5-1 함수의 다양한 활용방법

자바스크립트에서 함수는 자료이므로 변수에 할당할 수 있고, 함수를 함수의 매개 변수로 전달 해서 활용할 수 있습니다. 이번에는 함수를 매개변수로 전달하는 특성을 자세하게 살펴보겠습니다.

다른 프로그래밍 언어는 함수를 지정된 위치에서 만들어야 하지만, 자바스크립트는 '**함수도 하나의 자료**'라는 개념을 가지고 있어서 **중간에 만들 수 있습니다.**
자바스크립트로 개발을 하게 되면 **느린 동작**과 **변수와 함수의 너무나 강력한 유연성 때문에 제대로 활용이 어려웠습니다.** 하지만 자바스크립트 속도는 **크롬 웹 브라우저가 등장한 이후에 급속도로 빨라졌고, 비동기 프로그래밍 등장으로 문법적 가치를 인정받아 지금까지 사용**되고 있습니다.

콜백 함수

자바스크립트 함수도 하나의 자료형이므로 **매개변수로 전달**할 수 있습니다. 이렇게 **매개변수로 전달하는 함수는 콜백** callback 함수라고 합니다. 다른 프로그래밍 언어에서 찾아보기 힘든 개념이므로 처음 접한다면 어렵게 느껴질 수 있습니다. 코드를 살펴보며 콜백 함수에 대하여 살펴보도록 하겠습니다.

5-1 함수의 다양한 활용방법

선언적 함수를 활용한 콜백 함수 및 익명함수를 활용한 콜백 함수

JS

```js
// callThreeTimes() 함수는 함수를 매개변수로 받아 해당 함수를 3번 호출합니다.
function callThreeTimes(callback_func) {
  for (let i = 0; i < 3; i ++){
    callback_func(i) // callback_func 이라는 매개변수는 함수이므로 호출 가능
  }
}

// print()를 생성합니다. 매개변수로 전달된 값 만큼 콘솔 출력합니다.
function print(i) {
  console.log(`${i}번째 함수 호출`);
}

// callThreeTimes를 호출합니다. 그리고 매개변수를 print()로 전달합니다.
callThreeTimes(print)
// 그럼 callThreeTimes()가 동작하고 그 내부에 있는 print()가 동작하게 됩니다.
// 이때 반복문 구조가 callThreeTimes()에 들어가 있기에 print()가 반복적으로 동작하게 됩니다.
```

JS

```js
function callThreeTimes(callback){
  for (let i = 0; i < 3; i++){
    callback(i)
  }
}

// 익명함수를 사용하여 결과 출력
callThreeTimes(function (i){
  console.log(`${i}번째 함수 호출`)
})
```

이전 예제의 선언적 함수를 익명 함수로 변경한다면 다음과 같이 코드를 구성할 수 있습니다.

5-1 함수의 다양한 활용방법

콜백 함수를 활용하는 forEach()

콜백 함수를 활용하는 가장 기본적인 함수는 forEach() 메소드 입니다. forEach() 메소드는 배열이 갖고 있는 함수(메소드)로써 단순하게 배열 내부의 요소를 사용해서 콜백 함수를 호출해줍니다.

배열이 갖고 있는 메소드 중에서 콜백 함수를 활용하는 메소드는 다음과 같은 형태의 콜백 함수를 사용합니다.

콜백 함수를 활용하는 forEach()

기본형 `function(value, index, array){}`

JS

```js
const numbers = [273, 52, 103, 32, 67];

// 매개변수로 value, index, array를 갖는 콜백함수를 사용합니다.
numbers.forEach(function (value, index, array){
    // numbers의 각각의 값을 반복적으로 호출합니다.
    console.log(`${index}번째 요소 : ${value}`)
})
```

5-1 함수의 다양한 활용방법

콜백 함수를 활용하는 함수 : map()

map() 메소드도 배열이 갖고 있는 함수 입니다. map() 메소드는 콜백 함수에서 리턴한 값들을 기반으로 새로운 배열을 만드는 함수입니다. 다음 코드에서는 콜백 함수에서 value * value를 하고 있어 모든 배열의 요소를 제곱한 새로운 배열을 만듭니다.

JS

```js
// 배열을 선언 합니다.
let numbers = [273, 52, 103, 42, 123]

// numbers 변수에 map()를 정의합니다.
numbers = numbers.map(function (value, index, array){
    // value * value 를 곱한 값을 리턴하게 되는데, 그 값을 numbers에 저장
    return value * value
})

// forEach() 함수에 console.log 자체를 넘겨서
// numbers의 값을 하나씩 반복적으로 출력하게 됩니다.
numbers.forEach(console.log);
```

JS

```js
// 배열을 선언 합니다.
let numbers = [273, 52, 103, 42, 123]

// 배열의 모든 값을 제곱
numbers = numbers.map(function (value){ // 함수 내부에서 value만 사용하므로
                                        // value만 매개변수로 전달
    return value * value
})

numbers.forEach(console.log);
```

앞서 forEach(), map() 함수의 **완전한 형태**를 보여 드리기 위해 **콜백 함수에 매개 변수를 value, index, array로 3개를 모두 입력** 했지만, 일반적으로는 **value만 또는 value 와 index만 사용하는 경우**가 많습니다.

콜백 함수의 매개변수는 모두 입력할 필요 없고, 사용하고자 하는 위치의 것만 순서에 맞춰 입력하면 됩니다.

5-1 함수의 다양한 활용방법

콜백 함수를 사용하는 함수 : filter()

filter() 메소드도 **배열이 갖고 있는 함수**입니다.

filter() 메소드는 **콜백 함수에서 리턴하는 값이 true 인것들만 모아서 새로운 배열을 만드는 함수**입니다.

JS

```
// 데이터 생성
const numbers = [0,1,2,3,4,5]
// 함수가 리턴 하는 값 중에서 값이 true인 것들만 모아서 새로운 배열을 만드는 함수
const evenNumbers = numbers.filter(function (value){
    // 배열 numbers의 value들을 2와 나눈 나머지가 0인 것들 중 '참' 값만 배열에 저장.
    return value % 2 === 0
})

console.log(`원래 배열 : ${numbers}`)
console.log(`짝수만 추출 : ${evenNumbers}`)
```

화살표 함수

앞에서 살펴본 map(), filter() 함수 처럼 **단순한 형태의 콜백 함수를 쉽게 입력하고자 화살표 함수라는 함수** 생성 방법이 있습니다. 화살표 함수는 **function 키워드 대신 화살표(=>)를 사용**하며, 다음과 같은 형태로 생성하는 간단한 함수입니다.

