A.I with Javascript

수업의 목표

자바스크립트의 **기본 문법**과 **함수의 동작 및 구현 방법**을 터득합니다. 더불어 **텐서플로우 JS와 Danfo.js**도 함께 살펴보며 자바스크립트의 유연함을 동시에 공부하여 보겠습니다.

JS

TF.JS

Danfo.
js

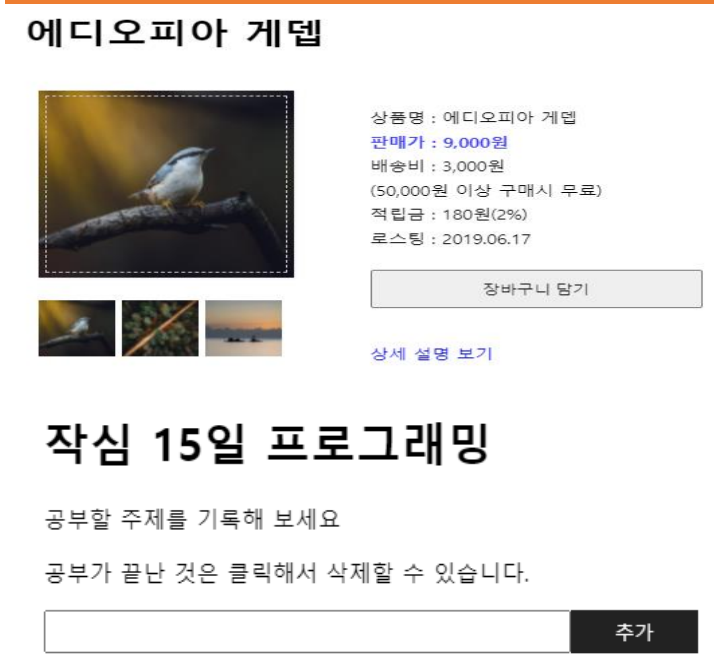
p5js

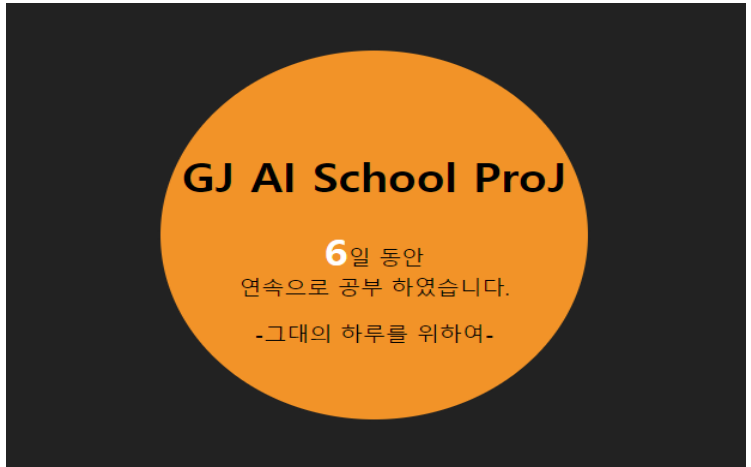
https://www.tensorflow.org/js/models?hl=ko

https://js.tensorflow.org/api/latest/?hl=ko

https://editor.p5js.org/ml5/sketches

수업을 통해 만들어 낼 수 있는 결과물





XOR GATE
(https://codepen.io/AI_R_IM/pen/mdBEJKz)

XOR part 1 ~ 5

https://codepen.io/AI_R_IM/pen/eYGzpve?editors=0011

https://codepen.io/AI_R_IM/pen/NWarGjr?editors=1011

https://codepen.io/AI_R_IM/pen/bGoeVrp?editors=1111

https://codepen.io/AI_R_IM/pen/QWqEjqd?editors=1111

https://codepen.io/AI_R_IM/pen/bGoeVYe?editors=1111

```
"학습 전 ▼"
"Num correctly predicted : 51 of 100"
"Accuracy : 0.51"
"===================="
"Iteration : 0 loss : 1.5372608900070 19"
"Iteration : 10 loss : 0.35652756690979004"
"Iteration : 90 loss : 0.14262928068637848"
"===================="
"학습 후 ▼ "
"Num correctly predicted : 97 of 100"
"Accuracy : 0.97"
```

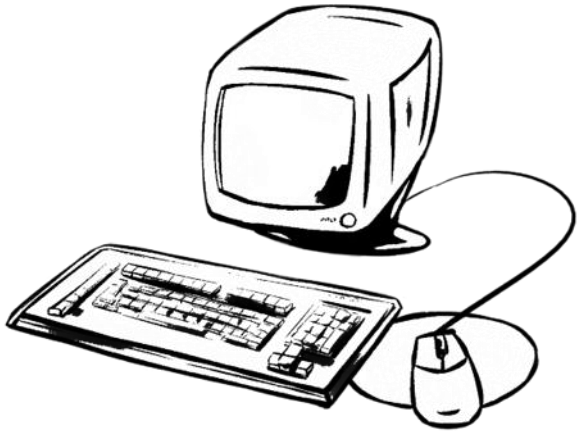
0.JavaScript

웹 문서의 뼈대를 이루는 것은 **HTML**입니다. 여기에 **CSS**를 사용하면 **웹 문서를 일목요연하게 정리**하고 **미적인 효과를 추가**할 수 있습니다. 하지만 다른 사이트보다 **내용을 강조**하고 **사용자의 동작에 즉시 반응**하려면 **자바스크립트**를 사용해야 합니다.

웹 문서의 뼈대인 HTML과 웹 문서를 꾸며 주는 CSS를 바탕으로 **웹 개발에 꼭 필요한 자바스크립트**를 살펴보도록 하겠습니다.

유튜브나 페이스북 같은 규모가 큰 웹 서비스를 개발하려면 리엑트(React)나 앵귤러(Angular)와 같은 도구를 사용해야 합니다. 이것들도 모두 **자바스크립트를 기본**으로 하고 있습니다. 그렇기에 웹 개발에 있어서 자바스크립트는 기본이 되고 있습니다.

자비스크립트는 원래 웹 사이트에 움직이는 효과를 줄 때 사용하는 언어였습니다.
그래서 처음에는 자바스크립트가 웹을 공부할 때 필수 요소는 아니었습니다.
하지만 지금은 자바스크립트로 서버나 웹에서 작동하는 프로그램을 만드는 등
웹의 모든 부분을 다룰 수 있는 핵심 언어가 되었습니다.



0-1. 웹 브라우저 그리고 JavaScript

웹 브라우저에는 **자바스크립트 소스**를 읽고 처리하는 **해석기**(JavaScript Interpreter)가 있습니다. 따라서 자바스크립트 소스는 웹 문서에서 **태그를 이용**해 작성할 수 있습니다. 또는 자바스크립트 소스만 별도 스크립트 파일로 작성한 후 **웹 문서와 연결**해서 사용할 수도 있습니다.

웹 문서 안에 <script> 태그로 자바스크립트 작성하기

자바스크립트 소스 코드가 짧을 경우 웹 문서에서 자바스크립트를 실행할 위치에 바로 코드를 작성할 수 있습니다. 웹 문서에서 <script>와 </script> 태그 사이에 실행할 자바스크립트 소스를 작성하는 것이죠. <script> **태그는 웹 문서 안의 어디든 위치**할 수 있고 **삽입된 위치 그 자리에서 바로 스크립트가 실행**됩니다. 또한 <script> 태그는 하나의 문서에서 여러 개 사용할 수도 있습니다.

자바스크립트는 웹 문서에서 **이미지나 텍스트** 등의 요소를 제어하는 경우가 많으므로 되도록이면 이미지나 텍스트 등을 다 표시한 후에 실행하는 것이 좋습니다. 그래서 **</body> 태그 직전에 자바스크립트 소스**를 삽입하는 경우도 있습니다.

0-1. 웹 브라우저 그리고 JavaScript

HTML, CSS와 달리 **자바스크립트**에서는 **영어 대소 문자를 구별**하므로 소스를 작성할 때 주의해야 합니다. 예를 들어 sum이나 Sum, SUM은 모두 다르게 인식합니다. 따라서 **변수 이름이나 함수를 지정할 때에는 영문 대소 문자를 정확하게 구별**해야 합니다.

다음은 텍스트를 클릭했을 때 글자색을 바꾸는 예제입니다. **<script>와 </script> 태그 사이에 자바스크립트 소스**를 작성하고 </body> 태그 바로 앞에 삽입하는 것을 확인할 수 있습니다.

HTML https://codepen.io/AI_R_IM/pen/abLvpYo

```
16 <body>
17   <h1 id="heading">인공지능</h1>
18   <p id="text">위 텍스트를 클릭해 보세요</p>
19
20   <script>
21     var heading = document.querySelector('#heading');
22     heading.onclick = function() {
23       heading.style.color = "orange";
24     }
25   </script>
26 </body>
```

0-1. 웹 브라우저 그리고 JavaScript

이렇게 HTML 문서 안에 자바스크립트 소스를 작성하면 웹 문서에서 바로 확인할 수 있는 장점도 있지만 단점이 더 많습니다. **우선 HTML 태그와 자바스크립트 소스가 함께 섞여 있어서 웹 문서가 복잡해 보인다는 것입니다.** 특히 소스를 수정할 때 필요한 코드를 찾아보기가 쉽지 않죠.
그리고 **여러 웹 문서에서 같은 자바스크립트 소스를 사용하는 경우**에 똑같은 소스를 **반복해서 삽입**해야 합니다. 이때 자바스크립트 소스를 수정해야 한다면 **이 소스가 포함된 모든 웹 문서를 찾아다니며 수정**해야 합니다.
그래서 자바스크립트 소스를 작성할 때 **외부 스크립트 파일로 저장**해서 **웹 문서와 연결하는 방법을 많이 사용**합니다.

외부 스크립트 파일로 연결해서 자바스크립트 작성하기

CSS와 마찬가지로 자바스크립트 소스도 따로 파일로 저장한 후 웹 문서에 연결해서 사용할 수 있습니다. 이렇게 하면 웹 문서 안에는 직접 자바스크립트 소스가 드러나지 않고 HTML 태그와 CSS 만 유지할 수 있습니다.
외부 자바스크립트 파일은 <script> 태그 없이 자바스크립트 소스만 작성하고 확장자는 **000.js 파일로 저장**합니다.

0-1. 웹 브라우저 그리고 JavaScript

이번에는 **외부 스크립트를 연결해 글자색을 바꾸는 예제**를 알아보겠습니다. 여기에서는 **자바스크립트 소스를 change-color.js라는 외부 스크립트 파일로 저장**하고 웹 문서에서는 **<script> 태그를 이용해 연결**해 보세요.

(※ 본 수업 파일은 codepan을 활용하여 진행하기에 별도의 파일 및 폴더를 생성하지 않았습니다.)

HTML

https://codepen.io/AI_R_IM/pen/MWEaJBx

script-2.html

```html
<body>
  <h1 id="heading">인공지능</h1>
  <p id="text">위 텍스트를 클릭해 보세요</p>

  <script src="js/change-color.js"></script>
</body>
```

이 코드가 외부 스크립트 파일을 연결하여 줍니다.

CSS

JS

change-color.js

```js
var heading = document.querySelector('#heading');
heading.onclick = function() {
  heading.style.color = "orange";
}
```

0-1. 웹 브라우저 그리고 JavaScript

웹 브라우저에서 스크립트를 해석하는 과정

웹 문서에 **자바스크립트 소스가 포함**되어 있으면 **웹 브라우저는 어떤 과정으로 해석**하고 그 결과를 보여 줄까요?

앞에서 작성한 파일을 바탕으로 살펴보겠습니다.

HTML https://codepen.io/AI_R_IM/pen/abLvpYo

```html
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>글자색 바꾸기</title>
  <style>
    body { text-align:center; }
    #heading { color:blue; }
    #text {
      color:gray;
      font-size:15px;
    }
  </style>
</head>
<body>
  <h1 id="heading">인공지능</h1>
  <p id="text">위 텍스트를 클릭해 보세요</p>

  <script>
    var heading = document.querySelector('#heading');
    heading.onclick = function() {
      heading.style.color = "orange";
    }
  </script>
</body>
</html>
```

0-1. 웹 브라우저 그리고 JavaScript

웹 브라우저에서 스크립트를 해석하는 과정

1. 웹 브라우저는 1행에 있는 <!DOCTYPT html> 를 보고 이 문서가 **웹 문서라는 것**을 알게됩니다. 그리고 <html>과 </html> 태그 사이의 내용을 **HTML 표준에 맞춰 읽기** 시작합니다.

2. 웹 문서에서 HTML 태그의 순서와 포함 관계를 확인합니다. <head>와 </head> 태그, <body>와 </body> 태그 사이에 각각 어떤 태그가 있는지 확인합니다.
그리고 **태그 간에 관계는 어떻게 되어 있는지 등을 분석**합니다.

3. 태그 분석이 끝나면 <style> 과 </style> 태그 사이에 정보를 분석합니다.

HTML https://codepen.io/AI_R_IM/pen/abLvpYo

```
1  <!DOCTYPE html>
2  <html lang="ko">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>글자색 바꾸기</title>
7    <style>
8      body { text-align:center; }
9      #heading { color:blue; }
10     #text {
11       color:gray;
12       font-size:15px;
13     }
14   </style>
15 </head>
16 <body>
17   <h1 id="heading">인공지능</h1>
18   <p id="text">위 텍스트를 클릭해 보세요</p>
19
20   <script>
21     var heading = document.querySelector('#heading');
22     heading.onclick = function() {
23       heading.style.color = "orange";
24     }
25   </script>
26 </body>
27 </html>
```

HTML 분석기

CSS 분석기

JS 분석기

0-1. 웹 브라우저 그리고 JavaScript

웹 브라우저에서 스크립트를 해석하는 과정

4. <script> 태그를 만나면 **웹 브라우저 안에 포함된 자바스크립트 해석기에게 스크립트 소스를 넘깁니다.**
자바스크립트 해석기는 <script>와 </script> 사이의 소스를 해석합니다.

5. **2에서 분석한 HTML과 3에서 분석한 CSS 정보**에 따라 **웹 브라우저 화면에 표시**합니다.

6. 이제 웹 브라우저에서 **'인공지능' 텍스트를 클릭**하면 **분석해 놓은 자바스크립트를 실행**해서 그 **결과를 화면에 표시**합니다.

HTML https://codepen.io/AI_R_IM/pen/abLvpYo

```html
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>글자색 바꾸기</title>
  <style>
    body { text-align:center; }
    #heading { color:blue; }
    #text {
      color:gray;
      font-size:15px;
    }
  </style>
</head>
<body>
  <h1 id="heading">인공지능</h1>
  <p id="text">위 텍스트를 클릭해 보세요</p>

  <script>
    var heading = document.querySelector('#heading');
    heading.onclick = function() {
      heading.style.color = "orange";
    }
  </script>
</body>
</html>
```

HTML 분석기

CSS 분석기

JS 분석기

0-2. JavaScript 용어와 기본 입출력 방법

자바스크립트를 본격적으로 공부하기 전에 **프로그래밍과 관련된 몇 가지 기본 용어**를 알아보도록 하겠습니다. 그리고 자바스크립트 문법을 배우기 전에 **프로그램을 실행하는 데 필요한 자료를 입력하고 결과를 출력하는 간단한 방법**도 살펴보겠습니다.

식과 문

자바스크립트 언어의 큰 줄기는 **식**(expression)과 **문**(statement)입니다. 자바스크립트에서 **'식'은 '표현식'**이라고도 하는데, **연산식 뿐만 아니라 실제 값도, 함수를 실행하는 것도 식**이 됩니다. 즉, **어떤 값을 만들어 낼 수 있다면 모두 식**이 될 수 있으며, **식은 변수에 저장**됩니다. 다음은 자바스크립트의 여러가지 식을 나타내는 예제입니다.

자바스크립트의 다양한 식 나타내기

```
weight * 1.12 // 연산식은 식 입니다.
"hello A.I"   // 문자열도 식 입니다.
0.0001        // 숫자도 식 입니다.
```

이에 비해 **문**은 **명령**이라고 생각할 수 있습니다. 문의 끝에는 세미콜론(;)을 붙여 구분합니다.

'조건문'이나 **'제어문'** 등이 이에 해당하게 됩니다.

0-2. JavaScript 용어와 기본 입출력 방법

간단한 입출력 방법

소스를 입력하다 보면 **사용자에게 입력**받아야 하거나 **자바스크립트 실행 결과를 웹 브라우저에 표시**해야 할 때가 있습니다. 자바스크립트의 기본 입출력 방법을 살펴보겠습니다.

알림 창 출력하기

알림 창(alert)은 가장 많이 사용하는 **간단한 대화 상자**입니다. 웹 브라우저에서는 **작은 알림 창을 열어 메시지를 표시할 때 사용**합니다.

[기본형] alert(메시지)

알림 창을 만드는 방법은 다음과 같이 **alert()를 입력**하고 **괄호 안에 따옴표(“ ” 또는 ‘ ’)와 함께 메시지**를 넣어주면 됩니다.

JS

https://codepen.io/AI_R_IM/pen/yLzYMYd

```
alert("Hello A.I")
```

0-2. JavaScript 용어와 기본 입출력 방법

확인 창 출력하기

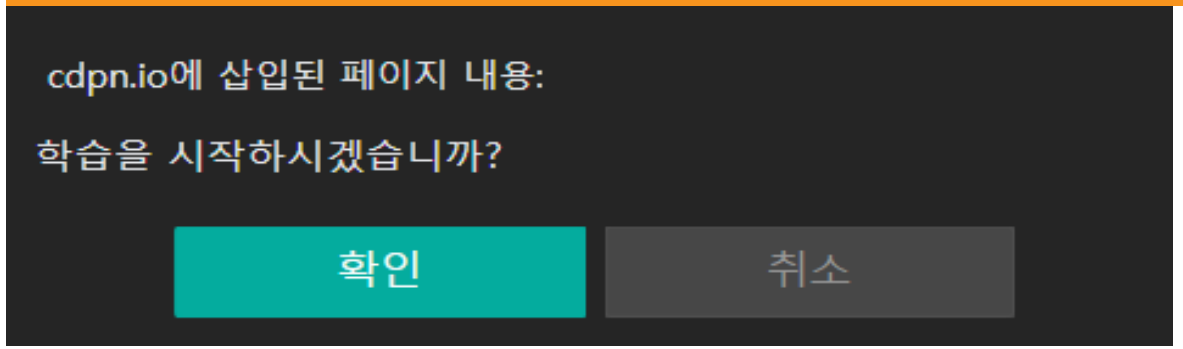
알림 창은 단순히 메시지를 보여 주는 기능만 하지만 **확인 창(confirm)**은 사용자가 [확인]이나 [취소] 버튼 중에서 직접 클릭할 수 있습니다. 그러면 **선택한 결과에 맞게 프로그램이 동작**합니다.

[기본형] confirm(메시지)

확인 창은 다음과 같이 버튼이 2개 있습니다. 사용자가 어떤 버튼을 눌렀는지 **결과를 변수에 저장**한 후 그 값에 따라 프로그램에서 처리합니다.

JS 2 unsaved changes ✕ https://codepen.io/AI_R_IM/pen/NWaGpRo?editors=1111

```
var reply = confirm('학습을 시작하시겠습니까?');
```

실행화면	결괏값
	확인 → True 취소 → False

0-2. JavaScript 용어와 기본 입출력 방법

프롬프트 창에서 입력받기

프롬프트 창(prompt)은 **텍스트 필드**가 있는 작은 창입니다. 텍스트 필드 안에 간단한 메시지를 입력할 수 있으며 **그 내용을 가져와 프로그램에서 사용**할 수 있습니다.

[기본형] prompt(메시지) 또는 prompt(메시지, 기본값)

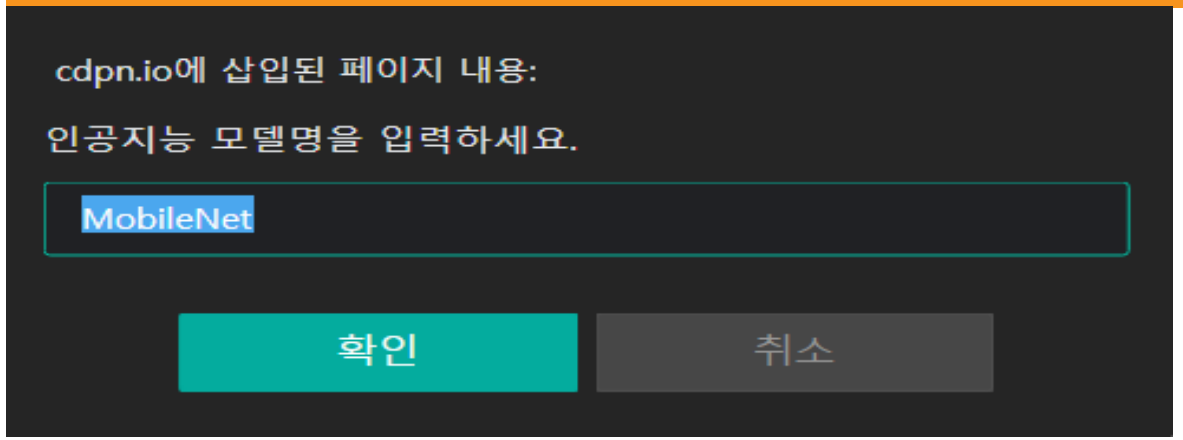
프롬프트 창을 만들 때는 **기본값을 지정**하거나 **지정하지 않을 수 있습니다.** 기본값을 지정하면 아래 그림과 같이 기본값이 표시됩니다. 기본값을 지정하지 않으면 빈 텍스트 필드로 표시됩니다.

JS 4 unsaved changes ✕

https://codepen.io/AI_R_IM/pen/yLzYMbL

```
var name = prompt("인공지능 모델명을 입력하세요.","MobileNet");
```

실행화면



결괏값

name(변수)에는
"MobileNet" 이라는 '문자열 타입' 저장

0-2. JavaScript 용어와 기본 입출력 방법

웹 브라우저 화면에 출력을 담당하는 document.write()문

앞으로 자바스크립트를 활용하게 된다면 document.write() 라는 명령문을 자주 만날 것입니다.
자바스크립트의 실행 결과는 텍스트나 이미지로 출력하거나, 따로 지정한 영역에 내용을 표시하는 경우가 많습니다. 하지만 단순히 브라우저 화면에서 결괏값을 확인하는 용도로는 document.write()을 종종 사용합니다.
우선 웹 문서(document)에서 괄호 안의 내용을 표시(write)하는 명령문 정도로 이해해 주세요.

document.write() 의 괄호 안에는 **실제 웹 브라우저 화면에 표시할 내용**이나 **어떤 결괏값이 저장된 변수**를 넣을 수도 있습니다. 괄호 안에서 큰 따옴표(“ ”)나 작은 따옴표(‘ ’) 사이에 입력한 내용은 웹 브라우저 화면에 그대로 표시됩니다. 물론 **따옴표 안에는 HTML 태그도 함께 사용**할 수 있습니다.
다음은 document.write()문에 <h1> 태그와 **출력할 내용을 입력**해서 **웹 브라우저에 제목을 표시하는 예제**입니다.

코드(https://codepen.io/AI_R_IM/pen/yLzYMzy)

HTML

```
<script>
  document.write("<h1>광주 인공지능 사관학교</h1>")
</script>
```

결괏값

광주 인공지능 사관학교

0-2. JavaScript 용어와 기본 입출력 방법

웹 브라우저 화면에 **표시할 내용과 변수를 섞어서** 나타낼 수도 있습니다. 이때 **+ 연산자**를 사용해서 연결해 줄 수도 있습니다. 여기에서 + 연산자는 더하기 기호가 아닌 **연결 연산자** 입니다. **내용과 변수를 연결**해 주는 역할을 하죠.

다음은 **name 변수에 저장된 값**과 **텍스트를 연결**해서 표시하는 예제입니다. 여기에서도 <big> 태그와 <b> 태그를 함께 사용해보도록 하겠습니다.

코드(https://codepen.io/AI_R_IM/pen/gOGamXd)	결괏값

HTML 1 unsaved changes X

```
<script>
  var name = prompt("사용할 인공지능 모델을 입력하세요.", "MobileNet");
  document.write("<b><big>" + name + "</big></b> 모델을 사용하겠습니다.");
  // document.write(`<b><big> ${name} </big></b> 모델을 사용하겠습니다.`);
</script>
```

MobileNet 모델을 사용하겠습니다.

0-2. JavaScript 용어와 기본 입출력 방법

콘솔 창에 출력하는 console.log( ) 문

console.log() 문은 **괄호 안의 내용을 '콘솔 창'에 표시**합니다. **콘솔 창**은 **웹 브라우저의 개발자 도구 창**에 포함되어 있는 공간입니다. **콘솔 창에서 소스 코드의 오류를 발견**하거나 **변숫값을 확인**할 수도 있습니다. console.log() 문의 괄호 안에는 변수가 들어갈 수도 있고 따옴표 사이에 표시할 텍스트를 넣을 수도 있습니다. 이때 따옴표 안에 HTML 태그는 사용할 수 없습니다.

다음은 프롬프트 창에서 이름을 입력받아 콘솔 창에 표시하는 예제입니다.

코드(https://codepen.io/AI_R_IM/pen/bGoVqvJ?editors=1111)

HTML

```
<body>
  <h1>인공지능 예측 모델</h1>

  <script>
    var name = prompt("예측에 활용할 모델 이름을 입력하세요.","MobileNet");
    console.log(name + "모델 사용을 진행합니다.");
  </script>
</body>
```

결괏값

인공지능 예측 모델

Console

```
"MobileNet모델 사용을 진행합니다."
```

0-2. JavaScript 용어와 기본 입출력 방법

콘솔 창에서 오류를 찾고 수정하기

자바스크립트 소스를 작성한 후 웹 브라우저에서 실행하면 **원하는 결과가 나오지 않거나 아예 실행되지 않는 경우**가 있습니다.

이런 경우 대부분 **오타 때문에 발생할 확률**이 높습니다.

오타를 콘솔 창에서 쉽게 찾아보는 방법을 알아보겠습니다.

분명 많은 코드가 적혀져 있지만 어떠한 동작도 하지 않고 있습니다. 이러한 경우 **console창을 열어보면 우리가 어떠한 실수를 하였는지** 알려주고 있습니다.

오류의 내용은 **간단한 영어 문장으로 표현**되고 있으므로 이를 확인하고 수정하시면 됩니다.

HTML https://codepen.io/AI_R_IM/pen/OJxypEd?editors=1001

```html
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>What time is it?</title>
  <style>
    body {
      font-size: 2em;
      text-align: center;
    }
  </style>
</head>
<body>
  <script>
    var now = new Date();
    var display = now.toLocaleTimeString();
    document.writ("현재 시각은 " + display);
  </script>
</body>
</html>
```

Console Clear ✕

```
Uncaught TypeError: document.writ is not a function
 at https://cdpn.io/cp/internal/boomboom/index.html?
key=index.html-3c106baf-830f-8fd6-6dc2-8b5e4a6da7e8:46
```

1-1. JavaScript 의 변수

자바스크립트 기본 문법

이번에는 **자바스크립트의 기본 문법**을 알아보겠습니다. **자바스크립트에서 데이터를 어떻게 분류**하고 **저장**하는지, 조건에 따라 데이터를 어떻게 처리하는지 살펴봅니다. 복잡하게 반복되는 과정을 간단하게 몇 줄 소스로 처리하는 방법도 함께 살펴보도록 하겠습니다.

변수란

변수(variable)란 **프로그램을 실행하는 동안 값이 여러 번 달라질 수 있는 데이터**를 가리킵니다. 반면에 값을 한번 지정하면 **바뀌지 않는 데이터를 상수**(constant)라고 합니다.

예를 들어 **인공지능의 순전파**(forward neuralnet)를 계산하는 프로그램을 만든다고 가정해 보겠습니다.

예측값 Pred = **독립변수 X** ✖ **가중치 W** ✚ **편향 B**

순전파를 계산하는 식에서 **독립변수 X의 값**은 변하지 않는 **'상수'** 입니다. 하지만 **가중치 W** 와 **편향 B**의 값은 **언제든 바뀔 수 있는 값**이니 **'변수'** 가 됩니다.

