. It is not suitable for passing sensitive information like passwords.
- It is used when the interaction is idempotent, and when you want to retrieve data, not modify it.

2. POST Method

- The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.
- Data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- Data sent by POST method can be accessed using `$_POST` array in PHP.
- POST requests cannot be cached, bookmarked, do not remain in the browser history, and do not have length restrictions.
- It is used when you want to send sensitive, confidential, large amounts of data, or non-idempotent requests.

Storing and Processing Data

The data collected by HTML forms can be stored and processed in various ways, typically involving a server-side language like PHP, JavaScript (Node.js), Python, Ruby, Java, etc., and a database system like MySQL, PostgreSQL, MongoDB, etc.

1. Storing Data: Once the form data is validated, it can be stored in a database for future use. The server-side language will connect to the database, create an appropriate SQL query to insert the data, and execute it.

2. Processing Data: This could mean many things depending on the application. It might involve checking the data against existing data in the database (e.g., check if a username already exists), performing calculations (e.g., calculating cost based on input quantities and prices in a shopping app), or updating related data in the database (e.g., updating an 'last active' timestamp in a user profile).
3. Returning a Response: After the data has been processed, the server will typically send a response back to the client. This might be as simple as a confirmation message, or it could involve returning some data to the client (e.g., search results, updates).

The methods `GET` and `POST` in an HTML form merely dictate how the data is sent to the server, not how it's processed. Both `GET` and `POST` can lead to a new HTML page, but the data handling and processing happens server-side, which involves a backend language like PHP, Node.js, Python, etc.

HTML Forms and the GET/POST Methods:

- `GET`: When a form uses the `GET` method, the form data is appended to the URL as query parameters when the form is submitted. This can be useful when you want to allow users to bookmark a specific state of the form or when the form is idempotent (i.e., it doesn't cause a change on the server).
- `POST`: When a form uses the `POST` method, the form data is included in the body of the HTTP request when the form is submitted. This is typically used when the form data is expected to cause a change on the server (such as creating a new record in a database, updating a record, etc.), or when you're sending sensitive or a large amount of data.

Server-Side Processing:

Merely sending the form data to the server does not imply that it is processed. Processing the form data is a separate step that is handled by your server-side code:

- When the server receives a request containing form data, your server-side code needs to parse the form data from the request.
- Then, it needs to do something with that data—typically, this might involve writing the data to a database, reading from a database, updating a database, sending an email, etc.
- The server-side code may then generate a response to send back to the client. This could be a whole new HTML page, a redirect instruction, a small snippet of data, or a status code.

It's important to note that the `GET` and `POST` methods in an HTML form are completely separate from the HTTP GET and POST requests your server-side code might make when interacting with other servers or APIs. The HTTP methods used in an HTML form dictate how the form data is sent from the client to your server, while the HTTP methods used in server-side code dictate how your server interacts with other servers.

Resources

Module 11 Codepen Collection: https://codepen.io/collection/qOyyKJ

Httpbin: https://httpbin.org/

Forms and Form Validation

- MDN Web Docs: Your first HTML form

- MDN Web Docs: Form data validation

Form Elements and Their Events

- MDN Web Docs: HTML Form Elements

- W3Schools: HTML Events

The Form Object and Its Attributes

- MDN Web Docs: Working with forms

- W3Schools: HTML Form Attributes

HTML Form Validation Attributes

- MDN Web Docs: Constraint validation

Client-Side Form Validation

- MDN Web Docs: Client-side form validation

JavaScript Form Validation

JavaScript Form Validation

JavaScript Validation API

GET and POST Methods

- W3Schools: HTML form method Attribute

General Resources on HTML Forms

- MDN Web Docs: Web forms — Working with user data

- W3Schools: HTML Forms

Video Tutorials
- Learn HTML Forms In 25 Minutes
- JavaScript Form Validation

Module 11 Day 2 Lesson 2

Basic Data Structures: Introduction

Data structures are fundamental to programming and can be thought of as collections of data in a specific organization or structure. This allows efficient access and modification of the data. Different types of data structures include Arrays, Objects, Sets, and Maps, which are commonly used in web development.