JS

```
const array = [0,1,2,3,4,5,6,7,8,9];

console.log(
    // array에 접근하여 map()을 적용시켜 값을 하나하나 부릅니다.
    // 그리고 value 끼리 제곱을 하여 저장하도록 합니다.
    array.map((value) => value * value)
    );
```

5-1 함수의 다양한 활용방법

배열의 메소드와 화살표 함수

이 코드는 어떤 메소드가 리턴하는 값을 기반으로 해서 함수를 줄줄이 사용하는 방법을 보여줍니다. 이를 '메서드 체이닝' 이라고 부르기도 합니다.

JS

```js
let numbers = [0,1,2,3,4,5,6,7,8,9]

// 메소드 체이닝(method chaining)
// numbers배열에 짝수인 조건(value % 2 === 0)을 걸어 참인 값(filter)만 가져옵니다.
// [0,2,4,6,8]
numbers.filter((value) => value % 2 === 0).
// 위에서 얻은 값을 가지고 하나씩 제곱을 취합니다.
// [0, 4, 16, 36, 64]
map((value) => value * value).
// 그 값을 하나씩 console.log()로 출력하여 줍니다.
forEach((value) => {
  console.log(value)
})
```

5-1 함수의 다양한 활용방법

타이머 함수

자바스크립트에는 다음과 같이 **특정 시간마다 또는 특정 시간 이후에 콜백 함수를 호출할 수 있는 타이머 함수**들이 있습니다. 이 **함수를 사용하면 시간과 관련된 처리**를 할 수 있습니다.

타이머를 종료하고 싶을 때도 함께 살펴보도록 하겠습니다.

JS

```js
// setTimeout(함수, 시간)
setTimeout(() => {
    // 동작하는 함수를 정의
    console.log(`1초 후에 실행됩니다.`)
    // 함수가 몇 초 후에 동작하는지 알려줍니다. 1000ms == 1초
}, 1 * 1000);

// 카운트 변수 생성
let count = 0;

// setInterval(함수, 실행) - 특정 시간 마다 함수를 호출합니다.
setInterval(() => {
    // 호출될 함수를 정의합니다.
    console.log(`1초마다 실행됩니다.${count}번째`);
    // count변수의 값을 1씩 증가시켜 줍니다.
    count++
    // 1초씩 계속해서 수행합니다.
}, 1 * 1000)
```

5-1 함수의 다양한 활용방법

타이머를 종료하고 싶을 때는 clearTimeout() 함수와 clearInterval() 함수를 사용합니다.

지금까지 함수와 관련된 기본적인 내용을 모두 살펴보았습니다. 함수를 실제로 사용해보면 그 활용이 무궁무진하다는 것을 알 수 있습니다.

JS

```
let id // 함수의 결괏값을 담는 변수입니다.
let count = 0

// setInterval(함수, 시간) - 특정 시간마다 함수 호출
// 함수의 결괏값을 id에 저장합니다.
// 변수 id는 1초마다 수행되는 결괏값을 id에 저장합니다.
id = setInterval(() => {
    console.log(`1초 마다 실행됩니다.(${count}번째)`);
    count++
}, 1 * 1000) // 1000ms 마다 실행


// 특정 시간 후에 함수를 한 번 호출합니다.
// 5초 후에 수행됩니다.
setTimeout(() => {
    // 5초후에 동작하는 코드입니다.
    console.log(`타이머를 종료합니다.`);

    //clearInterval(타이머 id) 함수로 설정한 타이머를 제거합니다.
    //clearTimeout(타이머 id) 함수로 설정한 타이머를 제거합니다.

    // clearInterval(타이머 id) 함수로 설정한 타이머를 제거합니다.
    clearInterval(id);
}, 5 * 1000) //
```

5-1 함수의 다양한 활용방법

즉시 호출 함수

기본형 `(function () { })()`

여러 웹 사이트의 자바스크립트 코드를 보면 다음과 같이 **익명 함수를 생성**하고 곧바로 즉시 호출하는 패턴을 많이 볼 수 있습니다.
일반적으로 자바스크립트는 HTML 페이지 내부에서 사용할 때 script 태그를 여러 개 사용하고 코드를 입력합니다.
이렇게 코드가 여러 곳에서 사용되면 변**수 이름이 충돌할 가능성이 매우 높습니다**. 예를 들어 다음 코드를 살펴보도록 하죠.

상단의 코드블록만 동작을 수행하고 있습니다. 이렇게 변수가 존재하는 범위를 **스코프(scope)**라고 부릅니다. 이 스코프는 **같은 단계에 있을 경우 무조건 충돌**이 일어납니다. 자바스크립트에서 이러한 스코프 단계를 변경하는 방법은 **중괄호를 사용해서 블록**을 만들거나 **함수를 생성해서 블록을 만드는 방법**입니다.

HTML — 같은 스코프(scope)

```
<!-- AAA.js에서 가져온 자바스크립트 코드 -->
<script>
  let pi = 3.14;
  console.log(`파이 값은 ${pi} 입니다.`);
</script>

<!-- BBB.js에서 가져온 자바스크립트 코드 -->
<script>
  let pi = 3.141592;
  console.log(`파이 값은 ${pi} 입니다.`);
</script>
```

Console — Clear

```
"파이 값은 3.14 입니다."
Uncaught SyntaxError: Identifier 'pi' has already been declared
```

5-1 함수의 다양한 활용방법

블록과 함수 블록을 사용해 이름 충돌 문제 해결하기

이렇게 **블록 내부에서 같은 이름으로 변수를 선언**하면 변수가 외부 변수와 충돌하지 않고 외부 변수를 가립니다. 내부 블록에서는 내부 블록에서 선언한 변수만 볼 수 있습니다.

