AltSchool School of Engineering

Class notes for the 1st Semester

Are you ready to learn Software Engineering focusing on web technologies? Press `space` on your keyboard →

Table of contents

What is Software Engineering?

Software Engineering is the systematic application of engineering principles to the design, development, maintenance, testing, and evaluation of software.

It involves a disciplined approach to analyzing user needs, planning and managing projects, creating software systems, and ensuring their reliability, efficiency, and maintainability.

You will need a blend of technical Skills, engineering methods and project management to produce high-quality software systems.

Read more about Software Engineering?

Foundational Requirement

- coding
- coding
- coding
- coding

Why Web and Cloud?

- Career Advantages
    - High Demand and Competitive Salaries
    - Remote Work
    - Entrepreneurial Options
- Technical Advantages
    - Scalability, Cost Efficiency, Security and CI/CD
    - Access to Advanced Tools and Services
    - Global Reach
- Educational and Community Benefits
    - Extensive Learning Resources
    - Active Community Support and Open Source
- Innovation and Future-Proofing
    - Cutting-Edge Technologies
    - Adaptability

Focus of this Class

- HTML

- CSS

- JavaScript

- Git and GitHub | Open Source

Tools

- Visual Studio Code or any equivalent (JetBrains Webstorm IDE, Zed, Sublime Text)

- Git and GitBash for windows, Git only for mac and linux

- Nodejs, Python

Accounts

Some Account You Expected to (Create || Have)

- GitHub and any other equivalent in GitLab or BitBucket
- LinkedIn
- Twitter or X
- Stackblitz or Codesandbox
- Codepen
- Stackoverflow
- ChatGPT
- Figma
- Dev.to | hashnode | Medium
- Slack or Discord
- Netlify | Vercel | Render
- Personal Website. we will teach to create your own.

Installations

Algorithms and FlowCharts

Definition: An algorithm is a step-by-step procedure or a set of rules designed to perform a specific task or solve a particular problem. It is a sequence of instructions that are followed to achieve a desired outcome.

Example: An algorithm for making a cup of tea might include steps like boiling water, adding a tea bag to a cup, pouring the hot water into the cup, letting it steep for a few minutes, and then removing the tea bag.

♀ Tip

1. Patterns Reduce Complexity like two-pointer, sliding window, divide and conquer, dynamic programming, etc.
2. Optimize Thinking for Efficiency
3. Develop Transferable Skills
4. Practice and Apply

✓ Success

Building intuition around patterns will significantly improve your problem-solving skills, speed, and confidence in tackling algorithmic challenges.

flowchart

Definition: A flowchart is a visual representation of the steps in a process or system using symbols, arrows, and text. It depicts the sequence of operations or steps, making it easier to understand how a process flows from start to finish.

Example: A flowchart for logging into a website might start with a "Start" symbol, followed by a decision symbol asking if the user has entered their username and password, arrows leading to "Enter username" and "Enter password" steps, and an end symbol once the login process is successful.

Flowchart for finding the greatest of three numbers a, b, and c
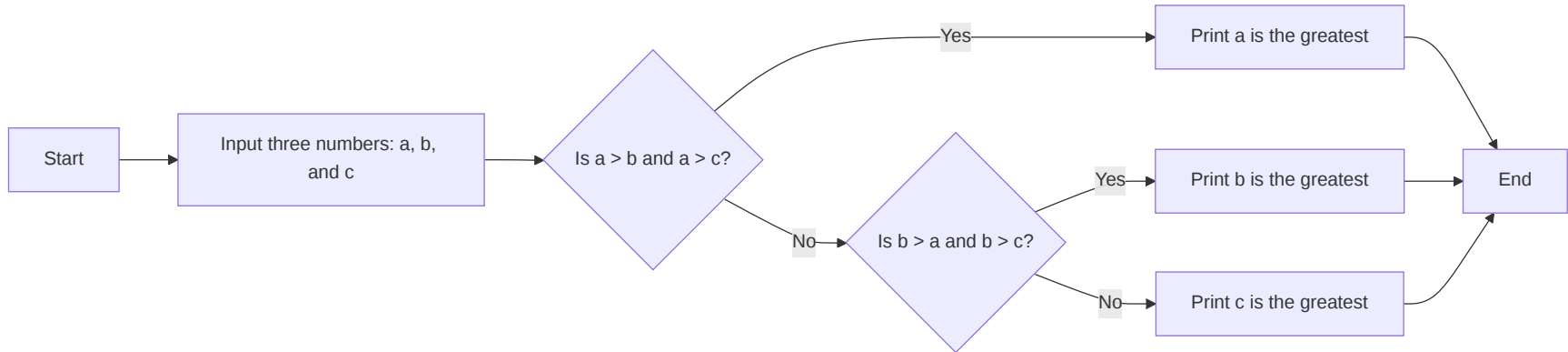
Problem Solving

Use ChatGPT to research what the key components of problem solving are. Try to use it to solve any problem you have.

Programming Paradigms

- Procedural Programming

- Object-Oriented Programming (OOP)

- Functional Programming

- Logic Programming

- Declarative Programming

- Concurrent Programming

- Event-Driven Programming

Each paradigm brings its own way of thinking and problem-solving, making some paradigms more suitable for certain types of tasks than others. Modern programming often involves a combination of these paradigms to leverage their respective strengths.

Programming Concepts

- Variables
- Data Types
- Control Structures
- Functions (or Methods)
- Data Structures
- Algorithms
- Object-Oriented Concepts
  - Encapsulation
  - Polymorphism
  - Inheritance

- Recursion
- Error Handling
- Memory Management
- Concurrency
- File I/O

Understanding these concepts is fundamental to mastering programming and can significantly improve your ability to develop complex and efficient software solutions.

Variables

Storage locations in memory with a name, used to hold data.

```
let name = 'AltSchool'
let age = 99

console.log({name, age})
```

```
{
  "name": "AltSchool",
  "age": 99
}
```

```
1    name = 'AltSchool'
2    age = 99
3
4    print(name, age)
```

Data Types

Classification of data items, defining the operations that can be performed on them.

Primitive types: int, char, float, boolean, number, bigint, symbol, string, undefined, null,

Composite types: arrays, structs, classes.

Abstract data types: List, Stack, Queue, etc.

```js
let name = 'AltSchool'
let age = 99
let isStudent = true

const arrayOfScore = [99, 40, 50]
const person = { name: name, age: age, isStudent: isStudent }

console.log(arrayOfScore)
console.log(person)
```

```
[99, 40, 50]
{
  "name": "AltSchool",
  "age": 99,
  "isStudent": true
}
```

Control Structures

Direct the order of execution of statements in a program.

Conditional statements: if, else, switch.

Loops: for, while, do-while.

Branching: break, continue, return

```
1  let name = 'AltSchool'
2  let age = 99
3
4  if (condition) {
5    // do something
6  } else if (condition) {
7    // do something else if
8  } else {
9    // finally do something
10  }
```

Functions (or Methods)

Blocks of code designed to perform a particular task, reusable throughout the program.

Key Points

- Definition and calling.

- Parameters and return values.

- Scope and lifetime of variables.

Data Structures

Ways of organizing and storing data to enable efficient access and modification

Key Points: Linear: Arrays, Linked Lists. Non-linear: Trees, Graphs. Abstract: Stack, Queue, Map, Set

Data Structures More (Trees)

Binary Search Tree, AVL Tree, Red-Black Tree Segment Tree Fenwick Tree (Binary Indexed Tree)

TREE
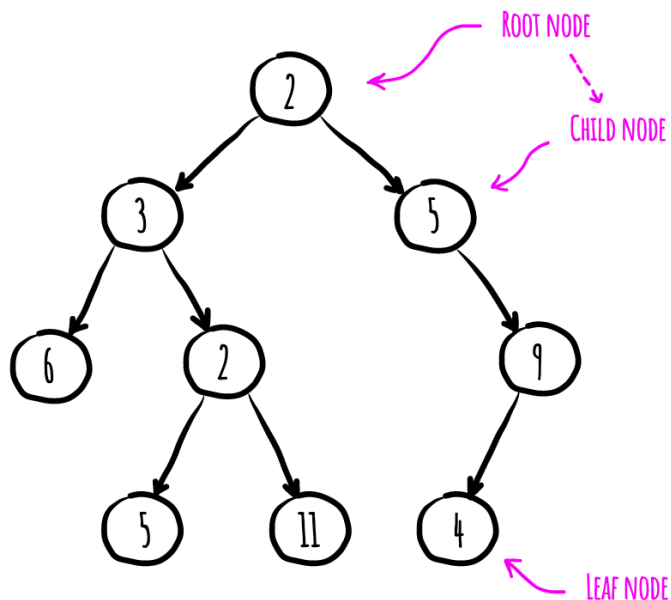
A collection of nodes, where each node is a data structure consisting of a value, together with a list of references to nodes, with the constraints that:

- no reference is duplicated,
- and none points to the root.

There are many kinds of trees:

Binary Search Tree

AVL Tree

Fenwick Tree

Red-Black Tree

Segment Tree

OKSO.APP

Algorithms

Step-by-step procedures or formulas for solving problems.

Key Points

Sorting: Bubble sort, Quick sort, Merge sort. Searching: Linear search, Binary search. Complexity: Big O notation for time and space.

Object-Oriented Concepts

Principles used in OOP to create objects that model real-world entities

Key Points

Classes and Objects. Encapsulation, Inheritance, Polymorphism, Abstraction. Constructors and destructors

Encapsulation - Hiding the internal state and requiring all interaction to be performed through an object's methods

Polymorphism - The ability of different classes to be treated as instances of the same class through a common interface

Inheritance - is a mechanism that allows a class to inherit properties and behaviors from another class

Recursion

A function calling itself to solve a smaller instance of the same problem.

Notable Key Points

- Base case and recursive case.
- Stack overflow and efficiency considerations.
- Examples: Factorial, Fibonacci sequence

Error Handling

Mechanisms to handle runtime errors or exceptional conditions.

Key Points

- Try, catch, finally blocks.

- Throwing exceptions.