1-1. JavaScript 의 변수

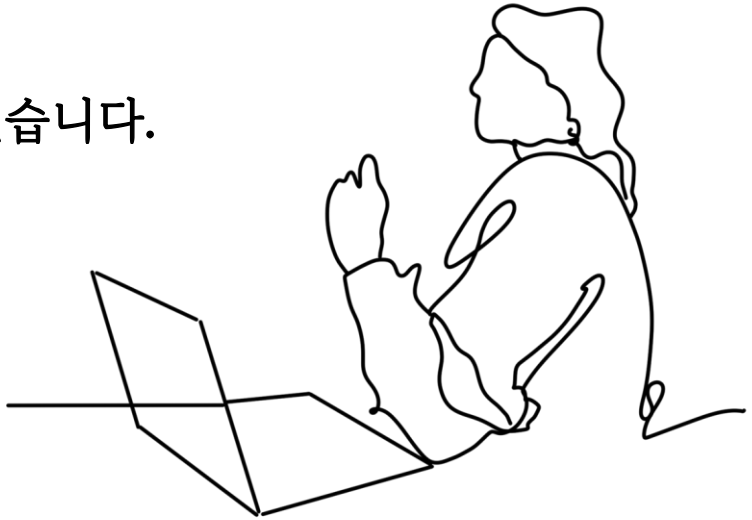
변수 선언의 규칙

변수를 사용하려면 **변수를 구별할 수 있도록 이름**을 붙여 주어야 하는데, 이것을 **변수 선언** 이라고 합니다. 변수 선언은 값을 지정할 **컴퓨터 메모리 공간에 스티커를 붙이는 것**과 같습니다.

우리는 프로그램 안에서 사용할 값이 메모리의 어느 위치에 저장되는지 신경 쓰지 않아도 됩니다. **변수 이름만 기억**해 놓으면 **값을 쉽게 가져와서 사용**할 수 있기 때문입니다. 또한 바뀐 값을 다시 같은 변수에 저장할 수 도 있습니다. 따라서 프로그램에서 사용할 **변수 이름은 서로 다르게** 만들어야 합니다.

자바스크립트에서 **변수를 선언할 때는 지켜야 할 몇 가지 규칙**이 있습니다.

예를 들어 **영어 대소 문자를 구별**해야 하고 **자바스크립트의 예약어는 사용할 수 없습니다.**



1-1. JavaScript 의 변수

예측값 Pred	=	독립변수 X	×	가중치 W	+	편향 B
var pred		var input_x		var weight		var bias

변수 선언하기

자바스크립트에서 변수 선언은 다음과 같이 **var 라는 예약어** 뒤에 **변수 이름**을 적으면 됩니다.

[기본형] var 변수명

예를 들어 다음과 같이 **변수 선언**을 할 수 있죠.

또는 **변수 4개를 한꺼번에 선언**할 수도 있습니다. 다음과 같이 var를 한 번만 쓰고 **여러 변수를 쉼표로 구분하여 같은 줄에 선언**합니다.

변수 선언하기

JS

```js
var pred;        // 예측 결괏값(변수)
var input_x;     // 독립변수 x(상수)
var weight;      // 가중치 w(변수)
var bias;        // 편향 b(변수)
```

JS

```js
var pred, input_x, weight, bias;
```

1-1. JavaScript 의 변수

변수를 선언했으면 **'='** 기호를 사용해서 변수에 값을 저장할 수도 있습니다. 이것을 **값 할당** 이라고 합니다. 값 할당은 **변수를 선언한 후에 따로** 할 수도 있고, **변수를 선언하면서 동시**에 할 수도 있습니다.

예를 들어 **독립변수 X, 가중치 W와 B의 값을 임의로 설정**하고 **변수에 값 할당**을 한다면 **다음과 같이 코드를 작성** 할 수 있습니다.

변수 선언과 값 할당

JS

```js
// 변수 선언과 값 할당 따로 하기
var input;
input_x    = 2023;
// 변수 선언과 값 할당 같이 하기
var weight = 0.123;
var bias   = 0.0;
```

1-1. JavaScript 의 변수

Forward Neuralnet 계산 프로그램 만들기

이제 변수의 개념을 익혔으니 **순전파를 계산하는 자바스크립트 프로그램**을 만들어 보겠습니다.

여기서 **독립변수 X 의 값은 입력(변수)**받으며, **가중치와 편향의 값은 각각 0.123, 0.00 으로 고정(상수)**하도록 하겠습니다.

결과로 나타나는 **예측값(pred)은 독립변수 X의 값에 따라 달라지게 된다는 점** 또한 고려해야 합니다.

예측값 Pred	=	독립변수 X	×	가중치 W	+	편향 B
var pred		var input_x		var weight		var bias
		prompt()		0.123		0.00

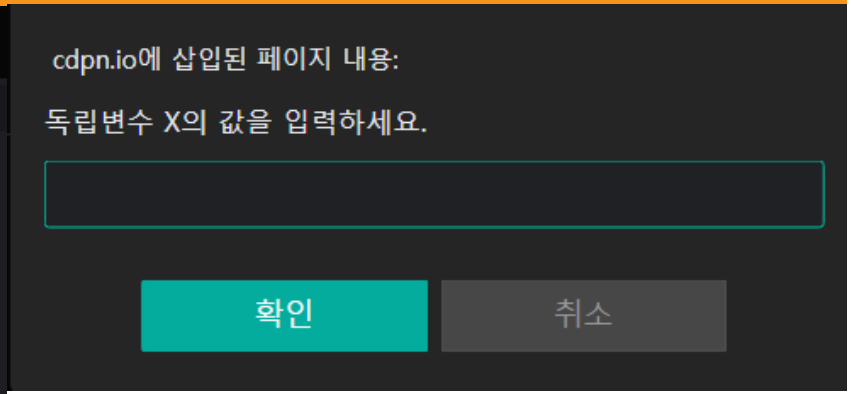
1-1. JavaScript 의 변수

HTML 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/OJxymyK)

HTML

```html
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Forward Neuralnet</title>
</head>
<body>
  <script>
    var weight = 0.123;
    var bias   = 0.0;
    var input_x;
    var pred;

    input_x = prompt("독립변수 X의 값을 입력하세요.", "");
    pred = input_x * weight + bias;
    document.write(`입력된 독립변수는 ${input_x} 입니다.`);
    document.write(`<p>가중치 W(${weight})와 편향 B(${bias})의 값이 다음과 같다면</p>`);
    document.write(`<p>예측값(pred)은 ${pred} 입니다.</p>`)
  </script>
</body>
</html>
```

결과 확인하기



입력된 독립변수는 111 입니다.

가중치 W(0.123)와 편향 B(0)의 값이 다음과 같다면

예측값(pred)은 13.653 입니다.

1-2. JavaScript 의 자료형 이해하기

자료형이란

우리는 '10' 이나 '-15'가 숫자인 것을 알 수 있고, '인공지능'이 문자라는 것을 바로 알 수 있습니다. 하지만 컴퓨터에게 일을 시킬 때는 **'10'은 숫자**이고, **'인공지능'은 문자**라는 것을 **따로 알려 줘야 처리**할 수 있습니다.
이렇게 컴퓨터가 처리할 수 있는 **자료의 형태를 자료형**(data type)이라고 합니다.

자바스크립트의 자료형은 **숫자, 문자열, 논리형**과 같은 **기본 유형**과 **배열, 객체를 다루는 복합 유형** 그리고 **undefined, null 같은 특수 유형**이 있습니다.

숫자형

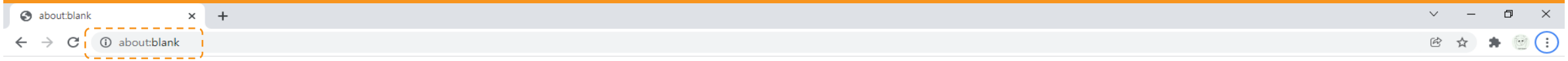
자바스크립트에서 숫자형(number)은 정수와 실수로 나눠 구분합니다. 2가지 숫자형을 살펴보겠습니다.

1-2. JavaScript 의 자료형 이해하기

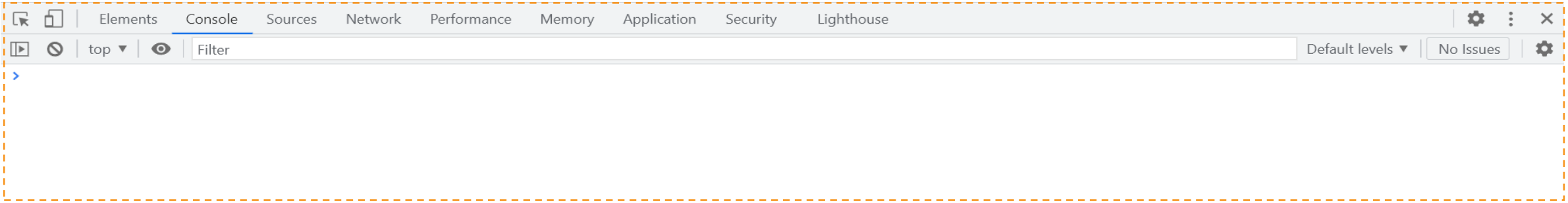
정수

정수는 소수점 없는 숫자입니다. 웹 브라우저 창을 열고 주소 표시줄에 'about:blank'를 입력하면 빈 문서가 나옵니다. 이 상태에서 Ctrl + Shift + J 를 누르면 하단에 콘솔 창이 나타납니다.

웹 브라우저의 빈 문서에서 콘솔 창 열기



콘솔 창



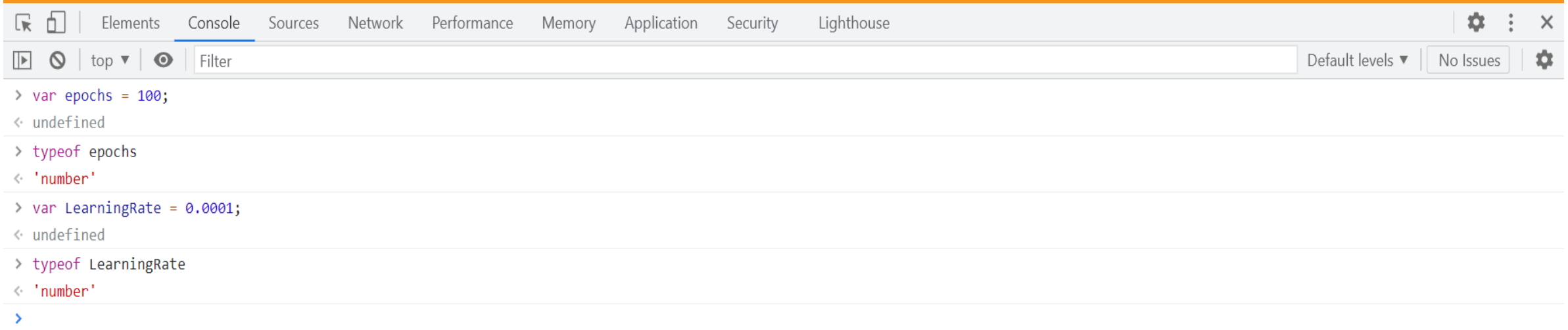
이 상태에서 'typeof' 연산자와 숫자 '100'을 입력하고 Enter를 눌러보세요. 숫자 100의 자료형인 number가 다음 행에 나타나게 될 것입니다. 즉, 자바스크립트에서는 100을 숫자형으로 인식한다는 것이죠.

1-2. JavaScript 의 자료형 이해하기

이 상태에서 'typeof' 연산자와 숫자 '100'을 담는 변수 하나를 생성하고 **Enter**를 눌러보세요. 인공지능에서 **학습의 횟수를 의미하는 변수 epochs의 자료형인 number**가 다음 행에 나타나게 될 것입니다.

즉, 자바스크립트에서는 100을 숫자형으로 인식한다는 것이죠.

콘솔 창에서 typeof() 로 자료형 확인하기 – 숫자형



실수

실수는 **소수점이 있는 숫자**를 가리킵니다. 예를 들어 **학습률(η)의 경우 기본적으로 0.0001이란 값**을 기본적으로 사용합니다. 이 값은 **정수와 마찬가지로 숫자형**입니다. C나 자바와 같은 프로그래밍 언어에서는 정수와 실수를 명확하게 구별하고 처리 방법도 다르지만, **자바스크립트에서는 정수와 실수를 함께 묶어 숫자형** 이라고 합니다.

1-2. JavaScript 의 자료형 이해하기

문자열

문자열은 **작은 따옴표**나 **큰 따옴표**로 묶은 데이터를 의미합니다. 그래서 **숫자도 작은 따옴표나 큰 따옴표로 묶으면 문자열로 인식**합니다. 단, **작은 따옴표로 시작한 문자열 데이터는 작은 따옴표로 끝맺음**을 해야 합니다. 큰 따옴표도 마찬가지 입니다.

콘솔 창에서 문자열 입력하기

Elements Console Sources Network Performance Memory Application Security Lighthouse

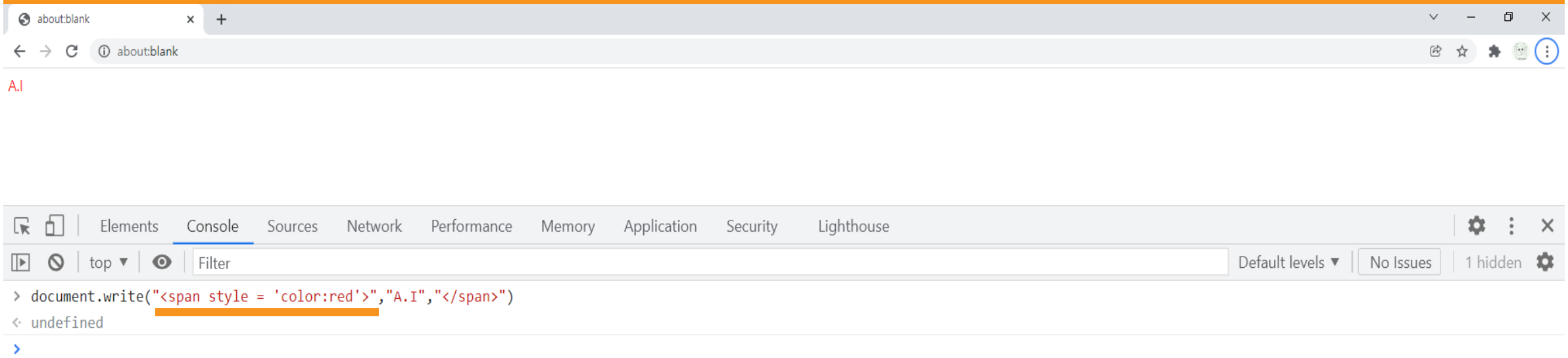
top ▼ Filter Default levels ▼ No Issues

```
> var layer_1_activation = 'sigmoid';
  var layer_2_activation = 'softmax';
  var epochs = '1000';
<· undefined
> typeof layer_1_activation, layer_2_activation
<· 'softmax'
> typeof(layer_1_activation);
<· 'string'
> typeof(layer_2_activation);
<· 'string'
> typeof(epochs);
<· 'string'
>
```

1-2. JavaScript 의 자료형 이해하기

만약 **큰 따옴표로 묶은 문자열 안에 또 다른 문자열**을 넣으려면 어떻게 해야 할까요? 예를 들어 문자열 안에 처럼 다른 문자열을 넣으려면 **큰 따옴표가 중복되지 않도록 속성값은 작은 따옴표로 묶어서 표현**해야 합니다.

따옴표를 중복해서 활용하기

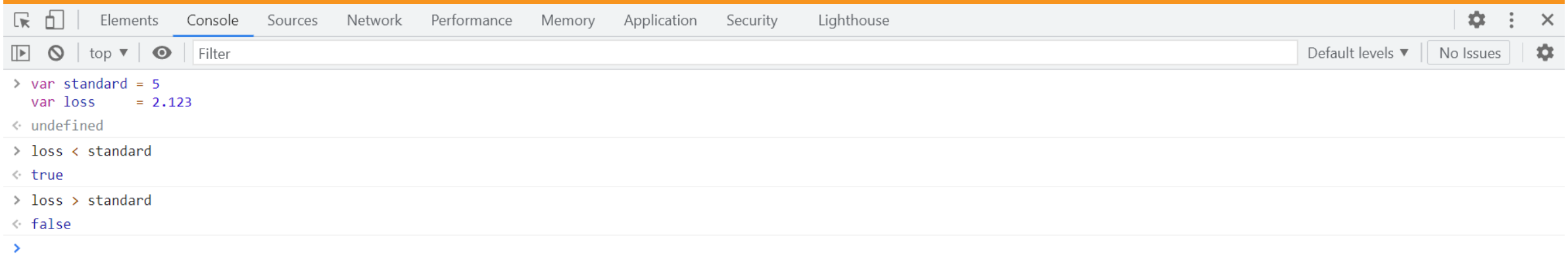


1-2. JavaScript 의 자료형 이해하기

논리형

논리형은 **불린유형**이라고도 하며, **참(True)이나 거짓(False)의 값을 표현하는 자료형**입니다. 어떤 조건을 확인해서 그 **조건이 맞으면 true, 맞지 않으면 false의 결괏값**을 나타냅니다. 주로 프로그램에서 조건을 확인할 때 논리형 데이터를 사용합니다.

논리형 데이터



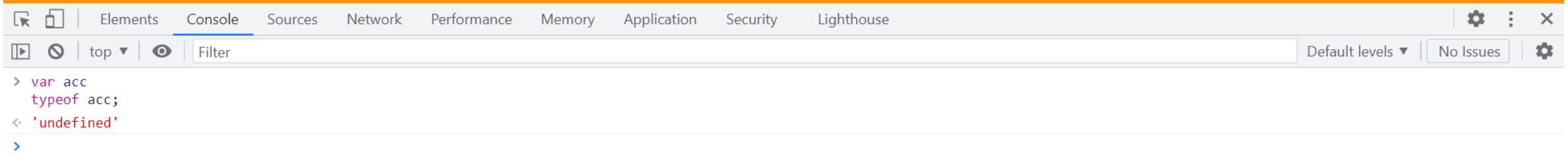
1-2. JavaScript 의 자료형 이해하기

undefined 유형과 null 유형

undefined는 **자료형이 정의되지 않았을 때의 데이터 상태**를 나타냅니다. 자바스크립트에서는 변수를 선언할 때 **자료형을 미리 지정하지 않고 값을 할당할 때 결정**합니다. 그래서 **변수 선언만 하고 값이 할당되지 않은 자료형**을 **undefined**라고 합니다. 즉, undefined은 단순히 '**변수에 값이 할당되지 않았다**'는 의미입니다.

반면에 null은 '데이터의 값이 유효하지 않은 상태'를 나타냅니다. undefined와 비슷해 보이지만 구별해야 합니다.

undefined



1-2. JavaScript 의 자료형 이해하기

배열

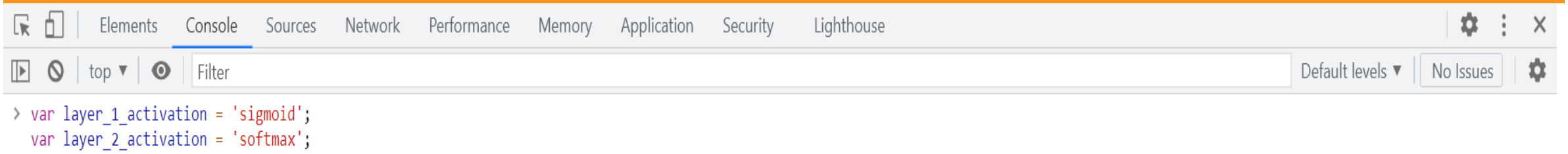
지금까지 살펴본 **자료형은 변수 하나에 값도 하나만 저장**할 수 있지만, **배열(array)은 하나의 변수에 값을 여러 개 저장**할 수 있습니다. 배열은 **여러 개의 데이터값을 하나의 배열 이름으로 묶어서 선언**합니다.

다음과 같이 **데이터 값을 쉼표로 구분해서 대괄호([ ])로 묶으면 배열을 선언**할 수 있는데, **대괄호 안에 값을 입력하지 않으면 빈 배열**이 만들어 집니다. 물론 빈 배열도 배열을 선언한 것입니다.

[기본형] 배열명["값1", "값2", ...]
배열명 [ ]

예를 들어 **인공지능 구현을 위해 정의해야 하는 각 레이어마다의 활성화 함수를 할당하는 프로그램**을 생각해보겠습니다. 만약 **배열을 사용하지 않는다면 매우 번거로운 과정**을 거쳐야 합니다.

배열을 사용하지 않는 경우



1-2. JavaScript 의 자료형 이해하기

만약 **배열을 활용**한다면 다음과 같이 **간단하게 1줄짜리 배열**로 작성할 수 있습니다. 그리고 activation 에 저장된 4개의 값이 나타나는데, **이때 데이터 옆에 있는 번호 0 ~ 3은 인덱스(index)**라고 합니다.

이처럼 **배열은 여러 개의 요소로 구성되고 각 요소는 자신만의 번호**를 갖게 됩니다.

만약 **원하는 요소가 있다면 그 요소의 인덱스 번호를 대괄호를 통해 호출하여 확인**할 수도 있습니다.

배열을 사용하는 경우

Elements Console Sources Network Performance Memory Application Security Lighthouse

top ▼ Filter Default levels ▼ No Issues

```
var activation = ['sigmoid', 'softmax', 'relu', 'elu'] ;
undefined
> activation
▼(4) ['sigmoid', 'softmax', 'relu', 'elu']
    0: "sigmoid"
    1: "softmax"
    2: "relu"
    3: "elu"
    length: 4
  ▶[[Prototype]]: Array(0)
> activation[0]
'sigmoid'
> activation[1]
'softmax'
> activation[2]
'relu'
>
```

원하는 요소를 꺼내기 위해 인덱스를 활용

1-3. JavaScript 의 연산자 알아보기

연산자란 **프로그램에서 특정한 동작을 하도록 지시하는 기호**입니다. 프로그래밍에서 '연산'이란 사칙연산은 물론 문자열과 문자열을 연결해서 새로운 문자열을 만들고 값의 크기를 비교하는 등 여러가지 동작을 의미합니다. 그리고 **이런 연산자를 지시하는 기호**가 **연산자**입니다.

산술 연산자

산술 연산자는 **우리가 잘 알고 있는 수학 계산을 할 때 사용하는 연산자** 입니다. 연산자의 왼쪽이나 오른쪽에 있는 **연산 대상이 '피연산자'** 라고 하는데, **산술 연산자에서 피연산자는 숫자나 변수**가 옵니다.

		연산자		연산자		연산자
예측값 Pred	=	독립변수 X	×	가중치 W	+	편향 B
피연산자		피연산자		피연산자		피연산자

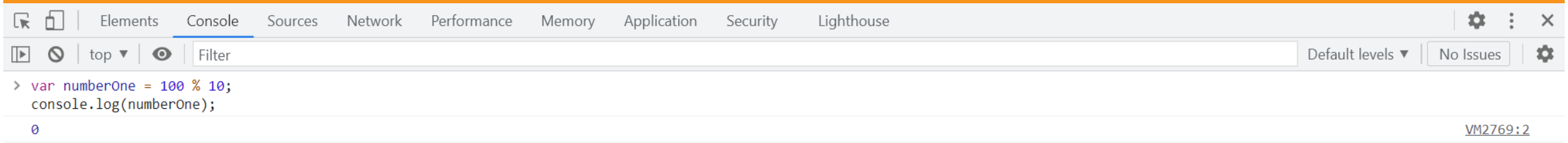
1-3. JavaScript 의 연산자 알아보기

산술 연산자		
종류	설명	예시
+	두 연산자의 값을 더합니다.	c = a + b
-	첫 번째 피연산자의 값에서 두 번째 피연산자의 값을 뺍니다.	c = a – b
*	두 연산자의 값을 곱합니다.	c = a * b
/	첫 번째 피연산자 값을 두 번째 피연산자 값으로 나눕니다.	c = a / b
%	첫 번째 피연산자 값을 두 번째 피연산자 값으로 나눈 나머지를 구합니다.	c = a % b
++	피연산자를 1 증가시킵니다.	a++
--	피연산자를 1 감소시킵니다.	b--

자주 헷갈리는 **나머지(%)**를 살펴보겠습니다. 나머지 연산자의 결괏값은 나눈 후에 **남은 나머지 값**이 됩니다.

예를 들어 다음 연산식에서 **numberOne은 100을 10으로 나눈 나머지인 0**이 됩니다.

나머지 연산자 활용 예제 1



1-3. JavaScript 의 연산자 알아보기

인공지능 프로그램을 만드는 과정에서도 **나머지는 요긴하게 사용**되어 집니다. 예를들면 학습을 하는 과정을 매번 출력하게 되면 오히려 더 많은 정보로 인해 혼란스러울 수도 있습니다. 그렇기에 **원하는 반복 주기만큼 학습 결과를 보여주기 위해 나머지(%)를 활용**할 수 있습니다.

나머지 연산자 활용 예제 2

about:blank

about:blank

현재 반복 주기는 10 이며, 총 반복 횟수는 100 입니다.

Elements Console Sources Network Performance Memory Application Security Lighthouse

top ▼ Filter Default levels ▼ No Issues 1 hidden

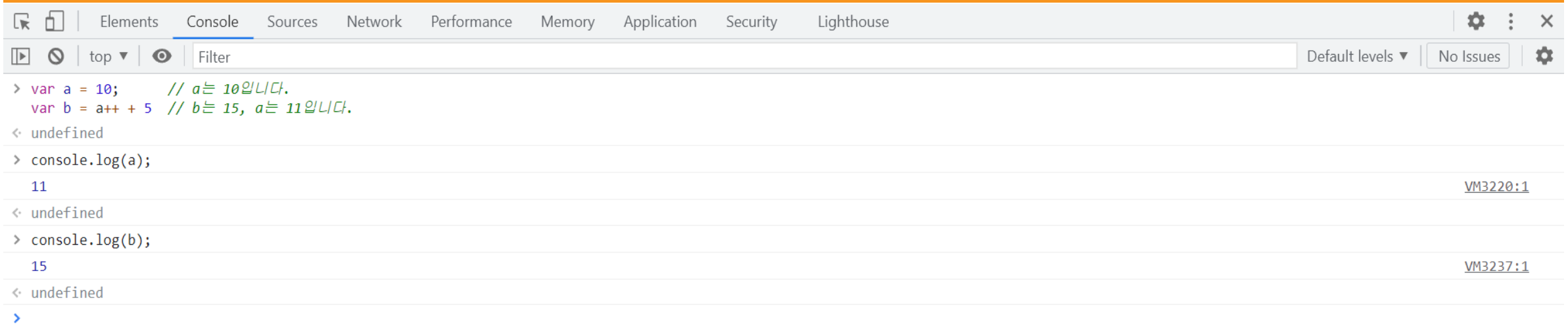
```
> var epochs = 100;
  var report = 10

  if (epochs % report == 0){
      document.write(`<h1>현재 반복 주기는 ${report} 이며, 총 반복 횟수는 ${epochs} 입니다.</h1>`);
  }
<· undefined
```

이후 배우게 될 조건문입니다.

1-3. JavaScript 의 연산자 알아보기

다음으로 살펴볼 산술 연산자는 **증가(++), 감소(--) 연산자** 입니다. 이 두 연산자는 **변수를 사용하기 전후에 변숫값을 1만큼 증가시키거나 감소**시키게 됩니다. 그리고 **증가, 감소 연산자를 변수 앞 또는 뒤에 붙였을 때 서로 다른 결괏값**을 보여줍니다.

변수 뒤에 증가 연산자 붙이기



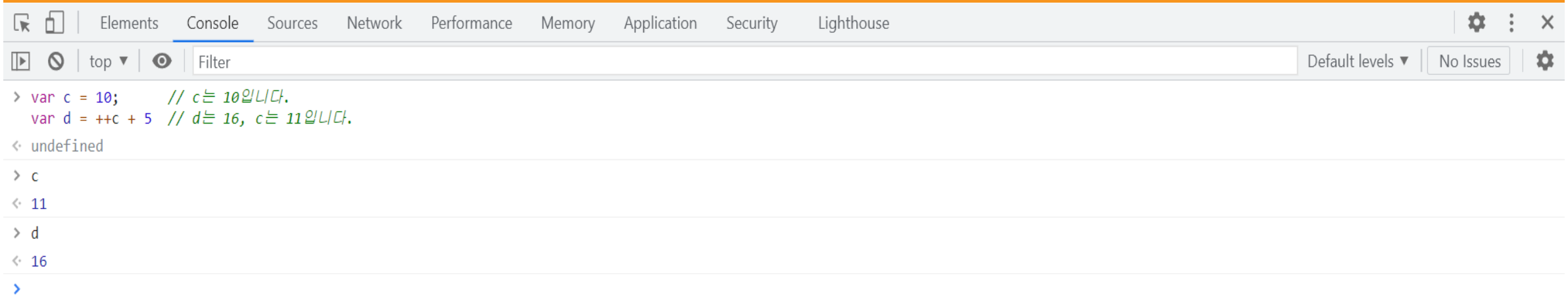
a++는 연산식을 먼저 실행한 후 **a에 1을 증가**시켜 줍니다. 그래서 값을 확인해 보면 **변수 b가 15가 되는 것**입니다.

