DAY 1 Html, CSS, Javascript

HTML (Describes the structure of the web page)

Key Concepts:

- Document structure with <!DOCTYPE>, <html>, <head>, and <body> tags
- Semantic elements like headings (<h1> to <h6>)
- Text formatting with paragraph tags (<p>)
- Interactive elements like buttons
- Form creation with various input types

Code Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My First Page</title>
  <link rel="stylesheet" href="styles.css">
  <script src="script.js"></script>
</head>
<body>
  <div>
    <h1>Main Heading</h1>
    <p id="content">This is a paragraph.</p>
    <button id="myButton" onclick="handleClick()">Click Me</button>
  </div>
</body>
</html>
```

CSS (Styling of the web page)

Key Concepts:

- Selectors (element, class, ID)
- Box model (margin, border, padding)
- Color and typography properties

Code Example:

```
body {
  font-family: sans-serif;
  background-color: #eaeaea;
  margin: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.container {
  background-color: white;
  padding: 20px;
  border-radius: 6px;
  width: 280px;
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
}

#content {
  font-style: italic;
}
```

JavaScript (Behaviour of the web page)

Key Concepts:

- Variables and data types
- Control structures (conditionals, loops)
- Functions
- Array manipulation methods
- Local storage for data persistence
- DOM manipulation

Code Examples:

Variables and Basic Operations:

```
// Variable declarations
let username = "Abden";
let age = 25;
const isLoggedIn = true;

// Array operations
let names = ["Aaryan", "Abden", "Alex"];
names.forEach(name => console.log(name));
names.push("New Name");
```

Local Storage:

```
// Store user details
function storeUserDetails() {
  const userDetails = {
    fullName: document.getElementById('fullname').value,
    username: document.getElementById('username').value,
    email: document.getElementById('email').value
  };
```

```
    localStorage.setItem('userDetails', JSON.stringify(userDetails));
}

// Retrieve user details
function displayUserDetails() {
    const userDetails = localStorage.getItem('userDetails');
    if (userDetails) {
        const data = JSON.parse(userDetails);
        document.getElementById('userDetails').innerHTML = `
            <p><strong>Name:</strong> ${data.fullName}</p>
            <p><strong>Username:</strong> ${data.username}</p>
        `;
    }
}
```

Git Version Control

Git provides tools for tracking changes and collaborating on code.

Key Git Commands:

```
git init
git add index.html
git add .
git commit -m "Add login functionality"
git remote add origin https://github.com/abden/repo.git
git push -u origin main
git pull origin main
```

DAY 2 Event, Validation, Promises

EVENT (Something a browser or user does)

- Page has finished loading
- HTML input was changed
- An HTML button was clicked (submit button onclick())
- **setTimeout** delays a task to run once after a set time.
- **setInterval** keeps running a task repeatedly at regular time intervals.
- **clearInterval** stops the repeated task started by setInterval.

Page on load event

```
<body onload="alert('Page is loaded')">
```

Button Clicked event

```
<button
    onclick="this.innerHTML = Date()">Time is :
</button>
```

Mouse Entered/left event

```
<button
    onmouseenter="this.innerHTML = 'Mouse Entered'"
    onmouseleave="this.innerHTML = 'Mouse Left'">Mouse
</button>
```

Key pressed

```
<input type="text" onkeypress="pressed()"/>
```

Set Timeout, setInterval, clearInterval (There are other events other than these 3)

```
<button onclick="setTimeout(buttonclick,2000)">Timeout</button>\
<button onclick="meow= setInterval(btnclick,2000)">Interval</button>
<button onclick="clearInterval(meow)">Stop Interval</button>
```

Form Validation

- Form to add vendors, users, with ID, name, rating, and products and so on
- Validation for unique IDs, name length (>=5), and products
- Dynamic product fields (add/remove)
- Display vendors sorted by rating
- Search functionality
- View detailed information in a modal
- Clear all vendor data functionality
- Store vendors as objects in **localStorage**
- Sort vendors by rating
- Validate input before saving
- Learned how to delete data from local Storage

Promises

- Learned about Promises, which are objects representing the eventual completion or failure of asynchronous operations.
- Can be used for Error Handling
- States: **resolve(), reject()**
- Similar to try catch finally

Fetch an image from a UPI

- Get image from an API
- Store it into a variable
- fetch(), then()
- async function

  async function getCat()
  await fetch(url).then(res => res.json()).then(data => data.url)

- await to fetch a data from API

Day 3 TypeScript

- **npm -v**
- **npm init -y**
- **npm install typescript –-save-dev**
- **npm tsc init**
- **npx tsc filename.ts**
- **node filename.js**

TypeScript adds static typing to JavaScript, which helps **catch errors before runtime.**

- **Basic Types & Functions**:

Started with simple functions and type annotations

