

CH11 자바스크립트

전체 소스 코드

<https://github.com/codingspecialist/javascript-class>

자바스크립트(JavaScript)는 웹 브라우저에서 동작하는 스크립트 언어 중 하나입니다. HTML, CSS와 함께 웹 프로그래밍을 할 수 있도록 고안되었으며, 다양한 용도로 사용됩니다.

자바스크립트는 클라이언트 측에서 실행되는 스크립트 언어로, 웹 페이지를 동적으로 만들고, 사용자와 상호 작용하는 기능을 구현할 수 있습니다. 예를 들어, HTML 요소의 속성 값을 변경하거나, 이벤트 핸들러를 등록 하여 사용자의 동작에 따라 새로운 기능을 수행하도록 할 수 있습니다. 또한, AJAX를 이용하여 서버와 비동기 적으로 데이터를 주고받는 기능을 구현할 수도 있습니다.

자바스크립트는 브라우저를 제어하는 데 주로 사용되지만, Node.js와 같은 서버 측 프레임워크를 이용하여 서버 측에서도 사용될 수 있습니다. 이러한 이유로, 자바스크립트는 현재 가장 널리 사용되는 언어 중 하나이며, 다양한 분야에서 사용되고 있습니다.

1. 자바스크립트 자료형

1.1 자료형

자바스크립트에서는 다음과 같은 기본 자료형을 제공합니다.

1. 숫자(Number)

- 정수와 실수를 모두 표현할 수 있습니다.

2. 문자열(String)

- 문자열을 표현할 수 있습니다.

3. 불리언(Boolean)

- true와 false 두 가지 값 중 하나를 가집니다.

4. null

- 값이 없음을 나타내는 자료형입니다.

5. undefined

- 값이 할당되지 않은 상태를 나타내는 자료형입니다.

6. 객체(Object)

- 다양한 형태의 데이터를 저장할 수 있는 자료형입니다.

7. 배열(Array)

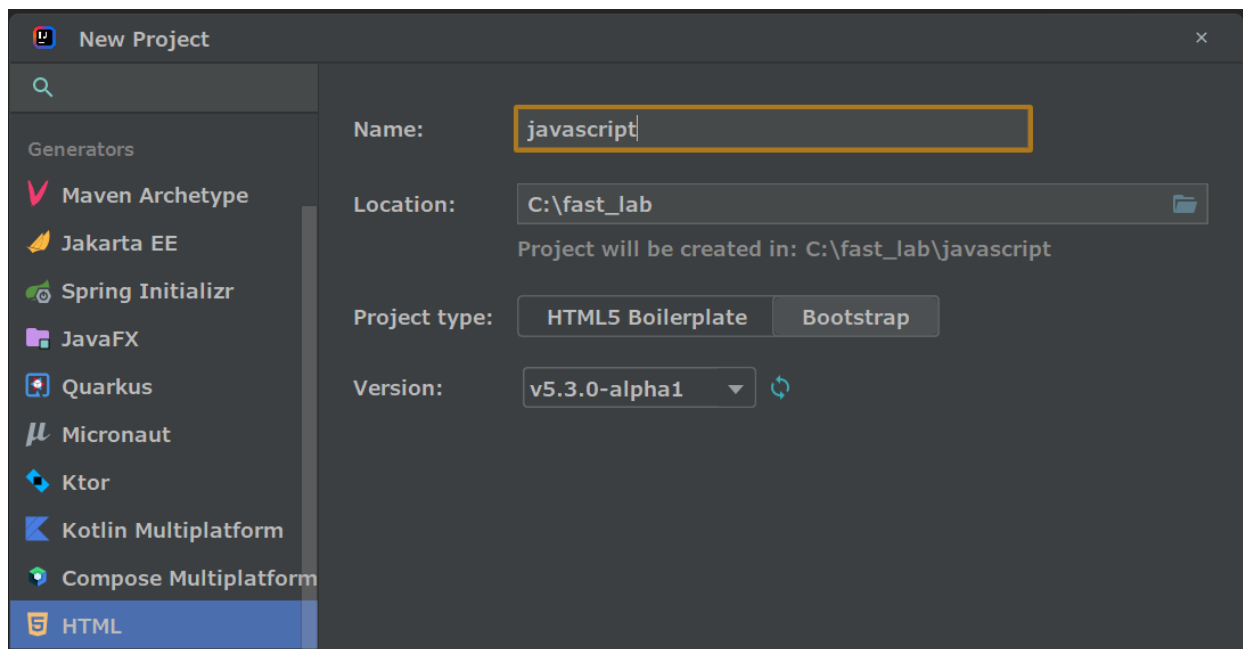
- 여러 개의 값을 순서대로 저장하는 자료형입니다.

