ENSF 381
Full Stack Web Development

Lecture 34:
Recap: HTML, CSS, and JavaScript

Ahmad Abdellatif, PhD

UNIVERSITY OF CALGARY

Final Exam

- **Date**: April 17th, 2025
- **Time**: 5:00 PM – 7:30 PM
- **Location**: Aux GYM

Final Exam

- **The final exam is closed book.**
- **Students must pass the final exam in order to pass the course.**
- **The passing mark for the exam is 50%.**
- **All materials covered throughout the semester will be included in the final exam. The material covered in review classes is intended for review purposes only.**
- **Each student is required to bring their UCID.**
- Students who arrive late will not be admitted after **thirty minutes** have elapsed from the start of the examination.
- Practice questions provided are intended for training purposes **and may or may not appear in the final exam.**
- The final exam may consist of a diverse range of question formats, including but not limited to coding questions, and questions requiring detailed responses. **These formats may differ from those used in the practice questions.**

Practice questions – cont.

Q1) What is HTML?

A standard markup language used for creating documents intended for presentation in a web browser.

Q2) What are the issues that occur within the client-side code of a web application? Mention two issues.

- Syntax Errors
- Runtime Errors

Practice questions – cont.

Q3) Describe the following:

- **Syntax Errors:**

  Errors in the code that violate the syntax rules of the programming language.

- **Runtime Errors:**

  Errors that occur during the execution of the code.

Practice questions – cont.

Q4) What is the output of the following code snippet:

```
function greet(name, callback) {
  const message = "Hello, " + name + "!";
  callback(message);
}

function displayMessage(message) {
  console.log(message);
}

greet("Alice", displayMessage);
```

Output:

Hello, Alice!

Practice questions – cont.

Q5) Identify and describe the error in the provided JavaScript code and propose a solution to handle it.

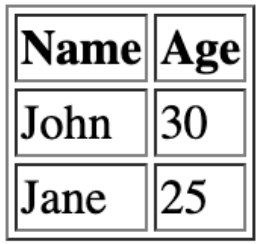
```
var numbers = [1, 2, 3, 4, 5];
var total = 0;

for (var i = 0; i <= numbers.length; i++) {
  total += numbers[i];
}

console.log("The total is:", total);
```

Issue: The loop keeps going until ‘i’ is equal to or greater than the length of the numbers array. This makes the loop run one extra time, trying to access an index that does not exist in the numbers array.

Solution: Changing the loop condition to “i < numbers.length”.

Practice questions – cont.

- Please complete the provided HTML code snippet to create a table similar to:



Name	Age
John	30
Jane	25

```
<html lang="en">
<head> <title>Table Example</title> </head>
<body>
<table border="1">

   <tr>
     <th>Name</th>
     <th>Age</th>
   </tr>
   <tr>
     <td>John</td>
     <td>30</td>
   </tr>
   <tr>
     <td>Jane</td>
     <td>25</td>
   </tr>

</table>
</body> </html>
```

Recap

What is full stack development?

The process of **designing**, **developing**, **testing**, and **deploying** end-to-end web application.

It involves working with various technologies and tools:

- Front-end web development.
- Back-end web development.
- Database development.

What is frontend?

Everything a user sees and interacts with when they click on a link or type in a web address. It includes:

- Content and styles.
- Buttons.
- Links.
- Textboxes.
- Images.
- Videos.

Also called **client-side**

What is backend?

- Backend development is responsible for manipulating and storing user data.
- Backend contains the **business logic** of the web application.
- Pertains to the underlying processes and essential components responsible for the seamless and proper functioning of the front-end.
- Also called **server-side**.
- Users have no direct access to or interact with.

Recap - Continue

Frontend technologies:

- HTML
- CSS
- JavaScript

Backend technologies:

- Python
- PHP
- Node.js

Terminologies

- **Application Programming Interface (API)**: a set of rules and protocols that allows one software application to interact with another. The frontend makes requests to the backend API to retrieve or send data, and the backend processes these requests.

- **Cross-Browser Compatibility**: ensuring that websites function consistently across different browsers.

- **Hosting**: the process of storing, serving, and making a website or web application accessible on the Internet.

Frontend errors

Issues and problems that occur within the client-side code of a web application.

- Syntax Errors: errors in the code that violate the syntax rules of the programming language.
- Runtime Errors: errors that occur during the execution of the code.
- Logical Errors: errors in the logic of the code that cause unexpected behavior or incorrect output.

What is HTML?

- The standard markup language used for creating documents intended for presentation in a web browser.
- It defines the content and structure of web page.
- The server sends HTML documents to the web browser, which then renders the documents into multimedia web pages.