- Custom exception classes

Memory Management

Techniques to control the allocation, use, and deallocation of memory

Concurrency and Asynchronous Operations

Running multiple computations simultaneously

File I/O

Reading from and writing to files. File streams, Opening, reading, writing, and closing files, Binary vs text files

HTML

Getting Up and Running with HTML, its important to understand the basic structure of an HTML document. Document Type and HTML content itself (html tag) which contains the Metadata (head tag and its related tags) and Body, what is displayed in the browser visible area (possible elements that can be in the body)

Getting Up and Running with HTML

HTML(HyperText Markup Language) is the foundation of basically every web page, basically, it is the core language of the World Wide Web. It's how we tell browsers to structure content into paragraphs, headings, images, links, lists, forms, tables, buttons, and more. If you're interested in building a website, web development, or just coding in general, learning HTML is a great place to start.

opening tag content closing tag

`<h1>` `Frank Ocean` `</h1>`
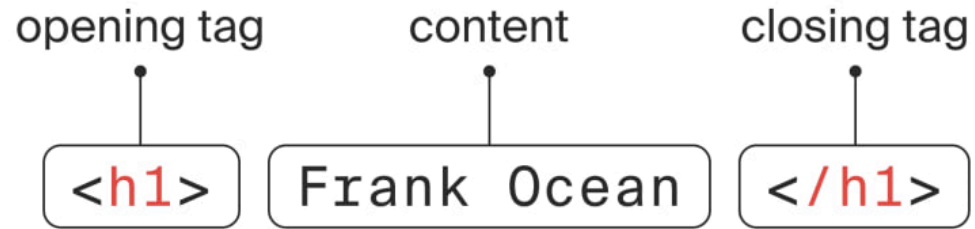
Let's break it down a bit

- HTML uses "markup" to annotate text, images, and other content for display in a Web browser. This markup is called ELEMENT or TAG.HTML Element or Tag includes `<head>`, `<title>`, `<body>`, `<header>`, `<footer>`, `<article>`, `<section>`, `<p>`, `<div>`, `<span>`, `<img>`, `<aside>`, `<audio>`, `<canvas>`, `<datalist>`, `<details>`, `<embed>`, `<nav>`, `<search>`, `<output>`, `<progress>`, `<video>`, `<ul>`, `<ol>`, `<li>` and many others.

- "HyperText" is text on a web page that contains references to another web page. You probably know these as hyperlinks. We use hyperlinks to jump to another section of the same page, a different page on the current website, or a completely new website. Hyperlinks can also open a PDF, email, or multimedia, like a video or audio file.

- Linking information together in this way was a revolutionary step in building the web. Together, HTML and the internet make it possible for anyone to access all types of information around the world, in any order they want.

- Finally, "Language" is the simplest part of the acronym to understand. Like any language, HTML has a unique syntax and alphabet. But what kind of language is it, exactly? It's a markup language.

HTML Element

An HTML element or "tags", which consist of the element name surrounded by < and > like `<p>` . The name of an element inside a tag is case-insensitive. That is, it can be written in uppercase, lowercase, or a mixture. For example, the `<title></title>` tag can be written as `</Title></Title>` , `<TITLE>` `</TITLE>` , or in any other way. However, the convention and recommended practice is to write tags in lowercase. These tags or elements can be either self closing `<img/>` or normal/common once with opening tag `<p>` and a closing tag `</p>` .

⚠ Danger	⚠ Danger	⚠ Danger
There are three rendering modes in modern browsers: Quirks Mode(Old, pre-standardized HTML CSS behavior), Standards Mode(modern HTML/CSS standards), and Almost Standards Mode (slight compatibility)	A page will enter into quirks mode if the doctype is not declared or is declared incorrectly. Browser can auto complete tags, but it is recommended to close tags properly. this can lead to unexpected results.	If you decide to use the opening tag of an element, you have a duty to close the tag if it is not a self-closing tag.

First HTML Document

create a folder and add an `index.html` file either using your vs code or terminal or file explorer. Use the liveserver extension to view the file in the browser.

> ⓘ **Info**
>
> Always open your vscode on the right folder your created for your project not on the root folder of your computer or desktop or documents or downloads.

```html
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Responsive Web Page</title>
7       <meta name="description" content="This is a responsive web page">
8   </head>
9   <body>
10      <h1>Welcome to the world of HTML</h1>
11  </body>
12  </html>
```

Check this out:


```
User                              Browser

      <h1>Welcome to the world of HTML</h1>

         Breakdown of the HTML element

              Opening tag: <h1>

         Content: Welcome to the world of HTML

              Closing tag: </h1>

User                              Browser
```
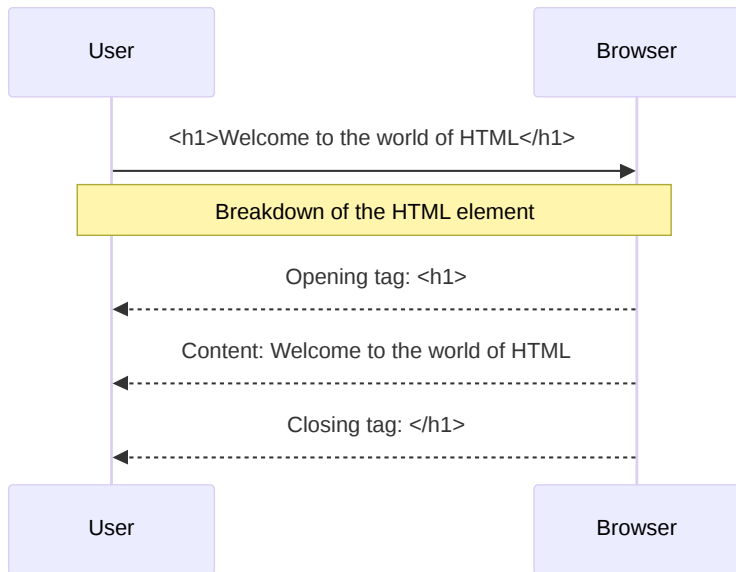
Explanation

The diagram explains what happens when a user view the first html page we created in a browser using right localhost:port / 127.0.0.1:port.

The browser reads the content of the page and displays it in the browser window. The content of the page is enclosed within the `<h1>` tags, which define a top-level heading.

The `h1` is an instruction to the browser to display the text as a top-level heading.

The browser renders or paint the content between the tags on the screen.

The text "Welcome to the world of HTML" is displayed as the main heading of the page.

Self Closing Tags

HTML <img/> Tag

We have been talking about tags in our previous teachings but this time you'll be seeing another form of tag that's called self closing tags under which the popular image tag fall under.

Self-closing tags, also known as void elements, are a feature in HTML and XML where the tag does not require a separate closing tag. Instead, the tag is closed within itself. This is useful for elements that do not have any content between an opening and a closing tag. Here are some examples and details about self-closing tags:

- `<img />` : Defines an image.
- `<br />` : Inserts a line break.
- `<hr />` : Creates a horizontal rule (a line).
- `<input />` : Defines an input field.
- `<meta />` : Provides metadata about the HTML document.
- `<link />` : Defines the relationship between a document and an external resource (most commonly used to link to stylesheets).

More on the Self Closing Tags and Difference between `tags` and `elements`

The closing tag is the same tag as the opening tag, preceded by a slash.

Examples of html element that are self-closing includes `img`, `input`, `br`, `hr` and others.

Elements and tags aren't the exact same thing, though many people use the terms interchangeably. The tag name is the content in the angle brackets. The tag includes the angle brackets and the name of the element. In this case, `<h1>`. An "element" is the opening and closing tags, and all the content between those tags, including nested elements.

Note: Browsers do not display the tags. The tags are used to interpret the content of the page.

HTML can be categorized to block and inline element. Block elements typically start on a new line and take up the full width available. Inline elements do not start on a new line and only take up as much width as necessary using the size of its content. `div` is a block element while `span` is an inline element. Multiple inline element can stay on a block.

In HTML5, the closing slash is optional, and self-closing tags can be written without it. However, for compatibility with XML parsers, it is sometimes included. XHTML requires the self-closing tags to include the

Syntax:

In HTML, self-closing tags can be written in two ways:

HTML5 Syntax:

- `<img src="image.jpg" alt="Description">`

- `<br>`

This syntax is valid in HTML5 and does not require a closing slash (/).

XHTML Syntax:

- `<img src="image.jpg" alt="Description" />`

-

This syntax is required in XHTML, which is a stricter form of HTML based on XML.

HTML Page

Block element

Inline element

Elements can be nested inside each other:

Block element

Inline element

Code Example on Self-Closing Tags

```html
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Self-Closing Tags Example</title>
7       <link rel="stylesheet" href="styles.css" />
8   </head>
9   <body>
10      <h1>Self-Closing Tags Example</h1>
11      <span>I will only contain the size of my content:
12        Use <code>br</code> to break the content to the next page
13      </span>
14      <img src="image.jpg" alt="A beautiful view" />
15      <br />
16      <input type="text" name="username" placeholder="Enter your username" />
17      <hr />
18  </body>
19  </html>
```

Document Structure

HTML documents include a document type declaration and the `<html>` root element. Nested in the `<html>` element are the document head and document body. While the head of the document isn't visible to the sighted visitor, it is vital to make your site function. It contains all the meta information, including information for search engines and social media results, icons for the browser tab and mobile home screen shortcut, and the behavior and presentation of your content.

- `<DOCTYPE html>`

- `<html>`

- `<head>`

- `<head/>`

- `<body>`

- `<body/>`

- `<html/>`

DOCTYPE

`<!DOCTYPE html>` To start an HTML document you need to type `<!DOCTYPE html>` at the top of the document, this may look like an HTML element because it's wrapped in tags but it isn't. It's a special kind of node called declaration which tells the browser to use standards mode. If this `<!DOCTYPE html>` is omitted, as quirks mode.

`<html>`

The `<html>` element is the root element for an HTML document. It is the parent of the `<head>` and `<body>` containing everything in the HTML document other than the doctype. If omitted it will be implied, but it is important to include it, as this is the element on which the language of the content of the document is declared.

> (i) **Info**
>
> Note: The `lang` attribute added to the html tag to give this `<html lang="en">` tag defines the main language of the document. The value of the lang attribute is a two- or three-letter ISO language code followed by the region. The region is optional, but recommended, as a language can vary greatly between regions.

`<head>`

Nested between the opening and closing `<html>` tags, we find the two children: `<head>` and `<body>` .let us discuss the several elements that can be in the `<head>` :

- `<meta />`
- `<link />`
- `<script>`
- `<style>`

- `<base>`
- `<title>`
- `<noscript>`

The `<head>` which can also be referred to as document metadata header, contains all the metadata for a site or application and some of these meta tags are:

- `<meta charset="UTF-8"/>`
- `<meta name="viewport" content="width=device-width, initial-scale=1.0"/>`
- `<title>Learning HTML</title>`
- `<link rel="stylesheet" href="./style.css"/>`

Read more about what elements should be included in the head   OLUWASETEMI/HEAD

Character encoding

```
1    <meta charset="UTF-8"/>;
```

By declaring UTF-8 (case-insensitive), you can even include emojis in your title (but please don't).

The character encoding is inherited into everything in the document, even `<style>` and `<scipt>` This little declaration means you can include emojis in class names and the selectorAPI (again, please don't). If you do use emojis, make sure to use them in a way that enhances usability without harming accessibility.

