2차시 스크립트

Recap

1. 파일 만드는것부터 시작
2. live server 다운로드하라고 하고 화면분할
3. 최상위 html, 그다음 head, body tab으로 템플릿 생성
4. style태그가 head에 들어가고 맨 아래에 script
5. header main footer 구조인데 사실 div랑 똑같다. 접근성을 위해서 존재하는것

```
<!DOCTYPE html>
<html lang="ko">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>간단한 자기소개 페이지</title>
  <style>
   /* 전체 페이지 스타일 */
   body {
    background-color: green;
    color: #333;
    margin: 0;
    padding: 0;
    text-align: center;
   }

   /* 헤더 스타일 */
   header {
    background-color: yellow;
    color: white;
    padding: 20px;
    margin-bottom: 20px;
   }

   /* 메인 콘텐츠 스타일 */
```

```
    main {
      padding: 20px;
    }

    h1 {
      margin-bottom: 10px;
    }

    p {
      font-size: 1.1em;
      line-height: 1.6;
    }

    /* 버튼 스타일 */
    button {
      background-color: brown;
      color: white;
      border: none;
      padding: 10px 20px;
      cursor: pointer;
      border-radius: 5px;
      font-size: 1em;
      margin-top: 20px;
    }

    button:hover {
      background-color: pink;
    }

    /* 이미지 스타일 */
    img {
      width: 150px;
      height: 150px;
      border-radius: 50%;
      margin-top: 20px;
      border: 3px solid black;
    }
  </style>
```

```
  </head>
  <body>
    <header>
      <h1>안녕하세요! 저는 호랑이입니다.</h1>
    </header>
    <main>
      <img src="profile.jpg" alt="프로필 사진" />
      <p>쿠러그에서 웹개발을 공부하고 있습니다.</p>
      <p>HTML, CSS, JavaScript를 공부 중이에요!</p>
      <button onclick="showMessage()">클릭해보세요</button>
      <p id="message"></p>
    </main>
    <script>
      function showMessage() {
        const messageElement = document.getElementById("message");
        messageElement.textContent = "반갑습니다! 😊";
        messageElement.style.color = "red";
        messageElement.style.fontWeight = "bold";
      }
    </script>
  </body>
</html>
```

강의 목표

- JavaScript의 기초 개념 및 필수 함수 이해
- DOM을 조작하여 동적인 웹페이지를 만들 수 있다.
- 이벤트 핸들링을 사용하여 사용자와 상호작용하는 기능을 구현할 수 있다.
- 실습: 버튼 클릭 시 내용 변경하는 interactive 페이지 만들기

1.1 변수, 자료형

- **변수 선언:** `var`, `let`, `const`
  - `var`: 함수 스코프, 호이스팅 특징 있음
  - `let`과 `const`: 블록 스코프, `const`는 재할당 불가

- **기본 자료형:** 문자열(String), 숫자(Number), 불리언(Boolean), null, undefined, 객체(Object), 배열(Array)

```
let username = "홍길동";  // 문자열
const PI = 3.14;        // 숫자 (상수)
let isActive = true;    // 불리언
let data = null;        // 널
let result;             // undefined
let person = { name: "홍길동", age: 30 }; // 객체
let list = [1, 2, 3, 4, 5]; // 배열
```

```
console.log(10 + 5);        // 15
console.log(10 === "10");   // false, 타입까지 비교 ==와 다르다
console.log(10 > 5 && 5 < 10); // true
```

> 예상질문: 왜 console.log일까?
>
> **1. 브라우저의 실행 환경과 콘솔**
>
> - 자바스크립트는 원래 웹 페이지를 동적으로 만들기 위해 브라우저에서 실행되도록 만들어졌어요. 브라우저에는 사용자가 직접 보는 화면(HTML 요소)과 개발자가 디버깅하거나 로그를 확인할 수 있는 **개발자 도구**(Developer Tools)가 있어요.
> - 이 개발자 도구 안에 **콘솔**(Console)이라는 영역이 있는데, 여기서 개발자는 코드 실행 중 변수 값이나 메시지를 확인할 수 있어요. 그래서 자바스크립트는 출력 함수를 console 객체에 연결한 거예요.
> - 즉, console.log는 "콘솔에 로그를 남긴다"는 뜻으로, 브라우저의 콘솔 창에 데이터를 출력하는 역할을 합니다.
>
> **2. 왜 print가 아닌 log인가?**
>
> - 다른 언어(예: Python)에서 print는 보통 종이에 출력하거나 터미널에 텍스트를 뿌리는 느낌으로 설계됐어요. 하지만 브라우저 환경