그렇다면 증가 연산자(++)를 변수 앞에 붙이면 어떻게 될까요?

1-3. JavaScript 의 연산자 알아보기

++c는 변수 c에 1을 증가시킨 후 연산식을 실행하게 됩니다. 변수 d와 c의 값을 확인하면 변수 d는 16이 되고, c는 11이 됩니다.

변수 앞에 증가 연산자 붙이기



이렇듯 **증가, 감소 연산자가 피연산자 뒤(Ex a++)**에 있을 때는 **연산식의 처리를 끝낸 다음에 적용**됩니다. 반대로 **피연산자 앞에 있을 때(Ex ++a)는 연산식을 처리하기 전에 적용**됩니다. 즉, 앞에 두 예제에서 a++는 기존의 a값(10)과 5를 더하여 b에 할당한 다음에 비로소 a 값을 1만큼 증가시켰지만, ++c는 c값을 1만큼 증가시킨 다음에 5를 더하여 d에 할당합니다.

비슷해 보이는 연산식이지만 **증가, 감소 연산자를 피연산자 앞뒤 어디에 붙이느냐에 따라 큰 차이**가 있습니다.

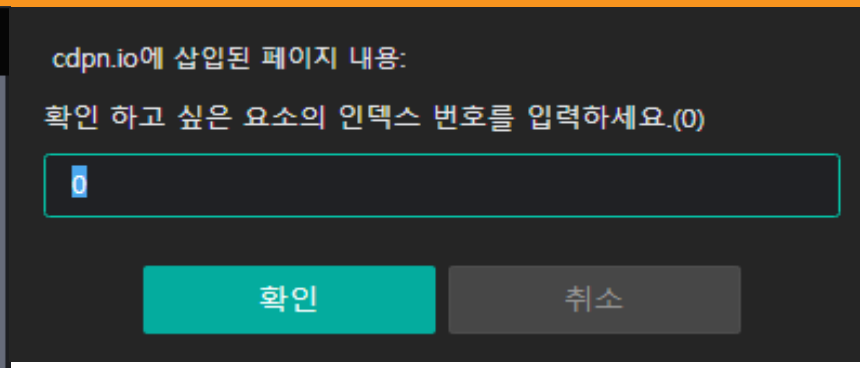
1-3. JavaScript 의 연산자 알아보기

증가 연산자(++)를 인공지능 프로그램에서 사용되는 예제는 다음과 같습니다.

JS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/dyVYmeM)

```js
var ys          = [0,1,1,0] // 실제 정답을 배열로 작성하였습니다.
var predictedYs = [0,1,1,0] // 인공지능의 예측 결괏값을 배열로 작성하였습니다.
var predicted   = 0         // 정답 여부를 확인하기 위한 변수 선언
// 사용자에게 입력값을 받는 과정입니다.
var index = prompt("확인 하고 싶은 요소의 인덱스 번호를 입력하세요.(0)","0")

// 이후에 배우게 될 조건문입니다.
if (ys[index] == predictedYs[index]) {
    // 위의 코드가 참인 경우, predicted 변수에 증가 연산자로 1 증가 시켜줍니다.
    predicted++;
    document.write(`실제 정답 : ${ys[index]} , 인공지능 예측 : ${predictedYs[index]}`);
    document.write(`<p>확인 결과 : ${predicted}(정답인 경우 1)</p>`);
}
```

결과 확인하기



실제 정답 : 1 , 인공지능 예측 : 1

확인 결과 : 1(정답인 경우 1)

조건문이나 반복문을 사용하는 경우 증가 연산자를 종종 활용하고는 합니다.

1-3. JavaScript 의 연산자 알아보기

할당 연산자

할당 연산자는 **연산자 오른쪽의 실행 결과를 왼쪽 변수에 할당**하는 연산자로 대입 연산자라고도 합니다.

변수에 값을 할당하거나 연산식의 결과를 변수에 저장할 때 할당 연산자를 사용합니다.

자바스크립트에서는 할당 연산자와 산술 연산자를 합쳐서 하나의 할당 연산자로 표시할 수 있습니다. 자바스크립트에서 사용하는 할당 연산자는 다음과 같습니다.

할당 연산자		
종류	설명	예시
=	연산자 오른쪽의 값을 왼쪽 변수에 할당합니다.	y = x + 3
+=	y = y + x를 의미합니다.	y += x
-=	y = y – x를 의미합니다.	y -= x
*=	y = y * x를 의미합니다.	y *= x
/=	y = y / x를 의미합니다.	y /= x
%=	y = y % x를 의미합니다.	y %= x

1-3. JavaScript 의 연산자 알아보기

예를들어 **이진 분류 문제**에 가장 많이 사용되는 **손실 함수**는 **로그 손실**(Log Loss)이라 부르기도 합니다. 로그 손실은 예측 입력이 0과 1 사이의 확률값인 **이진 분류 모델의 성능을 측정**합니다. 머신러닝 모델의 목표는 **손실을 최소화 하는것**이죠. 로그 손실이 0이 되어야 완벽한 인공지능 모델이라 할 수 있습니다. 반대로, 예를 들어 실제 정답이 1이지만 예측 확률이 0.012라면 손실값은 크게 측정될 것입니다.

로그 손실의 방정식은 다음과 같습니다.

$$-(y\log(p+eps)+(1-y)\log(1-p+eps))$$

여기서 **y는 실제 값(0또는 1)** 이고 **p는 예측 확률(0과 1사이의 값)**입니다. 그리고 **예측 결과에 eps(엡실론, epsilon)을 추가해 로그가 0을 취하지 않도록** 하였습니다.

그럼 다음과 같은 방정식을 **자바스크립트 코드를 통해 작성**하여 보겠습니다.

1-3. JavaScript 의 연산자 알아보기

JS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/abLvGEp)

JS

```javascript
var y    = [0,1,1,0];                        // 실제 정답을 입력하여 줍니다.
var pred = [0.0012, 0.955, 0.899, 0.945];    // 인공지능의 예측값을 입력하여 줍니다.
                                             // 배열의 마지막 수는 일부로 오답에 가깝게 예측하였습니다.

var eps  = 0.00001;  // log(0)을 방지하기 위해 아주 작은 수를 더해줍니다.

var index = prompt("손실 확인이 필요한 인덱스 번호를 입력하여 주세요.(0 ~ 3)", "0");

// 수식을 그대로 코드화 시켜주도록 합니다.
// 여기서 log()는 Math.log()를 활용하여 줍니다.
var log_loss = -(y[index] * Math.log(pred[index] + eps) + (1 - y[index]) * Math.log(1 - pred[index] + eps));

document.write(`실제 정답 : ${y[index]}`);
document.write(`<p>예측 결과 : ${pred[index]}</p>`);
document.write(`<p>손실 : ${log_loss}</p>`);
```

결과 확인하기

cdpn.io에 삽입된 페이지 내용:

손실 확인이 필요한 인덱스 번호를 입력하여 주세요.(0 ~ 3)

0

확인 취소

정답에 근접한 경우 / 정답에서 멀어진 경우

실제 정답 : 1	실제 정답 : 0
예측 결과 : 0.955	예측 결과 : 0.945
손실 : 0.046033467352041085	손실 : 2.9002402920947694

손실값의 차이가 확연하게 차이가 납니다.

1-3. JavaScript 의 연산자 알아보기

비교 연산자

비교 연산자는 피연산자 2개의 값을 비교해서 **참이나 거짓으로 결괏값을 반환**합니다. 이 연산자는 주로 **두 값을 비교**하므로 **어떠한 조건을 체크할 때 사용**합니다. 자바스크립트에서 사용하는 비교 연산자는 다음과 같습니다.

비교 연산자			
종류	설명	조건식	결괏값
==	피연산자가 서로 같으면 true 입니다.	3 == “3”	True
===	피연산자도 같고 자료형도 같으면 true 입니다.	A === “3”	False
!=	피연산자가 서로 같지 않으면 true 입니다.	3 != “3”	False
!==	피연산자가 같지 않거나 자료형이 같지 않으면 true입니다.	3 !== “3”	True
<	왼쪽 피연산자가 오른쪽 피연산자보다 작으면 true 입니다.	3 < 4	True
<=	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같으면 true 입니다.	3 <= 4	True
>	왼쪽 피연산자가 오른쪽 피연산자보다 크면 true 입니다.	3 > 4	False
>=	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같으면 true 입니다.	3 >= 4	False

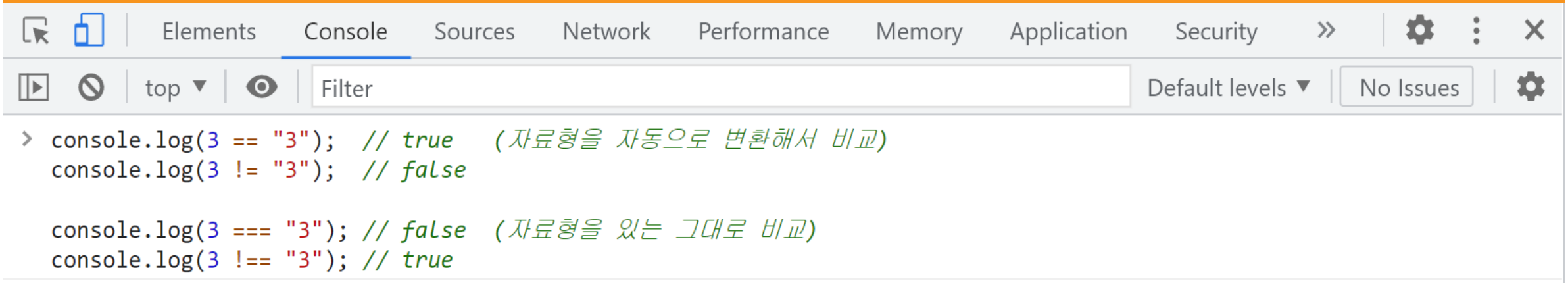
1-3. JavaScript 의 연산자 알아보기

==, != 연산자와 ===, !== 연산자 비교

위 표에서 피연산자가 서로 같은지 다른지 비교하는 연산자 두 그룹이 있었습니다. ==와 != 연산자 그리고 === 와 !=== 연산자는 서로 비슷하면서도 다르다는 것을 확인하셨을 것입니다. 지금부터 그 차이를 알아보겠습니다.

== 연산자와 != 연산자는 피연산자의 자료형을 자동으로 변환해서 비교합니다. 숫자 3과 문자열 “3”을 비교하면 왼쪽의 숫자 3을 문자열로 비교해서 비교합니다.

JS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/abLvGEp)



반면에 **=== 연산자와 !== 연산자는 피연산자의 자료형을 변환하지 않습니다.** 그래서 프로그램에서 값을 비교할 때는 **자료형을 자동으로 변환하지 않기 위해 === 연산자와 !== 연산자를 더 많이 사용**하고는 합니다.

1-3. JavaScript 의 연산자 알아보기

비교 연산자는 앞으로 배우게 될 반복문에서도 자주 사용되고는 합니다. 비교 연산자를 통해 인공지능 프로그램의 일부를 만들어 보도록 하겠습니다.

JS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/xxXwjpq?editors=0010)

JS

```js
var ys          = [0,1,1,0]; // 실제 정답을 배열 형태로 정의.
var predictedYs = [0,1,1,1]; // 인공지능 예측값을 배열 형태로 정의.
var predicted_count = 0      // 정답을 카운트하는 변수입니다.

// 이후에 배우게 될 반복문과 조건문 구조 입니다.
// 정답을 얼마나 맞췄는지 카운트를 하는 반복문 및 조건문 구조 입니다.
// 1. i의 값을 0으로 초기화 합니다.
// 2. i < ys.length 조건이 거짓이 될 때 까지 반복하여 줍니다.(y의 개수만큼 반복)
// .length는 ys가 가지고 있는 값의 수를 세어줍니다.
// 3. 반복이 진행되면 i의 값을 1씩 증가시켜줍니다.
for (var i = 0; i < ys.length; i++){
    // 조건문 구조입니다.
    // 1. 각각의 ys,predictedYs의 요소를 비교하여 줍니다.
    // 2. 위 조건이 참인 경우 아래의 코드를 수행하여 줍니다.
    if (ys[i] == predictedYs[i]){
        // 3. predicted_count의 값을 1 증가시켜 줍니다.
        predicted_count ++
    }
}

// html과 css를 한꺼번에 정의하여 줄 수도 있습니다.
document.write(`<h1 style="text-align: center;">문제는 총 ${ys.length}개 입니다.</h1>`)
document.write(`<hr>`)
// 정확도를 산출하는 과정을 자바스크립트 연산자 기능과 lenght 속성을 사용하였습니다.
document.write(`<h2 style="text-align: center;">정답을 맞춘 횟수는 총 ${predicted_count}회 이므로 정확도는
${(predicted_count/ys.length)*100} % 입니다.</h2>`)
```

결괏값

문제는 총 4개 입니다.

정답을 맞춘 횟수는 총 3회 이므로 정확도는 75 % 입니다.

1-3. JavaScript 의 연산자 알아보기

논리 연산자

논리 연산자는 **불리언 연산자**라고도 하며 **true, false를 처리하는 연산자** 입니다. 즉, **true와 false 자체가 피연산자인 연산자**입니다. 논리 연산자는 주로 프로그램에서 **조건을 체크할 때 사용**합니다. 자바스크립트에서 사용하는 논리 연산자는 아래와 같습니다.

논리 연산자		
종류	기호	조건식
OR 연산자	||	피연산자 중 **하나만 true여도 true**가 됩니다.
AND 연산자	&&	피연산자가 **모두 true일 경우에만 true**가 됩니다.
NOT 연산자	!	피연산자의 **반댓값을 지정**합니다.

1-3. JavaScript 의 연산자 알아보기

논리 연산자는 인공지능 프로그램을 만드는 과정에서도 유용하게 사용됩니다. 예를 들어 **XOR 게이트의 논리식**을 생성하기 위해 **자바스크립트 코드**로 구현해 보겠습니다.

우선 문제에서 등장한 XOR 게이트의 논리식은 다음과 같습니다.
하지만 문제의 난이도를 높이기 위해 **X1과 X2의 값 범위를 -1 ~ 1 사이의 값으로 조정**하겠습니다.
여기서 **양수인 경우를 1(+), 음수인 경우를 0(-)으로 대체**하여 주겠습니다.

조정된 XOR 게이트 논리식		
입력 1	입력 2	출력
X1 (-1 ~ 1)	X2 (-1 ~ 1)	Y
0 (-)	0 (-)	0(-)
1 (+)	0 (-)	1(+)
0 (-)	1 (+)	1(+)
1 (+)	1 (+)	0(-)

1-3. JavaScript 의 연산자 알아보기

JS,CSS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/KKXdrjb)

JS

```javascript
// 독립변수 x와 종속변수 y를 생성하기 위해 빈 변수를 선언합니다.
var x, y ;
var iter = prompt("XOR Gate를 몇 회 반복 실험하시겠습니까?(최대 10회)", "10");

document.write(`<h1 style = "text-align : center"> XOR Gate </h1><br><hr>`)

// x의 값은 [x1, x2] 이렇게 배열의 형태로 생성.
// x값의 범위는 -1 ~ 1 사이를 갖도록 합니다.

// 1. x에 할당된 두 값의 부호를 먼저 비교합니다.
// 2. 두 값이 모두 같은 양의 부호이거나 혹은(or) 두 값이 모두 같은 음의 부호라면
// y는 0의 값을 갖도록 합니다.
// 위 조건이 아니라면(else) 즉, 한 쪽이라도 다른 부호라면
// 3. y는 1의 값을 갖도록 합니다.
for (var i = 0; i < iter; i++){
  x = [Math.random() * 2 - 1, Math.random() * 2 - 1]
  if ((x[0] > 0 && x[1] > 0) || (x[0] < 0 && x[1] < 0)){
  y = 0 } else { y = 1
}
// 결과를 화면으로 출력하는 과정입니다.
  document.write(`<h3>첫 번째 X의 값 : ${x[0]}</h3>`);
  document.write(`<h3>두 번째 X의 값 : ${x[1]}</h3>`);
  document.write(`<h3>Y 값 : ${y}</h3>`);
  document.write(`<hr id = "result_line">`)
}
```

CSS

```css
#result_line {
  border : none;
  border-top: 3px dotted gray;
}
```

결괏값

cdpn.io에 삽입된 페이지 내용:

XOR Gate를 몇 회 반복 실험하시겠습니까?(최대 10회)

10

확인 취소

XOR Gate

첫 번째 X의 값 : 0.17658017218543076

두 번째 X의 값 : 0.06017172800753556

Y 값 : 0

첫 번째 X의 값 : 0.9564420983847652

두 번째 X의 값 : -0.34461541638077176

Y 값 : 1

첫 번째 X의 값 : 0.04680971470448014

두 번째 X의 값 : -0.10840209945328105

Y 값 : 1

1-4. JavaScript 의 조건문 알아보기

프로그래밍을 할 때는 **특정 조건과 명령에 따라 실행 순서**를 정해야 합니다. 이때 **특정 조건이 맞는지 확인**하기 위해 **조건문을 사용**하는데, **프로그래밍에서 가장 많이 사용하는 구문**입니다. 여기에서는 자바스크립트의 조건문과 조건 연산자를 살펴보겠습니다.

if문과 if ~ else 문 알아보기

if문이나 if ~ else 문을 사용하면 **스크립트 안에서 조건을 체크**할 수 있습니다. **if 문**에서는 괄호 안의 조건을 체크해서 **결괏값이 true면 if 문 다음에 나오는 명령을 실행**하고, **false 이면 아무것도 하지 않습니다.** 조건을 만족했을 때 실행할 명령이 여러 개라면 중괄호({ })로 묶고 그 안에 명령을 나열하여 순서대로 한꺼번에 처리하라고 표시합니다.

```
[기본형] if (조건) {
            조건 결괏값이 true일 때 실행할 명령
         }
```

if 문은 결괏값이 true일 때만 실행하므로 **true가 아닐 때 명령을 수행하려면 한계**가 있습니다. 그래서 등장하는 것이 else 문 입니다.

1-4. JavaScript 의 조건문 알아보기

if ~ else 문은 if 조건의 결괏값이 true가 아닐 때 실행할 명령을 else 문 다음에 추가 합니다.

```
[기본형] if (조건) {
           조건 결괏값이 true일 때 실행할 명령
         } else {
           조건 결괏값이 false일 때 실행할 명령
         }
```

다음은 **사용자가 입력한 숫자가 3의 배수인지 확인하는 예제**입니다. 3의 배수는 어떻게 확인할 수 있을까요? 입력받은 변수 userNumber를 3으로 나눈 후 그 나머지가 0이면 3의 배수이고 그렇지 않으면 3의 배수가 아닙니다.

3의 배수 확인하기 – 첫 번째 예제

Elements Console Sources Network Performance Memory

top ▾ Filter

```
> var userNumber = prompt("3의 배수 확인을 위한 값을 입력하세요.", "3");
  // 조건문 구조를 사용합니다.
  if(userNumber % 3 === 0) {
    // 조건이 참인 경우 동작하는 코드
    alert("3의 배수입니다.")} else {
    // 조건이 거짓인 경우 동작하는 코드
    alert("3의 배수가 아닙니다.")}
```

결괏값

이 페이지 내용:

3의 배수입니다.

확인

이 페이지 내용:

3의 배수가 아닙니다.

확인

1-4. JavaScript 의 조건문 알아보기

하지만 프로그래밍을 할 때 **하나의 if ~ else 문으로는 해결되지 않는 상황이 더 많습니다.** 이럴 때는 **다른 if ~ else 문 안에 또 다시 if ~ else문을 사용**할 수 있습니다. 이렇게 다른 **if ~ else 문 안에 포함되는 if ~ else 문**을 **중첩된 if ~ else 문**이라고 합니다.

사실 앞에서 앞에서 살펴본 '3의 배수 확인하기 – 첫 번째' 예제는 **사용자가 프롬프트 창에서 값을 입력하지 않고 [취소] 버튼을 눌렀을 때의 동작도 고려**해야 합니다. 사용자가 [취소] 버튼을 눌렀는지 확인하고, 숫자를 입력한 후 [확인] 버튼을 눌렀을 때만 3의 배수를 체크해야 합니다. 이제 앞에서 살펴본 예제를 수정해 보겠습니다.

프롬프트 창에서 **[취소] 버튼을 누르면 변수에는 null이 저장**됩니다. 따라서 userNumber의 변숫값이 null인지 아닌지를 확인하면 됩니다. 이때도 if ~ else 문을 사용할 수 있습니다.

userNumber 값이 null이 아니면 3의 배수인지 확인하고, null 이면 숫자가 입력되지 않았다고 알려줍니다. 이 과정을 **순서도로 작성**하면 다음과 같습니다.

1-4. JavaScript 의 조건문 알아보기

3의 배수 확인하기 – 두 번째 예제(https://codepen.io/AI_R_IM/pen/abLvPrw)

JS

```js
var userNumber = prompt("3의 배수를 확인할 숫자를 입력하세요.", "21");

// 첫 번째 조건은 변수 userNumber가 null이 아닌지를 확인합니다.
if (userNumber !== null){
  // null이 아니라면 즉 조건이 참이라면 다음 조건을 수행합니다.
  // 이번 조건은 변수 userNumber가 3의 배수인지를 확인하는 조건입니다.
  // 만약 이 조건이 거짓이라면 하단의 ★★코드로 넘어갑니다.
  if(userNumber % 3 === 0){
    // 3의 배수가 맞다면 아래 코드를 수행합니다.
    alert("3의 배수 입니다.");
    // 3의 배수가 아니라면 아래코드를 수행합니다.
  } else {
    alert("3의 배수가 아닙니다.");
  }
// ★★ 첫 번째 조건이 거짓인 경우, 즉 변수 userNumber가 null 인 경우라면
}else{
  // 아래 코드를 수행하여 줍니다.
  alert("입력이 취소됐습니다.");
}
```

결괏값

cdpn.io에 삽입된 페이지 내용:

3의 배수를 확인할 숫자를 입력하세요.

21

확인 취소

cdpn.io에 삽입된 페이지 내용:

3의 배수 입니다.

확인

cdpn.io에 삽입된 페이지 내용:

입력이 취소됐습니다.

확인

1-4. JavaScript 의 조건문 알아보기

조건 연산자로 조건 체크하기

만약 **조건이 하나**이고 **true일 때와 false일 때 실행할 명령이 각각 하나뿐**이라면 if ~ else 문 대신에 **조건 연산자**를 사용하는 것이 간단합니다.

[기본형] (조건) ? true일 때 실행할 명령 : false 일 때 실행할 명령

예를 들어 앞에서 살펴본 '3의 배수 확인하기 – 두 번째 예제'에서 중첩된 **if ~ else 문 대신에 조건 연산자를 사용**한다면 다음과 같이 작성할 수 있습니다.

조건 연산자를 사용하여 3의 배수 확인하기 예제

Elements Console Sources Network Performance Memory Application Security

top Filter Default levels No Issues

```
> var userNumber = prompt("숫자를 입력하세요");

  // 조건 연산자를 활용하는 예제입니다.
  // if문을 활용하여 첫 번째 조건을 입력합니다.
  if (userNumber !== null)
    // 첫 번째 조건이 참인 경우 동작하는 코드입니다.
    // (조건) ? (조건이 참인 경우 동작하는 코드) : (조건이 거짓인 경우 동작하는 코드)
    (userNumber % 3 === 0) ? alert("3의 배수입니다.") : alert("3의 배수가 아닙니다.");
    // 첫 번째 조건이 거짓인 경우 동작하는 코드입니다.
  else
    alert("입력이 취소됐습니다.");
```

1-4. JavaScript 의 조건문 알아보기

switch 문

자바스크립트에서 조건을 체크할 때는 if ~ else 문을 사용하지만, **처리할 명령이 많다면 if ~ else 문을 여러 개 사용하는 것 보다 switch 문이 더 편리**합니다.

switch 문에서 조건을 체크한 후 **case 문을 사용하여 명령을 처리**할 수 있습니다.

switch 문의 기본 형식은 다음과 같습니다.

기본형

```
switch(조건){
    case 값1 : 명령1
        break
    case 값2 : 명령2
        break
    ...
    default : 명령n
}
```

switch 문의 조건은 **아래에 있는 case 문의 값과 일대일로 일치**해야 합니다. 조건과 일치하는 case 문의 명령을 실행한 후에는 switch 문을 완전히 빠져나옵니다.

switch의 조건이 case의 '값1'과 일치하면 '명령1'을 실행하고, 다음에 있는 **break문을 만나 switch 문을 빠져나갑니다.** 이런 식으로 조건에 따라 다른 명령을 처리할 수 있습니다.

하지만 조건과 일치하는 case 문이 없다면 어떻게 해야 할까요? 이럴 때는 **마지막에 있는 default 문을 실행**해야 합니다. default 문은 switch 문의 마지막에 작성하며 여기에는 break 문을 쓰지 않습니다.

1-4. JavaScript 의 조건문 알아보기

다음은 사용자에게 하나의 값을 받아 변수에 저장하고, switch문을 이용해 변숫값을 체크하는 예제 입니다.

JS,CSS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/MWEaLWj?editors=0010)

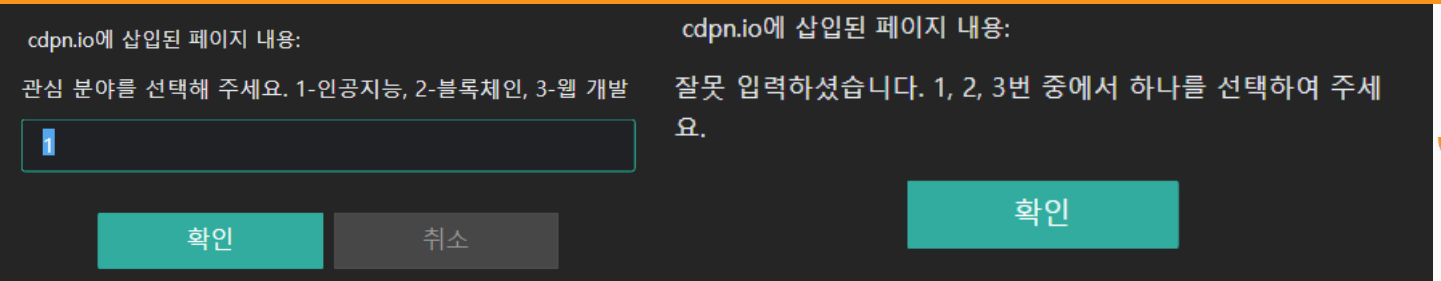
JS

```js
var session = prompt("관심 분야를 선택해 주세요. 1-인공지능, 2-블록체인, 3-웹 개발","1");

switch(session) { //switch(조건)
  case "1" :      //case 값1 : 명령1
    document.write("<p>인공지능은<br><strong>광주 인공지능 사관학교</strong><br>에서 진행됩니다.</p>")
    break;        //break
  case "2" :      // ...
    document.write("<p>블록체인은<br><strong>K-Digital Training</strong><br>에서 진행됩니다.</p>");
    break;
  case "3" :
    document.write("<p>웹 개발은<br><strong>ICT 이노베이션 스퀘어</strong><br>에서 진행됩니다.</p>")
    break;
  //default: 명령n
  default: alert("잘못 입력하셨습니다. 1, 2, 3번 중에서 하나를 선택하여 주세요.");
}
```

CSS

```css
body {
  background-color: rgb(250, 250, 250);
  color: rgb(0, 0, 0);
}

p {
  margin-top: 80px;
  font-size: 2em;
  font-weight: 700;
  text-align: center;
  text-shadow: 1px 2px 1px rgb(109, 109,109);
}

p strong {
  font-size: 2.2em;
  color: rgb(247, 148, 30);
}
```

결괏값



1-4. JavaScript 의 조건문 알아보기

자리 배치도 만들기 – 첫 번째 예제

지금까지 살펴본 내용들을 바탕으로 자리 배치도 프로그램을 만들어 보겠습니다.

1단계. 프로그램의 흐름 생각하기

프로그램을 작성하기 전에 **필요한 변수와 변수 이름, 연산식을 어떻게 할지 작성**하여 보세요.

- 입력할 값의 변수 : 입장객 수(memNum), 한 줄에 앉을 사람의 수(colNum)
- 계산할 값의 변수 : 필요한 줄의 개수(rowNum)

먼저 입장객 수와 한 줄에 앉을 사람 수가 주어졌을 때 **줄의 개수를 어떻게 계산할 지 생각**하여 봅시다.