Sets

A Set is a special type of data structure provided by ES6 (ECMAScript 2015) in JavaScript. A Set is a collection of unique values. Any type of value can be added to a Set: objects, primitive values, or both.

Creating a Set in JavaScript:

```
let mySet = new Set();
```

Adding values to a Set:

```
mySet.add(1); // Set { 1 }
mySet.add(2); // Set { 1, 2 }
mySet.add(1); // Set { 1, 2 } - no change, as 1 is already in the set
```

Maps

A Map is a data structure that pairs keys with values. Like Sets, Maps were also introduced in ES6. A Map allows keys of any type and maintains the insertion order.

Creating a Map in JavaScript:

```
let myMap = new Map();
```

Adding values to a Map:

```
myMap.set('name', 'Alice'); // Map { 'name' => 'Alice' }
myMap.set('age', 30); // Map { 'name' => 'Alice', 'age' => 30 }
```

Arrays

An Array is a linear data structure consisting of a collection of elements, identified by integer index. Arrays are one of the simplest and most commonly used data structures.

Creating an Array in JavaScript:

```
let myArray = [];
```

Adding values to an Array:

```
myArray.push(1); // [1]
myArray.push(2); // [1, 2]
```

Objects

An Object in JavaScript is an unordered collection of key-value pairs. If a map is a more general form of an array, then an object is a more general form of a map.

Creating an Object in JavaScript:

```
let myObject = {};
```

Adding values to an Object:

```
myObject['name'] = 'Alice'; // { 'name': 'Alice' }
myObject['age'] = 30; // { 'name': 'Alice', 'age': 30 }
```

Filtering Data Structures: Arrays

Filtering is a method provided by the array to check each element of the array and return a new array that fulfills the condition provided for the filter.

Filtering in an Array:

```
let numbers = [1, 2, 3, 4, 5];
let evenNumbers = numbers.filter(number => number % 2 === 0);
console.log(evenNumbers); // [2, 4]
```

Filtering Data Structures: Objects

Filtering objects can be achieved by turning the objects into an array, applying filter, then turning it back into an object.

Filtering in an Object:

```
let student = {
  name: 'Alice',
  age: 20,
  grade: 'A',
};

let filteredStudent = Object.keys(student)
.filter(key => key !== 'age')
.reduce((obj, key) => {
  obj[key] = student[key];
  return obj;
}, {});

console.log(filteredStudent); // { name: 'Alice', grade: 'A' }
```

Filtering Data Structures: Sets

Filtering in Sets involves converting the Set into an Array, applying the filter, and converting it back to a Set if necessary.

Filtering in a Set:

```
let mySet = new Set([1, 2, 3, 4, 5]);
mySet = new Set([...mySet].filter(x => x%2 === 0));
console.log(mySet); // Set {2, 4}
```

Filtering Data Structures: Maps

Filtering Maps is similar to Sets - convert the Map into an Array, apply the filter, then convert back to a Map.

Filtering in a Map:

```
let studentMap = new Map();
studentMap.set('name', 'Alice');
studentMap.set('age', 20);
studentMap.set('grade', 'A');

let filteredMap = new Map([...studentMap].filter(([k, v]) => k !== 'age'));
```