8. 함수(Function)

- 일련의 작업을 수행하는 코드 블록입니다.

자바스크립트는 동적 타이핑(dynamic typing)을 지원하므로, 변수를 선언할 때 데이터 타입을 명시할 필요가 없습니다. 변수에 할당되는 값에 따라 자동으로 데이터 타입이 결정됩니다.

1.2 프로젝트 생성



1.3 자바스크립트 자료형 실습

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
  </head>
  <body>
    <script>
      let n1 = 1;
      let n2 = 1.5;
      console.log("정수", n1);
      console.log("실수", n2);
    </script>
  </body>
</html>
```

```
let n3 = "자바스크립트 자료형 강좌";
console.log("문자열", n3);

let n4 = true;
console.log("불리언", n4);

let n5 = null;
console.log("널", n5);

let n6 = undefined;
let n7;
console.log("언디파인", n6);
console.log("언디파인", n7);

let user = {
  id: 1,
  username: "ssar",
  password: "1234"
};
console.log("객체", user);
console.log("객체", user.id);
console.log("객체", user.username);
console.log("객체", user.password);

let arr = [1,2,3];
console.log("배열", arr);
console.log("배열", arr[0]);
console.log("배열", arr[1]);
console.log("배열", arr[2]);

function fun1(){
  console.log("fun1 함수");
}
fun1();

let fun2 = function(){
  console.log("fun2 함수");
};
fun2();

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

정수	1
실수	1.5
문자열	자바스크립트 자료형 강좌
불리언	true
널	null
언디파인	undefined
언디파인	undefined
객체	▼ Object ⓘ id: 1 password: "1234" username: "ssar" ▶ [[Prototype]]: Object
객체	1
객체	ssar
객체	1234
배열	▼ Array(3) ⓘ 0: 1 1: 2 2: 3 length: 3 ▶ [[Prototype]]: Array(0)
배열	1
배열	2
배열	3
fun1	함수
fun2	함수

1.4 호이스팅 참고

<https://hanamon.kr/javascript-%ED%98%B8%EC%9D%B4%EC%8A%A4%ED%8C%85%EC%9D%B4%EB%9E%80-hoisting/>

2. 자바스크립트 이벤트

2.1 이벤트

우리는 자바스크립트를 HTML 요소의 속성 값을 변경하거나, 이벤트 핸들러 등록, AJAX 통신으로만 사용할 것이다.

1. HTML 요소에 직접 이벤트 핸들러 등록하기

```
<button onClick="alert('Hello World!')">Click me</button>
```

이 방법은 간단하지만, 이벤트 핸들러가 HTML과 JavaScript 코드에 모두 들어가므로 유지 보수가 어려울 수 있습니다.

2. addEventListener() 메서드를 사용하여 이벤트 핸들러 등록하기

```
<button id="myBtn">Click me</button>
<script>
  const myBtn = document.getElementById("myBtn");
  myBtn.addEventListener("click", function() {
    alert("Hello World!");
  });
</script>
```

이 방법은 HTML과 JavaScript 코드를 분리하여 유지 보수를 쉽게 할 수 있습니다. addEventListener() 메서드는 이벤트 이름과 이벤트 핸들러 함수를 인자로 받습니다.

1. `click` : 요소를 클릭할 때 발생합니다.
2. `dblclick` : 요소를 더블 클릭할 때 발생합니다.
3. `mousedown` : 요소에서 마우스 버튼이 눌릴 때 발생합니다.
4. `mouseup` : 요소에서 마우스 버튼이 떼어질 때 발생합니다.
5. `mousemove` : 요소에서 마우스가 움직일 때 발생합니다.
6. `mouseover` : 요소에 마우스 커서가 올라갈 때 발생합니다.
7. `mouseout` : 요소에서 마우스 커서가 벗어날 때 발생합니다.
8. `keydown` : 키보드에서 키를 누를 때 발생합니다.
9. `keyup` : 키보드에서 키를 떼를 때 발생합니다.
10. `submit` : 폼을 제출할 때 발생합니다.
11. `change` : 요소의 값이 변경될 때 발생합니다.
12. `load` : 웹 페이지나 이미지 등이 로딩되었을 때 발생합니다.