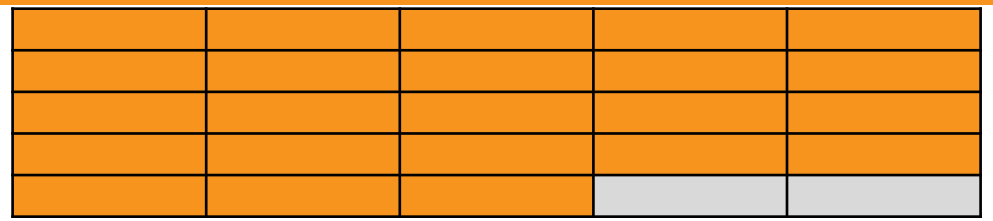
예를 들어 전체 **입장객 수가 20명**이고 **1줄에 5명씩** 앉아야 한다면 '20/5' 를 계산하여 **총 4줄이 필요**합니다.

그럼 **입장객 수가 23명**이고 **1줄에 5명씩** 앉아야 한다면 총 몇 줄이 필요할까요? **'23/5'**를 계산하면 **몫은 4**지만

나머지 값(3)이 있으므로 **필요한 줄은 총 5줄**이 됩니다.

자리 배치도 프로그램의 흐름 생각하기

입장객 수 : 23명
한 줄에 앉을 수 있는 인원 : 5명



입장객 | 빈 좌석

즉, 입장객 수를 한 줄에 앉을 사람 수로 나눴을 때 **나머지가 없는 연산식과 나머지가 있는 연산식**을 다음과 같이 생각해 볼 수 있습니다.

- 나머지가 없을 때 : rowNum = memNum / colNum
- 나머지가 있을 때 : rowNum = (memNum / colNum) + 1

1-4. JavaScript 의 조건문 알아보기

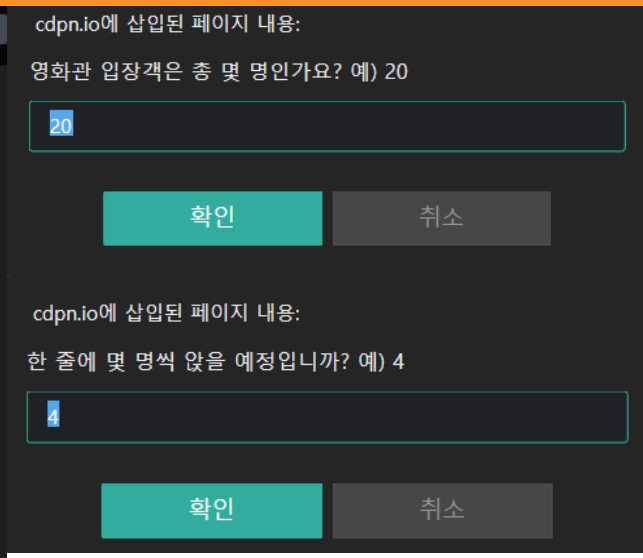
2단계. 자바스크립트 코드 작성하고 결과 확인하기

JS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/ZEXbwEB?editors=0010)

JS

```js
var memNum = prompt("영화관 입장객은 총 몇 명인가요? 예) 20", "20");
var colNum = prompt("한 줄에 몇 명씩 앉을 예정입니까? 예) 4", "4");

document.write(`<h1 style = "text-align : center;">GCV 영화관 자리배치도 프로그램</h1>`);
// 1단계에서 생각해 보았던 프로그램의 흐름을 떠올려 봅니다.
// 입장객 인원과 한 줄에 앉을 인원수를 나눴을 때 결괏값이 0인지를 확인합니다.
if(memNum % colNum === 0) {
  // 위 조건이 참이라면 연산을 진행하고 값을 변수 rowNum에 저장합니다.
  // ※parseInt()는 실수값을 정수로 반환해주는 함수입니다.
  rowNum = parseInt(memNum / colNum);
// 위 조건이 거짓이라면 딱 나눠떨어지지 않는다는 의미입니다.
} else {
  // 그래서 결괏값에 1을 더해주는 수식을 추가하여 줍니다.
  rowNum = parseInt(memNum / colNum) + 1;
}
document.write(`<p style = "text-align : center;"><안내> 모두 ${rowNum}개의 줄이 필요합니다.</p>`);
```

결괏값



GCV 영화관 자리배치도 프로그램

<안내> 모두 7개의 줄이 필요합니다.

※ prompt() 문을 사용해 입력받은 값은 **기본적으로 문자열로 저장**됩니다. **문자열 값**이라도 **산술 연산에 사용할 경우 자동으로 숫자형으로 변환**됩니다. 하지만 **이렇게 데이터의 유형이 자동으로 변환**되기 때문에 **예상치 못한 곳에서 다르게 변환**될 수 있습니다. 그래서 parselnt() **함수를 사용해 입력받은 값을 일부로 정수로 반환**하여 줍니다.

1-5. JavaScript 의 반복문 살펴보기

반복문은 **어떤 동작을 여러 번 실행할 때** 사용합니다. **불필요한 명령을 여러 번 복잡하게 쓰지 않고 반복문 하나로 간단하게 코드를 작성**할 수 있습니다.

for 문 사용하기

for 문은 **자바스크립트에서 가장 많이 사용하는 반복문** 입니다. for 문은 **값이 일정하게 커지면서 명령을 반복하여 실행**할 때 사용합니다. for문의 기본 형식은 다음과 같습니다.

기본형

```
for(초깃값; 조건; 증가식) {
    실행할 명령
}
```

1) **초깃값** : 카운터 변수를 초기화 합니다. **초깃값은 0이나 1부터 시작**합니다.

2) **조건** : 명령을 반복하기 위해 **조건을 체크**합니다. 이 **조건을 만족해야 그 다음에 오는 명령을 실행**할 수 있습니다.

3) **증가식** : **명령을 반복한 후 실행**합니다. 보통 카운터 변수를 1 증가시키는 용도로 사용합니다.

그렇다면 반복문을 활용한 간단한 예제를 살펴보도록 하겠습니다.

1-5. JavaScript 의 반복문 살펴보기

for문과 조건문 if를 활용한 예제	결괏값

Elements Console Sources Network Performance

top ▼ Filter

```
> var epochs = 100;
  var accs   = 0;
  for(var i = 0; i < epochs; i++){
      document.write(`Now Accuracy Score: ${accs += 1}<br>`);
      if(accs === 100){
          document.write(`======================`)
          document.write(`<p>Training End</p>`);}
  }
```

```
Now Accuracy Score: 1
Now Accuracy Score: 2
Now Accuracy Score: 3
~
Now Accuracy Score: 98
Now Accuracy Score: 99
Now Accuracy Score: 100
======================

Training End
```

본 예제는 학습 횟수(epoch)가 늘어날 수록 **인공지능의 정확도가 1씩 증가한다는 가정**을 갖는 프로그램입니다. for()안에는 세 가지 요소가 눈에 들어 오는데, **초깃값은 0부터 시작**하며, **조건은 i의 값이 epochs 보다 작을 때까지** 입니다. 그리고 증가식은 **i의 값이 증가 연산자(++)를 통해 1씩 증가**하고 있습니다.

그리고 **조건문을 통해 정확도가 100을 달성했는지 지속적으로 확인**을 합니다. **만약 100을 달성하였다면** 학습을 더 이상 할 필요가 없기에 **'Training End' 메시지를 출력**하여 주도록 합니다.

1-5. JavaScript 의 반복문 살펴보기

중첩된 for문을 사용하기

for 문 안에 다른 for문을 넣어 사용할 수도 있는데 이것을 **중첩 for문** 이라고 합니다. for문은 중첩해서 사용하는 경우가 많습니다. 예를들어 인공지능 프로그램에서도 **epoch과 mini batch 개념**을 프로그래밍 할 때 **중첩 for문을 사용**하곤 하죠. 이번에는 좀 더 간단한 예제인 **구구단을 출력하는 과정**을 함께 살펴보도록 하겠습니다.

구구단의 구성

우선 구구단을 떠올려 볼 필요가 있습니다. 구구단은 다음과 같이 정렬되어 있으며, **기준이 되는 왼쪽의 값**과 **반복적으로 1부터 9까지 곱해주는 오른쪽의 값이 존재**합니다. 그리고 이를 1단부터 9단까지 차례대로 **연산을 반복**하여 주고 있습니다.

다시말해 **각각의 단을 개별적으로 연산하는 반복문**(1 × 1, 1 × 2, 1 × 3, ...)과 1단 부터 9단까지 반복적으로 연산을 수행(**1단, 2단, 3단, ...**)해야 하는 반복문이 필요하단 것을 알 수 있습니다.

그럼 이를 자바스크립트 코드로 구현해보도록 하겠습니다.

외부 반복문	외부 반복문	외부 반복문
1단	2단	3단
내부 반복문	내부 반복문	내부 반복문
1 × 1 = 1	2 × 1 = 2	3 × 1 = 3
1 × 2 = 2	2 × 2 = 4	3 × 2 = 6
1 × 3 = 3	2 × 3 = 6	3 × 3 = 9
...	...	...

1-5. JavaScript 의 반복문 살펴보기

JS 파일에 코드 입력하기
(https://codepen.io/AI_R_IM/pen/bGoVJdj?editors=0010)

JS

```javascript
var i, j;  // 1 × 1 = 1, 1 × 2 = 1, 1 × 3 = 1 에서 가장 왼쪽의 값 i
           // 1 × 1 = 1, 1 × 2 = 1, 1 × 3 = 1 에서 1씩 증가하는 값 j

// 첫 번째 반복문은 외부 반복문 구조 입니다.
// 단을 처리하는 과정이죠.
for (i = 1; i <= 9; i++){
  document.write(`<h2>${i}단</h2>`);
  // 안쪽의 반복문 구조 입니다.
  // 여기서는 a × b = c 의 연산식을 코드화 하여 줍니다.
  for (j = 1; j <= 9; j++){
    document.write(`${i} × ${j} = ${i*j} <br>`)
    // 즉 1단의 순서에서는 1단의 연산과정을 반복하고,
    // 2단의 순서에서는 2단의 연산과정을 반복합니다.
    // 이러한 과정을 9단까지 이어서 진행하는 것이죠.
  }
}
```

결괏값

1단	2단
1 × 1 = 1	2 × 1 = 2
1 × 2 = 2	2 × 2 = 4
1 × 3 = 3	2 × 3 = 6
1 × 4 = 4	2 × 4 = 8
1 × 5 = 5	2 × 5 = 10
1 × 6 = 6	2 × 6 = 12
1 × 7 = 7	2 × 7 = 14
1 × 8 = 8	2 × 8 = 16
1 × 9 = 9	2 × 9 = 18

그런데 구구단 시트를 만들고 보니 **한눈에 보기 어렵습니다**. 이러한 상황일 때는 **CSS를 활용하여 조금 더 직관적으로 HMTL 문서를 꾸며**줄 수 있습니다.

1-5. JavaScript 의 반복문 살펴보기

각 단을 태그로 묶고 가로로 나열하여 구구단을 보기 좋게 만들어 보도록 하겠습니다.

JS, CSS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/dyVYLad?editors=0110)

JS

```javascript
var i, j;  // 1 × 1 = 1, 1 × 2 = 1, 1 × 3 = 1 에서 가장 왼쪽의 값 i
           // 1 × 1 = 1, 1 × 2 = 1, 1 × 3 = 1 에서 1씩 증가하는 값 j

// 첫 번째 반복문은 외부 반복문 구조 입니다.
// 단을 처리하는 과정이죠.
for (i = 1; i <= 9; i++){
  document.write("<div>"); // <div> 태그를 추가하여 줍니다.
  document.write(`<h2>${i}단</h2>`);
  // 안쪽의 반복문 구조 입니다.
  // 여기서는 a × b = c 의 연산식을 코드화 하여 줍니다.
  for (j = 1; j <= 9; j++){
    document.write(`${i} × ${j} = ${i*j} <br>`)
    // 즉 1단의 순서에서는 1단의 연산과정을 반복하고,
    // 2단의 순서에서는 2단의 연산과정을 반복합니다.
    // 이러한 과정을 9단까지 이어서 진행하는 것이죠.
  }
  document.write("</div>"); // <div> 태그를 닫아주도록 합니다.
}
```

CSS

```css
div {
  /* <div>로 각 단마다 묶어준 것에 몇 가지 요소를 추가하여 줍니다. */
  display: inline-block;       /* 컨텐츠 영역만큼을 활용하도록 합니다. */
  padding: 0 20px 30px 20px;   /* 영역의 범위는 적절하게 설정합니다. */
  margin: 15px;
  border: 1px solid #ccc;      /* 영역의 경계선은 일반 선으로 정의하여 줍니다. */
  line-height: 2;
}
div h2 {
  /* div 태그 안에 h2 태그에 대한 스타일을 정의합니다.    */
  text-align: center;
  font-weight: bold;
}
```

결괏값

1단	2단	3단	4단	5단	6단
1 × 1 = 1	2 × 1 = 2	3 × 1 = 3	4 × 1 = 4	5 × 1 = 5	6 × 1 = 6
1 × 2 = 2	2 × 2 = 4	3 × 2 = 6	4 × 2 = 8	5 × 2 = 10	6 × 2 = 12
1 × 3 = 3	2 × 3 = 6	3 × 3 = 9	4 × 3 = 12	5 × 3 = 15	6 × 3 = 18
1 × 9 = 9	2 × 9 = 18	3 × 9 = 27	4 × 9 = 36	5 × 9 = 45	6 × 9 = 54

1-5. JavaScript 의 반복문 살펴보기

while 문과 do ~ while 문 사용하기

while 문은 **조건이 true인 동안 명령을 반복**합니다. for문과 마찬가지로 while 문도 **조건을 체크하고 true 일 때만 명령을 반복하여 실행**합니다.
결국 조건이 false라면 명령은 한 번도 실행하지 않을 수 있습니다.

기본형
```
while(조건){
    실행할 명령
}
```

while 문과 달리 **do ~ while 문은 조건이 맨 뒤에** 붙습니다.
그리고 **do 문은 일단 명령을 한번 실행한 후 while 문에서 조건을 체크**합니다.
그러므로 **조건이 false 라 하더라도 일단 명령은 최소한 한 번 실행**합니다.

기본형
```
do {
    실행할 명령
} while(조건)
```

지금까지 살펴본 for문, while문, do ~ while 문은 특정 명령을 여러 번 반복해서 실행할 수 있다는 공통점이 있습니다.
그렇다면 **어떤 상황일 때 어떤 반복문을 골라서 사용**해야 할까요?
당연하게도 프로그래밍을 진행하시면서 **프로그램에 맞는 반복문을 사용하는 것**입니다. 그렇다면 지금 내가 필요한 반복은 어떠한 반복의 구조를 가지는지, 즉 **프로그램에 대한 좀 더 자세한 이해가 뒷 받침**되어야 적절한 반복문을 선택하여 활용할 수 있습니다.

1-5. JavaScript 의 반복문 살펴보기

break 문과 continue 문 사용하기

반복문은 **지정한 횟수만큼 명령을 반복**할 때 사용합니다. 하지만 **특정 조건에서 반복문을 멈춰야 하거나 반복문 중간에 앞으로 되돌아가야 할 경우**가 있습니다. 이러한 상황에서 **break 문과 continue 문**이 필요합니다.

멈추는 break 문

반복문에서 조건의 역할은 **명령이 조건에 맞는지 체크**하고 **명령을 반복**합니다. 또한 **조건 안에는 종료 조건도 포함**되어 있습니다.

예를 들어 다음 반복문 2개를 살펴보겠습니다.

하지만 **종료 조건이 되기 전에 반복문을 빠져나와야 할 경우**도 있습니다.

예를 들어 인공지능 프로그램을 만드는 과정에서 **손실함수의 값이 터무니 없이 커질 때는 더 이상의 학습이 의미가 없습니다.**

이럴 때 **break 문을 사용**할 수 있습니다.

while문과 for문의 종료 조건

```
var i = 0;
while(i < 5) {
    i++
    console.log(i);
}
```

```
for(i = 0; i < 10; i++){
    console.log(i);
}
```

기본형 `break`

1-5. JavaScript 의 반복문 살펴보기

break 문은 단독으로 사용할 수도 있고, 반복문을 끝낼 조건과 함께 사용할 수도 있습니다. 예를 들어 앞에서 작성한 구구단 프로그램은 9단까지 모두 표시했는데, **3단 까지만 필요**할 수도 있겠죠? 이러한 **사용자의 니즈(needs)를 반영**하여 프로그램을 수정해보도록 하겠습니다.

JS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/VwMvOWV?editors=0010)

JS

```js
var i, j;  // 1 × 1 = 1, 1 × 2 = 1, 1 × 3 = 1 에서 가장 왼쪽의 값 i
           // 1 × 1 = 1, 1 × 2 = 1, 1 × 3 = 1 에서 1씩 증가하는 값 j

// 첫 번째 반복문은 외부 반복문 구조 입니다.
// 단을 처리하는 과정이죠.
for (i = 1; i <= 9; i++){
  document.write("<div>"); // <div> 테그를 추가하여 줍니다.
  document.write(`<h2>${i}단</h2>`);
  // 안쪽의 반복문 구조 입니다.
  // 여기서는 a × b = c 의 연산식을 코드화 하여 줍니다.
  for (j = 1; j <= 9; j++){
    document.write(`${i} × ${j} = ${i*j} <br>`)
    // 즉 1단의 순서에서는 1단의 연산과정을 반복하고,
    // 2단의 순서에서는 2단의 연산과정을 반복합니다.
    // 이러한 과정을 9단까지 이어서 진행하는 것이죠.
  }
  document.write("</div>"); // <div> 테그를 닫아주도록 합니다.

  // 만약 단을 제어하는 i의 값이 3이라면
  // 반복을 종료하도록 break문을 걸어주도록 합니다.
  if (i === 3) break;
}
```

break문을 사용하여
3단까지만 출력하였습니다.

1-5. JavaScript 의 반복문 살펴보기

건너뛰는 continue 문

continue문은 **주어진 조건에 해당하는 값을 만나면 해당 반복문을 건너뜁니다.** 그리고 **반복문의 맨 앞으로 되돌아가 다음 과정으로 넘어가도록 합니다.** 쉽게 말해 반복 과정을 한 차례 건너뛰게 하는 거죠. continue 문의 기본 형식은 다음과 같습니다

기본형	`continue`

다음은 숫자 1부터 10까지 중에서 짝수만 더하는 예제입니다. 이 말인 즉, **홀수는 건너뛰어야 하는 상황**인 것이죠. **i % 2** 는 **짝수인지 홀수인지를 체크하는 연산**입니다. i를 2로 나누어 나머지가 0이라면 i는 짝수, **나머지가 1이라면 홀수**가 됩니다. 이 점을 이용하여 continue를 활용해 보도록 하겠습니다.

1부터 10까지 짝수만 더하기

```
var i;
var n = 10;
var sum = 0;

for (i = 1; i <= n; i++){
  if (i % 2 === 1) // i의 값이 홀수라면 건너뛰어
    continue       // 다시 위 반복문으로 올라갑니다.
  sum += i;        // 위 조건문에 해당하지 않은 경우에만 수행
                   // 즉, 짝수만 더해줍니다.(2,4,6,8,10)

  console.log(`${i} --- ${sum}`);
}
```

결괏값

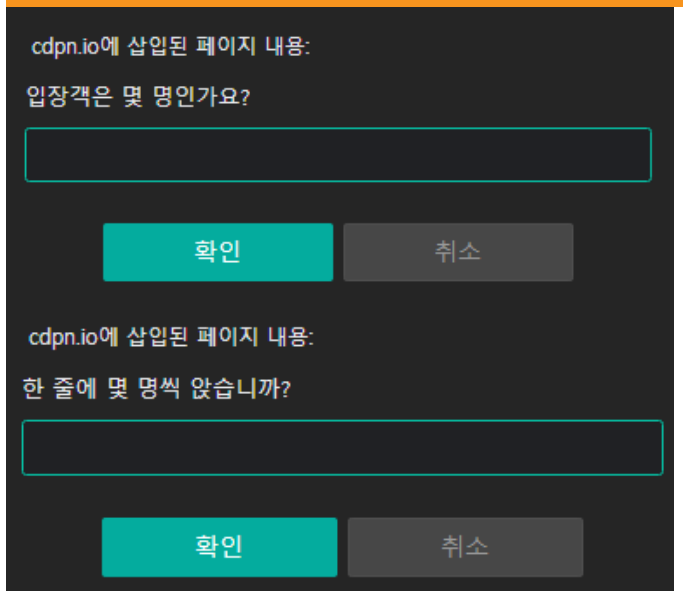
```
2 --- 2
4 --- 6
6 --- 12
8 --- 20
10 --- 30
```

1-5. JavaScript 의 반복문 살펴보기

자리 배치도 만들기 – 두 번째 예제

이번에는 이전에 살펴보았던 영화관 좌석 배치도를 지금까지 배운 내용을 바탕으로 발전된 프로그램을 만들어 보도록 하겠습니다.

결괏값



CVG 좌석 배치도

좌석 1	좌석 2	좌석 3	좌석 4	좌석 5	좌석 6	좌석 7
좌석 8	좌석 9	좌석 10	좌석 11	좌석 12	좌석 13	좌석 14
좌석 15	좌석 16	좌석 17	좌석 18	좌석 19	좌석 20	좌석 21
좌석 22	좌석 23	좌석 24				

1-5. JavaScript 의 반복문 살펴보기

HTML, CSS, JS 파일에 코드 입력하기(https://codepen.io/AI_R_IM/pen/MWEaLWe?editors=0011)

HTML

```html
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>좌석 배치도</title>
</head>
<body>
    <h1>CVG 좌석 배치도</h1>
</body>
</html>
```

CSS

```css
table, td {
  margin: auto;              /* 테이블 중앙 정렬 */
  border:1px solid #ccc;     /* 테이블 요소마다 선을 그어줍니다.*/
  border-collapse: collapse; /* 전체 테이블과 요소마다의 선을 겹쳐서 한 줄로 나타냅니다. */
}
td {
  padding:5px;
  font-size:0.9em;
}
h1 {
  text-align: center;       /* h1 테그를 중앙정렬 합니다. */
}
```

JS

```js
var i, j;
var memNum = prompt("입장객은 몇 명인가요?");      // 전체 입장객
var colNum = prompt("한 줄에 몇 명씩 앉습니까?");  // 한 줄에 앉을 사람

if(memNum % colNum == 0)              // 만약 한 번에 나눠 떨어진다면
  rowNum = parseInt(memNum / colNum); // 단순 나눗셈 결괏값을 rowNum에 저장
else                                  // 나눠 떨어지지 않았다면
  rowNum = parseInt(memNum / colNum) + 1; // 결괏값에 1을 더해 값을 저장

// 테이블로 나타내주기 위해 <table> 테그를 추가하여 주겠습니다. <table> <tr> <td>
document.write("<table>");
for(i = 0; i < rowNum; i++) {   // <table> 테그 하단에 <tr> 테그를 좌석 줄 만큼 추가하여 주는 코드입니다.
  document.write("<tr>");
  // 다음은 각 줄마다 좌석을 추가하기 위한 중첩 반복문 구조입니다.
  // 한 줄에 몇 명씩 앉을 것인지를 입력받았으니 이 값을 활용하여 줍니다.
  for(j = 1; j <= colNum; j++) {
    seatNo = i * colNum + j;   // 각 줄마다의 해당 좌석번호를 연산하는 코드입니다.
    if(seatNo > memNum) break; // 만약 좌석의 값이 입장객 보다 많아지는 경우라면 break문을 동작시켜 주겠습니다.
    document.write("<td> 좌석 " + seatNo + " </td>"); // 그리고 <td> 테그를 통해 좌석번호를 반복해서 입력하여 줍니다.
  }
  document.write("</tr>");     // 테이블 줄 바꿈을 위한 </tr> 테그
}
document.write("</table>");
```

2-1. JavaScript 의 함수

함수는 프로그래밍에서 **가장 중요한 뼈대**를 이룹니다. 실제로 웹 브라우저에서는 함수를 사용해서 명령을 내리기 때문입니다. 자바스크립트의 함수를 구성하는 여러 명령문은 앞으로 계속 배워나갈 것입니다.

여러 동작을 묶은 덩어리, 함수

자바스크립트 프로그램은 **단순히 동작 하나만 실행되는 게 아니라 여러 가지 동작이 연결**됩니다. 이렇게 동작해야 할 목적대로 묶은 명령을 **함수**라고 합니다. 함수를 사용하면 각 명령의 시작과 끝을 명확하게 구별할 수 있고, 묶은 기능에 이름을 붙여서 어디서든 같은 이름으로 명령을 실행할 수 있습니다.

자바스크립트에는 여러 함수가 미리 만들어져 있기도 합니다. alert(), prompt() 등 이 대표적인 예입니다. 우리는 alert() 함수의 내부가 어떻게 생겼는지 모르지만 alert() 함수가 알림 창을 표시할 때 쓴다는 것을 알고 있습니다. 이렇게 **미리 만들어 놓은 함수를 내장 함수**라고 합니다.

함수는 왜 사용해야 할까?

개발자는 프로그래밍을 할 때 alert() 함수처럼 **자바스크립트에 들어 있는 함수를 가져다 사용**하거나, **자신이 필요한 명령을 직접 함수로 만들어서 사용**합니다. 매번 동일한 기능을 하나하나 구현하는 것 보다는 **하나의 목적에 맞게 기능을 묶어 따로 관리하는 것이 훨씬 더 편리**하기 때문이죠.

2-1. JavaScript 의 함수

함수 선언 및 호출

함수를 사용하는 가장 기본적인 방법은 **함수를 만들 때 이름을 붙이고 필요할 때마다 함수 이름을 사용해 실행**하는 것입니다.

함수가 어떤 명령을 처리할지 미리 알려 주는 것을 **'함수를 선언한다'** 또는 **'함수를 정의한다'**고 합니다. 함수를 선언할 때는 **함수마다 서로 다른 이름**을 붙여 나중에 사용할 때 알아보기 쉽도록 합니다.

기본형

```
function 함수명() {
    명령
}
```

함수를 선언할 때는 다음과 같이 **예약어 function을 사용**하고, **중괄호({ }) 안에 실행할 여러 명령**을 넣습니다.

하지만 프로그램에서는 함수를 선언하는 것만으로는 함수가 실행되지 않습니다. **함수를 선언한 후**에 따로 **실행하는 코드를 작성**해야 됩니다. **함수를 실행하려면 다음과 같이 미리 선언한 함수명을 써서 사용**합니다.

기본형

```
함수명() 또는 함수명(변수)
```

이렇게 선언한 함수를 사용하는 것을 **'함수를 호출한다'** 또는 **'함수를 실행한다'** 고 합니다.

2-1. JavaScript 의 함수

예를 들어 평균을 구하는 간단한 프로그램을 생각해 봅시다. 먼저 함수를 사용하지 않고 필요한 명령을 순서대로 나열하면 다음과 같이 작성할 수 있습니다.

함수를 사용하지 않고 평균값 구하기	결괏값

`var x_list = [1,2,3,4,5,6,7,8,9,10];`
 	
`var list_len = x_list.length;`
`var list_sum = 0;`
 	
`for (var i = 0; i < list_len; i++){`
`  list_sum += x_list[i];`
`  var avg = list_sum / list_len;`
`}`
 	
`console.log(`${x_list}의 평균값은 ${avg}입니다.`);`
	1,2,3,4,5,6,7,8,9,10의 평균값은 5.5입니다.

이렇게 **한 번만 실행하고 끝나는 프로그램**도 있지만, **평균을 구하는 과정을 여러 번 실행**해야 한다면 어떻게 할까요? 이럴 때 함수가 필요합니다.

이번에는 **함수를 사용하여 평균을 구하는 프로그램**을 만들어 보겠습니다.

2-1. JavaScript 의 함수

평균을 구하는 함수 구축	결괏값

```
function average_func(x_list){
    var list_len = x_list.length;
    var list_sum = 0;

    for (var i = 0; i < list_len; i++){
        list_sum += x_list[i];
        var avg = list_sum / list_len;
    }
    console.log(avg);
}
```

```
var x_list = [1,2,3,4,5];
average_func(x_list = x_list);
```

```
3
```

함수를 호출하는 과정은 다음과 같습니다.

x_list 변수가 함수를 통과한 후 반환된 결괏값입니다.

웹 브라우저에서 자바스크립트 소스를 해석할 때에는 **함수 선언 부분을 가장 먼저** 합니다. 그래서 개발자가 원하는 어느 곳에 함수를 선언해 놓기만 하면 **선언한 위치와 상관없이 함수를 실행**할 수 있습니다.

함수 선언 위치는 프로그램 흐름에 영향을 주지 않습니다. 그래서 보통 한 파일 안에 여러 함수를 선언했을 때 스크립트 소스의 앞부분이나 뒷 부분에 함수 선언을 모아 놓고 필요할 때마다 함수를 호출하여 줍니다.