```
let name: string = "aaryan";
```

- **Interfaces**:

Learned how to define object structures (similar to Java interfaces)

```
interface User {
 name: string;
 id: number;
 role: roles;
}
```

- **Custom Types**:

Created union types to restrict values

```
type roles = "Trainer" | "Developer" | "Tester";
```

- **Classes**:

Implemented classes with constructors and properties

```
class UserAccount {
 name: string;
 id: number;
}
```

- **Interface Implementation**:

Made classes implement interfaces

```
class Vendor implements VendorInterface {...}
```

- **Optional Properties**:

Used the ? symbol for optional properties

```
type Vendor = {
  lname?: string; // optional
}
```

- **Arrays & Sorting**:

Worked with typed arrays and sorting functions

```
const vendors: Vendor[] = [...];
vendors.sort((a, b) => a.id - b.id);
```

- **Type 'any'**:

Used for values where the type isn't known

```
productName: any;
```

- **String Interpolation**:

Used template literals for string formatting

```
console.log(` ID: ${e.id}, Name: ${e.name}` );
```

- **Conditional Expressions**:

Implemented ternary operators

```
e.lname != undefined ? e.name + " " + e.lname : e.name
```

- **Promise to Get the response from API:**

Get the data from url in json format data and display the json data using Promise<Response>.

```
function getFacts(url: string): Promise<Response> {
  return fetch(url).then(res => res.json());
getFacts("https://cataas.com/cat?width=200;height=200;json=true").then(data => console.log(data));
```

Day 4 MCQ Assignment, Practicing Typescript

1. **Microsoft**

Microsoft created and maintains TypeScript as an open-source programming anguage. It can be used to develop JS applications in both client and server side.

2. **Node**

Commonly used with node.js which runs javascript on server side and other backend technologies aswell.

3. **JavaScipt**

Type Script is superset of JavaScript that adds static typing. After compilation TypeScript is converted/compiled to JavaScript.

4. **extends**

Inheritance is implemented using the extends keyword similar to JavaScript Inheritance

5. **var x = "string";**

The variable has no type defined after declaring the variable there should be a colon space type.

This a JavaScript syntax, not a TypeScript syntax

6. **var x: number = 999;**

This is TypeScript syntax,

The variable x of number type is stored the value 999.

7 .**ts**

TypeScript files are typically stored in .ts format, for JavaScript it is .js

8. **tsc filename.ts**

tsc is the command to run TypeScript compiler to compile it to .js file, we can also executa a .ts file using node tsc filename.ts

9. **tsc filename.ts -w**

Make the compiler continuously watch for changes in TypeScript files and recompile automatically when changes are detected.

10. **super()**

use the super() keyword within the child class's constructor.

```
class Parent {
  constructor(public name: string) {
    console.log("Parent constructor called");
  }
}

class Child extends Parent {
  constructor(name: string, public age: number) {
    super(name);
    console.log("Child constructor called");
  }
```

```
}
const child = new Child("John", 25);
```

1. **Sort**

Practiced on how to use the sort function efficiently.

```
vendors.sort((a, b) => a.id - b.id);
```

2. **Filter**

Go through each item in the products array.

For every item (p), check if its price is less than or equal to maxPrice.

Only the products that pass this test will be included in the result.

filter() is a built-in method that returns a new array of items that match a condition.

```
products.filter(p => p.price <= maxPrice);
```

3. **Finding**

It stops at the first match.

If found, it prints the details

If not found, it says “Product not found.”