Document title

The `<title>` element is metadata that represents the title of the overall HTML document (not the document's content.)

The contents for the document title, the text between the opening and closing `<title>` tags, are displayed in the browser tab, the list of open windows, the history, search results, and, unless redefined with `<meta>` tags, in social media cards.

Viewport metadata

The other meta tag that should be considered essential is the viewport meta tag, which helps site responsiveness, enabling content to render well by default, no matter the viewport width and also enhances the user experience.

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

name="viewport": This specifies that the meta tag is providing information about the viewport. The viewport is the user's visible area of a web page, which varies with the device used to view the site (desktop, tablet, mobile phone).

content="width=device-width, initial-scale=1.0": This attribute contains the settings for the viewport. It is a comma-separated list of properties and values. In this case, it contains two key properties:

width=device-width: This sets the width of the viewport to be equal to the width of the device. It ensures that the webpage is not scaled down or up but instead uses the full width of the device's screen. initial-scale=1.0: This sets the initial zoom level when the page is first loaded. A scale of 1.0 means no zoom, i.e., the page content appears at 100% of its size.

Body

The `<body>` tag defines the document's body.

The `<body>` element contains all the contents of an HTML document, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.

Note: There can only be one `<body>` element in an HTML document.

Inside the `<body>` tags of an HTML document, you can find a wide variety of elements that are used to structure and present the content of a webpage. Here are some common HTML tags that are typically found inside the `<body>` tag:

```
1   <body>
2
3   1. Headings:
4   <h1> to <h6>
5
6   2. Paragraph:
7   <p>
8
9   3. Links:
10  <a>
```

Contd.

```
1    4. Lists:
2    Ordered List: <ol>, <li>
3    Unordered List: <ul>, <li>
4    Description List: <dl>, <dt>, <dd>
5
6    5. Tables:
7    <table>, <tr>, <th>, <td>, <thead>, <tbody>, <tfoot>
8
9    6. Forms:
10   <form>, <input>, <textarea>, <button>, <select>, <option>, <label>, <fieldset>, <legend>
11
12   7. Images:
13   <img>
14
15   8. Media:
16   <audio>, <video>, <source>
17
18   9. Embedded Content:
19   <iframe>, <embed>, <object>, <param>
20
21   10. Sections and Grouping Content:
22   <div>, <span>, <header>, <footer>, <main>, <section>, <article>, <nav>, <aside>
```

Contd.

```
11. Text Formatting:
<b>, <i>, <strong>, <em>, <small>, <mark>, <del>, <ins>, <sub>, <sup>

12. Interactive Elements:
<button>, <details>, <summary>

13. Semantic Elements:
<figure>, <figcaption>, <time>, <progress>, <meter>

14. Script and Styles:
<script>, <noscript>, <style>

</body>
```

Let's put some of the tags in usage

```html
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Responsive Web Page</title>
7   </head>
8   <body>
9       <h1>Welcome to my responsive web page!</h1>
10      <p>This page looks good on both desktop and mobile devices.</p>
11  </body>
12  </html>
```

ⓘ Info

Install two extensions in your vscode to make coding easy, liveserver from Ritwick Dey and Official prettier formatter to format your code.
It might be very helpful to learn some vscode shortcuts to make you fast when working with vscode.

Breakdown of the Code:

1. `<DOCTYPE html >` This declaration defines the document type and version of HTML being used. `<DOCTYPE html>` specifically tells the browser that this document is written in HTML5, which is the latest version of HTML.

2. `<html lang="en">` This is the opening tag for the HTML document. The lang="en" attribute specifies the language of the document, which in this case is English. This helps search engines and browsers understand the primary language of the content.

3. `<head>` The `<head>` element contains meta-information (metadata) about the document that is not displayed on the page. It includes links to scripts, stylesheets, and other metadata.

Within the `<head>` : `<meta charset="UTF-8"/>` This tag specifies the character encoding for the HTML document. UTF-8 is a universal character set that supports many different characters from various languages. It ensures that the text is properly displayed.

`<meta name="viewport" content="width=device-width, initial-scale=1.0"/>` This tag controls the viewport's size and scale on different devices. It ensures the website is responsive and adjusts to different screen sizes, width=device-width sets the viewport to match the device's width. initial-scale=1.0 sets the initial zoom level to 100%.

Contd

4. `<title>` Responsive Web Page `</title>` The `<title>` tag defines the title of the HTML document, which appears in the browser's title bar or tab. It is also used by search engines as the title of the page in search results.

5. `<body>` The `<body>` element contains all the content that is displayed on the web page, such as text, images, links, etc.

Within the `<body>` : `<h1>` Welcome to my responsive web page! `</h1>` The `<h1>` tag defines a top-level heading on the page. This is often the main heading and is typically the most prominent piece of text.

`<p>` is page looks good on both desktop and mobile devices. `</p>` The `<p>` ag defines a paragraph of text. It contains the main body text and ensures the content is properly formatted and readable.

Understanding Semantic HTML

Semantic HTML is the practice of using HTML elements to structure your content based on their meaning and purpose, rather than their appearance. By using semantic markup, you provide context and meaning to the content, making it easier for both humans and machines (like search engines and assistive technologies) to understand the structure and purpose of the content.

Meaning Over Appearance

This emphasizes that HTML elements should be chosen based on their semantic meaning, not their visual appearance. For example, don't use an `<h1>` element just because it renders text as large and bold by default; use it to represent the main heading or title of the content

Non-semantic vs. Semantic Markup

```
1  <div>
2    <div>Semantic Markup</div>
3    <div>
4      <span>one word</span>
5      <span>one word</span>
6      <span>one word</span>
7      <span>one word</span>
8    </div>
9  </div>
10 <div>
11   <div>
12     <div>five words</div>
13   </div>
14   <div>
15     <div>three words</div>
16     <div>forty-six words</div>
17     <div>forty-four words</div>
18   </div>
19 </div>
20 <div>
21   <div>five words</div>
22 </div>
23
24 <!-- In this example, the use of &lt;div&gt; and &lt;span&gt; elements provides no semantic meaning or context about the conter
```

Accessibility and Machine-Readability

This highlights how semantic markup improves accessibility and machine-readability. It shows examples of how browser developer tools display the Accessibility Object Model (AOM) differently for non-semantic and semantic markup. Assistive technologies like screen readers rely on the AOM to interpret the content structure and meaning correctly.

Roles and Landmarks

This explains the concept of roles and landmarks in semantic HTML. Semantic elements like <header>, <nav>, <main>, and <footer> have implicit roles that identify them as landmarks for assistive technologies. This helps users navigate the content more easily.

Using the `role` attribute

While semantic elements have implicit roles, the content mentions that the role attribute can be used to assign a specific role to any element. However, it recommends using the appropriate semantic element instead of relying on the role attribute whenever possible.

```
1   <div role="banner">
2     <span role="heading" aria-level="1">Three words</span>
3     <div role="navigation">
4       <a>one word</a>
5     </div>
6   </div>
```

Choosing the Right Elements

This emphasizes the importance of choosing the right HTML elements based on their semantic meaning and functionality, not just their visual appearance. It encourages developers to ask themselves, "Which element best represents the function of this section of markup?" when writing HTML.

> ⓘ **Info**
>
> In summary, this stresses the significance of using semantic HTML for improved accessibility, machine-readability, and overall content structure and meaning.

Heading & Sections

- `<header>` is used for introductory content at the top of a page, section, or article. This could include logos, titles, navigation menus, etc.

- `<nav>` is used to wrap major navigation blocks like menus.

- `<main>`

  represents the main content area of the page, unique to that specific page. There should only be one `<main>` per page.

- `<article>` is used for self-contained pieces of content that could be distributed or reused independently, like blog posts or news articles.

- `section` is used to group related content together, like chapters or sections of a guide or tutorial.

- `aside` holds tangentially related content, like sidebars or inserts, that are separate from the main content flow.

- `footer` is used for footer content like copyright notices, contact information, or related links at the bottom of a page, section, or article.

attribute name   attribute value

```
<a href = "http://info.cern.ch" >Link text</a>
```

Attributes

Boolean attributes

If a boolean attribute is present, it is always true. Boolean attributes include autofocus, inert, checked, disabled, required, reversed, allowfullscreen, default, loop, autoplay, controls, muted, readonly, multiple, and selected. If one (or more) of these attributes is present, the element is disabled, required, readonly, etc. If not present, it isn't.

```
1    <input required>
2    <input required="">
3    <input required="required">
```

Enumerated attributes

are sometimes confused with boolean attributes. They are HTML attributes that have a limited set of predefined valid values. Like boolean attributes, they have a default value if the attribute is present but the value is missing. For example, if you include `<style contenteditable>`, it defaults to

`<style contenteditable="true">` .

Global attributes

They are attributes that can be set on any HTML element, including elements in the `<head>`. There are more than 30 global attributes. While these can all, in theory, be added to any HTML element, some global attributes have no effect when set on some elements; for example, setting hidden on a `<meta>` as meta content is not displayed.

- `id` The global attribute id is used to define a unique identifier for an element. It serves many purposes, including: - The target of a link's fragment identifier. - Identifying an element for scripting. - Associating a form element with its label. - Providing a label or description for assistive technologies. - Targeting styles with (high specificity or as attribute selectors) in CSS.

- `class` The class attribute provides an additional way of targeting elements with CSS (and JavaScript), but serves no other purpose in HTML (though frameworks and component libraries may use them). The class attribute takes as its value a space-separated list of the case-sensitive classes for the element.

- `style` The style attribute enables applying inline styles, which are styles applied to the single element on which the attribute is set. The style attribute takes as its value CSS property value pairs, with the value's syntax being the same as the contents of a CSS style block: properties are followed by a colon, just like in CSS, and semicolons end each declaration, coming after the value.

- `tabIndex` The tabindex attribute can be added to any element to enable it to receive focus. The tabindex value defines whether it gets added to the tab order, and, optionally, into a non-default tabbing order.The tabindex attribute takes as its value an integer. A negative value (the convention is to use -1) makes an element capable of receiving focus, such as via JavaScript, but does not add the element to the tabbing sequence. A tabindex value of 0 makes the element focusable and reachable via tabbing, adding it to the default tab order of the page in source code order. A value of 1 or more puts the element into a prioritized focus sequence and is not recommended.

- `hidden` Hides an element from view, though it remains in the DOM.

- `inert` It is a boolean value that, when present, makes the browser "ignore" user input events for the element, including focus events and events from assistive technologies. The browser may also ignore page search and text selection in the element.

- `lang` Specifies the language of the element's content.

- `aria-*` (ARIA attributes) ARIA attributes are used to enhance the accessibility of web content for users with disabilities. They provide additional information about the semantics and behavior of elements, making it easier for assistive technologies to interpret and interact with the content.