2-2. JavaScript 의 var를 사용한 변수의 특징

변수의 적용 범위 스코프 알아보기

기존에 var 가 있는데 왜 let과 const가 필요할까요? 그 이유를 확인하려면 먼저 var 예약어를 사용한 변수의 특징을 알아야 합니다.

자바스크립트에서 변수를 선언하고 사용할 때 **변수가 적용되는 범위를 스코프(scope)**라고 합니다. **변수가 어디까지 유효한지 범위를 가리키는 영역**이 필요한 것이죠. 자바스크립트 함수에서 변수를 사용할 때는 이 영역에 주의해야 합니다. **한 함수 안에서만 사용**하고 다른 함수에서는 사용할 수 없는 변수가 있는가 하면, 어느 함수에서나 다 사용할 수 있는 변수가 있습니다. 한 함수 안에서만 사용할 수 있는 변수를 '지역 변수' **혹은** '로컬 변수' 라고 하고, 스크립트 소스 전체에서 사용할 수 있는 변수를 '전역 변수' **또는** '글로벌 변수' 라고 합니다.

var 예약어로 지역변수 선언하기

```
function average_func(x_list){
    var list_len = x_list.length;
    var list_sum = 0;

    for (var i = 0; i < list_len; i++){
        list_sum += x_list[i];
        var avg = list_sum / list_len;
    }
    console.log(avg);
}
```

함수 안에서만 쓸 수 있는 지역 변수

지역 변수는 **함수 안에서 선언하고 함수 안에서만 사용**합니다. 지역 변수를 선언하려면 예약어 var 와 함께 변수 이름을 지정해야 합니다. 앞선 예제를 보면 average_func() 라는 함수 안에 var를 사용해 list_len, list_sum이라는 변수를 선언하였습니다.

2-2. JavaScript 의 var를 사용한 변수의 특징

오른쪽 결괏값을 살펴보면 Uncaught ReferenceError: avg is not defined 라는 오류 메시지가 나옵니다. **avg 변수가 정의되지 않았다는 뜻**이죠. 우린 분명히 avg 라는 변수를 생성하였습니니다. 하지만 변수를 정의하는 과정에서 **var로 선언하였기에 이 변수는 함수 안에서만 쓸 수 있는 '지역변수'**입니다.

스크립트 안에서 자유롭게 쓸 수 있는 전역 변수

전역 변수는 **적용 범위를 제한하지 않고** 쓸 수 있습니다. 즉, 전역 변수는 지역 변수와 달리 **스크립트 소스 코드 전체에서 사용**할 수 있습니다.

전역 변수로 사용하려면 함수 밖에서 선언하거나 함수 안에서는 **var 예약어를 빼고 선언**해야 합니다. 함수 안에서 선언한 변수라 하더라도 **var 예약어를 사용하지 않으면 전역 변수**가 됩니다.

함수 밖에서 지역 변수를 사용하면 나타나는 오류

```
function average_func(x_list){
    var list_len = x_list.length;
    var list_sum = 0;

    for (var i = 0; i < list_len; i++){
        list_sum += x_list[i];
        var avg = list_sum / list_len;
    }
    console.log(avg);
}
console.log(avg)
```

지역 변수 선언

```
▶Uncaught ReferenceError: avg is not defined
    at <anonymous>:1:13
```

var 예약어를 사용한 지역 변수와 전역 변수

```
function addNumber(a,b){
    var sum = a + b;
    multi_global = a * b;
}
undefined
addNumber(5,10)
undefined
multi_global
50
```

지역 변수 선언

전역 변수 선언

오류가 뜨지 않고 정상적으로 출력되고 있습니다.

2-2. JavaScript 의 var를 사용한 변수의 특징

var와 hoisting

자바스크립트에서 **변수를 사용할 때 조심해야 할 개념**이 있습니다. 바로 **호이스팅(hoisting)**입니다. 호이스팅이란 '끌어올린다'를 뜻합니다. 상황에 따라 변수의 선언과 할당을 분리해서 **'선언한 부분'을 스코프의 가장 위쪽으로 끌어올린다는 것**입니다. 끌어올린다고 해서 실제로 소스코드를 끌어올린다는 것은 아니고 그런 식으로 해석한다는 의미입니다.
예제와 함께 자세하게 살펴보겠습니다.

변수와 hoisting

다음 예제에서 **변수 x의 값은 할당되어 잘 출력**되고 있습니다. 하지만 학습률의 변수값은 아직 할당되지 않은 자료형인 **undefined**로 나타납니다.

자바스크립트 해석기는 함수 소스를 훑어보면서 **var를 사용한 변수는 따로 기억**해 둡니다. 즉, 변수를 실행하기 전이지만 **'이런 변수가 있구나'** 하고 기억해 두기 때문에 마치 **선언한 것과 같은 효과**가 있는 것이죠.
이 개념이 바로 **hoisting**입니다.

```
var x = 10;

function html_number(x){
    document.write(`x is ${x}<br>`);
    document.write(`LearningRate is ${LearningRate}`);
    var LearningRate = 0.0001;
}
undefined
html_number(10)
undefined
```

변수와 값 모두 선언되었지만 호이스팅으로 인해 변수 선언은 이미 완료.

결괏값

```
x is 10
LearningRate is undefined
```

변수의 선언만 된 상태

2-2. JavaScript 의 var를 사용한 변수의 특징

변수의 재선언과 재할당

var를 사용한 변수는 호이스팅 외에도 **재선언과 재할당**을 할 수 있습니다.

어떻게 보면 너무나 당연한 개념이지만 중요한 개념입니다.

우선 **재선언과 재할당**이 무엇인지 살펴보겠습니다.

일종의 **예측**을 위해서는 **적절한 값**의 **weight, bias 변수**가 필요합니다. 즉 **계속해서 값에 변화**를 줘야 하는 것이죠.

지금은 얼마만큼의 변화를 줘야 하는지 모르는 상황이기에 **아주 약간씩 값을 키워보려** 합니다. 이러한 상황에서 값을 다시 **재할당** 시켜줍니다. 그리고 변화된 weight, bias 값을 바탕으로 **다시 예측값을 구해 변수를 재선언** 해주도록 하였습니다.

이러한 개념이 **var 타입 변수의 재선언과 재할당**입니다.

https://codepen.io/AI_R_IM/pen/qBPbBVW

var 변수의 선언

```
var weight = Math.random() * 2 - 1;
var bias   = 0.0;
var input_x = [10.3];
var real_y  = [3];

var pred_y = (input_x * weight) + bias;
```

함수 정의

```
function loss(input_x, real_y, pred_y){

  var loss_error  = (real_y - pred_y);
  var loss_square = loss_error ** 2;
  var mse         = loss_square / input_x.length
  console.log(`mse : ${mse}`)
}
```

```
console.log(`before - weight : ${weight} bias : ${bias}`)
loss(input_x, real_y ,pred_y);
```

변수 재할당

```
weight += 0.01;
bias += 0.01;
```

변수 재선언

```
var pred_y = (input_x * weight) + bias;
console.log(`before - weight : ${weight} bias : ${bias}`)
loss(input_x, real_y ,pred_y);
```

2-3. JavaScript 의 let과 const

let을 사용한 변수의 특징

이전에는 자바스크립트 프로그램에서는 **var 예약어만으로 변수를 선언**하였습니다. 앞에서 살펴본 것 처럼 var 예약어를 빠뜨리면 **의도하지 않게 전역 변수**가 되기도 하고, 프로그램 길이가 길어지다 보니 실수로 사용하는 변수를 재선언하거나 값을 재할당 해버리는 경우가 생기기도 하였습니다. 즉 실수가 많아진 것이죠.

그래서 ES6에서는 **변수는 선언하기 위한 예약어로 let과 const가 추가**되었고, **되도록이면 var 예약어는 사용하지 않을 것을 권장**합니다.

예약어 var과 let, const의 가장 큰 차이는 **스코프의 범위** 입니다. var는 함수 영역의 스코프를 가지지만 **let과 const는 블록 영역의 스코프**를 가집니다.

블록 안에서만 쓸 수 있는 변수

let 예약어로 선언한 변수는 **변수를 선언한 블록({ }로 묶은 부분)에서만 유효**하고 블록을 벗어나면 사용할 수 없습니다. 예제를 통해 좀 더 자세하게 살펴보도록 하겠습니다.

2-3. JavaScript 의 let과 const

다음 예제를 보면 calcSum() 함수에서 변수 sum은 let 예약어를 사용하여 선언합니다. 즉, sum은 함수 calcSum()의 블록 안에서만 사용할 수 있습니다. 또한 for문의 카운터 변수 i도 let 예약어를 사용하였습니다. 그렇게 되면 변수 i는 for문을 벗어나게 되면 사용할 수 없습니다.

블록 변수 선언하기(https://codepen.io/AI_R_IM/pen/JjrGReN?editors=0011)

JS

```
function calcSum(n) {
  let sum = 0;
  for (let i = 1; i < n + 1; i++) {
    sum += i;
  }
  console.log(sum); // let으로 선언한 변수 i는 중괄호{}안 의 범위를 갖습니다.
  console.log(i);
}
calcSum(10);
```

```
55

Uncaught ReferenceError: i is not defined
 at https://cdpn.io/cp/internal/boomboom/pen.js?key=pen.js-52a085f6-f03e-cf67-74b8-b067d05a6a12:7
```

재할당은 가능하지만 재선언은 할 수 없는 변수 let

let 예약어를 사용하면 선언한 변수는 값을 재할당할 수는 있지만 변수를 재선언할 수는 없습니다. 이렇게 되면 var와 같이 실수로 같은 이름의 변수를 사용할 걱정이 사라지게 되는 것이죠.

더불어 let 예약어는 hoisting이 없습니다. 그렇기에 let 예약어를 통해 변수를 선언하기 전에 사용하게 되면 오류 메시지를 출력하게 됩니다.

2-3. JavaScript 의 let과 const

호이스팅이 없는 let 변수(https://codepen.io/AI_R_IM/pen/JjrGRQy?editors=1011)

JS

```js
// 이전에 살펴보았던 XOR GATE에 필요한 데이터생성 함수 구축 과정입니다.
function getRandomSample() {
  let x, y // let 예약어를 통해 x,y 변수를 미리 선언하였습니다.
  x = [Math.random() * 2 - 1, Math.random() * 2 - 1]
  if ((x[0] > 0 && x[1] > 0) || (x[0] < 0 && x[1] < 0 )){
    y = 0
  } else {
    y = 1
  return [x, y]
}}

// 함수 호출 및 결괏값 저장
result = getRandomSample();
console.log(result);
```

HTML

```html
<body>
  <h1>XOR Gate</h1>
  <p>X1 : <span id = 'xor_x1'></span></p>
  <p>X2 : <span id = 'xor_x2'></span></p>
  <p>Y  : <span id = 'xor_y'></span></p>
</body>
</html>
```

결괏값

XOR Gate

X1 : 0.4792108463423057

X2 : -0.4641981343527659

Y : 1

2-3. JavaScript 의 let과 const

const를 사용한 변수의 특징

const 역시 let과 마찬가지로 **변수를 선언할 때 사용하는 예약어** 입니다. 이름에서 예상할 수 있듯이 **const로 선언한 변수는** 상수 변수(constant)입니다. **상수는 프로그램안에서 변하지 않는 값**을 뜻합니다. 즉, 변하지 않는 값을 변수로 선언할 때 예약어 const를 사용합니다. 프로그램 안에서 **특정한 상숫값을 자주 사용한다면 변수에 담아서 사용하는 것이 편리**합니다.

const로 할당된 변수는 **재선언하거나 재할당할 수 없으며**, let 예약어를 사용한 변수처럼 **블록 레벨의 스코프**를 가집니다. 다음은 const를 사용해 **인공지능 프로그램에 사용되어지는 변수**들을 선언하고 값을 할당하는 예제입니다. 이름이 같은 변수는 재선언할 수 없습니다.

재선언 할 수 없는 const 변수 선언

JS

```js
const dimIn = 2  // 입력값 차원 수를 정의합니다.
const dimOut = 1 // 출력값 차원 수를 정의합니다.

const numNeurons1 = 20 // 첫 번째 은닉계층의 노드 수를 정의합니다.
const numNeurons2 = 5  // 두 번째 은닉계층의 노드 수를 정의합니다.

const learningRate = 0.01  // 학습률을 정의합니다.
const numIterations = 100  // 학습 횟수를 정의합니다.(epochs)
const batchSize = 20       // 배치사이즈를 정의합니다.
```

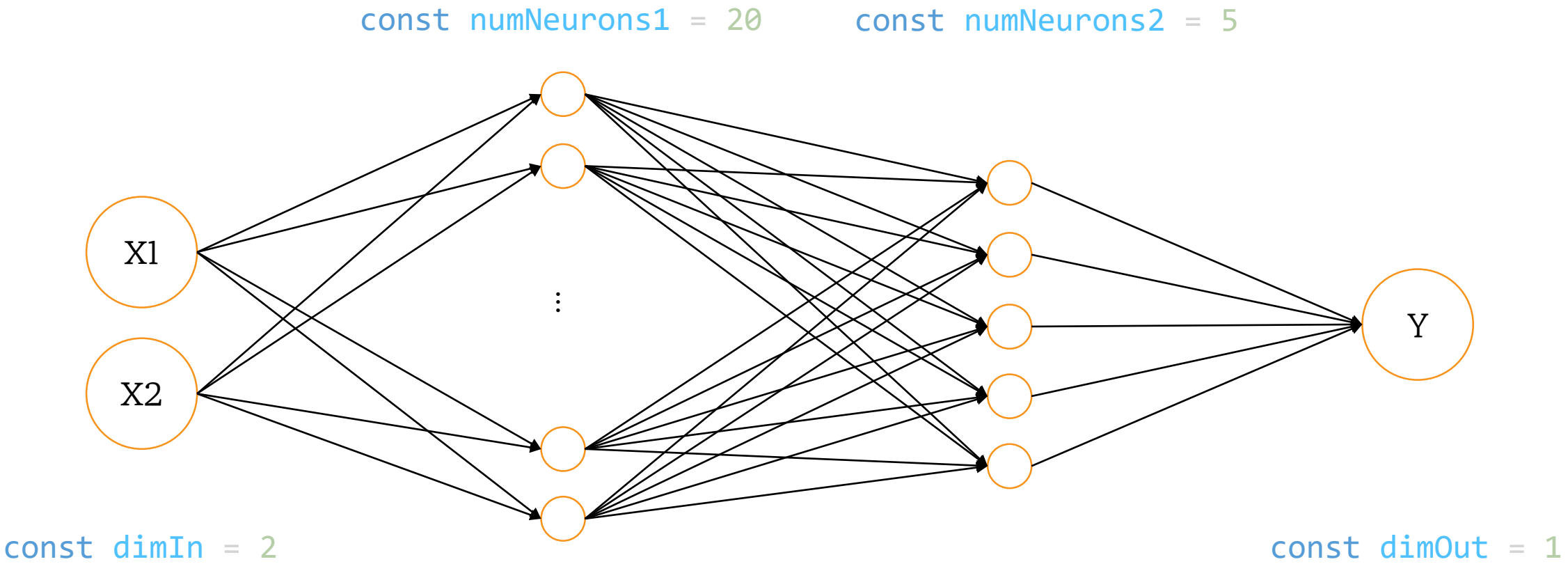
2-3. JavaScript 의 let과 const

인공지능 프로그램으로 XOR 문제 해결을 위한 const 변수 선언하기

JS

```js
const dimIn = 2  // 입력값 차원 수를 정의합니다.
const dimOut = 1 // 출력값 차원 수를 정의합니다.

const numNeurons1 = 20 // 첫 번째 은닉계층의 노드 수를 정의합니다.
const numNeurons2 = 5  // 두 번째 은닉계층의 노드 수를 정의합니다.
```

XOR 문제를 해결하기 위해 신경망에 필요한 변수들을 선언하였습니다. 이 과정에서 왠만하면 변경할 일이 없는 값들은 const 예약어를 사용해 변수를 선언합니다.



2-4. JavaScript 의 함수 표현식

지금까지 살펴본 함수의 사용 방법은 **함수명을 선언하고 그 이름을 사용해 호출**하였습니다. 이 방법은 함수 이름만 알면 어디서나 호출해서 실행할 수 있어서 많이 사용하였습니다. 이 밖에도 따로 **함수 이름을 지정하지 않고 사용**하거나, **함수를 호출하지 않고 바로 실행하는 방법**도 있습니다. 이번 절에서는 **익명 함수**와 **즉시 실행 함수**, **화살표 함수**를 살펴보겠습니다.

익명 함수

익명 함수는 이름이 없는 함수를 말합니다. 즉, **익명 함수를 선언할 때는 이름을 붙이지 않습니다.** 예를 들어 다음과 같이 이름 없이 바로 선언할 수 있습니다.

익명 함수는 이름이 없는데 어떻게 실행해야 할까요? 익명 함수는 **함수 자체가 식** 이므로 **함수를 변수에 할당**할 수 있으며, **또한 다른 함수의 매개변수로 사용**할 수도 있습니다.

다음은 **두 수를 더하는 익명 함수를 변수 sum에 할당하는 예제**입니다. 이렇게 **변수에 저장된 익명 함수는** 함수 이름 대신 변수를 이용해 함수를 실행합니다.

익명 함수 실행하기

JS

```js
let a = 10;
let b = 15;
var sum = function(a,b) {
  return a + b;
}

document.write(`함수 실행 결과 : ` + sum(10, 20));
```

2-4. JavaScript 의 함수 표현식

즉시 실행 함수

일반적으로 함수는 선언하고 필요할 때마다 호출해서 실행하는 방법을 많이 사용합니다. 하지만 **한 번만 실행하는 함수라면 함수를 정의하면서 동시에 실행**할 수 있습니다. 바로 **즉시 실행 함수** 입니다. 즉시 실행 함수는 **함수를 실행하는 순간에 자바스크립트 해석기에서 함수를 해석**합니다. 즉시 실행 함수의 기본 형식은 다음과 같습니다. 함수를 식 형태로 선언하므로 마지막에 세미콜론(;)을 붙여주도록 합니다.

기본형

```
(function(매개변수){
    명령
}(인수));
```

예를 들어 사용자에게 이름을 받아서 인사말을 표시하는 함수는 다음과 같이 작성할 수 있습니다.

즉시 실행 함수 만들기(https://codepen.io/AI_R_IM/pen/qBPbqZx?editors=0111)

```
(function() {
  var userName = prompt("참석자의 성함을 입력하여 주세요.");
  document.write(`안녕하세요. <span class = 'accent'> ${userName} </span>님.`)
}());
```

명령

2-4. JavaScript 의 함수 표현식

만약 **매개변수를 사용하는 함수**라면 **선언 부분 끝에 함수 실행을 위한 인수**를 넣어줍니다. 예를 들어 두 수를 더하는 함수는 다음과 같이 작성할 수 있습니다.

매개변수가 있는 즉시 실행 함수 만들기

매개 변수

```
(function(a,b) {
  sum = a + b;
}(100,200)); 인수
document.write(`함수 실행 결과 : ${sum}`);
```

화살표 함수

ES6 버전부터는 **=> 표기법(화살표 표기법)**을 사용해 함수 선언을 좀 더 간단하게 작성할 수 있습니다. 이 방법은 간단히 **화살표 함수**라고 하는데 **익명 함수에서만 사용**할 수 있습니다. 화살표 함수의 기본 형식은 다음과 같이 **매개변수와 함수 내용으로 구성**됩니다.

기본형 (매개변수) => { 함수 내용 }

지금부터 매개변수의 개수에 따라 함수를 작성하는 방법과 화살표 함수의 표기법을 간단하게 살펴보겠습니다.

2-4. JavaScript 의 함수 표현식

매개변수가 없을 경우

매개변수가 없을 경우 **매개변수를 넣는 괄호 안은 비워**둡니다.

예를 들어 매개 변수 없이 간단한 인사말을 화면에 표시하는 함수는 다음과 같이 작성할 수 있습니다.

매개변수가 없는 함수

안녕하세요

```js
const hi = function() {
  return "안녕하세요";
}

document.write(hi());
```

이 함수를 **화살표 함수로 작성**하면 다음과 같습니다.

그리고 **중괄호 안에 함수 내용이 한 줄뿐이라면 중괄호를 생략**해서 다음과 같이 작성할 수도 있습니다.

이때 **return문은 생략된 것으로 간주**합니다.

매개변수가 없는 화살표 함수

```js
// 매개변수가 없는 화살표 함수
const hi = () => {
  return '안녕하세요'
};
document.write(hi());

// 매개변수가 없는 화살표 함수(중괄호 생략)
const hi = () => '안녕하세요' ;
document.write(hi());
```

2-4. JavaScript 의 함수 표현식

매개변수가 1개인 경우

매개변수가 1개인 함수, 화살표 함수

참여자님, 안녕하세요.

JS

```js
// 매개변수가 1개인 함수
let hi = function(user) {
  document.write(user + "님, 안녕하세요.");
}

hi('참여자');

// 매개변수가 1개인 화살표 함수
let hi = user => {
  document.write(user + "님, 안녕하세요.");
}

hi('참여자');
```

2-4. JavaScript 의 함수 표현식

매개변수가 2개 이상인 경우

매개변수가 **둘 이상인 경우 화살표 함수**는 **(매개변수) => { ... }** 처럼 사용합니다. 다음은 매개변수가 2개인 경우에 사용하는 함수입니다. 매개변수를 추가하려면 괄호 안에 쉼표(,)로 구분합니다.

매개변수가 2개인 함수

3

JS

```js
// 매개변수가 2개인 함수
let sum = function(a, b) {
  return a + b;
}

document.write(`${sum(1,2)}`);

// 매개변수가 2개인 화살표 함수
let sum = (a,b) => a + b;

document.write(`${sum(1,2)}`);
```

2-4. JavaScript 의 함수 표현식

화살표 함수는 인공지능 프로그램을 만드는 과정에서도 종종 사용되고는 합니다. 더불어 map()도 함께 살펴보겠습니다. 우선 map()입니다. map의 **기본 원리**는 간단합니다. **반복문을 돌며 배열 안의 요소들을 1대 1로 짝지어 주는 것**입니다. 그래서 이름이 **map**입니다. **매핑한다고 표현**하죠. 어떻게 **짝지어줄 것인가 정의한 함수를 메서드의 인자로 넣어주면 되는 것**입니다.

map()를 사용하는 방법

JS

```js
const x_array = [1, 2, 3];

// 변수 x_array에 map()을 적용하여 접근합니다.
// x_array에 속한 값들을 하나씩 x로 꺼내 중괄호 안의 명령을 수행합니다.
let result = x_array.map((x) =>
  // 명령 수행
  console.log(x)
);
result;
```

JS

```js
// x_array안에 담긴 값들을 하나씩 x에 할당하여줍니다.
result = x_array.map((x) => {
  // 꺼내진 x값을 가지고 명령을 수행합니다.
  return x + 1;
});

result;
```

2-4. JavaScript 의 함수 표현식

즉시 실행 함수 만들기(https://codepen.io/AI_R_IM/pen/gOGPLNp?editors=1011)

이후에 만들 함수 중 하나 입니다. 이 함수의 경우 map()을 활용하여 예측값을 얻는 과정이 포함되어 있습니다. 또한 TF.JS를 활용하게 됩니다.

이 과정은 이후에 천천히 살펴보도록 하겠습니다.

JS

```js
function test(xs, ys) {
  // tf.tidy()는 지금 당장 중요하지 않으니 그냥 넘어가주세요.
  tf.tidy(() => {

    // map()를 사용하였습니다.
    // xs에 있는 것을 하나씩 x에 할당하여
    const predictedYs = xs.map((x) =>
                              //예측함수(predict())를 통과시켜 줍니다.
                              Math.round(predict(tf.tensor2d(x, [1,2])).dataSync())
                             ); // 그리고 결괏값은 const predictedYs에 담아주도록 합니다.

    let predicted = 0 // 정답을 맞춘 수 만큼 카운트를 수행하기 위해 let 타입 변수로 초기화.

    // xs의 수 만큼 반복문을 동작시켜줍니다.
    for (let i = 0; i < xs.length; i++){
      // 만약 실제 결과(ys)와 예측 결괏값(predictedY)가 같다면
      if (ys[i] == predictedYs[i]){
        predicted ++ // 증감 연산자로 predicted 변수를 1씩 증가 시켜 줍니다. .
      }
    }
    // 반복문이 모두 종료되었다면 결과를 출력하여 줍니다.
    console.log(`Num correctly predicted : ${predicted} of ${xs.length}`)
    console.log(`Accuracy : ${predicted / xs.length}`)
    })
}
```

2-5. JavaScript 의 이벤트와 이벤트 처리기

이벤트 알아보기

인터넷 초기 시절에 웹 페이지는 텍스트와 이미지를 적절히 배열해 놓고 정보를 제공하는 것만으로도 충분했지만 요즘은 그렇지 않습니다.

메인 메뉴를 클릭하면 서브 메뉴가 자연스럽게 펼쳐지는 것은 기본이고, 어떤 사이트는 페이지 로딩이 끝나면 배경 화면이 움직이기도 합니다. 이런 효과는 브라우저의 이벤트(event) 개념이 도입되면서 새롭게 구현된 것입니다.

웹 브라우저를 기본으로 실행하는 자바스크립트에서 이벤트는 중요한 개념이죠.

이벤트는 웹 브라우저나 사용자가 행하는 어떤 동작을 말합니다. 예를 들어 키보드에서 키를 누르는 것도, 웹 브라우저에서 새로운 페이지를 불러오는 것도 이벤트 입니다.

하지만 웹 브라우저 안에서 이뤄지는 모든 동작이 이벤트는 아닙니다. 이벤트는 웹 페이지를 읽어 오거나 링크를 클릭하는 것 처럼 웹 문서 영역을 벗어나 클릭하는 행위는 이벤트가 아닙니다. 예를 들어 브라우저 창 맨 위의 제목 표시줄을 클릭하는 것은 이벤트라 하지 않죠.

자바스크립트의 이벤트는 주로 마우스나 키보드를 사용할 때, 웹 문서를 불러오 때, form에 내용을 입력할 때 발생합니다.

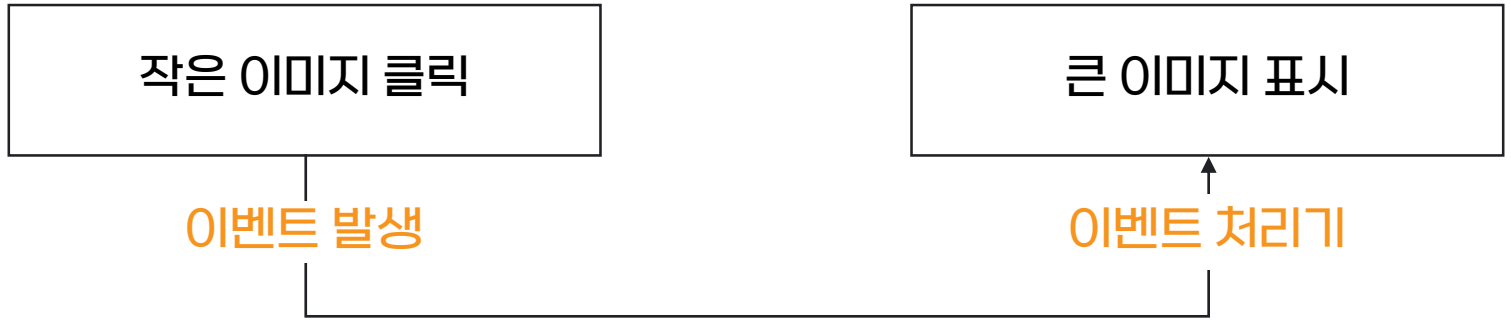
2-5. JavaScript 의 이벤트와 이벤트 처리기

마우스 이벤트

마우스 이벤트는 마우스를 이용해서 **버튼이나 휠 버튼을 조작할 때 발생**합니다.

이벤트 처리기 알아보기

이벤트에서는 작은 이미지를 클릭하면 큰 이미지를 보여 주거나 폼에서 항목을 선택하면 해당 페이지로 이동하는 등 여러 가지 동작을 실행합니다. 아래 그림과 같이 **웹 문서에서 이벤트가 발생하면 처리하는 함수를 이벤트 처리기 또는 이벤트 핸들러**라고 합니다.



이벤트를 처리하는 가장 기본적인 방법은 **이벤트가 발생한 HTML 태그에 이벤트 처리기를 직접 연결**하는 것입니다. 이 방법은 자바스크립트 초기 버전부터 사용했으며 지금도 많이 사용하고 있죠.