3. jQuery를 이용하여 이벤트 핸들러 등록하기

```
<button id="myBtn">Click me</button>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
    $("#myBtn").on("click", function() {
        alert("Hello World!");
    });
</script>
```

이 방법은 jQuery 라이브러리를 사용하여 간단하게 이벤트 핸들러를 등록할 수 있습니다. on() 메서드를 사용하여 이벤트 이름과 이벤트 핸들러 함수를 인자로 받습니다.

2.2 이벤트 맛보기 실습

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <button onclick="alert('Hello World!')">Click me</button>
    <button id="myBtn">Click me</button>

    <script>
        const myBtn = document.querySelector("#myBtn");
        myBtn.addEventListener("click", function() {
            alert("Hello World!");
        });
    </script>

</body>
</html>
```

← → ↻ ⓘ localhost:63342/javascript/ex02.html?_ijt=7qe2enohtmpcl1biiu9d2tt94

JUnit 유용한사이트 flutter 스프링부트

Click me

Click me

localhost:63342 says

Hello World!

OK

3. DOM 숨기기

숨기기

display로 숨기기

visible로 숨기기

내부박스1

내부박스2

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <style>
    .box {
      border: 1px solid black;
      padding: 10px;
    }
  </style>
</head>
<body>
  <h1>숨기기</h1>
  <button onclick="hideDisplay()">display로 숨기기</button>
  <button onclick="hideVisible()">visible로 숨기기</button>
  <div class="box">
    <div class="box" id="innerBox1">
      내부박스1
    </div>
    <div class="box" id="innerBox2">
      내부박스2
    </div>
  </div>

  <script>
    function hideDisplay(){
      let el = document.querySelector("#innerBox1");
      el.style.display = "none";
    }
    function hideVisible(){
      let el = document.querySelector("#innerBox2");
```

```

        el.style.visibility = "hidden";
    }
</script>
</body>
</html>

```

4. DOM 나타내기

나타내기

display로 나타내기

visible로 나타내기

내부박스1

내부박스2

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <style>
    .box {
      border: 1px solid black;
      padding: 10px;
    }
    #innerBox1 {
      display: none;
    }
    #innerBox2 {
      visibility: hidden;
    }
  </style>
</head>
<body>
  <h1>나타내기</h1>
  <button onclick="showByDisplay()">display로 나타내기</button>
  <button onclick="showByVisible()">visible로 나타내기</button>
  <div class="box" id="outerBox">

```



```

<div class="box" id="innerBox1">
    내부박스1
</div>
<div class="box" id="innerBox2">
    내부박스2
</div>
</div>

<script>
    function showByDisplay(){
        let el = document.querySelector("#innerBox1");
        el.style.display = "block";
    }
    function showByVisible(){
        let el = document.querySelector("#innerBox2");
        el.style.visibility = "visible";
    }
</script>
</body>
</html>

```

5. DOM 추가하기

추가하기

append로 추가하기

prepend로 추가하기

before로 추가하기

after로 추가하기

내부박스3

내부박스2

내부박스1

내부박스4

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

```

```

<title>Title</title>
<style>
    .box {
        border: 1px solid black;
        padding: 10px;
    }
</style>
</head>
<body>
    <h1>추가하기</h1>
    <button onclick="addAppend()">append로 추가하기</button>
    <button onclick="addPrepend()">prepend로 추가하기</button>
    <button onclick="addBefore()">before로 추가하기</button>
    <button onclick="addAfter()">after로 추가하기</button>
    <div class="box" id="outerBox">

</div>

<script>
    function addAppend(){
        let newEl = document.createElement("div");
        newEl.setAttribute("class", "box");
        newEl.setAttribute("id", "innerBox1");
        newEl.innerHTML = "내부박스1";

        let el = document.querySelector("#outerBox");
        el.append(newEl);
    }
    function addPrepend(){
        let newEl = document.createElement("div");
        newEl.setAttribute("class", "box");
        newEl.setAttribute("id", "innerBox2");
        newEl.innerHTML = "내부박스2";

        let el = document.querySelector("#outerBox");
        el.prepend(newEl);
    }
    function addBefore(){
        let newEl = document.createElement("div");
        newEl.setAttribute("class", "box");
        newEl.setAttribute("id", "innerBox3");
        newEl.innerHTML = "내부박스3";

        let el = document.querySelector("#outerBox");
        el.before(newEl);
    }
    function addAfter(){