Others include `dir`, `draggable`, `spellcheck`, `title`, `accesskey`.

Contd(Attribute)

- `role` The role attribute can be used to provide semantic meaning to content, enabling screen readers to inform site users of an object's expected user interaction.

```
1   <div role="banner">
2     <span role="heading" aria-level="1">Three words</span>
3     <div role="navigation">
4       <a>one word</a>
5       <a>one word</a>
6     </div>
7   </div>
```

- More on `aria-*` WAI-ARIA states and properties are used to enhance the accessibility of web content for users with disabilities. A list item `html:li` has been used to create a checkable menu item, and JavaScript events will capture mouse and keyboard events to toggle the value of aria-checked. A role is used to make the behavior of this simple widget known to the user agent.

```
1   <li role="menuitemcheckbox" aria-checked="true">Sort by Last Modified</li>
```

```
1   [aria-checked="true"] { font-weight: bold; }
2   [aria-checked="true"]::before { background-image: url(checked.gif); }
```

Contd-2

- `contenteditable` An element with the contenteditable attribute set to true is editable, is focusable, and is added to the tab order as if tabindex="0" were set. Contenteditable is an enumerated attribute supporting the values true and false, with a default value of inherit if the attribute is not present or has an invalid value.

These three opening tags are equivalent:

```
1    <style contenteditable>
2    <style contenteditable="">
3    <style contenteditable="true">
```

If you include `<style contenteditable="false">`, the element is not editable (unless it's by default editable, like a `<textarea>`). If the value is invalid, such as `<style contenteditable="😀">` or `<style contenteditable="contenteditable">`, the value defaults to inherit.

Contd-3

- custom attribute `data-*` You can create any custom attribute you want by adding the `data-` prefix. You can name your attribute anything that starts with `data-` followed by any lowercase series of characters that don't start with xml and don't contain a colon (:). They are used to store extra information about an element. `data-*` attributes allow us to store extra information on standard, semantic HTML elements without other hacks such as non-standard attributes, or extra properties on DOM.

```
1   <blockquote data-machine-learning="workshop"
2     data-first-name="Blendan" data-last-name="Smooth"
3     data-formerly="Margarita Maker" data-aspiring="Load Balancer"
4     data-year-graduated="2022">
5     HAL and EVE could teach a fan to blow hot air.
6   </blockquote>
7
8   <div data-user-id="12345" data-role="admin"></div>
9
10  <article id="electric-cars" data-columns="3" data-index-number="12314" data-parent="cars"> </article>
```

> 💡 Tip
>
> Attributes created with data-* can be accessed in JavaScript with `dataset` property of an element

Text Basics

- `h1` to `h6` are used for headings, with `h1` being the highest level. There should only be one `h1` per page, with subsequent headings following a logical hierarchy (e.g., `h1`, `h2`, `h3`, `h4`, `h5`, etc.).

- `p` is used for regular paragraph text.

- `blockquote` is used for longer quoted text, often from another source. It can optionally contain a `cite` element for attributing the source.

- `cite` is used for citing the source of a quote, reference, or other creative work.

- HTML entities are special character codes that start with an ampersand (&) and end with a semicolon (;). They are used to display characters that are reserved in HTML, or that are not present on the keyboard. `©` is an example of an HTML entity that displays the copyright symbol ©. `<` and `>` are used to display the less-than (<) and greater-than (>) symbols, respectively. `™` is used to display the trademark symbol ™.

  others include: `q`, `sub`, `sup`, `mark`, `i` `dd`, `dl`, `dt`, `figcaption`, `figure`, `hr`, `li`, `menu`, `pre`, `abbr`, `bdi`, `bdo`, `br`, `code`, `data`, `dfn`, `em`, `kbd`, `rp`, `rt`, `ruby`, `s`, `del`, `ins`, `samp`, `small`, `span`, `strong`, `sub`, `time`, `u`, `var`, `wbr`

```html
1  <section class="feedback" id="feedback">
2    <h2>What it's like to learn good and do other stuff good too</h2>
3    <ul>
4      <li>
5        <blockquote>
6          Two of the most experienced machines and human controllers teaching a class?
7          Sign me up! HAL and EVE could teach a fan to blow hot air.
8          <br>
9          If you have electricity in your circuits and want more than to just
10         fulfill your owner's perceived expectation of you, learn the skills to take over the world.
11         This is the team you want teaching you!
12       </blockquote>
13       <p>--Blendan Smooth,<br> Former Margarita Maker, <br> Aspiring Load Balancer</p>
14     </li>
15     <li>
16       <blockquote>
17       Hal is brilliant. Did I mention Hal is brilliant?
18       He didn't tell me to say that.
19       He didn't tell me to say anything. I am here of my own free will.
20       </blockquote>
21       <p>--Hoover Sukhdeep,<br>
22         Former Sucker, <br>
23         Aspiring DDoS Cop</p>
24     </li>
25   </ul>
26 </section>
```

Link

- `<a href="url">Link Text</a>` is used for creating hyperlinks, with the `href` attribute specifying the URL or file path.
- The `target` attribute controls how the link is opened, like `_self` for the same window or `_blank` for a new window/tab. download attribute is used to download the linked file instead of opening it.
- It's important to use descriptive link text that makes sense out of context, like "Read more about accessibility" instead of "Click here".
- Fragment Identifiers are used to link to a specific section of a page using id attribute as href like `<a href="#section">Link Text</a>`.
- Most website will have a skip to content link at the top of the page to allow keyboard users to skip the navigation and go straight to the main content. This is done by linking to the main content with a fragment identifier like `<a href="#main-content">Skip to main content</a>`.
- Absolute URLs include the full address of the page, like `<a href="https://www.oluwasetemi.dev">Link Text</a>`. Relative URLs are shorter and only include the path from the current page, like `<a href="/about">Link Text</a>`. Relative URLs are often used for internal links within the same website.

Navigation

- `<nav>` is used to wrap major navigation blocks like menus, as mentioned earlier.

- `<ul>` is used for unordered lists, which are typically displayed with bullet points. `<ol>` is used for ordered lists, which are typically displayed with numbers or other ordered indicators. `<li>` is used for individual list items within `<ul>` or `<ol>` elements.

```
 1    <nav aria-label="breadcrumbs">
 2      <ol role="list">
 3        <li>
 4          <a href="/">Home</a>
 5        </li>
 6        <li>
 7          <a href="/learn">Learn</a>
 8        </li>
 9        <li>
10          <a href="/learn/html">Learn HTML!</a>
11        </li>
12        <li aria-current="page">
13          Navigation
14        </li>
15      </ol>
16    </nav>
```

```
<a href="#main" class="visually-hidden">Skip to main</a>
<!-- fragement identifier -->
<nav aria-label="Breadcrumbs">
  <!--  role="list" was added back in because some CSS
display property values remove the semantics from some
elements.  -->
  <ol id="top" role="list">
    <li><a href="#">Ho<span>me</span></a></li>
    <li><a href="/living">Living Room</a></li>
    <li><a aria-current="page"
href="/living/couch">Couches</a></li>
    <li><a href="/living/couch/sectional">Sectional</a>
</li>
  </ol>
</nav>
<main id="main">
  Here is the main content

  <footer>I'm at the bottom<span><a href="#top">Got to the
Top</a></span></footer>
</main>
```

HTML Tables

HTML tables are used for displaying tabular data with rows and columns. They provide a semantic way to structure and present data that needs to be compared, sorted, calculated, or cross-referenced.

Table Structure

A table is defined using the <table> element, which wraps all the table content. Inside the <table>, you can have the following elements:

- `<caption>` : Provides a descriptive title for the table.

- `<thead>` : Contains the table header rows.

- `<tbody>` : Contains the table body rows.

- `<tfoot>` : Contains the table footer rows (optional).

Within these sections, you'll use `<tr>` for table rows and `<th>` for table header cells or `<td>` for table data cells

```html
<table>
  <caption>Student Grades</caption>
  <thead>
    <tr>
      <th>Name</th>
      <th>Grade</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>John</th>
      <td>85</td>
    </tr>
    <tr>
      <th>Emily</th>
      <td>92</td>
    </tr>
  </tbody>
</table>
```

Accessibility and Semantics

Using proper table structure and semantic elements is crucial for accessibility. Screen readers and assistive technologies rely on this structure to understand the tabular data and its relationships.

`<th>` cells have implicit ARIA roles of columnheader or rowheader, depending on the scope attribute. The scope attribute can be set to col, row, colgroup, or rowgroup to explicitly define the header's scope. The headers attribute can be used to associate data cells with their corresponding header cells in complex tables.

Merging cells

Similar to MS Excel, Google Sheets, and Numbers, it is possible to join multiple cells into a single cell. This is done with HTML! The colspan attribute is used to merge two or more adjacent cells within a single row. The rowspan attribute is used to merge cells across rows, being placed on the cell in the top row.

Styling and Responsiveness

Tables can be styled using CSS, but it's recommended to avoid using deprecated attributes like cellpadding, cellspacing, or align. Instead, use modern CSS properties like border-collapse, border-spacing, and caption-side.

```
1   <table>
2     <caption>Alt Alumni</caption>
3     <thead>
4       <tr>
5         <th rowspan="2" id="name" scope="col">Name</th>
6         <th colspan="2" id="path">Career path</th>
7         <th rowspan="2" id="year">Year</th>
8       </tr>
9       <tr>
10        <th id="past" scope="col">Past</th>
11        <th id="future" scope="col">Destiny</th>
12      </tr>
13    </thead>
14    <tbody>
15      <tr>
16        <th id="hal" scope="row">Hal Gibrah</th>
17        <td headers="hal path past">Calculator</td>
18        <td headers="hal path future">Mars rover</td>
19        <td>2020</td>
20      </tr>
21      <tr>
22        <th id="cathy" scope="row">James Bond</th>
23        <td headers="cathy path past">Waste disposal</td>
24        <td headers="cathy path future">Automated teller</td>
```

Form

The HTML <form> element identifies a document landmark containing interactive controls for submitting information. Nested in a <form> you'll find all the interactive (and non-interactive) form controls that make up that form.

- Forms are created using the <form> element, which contains interactive controls for submitting information. The <form> element has attributes like action (URL for processing the form data) and method (HTTP method for submission, e.g., GET or POST).

- Form controls, such as input fields, radio buttons, checkboxes, and submit buttons, are nested within the <form> element.

- HTML attributes can enforce required fields, define validation criteria, and prevent form submission until the data matches the required criteria.

- Submitting a form is typically done by activating a submit button, which sends the form data as name/value pairs to the specified URL.