```
products.find(p => p.name === productName);
```

4. **Every/Some**

every - checks if all items match the condition.

some - checks if at least one item matches.

```
const allUnder100 = products.every(p => p.price < 100);

const anyUnder30 = products.some(p => p.price < 30);
```

5. **Mapping**

map() goes through every product.

For each product, it returns just the name.

You get a new array of only names (original array stays the same).

```
// Map to get only names

const productNames = products.map(p => p.name);
```

6. **Reduce**

(sum, p) => sum + p.price \\Adds the current product's price to the total sum.

0 is the starting value of the sum.

It goes through all items and returns one final result:

```
const totalPrice = products.reduce((sum, p) => sum + p.price, 0);
```

1. **Decorators/Modules**

Decorators = add logic to classes/functions automatically.

```
// Enable "experimentalDecorators": true in tsconfig.json

function Logger(constructor: Function) {

 console.log("Class created:", constructor.name);
```

```
}

@Logger

class Product {

 constructor(public name: string, public price: number) {}

}

const p1 = new Product("Book", 100);
```

Modules = split code into files and share using export / import

mathUtils.ts (a separate file)

```
export function add(a: number, b: number): number {

 return a + b;

}

export const PI = 3.14;
```

app.ts

```
import { add, PI } from './mathUtils';

let result = add(10, 5);

console.log(` Result: ${result}` );

console.log(` Value of PI: ${PI}` );
```

Day 5 Operators

1. **Keyof operator**

The keyof operator in TypeScript is used to extract the keys of an object type as a union of string literal types. It provides a way to ensure type safety when working with object properties dynamically.

```
function getProperty<T, K extends keyof T>(obj: T, key: K): T[K] {
    return obj[key];
}
```

2. **Rest operator**

```
function sum(...numbers: number[]): number {
  return numbers.reduce((total, num) => total + num, 0);
}
const res = sum(1,2,3);
```

Collects all remaining arguments into an array. When you don't know how many arguments will be passed to a function, use rest operator.

Rest Operator is used in function **parameters**. **Spread Operator** (also ...) is used to **expand** elements, like:

```
sumof(...products)
```

3. **Overloading**

Define **multiple function signatures** for a single method, each with different parameter types or counts.

In TypeScript, **you define overloads with multiple function signatures**, and then provide **one actual implementation** that handles all cases.

```
speak(s: string): string;
speak(n: number): string;
speak(b: boolean): string;
```
These are the **overload declarations**. They tell TypeScript what calls are allowed

```
speak(arg: any): any {
    if (typeof arg === 'number') {
        return `Meow number ${arg}`;
    }
    if (typeof arg === 'string') {
        return `Meow string ${arg}`;
    }
    if (typeof arg === 'boolean') {
        return `Meow boolean ${arg}`;
    }
}
```
This is the **actual implementation** that handles **both overloads**. TypeScript only allows **one implementation**, and it must be compatible with all the declared signatures.

4. **Modules (import/export)**

Consider two files, one for importing and exporting. Now we will split into files and share using **import** and **export.**

```
//mathUtils.ts(a separate file)
export function add(a: number, b: number): number {
    return a + b;
}
export const PI = 3.14;
export function area(radius: number): number {
    return PI * radius * radius;
}
//sub.ts(a separate file)
export default function subtract(a: number, b: number): number {
    return a - b;
}
```
When we use **export**, we are making **functions, variables, classes, or interfaces available to other files**. **default** is used when you're exporting **one main thing** from a module.

```
//app.ts(another file)
import {add, PI } from './mathUtils';
let result = add(10, 5);
console.log(`Result: ${result}`);
console.log(`Value of PI: ${PI}`);

//subfun.ts(a separate file)
```

```
import subtract from './sub.ts';
console.log(subtract(5,10));
```

When we use import to **bring in code that was exported from another module**.

If we want to **import everything** from a module use import * as anyname from ‘./’;

```
import * as MathUtils from './mathUtils';

console.log(MathUtils.add(5,10));
console.log(MathUtils.PI);
```

To Rename things when importing we can do it like this

```
import { multiply as mul } from "./1_export";
console.log(mul(4, 5));
```