HTML

```
<script>
  let pi = 3.14; // 변수 선언
  // 블록을 사용한 스코프 생성
  {
    let pi = 3.141592; // 블록 안에서의 변수선언
    console.log(`파이 값은 ${pi}입니다.`);
  }
  console.log(`파이 값은 ${pi} 입니다.`);

  // 함수 블록을 사용한 스코프 생성
  function sample() {
    let pi = 3.141592;
    console.log(`파이 값은 ${pi}입니다.`);
  }
  sample(); // 함수 실행

  // 첫 번째 줄 에서 정의한 변수
  console.log(`파이 값은 ${pi}입니다.`)
</script>
```

5-1 함수의 다양한 활용방법

익명 함수와 선언적 함수 차이

while 반복문과 for 반복문은 2가지 모두 많이 사용되지만, 사용하는 상황이 조금씩 다릅니다. **while 반복문은 조건을 중심으로 반복**할 때, **for 반복문은 횟수를 중심으로 또는 배열 등을 중심으로 반복**합니다.
그런데 익명 함수와 선언적 함수는 사용하는 상황이 비슷합니다. **기본적으로는 혼자 개발할 때는 자신이 편하다고 생각하는 것을 사용**하고 다른 사람들과 함께 개발할 때는 모두가 편하다고 생각하는 것을 사용하면 됩니다.
다만 최근에는 안전 등의 이유로 **익명 함수를 선호하는 편**입니다.

익명 함수는 우리가 코드를 읽을 때와 같은 순서로 함수가 선언되지만, **선언적 함수는 우리가 코드를 읽는 순서와 다른 순서로 함수가 선언**됩니다. **함수를 같은 이름으로 덮어쓰는 것은 굉장히 위험한 일입니다.** 그래서 안전하게 사용할 수 있는 익명 함수를 더 선호

5-1 함수의 다양한 활용방법

익명 함수의 사용

익명 함수는 **순차적인 코드 실행에서 코드가 해당 줄을 읽을 때 생성**됩니다. 따르서 다음과 같은 코드가 있다면 **위에서 아래로 차례대로 코드가 실행**되면서 func_A**라는 변수**에 **'2번째 익명 함수입니다.'**를 **호출하는** 함수가 할당됩니다.

선언적 함수 호출

선언적 함수는 **순차적인 코드 실행이 일어나기 전에 생성**됩니다. 따라서 **선언적 함수는 같은 불록이라면 어디에서 함수를 호출**해도 상관없습니다 다음 코드와 같이 선언적 함수를 생성하기 전에 함수를 호출해도 함수가 이미 생성된 상태이므로 아무 문제 없이 실행됩니다.

또한 선언적 함수도 입력한 순서대로 생성되고 같은 이름이라면 덮어쓰므로 실행 결과 또한 '2번째 선언적 함수입니다.'를 출력합니다.

JS

```
// 변수를 선언합니다.
let func_A

// 익명 함수를 선언합니다.
func_A = function() {
    console.log(`1번째 익명 함수입니다.`);
}

// 익명 함수를 두 번 선언합니다.
func_A = function() {
    console.log(`2번째 익명 함수입니다.`);
}

func_A(); // 익명 함수를 호출하면 두 번째 선언된 함수가 호출됩니다.
```

JS

```
func_B() // 선언적 함수를 호출합니다.
         // (※ 선언적 함수를 생성하는 코드 앞에 입력)

// 선언적 함수를 2번 생성합니다.
function func_B () {
    console.log(`1번째 선언적 함수입니다.`);
}

function func_B () {
    console.log(`2번째 선언적 함수입니다.`);
}
```

6-1 자바스크립트에서 객체의 기본

객체의 기본

객체(object)란 **추상적 의미**로, 한 마디로 정의한다면 '**실제로 존재하는 무엇인가**'를 의미합니다. 그리고 이를 '**이름**'과 '**값**'으로 구성된 속성을 가진 **자바스크립트의 기본 데이터 타입**으로 이야기 할 수 있습니다. 이전에 살펴봤던 배열(array)도 객체라고 할 수 있죠.

객체

자바스크립트에서 여러 자료를 다룰 때는 **객체(object)를 사용**합니다. 이전에 살펴보았던 배열도 여러 자료를 다룰 수 있습니다. 그렇게 할 수 있던 이유는 **배열도 객체**이기 때문입니다.

배열을 typeof로 실행해 보면 **object 라는 문자열이 출력**됩니다. 이때 출력한 **object가 바로 객체**이죠.

배열은 다음과 같이 선언할 수 있습니다.

객체를 의미하는 object를 출력

```
> typeof([])
< 'object'
```

배열의 선언 및 인덱스를 활용한 접근

```
> const array = ['weight', 'bias', 'LearningRate', 'epochs'];
  console.log(array[0]);
  console.log(array[1]);

  weight

  bias
```

배열에는 **인덱스와 요소**가 있습니다. 요소를 사용하려면 위 처럼 **배열 이름 뒤에 인덱스로 접근**할 수 있습니다.

6-1 자바스크립트에서 객체의 기본

배열은 객체를 기반으로 만들어졌으므로 배열과 객체는 상당히 비슷합니다. 다른 점이 있다면 **배열은 요소에 접근할 때 인덱스를 사용**하지만, **객체는 key를 사용**합니다. 우선 객체를 만들어 서로 비교해가며 무엇이 비슷한지 살펴보겠습니다.

객체는 중괄호{...}로 생성하며, 다음과 같은 형태의 자료를 **쉼표(,)로 연결해서 입력**합니다.

```
> const ai_model = {
      model_layer : 'BI_LSTM, BI_LSTM, BI_LSTM',
      param : 2104,
      hidden_activation : 'elu',
      output_activation : 'softmax'
  }
```

'대괄호'를 사용한 접근

```
> console.log(ai_model['model_layer']);
  console.log(ai_model['param']);
  console.log(ai_model['hidden_activation']);
  console.log(ai_model['output_activation']);
  BI_LSTM, BI_LSTM, BI_LSTM
  2104
  elu
  softmax
```

'온점'를 사용한 접근

```
> console.log(ai_model.model_layer);
  console.log(ai_model.param);
  console.log(ai_model.hidden_activation);
  console.log(ai_model.output_activation);
  BI_LSTM, BI_LSTM, BI_LSTM
  2104
  elu
  softmax
```

6-1 자바스크립트에서 객체의 기본

속성과 메소드

배열 내부에 있는 값을 요소라고 합니다. 반면 객체 내부에 있는 값은 속성이라고 합니다. 배열의 요소와 마찬가지로 객체의 속성도 모든 형태의 자료형을 가질 수 있습니다.

```
> const object = {
      number : 0.0001,
      string : 'CNN_layer',
      boolean : true,
      array : [0.112, 0.332, 1.232],

      method : function() {}
  }
```

속성과 메소드 구분하기

객체의 속성 중 **함수 자료형인 속성을 특별히 메소드(method)**라고 부릅니다. 다음 코드에서 **객체 pet은 name 속성과 eat 속성**을 가지고 있는데, eat 속성처럼 입력값을 받아 무언가 한 다음 결과를 도출해 내는 함수 자료형을 특별히 **eat() 메소드** 라고 합니다.

```
> const pet = {
      name : '미유',
      eat : function(food){}
  }

  pet.eat()
```

6-1 자바스크립트에서 객체의 기본

메소드 내부에서 this 키워드 사용하기

메소드 내에서 **자기 자신이 가진 속성**을 출력하고 싶을 때는 **자신이 가진 속성임을 분명하게 표시**해야 합니다.