Form(Radio-Button)

- Radio buttons in a group share the same name attribute, which ensures that only one can be selected at a time.
- Each radio button should have a unique value attribute to identify the selected option.
- To pre-select a radio button, include the checked attribute.
- To make a selection from a group of radio buttons required, add the required attribute to at least one radio button in the group.

```html
<fieldset>
  <legend>Who is your favorite student?</legend>
  <ul>
    <li>
      <label>
        <input type="radio" value="blendan" name="machine"> Blendan Smooth
      </label>
    </li>
    <li>
      <label>
        <input type="radio" value="hoover" name="machine"> Hoover Sukhdeep
      </label>
    </li>
    <li>
      <label>
        <input type="radio" value="toasty"  name="machine"> Toasty McToastface
      </label>
    </li>
  </ul>
</fieldset>
```

form(Checkboxes)

▪ Checkboxes with the same name in a group are submitted together, allowing multiple selections.

▪ If no value attribute is provided for a checkbox, the value defaults to "on", which may not be helpful.

▪ To make a checkbox required, add the "required" attribute to that specific checkbox.

Form(Label & fieldsets)

▪ Every form control should have an associated <label> element, either explicitly using the for attribute or implicitly by nesting the control within the <label> tags.

▪ Labels provide accessible names for form controls and increase the clickable area for better usability.

▪ Groups of related form controls, like radio buttons or checkboxes, should be grouped within a <fieldset> element, with a <legend> providing the label for the group.

▪ <fieldset> elements can be nested to create hierarchical groupings.

```
1    <label for="full_name">Your name</label>
2    <input type="text" id="full_name" name="name">
```

Input types & dynamic Keyboards

- There are 22 different input types in HTML, each optimized for a specific kind of data entry (e.g., text, email, url, tel, number, date, etc.).
- On devices with dynamic keyboards (e.g., smartphones), the input type determines the type of keyboard displayed, making data entry more efficient and accurate.

Accessing the Microphone and Camera

- The `<input type="file">` element allows users to upload files of specific types, defined by the accept attribute.
- The capture attribute, when set to "user" or "environment", allows users to directly capture media from their device's camera or microphone.
- This feature enables creating new media files within a form, without requiring a separate file upload.

Built-in Validation

- HTML attributes like required, pattern, min, max, minlength, and maxlength enable defining validation criteria for form controls.
- When a user attempts to submit a form, client-side constraint validation checks if the entered values meet the defined criteria.
- If any values are invalid, form submission is blocked, and the browser displays an error message in the first incorrect form control, giving it focus.
- CSS pseudo-classes like , , , and can be used to style form controls based on their validation state.
- JavaScript can be used to provide custom error messages during constraint validation or enhance the user experience with dynamic updates.

Example

- This example includes a nested `<form>` with input fields (text and number), a `<select>` dropdown, and two submit buttons.
- One submit button closes the dialog without submitting data (using formmethod="dialog" and formnovalidate).
- The other submit button submits the form data via POST to a specified URL (thankyou.php), after client-side validation.
- The input fields have the required attribute, and the number input has a defined step value.
- This example showcases implicit labels, instructions for form controls, and the potential for customizing error messages using JavaScript.

```
1   <dialog open aria-labelledby="dialogid">
2     <form action="thankyou.php">
3       <button type="submit" aria-label="close"
4        formmethod="dialog" formnovalidate>X</button>
5       <h2 id="dialogid">Application</h2>
6       <p>All fields are required</p>
7       <p>
8         <label>Name:
9           <input type="text" name="name" required />
10        </label>
11      </p>
12      <p>
13        <label>Warranty:
14          <input type="number" min="0" max="10"
15           name="warranty" required />
16        </label>
17      </p>
18      <p>
19        <label>Power source:
20          <select name="powersoure">
21            <option>AC/DC</option>
22            <option>Battery</option>
23            <option>Solar</option>
```

HTML Images Syntax

Images can improve the design and the appearance of a web page.

The HTML `<img>` tag is used to embed an image in a web page.

Images are not technically inserted into a web page; images are linked to web pages. The `<img>` tag creates a holding space for the referenced image.

The `<img>` tag is empty, it contains attributes only, and does not have a closing tag.

The `<img>` tag has two required attributes:

- src - Specifies the path to the image
- alt - Specifies an alternate text for the image

```
1    <img src="url" alt="alternatetext">
```

The src Attribute

The required src attribute specifies the path (URL) to the image.

Note: When a web page loads, it is the browser, at that moment, that gets the image from a web server and inserts it into the page. Therefore, make sure that the image actually stays in the same spot in relation to the web page, otherwise your visitors will get a broken link icon. The broken link icon and the alt text are shown if the browser cannot find the image.

Example:

```
1    <img src="img_chania.jpg" alt="Flowers in Chania">
```

Image as a Link

To use an image as a link, put the `<img>` tag inside the `<a>` tag:

Example:

```
1    <a href="default.asp">
2      <img src="smiley.gif" alt="HTML tutorial" style="width:42px;height:42px;">
3    </a>
```

The alt Attribute

The required alt attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

The value of the alt attribute should describe the image:

```
1    <img src="wrongimagelink.gif" alt="Flowers in Chania">
```

If a browser cannot find an image, it will display the value of the alt attribute:

> 💡 **Tip**
>
> You can use the width and height attributes in the image tag to define the width and heigh of the image in pixels.

Example:

```
1    <img src="img_girl.jpg" alt="Girl in a jacket" width="500" height="600">
```

picture

Picture element contains zero or more `<source>` elements and one `<img>` element to offer alternative versions of an image for different display/device scenarios.

The browser considers each child source element and choose the best match among them, if no matches are found the URL of the image src attribute is selected. The selected image is then presented in the space occupied by the image element.

The media attribute specifies a media condition similar to a media query. If the media attribute is present, the `<source>` element will only be used if the media condition is true.

The srcset attribute allows you to specify different image sources for different display scenarios. The browser will choose the best image source based on the display characteristics of the device using a width descriptor `w`, pixel density descriptor `x`, or a combination of the two.

The type attribute specifies the MIME type of the resource. The browser will only use the source if it supports the MIME type.

```
1    <picture>
2      <source srcset="mdn-logo-wide.png" media="(min-width: 600px)" />
```

HTML `<audio>` Tag

The `<audio>` tag is used to embed sound content in a document, such as music or other audio streams.

The `<audio>` tag contains one or more `<source>` tags with different audio sources. The browser will choose the first source it supports.

The text between the `<audio>` and `</audio>` tags will only be displayed in browsers that do not support the `<audio>` element.

Note: There are three supported audio formats in HTML: MP3, WAV, and OGG.

Example:

```
1    <audio controls>
2      <source src="horse.ogg" type="audio/ogg">
3      <source src="horse.mp3" type="audio/mpeg">
4      Your browser does not support the audio tag.
5    </audio>
```

Audio Tag Attributes

Code Example:

```
1    <body>
2        <h1>Audio Tag with Various Attributes</h1>
3        <audio controls autoplay loop muted preload="auto" crossorigin="anonymous">
4            <source src="audiofile.mp3" type="audio/mpeg">
5            <source src="audiofile.ogg" type="audio/ogg">
6            Your browser does not support the audio element.
7        </audio>
8    </body>
```

`<audio>` : The audio element that includes multiple attributes:

- controls: Adds playback controls.
- autoplay: The audio will play automatically when ready.
- loop: The audio will loop continuously.
- muted: The audio will be muted initially.
- preload: Specifies that the audio should be preloaded.

Contd.

- crossorigin: Specifies how the element handles cross-origin requests.
- <source> Defines multiple sources for the audio file in different formats (MP3 and OGG) for better compatibility.

Note: In HTML, attributes provide additional information about an element and modify its behavior or appearance.

HTML `<video>` Tag

`<video>` : The Video Embed element The `<video>` HTML element embeds a media player which supports video playback into the document. You can use `<video>` for audio content as well, but the `<audio>` element may provide a more appropriate user experience.

Video Tag Attributes

```html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Video Tag Example</title>
7  </head>
8  <body>
9      <h1>Video Tag with Various Attributes</h1>
10     <video controls autoplay loop muted preload="auto" crossorigin="anonymous">
11         <source src="videofile.mp4" type="video/mp4">
12         <source src="videofile.webm" type="video/webm">
13         Your browser does not support the video element.
14     </video>
15 </body>
16 </html>
```

Code Explanation:

- `<!DOCTYPE html>` : Declares the document type and version of HTML.

- `<html lang="en">` : Sets the language of the document to English.

- `<head>` : Contains meta-information about the document, including character set and viewport settings.

- `<title>` : Sets the title of the document, which appears in the browser tab.

- `<body>` : Contains the content of the document.

- `<h1>` : A heading element for the title of the page.

- `<video>` : The video element that includes multiple.

  - attributes:
  - `controls` : Adds playback controls.
  - `autoplay` : The video will play automatically when ready.
  - `loop` : The video will loop continuously.
  - `muted` : The video will be muted initially.
  - `preload` : Specifies that the video should be preloaded.
  - `crossorigin` : Specifies how the element handles cross-origin requests.

Cont'd Code Explanation

- <source>: Defines multiple sources for the video file in different formats (MP4 and WebM) for better compatibility.
- Fallback text: "Your browser does not support the video element." This text will be displayed if the browser does not support the <video> element.

HTML API

For us to access and manipulate documents we need the DOM(Document Object Model) and this is also an example of API. The DOM is the tree of all the nodes in the document. Some nodes can have children, others can't. The tree includes elements, along with their attributes, and text nodes.

The browser provides numerous APIs providing natively supported methods, events, and property querying and updating. Element nodes contain information about all the attributes set on the element. You can use HTML interfaces to access information about an element's attributes. For example, we can use `HTMLImageElement.alt` get the alt attributes of all the images:

```
1    let allImages = document.querySelectorAll('img');
2    allImages.forEach((imageInstance) => {
3      console.log(imageInstance.alt);
4    });
```

The HTML interface APIs is not limited to accessing attribute values. The DOM provides insight into the current state of the UI. HTML APIs can access all of that information. You can access the length of a video, where a view is in the current playback, and if the video (or audio) has finished playing with `HTMLMediaElement.duration`, `HTMLMediaElement.currentTime`, and `HTMLMediaElement.ended` respectively.