Recap: paragraphs – example

```
<!DOCTYPE html>
<html>
<head>
<title>ENSF381</title>
</head>
<body>

<p>Welcome to the ENSF381 course.</p> <p>We will be
learning full-stack web development.</p>

</body>
</html>
```

JSBin

Recap: paragraphs – example

Attributes - example

```
<!DOCTYPE html>
<html>
<head>
<title>ENSF381</title>
</head>
<body>

<p style="color:red">Welcome to the ENSF381 course.</p>
<p style="color:blue;font-size:20px;">We will be learning
full-stack web development.</p>

</body>
</html>
```

JSBin

Attributes - example

Common attributes

- **id:** uniquely identifies an element on the page.
- **class:** assigns one or more class names to an element, used for styling with CSS.
- **style:** applies inline CSS styles to an element.
- **src:** specifies the source URL or file path for elements like images or iframes.

Other attributes

Boolean Attributes: they don't require a value. If present, they are considered **true**; if absent, they are **false**.

Example:

```
<input type="checkbox" checked>
```

Event Attributes: attributes like **onclick**, and **onmouseover**.

Example:

```
<button onclick="myFunction()">Click me</button>
```

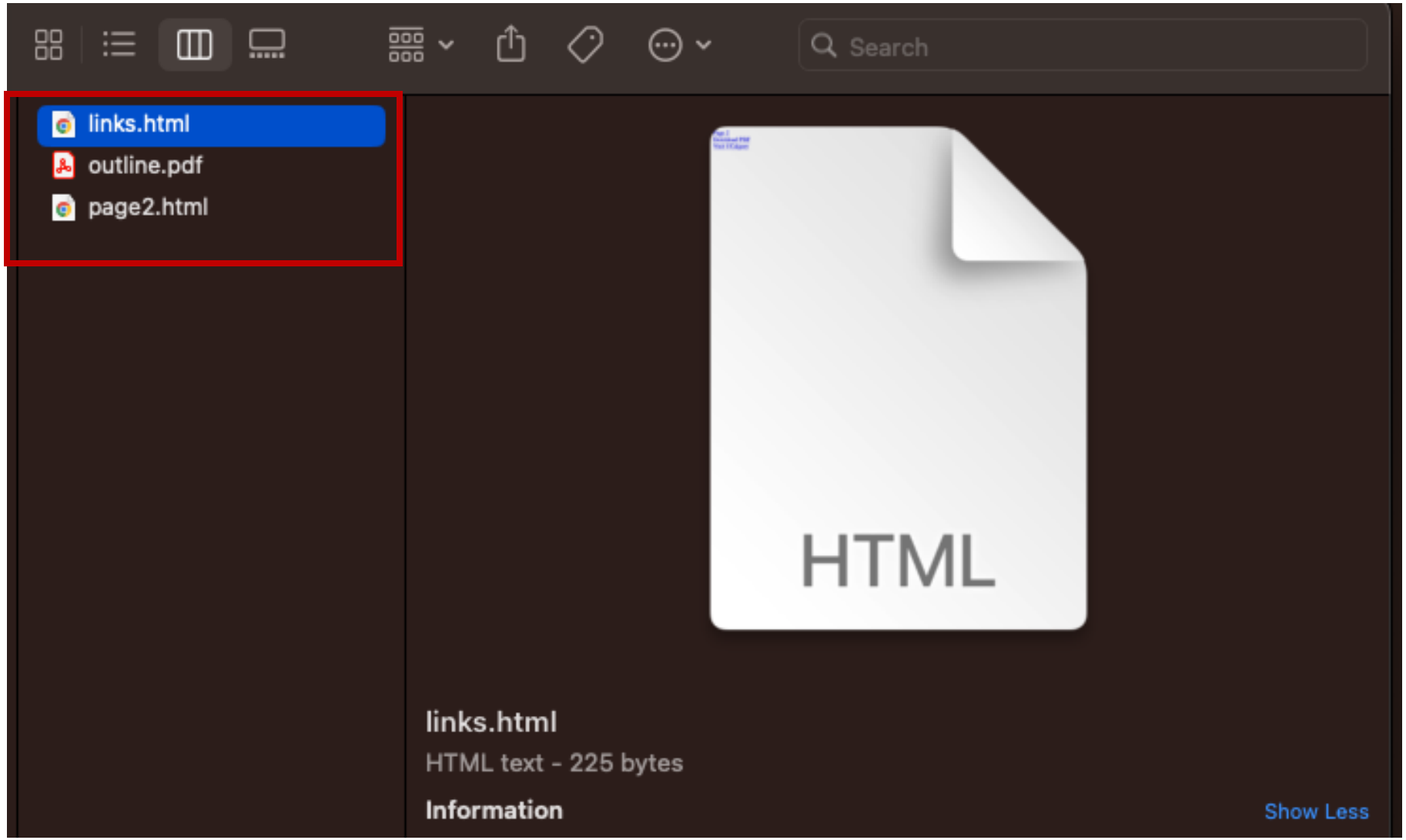
Links

- Allow you to navigate between different web pages.