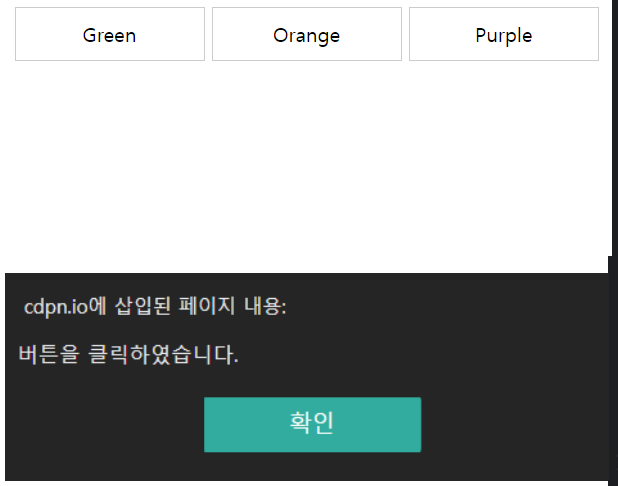
2-5. JavaScript 의 이벤트와 이벤트 처리기

이벤트 처리기의 기본 형식은 다음과 같습니다. 먼저 HTML 태그 안에서 **'on' 다음에 '이벤트 명'을 붙여** 속성 이름을 만들고, 그 다음에 실행할 **이벤트 처리기의 함수명을 작성**하면 됩니다.

기본형 <태그 on이벤트명 = '함수명'>

예를 들어 **마우스로 클릭했을 때의 이벤트 이름**은 **click** 입니다. <a> 태그 안에 onclick(on+click) 속성을 만들고 실행할 함수를 연결합니다. 다음은 목록에서 각 버튼을 클릭하면 알림 창을 표시하는 예제입니다.

버튼을 클릭하면 알림 창 표시하기 – 첫 번째 예제(https://codepen.io/AI_R_IM/pen/dyVGvBd)



HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <ul>
    <li><a href = '#' onclick = "alert('버튼을 클릭하였습니다.')">Green</a></li>
    <li><a href = '#' onclick = "alert('버튼을 클릭하였습니다.')">Orange</a></li>
    <li><a href = '#' onclick = "alert('버튼을 클릭하였습니다.')">Purple</a></li>
  </ul>
</body>
</html>
```

2-5. JavaScript 의 이벤트와 이벤트 처리기

이벤트가 발생한 후에 **여러가지 명령을 실행**해야 한다면, 그 **명령을 묶어서 하나의 자바스크립트 함수**로 만드는 것이 좋습니다. 그리고 **이벤트가 발생할 때 함수 이름과 인수를 지정하여 실행**합니다. 다음은 **웹 요소의 배경색을 바꾸는 changeBg(color)함수를 미리 만들어서 실행하는 예제**입니다. 항목을 클릭할 때마다 적힌 글자의 색상을 인수로 넘겨 changeBg(color) 함수를 실행합니다. (※css 코드도 확인해 주세요)

버튼을 클릭하면 알림 창 표시하기 – 두 번째 예제(https://codepen.io/AI_R_IM/pen/eYGJoxN)

HTML

```html
<body>
  <ul>
    <li><a href="#" onclick="changeBg('green')">Green</a></li>
    <li><a href="#" onclick="changeBg('orange')">Orange</a></li>
    <li><a href="#" onclick="changeBg('purple')">Purple</a></li>
  </ul>
  <div id="result"></div>

  <script>
    function changeBg(color) {
      var result = document.querySelector('#result');
      result.style.backgroundColor = color;
    }
  </script>
```

changeBg(color) 함수

id가 result인 것을 찾아서 이를 변수화 합니다.
onclick 함수가 활성화 되면
함수 안의 글자를 넘겨서 배경 색상을 전환하여 줍니다.

2-5. JavaScript 의 이벤트와 이벤트 처리기

버튼을 클릭해서 상세 설명 표시하거나 닫기

여기에서는 **[상세 설명 보기]** 버튼을 클릭하면 상세 설명 글을 표시하고, [상세 설명 닫기] 버튼을 클릭하면 표시된 설명 글을 사라지게 하는 프로그램을 만들어보도록 하겠습니다.

버튼 클릭 예제 결괏값 (https://codepen.io/AI_R_IM/pen/rNGxgaL?editors=1010)

Google

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nisl tincidunt eget nullam non. Quis hendrerit dolor magna eget est lorem ipsum dolor sit.

상세 설명 닫기

상세 설명 보기

2-5. JavaScript 의 이벤트와 이벤트 처리기

HTML

```html
<body>
  <!-- 전체를 item으로 묶어 주도록 합니다.    -->
  <div id="item">
    <!-- 이미지를 띄워주는 코드입니다. -->
    <img src="https://www.google.co.kr/images/srpr/logo11w.png" alt="구글 이미지">
    <!-- 버튼을 생성하는 코드입니다. 버튼을 클릭(onclick)하게 되면 showDetail() 라는 함수가 동작하게 합니다. -->
    <button class="over" id="open" onclick = "showDetail()">상세 설명 보기</button>
    <!-- 상세 설명을 위한 칸 입니다. -->
    <div id="desc" class="detail">
      <h4>Google</h4>
      <!-- 로렘 입숨을 통해 무작위로 생성된 글 입니다.-->
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nisl tincidunt eget nullam non. Quis hendrerit dolor magna eget est lorem ipsum dolor sit.</p>
      <!-- 다시한번 버튼을 생성합니다. 이 버튼은 상세 설명을 닫는 버튼입니다.
      마찬가지로 버튼을 클릭하면 hideDetail()가 실행됩니다.-->
      <button id="close" onclick = "hideDetail()">상세 설명 닫기</button>
    </div>
  </div>
</body>
```

2-5. JavaScript 의 이벤트와 이벤트 처리기

JS

```javascript
// 상세 설명을 보여주는 showDetail()함수 입니다.
function showDetail() {
  // id가 desc 인 것을 찾습니다. 그리고 버튼을 클릭(함수가 동작)하면 상세정보가 출력(block)되게끔 합니다.
  document.querySelector('#desc').style.display = "block";
  // 동시에 버튼 자체는 사라지게끔 합니다.
  // 버튼의 id는 open 이었으니 이를 none으로 처리하는 것입니다.
  document.querySelector('#open').style.display = "none";
}

// 상세 설명을 닫는 hideDetail() 함수 입니다. 이는 위와는 반대되는 성질을 갖습니다.
function hideDetail() {
  document.querySelector('#desc').style.display = "none";    // 상세 설명 부분을 화면에서 감춤
  document.querySelector('#open').style.display = "block";   // '상세 설명 보기' 단추를 화면에 표시
}
```

2-6. DOM을 이용한 이벤트 처리기

지금까지 **이벤트 처리기**를 지정하는 방법은 HTML이 주인이 되어 자바스크립트의 함수를 불러와서 사용하였습니다. 하지만 DOM을 사용하면 자바스크립트가 주인이 되어 HTML의 요소를 가져와서 이벤트 처리기를 연결합니다. HTML과 자바스크립트 중에서 어느 쪽을 중심으로 잡는가에 따라 지정하는 방법이 다르다는 것을 알아보도록 하겠습니다. (※ DOM은 이후에 좀 더 깊게 다뤄볼 예정입니다. 여기서는 DOM을 이용해 이벤트 처리기를 지정하는 방법이 있다는 정도만 알고 넘어가겠습니다.)

예를 들어 웹 요소를 클릭했을 때 실행할 함수를 연결하려면 다음과 같은 기본 형식으로 사용합니다.

기본형

```
웹 요소.onclick = 함수;
```

자바스크립트에서는 웹 요소를 여러 방법으로 가져올 수 있는데 그중에서 함수 querySelector()를 사용하여 가져오는 것이 쉽습니다. querySelector()의 괄호 안에는 클래스 이름이나 id이름 또는 다양한 선택자를 넣을 수 있습니다. 다음은 id = 'change'인 버튼을 클릭했을 때 글자색을 바꾸는 예제입니다. querySelector()를 사용해 버튼 요소를 가져오고 changeBttn변수에 저장합니다. 그리고 changeBttn변수에 onclick을 사용해 changeColor() 함수를 연결합니다. 이때 주의할 점은 함수의 이름만 사용하고 괄호는 붙이지 않아야 한다는 것입니다.

2-6. DOM을 이용한 이벤트 처리기

DOM 을 활용한 글 색상 변경 예제(https://codepen.io/AI_R_IM/pen/ExwPzEb?editors=1010)

글자색 바꾸기

Senectus et netus et malesuada. Nunc pulvinar sapien et ligula ullamcorper malesuada proin. Neque convallis a cras semper auctor. Libero id faucibus nisl tincidunt eget. Leo a diam sollicitudin tempor id. A lacus vestibulum sed arcu non odio euismod lacinia. In tellus integer feugiat scelerisque. Feugiat in fermentum posuere urna nec tincidunt praesent.

글자색 바꾸기

Senectus et netus et malesuada. Nunc pulvinar sapien et ligula ullamcorper malesuada proin. Neque convallis a cras semper auctor. Libero id faucibus nisl tincidunt eget. Leo a diam sollicitudin tempor id. A lacus vestibulum sed arcu non odio euismod lacinia. In tellus integer feugiat scelerisque. Feugiat in fermentum posuere urna nec tincidunt praesent.

HTML

```html
<body>
  <!-- button 태그를 사용해 버튼을 생성하고 id는 change로 설정   -->
  <button id="change">글자색 바꾸기</button>
  <p>Senectus et netus et malesuada. Nunc pulvinar sapien et ligula ullamcorper malesuada proin. Neque convallis a cras semper auctor. Libero id faucibus nisl tincidunt eget. Leo a diam sollicitudin tempor id. A lacus vestibulum sed arcu non odio euismod lacinia. In tellus integer feugiat scelerisque. Feugiat in fermentum posuere urna nec tincidunt praesent.</p>
</body>
```

2-6. DOM을 이용한 이벤트 처리기

JS

```js
// 방법 1: 웹 요소를 변수로 지정 & 미리 만든 함수 사용
// 문서에서(document) id가 change 라는 요소를 찾아 선택(querySelector)합니다.
// 선택된 요소는 변수화(var changeBttn) 하여줍니다.
var changeBttn = document.querySelector("#change");
// changeBttn 변수에 onclick함수를 적용하고 이벤트가 발생하면 동작할 함수(changeColor)를 알려줍니다.
changeBttn.onclick = changeColor;

// 위에서 사용된 changeColor()를 정의합니다.
function changeColor() {
  // 문서(document)에서 요소 p를 찾습니다(querySelector).
  // 그리고 스타일 시트의 문법을 적용하여 색상을 원하는 색으로 변경하여 줍니다.
  // .style.color = "#f00";
  document.querySelector("p").style.color = "#f00";
}
```

2-6. DOM을 이용한 이벤트 처리기

개발자의 특성에 맞게 다양한 방법으로 정의할 수 있습니다.

JS

```js
// 방법 2: 웹 요소를 따로 변수로 만들지 않고 사용
// 웹에서 ID가 change인 것을 찾습니다. 그리고 onclick 이벤트가 발생하면 즉시 changeColor 함수가 동작합니다.
// document.querySelector("#change").onclick = changeColor;

// 함수는 위와 동일합니다.
// function changeColor() {
//   document.querySelector("p").style.color = "#f00";
// }

// 방법 3: 직접 함수를 선언 (익명함수)
// 이전에 배웠던 익명함수를 선언하는 방식을 활용합니다.
// 마찬가지로 문서에서 ID가 change인 것을 찾습니다. 그리고 onclick 함수가 동작하면
// 바로 함수가 동작합니다.
// document.querySelector("#change").onclick = function() {
//   document.querySelector("p").style.color = "#f00";
// };
```

3-1. 자바스크립트와 객체

객체란?

프로그래밍 언어에서 객체는 여러 가지 의미로 해석할 수 있지만, **자바스크립트에서 객체**는 **프로그램에서 인식할 수 있는 모든 대상**을 가리킨다는 정도로 이해해 두세요. 자바스크립트는 웹 사이트나 웹 어플리케이션을 개발하는 언어이므로 **웹과 관련된 대상을 모두 객체로 인식**합니다. 그렇다면 자바스크립트의 객체는 어떤 것이 있는지 살펴보도록 하겠습니다.

자바스크립트에서 사용하는 객체는 다음과 같습니다.

문서 객체 모델(DOM) : 웹 문서 자체도 객체이고 그 안에 삽입되어 있는 이미지와 링크, 텍스트 필드 등 모두 객체 입니다. 일반적으로 웹 문서에서 삽입하는 요소는 document, image, link 객체 등이 있습니다.

브라우저 관련 객체 : 웹 브라우저에서 사용하는 정보도 객체로 나타낼 수 있습니다. 사용하는 브라우저 정보를 담고 있는 navigator 객체를 비롯해 history, location, screen 객체 등이 있습니다.

내장 객체 : 웹 프로그래밍을 할 때 자주 사용하는 요소는 자바스크립트 안에 미리 객체로 정의되어 있는데, 이를 내장 객체 라고 합니다. 예를 들어 날짜, 시간과 관련된 프로그램을 개발하려면 Date 객체를 가져와 쉽게 사용할 수 있습니다.

3-1. 자바스크립트와 객체

객체란?

흔히 자바스크립트는 모든 것이 객체라는 말이 있을 정도로 자바스크립트에서 객체는 중요한 개념입니다.

객체의 인스턴스 만들기

자바스크립트에서 객체는 참조 형태로 사용해야 합니다. 즉, 객체 자체가 아니라 인스턴스(instance)의 형태로 만들어서 사용해야 합니다. 자바스크립트 안에 정의된 객체는 그대로 두고 객체와 똑같은 속성과 기능을 만들 수 있습니다. 자바스크립트의 객체가 '틀' 이라면 그 틀을 기본으로 해서 계속 같은 모양으로 찍어 내는 것이 인스턴스입니다. 그리고 그 인스턴스에 식별자를 붙여 사용하죠.

객체의 인스턴스를 만들 때는 다음과 같이 new라는 예약어를 사용합니다. new 뒤에 만들려고 하는 객체 이름을 써주시면 됩니다.

기본형	new 객체명

3-1. 자바스크립트와 객체

예를 들어 **현재 날짜와 시간**을 바탕으로 **전동 자전거의 헤드라이트를 밝히거나 끄는 프로그램**을 작성한다고 생각해보겠습니다. 그렇다면 **기본적으로 현재 시간을 받는 무엇인가가 필요**하게 됩니다. 자바스크립트에서는 날짜나 시간 정보는 **Date 객체에 저장하고 관리**합니다. 그래서 날짜나 시간 관련 프로그램을 작성하려면 가장 **먼저 Date 객체의 인스턴스**를 만들고, 그 **인스턴스를 변수에 저장**해서 사용합니다.

Date 객체의 인스턴스 만들기

JS

```js
var now = new Date(); // Date 인스턴스 만들고 변수에 저장
                      // 현재 시간을 알려주는 Date()가 '틀' 이 되는 것이며
                      // 틀에서 찍혀 나온 현재 시간이 now 라는 변수가 됩니다.

document.write(`현재 시각 : ${now}`);
```

다음 코드는 **new 예약어를 사용**하여 **Date 객체의 인스턴스**를 만듭니다. 그리고 변수 now에 저장한 변숫값을 브라우저에 나타냈습니다.

3-1. 자바스크립트와 객체

프로퍼티와 메서드 이해하기

객체에는 프로퍼티(property)와 메서드(method)가 있는데, 쉽게 말해 **프로퍼티**는 **객체의 특징이나 속성**을 나타내고, **메서드**는 **객체에서 할 수 있는 동작**을 표현합니다.

마침표 표기법으로 프로퍼티와 메서드 작성하기

인스턴스는 **객체의 프로퍼티와 메서드를 그대로 물려받아서 똑같이 사용**합니다. 프로퍼티와 메서드를 표시하려면 **인스턴스 명 뒤에 마침표(.)**를 붙이고 **객체의 프로퍼티나 메서드 이름을 작성**합니다. 이때 **메서드는 함수와 같은 역할**을 하므로 **getHours() 처럼 이름 옆에 괄호**를 넣어야 합니다. 앞서 만든 예제에 인스턴스 메서드를 추가해 보겠습니다. Date 객체에는 현재 날짜와 시간 정보를 로컬 형식으로 바꿔 주는 toLocalString() 메서드가 있습니다. 다음과 같이 **인스턴스 now를 만든 후 toLocalString 메서드를 적용**해 보겠습니다.

인스턴스와 객체(https://codepen.io/AI_R_IM/pen/xxXZNvJ)

```
var now = new Date(); // Date 인스턴스 만들고 변수에 저장
                      // 현재 시간을 알려주는 Date()가 '틀' 이 되는 것이며
                      // 틀에서 찍혀 나온 현재 시간이 now 라는 변수가 됩니다.

document.write(`현재 시각 : ${now.toLocaleDateString()}`);
// now 인스턴스에 Date() 객체 에서 사용할 수 있는 toLocaleDateString()를 붙여
// 사용할 수 있습니다.
```

3-2. 자바스크립트의 내장 객체

Array 객체

Array 객체는 **자바스크립트의 여러가지 내장 객체 중에서 배열**을 다룹니다. 배열은 자바스크립트에서 자주 사용하는 자료형이므로 **Array 객체의 주요 프로퍼티와 메서드를 잘 알고 사용하는 것이 중요**합니다.

Array 객체로 배열 만들기

자바스크립트에서는 **배열을 쉽게 만들고 다룰 수 있는 Array 객체가 내장**되어 있습니다. **Array 객체를 사용하려면 먼저 인스턴스**를 만들어야 합니다. 먼저 초깃값이 없는 상태에서 단순히 객체의 인스턴스만 만든다면 new 예약어를 사용해 변수에 할당하면 됩니다. 이때 **다음과 같이 배열의 크기를 지정하지 않을 수도 있고, 몇 개의 요소가 있는지 크기를 지정**할 수도 있습니다.

Array 객체 인스턴스 만들기(https://codepen.io/AI_R_IM/pen/BawjgzK?editors=0011)

JS

```js
var numbers_a = new Array();  //배열의 크기를 지정하지 않음
console.log(numbers_a);
var numbers_b = new Array(4); //배열의 크기 지정
console.log(numbers_b);

var numbers_c = ['one', 'two','three', 'four']; //배열 만들기
console.log(numbers_c);
var numbers_d = Array('one', 'two','three', 'four'); //배열 만들기
console.log(numbers_d);

console.log(numbers_c === numbers_d);
```

```
▶ []
▶ (4) [empty × 4]
▶ (4) ['one', 'two', 'three', 'four']
▶ (4) ['one', 'two', 'three', 'four']
false
```

3-2. 자바스크립트의 내장 객체

Array 객체의 length 프로퍼티 사용하기

배열 요소는 프로그램 안에서 얼마든지 추가하고 삭제할 수 있으므로 요소의 개수를 알고 사용하는 것이 중요합니다. 그렇다면 배열 요소의 개수는 어떻게 할 수 있을까요? 바로 Array 객체에 있는 length 프로퍼티를 사용하면 됩니다. 이 프로퍼티에는 배열 요소의 개수가 저장되어 있습니다.

다음과 같이 마침표(.)를 사용해 length 프로퍼티를 작성해 보겠습니다.

프로퍼티 length 활용하기

JS

```js
var numbers = ['one', 'two', 'three', 'four']; // 배열 선언

for (i = 0; i < numbers.length; i++) {
  document.write(`<p> ${numbers[i]} <p>`);
}
```

one

two

three

four

Array 객체의 메서드

종류	설명
concat	기존 배열에 요소를 추가해 새로운 배열을 만듭니다.
every	배열의 모든 요소가 주어진 함수에 대해 참이면 true를 반환하고 그렇지 않으면 false를 반환합니다.
filter	배열 요소 중에서 주어진 필터링 함수에 대해 true인 요소만 골라 새로운 배열을 만듭니다.
forEach	배열의 모든 요소에 대해 주어진 함수를 사용합니다.
indexOf	주어진 값과 일치하는 값이 있는 배열 요소의 첫 인덱스를 찾습니다.
join	배열 요소를 문자열로 합칩니다. 이때 구분자를 지정할 수 있습니다.
push	배열의 맨 끝에 새로운 요소를 추가한 후 length를 반환합니다.
unshift	배열의 시작 부분에 새로운 요소를 추가합니다.
pop	배열의 마지막 요소를 꺼내 그 값을 결과로 반환합니다.
shift	배열에서 첫 번째 요소를 꺼내 그 값을 결과로 반환합니다.
splice	배열에 요소를 추가하거나 삭제합니다.
slice	배열에서 특정한 부분만 잘라냅니다.
reverse	배열의 배치 순서를 역순으로 바꿉니다.
sort	배열 요소를 지정한 조건에 따라 정렬합니다.
toString	배열에서 지정한 부분을 문자열로 반환합니다. 이때 각 요소는 쉼표(,)로 구분합니다.

3-2. 자바스크립트의 내장 객체

배열끼리 합치는 concat() 메서드

concat() 메서드는 **서로 다른 배열 2개를 합쳐서 새로운 배열**을 만들어 줍니다. **어느 배열을 먼저 쓰는가**에 따라 기준이 달라지고, 결과 배열의 순서도 달라집니다.

다음은 **nums와 chars라는 배열 2개를 합치는 예제**입니다. 둘 중 **어느 배열을 기준**으로 하느냐에 따라 **결과 배열이 달라집니다.**

배열 2개를 합쳐서 새로운 배열 만들기

JS

```js
var nums = [1,2,3];
var chars = ['a','b','c'];

var nums_chars = nums.concat(chars);
var chars_nums = chars.concat(nums);

console.log(`nums.concat(chars)는 ${nums_chars} 입니다.`);
console.log(`chars.concat(nums)는 ${chars_nums} 입니다.`);
```

```
"nums.concat(chars)는 1,2,3,a,b,c 입니다."
"chars.concat(nums)는 a,b,c,1,2,3 입니다."
```

3-2. 자바스크립트의 내장 객체

배열 안의 요소끼리 합치는 join() 메서드

join() 메서드는 배열 요소를 연결해서 하나의 문자열로 만들어 줍니다. 이때 각 요소 사이에 원하는 구분자('/')를 넣을 수도 있는데, 따로 지정하지 않으면 **요소를 쉼표(,)로 구분**합니다.다음 예제는 배열 nums의 요소를 구분자 없이 연결해서 문자열로 만드는 예제입니다.

join()는 배열안의 요소끼리 합쳐줍니다.

JS

```js
var nums = [1,2,3];
var chars = ['a','b','c'];

var string1 = nums.join();
console.log(`nums.join()는 ${string1} 입니다.`);

var string2 = chars.join('/');
console.log(`chars.join('/')는 ${string2} 입니다.`);
```

```
"nums.join()는 1,2,3 입니다."
"chars.join('/')는 a/b/c 입니다."
```

3-2. 자바스크립트의 내장 객체

새로운 요소를 추가하는 push(), unshift() 메서드

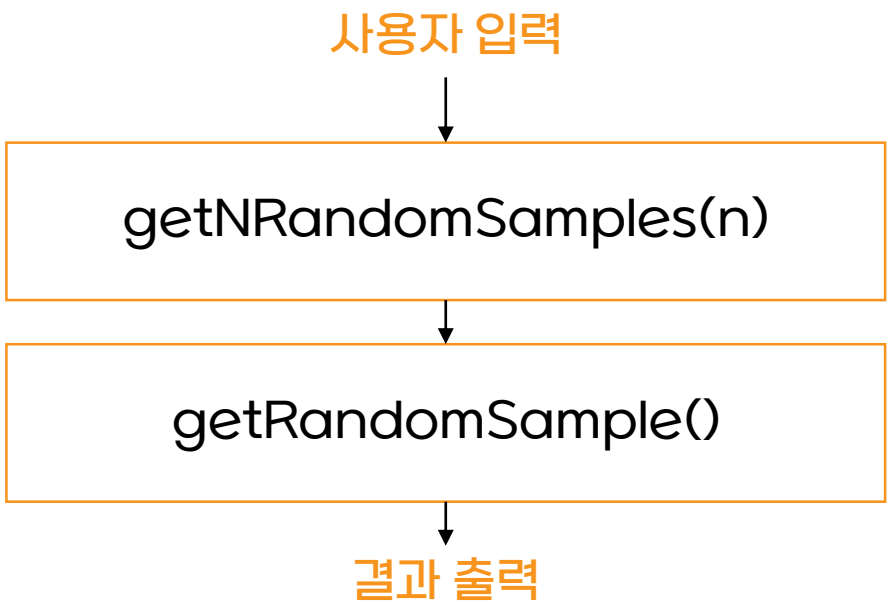
배열에 새로운 요소를 추가하려면 push()나 unshift() 메서드를 사용합니다. 배열의 맨 끝에 요소를 추가하려면 push() 메서드를, 배열의 맨 앞에 요소를 추가하려면 unshift()메서드를 사용합니다.

이러한 push()는 인공지능 프로그램을 만드는데 있어서도 종종 활용되고는 하니 한 가지 간단한 예제를 살펴보겠습니다.

push()를 활용해 XOR GATE를 생성하는 함수(https://codepen.io/AI_R_IM/pen/abLdgpQ?editors=0011)

무작위로 생성된 데이터입니다.