자기 자신이 가진 속성이라는 것을 표시할 때는 **this 키워드를 사용**합니다.

```
> const pet = {
      name : '미유',
      eat : function (food) {
          // this.name를 통해 자신이 가진 속성을 출력하고자 합니다.
          alert(this.name + '은/는' + food + '을/를 먹습니다.')
      }
  }

  pet.eat('츄르');
```

동적으로 객체 속성 추가 / 제거

객체를 처음 생성한 후에 **속성을 추가하거나 제거하는 것을 '동적으로 속성을 추가한다'** 또는 **'동적으로 속성을 제거한다'** 고 표현합니다.

다음과 같이 객체를 생성한 후 속성을 지정하고 값을 입력하면 됩니다.

우선 **json.stringify()를 사용**했는데, 이 함수는 잠시 후에 살펴보도록 하겠습니다. **일단 객체를 콘솔 출력에서 쉽게 볼 수 있는 방법**이라 기억하겠습니다.

```
> const student = {}
  student.name = 'lee'
  student.job  = 'developer'
  student.language = 'python'

  console.log(JSON.stringify(student, null, 2))
```

```
{
  "name": "lee",
  "job": "developer",
  "language": "python"
}
```

6-1 자바스크립트에서 객체의 기본

객체의 속성을 제거할 때는 delete 키워드를 사용합니다.

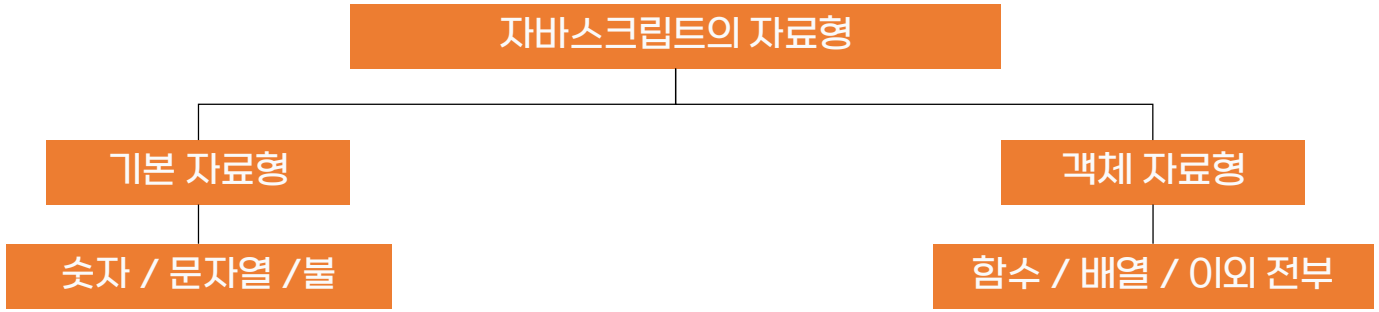
```
> const student = {}
  student.name = 'lee'
  student.job  = 'developer'
  student.language = 'python'

  delete student.language

  console.log(JSON.stringify(student, null, 2))
```

```
{
  "name": "lee",
  "job": "developer"
}
```

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

현대 프로그래밍 언어는 모두 객체 지향이라는 패러다임을 기반으로 합니다. 그래서 모든 개발이 객체로 이루어집니다. 이번에는 객체와 관련된 기본적인 문법들을 확실하게 익혀보도록 하겠습니다.

자바스크립트에서 사용하는 자료는 크게 **기본 자료형**과 **객체 자료형**으로 구분할 수 있으며, 다음 그림과 같이 나타낼 수 있습니다.



유연함의 대명사인 자바스크립트는 **기본 자료형이 객체 자료형이 될 수 도 있습니다.** 어떤 경우에 그렇게 되는지 알아보고, 이를 활용하는 **prototype 객체**를 알아보겠습니다.

그리고 자바스크립트의 기본적인 객체들이 갖고 있는 속성과 메소드를 살펴보겠습니다. 마지막으로 외부 자바스크립트를 읽어들이고 사용하는 방법을 알아보겠습니다.

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

객체 자료형

속성과 메소드를 가질 수 있는 **모든 것은 객체**입니다. 예를 들면 **배열도 객체**입니다. 다음과 같이 **a라는 이름의 배열을 선언**하고 **배열에 속성을 지정**한 후 확인해 보면 속성을 가질 수 있다는 것을 알 수 있습니다.

객체 자료형

```
> const a = [];
< undefined
```
빈 배열 선언

```
> a.sample = 10
< 10
```
배열에 속성 추가

```
> a.sample
< 10
```
배열값 확인

```
> a
< ▼ [sample: 10] i
    sample: 10
    length: 0
  ▶ [[Prototype]]: Array(0)
> Array.isArray(a)
< true
```
배열 여부 확인

```
> function b () {}
< undefined
> b.sample = 10
< 10
> b.sample
< 10
```

```
> b
< ƒ b () {}
```

함수도 객체 입니다. 다음과 같이 **함수 b를 선언**하고 **함수에 속성을 지정한 후 확인해 보면 함수가 속성을 가질 수 있다는 것**을 알 수 있습니다.

그래서 **typeof 연산자를 사용해서 배열의 자료형을 확인해보면 'object' 라고 객체가 출력**됩니다.

배열인지 확인하려면 **Array.isArray() 메소드**를 사용합니다.

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

기본 자료형

자바스크립트에서는 **실체가 있는 것을 (undefined와 null 등이 아닌것) 중에 객체가 아닌 것을 기본 자료형** 이라고 부릅니다. **숫자, 문자열, 불이 바로 기본 자료형**이죠.

이러한 자료현은 객체가 아니므로 속성을 가질 수 없습니다. 예를들어 **숫자에 속성을 추가**해 보겠습니다.

```
> const c = 1000;
  c.sample = 10; ← 속성 추가
  (console.log(c.sample))
  undefined
```

속성을 만들 수 있는 것처럼 보이지만 실제로 속성이 만들어지지 않았습니다.