```

```

        let newEl = document.createElement("div");
        newEl.setAttribute("class", "box");
        newEl.setAttribute("id", "innerBox4");
        newEl.innerHTML = "내부박스4";

        let el = document.querySelector("#outerBox");
        el.after(newEl);
    }
</script>
</body>
</html>

```

6. DOM 삭제하기

삭제하기

remove로 삭제하기

내부박스1

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <style>
    .box {
      border: 1px solid black;
      padding: 10px;
    }
  </style>
</head>
<body>
  <h1>삭제하기</h1>
  <button onclick="del()">remove로 삭제하기</button>
  <div class="box" id="outerBox">
    <div class="box" id="innerBox1">
      내부박스1
    </div>
  </div>

```

```

</div>

<script>
  function del(){
    let el = document.querySelector("#innerBox1");
    el.remove();
  }
</script>
</body>
</html>

```

7. DOM LIST 추가하기

반복문으로 리스트 만들기

render

제목1 입니다

내용1 입니다

삭제

제목2 입니다

내용2 입니다

삭제

제목3 입니다

내용3 입니다

삭제

제목4 입니다

내용4 입니다

삭제

```

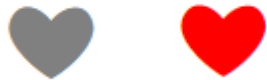
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <style>

```



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

8. DOM Toggle 하기



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <style>
    .like-heart {
      font-size: 24px;
      color: gray;
      cursor: pointer;
    }

    .like-heart.liked {
      color: red;
    }
  </style>
</head>
<body>

  <span class="like-heart" onclick="likeToggle()">&#10084;</span>

  <script>
    // https://www.htmlsymbols.xyz/heart-symbols
    function likeToggle() {
      let likeHeart = document.querySelector('.like-heart');
      likeHeart.classList.toggle('liked');
    }
  </script>
```

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

9. Promise

자바스크립트 Promise는 비동기 작업을 처리하기 위한 객체입니다. Promise는 어떤 작업의 결과를 반환하는 객체로서, 이를 통해 비동기적으로 실행되는 작업을 처리하고 그 결과를 콜백 함수 등을 이용해 처리할 수 있습니다.

Promise는 총 3가지 상태를 갖습니다.

1. 대기(pending): Promise 객체가 생성되었으나 아직 처리가 진행되지 않은 상태입니다.
2. 이행(fulfilled): Promise 객체가 처리를 완료하여 결과를 반환한 상태입니다.
3. 거부(rejected): Promise 객체가 처리를 실패하였거나 오류가 발생한 상태입니다.

자바스크립트 엔진은 싱글 스레드로 동작하며, 이벤트 루프를 통해 비동기 작업을 처리합니다.

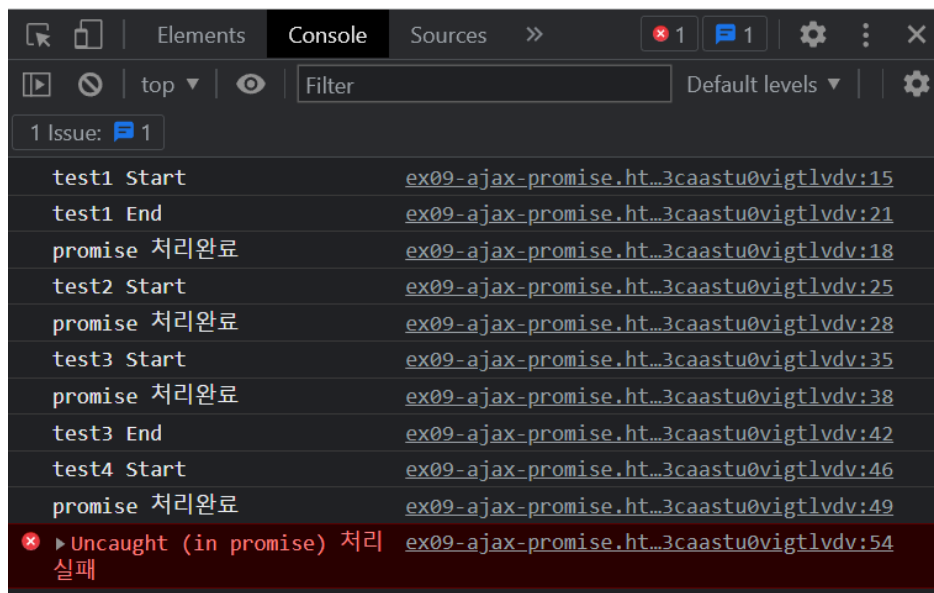
이벤트 루프는 Call Stack, Message Queue, Event Loop로 구성되어 있습니다.