- Links can point to various destinations such as other web pages, files, images.

```
<a href="destination">Link Text</a>
```

Links - example



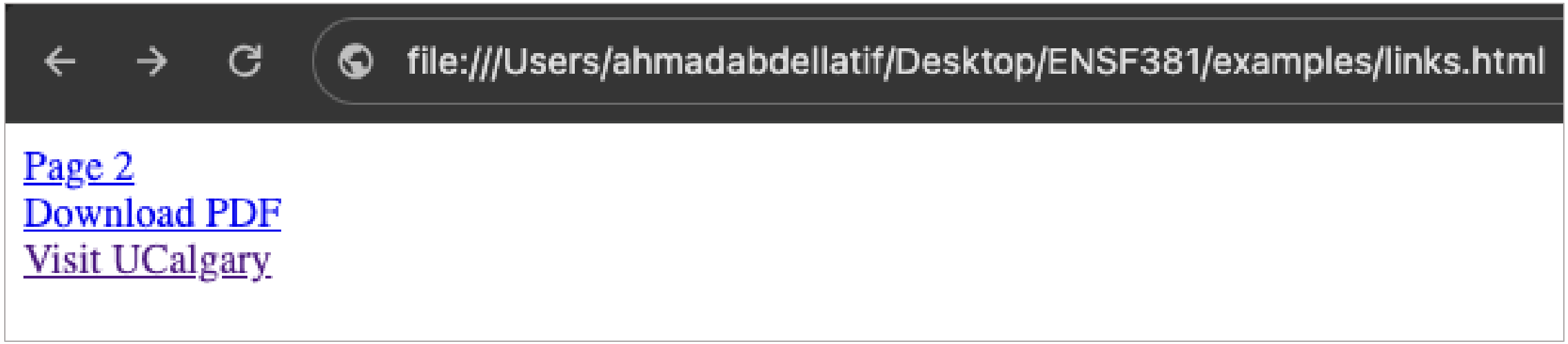
Links – example (links.html)

```
<!DOCTYPE html>
<html>
<head>
<title>ENSF381</title>
</head>
<body>

<a href="page2.html">Page 2</a>
<br>

<a href="outline.pdf">Download PDF</a>
<br>

<a href="https://www.ucalgary.ca">Visit UCalgary</a>

</body></html>
```

Links – example (page2.html)

```
<!DOCTYPE html>
<html>
<head>
<title>ENSF381</title>
</head>
<body>

<h1> Welcome to Page 2 </h1>

</body>
</html>
```

Links - example



What is HTML table?

- HTML table is a structured set of data organized in rows and columns.
- It is created using the <table> element in HTML.
- Tables allow us to:
  - Organize and display data in a structured format.
  - Structure the layout of the webpage.

What is HTML table?

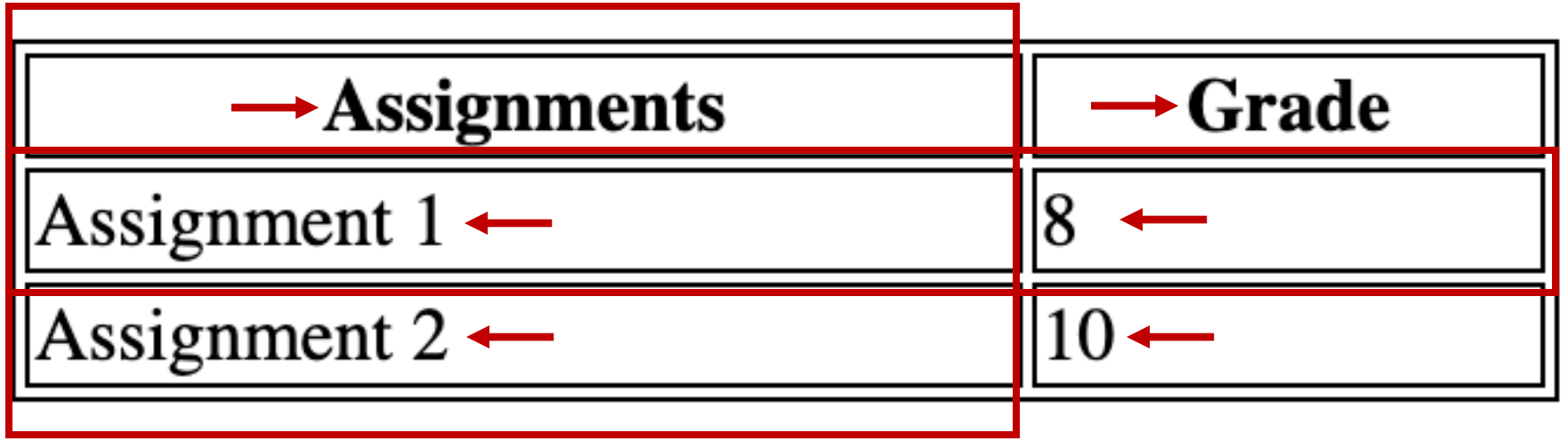


Table structure

- Table (<table>): the container element for the entire table.
- Table row (<tr>): defines a row in the table.
- Table header (<th>): defines header cells in a table. Header cells are typically placed in the first row or first column of the table and contain labels or headings.
- Table data (<td>): defines data cells in a table. These cells **contain the actual data of the table**.