> 에서는 "출력"이 반드시 화면에 보이는 게 아니라, 개발자가 디버깅하거나 정보를 기록(log)하는 데 초점이 맞춰져 있어요.
>
> - log라는 이름은 "로그를 기록하다"라는 뜻에서 왔고, 이는 개발자가 코드를 추적하거나 문제를 분석할 때 쓰는 용도와 잘 맞아요. 브라우저 콘솔은 단순히 출력뿐 아니라 에러, 경고, 정보 같은 다양한 로그를 남길 수 있는 도구라서 log라는 이름이 더 적합했던 거죠.
> - 실제로 console 객체에는 console.log 외에도 console.error, console.warn, console.info 같은 메서드가 있어서, 단순한 "출력"보다 "로그 기록"이라는 개념이 더 포괄적이에요.

1. `var` 은 함수 스코프 (Function Scope)

- `var`로 선언된 변수는 **함수 단위**로 범위가 정해져요. 즉, 함수 안에서 선언되면 그 함수 전체에서 사용할 수 있고, 함수 밖에서는 접근할 수 없어요.
- 하지만 함수 안의 중괄호 `{}` (예: `if`, `for` 같은 블록)에서는 범위를 제한하지 않아요. 블록 밖에서도 변수가 살아있죠.

예시:

```
function example() {
  if (true) {
    var x = 10;
  }
  console.log(x); // 10 출력 (if 블록 밖에서도 접근 가능)
}
example();
```

- `var x`는 `if` 블록 안에 선언됐지만, 함수 스코프라서 함수 전체에서 유효해요.

2. `let` 은 블록 스코프 (Block Scope)

- `let`으로 선언된 변수는 **중괄호 `{}` 단위**로 범위가 정해져요. 즉, `{}` 안에서 선언되면 그 블록 안에서만 사용할 수 있고, 블록 밖에서는 접근할 수 없어요.
- `if`, `for`, `while` 같은 블록 안에서 선언된 변수는 블록이 끝나면 사라진다고 생각하면 돼요.

예시:

```
function example() {
  if (true) {
    let y = 20;
    console.log(y); // 20 출력 (블록 안에서만 유효)
  }
  console.log(y); // 오류! y는 블록 밖에서 접근 불가
}
example();
```

차이점 요약

- `var`: 함수 전체에서 유효. 중괄호는 무시하고 함수 끝까지 살아있음.
- `let`: 중괄호 `{}` 안에서만 유효. 블록을 벗어나면 사라짐.

추가 팁

- `var`은 옛날 자바스크립트에서 쓰던 방식이라 요즘은 `let`이나 `const`를 더 많이 써요. `let`이 범위를 더 명확히 제어할 수 있어서 코드가 덜 헷갈리거든요.
- 예외적으로 전역에서 `var`을 쓰면 `window` 객체에 붙지만, `let`은 그렇지 않아요.

1.2 함수

함수는 특정 작업 수행을 위한 코드 블록이다. 입력을 받아서 어떤 작업을 한 결과를 반환하는 역할을 한다. 코드의 재사용이 쉬워지고 코드를 작게 나눠서 관리할 수 있게 해준다.
함수 선언식은 가장 기본적인 방법으로 특징은 코드 어디에서나 호출할 수 있는 호이스팅이 발생한다. 함수 선언문은 실제 코드보다 위쪽에서 호출해도 동작한다는 뜻

함수 표현식은 함수를 변수에 할당하는 방식으로 함수이름 없이 익명함수를 줄 수 있고 변수에 할당해 놨다가 나중에 호출할 수 있다. 선언식도 나중에 호출하니까 큰 차이는 없는데 호이스팅이 안된다는 점이 다르다. 즉 변수가 할당된 후에만 호출할 수 있다.

화살표 함수는 es6부터 도입된 기능으로 함수를 간단하게 작성하기 위한 것

(다른점은 this를 가지지 않는다.)

```
// 함수 선언식
function greet(name) {
   return `안녕, ${name}!`;
}
```

```
// 함수 표현식
const square = function(n) {
    return n * n;
};

// 화살표 함수
const multiply = (a, b) => a * b;
```

1.3 조건문 반복문

- **조건문:** `if` , `else if` , `else` , `switch`
- **반복문:** `for` , `while` , `do while`

```
let score = 85;
if (score >= 90) {
    console.log("A");
} else if (score >= 80) {
    console.log("B");
} else {
    console.log("C");
}

for (let i = 0; i < 5; i++) {
    console.log(i);
}

while(true) {
 ...
}
```

1.4 배열 메서드

```
let nums = [1, 2, 3, 4, 5];

// forEach: 각 요소 순회
nums.forEach(num => console.log(num));
```

```
// map: 배열 변환
let squared = nums.map(num => num ** 2);

// filter: 조건에 맞는 요소 추출
let evens = nums.filter(num => num % 2 === 0);

// reduce: 누적값 계산
let sum = nums.reduce((acc, cur) => acc + cur, 0);

const arr = ['apple', 'banana', 'orange'];
const result = arr.reduce((acc, curr, index) => {
  acc[index] = curr;
  return acc;
}, {});
console.log(result); // {0: 'apple', 1: 'banana', 2: 'orange'}
```

1.5 콜백함수

콜백 함수란?