Call Stack은 현재 실행 중인 함수를 기록하는 스택입니다.

Message Queue는 비동기 작업이 완료되었을 때 등록되는 작업 목록입니다.

Event Loop는 Call Stack과 Message Queue를 주시하면서 Call Stack이 비어있을 때 Message Queue에서 작업을 꺼내와 실행합니다.

그림 설명 필요



Promise1 Test

Promise2 Test

Promise3 Test

Promise4 Test

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

<button onclick="test1()">Promise1 Test</button>
<button onclick="test2()">Promise2 Test</button>
<button onclick="test3()">Promise3 Test</button>
<button onclick="test4()">Promise4 Test</button>
<script>
  function test1() {
    console.log('test1 Start');
    new Promise(() => {
      setTimeout(() => {
        console.log("promise 처리완료");
      }, 2000);
    });
    console.log('test1 End');
  }

  async function test2() {
    console.log('test2 Start');
    await new Promise(() => {
      setTimeout(() => {
        console.log("promise 처리완료");
      }, 2000);
    });
    console.log('test2 End');
  }

  async function test3() {
    console.log('test3 Start');
    await new Promise((resolve) => {
      setTimeout(() => {
        console.log("promise 처리완료");
        resolve("처리 완료");
      }, 2000);
    });
    console.log('test3 End');
  }

  async function test4() {
    console.log('test4 Start');
```



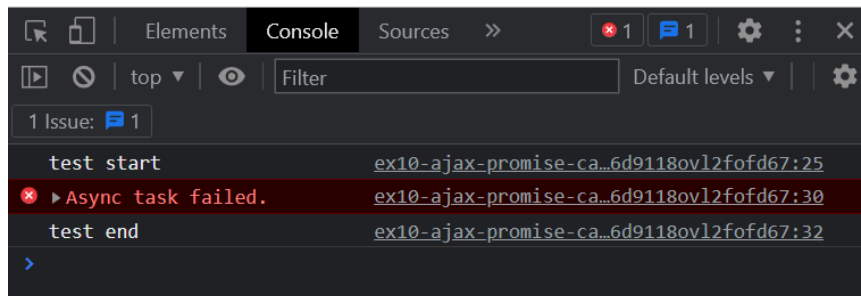
```

        await new Promise((resolve, reject) => {
            setTimeout(() => {
                console.log("promise 처리완료");
                reject("처리 실패");
            }, 2000);
        });
        console.log('test4 End');
    }
</script>
</body>
</html>

```

10. Promise Callback

Promise가 실행된 뒤, 거부 당하거나, 성공했을 때의 콜백 메서드 만들어보기



Promise Callback Test

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

    <button onClick="runAsyncTask()">Promise Callback Test</button>

<script>
    function doAsyncTask() {
        return new Promise(function(resolve, reject) {
            setTimeout(function() {
                let success = false;
                if (success) {
                    resolve('Async task completed successfully.');

```

```

        reject('Async task failed.');
```

```

    }
    }, 1000);
});
}

async function runAsyncTask() {
    console.log("test start");
    try {
        const result = await doAsyncTask();
        console.log(result);
    } catch (error) {
        console.error(error);
    }
    console.log("test end");
}
</script>
</body>
</html>
```

11. Ajax Fetch

AJAX는 "Asynchronous JavaScript and XML"의 약자로, 자바스크립트를 사용하여 비동기적으로 서버와 데이터를 주고받는 기술입니다.