Table - example

```
<!DOCTYPE html>
<html>
<head>
<title>Tables</title>
</head>
<body>
<table>
  <tr>
    <th>Assignments</th>
    <th>Grade</th>
  </tr>
  <tr>
    <td>Assignment 1</td>
    <td>8</td>
  </tr>
  <tr>
    <td>Assignment 2</td>
    <td>10</td>
  </tr>
</table>
</body></html>
```

JSBin

Table - example

Adding another row - example

```
<table border="1px">
    <tr>
      <th>Assignments</th>
      <th>Grade</th>
    </tr>
    <tr>
      <td>Assignment 1</td>
      <td>8</td>
    </tr>
    <tr>
       <td>Assignment 2</td>
       <td>10</td>
    </tr>
    <tr>
       <td>Assignment 3</td>
       <td>9</td>
    </tr>
</table>
```

Adding a new row

JSBin

Adding another row - example

Merging cells

- Combining two or more adjacent table cells into a single, larger cell.

- This can be done either horizontally (using the `colspan` attribute) or vertically (using the `rowspan` attribute).

- Useful technique when you want to **create a more complex layout** or represent data in a way that requires **spanning across multiple rows or columns**.

Merging cells - example

```
<table border="1">
  <tr>
    <td>Cell 1</td>
    <td colspan="2" bgcolor="red">Merged Cells</td>
  </tr>
  <tr>
    <td rowspan="2" bgcolor="orange">Merged Cells</td>
    <td>Data 1,2</td>
    <td>Data 1,3</td>
  </tr>
  <tr>
    <td>Data 2,2</td>
    <td>Data 2,3</td>
  </tr>
</table>
```

JSBin

Merging cells - example

<table>
<tr><td>Cell 1</td><td colspan="2">Merged Cells</td></tr>
<tr><td rowspan="2">Merged Cells</td><td>Data 1,2</td><td>Data 1,3</td></tr>
<tr><td>Data 2,2</td><td>Data 2,3</td></tr>
</table>

What are HTML forms?

- HTML forms (`<form>`) are a crucial part of web development, allowing users to interact with web pages by entering and submitting data.
- Used to collect information from users, such as text input, checkboxes, and radio buttons.
- They provide a structured way to gather user input and send it to a server for processing.

Form Elements

HTML provides several form elements that enable the creation of interactive forms on web pages. Here are some of the key form elements:

- Text Area (`<textarea>`): allows users to input multiline text.
- Dropdown (`<select>`): creates a dropdown list, and it is often used in conjunction with `<option>` elements.
- Button (`<button>`): represents a clickable button, often used to submit a form or trigger a JavaScript function.
- Label (`<label>`): associates a label with a form control.
- Input (`<input>`): used for various types of user input, such as text, password, checkboxes, radio buttons, etc.

Input attributes

HTML `<input>` elements can have various attributes that control their behavior and appearance:

- name: provides a name for the input control, which is used to identify the form data when submitted.
- value: sets the initial value of the input control.
- placeholder: specifies a short hint that describes the expected value of the input field.
- required: indicates that the input field must be filled out before submitting the form.
- disabled: disables the input control, making it non-editable.
- maxlength: specifies the maximum number of characters allowed in the input field.
- autocomplete: indicates whether the browser should provide autocomplete suggestions for the input.
- for: used in conjunction with the `<label>` element to associate the label with a specific form control, typically an input element.

Build a registration form using different input types - example

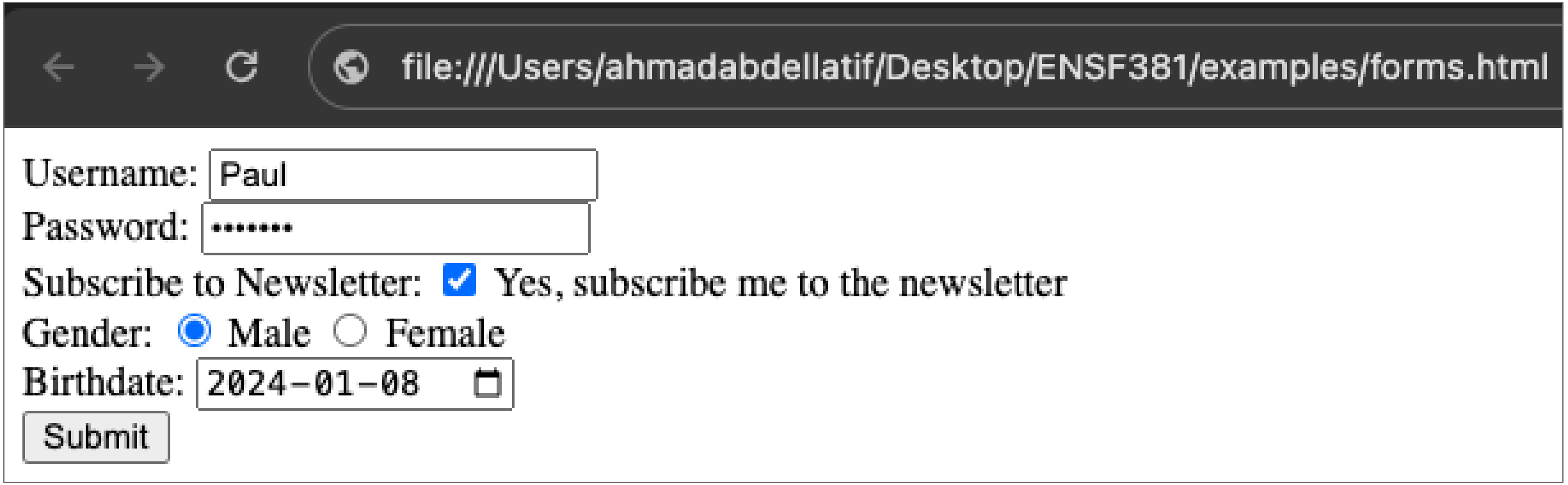
```
<form>
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" placeholder="Enter your username">
  <br>

  <label for="password">Password:</label>
  <input type="password" id="password" name="password" placeholder="Enter your password">
  <br>

  <label>Subscribe to Newsletter:</label>
  <input type="checkbox" id="subscribe" name="subscribe" value="yes">
  <label for="subscribe">Yes, subscribe me to the newsletter</label>
  <br>
```