콜백 함수는 다른 함수에 인자로 전달되어, 특정 작업이 완료되거나 이벤트가 발생했을 때 호출되는 함수.

즉, 함수가 실행되는 "콜백"을 나중에 호출하도록 미리 전달하는 방식으로, 비동기 처리나 이벤트 핸들링에서 매우 유용하게 사용됨.

콜백 함수 사용 예시

```
// 배열의 각 요소를 처리하는 forEach 메서드에 전달하는 콜백 함수
[1, 2, 3].forEach(function(item) {
    console.log(item);
});
```

또한, 이벤트 핸들러에서도 콜백 함수가 사용된다.

```
document.getElementById("btn").addEventListener("click", function() {
```

```
    console.log("버튼이 클릭됨");
});
```

이처럼, 콜백 함수는 특정 작업이 끝난 후 실행해야 할 코드를 미리 등록하는 역할을 한다.

> 예상 질문:
>
> **Q:** 콜백 함수와 동기/비동기 함수의 관계는?
>
> **A:** 콜백 함수는 주로 비동기 작업(예: HTTP 요청, 타이머, 이벤트 처리)에서 사용되지만, 동기 작업에서도 함수에 인자로 전달될 수 있다. 비동기 작업에서 콜백을 사용하면, 작업이 완료된 후 특정 코드를 실행할 수 있게 되어 효율적인 처리가 가능하다.

---

2. DOM조작 및 이벤트 핸들링 (여기서부터 ppt로 강의)

2.1 DOM의 개념

- **정의:**

  DOM은 HTML 문서를 객체(object)와 노드(node)들의 트리 구조로 표현한 모델이야. 브라우저는 HTML 파일을 읽고 이 트리 형태의 DOM을 만들어, JavaScript로 문서의 내용을 동적으로 접근하고 조작할 수 있도록 해.

- **왜 필요한가?**
  - 동적인 웹 페이지를 구현하기 위해 HTML 요소를 추가, 삭제, 수정할 필요가 있음.
  - 사용자와의 상호작용(예: 버튼 클릭, 입력값 처리) 시, DOM을 통해 화면에 즉각 반영할 수 있어.
  - 예를 들어, 투두리스트 애플리케이션에서 사용자가 새 항목을 추가하면, 자바스크립트를 통해 DOM에 새로운 `<li>` 요소를 추가하여 바로 화면에 보여줄 수 있어.

  1. 브라우저가 HTML 파일을 읽어 들이면, 이를 기반으로 DOM 트리를 생성.
  2. JavaScript 코드가 실행되며 DOM API를 사용해 트리의 특정 노드를 선택하거나 조작.

3. 예를 들어, 사용자가 버튼을 클릭하면 해당 이벤트가 발생하고, 등록된 이벤트 핸들러가 호출되어 DOM의 내용을 변경.
4. 변경된 DOM은 브라우저에 즉시 반영되어 사용자에게 새로운 화면으로 보여짐.

2.2 DOM의 구조와 동작 원리

- **트리 구조:**
  - HTML 문서의 모든 요소는 트리 형태로 구성돼. 최상위 노드는 `document` 객체이고, 그 아래에 `<html>` 요소, 그 다음 `<head>`, `<body>` 등으로 구성돼.
  - 각 요소는 노드(node)라고 부르며, 자식 노드, 형제 노드 등 계층 구조를 형성함.
- **동작 방식:**
  - JavaScript를 사용하여 DOM에 접근하면, `document.getElementById()`나 `document.querySelector()` 같은 메서드를 통해 특정 노드를 선택할 수 있어.
  - 선택한 노드의 속성이나 내용을 변경하거나, 새로운 요소를 추가할 수 있음.
  - 예를 들어, 사용자가 버튼을 클릭하면 이벤트 핸들러가 실행되어, 특정 요소의 텍스트 내용을 바꾸거나 스타일을 수정할 수 있음.
- HTML 요소 선택: `getElementById()`, `querySelector()`
- 요소 생성 및 조작: `createElement()`, `appendChild()`, `textContent` 변경 등

```
// 요소 선택
let title = document.getElementById("title");
let button = document.querySelector("button");

// 요소 내용 변경
title.textContent = "새로운 제목";

// 새로운 요소 생성
let newItem = document.createElement("li");
newItem.textContent = "리스트 아이템";
document.querySelector("ul").appendChild(newItem);
```