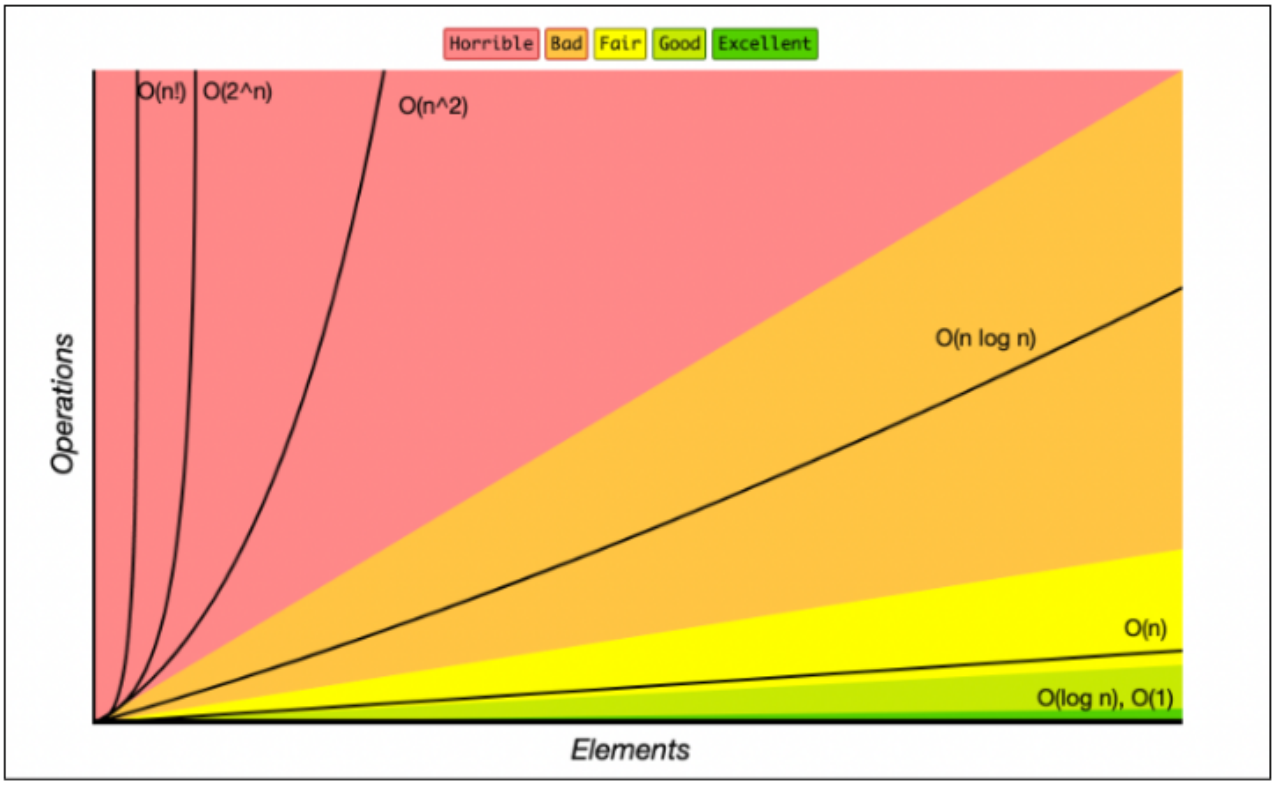
```
console.log(filteredMap); // Map { 'name' => 'Alice', 'grade' => 'A' }
```

A Primer on Big O Notation

Big O notation is used in computer science to describe the performance or complexity of an algorithm. Big O specifically describes the worst-case scenario, or the maximum amount of time taken by an algorithm. Here are a few examples:

1. O(1): Constant time complexity. No matter how much data, the operation takes the same amount of time.
2. O(n): Linear time complexity. The time taken grows linearly with the size of the data.
3. O(n^2): Quadratic time complexity. Time taken grows quadratically with the size of the data.

Understanding Big O notation and time complexity is essential for writing efficient code, particularly in applications dealing with large amounts of data.



https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/

Big O Notation Summary

Notation	Type	Examples	Description
O(1)	Constant	Hash table access	Remains constant regardless of the size of the data set
O(log n)	Logarithmic	Binary search of a sorted table	Increases by a constant. If n doubles, the time to perform increases by a constant, smaller than n amount
O(< n)	Sublinear	Search using parallel processing	Performs at less than linear and more than logarithmic levels
O(n)	Linear	Finding an item in an unsorted list	Increases in proportion to n. If n doubles, the time to perform doubles
O(n log(n))	n log(n)	Quicksort, Merge Sort	Increases at a multiple of a constant
$O(n^2)$	Quadratic	Bubble sort	Increases in proportion to the product of n*n
$O(c^n)$	Exponential	Travelling salesman problem solved using dynamic programming	Increases based on the exponent n of a constant c
O(n!)	Factorial	Travelling salesman problem solved using brute force	Increases in proportion to the product of all numbers included (e.g., 1*2*3*4...)