Build a registration form using different input types - example

```
<label>Gender:</label>
<input type="radio" id="male" name="gender" value="male">
<label for="male">Male</label>
<input type="radio" id="female" name="gender" value="female">
<label for="female">Female</label>
<br>

<label for="birthdate">Birthdate:</label>
<input type="date" id="birthdate" name="birthdate">
<br>

<button type="submit">Submit</button>
</form>
```

JSBin

Build a registration form using different input types - example



What is CSS?

- Stands for Cascading Style Sheets.
- Used for describing the presentation of a document written in a markup language like HTML.
- Allows web developers to control the layout, appearance, and formatting of HTML elements within a web page.

CSS Syntax – example 2

```
<!DOCTYPE html>
<html>
<head>
<title>CSS Example</title>
  <style>
  h1 {
    color: blue;
    }
  p {
      color: pink;
    }
  .my-class {
      font-size: 30px;
      color: orangered;
    }
  #my-id {
      background-color: lightgray;
    }
  </style>
  </head>
```

CSS Syntax – example 2

```
<body>
 <h1>Hello, CSS!</h1>

 <p>Welcome to CSS world!</p>

 <h2 class="my-class">This is h2 tag</h2>
 <p class="my-class">This is the p tag that it has a class property
</p>

 <div id="my-id">This is the div tag</div>

</body>
</html>
```

JSBin

CSS Syntax – example 2

CSS Specificity

- A set of rules that dictate which styles are applied to an element when conflicting styles exist.

- It determines which style declarations are the most relevant or specific to an element and should therefore be applied.

- The specificity of a style is calculated based on the following factors, in order of importance:
  - **Inline styles** - <h1 style="color: pink;">
  - **IDs** - #unique-element {color: blue;}
  - **Classes** - .example-class {font-size: 16px;}
  - **Elements** - p{font-family: "Arial";}

CSS Specificity - example

```
<html>
<head>
  <style>
p {color: red;}
  </style>
</head>
<body>

<p>Hello World!</p>

</body>
</html>
```

JSBin

Example adapted from: https://www.w3schools.com/css/css_specificity.asp

CSS Specificity - example

CSS Specificity - example

```
<html>
<head>
  <style>
.test {color: green;}
p {color: red;}
  </style>
</head>
<body>

<p class="test">Hello World!</p>

</body>
</html>
```

JSBin

CSS Specificity - example

CSS Specificity - example

```
<html>
<head>
  <style>
    #demo {color: blue;}
    .test {color: green;}
    p {color: red;}
  </style>
</head>
<body>

<p id="demo" class="test">Hello World!</p>

</body>
</html>
```

JSBin

CSS Specificity - example

CSS Specificity - example

```
<html>
<head>
  <style>
#demo {color: blue;}
.test {color: green;}
p {color: red;}
  </style>
</head>
<body>

<p id="demo" class="test" style="color: pink;">Hello World!</p>

</body>
</html>
```

JSBin

CSS Specificity - example

Margins

- Refer to the **space around** an element's content.
- Margins create space between an element's border and adjacent elements.
- It can be set for all sides (top, right, bottom, left) or individually. The values can be specified in various units such as pixels, ems, rems, percentages, etc.

Padding

- The space between an element's content and its border.
- Used to create space within an element and separates the content from the border.
- Padding is important for controlling the internal spacing of an element and ensuring that the content doesn't touch the borders or other adjacent elements.

Display property

- Used to define how an HTML element should be displayed on the web page.