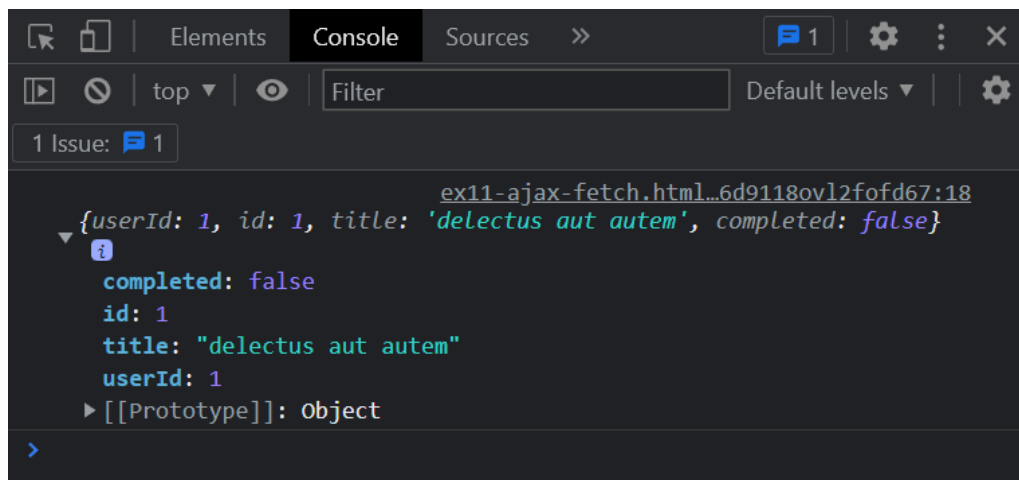
기존에는 웹 페이지에서 사용자 입력을 받고, 서버로 데이터를 전송하여 새로운 페이지를 받아와서 보여주는 방식이었습니다. 이러한 방식은 화면이 깜빡이는 등의 불편한 경험을 제공하며, 데이터의 양이 많아지면 더욱 느려질 수 있습니다.

AJAX는 이러한 문제점을 개선하기 위해, 웹 페이지에서 동적으로 데이터를 가져오거나 업데이트하는 기술입니다. AJAX를 사용하면, 서버와 비동기적으로 데이터를 주고받아 필요한 부분만 화면을 갱신할 수 있으며, 화면 전환이 발생하지 않아 사용자가 불편한 경험을 하지 않습니다.

AJAX는 자바스크립트를 사용하여 서버에 요청을 보내고, 이에 대한 응답을 처리합니다. 이를 위해 `XMLHttpRequest` 객체를 사용하여 서버와 통신합니다. 최근에는 `fetch` API를 사용하는 방법이 많이 사용되고 있습니다.

AJAX를 사용하면 사용자 경험이 개선되고, 서버의 부하가 감소하여 웹 애플리케이션의 성능을 개선할 수 있습니다.

그림 설명 필요 (CSR과 SSR 차이)



통신하기

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

<button onclick="fetchTodo()">통신하기</button>
<script>
  // https://jsonplaceholder.typicode.com
  // async -> 비동기함수가 되기 위한 키워드
  // await -> Promise가 처리될 때까지 기다리는 키워드
  async function fetchTodo() {
    try {
      let promiseData = await
fetch("https://jsonplaceholder.typicode.com/todos/1");
      let todos = await promiseData.json();
      console.log(todos);
    } catch (err) {
      console.log(err);
    }
  }
</script>
</body>
</html>
```

12. AJAX Fetch Render

<https://jsonplaceholder.typicode.com/>

통신하기

delectus aut autem

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <style>
    .box {
      border: 1px solid black;
      padding: 10px;
    }
  </style>
</head>
<body>
  <input type="text" id="num" rrr="100" value="1"/> <button onclick="fetchTodo()">통신하기
</button>
  <div class="box" id="box">

</div>
<script>
  async function fetchTodo() {
    let el = document.querySelector("#num");
    let id = el.value;
    let rrr = el.getAttribute("rrr");
    console.log("기본속성이 아니면 getAttribute()", rrr);
    try {
      let promiseData = await
fetch("https://jsonplaceholder.typicode.com/todos/"+id);
      let todo = await promiseData.json();
      render(todo);
    } catch (err) {
      console.log(err);
    }
  }
```

```

    }

    function render(todo){
      let le = document.querySelector("#box");
      le.innerHTML = todo.title;
    }
  </script>
</body>
</html>