문자열과 불 자료형도 기본 자료형이므로 **속성이 추가되지는 않습니다.**

자바스크립트는 **기본 자료형을 객체로 선언하는 방법**을 제공합니다.

숫자, 문자열, 불 등으로 자료형을 변환하는 함수

(Number(), String(), Boolean())는 다음과 같이 사용합니다.

이렇게 사용하면 **숫자 객체, 문자열 객체, 불 객체를 생성**할 수 있습니다.

```
> new Number(0.001);
  new String("AI");
  new Boolean(true);

  const f = new Number(123);
  console.log(typeof f);

  f.sample = 10;
  속성을 가질 수 있습니다.
  console.log(f.sample);
  console.log(f);
  console.log(f + 0);
  console.log(f.valueOf());
```

new Number()를 사용해서 숫자를 생성하면 숫자와 관련된 연산자도 모두 활용할 수 있으며, 속성과 메소드를 활용할 수 있습니다.

숫자와 똑같이 활용할 수 있고 valueOf() 메소드를 사용해서 값을 추출할 수도 있습니다.

결괏값

```
object

10
▶ Number {123, sample: 10}
123
123
```

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

new 키워드를 사용하지 않을 때 주의점

new 키워드를 사용하지 않으면 함수가 자료형 변환 기능으로 작동합니다.

```
> const g = Number(123); // 기본 자료형
  typeof g
'number'
> const g = new Number(123); // 객체로 선언
  typeof g
'object'
```

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

기본 자료형의 일시적 승급

```
> 'hello_ai'.length
< 8
```

```
length
anchor          String
at
big
blink
bold
charAt
charCodeAt
codePointAt
concat
```

문자열 자료형 등을 생성하고 뒤에 온점을 찍으면 다음과 같이 자동 완성 기능으로 **메소드들이 등장**합니다. 당연히 실행도 가능하죠.

하지만 기본 자료형은 속성과 메소드를 가질 수 없습니다. 그런데 어떻게 메소드를 가질 수 있는 것일까요?

자바스크립트는 사용의 편리성을 위해서 **기본 자료형의 속성과 메소드를 호출할 때**(기본 자료형 뒤에 온점을 찍고 무언가 하려고 하면) 일시적으로 기본 자료형을 객체로 승급시켜줍니다. 그래서 속성과 메소드를 사용할 수 있는 것이죠.

따라서 기본자료형의 경우 속성과 메소드를 사용할 수는 있지만, 속성과 메소드를 추가로 가질 수는 없습니다.

```
> const h = 'hello ai';
// 속성을 추가해보겠지만...
h.sample = 10;
// 추가는 되지 않습니다.
console.log(h.sample);
// 하지만 일시적으로 승급은 가능
console.log(h.length);
```

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

프로토타입으로 메소드 추가하기

숫자 객체 전체에 어떤 속성과 메소드를 추가할 수 있다면 **기본 자료형 숫자도 속성과 메소드를 사용**할 수 있습니다.

어떤 객체의 prototype 이라는 속성이 바로 객체 전용 틀 이라 고 할 수 있습니다. **prototype객체에 속성과 메소드를 추가**하면 모든 객체(와 기본 자료형)에서 해당 속성과 메소드를 사용할 수 있습니다.

예를 들면 다음과 같이 loss 라는 속성을 추가해보도록 하겠습니다. Number.prototype에 loss 라는 속성을 추가하면 기본 자료형 뒤에 온점을 찍고 해당 속성을 활용할 수 있습니다.

객체 자료형 숫자에 속성과 메소드를 추가할 수 있다.

```
// 객체 자료형 이름.prototype.메소드 이름 = function () {}
Number.prototype.loss = function (real_y) {
    // 객체에 할당된 값을 valueOf()로 꺼내어 줍니다.
    // 그 값을 가지고 pred, loss 연산을 취해줍니다.

    weight = Math.random() * 2 -1 ; // 가중치의 범위는 -1 ~ 1 입니다.
    bias = 0;                       // 편향의 값은 0 입니다.
    // 예측값을 구하는 과정에서 this.valueOf() 를 취해 loss()가 할당된 해당 변수의 값을 가져옵니다.
    pred_y = (this.valueOf() * weight) + bias;
    // loss값을 구해주도록 합니다. (MSE)
    loss = ((pred_y - real_y) ** 2) / (1);


    console.log(`weight : ${weight}\n bias : ${bias}\n pred_y : ${pred_y}\n loss : ${loss}`);
    return loss;
}
```

추가된 속성 활용

```
> const x_input = 10;
< undefined
> x_input.loss(real_y=10);
  weight : 0.6809006009245757
   bias : 0
   pred_y : 6.809006009245757
   loss : 10.182442649029689
< 10.182442649029689
```

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

조금 더 실용적인 예제를 만들어 보기

자바스크립트에서 **문자열 내부에 어떤 문자열이 있는지, 배열 내부에 어떤 자료가 있는지 확인**할 때는 **indexOf() 메소드**를 사용합니다. 문자열의 **indexOf() 메소드를 사용하는 예**를 살펴보겠습니다.

다음 코드는 **사용자가 입력한 문자열 내부에 욕설이 들어가 있는지 확인**합니다. **욕설이 있으면 해당 문자열이 시작하는 위치(인덱스)를 출력**하고, **없으면 -1을 출력**합니다.

```
> const j = prompt('메시지를 입력하여 주세요.','');
  console.log(j);
  와 이걸? 젠장!
<· undefined
```

```
> console.log(j.indexOf('젠장'));
  console.log(j.indexOf('나쁜놈'));
  console.log(j.indexOf('사랑해'));
  6
  -1
  -1
```

그렇다면 "**문자열**.indexOf(**문자열**) >= 0" 등의 코드를 사용하면 **문자열 내부에 어떤 문자열이 포함되어 있는지 ture 또는 false**로 얻을 수 있습니다.

문자가 포함되어 있다면 True를, 없다면 false를 반환하는 형태로 변경하면 **자연어 처리 과정에서 유용하게 사용**될 수 있을 것입니다.

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

사용자에게 입력값을 받는다.

↓

금칙어 포함 여부를 확인합니다.