- Takes various values, each of which dictates different rendering behaviors:
  - Block: starts on a new line and takes up the full width available.
  - Inline: It does not start on a new line and only takes up as much width as necessary.
  - Grid: allowing child elements to be placed into a grid layout.
  - Flex: allowing child elements to be laid out in any direction.

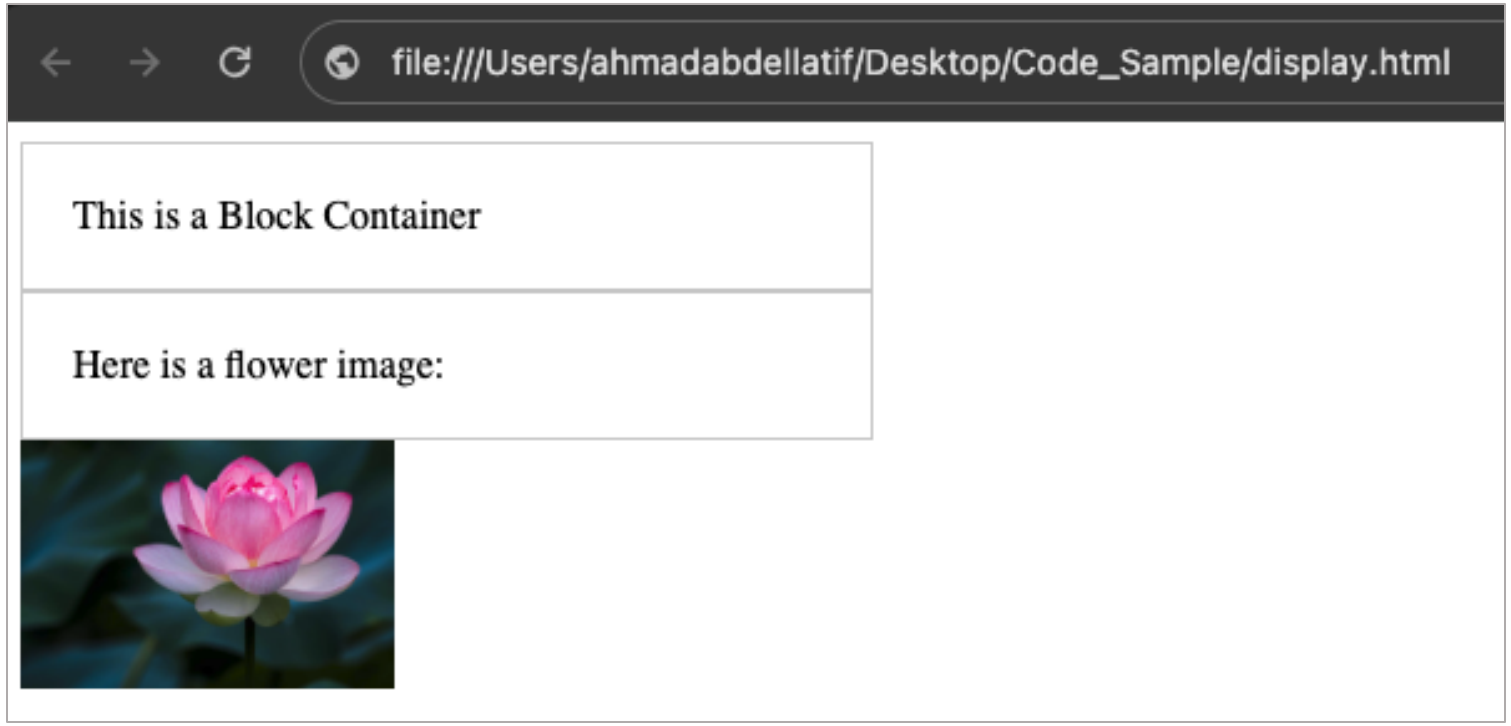
https://www.w3schools.com/css/css_display_visibility.asp

Display property – Block example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    span {
      display: block;
      width: 300px;
      border: 1px solid #ccc;
      padding: 20px;
    }
    img {
      max-width: 100%;
      height: 100px;
    }
  </style>
  <title>Block Example</title>
</head>
<body>
  <div>
    <span>This is a Block Container</span>
    <span>Here is a flower image:</span>
    <img src="https://wallpapers.com/images/featured-full/flower-pictures-unpxbv1q9kxyqr1d.jpg">
  </div>
</body>
</html>
```

JSBin

Display property – Block example



Display property – Inline example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .inline-container {
      display: inline;
      border: 1px solid #ccc;
      padding: 10px;
    }

    /* Add some styles for better visibility */
    .inline-container span {
      background-color: #f0f0f0;
      padding: 5px;
      margin-right: 5px;
    }

    a {
      color: #007bff;
    }
  </style>
  <title>Inline Example</title>
</head>
<body>

  <div class="inline-container">
    <span>Inline Container:</span>
    <a href="https://ucalgary.ca">University of Calgary</a>
  </div>
</body>
</html>
```

Applies the specified styles to the elements that are descendants of an element with the class "inline-container".