List of HTML API interfaces

- HTMLAnchorElement - `<a>`
- HTMLAreaElement - `<area>`
- HTMLAudioElement - `<audio>`
- HTMLBaseElement - `<base>`
- HTMLButtonElement - `<button>`
- HTMLCanvasElement - `<canvas>`
- HTMLDataElement - `<data>`
- HTMLDataListElement - `<datalist>`
- HTMLDetailsElement - `<details>`
- HTMLDialogElement - `<dialog>`

- HTMLEmbedElement - `<embed>`
- HTMLFieldSetElement - `<fieldset>`
- HTMLFormElement - `<form>`
- HTMLHtmlElement - `<html>`
- HTMLIFrameElement - `<iframe>`
- HTMLImageElement - `<img>`
- HTMLInputElement - `<input>`
- HTMLLabelElement - `<label>`
- HTMLLegendElement - `<legend>`
- HTMLLIElement - `<li>`

- HTMLLinkElement - `<link`
- HTMLMapElement - `<map>`
- HTMLMediaElement - `<audio>, <video>`
- HTMLMenuElement - `<menu>`
- HTMLMetaElement - `<meta>`
- HTMLModElement - `<ins>, <del>`
- HTMLMeterElement - `<meter>`
- HTMLObjectElement - `<object>`
- HTMLOListElement - `<ol>`
- HTMLOptGroupElement- `<optgroup>`
- HTMLOptionElement - `<option>`

- HTMLOutputElement - `<output>`
- HTMLPictureElement - `<picture>`
- HTMLProgressElement - `<progress>`
- HTMLQuoteElement - `<q>`, `<blockquote>`, `<cite>`
- HTMLScriptElement - `<script>`
- HTMLSelectElement - `<select>`
- HTMLSlotElement - `<slot>`
- HTMLSourceElement - `<source>`
- HTMLStyleElement - `<style>`
- HTMLTableCellElement - `<td>`, `<th>`

- HTMLTableColElement - `<col>, <colgroup>`
- HTMLTableElement - `<table>`
- HTMLTableRowElement - `<tr>`
- HTMLTableSectionElement - `<thead>, <tbody>, <tfoot>`
- HTMLTemplateElement - `<template>`
- HTMLTextAreaElement - `<textarea>`
- HTMLTimeElement - `<time>`
- HTMLTitleElement - `<title>`
- HTMLTrackElement - `<track>`
- HTMLVideoElement - `<video>`

Focusing

To improve user-accessibility in our code we have to put focus into consideration by ensuring that user knows which element has focus and this can be achieved by including `:focus`, `:focus-visible` or `:focus-within` styles on the element.

Interactive elements, including form controls, links, and buttons, are by default focusable and tabbable. Tabbable elements are part of the document's sequential focus navigation order. Other elements are `inert`, meaning they are not interactive. With HTML attributes, it is possible to make interactive elements `inert` and to make `inert` elements interactive.

Focus

By default, the navigation focus order in a webpage follows the visual and source code order. Although HTML attributes and CSS properties can change this order, doing so can negatively impact user experience. Modifying the tabbing order or visual rendering order can lead to confusion and a poor user experience. Therefore, it's recommended not to alter the perceived and actual tabbing order with CSS and HTML, as demonstrated by examples showing the negative effects of such changes.

Example 1

Click in any input, then hit the tab key.

1. [　　　]　　2. [　　　]　　3. [　　　]　　4. [　　　]

5. [　　　]　　6. [　　　]　　7. [　　　]　　8. [　　　]

9. [　　　]　　10. [　　　]　　11. [　　　]　　12. [　　　]

Example 2

In this example, CSS has created a divergence between the tabbing order and the visual order of the content:

Put your cursor in the text box, then hit your tab key several times:

This sentenced was written in reverse order then styled with CSS flexbox.

The flex-flow: row-reverse; declaration has reversed the visual order. In addition, the CSS order property was applied to the sixth word, "This", which visually moved that one word. The tabbing sequence is the order of the code, which no longer matches the visual order, creating a disconnect for keyboard users.

Note: The `contenteditable` and `tabindex` attributes, being global attributes, can be added to any element, making them focusable in the process. Focusable elements can also be focused with a mouse or pointer, by having the autofocus attribute set, or by script, such as with `element.focus()`. A `tabindex` attribute with a negative value makes the element focusable but not tabbable.

▶ `HTML Codes for the Example 1`

▶ `HTML Codes for the Example 2`

Details and Summary

Have you heard of a disclosure widget or come across it in a website before?A disclosure widget, also known as an expandable or collapsible section, is a UI component that allows users to hide or show contents.Mostly, developers achieve this accordion trick using CSS or JavaScript but we can easily get this done with these tags: <details> and <summary>

The `<details>` and `<summary>` elements are all you need: they are a built-in way to handle expanding and collapsing content. When a user clicks or taps a `<summary>`, or releases the `Enter` key when the `<summary>` has focus, the contents of the parent `<details>` toggle to visible!

```
1    <details>
2    <summary>AltSchool offers these courses</summary>
3    <!-- details -->
4    </details>
```

Creating an accordion with just HTML

Workshop reviews:
▶ Blendan Smooth
▶ Hoover Sukhdeep

Grouped Details and Summary or Nested Details and Summary or Accordion

This can be achieved by grouping the `<details>` elements with a `name` attribute. This way, only one `<details>` element can be open at a time.

Workshop reviews:

▼ Blendan Smooth

Two of the most experienced machines and human controllers teaching a class? Sign me up! HAL and EVE could teach a fan to blow hot air. If you have electricity in your circuits and want more than to just fulfill your owner's perceived expectation of you, learn the skills to take over the world. This is the team you want teaching you!

▶ Hoover Sukhdeep
▶ AltSchool offers these courses

Toggling visibility: the open attribute

The `<details>` element is the disclosure widget container. The `<summary>` is the summary or legend for its parent `<details>`. The summary is always displayed, acting as a button that toggles the display of the rest of the parent's contents. Interacting with the `<summary>` toggles the display of the self-labeled summary siblings by toggling the `<details>` element's open attribute.

The open attribute is a boolean attribute. If present, no matter the value or lack thereof, it indicates that all the `<details>` contents are shown to the user. If the open attribute is not present, only the contents of the `<summary>` are shown.

Because the open attribute is added and removed automatically as the user interacts with the control, it can be used in CSS to style the element differently based on its state.

Toggling the summary marker

If we pay attention to the disclosure widget, we will notice that there is an arrow to the inline-start of the summary. This arrow is a ::marker set on the <summary> element. You can style the disclosure triangle with CSS, including changing the marker used from a triangle to any other bullet type, including an image with list-style-image.

```
1    details summary::before {
2      /* all the styles */
3    }
4    details[open] summary::before {
5      /* changes applied when open only */
6    }
```

Remember, `<details>` and `<summary>` can be heavily styled and can even be used to create tool tips. But, if you're going to use these semantic elements for use cases in which the native semantics are a mismatch, always ensure that you maintain accessibility. HTML for the most part is by default accessible. Our job as developers is to ensure our content stays accessible.

Dialogs and Popovers

Have you seen the common dialog box on computers, websites etc? You can achieve that using the <dialog> element as this makes it easy to create popup dialogs and modals on a web page. Note: A modal which is also known as modal window or lightbox is a web page element that displays in front of and deactivates all other pages you have to perform the action requested by the modal or close it if you want to have access to your main content. Also, we have the non-modal which when pops up on the screen it gives users access to interact with content outside the box.

Modal Dialogs

Let's see how modal `<dialog>` works

Open Modal

Contd.

Dialogs are mostly used in cases that requires the immediate attention of the site user, it might be to convey important messages of notifications that requires user acknowledgment just like the `alert()` but dialogs provides more accessibility options and flexibility.

Also, we can use dialogs to seek confirmation from a user before proceeding with a particular action that may have some consequences.

Note: Non-modal dialogs opens a dialog as it name implies but without adding a backdrop which makes the background active
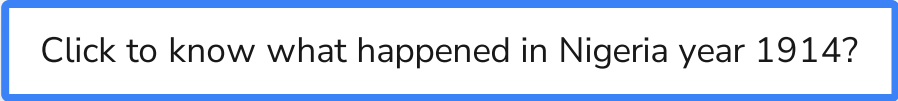
Popovers

Before we delve into popovers, I want you to understand that popovers are special because they allow users to interact with both the popover and the underlying content simultaneously. Though this can be achieved with non-modal dialogs, popovers are more lightweight and can be used for quick interactions, such as displaying tooltips, additional information, or menus, without disrupting the user's workflow. They are context-sensitive and can be dismissed easily, providing a seamless user experience.

We are going to follow these steps to create a popover

- Firstly, we will create a button to trigger the popover and an element(what we want to display) to trigger.

- We will set a `popover` attribute on the element which is going to be the popover(element to display).

- Then, we are going to add a unique `id` on the popover element(element to display).

- Lastly, to connect the button to the popover, we will set the buttons's `popovertarget` to the value of the popover element's id.

Popover Sample

Let's see how our `<popover>` works following the previous algorithm.

Click to know what happened in Nigeria year 1914?

JS-Free hamburger menu demo
With the Popover API

Click open the hamburger menu to see a demo of JavaScript-free interaction handling! By using the `popover` attribute, you can allow the browser to handle the keyboard management (including navigation via `esc`, `spacebar`, and `enter`), optional light-dismiss (clicking outside the boundaries of the popover), and click handlers such as on the open and close buttons.

When to use Dialogs and when to use Popovers

Dialogs are used when you need the full attention of the user, especially for critical alerts, confirmation prompts, or scenarios where user flow needs to be strictly controlled.

Developers should use popovers when they want to provide supplementary information about an activity without disrupting the user's workflow, because popovers are lightweight, allowing users to interact with both the popover and the underlying content simultaneously.

Popover Types

Sometimes, you might want to have more control over your popover, and this is where setting the `<popover>` attribute value to manual comes in. Previously we didn't set any value to the `<popover>` attribute, we just used it directly which explicitly means `<popover="auto">` and this allows the popover to close when we press the `esc` key or click outside the popover box in the UI.

To have control over our `<popover>` we are going to set the `<popover>` attribute value to manual just like this `<popover="manual">`, by doing this we will have to add a close button to control the closing of the popover because clicking away in the UI won't work any longer.

Popover with value set to manual

Let's see how our `<popover>` works following the previous algorithm.

Click to know what happened in Nigeria in 1914

Show popover dialog

```html
<dialog id="popover-dialog" popover="">
  Popover Dialog
  <br>
  <button">Close Dialog</button>
</dialog>
```

```html
<dialog id="modal-dialog">
  Modal Dialog
  <br>
  <form method="dialog">
    <button>Close Dialog</button>
  </form>
</dialog>
```

Show modal dialog

What are Web Components?