```
▼ (2) [Array(10), Array(10)]
  ▼ 0: Array(10)
    ▶ 0: (2) [-0.19534601052024758, 0.797833805053545]
    ▶ 1: (2) [-0.6214650417019705, 0.8766835093119751]
    ▶ 2: (2) [-0.16508277263326399, 0.8761225741372787]
    ▶ 3: (2) [-0.7457564524916949, 0.2706552613943969]
    ▶ 4: (2) [-0.5119465027562198, -0.5707295740202389]
    ▶ 5: (2) [-0.3381008243167565, -0.18630432111844497]
    ▶ 6: (2) [0.24254289603245693, -0.7489276313578546]
    ▶ 7: (2) [0.8444588959678727, 0.17203172500509245]
    ▶ 8: (2) [-0.5562152239732865, -0.1417878241577335]
    ▶ 9: (2) [-0.6735664635319067, 0.2981745919523848]
      length: 10
    ▶ [[Prototype]]: Array(0)
  ▶ 1: (10) [1, 1, 1, 1, 0, 0, 1, 0, 0, 1]
    length: 2
  ▶ [[Prototype]]: Array(0)
```



3-2. 자바스크립트의 내장 객체

JS

```js
// getRandomSample 함수는 하나의 데이터를 생성하는 예제입니다.
function getRandomSample() {
  // 독립변수 x, 종속변수 y 생성
  let x, y
  // x의 값은 [x1, x2] 이렇게 생성
  // x의 값은 -1 ~ 1 사이의 값을 생성하도록 합니다.
  x = [Math.random() * 2 - 1, Math.random() * 2 - 1]

  // xor에 맞게 변수를 생성
  // x1  x2  y
  // 0   0   0
  // 1   0   1
  // 0   1   1
  // 1   1   0

  // 만약 x1(x[0]), x2(x[1])의 값이 둘다 양수 혹은 둘다 음수라면
  if ((x[0] > 0 && x[1] > 0) || (x[0] < 0 && x[1] < 0 )){
    y = 0 // y 값은 0 이 됩니다.
  } else {
    y = 1 // 위 조건이 아니라면 즉,
          // 한쪽이라도 값의 부호가 다르다면 y값은 1이 됩니다.
  }
  // 그리고 x값과 y값을 반환합니다.
  return [x, y]
}
```

JS

```js
// getNRandomSamples 함수는 다수의 무작위 값을 생성합니다.
function getNRandomSamples(n) {
  // 다수의 데이터를 받을 변수를 let 예약어로 생성합니다.
  let xs = [] // 이 변수에는 사용자가 원하는 수 만큼의 데이터를 저장할 것입니다.
  let ys = []

  // 반복문 구조를 활용하여 사용자가 함수에 입력한 n값 만큼 반복
  for (let iter = 0; iter < n; iter++){
    let x, y // 하나 생성된 변수를 받기위한 변수 선언
    // 상단에서 정의한 XOR GATE 구조에 맞게 독립변수와 종속변수를 생성하는
    // 함수를 불러와줍니다.그리고 함수의 반환값을 [x,y]에 저장합니다.
    [x,y] = getRandomSample()

    // push()를 활용해 생성된 변수x,y를 각각 xs, ys의
    // 맨 끝에 저장합니다.
    // 만약 맨 앞에 저장하고자 한다면 unshift()를 사용합니다.
    xs.push(x)
    ys.push(y)
  }
  // 반복을 모두 마쳤다면 return을 통해 반환시켜줍니다.
  return [xs, ys]
}

xor_data = getNRandomSamples(10);
console.log(`무작위로 생성된 데이터입니다.`);
console.log(xor_data);
```

3-2. 자바스크립트의 내장 객체

날짜 계산하는 프로그램 만들기

Date 객체의 getTime() 메서드를 사용해 날짜를 계산하는 프로그램을 만들어 보겠습니다. 이 프로그램은 공부하기 시작한 날짜를 지정하고, 현재까지 며칠 동안 공부하였는지 계산하는 역할을 수행합니다.

날짜 계산하는 프로그램 만들기 (https://codepen.io/AI_R_IM/pen/oNGbrmg)

3-2. 자바스크립트의 내장 객체

HTML

```html
<body>
  <div id="container">
    <h1>GJ AI School ProJ</h1>
    <p><span class="accent" id="result"></span>일 동안 <br> 연속으로 공부 하였습니다.</p>
    <p>-그대의 하루를 위하여-</p>
  </div>
</body>
```

JS

```js
// 사용자에게 오늘의 날짜와 공부를 시작한 날짜를 받습니다.
var today = prompt("오늘 날짜를 입력하여 주세요.예) 2022-01-01", "2021-12-07")
var start = prompt("시작 날짜를 입력하여 주세요.예) 2022-01-01", "2021-12-08")

var now = new Date(today);            // 오늘 날짜를 객체로 지정
var firstDay = new Date(start);       // 시작 날짜를 객체로 지정

var toNow = now.getTime();          // 오늘까지 지난 시간(밀리 초)
var toFirst = firstDay.getTime();   // 첫날까지 지난 시간(밀리 초)
var passedTime = toNow - toFirst;   // 첫날부터 오늘까지 지난 시간(밀리 초)

passedTime = Math.round(passedTime/(1000*60*60*24));  // 밀리 초를 일 수로 계산하고 반올림

document.querySelector('#result').innerText = passedTime;
```

CSS

```css
body {
  background-color : #222; /* 전체 배경색  */
}
#container{
  margin:50px auto;   /* 중앙 정렬 */
  width:300px;        /* 너비 및 높이  */
  height:300px;
  border-radius:50%;      /* 경계선의 기울기 설정 (동그랗게) */
  border:2px double #222; /* 선의 두께 선의 종류 선의 색상  */
  background-color: rgb(247, 148, 30); /* 원의 배경색  */
  text-align: center;     /* 글자의 중앙정렬  */
}
h1 {
  margin-top:80px;
}
.accent {
  font-size:1.8em;
  font-weight:bold;
  color:red;
}
```

3-2. 자바스크립트 브라우저와 관련된 객체

시간 출력 프로그램 만들기

시간과 관련된 함수를 사용하여 현재 시간을 보여주는 프로그램을 작성하여 봅시다.

결괏값

현재 시각 오후 8:39:08

시간 출력 프로그램 만들기 (https://codepen.io/AI_R_IM/pen/KKXzdXX)

HTML

```html
<body>
  <!-- 전체 공간을 di container로 묶어줍니다.  -->
  <div id="container">
    <!-- 간단하게 현재 시각을 나타내는 문구를 생성합니다. -->
    <!-- 그리고 실제 시간이 표시되는 공간은 span 태그를 사용하여 줍니다.  -->
    <p>현재 시각 <span id="current" class="display"></span></p>
  </div>
</body>
```

JS

```js
// 일정한 시간 간격으로 작업을 수행하기 위한 setInterval 함수 사용
setInterval(displayNow, 1000);  // 1초(1000ms = 1s)마다 시간 계산 함수를 실행합니다.

function displayNow() {     // 시간 계산 함수

  var now = new Date();     // Date 객체의 인스턴스를 만듦. 현재시간을 담아줍니다.
  var currentTime = now.toLocaleTimeString();     //  toLocaleTmeString() 메서드를 사용해 해당 지역에 맞는 시간을 가져옴
  // 문서에서 id가 current인 것을 찾습니다. 그리고 currentTime변수의 값을 문서에 삽입하여 줍니다.
  document.querySelector("#current").innerHTML = currentTime;   // id="current" 인 요소에 현재 시간 표시
}
```

CSS

```css
/* 전체 설정 */
* {
  margin:0;
  padding:0;
  overflow:hidden;
}
/* 해당 내용을 문서 중앙에 배치 */
#container{
  display:flex;
  justify-content:center;
  align-items:center;
  min-height:100vh;
}
p {
  font-size:1.2em;
}
.display {
  font-size:1.5em;
  font-weight:bold;
  color:orange;
}
```

4-1. 자바스크립트의 문서 객체 모델

문서 객체 모델(DOM)이란

웹에서 **자바스크립트를 사용하는 이유**는 어떤 조건에 맞거나 사용자의 동작이 있을 때 웹 문서 전체 또는 일부분이 동적으로 반응하게 하는 것입니다. 이렇게 반응하게 하려면 웹 문서의 모든 요소를 따로 제어할 수 있어야 합니다.

예를 들어 웹 문서에 텍스트와 이미지가 들어 있다면 웹 브라우저는 마크업 정보를 보면서 텍스트 단락이 몇 개이고 그 내용이 무엇인지 살펴봅니다. 또한 이미지가 몇 개이고 이미지 파일 경로는 어떠한지 대체 텍스트는 무엇인지도 파악해서 이미지별로 정리해서 인식합니다.

그리고 이러한 텍스트와 이미지 요소를 브라우저가 제어하려면 두 요소를 따로 구별하여 인식해야 합니다. 이보다 더 많은 요소를 사용한 웹 문서라면 요소 사이의 포함 관계도 알아야 하고요.

이러한 모든 정보 요소를 자바스크립트로 가져와 프로그래밍 할 때 사용합니다. 이때 알아야 할 개념이 있는데 바로 문서 객체 모델(DOM: document object model)입니다.

DOM은 자바스크립트를 이용하여 웹 문서에 접근하고 제어할 수 있도록 객체를 사용해 웹 문서를 체계적으로 정리하는 방법

4-1. 자바스크립트의 문서 객체 모델

DOM은 웹 문서를 하나의 객체로 정의합니다. 그리고 웹 문서를 이루는 텍스트나 이미지, 표 등의 모든 요소도 각각 객체로 정의합니다. 예를 들어 웹 문서 전체는 document 객체이고, 삽입한 이미지는 image 객체 입니다.

이처럼 DOM은 웹 문서 문서와 그 안의 모든 요소를 객체로 인식하고 처리합니다.

앞에서 배웠던 브라우저 객체와 마찬가지로 웹 문서 객체도 다양한 프로퍼티와 메서드가 있습니다. 우선 문서 객체 모델의 구조를 나타내는 DOM 트리를 알아보겠습니다.

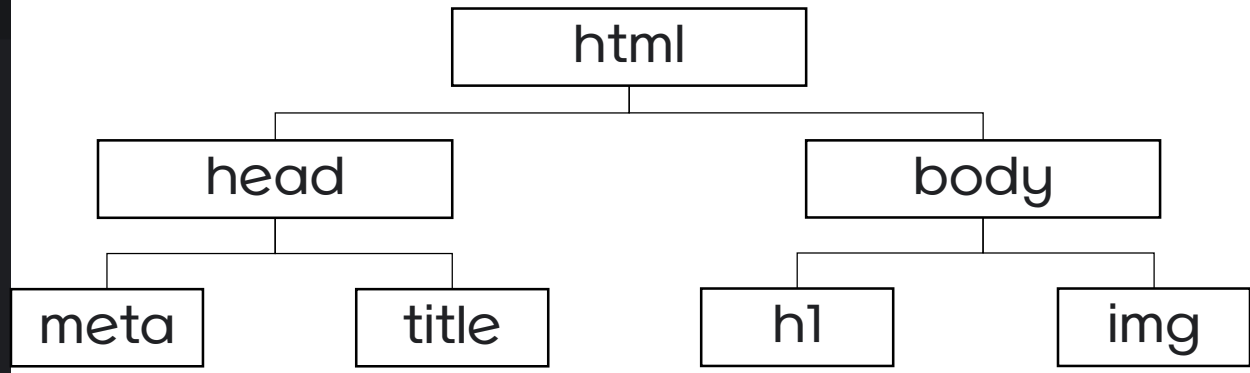
DOM 트리

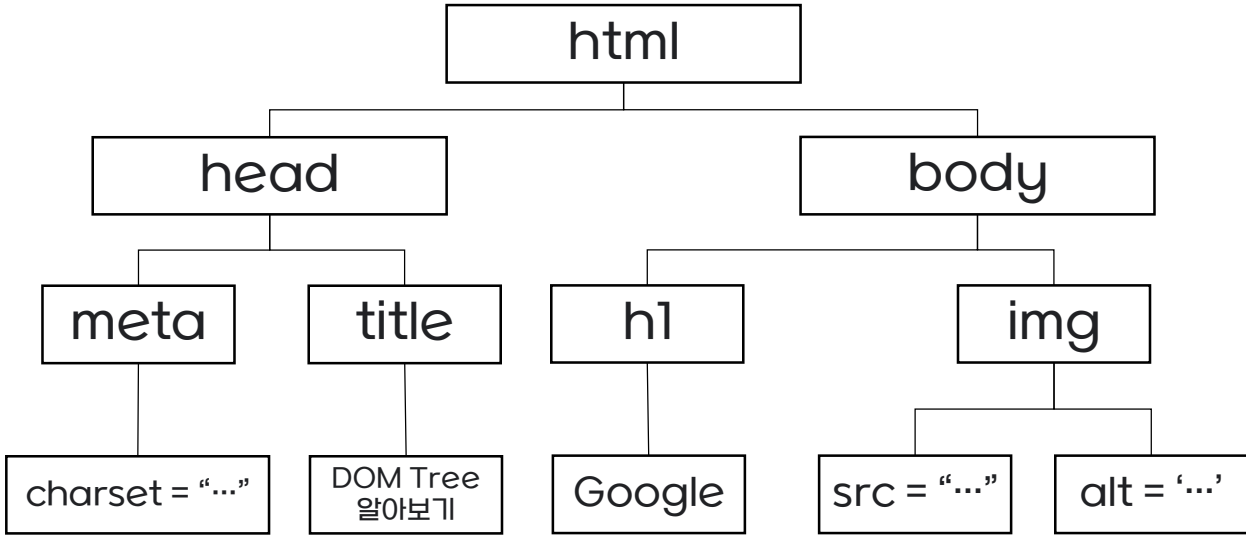
자바스크립트로 DOM을 조작하려면 실제 웹 문서가 DOM으로 어떻게 표현되는지 알아야 합니다. DOM은 웹 문서의 요소를 부모 요소와 자식 요소로 구분합니다. 예를들어 다음과 같은 간단한 소스를 생각해 봅시다.

4-1. 자바스크립트의 문서 객체 모델

HTML

```html
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <title>DOM Tree 알아보기</title>
</head>
<body>
  <h1>Google</h1>
  <img src="https://www.google.co.kr/images/srpr/logo11w.png" alt="">
</body>
</html>
```





이 예제에서 html 요소는 head, body 의 부모 요소이고, 다시 body 요소는 h1, img의 부모 요소가 됩니다.

한 단계 더 발전하여 DOM의 관점에서 부모 자식 관계를 생각해 보겠습니다. DOM은 문서안의 요소뿐만 아니라 각 요소에서 사용한 내용과 속성도 자식으로 나타냅니다.

다음 그림을 보면 h1 요소의 내용인 'google'은 h1의 자식이 되고 src, alt 속성은 img 요소의 자식이 되는 것이죠.

4-1. 자바스크립트의 문서 객체 모델

이렇게 부모와 자식 구조로 표시하면 마치 나무 형태가 되므로 DOM 트리 라고 합니다. DOM 트리에서 가지가 갈라져 나간 항목을 노드(node)라고 하며, DOM 트리의 시작 부분인 html 는 나무 뿌리에 해당한다 해서 루트 노드 라고 합니다.

루트 노드를 시작으로 웹 문서에서 사용한 요소는 계층 구조를 이룹니다. 따라서 각 노드 사이의 관계를 부모와 자식, 형제 간으로 표현할 수 있습니다.

이렇게 웹 문서를 해석할 DOM 구조를 만드는 것이 중요합니다. DOM 트리를 머릿속으로 그릴 수 있어야 자바스크립트를 사용해서 객체에 접근해서 원하는 부분을 수정할 수 있기 때문입니다.

4-2. DOM 요소에 접근하고 속성 가져오기

DOM 에 접근하기

CSS에서는 class, id, 태그 등의 스타일을 각각 구별하여 정의해야 합니다. 이때 class, id, 태그 등을 선택자(selector)라고 합니다. 이 선택자를 사용해서 DOM에 접근하는 방법을 알아보겠습니다.

id 선택자로 접근하는 getElementById()메서드

기본형 `요소명.getElementById("id명")`

HTML 태그의 id 속성은 HTML 요소가 문서 안에서 중복되지 않도록 사용하는 CSS 선택자입니다. 다음과 같이 getElementById() 메서드를 이용하면 특정한 id가 포함된 DOM 요소에 접근할 수 있습니다.

예를 들어 id값이 heading인 요소에 접근하려면 다음과 같이 사용합니다.

id 선택자로 DOM 요소에 접근하기

```
document.getElementById("heading")
```

class 값으로 접근하는 getElementsByClassName() 메서드

기본형 `요소명.getElementsByClassName("class명")`

getElementsByClassName() 메서드는 지정한 class 선택자 이름이 들어 있는 DOM 요소에 접근합니다.

class 선택자는 웹 문서 안의 **여러 요소에서 사용**할 수 있으므로 getElementsByClassName() 메서드를 사용하면 반환하는 요소가 2개 이상일 수 있습니다. 그래서 이름에 Element에 s를 붙인 것입니다. 예를 들어 웹 문서에서 class = 'bright' **속성이 있는 요소를 모두 찾으려면** 다음과 같이 사용합니다.

class 선택자로 DOM 요소에 접근하기

```
document.getElementsByClassName('bright')
```

4-2. DOM 요소에 접근하고 속성 가져오기

태그 이름으로 접근하는 getElementsByTagName() 메서드

class나 id를 지정하지 않은 DOM 요소에 접근하려면 **태그**를 이용합니다. getElementsByTagName() 메서드는 다음과 같이 **지정한 태그명을 사용한 DOM요소에 접근**할 수 있습니다.

기본형	요소명.getElementsByTagName("태그명")

예를 들어 웹 문서에서 <p> 태그를 사용한 모든 요소에 접근하려면 다음과 같이 사용합니다.

태그로 DOM 요소에 접근하기

```
document.getElementsByTagName('p')
```

다양한 방법으로 접근하는 querySelector(), querySelectorAll() 메서드

DOM 트리의 텍스트, 속성 노드까지 자유롭게 제어하려면 querySelector(), querySelectorAll() 메서드를 사용해야 합니다.

기본형	노드.querySelector(선택자) 노드.querySelectorAll(선택자 또는 태그)

id 선택자 처럼 반환값이 하나라면 querySelector() 메서드를 사용하고, **class 선택자나 태그 이름을 사용하여 여러 값이 한꺼번에 반환된 경우**에는 querySelectorAll() 메서드를 사용합니다.

querySelector(), querySelectorAll() 메서드에서 선택자를 표시할 때 id 이름앞에는 해시기호(#)를 붙이고, class 이름 앞에는 마침표(.)를 붙입니다. 태그는 기호 없이 태그명만 쓰면 됩니다. **querySelector() 메서드에서 class 이름으로 접근할 때는 class 이름을 사용한 여러 요소 중에서 첫 번째 요소만 반환**합니다.

다양한 방법으로 DOM 요소에 접근

```
document.querySelector('#heading') // id가 'heading' 인 요소
document.querySelector('.bright') //  class가 'bright' 인 요소
```

4-2. DOM 요소에 접근하고 속성 가져오기

웹 요소의 내용을 수정하는 innerText, innerHTML 프로퍼티

자바스크립트에서는 **웹 요소의 내용도 수정**할 수 있습니다. 가장 쉬운 **방법은 innerText 프로퍼티나 innerHTML 프로퍼티를 이용하는 것** 입니다. 이름만 봐도 알 수 있듯이, **innerText 프로퍼티는 텍스트 내용을 표시**하고, innerHTML 프로퍼티는 **HTML 태그 까지 반영하여 표시**합니다.

다음은 Date 객체를 사용해서 현재 날짜와 시간을 나타내는 예제입니다. 날짜와 시간을 그대로 **innerText 프로퍼티로 표시할 때와 innerHTML 프로퍼티로 태그와 같이 표시할 때 어떠한 차이가 있는지** 살펴봅시다.

결괏값

4-2. DOM 요소에 접근하고 속성 가져오기

https://codepen.io/AI_R_IM/pen/xxXVVKg?editors=1010

HTML

```html
<body>
  <button onclick="inntext()">innerText로 표시하기</button>
  <button onclick="innhtml()">innerHTML로 표시하기</button>
  <h1>현재 시각: </h1>
  <!-- id current로 묶어주겠습니다. -->
  <div id="current"></div>

  <script>
    // 현재 시간을 담는 변수 now를 생성합니다.
    var now = new Date();

    // inntext() 함수를 생성하겠습니다.
    function inntext(){
      // 문서상에 id가 current인 것을 불러와 현재 시간을 innerText로 작성하여 줍니다.
      document.getElementById("current").innerText = now ;
      // document.getElementById("current").innerText = <em> + now + <em> 코드를 작성해도
      // 출력되는 결과는 <em> 태그를 적용시키지 않습니다.
    }
    //innhtml() 함수를 생성합니다.
    function innhtml() {
      // 문서상 id가 current인 것을 불러와 현재 시간을 innerHTML로 작성하여 줍니다.
      // innerHTML은 <em> 태그까지 한꺼번에 출력하여 줍니다.
      document.getElementById("current").innerHTML = "<em>" + now + "</em>";
    }
  </script>
</body>
```

결과 화면

innerText로 표시하기 innerHTML로 표시하기

현재 시각:

Tue Dec 07 2021 22:44:11 GMT+0900 (한국 표준시)

innerText로 표시하기 innerHTML로 표시하기

현재 시각:

Tue Dec 07 2021 22:44:11 GMT+0900 (한국 표준시)

4-2. DOM 요소에 접근하고 속성 가져오기

속성을 가져오거나 수정하는 getAttribute(), setAttribute() 메서드

웹 요소를 문서에 삽입할 때 태그 속성을 함께 사용하면 **DOM 트리에 속성 노드가 추가되면서 속성값이 저장**됩니다. 이때 **속성에 접근하려면 getAttribute() 메서드**를 사용하고, 접근한 **속성의 값을 바꾸려면 setAttribute()**를 사용합니다.

다음은 id = 'cup' 인 이미지의 src 속성값을 가져오는 예제입니다. 큰 이미지를 클릭하면 그 이미지의 경로 속성을 가져와 알림 창에 표시합니다.

https://codepen.io/AI_R_IM/pen/poWyVxB

4-3. DOM 에서 이벤트 처리하기

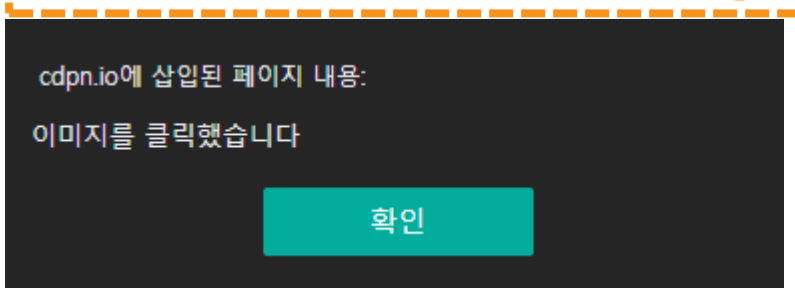
DOM 요소에 함수 직접 연결하기

이벤트 처리기 함수가 간단하다면 **DOM 요소에 직접 연결**할 수 있습니다. 예를 들어 **다음 예제는 img 요소를 가져와 변수에 저장한 후 onclick 을 사용해 변수에 이벤트 처리기(함수)를 직접 연결**합니다.

이미지를 클릭하면 알림창이 나타나지만 **이미지 외의 다른 곳을 클릭하면 실행되지 않습니다.**

https://codepen.io/AI_R_IM/pen/poWyVxB

이벤트 발생 X



HTML

```html
<img src="https://images.unsplash.com/photo-1637425827802-3540ebb274d2?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODkyNzM3NQ&ixlib=rb-1.2.1&q=85" width = "100%" id="cup">
```

JS

```js
// 문서상에서 id가 cup인 것을 찾아 가져옵니다.
var cup = document.querySelector("#cup");
// cup변수에 클릭 함수가 동작되면
cup.onclick = function(){ // 익명함수
  // 알람 메시지를 출력하도록 합니다.
  alert("이미지를 클릭했습니다");
}
```

4-3. DOM 에서 이벤트 처리하기

함수 이름을 사용해 연결하기

이벤트가 발생했을 때 실행할 함수를 따로 정의해 놓았다면 함수 이름을 사용해 연결할 수 있습니다. 이 경우에는 함수 이름 다음에 괄호 (( ))를 추가하지 않습니다.

다음 예제는 이미지를 변수화 한 후 click 이벤트가 발생했을 때 changePic() 함수를 실행합니다.

https://codepen.io/AI_R_IM/pen/eYGZKdJ?editors=1010

클릭

HTML

```html
<body>
  <div id="container">
    <img src="https://images.unsplash.com/photo-1636333329210-0006f2cb24cb?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODkyODA4Nw&ixlib=rb-1.2.1&q=85" width = "100%" id="cup">
  </div>
</body>
```

JS

```js
var cup = document.querySelector("#cup");  // id = cup인 요소를 가져옴

cup.onclick = changePic;  // cup을 클릭하면 changePic 함수 실행
// changePic() 함수의 역할은
function changePic() {
  // cup의 src를 다음 이미지로 값을 저장합니다.
  cup.src = "https://images.unsplash.com/photo-1637857778981-bf9bf903aa4c?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODkyODA4Nw&ixlib=rb-1.2.1&q=85";
}
```

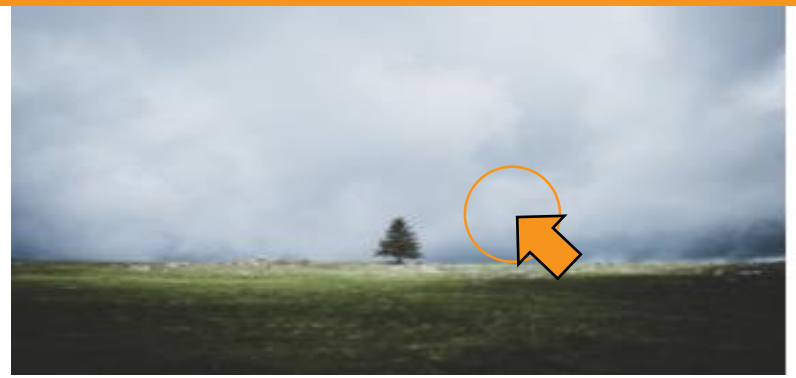
4-3. DOM 에서 이벤트 처리하기

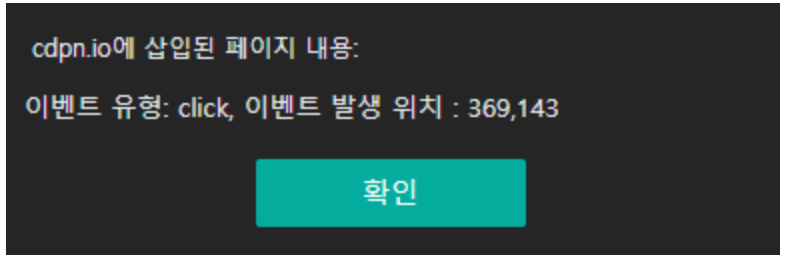
DOM의 event 객체 알아보기

DOM에는 이벤트 정보를 저장하는 event 객체가 있습니다. 이 객체에는 웹 문서에서 이벤트가 발생한 요소가 무엇인지, 어떤 이벤트가 발생했는지 등의 정보가 들어있습니다.

다음은 이미지에 발생한 이벤트의 유형과 위치를 알림 창으로 보여 주는 예제입니다. 여기에서 img 요소를 클릭하면 그 위치를 확인하기 위해 event 객체를 함수의 인수로 사용합니다.

https://codepen.io/AI_R_IM/pen/yLzOEbN?editors=1010





HTML

```html
<body>
  <div id="container">
    <img src="https://images.unsplash.com/photo-1635705276767-2a15aaf7f5de?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODkyODY3Mw&ixlib=rb-1.2.1&q=85"
        width = "100%" id="cup">
  </div>
</body>
```

JS

```js
var cup = document.querySelector("#cup");  // id = cup인 요소를 가져옴
// cup 변수에 onclick가 발생하면 함수 동작
cup.onclick = function(event) {
  // event의 타입 확인, 페이지의 x축 y축 위치 확인
  alert("이벤트 유형: " + event.type + ", 이벤트 발생 위치 : " + event.pageX + "," + event.pageY);
}
```

event 객체의 주요 프로퍼티와 메서드는 다음과 같습니다.

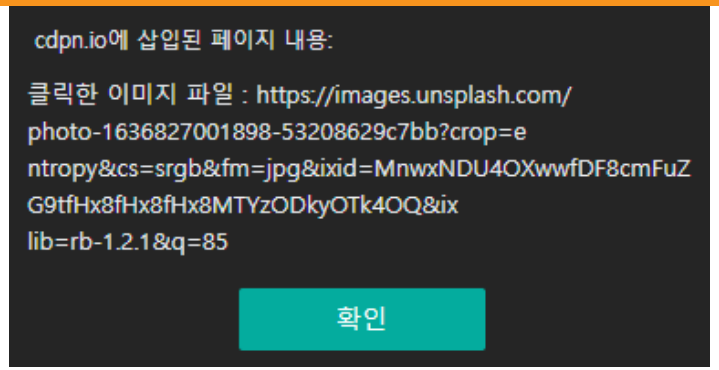
구분		설명
프로퍼티	altKey	이벤트가 발생할 때 [alt]를 눌렀는지 여부를 boolean값으로 반환합니다.
	button	마우스에서 누른 버튼의 키값을 반환합니다.
	charCode	Keypress 이벤트가 발생할 때 어떤 키를 눌렀는지 유니코드 값으로 반환합니다.
	clientX	이벤트가 발생한 가로 위치를 반환합니다.
	clientY	이벤트가 발생한 세로 위치를 반환합니다.
	ctrlKey	이벤트가 발생했을 때 Ctrl를 눌렀는지 여부를 boolean값으로 반환합니다.
	pageX	현재 문서 기준으로 이벤트가 발생한 가로 위치를 반환합니다.
	pageY	현재 문서 기준으로 이벤트가 발생한 세로 위치를 반환합니다.
	screenX	현재 화면 기준으로 이벤트가 발생한 가로 위치를 반환합니다.
	screenY	현재 화면 기준으로 이벤트가 발생한 세로 위치를 반환합니다.
	shiftKey	이벤트가 발생할 때 Shift 를 눌렀는지 여부를 Boolean 값으로 반환합니다.
	target	이벤트가 최초로 발생한 대상을 반환합니다.
	timeStamp	이벤트가 발생한 시간을 반환합니다.
	type	발생한 이벤트 이름을 반환합니다.

4-3. DOM 에서 이벤트 처리하기

event 객체에는 이벤트 정보만 들어있습니다. 만약 **이벤트가 발생한 대상에 접근**하려면 **이벤트 처리기에서 예약어 this를 사용**해야 합니다.

다음 예제처럼 클릭한 **이미지의 파일 경로를 알고 싶다면 this.src를 사용**합니다.

https://codepen.io/AI_R_IM/pen/bGopKMX



HTML

```html
<body>
  <div id="container">
    <img src="https://images.unsplash.com/photo-1636827001898-53208629c7bb?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODkyOTk4OQ&ixlib=rb-1.2.1&q=85" width = "50%" id="cup">
  </div>
</body>
```

JS

```js
// cup이라는 id를 찾아 변수화합니다.
var cup = document.getElementById("cup");
// 변수에서 클릭 이벤트가 발생하면
cup.onclick = function(event) {
  // 발생한 이벤트를 this로 처리할 수 있습니다.
  alert("클릭한 이미지 파일 : " + this.src);
}
```

4-3. DOM 에서 이벤트 처리하기

addEventListener() 메서드 사용하기

지금까지 살펴본 이벤트 처리기는 한 요소에 하나의 **이벤트 처리기만 연결**할 수 있습니다. 하지만 다음과 같이 addEventListner() 메서드와 event 객체를 사용하면 **한 요소에 여러 이벤트 처리기를 연결해 실행**할 수 있습니다. **addEventListner() 메서드**는 끝에 세미콜론(;)을 꼭 붙여야 합니다.

기본형