JSBin

Display property – Inline example

What is Flexbox?

- A layout model in CSS designed to create efficient and predictable layouts for user interfaces.

- Distribute space and align items within a container, even **when the size of the items is unknown or dynamic**.

- It provides an easy and clean way to design complex layouts.

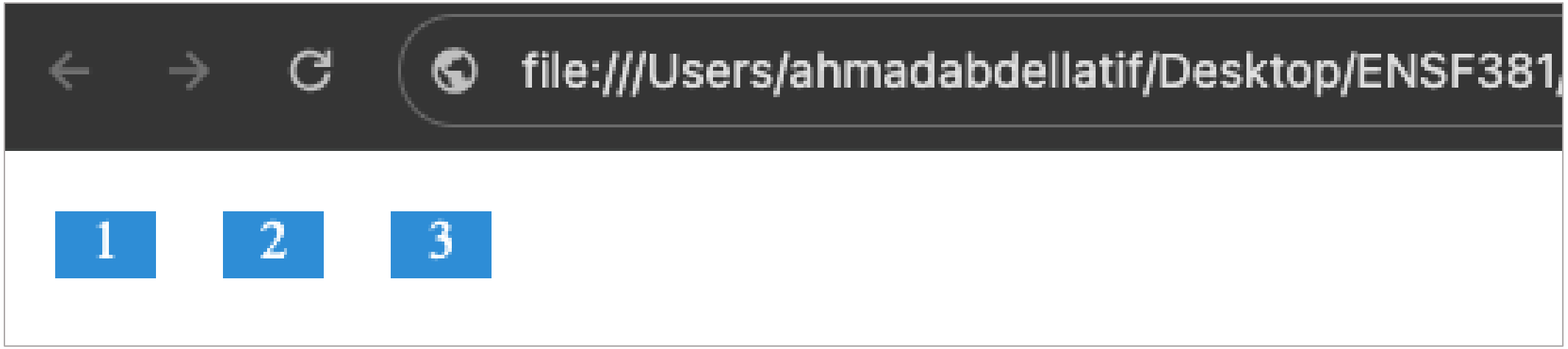
Flexbox - example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .flex-container {
      display: flex;
    }

    .flex-item {
      width: 30px;
      height: 20px;
      margin: 10px;
      background-color: #3498db;
      color: #ffffff;
      text-align: center;
    }
  </style>
</head>
<body>

<div class="flex-container">
  <div class="flex-item">1</div>
  <div class="flex-item">2</div>
  <div class="flex-item">3</div>
</div>
</body> </html>
```

Set the horizontal alignment of the text content within an element.

JSBin

Flexbox - example



What is JavaScript?

- High-level and dynamic programming language.
- One of the core technologies that enable interactive and dynamic content on websites.
- Used to enhance the user experience by providing features such as client-side validation, animations, and real-time updates without requiring a page reload.
- To include JavaScript in an HTML document, use the <script> tag.

Variables

- Variables are used to store data values.
- JavaScript has several data types, including:
  - Primitive Types: String, Number, Boolean, Null, Undefined.
  - Complex Types: Object, Array, Function.
- Declare variables using:
  - var: function-scoped, meaning their scope is limited to the function in which they are declared. If declared outside any function, they **become global**.
  - let: block-scoped, meaning their scope is **limited to the block** (enclosed by curly braces) in which they are declared. Variables declared with let cannot be Redeclared in the same scope.
  - const: variables must be initialized at the time of declaration, and their **values cannot be re-assigned after initialization**.

Strings

- A string is a sequence of characters enclosed in single (') or double (") quotes.

```
let singleQuotes = 'This is a string with single quotes.';
let doubleQuotes = "This is a string with double quotes.";
```

- Quotes within a string are permissible as long as they do not match the quotes enclosing the string.

- Strings can be concatenated using the + operator.

String useful method - examples

Code snippet

```
let str = "Hello, World!";
let len = str.length;
```

→ *len will be 13*

```
let str = "JavaScript";
let sub = str.substring(0, 4);
```

→ *sub will be "Java"*

```
let str = "JavaScript";
let sub = str.substr(4, 6);
```

→ *sub will be "Script"*

```
let str = "Hello, World!";
let newStr = str.replace("Hello", "Hi");
```

→ *newStr will be "Hi, World!"*

```
let str = "JavaScript";
let upperCase = str.toUpperCase();
let lowerCase = str.toLowerCase();
```

→ *upperCase will be "JAVASCRIPT"*
lowerCase will be "javascript"

```
let str = "   Hello, World!   ";
let trimmedStr = str.trim();
```

→ *trimmedStr will be "Hello, World!"*

```
let sentence = "Welcome to ENSF381 course.";
let indexOfENSF = sentence.indexOf("ENSF381");
let indexOfCourse = sentence.indexOf("course");
```

→ *indexOfENSF will be 11*
indexOfCourse will be 19

Result

What is Array?

- A data structure that allows you to store and organize multiple values under a single name.

- These values, known as elements, can be of the same or different data types and are accessed using an index.

What is the output?

Code snippet