```

13. ToDoList 만들어보기

디자인은 참고로 ChatGpt 사용함

To-Do List

todo 가져오기

☐

delectus aut autem

☒

~~quis ut nam facilis et officia qui~~

☒

~~fugiat veniam minus~~

☐

et porro tempora

☐

laboriosam mollitia et enim quasi adipisci quia
provident illum

☐

qui ullam ratione quibusdam voluptatem quia
omnis

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">

```

```

<title>To-Do List</title>
<link rel="stylesheet" href="/css/style.css">
</head>
<body>
<div class="container">
  <h1>To-Do List</h1>
  <button onclick="todoAllFetch()">todo 가져오기</button>
  <ul class="todo-list" id="todoBox">

  </ul>
</div>
<script>
  async function todoAllFetch() {
    let promiseData = await fetch("https://jsonplaceholder.typicode.com/todos");
    let todos = await promiseData.json();
    render(todos);
  }

  function render(todos) {
    let el = document.querySelector("#todoBox");
    for (const todo of todos) {
      let newEl = document.createElement("li");
      newEl.innerHTML = `
        <input type="checkbox" id="todo-${todo.id}" style="text-decoration:
none">

        <label for="todo-1">${todo.title}</label>
      `;
      el.append(newEl);
    }
    addCheckBoxEvent();
  }

  function addCheckBoxEvent(){
    const checkboxes = document.querySelectorAll('input[type="checkbox"]');
    checkboxes.forEach((checkbox) => {
      checkbox.addEventListener('click', () => {
        if (checkbox.checked) {
          checkbox.nextElementSibling.style.textDecoration = 'line-through';
        } else {
          checkbox.nextElementSibling.style.textDecoration = 'none';
        }
      });
    });
  }

</script>
</body>

```

```
</html>
```

```
.container {
  width: 400px;
  margin: 0 auto;
  text-align: center;
  font-family: Arial, sans-serif;
}

h1 {
  font-size: 2rem;
  margin: 0;
  padding: 20px 0;
  color: #444;
}

ul {
  list-style: none;
  padding: 0;
  margin: 0;
}

li {
  border-bottom: 1px solid #ddd;
  padding: 10px;
  font-size: 16px;
  display: flex;
  align-items: center;
}

li:last-child {
  border-bottom: none;
}



label {
  margin-left: 10px;
}

input[type="checkbox"] {
  margin: 0;
  transform: scale(1.5);
  margin-right: 10px;
}
```




위 디자인에서 checkbox 추가하고, checkbox 클릭시 선을 그어서 완료표시할 수 있는 이벤트 만들어줘



다음은 To-Do List 디자인에 checkbox를 추가하고, checkbox를 클릭하여 선을 그어 완료표시할 수 있는 기능을 추가한 예시입니다.  

html


 Copy code

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>To-Do List</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div class="container">
      <h1>To-Do List</h1>
      <ul class="todo-list">
        <li>
          <input type="checkbox" id="todo-1">
          <label for="todo-1">할 일 1</label>
        </li>
        <li>
          <input type="checkbox" id="todo-2">
          <label for="todo-2">할 일 2</label>
        </li>
        <li>
          <input type="checkbox" id="todo-3">
          <label for="todo-3">할 일 3</label>
        </li>
      </ul>
    </div>
    <script>
      const checkboxes = document.querySelectorAll('input[type="checkbox"]');
      checkboxes.forEach((checkbox) => {
        checkbox.addEventListener('click', () => {
          if (checkbox.checked) {
            checkbox.nextElementSibling.style.textDecoration = 'line-through';
          } else {
            checkbox.nextElementSibling.style.textDecoration = 'none';
          }
        });
      });
    </script>
  </body>
</html>
```



```
});  
</script>  
</body>  
</html>
```

CSS

 Copy code

```
.container {  
  width: 400px;  
  margin: 0 auto;  
  text-align: center;  
  font-family: Arial, sans-serif;  
}  
  
h1 {  
  font-size: 2rem;  
  margin: 0;  
  padding: 20px 0;  
  color: #444;  
}
```