```
요소.addEventListener(이벤트, 함수, 캡처 여부);
```

①이벤트 : **이벤트 유형을 지정**합니다. click과 keypress 처럼 **on 을 붙이지 않고 씁니다.**
②함수 : 이벤트가 발생하면 **실행할 명령이나 함수를 지정**합니다. 여기에서 **함수를 정의할 때는 event 객체를 인수**로 받습니다.
③캡처 여부 : **이벤트를 캡처하는지 여부를 지정**하며 기본값은 false이고 true와 false 중에서 선택할 수 있습니다. true 이면 캡처링, **false이면 버블링을 한다는 의미**입니다. 이벤트 캡처링은 DOM의 부모 노드에서 자식 노드로 전달되는 것이고, **이벤트 버블링은 DOM의 자식 노드에서 부모 노드로 전달**되는 것입니다.

다음은 이미지 위로 마우스 포인터를 올려놓으면 다른 이미지로 바뀌었다가 내려놓으면 다시 원래 이미지로 돌아오는 예제입니다.

4-3. DOM 에서 이벤트 처리하기

https://codepen.io/AI_R_IM/pen/abLNrrE

HTML

```html
<body>
  <div id="container">
    <img src="https://images.unsplash.com/photo-1637604591036-9127bdf2d553?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODk2NjY3NA&ixlib=rb-1.2.1&q=85" width = "100%" id="cover">
  </div>
</body>
```

JS

```js
// id가 cover 인 것을 변수 cover로 지정합니다.
var cover = document.getElementById("cover");
// addEventListener() 메서드에 mouseover 이벤트 즉 마우스를 올리면 changePic 함수 실행
cover.addEventListener("mouseover",changePic, false);
// addEventListener() 메서드에 mouseout 이벤트 즉 마우스를 내리면 originPic 함수 실행
cover.addEventListener("mouseout",originPic, false);

// 위에서 사용한 함수의 정의입니다.
function changePic() {
  // 함수가 동작하면 cover변수의 주소를 다음과 같은 이미지로 전환합니다.
  cover.src = "https://images.unsplash.com/photo-1637481108969-836f76676413?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODk2NjY3NA&ixlib=rb-1.2.1&q=85";
}
// 원본으로 돌려주는 함수입니다.
function originPic() {
  // 함수가 동작하면 cover 변수의 주소를 다음과 같은 원래 이미지로 전환합니다.
  cover.src = "https://images.unsplash.com/photo-1637604591036-9127bdf2d553?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODk2NjY3NA&ixlib=rb-1.2.1&q=85";
}
```

CSS

```css
#container {
  width:300px;
  margin:10px auto;
}
```

4-3. DOM 에서 이벤트 처리하기

이렇게 addEventListener() 메서드 안에 함수를 함께 선언하면 특정 이벤트에서 어떤 명령을 실행하는지 한눈에 **확인**할 수 있어서 더욱 편리합니다.

https://codepen.io/AI_R_IM/pen/ExwKBgE?editors=1010

HTML

```
<body>
  <div id="container">
    <img src="https://images.unsplash.com/photo-1637436221127-72bd9facc771?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODk2NzY2Nw&ixlib=rb-1.2.1&q=85" width = "100%" id="cover">
  </div>

  <script>
    // id가 cover인 것을 찾은 후 그것을 변수var cover로 저장합니다.
    var cover = document.getElementById("cover");
    // cover변수에 addEventListener()함수를 적용하는데, 이벤트(mouseover) 그리고 그 이벤트가 진행되면 수행될 함수를
    // 익명함수로 정의합니다.
    cover.addEventListener("mouseover",function() {
      cover.src = "https://images.unsplash.com/photo-1635696478172-d9a0f4215ac4?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODk2NzcyMQ&ixlib=rb-1.2.1&q=85";
    });

    // 같은 방식입니다. 메서드 안에서 함수 선언(익명함수)하는 과정입니다.
    cover.addEventListener("mouseout", function() {
      cover.src = "https://images.unsplash.com/photo-1637436221127-72bd9facc771?crop=entropy&cs=srgb&fm=jpg&ixid=MnwxNDU4OXwwfDF8cmFuZG9tfHx8fHx8fHx8MTYzODk2NzY2Nw&ixlib=rb-1.2.1&q=85";
    });
  </script>
</body>
```

4-4. DOM 에서 노드 추가 삭제하기

웹 문서에서 **처음 화면에는 내용이 보이지 않다가 클릭이나 이벤트가 발생하면 내용이 나타나는 경우**가 있습니다. 이러한 동작은 css의 display 속성을 사용해서 만들 수도 있지만 **DOM 트리에 새로운 노드를 추가하는 방법**도 있습니다. 이때 주의할 점은, **노드를 추가하면 단순히 요소 노드 뿐 아니라 텍스트와 속성 노드도 함께 추가**해야 한다는 것입니다.

노드 리스트란

DOM 에서 **새로운 노드를 만들어 추가하거나 삭제하려면 노드 리스트(node list)를 사용**해야 합니다. 그렇다면 노드 리스트란 무엇일까요?

DOM에 접근할 때 **querySelectorAll()메서드를 사용하면 노드를 한꺼번에 여러 개 가져올 수** 있습니다. 이때 노드 정보를 여러 개 저장한 것이 노드 리스트 입니다. 배열과 비슷하게 동작하죠. 다음 예제와 함께 노드 리스트를 살펴보겠습니다.

4-4. DOM 에서 노드 추가 삭제하기

https://codepen.io/AI_R_IM/pen/KKXzOwo?editors=1010

Web Programming

HTML

CSS

Javascript

```html
<body>
  <h1>Web Programming</h1>
  <ul id="itemList">
    <li>HTML</li>
    <li>CSS</li>
    <li>Javascript</li>
  </ul>
</body>
```

```
document.querySelectorAll('li');
▼NodeList(3) [li, li, li]
  ▶0: li
  ▶1: li
  ▶2: li
   length: 3
  ▶[[Prototype]]: NodeList
```

```
document.querySelectorAll('li')[1];
  <li>CSS</li>
```

위 코드는 **웹 브라우저에서 현재 문서에서 li 요소를 모두 가져오라는 뜻** 입니다. 결괏값으로 NodeList(3) [li, li, li] **라고 표시**됩니다. li 요소의 형태가 노드 리스트인 것을 알 수 있습니다. 이처럼 노드 리스트란 노드를 여러 개 저장한 데이터 형태를 말합니다. 콘솔창에서 NodeList 왼쪽에 있는 ▶ 를 클릭해 노드 리스트의 내용을 확인해 보면 밑으로 인덱스와 요솟값이 저장되고 length 속성에서 몇 개의 노드가 저장되었는지 표시됩니다.

배열은 아니지만 배열과 아주 비슷한 형태인 것을 알 수 있습니다.

노트 리스트는 배열과 비슷해서 인덱스 번호로 특정 위치의 노드에 접근할 수 있습니다. 다음과 같이 입력하면 li 노드 리스트 중에서 두 번째 노드를 가져올 수 있습니다.

4-4. DOM 에서 노드 추가 삭제하기

새로운 노드 추가 과정

DOM 트리를 구성하는 기본 원칙을 다시 한번 살펴보겠습니다.

1. 모든 HTML 태그는 **요소(element)노드** 입니다.
2. HTML 태그에서 사용하는 **텍스트 내용**은 **자식 노드인 텍스트(text) 노드**입니다.
3. HTML 태그에 있는 속성은 자식 노드인 속성(attribute) 노드 입니다.
4. 주석은 주석(comment)노드입니다.

이 원칙에 따라 어떠한 태그를 노드로 추가한다면 단순히 태그에 해당하는 요소 노드뿐만 아니라 텍스트 노드와 속성 노드도 추가해야 합니다. 예를 들어 <img> 태그를 요소로 추가한다면 어떤 노드를 더 추가해야 할까요? <img> 태그의 요소 노드와 속성에 해당하는 src, alt의 노드도 추가됩니다. 이 밖에도 사용하는 속성에 따라 더 많은 노드가 추가 되겠죠.

어떠한 웹 문서의 DOM 트리에 새로운 노드를 추가하려면 그 문서에 새로운 소스코드를 작성해야 합니다. 그래서 새 노드를 추가할 때는 웹 문서에 어떤 소스를 추가할지 먼저 생각하고, 그에 따라 요소 노드나 텍스트, 속성 노드 등을 만들어야 합니다.

텍스트 노드를 사용하는 새로운 요소 추가하기

여기에서는 **[더 보기] 링크를 클릭**하면 **그 아래 간단한 텍스트가 표시**되는 스크립트 소스를 작성해 보겠습니다. 문서에는 단순히 <p> 태그 하나가 추가되어 보이지만 DOM에서는 여러 단계가 필요하다는 것을 알 수 있습니다. 다음 단계를 따라 새로운 요소를 추가하는 방법을 살펴보고, 마지막으로 전체 코드를 완성해서 보겠습니다.

4-4. DOM 에서 노드 추가 삭제하기

1. 요소 노드 만들기 – createElement() 메서드

DOM에 새로운 요소를 추가할 때 가장 먼저 할 일은 **요소 노드를 만드는 것**입니다. 이때 사용하는 메서드는 **createElement()인데 괄호 안에 해당하는 요소 노드를 만들면** 됩니다. 예를들어 **새로운 <p>요소**를 만든다면 다음과 같이 작성합니다.

JS

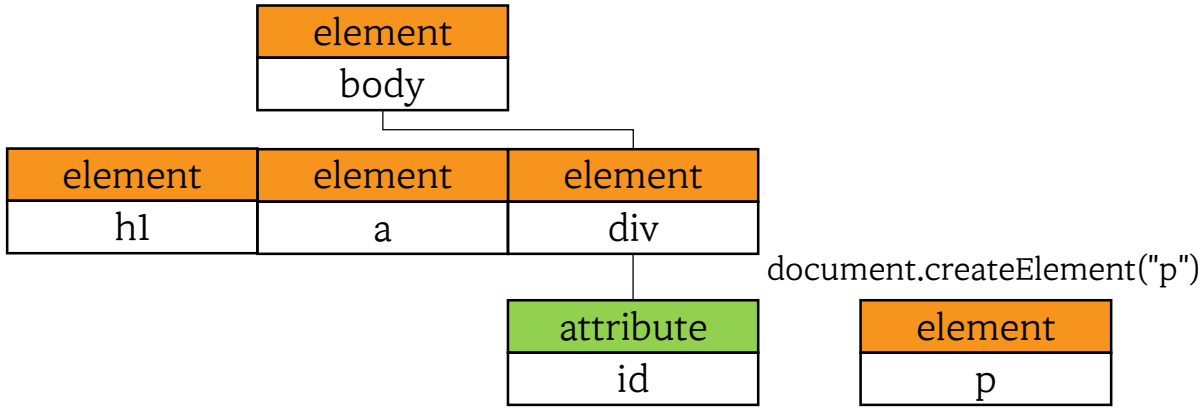
```js
var newP = document.createElement('p');
```

하지만 createElement() 메서드는 **새로운 노드를 만들 뿐** 아직 웹 문서에 새로운 노드를 추가한 것은 아닙니다.

https://codepen.io/AI_R_IM/pen/JjrXgeX?editors=1000

HTML

```html
<body>
  <div id="container">
    <h1>DOM을 공부합시다</h1>
    <!-- 클릭하면(onclick) addP 함수 실행(addP());
    클릭하면 한번만 실행을 위해 (this.onclick='';) 코드 추가 -->
    <a href="#" onclick="addP(); this.onclick='';">더 보기</a>
    <div id="info"></div>
  </div>
</body>
```



이제 <p> 태그의 내용에 해당하는 **텍스트 노드도 만들어야 합니다.**

<p> 노드를 추가한 DOM 트리의 모습은 다음과 같습니다.

4-4. DOM 에서 노드 추가 삭제하기

2. 텍스트 노드 만들기 – createTextNode() 메서드

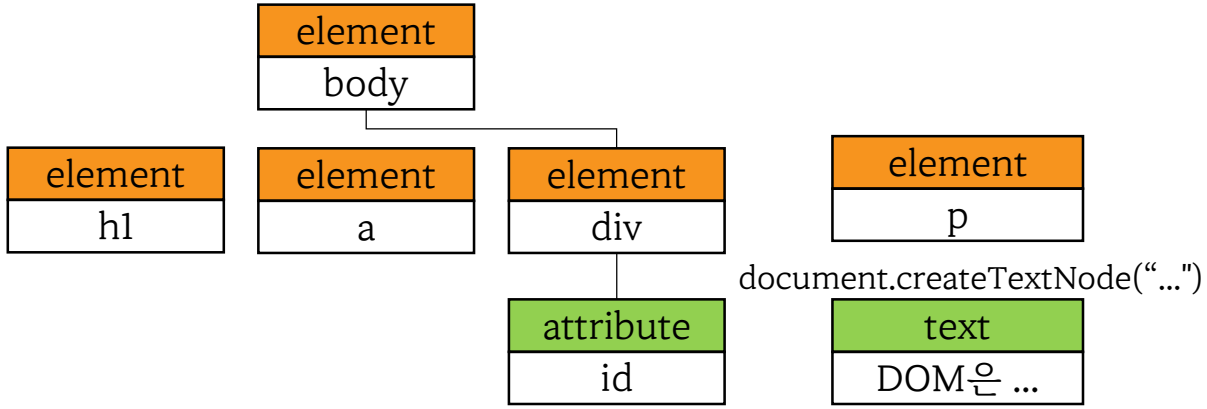
새로운 요소 노드를 만들었다면 그 다음은 **내용을 담는 텍스트 노드를 자식 노드로 만들어 연결**해야 합니다. 텍스트 노드를 만드는 메서드는 **createTextNode() 이고 다음과 같은 형식으로 사용**합니다.

기본형	`document.createTextNode(텍스트);`

새롭게 만든 <p> 요소에 들어갈 내용이 있는 텍스트 노드는 txtNode 변수로 저장합니다. 그리고 이것을 DOM트리에 추가하면 다음과 같이 나타낼 수 있습니다.

JS

```js
function addP() {
  var newP = document.createElement("p");
  var txtNode = document.createTextNode("DOM은 Document Object Model의 줄임말입니다.");
  //newP.appendChild(txtNode);
  //document.getElementById("info").appendChild(newP);
}
```



4-4. DOM 에서 노드 추가 삭제하기

3. 자식 노드 연결하기 – appendChild() 메서드

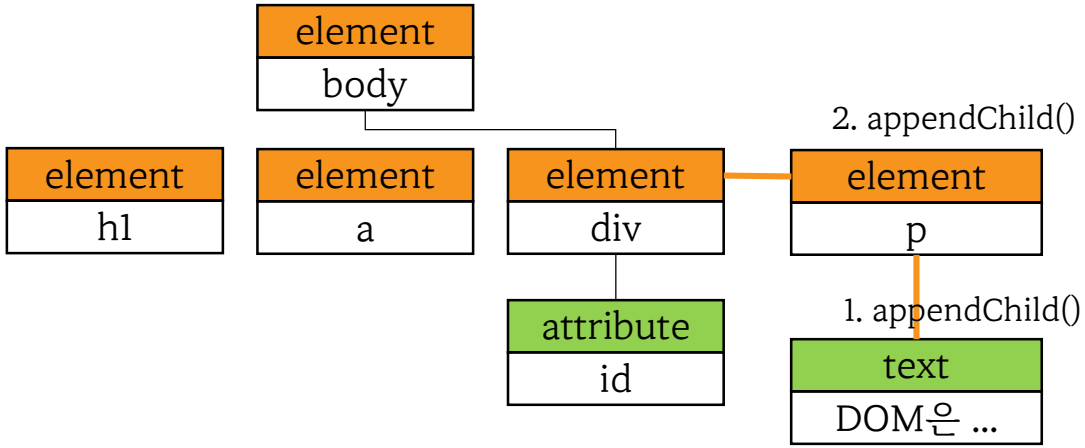
지금까지 새로운 p 노드(newP)와 사용할 텍스트 노드(txtNode)를 만들었습니다. **아직까지는 노드를 만든 상태이며 부모 노드와 자식 노드로 연결되지 않았습니다.**

appendChild() 메서드는 텍스트 노드나 속성 노드를 만든 후 **요소 노드에 연결할 때 사용**합니다. 이때 appendChild() 메서드를 사용해서 연결하는 노드는 자식 노드 중 맨 끝에 추가됩니다.

기본형 부모노드.appendChild(자식노드)

다음은 **텍스트 노드 txtNode**를 **요소 노드 newP의 자식 노드로 연결**하고, newP 노드를 **다시 의 자식 노드로 연결**합니다.

```js
function addP() {
  var newP = document.createElement("p");
  var txtNode = document.createTextNode("DOM은 Document Object Model의 줄임말입니다.");
  // 부모노드 newP에 자식노드 txtNode를 연결합니다.
  newP.appendChild(txtNode);
  // 그리고 다시 newP 노드를 <div id="info"></div>에 연결
  document.getElementById("info").appendChild(newP);
}
```



4-4. DOM 에서 노드 추가 삭제하기

4. 전체 소스 코드 완성하기

지금까지 요소를 생성하고 서로 연결하는 방법을 알아 보았으니 **전체 소스코드를 작성**해보겠습니다.

다음 예제를 보면 [더보기] 링크를 클릭하는 **부분에 addP() 함수와 this.onclick=" 코드가 추가**되었습니다. 이것은 **addP() 함수가 한 번만 실행**되도록 하기 위한 것입니다.

https://codepen.io/AI_R_IM/pen/JjrXgeX?editors=1000

HTML

```html
<body>
  <div id="container">
    <h1>DOM을 공부합시다</h1>
    <!-- 클릭하면(onclick) addP 함수 실행(addP());
    클릭하면 한번만 실행을 위해 (this.onclick='';) 코드 추가 -->
    <a href="#" onclick="addP(); this.onclick='';">더 보기</a>
    <div id="info"></div>
  </div>
</body>
```

결괏값

DOM을 공부합시다

더 보기

DOM을 공부합시다

더 보기

DOM은 Document Object Model의 줄임말입니다.

4-4. DOM 에서 노드 추가 삭제하기

속성값이 있는 새로운 요소 추가하기

HTML 태그에서는 **여러 가지 속성을 사용해서 웹 요소를 제어**합니다.

따라서 **새로운 요소를 만들면 그와 관련된 속성 노드도 만들어서 자식 노드로 연결**해야 합니다.

앞의 예제에 이어서 **'더보기' 링크를 클릭하면** 그 아래에 텍스트는 물론 **이미지까지 표시되는 스크립트 소스를 추가로 작성**해 보겠습니다.

웹 문서에서 새 이미지를 보여주려면 **img 요소 노드를 추가한 후 src 속성 노드를 만들어 자식 노드로 연결**해야 합니다.

https://codepen.io/AI_R_IM/pen/xxXOxzQ?editors=0010

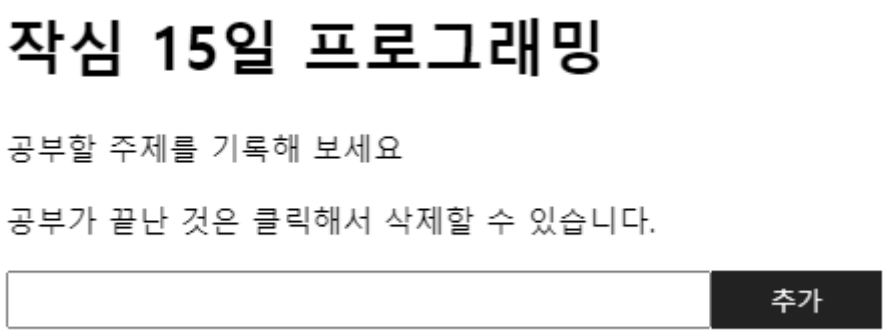
```
function addContents() {
  var newP = document.createElement("p");
  var txtNode = document.createTextNode("DOM은 Document Object Model의 줄임말입니다.");
  newP.appendChild(txtNode);

  // 1. createElement img 요소 노드를 생성하여 줍니다.
  var newImg = document.createElement("img");
  // 2. img 요소에는 src 속성이 필요하기에 이것도 변수화 하여 줍니다.
  var srcNode = document.createAttribute("src");
  // 3. img 요소에는 alt 속성이 필요하기에 이것도 변수화 하여 줍니다.
  var altNode = document.createAttribute("alt");
  // srcNode의 노드 에는 이미지 링크가 필요하기에 이를 value 속성을 통해 다음과 같이 저장하여 줍니다.
  srcNode.value = "http://drive.google.com/uc?export=view&id=1iO1o-StSqPF8KzUCfHTmXEf9WXh_DZVh";
  // 마찬가지로 alt에 들어갈 값도 저장하여 줍니다.
  altNode.value = "돔 트리 예제 이미지";
  // 속성 노드를 연결하여 주는 setAttributeNode() 입니다.
  // 여기까지는 속성노드 끼리만 연결한 것입니다.
  // srcNode(자식노드)를 newImg(부모노드)에 연결하여 줍니다.
  newImg.setAttributeNode(srcNode);
  // altNode(자식노드)를 newImg(부모노드)에 연결하여 줍니다.
  newImg.setAttributeNode(altNode);

  // 이제 본격적으로 웹 문서의 DOM에 추가해야 합니다.
  // 이를 위해 appendChild()를 사용하여 자식 노드, 부모 노드 연결을 수행하여 줍니다.
  document.getElementById("info").appendChild(newP);
  document.getElementById("info").appendChild(newImg);
}
```

4-4. DOM 에서 노드 추가 삭제하기

텍스트 필드에 입력한 값을 화면에 표시하기

텍스트 필드에 입력한 내용을 화면에 표시하려면 단순히 **document.write() 문으로 작성**할 수 있습니다. 하지만 이제는 DOM을 공부했으니, **텍스트 필드에 입력한 내용을 새로운 노드로 만들어서 화면에 표시하는 방법**을 연습해 보겠습니다.

https://codepen.io/AI_R_IM/pen/gOGMOVw



4-4. DOM 에서 노드 추가 삭제하기

작심 15일 프로그래밍

공부할 주제를 기록해 보세요

공부가 끝난 것은 클릭해서 삭제할 수 있습니다.

추가

HTML

```html
<body>
  <!-- 전체를 id=container 으로 묶어주도록 하겠습니다. -->
  <div id="container">
    <!--  메인 제목 및 설명을 다음과 같이 출력하여 줍니다.    -->
    <h1>작심 15일 프로그래밍</h1>
    <p>공부할 주제를 기록해 보세요</p>
    <!-- form action=""은 입력한 값을 처리하는 서버에게 전달합니다.
    지금은 서버를 다루지 않으므로 단순히 빈 칸으로 두겠습니다. -->
    <form action="">
      <!-- input 태그는 말 그대로 입력을 받습니다. autofocus는 시작하자마자 입력받을 수 있도록 합니다. -->
      <input type="text" id="subject" autofocus>
      <!-- 버튼을 클릭하면 newRegister()가 수행되며 값을 반환합니다. -->
      <button onclick="newRegister(); return false;">추가</button>
    </form>
    <hr>
    <!-- 사용자가 입력한 값이 출력되어지는 리스트를 생성합니다.
  이후에는 <li> 를 통해 값이 보여질 것입니다.  -->
    <ul id="itemList"></ul>
  </div>
</body>
```

4-4. DOM 에서 노드 추가 삭제하기

JS

```js
function newRegister() { // 버튼을 클릭하면 동작하는 함수입니다.
      // 필요한 변수를 먼저 생성합니다.
      // li 요소 노드 추가. <ul> 하단에 배치되어 리스트가 될 예정입니다.
      var newItem = document.createElement("li");
      // 입력 폼(subject)의 노드를 변수화 하여 줍니다.
      var subject = document.querySelector("#subject");
      // 입력 폼(subject)에서 입력된 값(value)을 createTextNode()를 통해 노드화 합니다.
      var newText = document.createTextNode(subject.value);
      // 노드화 된 newText를 자식노드로, 리스트가 될 newItem를 부모노드로 연결합니다.
      newItem.appendChild(newText);

      // <li>와 연결할 <ul> 태그의 id(#itemList)를 가져와 변수화 합니다.
      var itemList = document.querySelector("#itemList");
      // itemList가 기준이 되며 사용자가 입력한 노드(newItem)를 insertBefore()를 통해
      //첫 번째 노드 앞에 추가합니다. 즉 마지막에 입력한 값을 맨 위로 올려줍니다.
      itemList.insertBefore(newItem, itemList.childNodes[0]);

      // 새로운 값을 받기 위해 입력 폼의 값을 비워둡니다(지우기).
      subject.value="";

      //querySelectorAll('li')를 통해 모든 리스트를 가져와 items 변수화 합니다
      var items = document.querySelectorAll("li");  // 모든 항목 가져오기

      // 입력한 값 만큼 생성된 리스트 개수만큼 반복문을 돌려주는데,
      for(i=0; i<items.length; i++) {
        // 사용자가 입력한 것(items[i])을 클릭(click)하는 이벤트가 발생하면 addEventListener()
        items[i].addEventListener("click", function() {  // 익명 함수 실행
          // 만약 활성화(this)된 parentNode(부모노드)가 있다면
          if(this.parentNode)
            // 활성화된 부모 노드의 자식 노드를 지워줍니다.(removeChild)
            this.parentNode.removeChild(this);  // 부모 노드에서 삭제
        });
      }
    }
```

노드 삭제하기

노드 삭제하기

지금까지 새로운 노드를 추가하는 방법으로 createElement()와 appendChild() 메서드를 살펴보았습니다. 이번에는 DOM 트리에서 특정 노드를 삭제하는 방법을 살펴보겠습니다.

노드를 삭제할 때 기억해 둘 것은 부모 노드에서 자식 노드를 삭제해야 한다는 것입니다. 즉, 삭제해야 할 노드가 있다면 반드시 부모 노드를 먼저 찾아야 합니다. 그래서 노드를 삭제하는 메서드 외의 부모 노드를 찾는 프로퍼티가 필요합니다.

parentNode 프로퍼티

DOM트리의 노드는 바로 삭제할 수 없으므로 먼저 부모노드에 접근해야 합니다.

기본형	노드.parentNode

그러려면 삭제할 노드의 부모 노드를 찾아가는 방법을 알아야 합니다.

parentNode 프로퍼티는 현재 노드의 부모 노드에 접근해서 부모 노드의 요소 노드를 반환합니다.

아래와 같이 입력한 결괏값을 살펴보면 **li노드의 부모 노드인 ul 노드가 출력**되는 것을 확인할 수 있습니다.

```
// 모든 li 중에 [1] 번째의 부모노드(parentNode)
document.querySelectorAll("li")[1].parentNode
▶<ul id="itemList">…</ul>
```

4-4. DOM 에서 노드 추가 삭제하기

JS

```js
function newRegister() { // 버튼을 클릭하면 동작하는 함수입니다.
    // 필요한 변수를 먼저 생성합니다.
    // li 요소 노드 추가. <ul> 하단에 배치되어 리스트가 될 예정입니다.
    var newItem = document.createElement("li");
    // 입력 폼(subject)의 노드를 변수화 하여 줍니다.
    var subject = document.querySelector("#subject");
    // 입력 폼(subject)에서 입력된 값(value)을 createTextNode()를 통해 노드화 합니다.
    var newText = document.createTextNode(subject.value);
    // 노드화 된 newText를 자식노드로, 리스트가 될 newItem를 부모노드로 연결합니다.
    newItem.appendChild(newText);

    // <li>와 연결할 <ul> 태그의 id(#itemList)를 가져와 변수화 합니다.
    var itemList = document.querySelector("#itemList");
    // itemList가 기준이 되며 사용자가 입력한 노드(newItem)를 insertBefore()를 통해
    //첫 번째 노드 앞에 추가합니다. 즉 마지막에 입력한 값을 맨 위로 올려줍니다.
    itemList.insertBefore(newItem, itemList.childNodes[0]);

    // 새로운 값을 받기 위해 입력 폼의 값을 비워둡니다(지우기).
    subject.value="";

    //querySelectorAll('li')를 통해 모든 리스트를 가져와 items 변수화 합니다
    var items = document.querySelectorAll("li");  // 모든 항목 가져오기

    // 입력한 값 만큼 생성된 리스트 개수만큼 반복문을 돌려주는데,
    for(i=0; i<items.length; i++) {
      // 사용자가 입력한 것(items[i])을 클릭(click)하는 이벤트가 발생하면 addEventListener()
      items[i].addEventListener("click", function() {  // 익명 함수 실행
        // 만약 활성화(this)된 parentNode(부모노드)가 있다면
        if(this.parentNode)
          // 활성화된 부모 노드의 자식 노드를 지워줍니다.(removeChild)
          this.parentNode.removeChild(this);  // 부모 노드에서 삭제
      });
    }
  }
```

노드 삭제하기

removeChild() 메서드

기본형 부모노드.removeChild(자식노드)

removeChild() 메서드는 이름에서도 알 수 있듯이 자식 노드를 삭제하는 역할을 합니다. 부모 노드만 찾는다면 removeChild() 메서드를 사용하는 것은 간단합니다.

예를 들어 **li 노드를 삭제하려면 li의 부모 노드에서 삭제**해야 하므로 다음과 같이 작성하면 됩니다.