```
let fruits = ["apple", "orange",
"banana"];
console.log(fruits.length);
```

Output

→ 3

```
let colors = ["red", "green", "blue"];
let colorsString = colors.toString();
console.log(colorsString);
```

→ red,green,blue

```
let numbers = [1, 2, 3, 4, 5];
let slicedNumbers = numbers.slice(1, 4);
console.log(slicedNumbers);
console.log(numbers);
```

→ [2, 3, 4]
[1, 2, 3, 4, 5]

Destructuring objects

- Allows to extract values from objects or arrays and assign them to variables in a more concise and convenient way.

- It provides a cleaner syntax for **extracting multiple properties** from an object and assigning them to variables.

- It is commonly used in scenarios where you need to extract specific properties from objects, especially when dealing with API responses or complex data structures.

Destructuring objects - example

```
// Creating an object
let person = {
  name: "John Doe",
  age: 25,
  address: {
    city: "Example City",
    country: "Example Country"
  }
};

// Destructuring object properties
let { name, age, address } = person;

console.log(name); // Output: John Doe
console.log(age); // Output: 25
console.log(address); // Output: { city: 'Example City', country: 'Example Country'}
```

What are functions in JavaScript?

- Functions in JavaScript allow developers to structure their code in a more organized and modular manner.
- Enable abstraction by allowing developers to encapsulate a set of operations behind a single function.
- Can be reused across different parts of a program.
- Facilitate collaborative development by breaking down a large project into smaller, more manageable tasks.

Function - example

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Function Example</title>
</head>
<body>
    <script>
        function calculateSquare(number) {
            return number * number;
        }

        console.log(calculateSquare(5));

    </script>
</body></html>
```

Function – example 2

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Function Example</title>
</head>
<body>
    <script>

        function greetPerson(name) {
         // Using string template to create a greeting message
         const greeting = `Hello ${name}, welcome to ENSF381!`;

         // Returning the greeting
         return greeting;
        }

        let message = greetPerson('Eric');
        console.log(message);
    </script>
</body></html>
```

Arrow functions - example

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Function Example</title>
</head>
<body>

  <script>

    let greetPerson = (name='John') => `Hello ${name}, welcome to ENSF381!`;

    let message = greetPerson('Eric');
    console.log(message);

    message = greetPerson();
    console.log(message);

    </script>
</body></html>
```

ForEach loop - example

```
<!DOCTYPE html>
<html>
<head>
  <title>ForEach Example</title>
</head>
<body>
  <script>
    // Function to calculate the square of a number
    function calculateSquare(number) {
      console.log(number * number);
    }

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

    // Use forEach to iterate through the numbers array
    numbers.forEach(calculateSquare);

  </script>
</body>
</html>
```

JavaScript Promises

- Promises are a mechanism for handling asynchronous operations in a more structured and manageable way.

- An objects used to represent the eventual completion or failure of an asynchronous operation and its resulting value.

- You can think of a promise as:

“I Promise a Result! Either way, I’ll come back and let you know how it went”

Async/Await syntax

```
async function fetchData() {
  try {
    //make an asynchronous request to the specified URL
    const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');

    //Parse the response body as JSON; wait for this asynchronous operation to complete
    const data = await response.json();

    // Process the fetched data
    console.log('Fetched Data:', data);
  } catch (error) {
    // Handle errors
    console.error('Error fetching data:', error);
  }}
```

`async:` declares a function as asynchronous that returns a Promise.

`fetch(URL):` A function used to make network requests, typically to retrieve data from a server.

`await:` within an asynchronous function, you can use the await keyword to pause the execution of the function until the Promise is resolved.

Example on getting the element by ID and log its text content

```
<!DOCTYPE html>
<html>
<head>
  <title>Get Element by ID Example</title>
</head>
<body>

<!-- Get Element by ID Example -->
<p id="demoParagraph">This is a paragraph with an ID.</p>

<script>
var paragraphById = document.getElementById("demoParagraph");
if (paragraphById) {
  console.log("Element found by ID: " + paragraphById.textContent);
}
</script>
</body>
</html>
```

What are front-end frameworks?

- Front-end frameworks are predefined, standardized code libraries or structures that expedite and streamline the web development process.

- Over the years, frontend frameworks have evolved to meet the demands of modern web applications, incorporating best practices and innovative solutions.

Benefits of using front-end frameworks

- Code Organization and Structure: frameworks provide a structured and organized way to develop frontend applications.
- Reusability and Modularity: components, a fundamental concept in many frameworks, encourage code reusability.
- Consistent UI/UX Design: frameworks often come with design systems and UI components that ensure a consistent look and feel across the application.
- Improved Performance**:** optimizations, such as virtual DOM in React, contribute to improved performance by minimizing unnecessary re-renders.
- Cross-Browser compatibility: frameworks handle browser-specific quirks, ensuring a consistent experience across different browsers.

Questions