↓

여부에 따라 출력되어지는 결과값을 제어

JS

```
String.prototype.contain = function (data) {
  // 할당된 객체의 값을 받아 indexOf()를 통과시켜줍니다.
  // 찾고자 하는 문자가 있다면 0 이상의 값일 것이며,
  // 찾고자 하는 문자가 없다면 -1을 반환하게 됩니다.
  // 그래서 0과 비교해 찾고자 하는 값이 없다면 False를 반환,
  // 찾고자 하는 값이 있다면 True를 반환합니다.
  return this.indexOf(data) >= 0
}

Array.prototype.contain = function (data) {
  return this.indexOf(data) >= 0
}

// 비속어 검출 프로그램 - 첫 번째

// 사용자에게 문장을 입력받습니다.
const user_a = prompt('문장을 입력하여 주세요.', '안녕하세요 ~ ');
// 대화에서의 금칙어(비속어)를 선정합니다.
let text_filters = ['젠장'];
// 사용자의 대화에서 금칙어가 포함되어 있는지 여부를 확인합니다.
let filter_result = user_a.contain(text_filters)
// 사용자가 작성한 텍스트에 비속어가 포함되어 있는지 확인하여 줍니다.
alert(`${user_a} 은/는 비속어가 ${filter_result} 입니다.`);
```

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

Number 객체

기본 자료형과 연결된 객체에서 자주 사용하는 것만 기능적으로 살펴보겠습니다.

숫자 N번째 자릿수 까지 출력하기 : toFixed()

Number 객체에서 자주 사용하는 메소드는 toFixed() 입니다. **소수점 이하 몇 자리 까지만 출력**하고 싶을 때 사용합니다.

```
> const l = 123.456789
  l.toFixed(2)
<· '123.46'
> l.toFixed(3)
<· '123.457'
> l.toFixed(4)
<· '123.4568'
```

NaN과 Infinity 확인하기 : isNaN(), isFinite()

어떤 숫자가 **NaN(Not a Number)인지 Infinity 인지 확인**할 때는 **Number.isNaN() 메소드**와 **Number.isFinite() 메소드를 사용**하면 됩니다. 이 메소드 들은 숫자 자료 뒤에 온점을 찍고 사용하는 것이 아니라 **Number 뒤에 점을 찍고 사용**합니다.

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

String 객체 – 문자열 양쪽 끝의 공백 제거 : trim()

웹 어플리케이션에서는 **사용자가 전달해주는 데이터를 처리**해야 합니다. 그렇다면 **의도치 않게 문자 앞 뒤에 공백**을 넣어 전달하는 경우도 없지는 않을 것입니다. 이러한 경우를 위해 **문자열 trim()이라는 메소드를 사용하면 문자열 앞 뒤 공백을 제거**할 수 있습니다.

```
> const event_msg  = `
  배달의 민족 리뷰이벤트 참여
  console.log(event_msg);

  배달의 민족 리뷰이벤트 참여
<· undefined
> event_msg.trim()
<· '배달의 민족 리뷰이벤트 참여'
```

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

String 객체 – 문자열을 특정 기호로 잘라주는 split()

데이터를 수집하는 방법 중에는 '크롤링' 이라는 기법이 존재합니다. 크롤링을 진행하다보면 **다음과 같이 쉼표** 혹은 **다른 기호로 구분된 문자열을 읽어서 분해해야 하는 경우**가 있습니다.

이러한 경우 split() 메소드를 사용할 수 있습니다. split() 메소드는 문자열을 다른 문자열(매개변수)로 잘라서 배열을 만들어 리턴하는 메소드 입니다.

데이터 생성
(https://finance.naver.com/marketindex/)
에서 복사해온 자료입니다.

```
> let input = `미국 USD    1,183.30    1,204.00    1,162.60    1,194.80    1,171.80    1.000
  유럽연합 EUR    1,334.23    1,360.78    1,307.68    1,347.57    1,320.89    1.128`
undefined
> input = input.split('\n')
▼(2) ['미국 USD\t1,183.30\t1,204.00\t1,162.60\t1,194.80\t1,171.80\t1.000', '유럽연합 EUR\t1,334.23\t1,360.78\t1,307.68\t1,347.57\t1,320.89\t1.128']
    0: "미국 USD\t1,183.30\t1,204.00\t1,162.60\t1,194.80\t1,171.80\t1.000"
    1: "유럽연합 EUR\t1,334.23\t1,360.78\t1,307.68\t1,347.57\t1,320.89\t1.128"
    length: 2
  ▶[[Prototype]]: Array(0)
```

split('₩n')을 통해
줄바꿈 먼저 처리

```
> // input에 담긴 데이터를 하나씩 line으로 꺼내어주고,
  // 꺼낸 데이터에 split('\t')을 적용하여 데이터를 탭 단위로 분할하여 줍니다.
  input = input.map((line)=>line.split('\t'))
▼(2) [Array(7), Array(7)]
  ▶0: (7) ['미국 USD', '1,183.30', '1,204.00', '1,162.60', '1,194.80', '1,171.80', '1.000']
  ▶1: (7) ['유럽연합 EUR', '1,334.23', '1,360.78', '1,307.68', '1,347.57', '1,320.89', '1.128']
    length: 2
  ▶[[Prototype]]: Array(0)
```

split('₩t')을 통해
데이터를 각각 분리하여 줍니다.

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

JSON 객체

기본 자료형과 관련된 객체 외 **자바스크립트가 기본적으로 제공하는 내장 객체**를 몇 가지 살펴보겠습니다. 현재 가장 많이 사용되는 **자료 표현 방식은 JSON 객체** 입니다.

JSON은 **JavaScript Object Notation**의 약자로 **자바스크립트의 객체처럼 자료를 표현하는 방식**입니다. 다음은 JSON을 사용하여 '인공지능 모델'을 표현한 것입니다.

JSON 형식은 약간의 추가 규칙이 있습니다.

- ✓ 값을 표현할 때는 **문자열, 숫자, 불 자료형만 사용**할 수 있습니다
- ✓ **문자열은 반드시 큰따옴표**로 만들어야 합니다.
- ✓ 키(key)**에도 반드시 따옴표**를 붙여야 합니다.

대부분의 프로그래밍 언어는 JSON 형식의 문자열을 읽어들이는 기능이 있습니다. **자바스크립트 객체를 JSON 문자열로 변환할 때는 JSON.stringify()메소드를 사용**합니다.