Web components are a set of web standards that allow developers to create reusable, self-contained UI elements. These components can be seamlessly integrated into existing applications, just like regular HTML elements. The Web Component standard comprises three main parts:

- HTML Templates: The `<template>` element allows developers to declare fragments of HTML that can be cloned and inserted into the DOM using JavaScript. The contents of the `<template>` element are not rendered by default.
- Custom Elements: Custom Elements allow developers to define their own HTML elements with custom functionality. These elements can be created by extending the HTMLElement class using JavaScript.
- Shadow DOM: The Shadow DOM is an encapsulated DOM tree that is attached to a custom element. It provides a way to scope CSS styles and DOM structures to a specific component, isolating it from the rest of the document. This prevents naming conflicts and style clashes with the rest of the application.

The <template> Element

This section introduces the `<template>` element and demonstrates how to create a template for a star rating component. It also explains the concept of unnamed and named slots using the `<slot>` element.

- The `<template>` element is used to declare HTML fragments that can be cloned and inserted into the DOM using JavaScript. The contents of the `<template>` element are not rendered by default. In the given example, a template is created for a star rating component with a `<form>` element containing radio inputs and buttons.

Example(template)

```
1   <template id="star-rating-template">
2     <form>
3       <fieldset>
4         <legend>Rate your experience:</legend>
5         <rating>
6           <input
7             type="radio"
8             name="rating"
9             value="1"
10            aria-label="1 star"
11            required
12          />
13          <input type="radio" name="rating" value="2" aria-label="2 stars" />
14          <input type="radio" name="rating" value="3" aria-label="3 stars" />
15          <input type="radio" name="rating" value="4" aria-label="4 stars" />
16          <input type="radio" name="rating" value="5" aria-label="5 stars" />
17        </rating>
18      </fieldset>
19      <button type="reset">Reset</button>
20      <button type="submit">Submit</button>
21    </form>
22  </template>
```

Rate your experience:

Reset   Submit

Shadow DOM and Styling

This discusses the Shadow DOM and how it encapsulates CSS styles within a web component. It demonstrates how to apply styles to the shadow DOM and explains the usage of the and ::slotted() pseudo-classes.

- The Shadow DOM provides a way to scope CSS styles to a specific web component, isolating it from the rest of the document. This means that external CSS does not apply to the component, and component styles have no effect on the rest of the document, unless intentionally directed.

Another Example

- In the given example, a `<style>` element is included within the `<template>` to apply styles to the star rating component. These styles are encapsulated within the shadow DOM and do not affect the rest of the document.
- The `:host` pseudo-class is used to select the shadow host element (the custom element to which the shadow DOM is attached). The `::slotted()` pseudo-element is used to select slotted elements (elements inserted into named slots) from within the shadow DOM.
- The document also mentions the `::part()` pseudo-element, which allows styling elements within a shadow DOM from the global CSS scope. By adding a part attribute to elements in the `<template>`, those elements can be targeted using the `::part()` pseudo-element in the global CSS.

```html
1  <template id="card-template">
2    <style>
3      :host {
4        display: block;
5        margin-bottom: 20px;
6      }
7      .card {
8        border: 1px solid #ccc;
9        padding: 10px;
10       background-color: #f5f5f5;
11     }
12     ::slotted(h2) {
13       margin-top: 0;
14     }
15     ::slotted(p) {
16       color: #666;
17     }
18   </style>
19   <div class="card">
20     <slot name="card-header"></slot>
21     <slot name="card-content"></slot>
22   </div>
23 </template>
```

Slot

The <slot> element is used within the <template> to create placeholders for custom content. If a name attribute is provided, it creates a "named slot" that can be used to insert custom content within the web component. In the example, a named slot is created for the legend of the star rating component.

```
1  <template id="star-rating-template">
2    <form>
3      <fieldset>
4        <slot name="star-rating-legend">
5          <legend>Rate your experience:</legend>
6        </slot>
7      </fieldset>
8    </form>
9  </template>
```

Undefined Elements and Custom Elements

This explains how browsers handle undefined (unrecognized) elements and demonstrates how to define a custom element using JavaScript and the customElements.define() method.

- Browsers do not fail when encountering unrecognized HTML elements. Instead, they treat these elements as anonymous inline elements, similar to <span>>. In the given example, the <star-rating> element is initially treated as an unrecognized element, and its contents are displayed as if they were inside a <span> element.

- To define a custom element, JavaScript is required. The customElements.define() method is used to register a custom element by extending the HTMLElement class.

Undefined Elements and Custom Elements

- In the example, the star-rating custom element is defined, and a shadow DOM is attached to it using the attachShadow() method. The contents of the <template> element are cloned and appended to the shadow DOM, effectively encapsulating the star rating component.

```
1   customElements.define('star-rating',
2   class extends HTMLElement {
3     constructor() {
4       super(); // Always call super first in constructor
5       const starRating = document.getElementById('star-rating-template').content;
6       const shadowRoot = this.attachShadow({
7         mode: 'open'
8       });
9       shadowRoot.appendChild(starRating.cloneNode(true));
10    }
11  });
```

Do you know?

- Clipboard API provides the ability to respond to clipboard commands (cut, copy, and paste), as well as to asynchronously read from and write to the system clipboard.

```
1    navigator.clipboard
2      .readText()
3      .then(
4        (clipText) => (document.querySelector(".editor").innerText += clipText),
5      );
```

- `autocomplete` attribute provide hints to the browser about the type of data expected in the input field. e.g. name, honorific-prefix, tel, cc-number etc. `<input name="address_firstline" autocomplete="billing street-address" />`

- File System Access API provides access files and directories on the user's local device.

```
1    const handle = await window.showSaveFilePicker(opts);
```

- Badging API - set a badge on the web application's icon to notify about updated state in a less intrusive, persistent way. `navigator.setAppBadge(unreadCount)`

Do you know?

- `<datalist>` is used to provide a list of predefined options for an input element. The list attribute of the input element is used to associate the input with the datalist.

Choose a flavor:

```
1    <input name="country" list="countries">
2    <datalist id="countries">
3      <option>Afghanistan</option>
4      ...
5    </datalist>
```

- Web Share API exposes a mechanism for sharing content to various user-selected destinations. `navigator.share(shareData)`

- Launch Handler API allows PWAs to control how they are launched.

```
1    {
2      "launch_handler": {"client_mode": "navigate-new"}
3    }
```

Do you know?

- File Handling API allows PWAs to register themselves as handlers for certain file types or protocols.

```
1    {
2      "file_handlers": [{
3          "action": "/open-file",
4          "accept": {
5            "image/svg+xml": ".svg",
6            "image/png": ".png"
7          }
8      }]
9    }
```

- Window Controls Overlay API allows PWAs to display custom content over the title bar area, whose controls become an overlay. `"display_override": ["window-controls-overlay"]`

- Isolated Web Apps allow native like packaging, permission and signing updates for PWAs.

Do you know?

- HTML Media Capture allows users to capture media (audio, video, or images) using the device's camera or microphone. The capture attribute is used with the input element to specify the type of media to capture.
  `<input type="file" accept="video/*" capture>`

- `input.showPicker()` method is used to display the file picker dialog for the input element.

```
1    <input type="file" id="fileInput">
2    <button onclick="document.getElementById('fileInput').showPicker()">Select File</button>
```

- FormData API can be used tp easily extract and manipulate form data values via JS.

```
1    let fd = new FormData(form);
2    let data = JSON.stringify(Object.fromEntries(fd));
```

- Customizable Select allow styling control and customizing dropdown control, previously `<selectlist>` and `<selectmenu>` were used for this purpose.

```
1    select,
2      ::picker(select) {
3        appearance: base-select;
4      }
```

Do you know?

- The `:user-error` pseudo-class is used to style form elements that have invalid input. It is used to indicate that the user has made an error in the input field.

```css
1    input:user-error {
2      border-color: red;
3      background-color: #ffe6e6;
4    }
```

- plaintext-only value for contenteditable permits editing of the element's raw text, but not rich text formatting. `<h2 class="title" contenteditable="plaintext-only"></h2>`

- EditContext: The EditContext interface represents the text edit context of an element that was made editable by using the EditContext API.

```js
1    const canvas = document.createElement("canvas");
2    const editContext = new EditContext();
3    canvas.editContext = editContext;
```

- Lazy Loading: The loading attribute is used to specify whether an image/iframe should be loaded immediately or only when it is visible to the user. `<img src="picture.jpg" loading="lazy" />`

Do you know?

- The `caretPositionFromPoint()` method returns the caret's character offset. Check out this examples form MDN to see how it works.

```
1  const range = document.caretPositionFromPoint(e.clientX, e.clientY);
2  const textNode = range.offsetNode;
3  const offset = range.offset;
```

- Resource Hints (all) - allows work to begin on certain resources early to improve performance. `<link rel="pre* | dns-prefetch | modulepreload">`.

```
1  <link rel="preload" href="picture.jpg" />
2  <link rel="dns-prefetch" href="https://fonts.googleapis.com/" />
```

- Content-Security Policy (CSP) is an added layer of security that helps to detect and mitigate XSS and other attacks. `<meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://*; child-src 'none';">` `Content-Security-Policy: default-src 'self'`

- fetchpriority attribute is used to specify the priority of the fetch request. `<img src="picture.jpg" fetchpriority="high" />`

- `blocking="render"` attribute is used to specify that the resource should block rendering until it is loaded. `<script blocking="render" async src="async-script.js"></script>`

- `<model>` element allows embedding 3D graphical content into a webpage. `<model src="3d-assets/car"></model>`

- `<video src="fun.mp4" controlslist="nodownload"></video>`. The controlslist attribute is used to specify the controls that should be displayed in the video player.

- CSS Custom Hightlight API provides a mechanism for styling arbitrary text ranges on a document by using JavaScript to create the ranges, and CSS to style them.

```
1  ::highlight(my-custom-highlight) {
2    background-color: blue;
3  }
```

```
1  const parentNode = document.getElementById("foo");
2  const range1 = new Range();
3  range1.setStart(parentNode, 10);
4  range1.setEnd(parentNode, 20);
5  const highlight = new Highlight(range1);
6  CSS.highlights.set("my-custom-highlight", highlight);
```

- `setHtmlUnsafe()` is used to parse a string of HTML into a DocumentFragment, which then replaces the element's subtree in the DOM. `element.setHTMLUnsafe("<p>Unsafe HTML</p>")`;

Do you know?