2.3 이벤트 핸들링

- `addEventListener()`를 사용해 이벤트 등록
- 이벤트 객체 활용

```
button.addEventListener("click", function(event) {
    console.log("버튼이 클릭됨", event);
    title.textContent = "버튼 클릭 후 제목 변경";
});
```

2.4 렌더링 과정 개요

브라우저는 HTML, CSS, JavaScript를 처리하여 화면에 내용을 보여주는데, 이 과정은 크게 다음과 같은 단계로 이루어져 있어.

1. **HTML 파싱 및 DOM 생성:**
    - 브라우저는 HTML 문서를 읽어 들여 DOM(Document Object Model)이라는 트리 구조를 생성해.
    - 이 과정에서 HTML 요소들이 노드 형태로 메모리에 구성됨.
2. **CSS 파싱 및 CSSOM 생성:**
    - HTML과 함께 로드된 CSS 파일이나 인라인 스타일을 읽어 CSSOM(CSS Object Model)이라는 구조를 만든다.
    - CSSOM은 스타일 규칙들을 객체로 표현하여, 각 DOM 요소에 적용될 스타일 정보를 담음.
3. **렌더 트리 생성 (Render Tree):**
    - DOM과 CSSOM이 결합되어, 실제 화면에 렌더링될 요소들의 정보를 담은 렌더 트리를 만든다.
    - 렌더 트리는 화면에 표시되는 요소들만 포함하고, 숨겨진 요소들은 배제됨.
4. **레이아웃 (Layout 또는 Reflow):**
    - 렌더 트리를 기반으로 각 요소의 위치와 크기를 계산하는 단계다.
    - 이 과정에서 요소들이 실제 화면 상에서 어디에 배치될지 결정됨.
5. **페인팅 (Painting):**
    - 계산된 레이아웃 정보를 바탕으로 각 요소의 스타일(색상, 폰트, 그림자 등)을 픽셀 단위로 실제 화면에 그려 넣는 단계다.
    - 브라우저는 여러 개의 페인팅 단계를 거쳐 최종적으로 화면에 결과물을 표시함.

2.5 Reflow와 Repaint

- **Reflow (Layout):**
  - DOM이나 CSSOM의 변경으로 인해 요소의 크기, 위치, 구조 등이 변경될 때 발생한다.
  - 예를 들어, 새로운 요소가 추가되거나, 창의 크기가 변경되는 경우가 해당됨.
  - 리플로우는 비용이 큰 작업이기 때문에 불필요한 DOM 조작은 피하는 것이 좋다.
- **Repaint:**
  - 요소의 색상이나 시각적 스타일만 변경되는 경우에 발생한다.
  - 예를 들어, 배경색이나 글자 색을 변경할 때 리플로우 없이 페인팅 단계만 다시 실행됨.
  - Repaint는 리플로우보다는 덜 비용이 크지만, 자주 발생하면 성능에 영향을 줄 수 있다.

2.6 JavaScript, DOM, CSSOM의 적용 시기

- **HTML 파싱 시:**
  - 브라우저는 HTML을 읽으면서 DOM 트리를 구성한다.
  - 스크립트 태그(`<script>`)가 발견되면, 해당 JavaScript 코드를 즉시 실행(기본적으로는 동기 실행)하여 DOM 조작 등이 발생할 수 있다.
- **CSS 파싱 시:**
  - CSS 파일은 HTML과 병렬로 다운로드되지만, CSSOM 생성은 HTML 파싱과 동시에 이루어져 렌더 트리 생성에 필요하다.
  - JavaScript가 CSS 스타일을 변경하면, 브라우저는 CSSOM을 업데이트하고, 필요 시 레이아웃과 페인팅 과정을 다시 실행한다.
- **JavaScript 실행 시:**
  - JS 코드는 DOM과 CSSOM이 일부 혹은 전체 생성된 후 실행된다.
  - 이 때 DOM 조작, 이벤트 등록, 애니메이션 제어 등 다양한 작업을 수행할 수 있다.
  - JS가 DOM이나 CSSOM을 변경하면, 변경된 내용에 따라 Reflow/Repaint가 발생할 수 있다.
  - 기본적으로 `<script>` 태그는 동기적으로 실행되므로, 스크립트가 실행되는 동안 HTML 파싱이 중단된다. 이를 방지하기 위해 `defer`나 `async` 속성을 사용하여 스크립트 로딩 방식을 제어할 수 있다.