JSON을 사용하여 인공지능 모델을 표현

```
// 하나의 json 자료 예
// 키(key) 또한 반드시 따옴표를 붙여주셔야 합니다.
const data_1 = {
    "model_layer" : "CNN_layer",
    "node" : 20,
    "activation" : "relu"
}

// 다수의 json 자료 예
// 다수인 경우 대괄호로 전체를 묶고, 중괄호로 구분하여 줍니다.
const dataset = [{
    "model_layer_1" : "CNN_layer",
    "node" : 20,
    "activation" : "relu"
}, {
    "model_layer_2" : "LSTM_layer",
    "node" : 50,
    "activation" : "softmax"
}]
```

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

JSON.stringify() 메소드 활용 예제

```
// 다수의 json 자료 예
// 다수인 경우 대괄호로 전체를 묶고, 중괄호로 구분하여 줍니다.
const dataset = [{
    "model_layer_1" : "CNN_layer",
    "node" : 20,
    "activation" : "relu"
}, {
    "model_layer_2" : "LSTM_layer",
    "node" : 50,
    "activation" : "softmax"
}]

// 자료를 json으로 변환합니다.
// 2번째 매개변수는 객체에서 어떤 속성만 선택해서 추출하고 싶을 때 사용(잘 사용하지 않음)
// 3번째 매개변수는 들여쓰기 칸 수를 의미합니다.
console.log(JSON.stringify(dataset, null, 2));
console.log(JSON.stringify(dataset));
```

console.log(JSON.stringify(dataset, null, 2));

```
[
  {
    "model_layer_1": "CNN_layer",
    "node": 20,
    "activation": "relu"
  },
  {
    "model_layer_2": "LSTM_layer",
    "node": 50,
    "activation": "softmax"
  }
]
```

console.log(JSON.stringify(dataset));

매개변수를 하나만 넣으면 한 줄로 변환됩니다.
일반적으로 이렇게 사용합니다.

```
[{"model_layer_1":"CNN_layer","node":20,"activation":"relu"},{"model_layer_2":"LSTM_layer","node":50,"activation":"softmax"}]
```

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

JSON 문자열을 **자바스크립트 객체로 전개**할 때는 **JSON.parse() 메소드를 사용**합니다. 매개변수에 **JSON 형식의 문자열**을 넣어 주면 됩니다.

```
// 자료를 json으로 변환합니다.
const json_dataset = JSON.stringify(dataset);
console.log(json_dataset);

// JSON 문자열을 다시 자바스크립트 객체로 변환합니다.
console.log(JSON.parse(json_dataset));
```

※ typeof를 통해 확인해 보면 string 이라 출력됩니다.

JSON 형식

```
[{"model_layer_1":"CNN_layer","node":20,"activation":"relu"},{"model_layer_2":"LSTM_layer","node":50,"activation":"softmax"}]
```

```
▼(2) [{…}, {…}]
  ▼0:
      activation: "relu"
      model_layer_1: "CNN_layer"
      node: 20
    ▶[[Prototype]]: Object
  ▼1:
      activation: "softmax"
      model_layer_2: "LSTM_layer"
      node: 50
    ▶[[Prototype]]: Object
    length: 2
  ▶[[Prototype]]: Array(0)
```

자바스크립트 객체

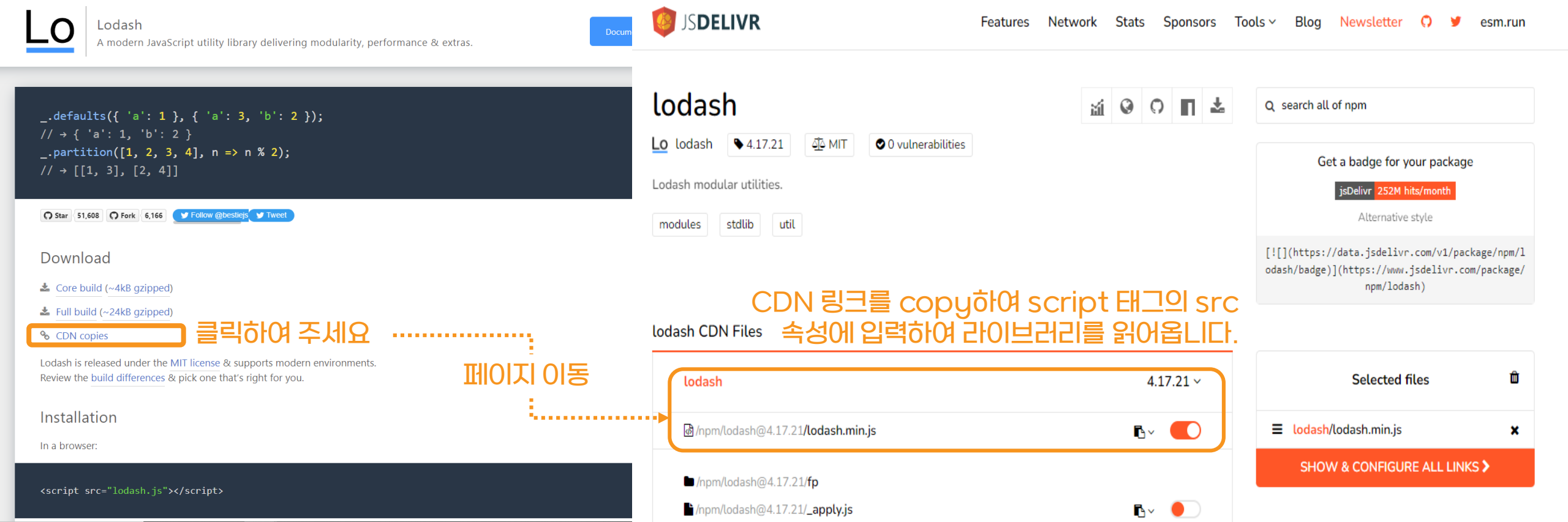
6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

외부 라이브러리

내가 만든 외부 자바스크립트 파일을 읽어들일 수도 있지만, **다른 사람이 만든 외부 자바스크립트 파일을 사용**할 수도 있습니다.

다른 사람들이 만든 다양한 함수와 클래스를 묶어서 제공해주는 것을 외부 라이브러리라고 부릅니다.

유틸리티 라이브러리는 개발할 때 보조적으로 사용하는 함수들을 제공해주는 라이브러리입니다. 최근 많이 사용되고 있는 **Lodash(Low dash) 라이브러리를 사용**(https://lodash.com/)해 보겠습니다



6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

CDN 이란 콘텐츠 전송 네트워크 입니다.