- parseHtmlUnsafe() is used to parse a string of HTML, which may contain declarative shadow roots, in order to create a new Document instance. `document.parseHTMLUnsafe("<p>Unsafe HTML</p>")`

- Intl.Segmenter API is used to break text into segments based on the language-specific rules. `const segmenter = new Intl.Segmenter("en", {granularity: "word"});`

- HTML Modules allow import HTML files via JS imports and access their elements and JS exports. Support for JSON modules is also available.

```
1    <script type="module">
2      import { TabList } from "./tablist.html" with { type: 'html' };
3      customElements.define("tab-list", TabList);
4    </script>
```

- `focusgroup` facilitate keyboard focus navigation using the keyboard arrow keys among a set of focusable elements. `<div focusgroup="wrap horizontal">`

- search element is Semantic element for wrapping search UI… `<search>...</search>`

Assignments

- Assignment 1
- Assignment 2

Assignment 1

Create a web page that should display your basic information, with ALT SCHOOL ID, a biography, relevant details explaining why you joined the program, and your goals for the School of Engineering program. Implement a fragment identifier feature to enable scrolling back to the top of the page. Create a folder and name the file index.html

Create a web page consisting of a form that mirrors the ALT SCHOOL application form (https://portal.altschoolafrica.com/auth/create-account), name the file form.html, connect the form.html to your first assignment via global navigation ensuring both files are in the folder of the first assignment, which should be at the top of both pages. You should create a footer with your address and a copyright. It must also be accessible, incorporating appropriate semantic HTML tags.

```
1    index.html
2    form.html
```

NB: Strictly without CSS. Submission details will be sent to you in due time.

Assignment 2

Build additional two web pages , the first is going to be a table about all the courses and schools existing at AltSchool Africa, with School of engineering we have frontend engineering, backend engineering, cloud engineering and cybersecurity. School of Product - Product Design, Marketing, Management. School of Data - Data Analysis, Science, Engineering.

School	Courses		
SOE	Frontend	Backend	Cloud
SOD	Design	Marketing	Marketing

Second page is a page where you use all the media element existing in html to create powerful message about yourself. Specially use picture element with more that 3 sources with responsiveness for mobile, tablet and laptop.

```
1  table.html
2  media.html
```

Terminal

- File and Directory Management:

`ls` - List files and directories. `cd` - Change directory. `mkdir` - Create a new directory. `touch` - Create an empty file. `cp` - Copy files or directories. `mv` - Move or rename files or directories. `rm` - Remove files or directories (with caution).

- Viewing and Editing Files:

`cat`, `less` - View file contents. `nano`, `vi`, `vim` - Basic introduction to terminal-based text editors.

- Others

`echo` - Print text to the terminal, `man` - Display the manual for a command, `clear` - Clear the terminal screen, `chmod` - Change file permissions, `chown` - Change file ownership.

Git

- Version Control: A system that records changes to files over time, allowing you to recall specific versions later.

- Distributed Version Control System (DVCS): A version control system that allows multiple developers to work on a project simultaneously.

- Git: A popular DVCS that tracks changes to files and directories, allowing you to collaborate with others and manage your project's history.

- ○ GitHub: A web-based platform that hosts Git repositories and provides collaboration tools for developers.

- Repository (Repo): A directory that contains your project files and a .git directory, which tracks the history of changes.

- Clone: Creating a copy of an existing Git repository.

- Commit: A snapshot of changes in the repository. It represents a point in the history of your project.

- Branch: A separate line of development, allowing you to work on different features or fixes without affecting the main codebase.

- Merge: Combining changes from one branch into another.

- Remote: A version of your project hosted on the internet or network (e.g., GitHub, GitLab).

- Pull: Fetching changes from a remote repository and merging them into your local repository.

- Push: Sending your commits to a remote repository.

- Pull Request: A request to merge changes from one branch into another.

- Fork: Creating a personal copy of a repository on GitHub.

- Issue: A task, bug, or feature request associated with a repository on GitHub.

- Markdown: A lightweight markup language used to format text on GitHub.

- README: A file that provides information about a project, typically written in Markdown.
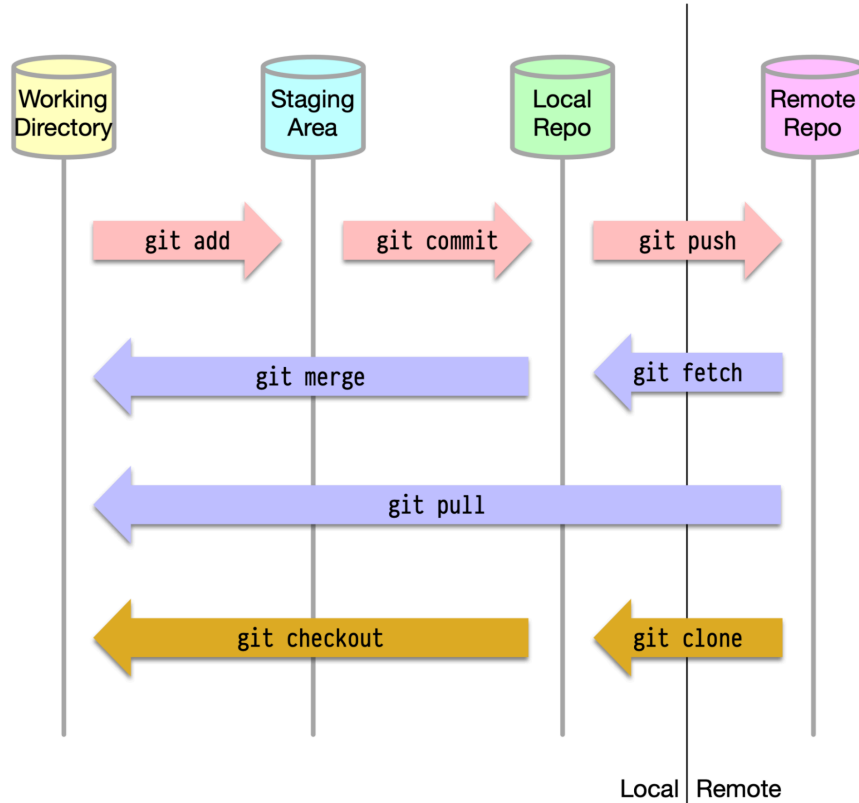
Git Commands

- `git init` - Initialize a new Git repository.
- `git clone` - Clone a repository into a new directory.
- `git add` - Add changes to the staging area.
- `git commit` - Record changes to the repository.
- `git status` - Show the working tree status.
- `git log` - Show commit logs.
- `git branch` - List, create, or delete branches.
- `git checkout` - Switch branches or restore working tree files.
- `git merge` - Join two or more development histories together.

- `git pull` - Fetch from and integrate with another repository or a local branch.
- `git push` - Update remote refs along with associated objects.
- `git remote` - Manage set of tracked repositories.
- `git fetch` - Download objects and refs from another repository.
- `git reset` - Reset current HEAD to the specified state.
- `git rebase` - Reapply commits on top of another base tip.
- `git revert` - Revert some existing commits.
- `git stash` - Stash the changes in a dirty working directory away.
- `git tag` - Create, list, delete or verify a tag object signed with GPG.

How Git Commands work

ByteByteGo.com

| Working Directory | Staging Area | Local Repo | Remote Repo |

git add → git commit → git push →

← git merge ← git fetch

← git pull

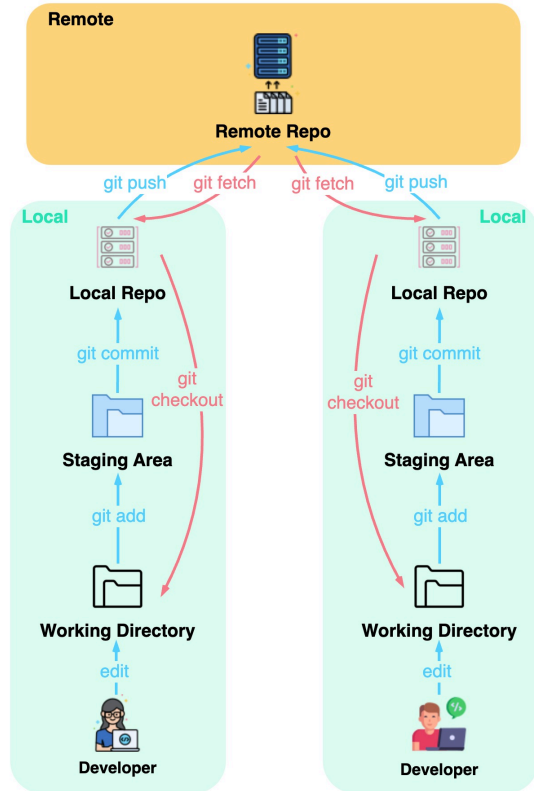← git checkout ← git clone

Local | Remote

Git maintains three local storages on our machine and one on a remote server like Github, which means that our code can be found in four places

- Working directory: where we edit files
- Staging area: a temporary location where files are kept for the next commit
- Local repository: contains the code that has been committed
- Remote repository: the remote server that stores the code

How does Git Work?   🗂 blog.bytebytego.com

Remote

Remote Repo

git push   git fetch   git fetch   git push

Local   **Local**

Local Repo   **Local Repo**

git commit   git commit

git checkout   git checkout

Staging Area   **Staging Area**

git add   git add

Working Directory   **Working Directory**

edit   edit

Developer   **Developer**

GitHub

- ⊙ GitHub is a web-based platform that hosts Git repositories and provides collaboration tools for developers. It allows you to store, manage, and share your code with others.

- ⊙ GitHub provides features such as pull requests, issues, and project boards to help you collaborate with your team and manage your projects more effectively.

- You can use ⊙ GitHub to host your code, track changes, and work on projects with other developers. It's a powerful tool for version control and project management.

Creating a Repository

- Either push an existing local repository to GitHub or create a new repository on ⊙ GitHub and clone it to your local machine.

```
1   git clone <repository-url>
2   gh repo create <repository-name>
3   git remote add origin <repository-url>
4   git push
```

Pull Requests

- A pull request is a way to propose changes to a repository on GitHub. It allows you to submit your changes for review and merge them into the main branch.

- When you create a pull request, you can compare the changes between two branches and request feedback from other developers. You can also discuss the changes, make additional commits, and address any feedback before merging the changes.

Creating a Pull Request

- To create a pull request, push your changes to a branch on GitHub and then open a pull request from that branch to the main branch.

```
1    git push origin <branch-name>
2    gh pr create
3    gh pr list
4    gh pr checkout <pull-request-number>
```

Open Source

- Open source software is software that is freely available to use, modify, and distribute. It is developed collaboratively by a community of developers who contribute their time and expertise to improve the software is used by individuals, businesses, and organizations around the world to build websites, applications, and other software products.

- Open source projects are typically hosted on platforms like GitHub, GitLab, and Bitbucket, where developers can access the source code, report issues, and contribute to the project.

Contributing and Creating Open Source Projects

- You can contribute to open source projects by fixing bugs, adding new features, and improving documentation. You can also create your own open source projects and share them with the community.

- Create tools, libraries, and frameworks that solve common problems and help other developers build better software. Open source projects can be a great way to showcase your skills, collaborate with others, and give back to the community.

Contributors

- RIDWANADEBOSIN
- OLUBEBE