3. Web api

Web API란?

- **개념**

  Web API는 웹 브라우저가 제공하는 여러 기능.
  브라우저 내장 API로는 DOM, Fetch, Canvas 등 다양한 API들이 포함됨.

- **어떻게 동작하는가?**
  - 브라우저가 HTML, CSS, JavaScript를 해석하면서 이 API들을 제공하여 개발자가 손쉽게 브라우저의 기능들을 사용할 수 있도록 해줘.
  - 예를 들어, Fetch API를 사용하면 서버에 HTTP 요청을 보내 데이터를 받아올 수 있고, Canvas API를 사용하면 그림을 그리거나 애니메이션을 만들 수 있어.
- 서버 통신, 데이터 저장, 위치 정보 서비스 등 다양한 기능을 웹 페이지에 손쉽게 구현할 수 있음.

3.2 localstorage

local storage는 web storage api의 한 부분(cookie, session, localstorage)

브라우저에 데이터를 영구적으로 저장할 수 있는 기능. key-value형태로 저장하고 브라우저 종료하거나 새로고침해도 데이터가 사라지지 않음

localStorage는 문자열만 저장 가능하기 때문에, 객체나 배열을 저장할 때는 `JSON.stringify()`로 문자열로 변환해 저장하고, 불러올 때는 `JSON.parse()`를 사용해 원래 자료형으로 복원해야 함.

하지만 안전한 방법은 아님! 비밀번호를 저장하거나 하기에는 좋은 방법은 아니다.

```
let todos = [
   { id: 1, task: "자바스크립트 공부", completed: false },
   { id: 2, task: "HTML 연습", completed: true }
];
localStorage.setItem("todos", JSON.stringify(todos));

// 데이터 읽기
let storedTodos = JSON.parse(localStorage.getItem("todos"));
console.log(storedTodos);
```

4. 실습

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>To doList</title>
    <style>
        body { margin: 20px; }
        #todo-form { margin-bottom: 20px; }
        #todo-list { list-style: none; padding: 0; }
        #todo-list li { padding: 10px; border-bottom: 1px solid #ccc; display: flex;
        .completed { text-decoration: line-through; color: gray; }
        button.delete { margin-left: auto; background-color: red; color: white; bor
    </style>
</head>
<body>
    <h1>투두리스트</h1>
    <form id="todo-form">
        <input type="text" id="todo-input" placeholder="할 일을 입력하세요" requi
        <button type="submit">추가</button>
    </form>
    <ul id="todo-list"></ul>

    <script>
        let todos = JSON.parse(localStorage.getItem("todos")) || [];

        const todoForm = document.getElementById("todo-form");
        const todoInput = document.getElementById("todo-input");
        const todoList = document.getElementById("todo-list");

        function saveTodos() {
            localStorage.setItem("todos", JSON.stringify(todos));
        }

        function renderTodos() {
            // 기존 리스트 초기화
```

```
        todoList.innerHTML = "";
        todos.forEach(todo => {
            const li = document.createElement("li");
            li.dataset.id = todo.id;

            // 완료 상태에 따른 클래스 추가
            if (todo.completed) {
                li.classList.add("completed");
            }

            // 투두 텍스트
            const span = document.createElement("span");
            span.textContent = todo.task;
            // 완료 상태 토글 이벤트
            span.addEventListener("click", () => {
                todo.completed = !todo.completed;
                saveTodos();
                renderTodos();
            });
            li.appendChild(span);

            // 삭제 버튼
            const delButton = document.createElement("button");
            delButton.textContent = "삭제";
            delButton.className = "delete";
            delButton.addEventListener("click", () => {
                // 클릭한 투두 삭제
                todos = todos.filter(item => item.id !== todo.id);
                saveTodos();
                renderTodos();
            });
            li.appendChild(delButton);

            todoList.appendChild(li);
        });
    }

    // 폼 제출 시 새로운 투두 추가
```

```
        todoForm.addEventListener("submit", (e) => {
            e.preventDefault();
            const task = todoInput.value.trim();
            if (task !== "") {
                // 고유 ID 생성 (타임스탬프 사용)
                const newTodo = {
                    id: Date.now(),
                    task: task,
                    completed: false
                };
                todos.push(newTodo);
                saveTodos();
                renderTodos();
                todoInput.value = "";
            }
        });

        // 페이지 로드 시 투두 렌더링
        renderTodos();
    </script>
</body>
</html>
```