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)

Resources

Module 11 Codepen Collection: https://codepen.io/collection/qOyyKJ

Basic Data Structures: Introduction

- freeCodeCamp Basic Data Structures
- Data Structures - GeeksforGeeks
- JavaScript data types and data structures - Mozilla Developer Network

Sets

- Sets - Mozilla Developer Network

Maps

- Maps - Mozilla Developer Network

Arrays

- JavaScript Arrays - W3Schools
- JavaScript Array Guide - Mozilla Developer Network

Objects

- JavaScript Objects - W3Schools
- JavaScript Object Guide - Mozilla Developer Network

Filtering Data Structures: Arrays, Objects, Sets, Maps

- JavaScript Array filter() Method - W3Schools
- How to Filter an Object with JavaScript

A Primer on Big O Notation

- Big O Notation - Khan Academy
- Big O Notation in JavaScript - codeburst

Videos

- Big-O Notation in 100 Seconds
- Learn Big O Notation In 12 Minutes
- Memory, Cache Locality, and why Arrays are Fast (Data Structures and Opt…

Other Visuals

- Big O Sketch Notes
- Girlie Mac Sketch Notes

Module 11 Day 2 Lesson 3

Trivia Game Part 1

Main Concepts:

1. JSON Data: The data for the trivia game is stored in two JSON files: `questions.json` and `users.json`. JSON (JavaScript Object Notation) is a lightweight data-interchange format that's easy for humans to read and write and easy for machines to parse and generate.

2. Arrays: Arrays are used extensively in the trivia game to store multiple related values. For example, the `questionsKeysArray` stores the keys from the `questions` object, and the `usersValuesArray` stores the values from the `users` object. Array methods like `forEach` are used to iterate over arrays and perform actions on each element.

3. Objects: Objects are used to store structured data. In this trivia game, each question and each user is represented as an object with various properties.

4. Sets: Sets are used to store unique values. In the trivia game, a `Set` is used to store the 10 random questions for the game (`randomTen`) and to store the users (`gameUsers`).

5. Maps: Maps are used to store key-value pairs, similar to objects, but with some differences that make them better suited for certain tasks. In the trivia game, a `Map` is used to store the detailed results for the current user (`currentUserDetailedResults`) and to store all users' stats (`usersStats`).

6. Event Listeners: Event listeners are used to respond to user interactions like clicks and input changes. In the trivia game, event listeners are added to the "Next" buttons, the answer buttons, the username input field, and the "Play Again" button.

7. DOM Manipulation: The Document Object Model (DOM) is used to interact with the webpage. In the trivia game, DOM manipulation is used to display the

questions and answers, to show and hide elements, to update the score, and to move the sections.

8. Control Flow: The trivia game uses various control flow structures, including `if` statements, `while` loops, and `for` loops, to control the flow of the game.

9. Functions: Functions are used to encapsulate blocks of code that perform specific tasks. In the trivia game, there are functions for checking the validity of the username, loading the questions and answers, checking the user's answer, moving to the next section, and handling the end of the game.

Resources:

JavaScript Fundamentals:

- MDN Web Docs: JavaScript Guide
- Eloquent JavaScript

JavaScript Maps:

- MDN Web Docs: Map
- JavaScript.info: Map and Set

JavaScript Objects:

- MDN Web Docs: Working with objects
- JavaScript.info: Objects

JavaScript Sets:

- MDN Web Docs: Set
- JavaScript.info: Set

JavaScript Array Methods (like `forEach`, `map`, `filter`, `reduce`):

- MDN Web Docs: Array
- JavaScript.info: Array methods

JavaScript Event Listeners:

- MDN Web Docs: Introduction to events
- JavaScript.info: Introduction to browser events

JavaScript JSON:

- MDN Web Docs: Working with JSON
- JavaScript.info: JSON methods

JavaScript DOM Manipulation:

- MDN Web Docs: Document Object Model (DOM)
- JavaScript.info: Document

Replit Example:

Trivia Game with Comments