일반적으로 **어떤 사이트는 어떤 특정한 지역의 서버에 위치**합니다. 예를 들어 **광주시 홈페이지는 우리나라에 서버가 존재**하고, **미국 국무부 홈페이지는 미국에 서버가 존재**합니다. 그렇기 때문에 **한국에서 미국 국무부 홈페이지의 데이터를 전송 받으면 속도가 느립니다.** 또한 여러 사정에 의해 데이터를 아예 받을 수 없는 경우도 생길 수 있죠.

만약 전 세계 여러 지역에 전송할 데이터를 창고처럼 준비해두고 사용자가 데이터를 요청했을 때 가장 가까운 지역에서 데이터를 전송해 준다면 훨씬 빠르게 데이터를 전송할 수 있습니다. 또한 가까운 지역에 문제가 있으면 그 다음으로 가까운 지역에서 데이터를 전송하면 데이터를 받을 수 없는 문제도 해결할 수 있습니다. **이러한 통신 네트워크를** CDN (Contents Delivery Network) 이라고 부릅니다. **Lodash 라이브러리를** CDN 링크로 사용한다는 것**은** 이러한 곳으로부터 Lodash 파일을 읽어들여서 사용한다는 것입니다.

```
<script src = "https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js"></script>
```

CDN 링크를 보면 min.js 를 볼 수 있는데, **데이터의 용량을 줄이고자 모든 코드를 응축한 것**입니다. 따라서 CDN과 함께 min 버전의 파일을 사용하면 script 태그의 src 속성에 **그냥 링크를 입력해도 해당 파일을 매우 빠르게 다운로드 받아서 사용**할 수 있습니다.

6-2 자바스크립트에서 객체의 속성과 메소드 사용하기

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

HTML

```
<!-- CDN 형식으로 라이브러리를 불러와주도록 합니다 -->
<script src = "https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js"></script>
```

JS

```
// dataset 객체를 생성하는데, json 형식으로 작성합니다.
const dataset = [{
  "model_layer_1" : "CNN_layer",
  "node" : 20,
  "activation" : "relu"
}, {
  "model_layer_2" : "LSTM_layer",
  "node" : 50,
  "activation" : "softmax"
}, {
  "model_layer_3" : "GRU_layer",
  "node" : 120,
  "activation" : "softmax"
}, {
  "model_layer_4" : "ANN_layer",
  "node" : 10,
  "activation" : "elu"
}]
```

JS

```
// Lodash 라이브러리는 _ 라는 이름의 객체안에 수 많은 메소드를 담고 있습니다.
// sortBy() 메소드는 배열을 어떤 것으로 정렬할지 지정하면,
// 지정한 것을 기반으로 배열을 정렬해서 리턴해주는 메소드 입니다.

// dataset 데이터에서 값을 하나씩 빼 data에 할당하고,
// data의 "node" 키를 기준으로 정렬을 수행 후 결괏값을 output에 저장
const output = _.sortBy(dataset, (data) => data.node);
// output 값을 JSON 형식으로 출력합니다.
console.log(JSON.stringify(output, null, 2));
```

결괏값

키 “node” 의 값 크기에 맞춰 정렬

```
"[
  {
    'model_layer_4': 'ANN_layer',
    'node': 10,
    'activation': 'elu'
  },
  {
    'model_layer_1': 'CNN_layer',
    'node': 20,
    'activation': 'relu'
  },
  {
    'model_layer_2': 'LSTM_layer',
    'node': 50,
    'activation': 'softmax'
  },
  {
    'model_layer_3': 'GRU_layer',
    'node': 120,
    'activation': 'softmax'
  }
]"
```

6-3 객체와 배열 고급

이번 내용은 자바스크립트를 활용해서 리액트, 뷰 프레임워크 등을 개발할 때 사용하는 내용입니다. 조금은 난이도가 있는 내용이지만 "이러한 문법도 있다" 정도로만 알아두시는 것을 목표로 함께 공부하여 봅시다.

속성 존재 여부 확인

객체 내부에는 **어떤 속성이 있는지 확인해보는 코드**는 굉장히 자주 사용하는 코드입니다. 내가 직접 코드를 작성할 때도, 남이 만든 코드를 이해할 때도 필요하므로 간단하게 살펴보겠습니다.

JS

```js
const object = {
  "model_name" : "XOR_model",
  "node" : 240,
  "output_activation" : "sigmoid"
}
```

key값 이 있는 경우

```js
// object에 접근하여 key값을 불러온 후 undefined과 비교합니다.
if(object.model_name !== undefined){
  // 값이 undefined이 아니라면 즉, 값이 있다면
  // 객체의 키 값이 있다고 출력하여 줍니다.
  // ※ Object.keys()를 활용하여 key에 접근할 수 있습니다.
  console.log(`현재 ${Object.keys(object)[0]} 'key'는 있습니다.`)
} else { // 값이 없다면 없다는 알림을 주도록합니다.
  console.log(`${Object.keys(object)[0]} 'key'는 없습니다. `)
}
```

key값 이 없는 경우

```js
// object에 접근하여 존재하지 않는 키 값을 불러와 존재 여부를 비교합니다.
if(object.acc == undefined){
  // 존재하지 않는 경우 존재하지 않는다 출력합니다.
  console.log(`현재 해당 속성은 존재하지 않습니다.`)
} else {
  console.log(`현재 ${Object.keys(object)}이 존재합니다.`)
}
```

6-3 객체와 배열 고급

기본 속성 지정하기

원하는 **키 값이 있는지를 확인**하고, **없는 경우에 바로 추가**할 수 있는 방법입니다. 이전에 살펴보았던 **조건 연산자를 활용**하여 보겠습니다.

```
> // 하나의 json 자료 예
  const data_1 = {
      "model_layer" : "CNN_layer",
      "node" : 20,
      "activation" : "relu"
  }
  // data_1.acc의 값과 undefined의 값이 참이라면 'None Acc'의 값을 변수에 저장
  // data_1.acc의 값과 undefined의 값이 거짓이라면 data_1.acc의 값을 변수에 저장(새로운 key 생성)

  // ※ (조건) ? 참인 경우 수행 : 거짓인 경우 수행
  data_1.acc = data_1.acc == undefined ? 'None Acc' : data_1.acc;
  console.log(JSON.stringify(data_1, null, 2));
```

```
  {
    "model_layer": "CNN_layer",
    "node": 20,
    "activation": "relu",
    "acc": "None Acc"
  }
```

6-3 객체와 배열 고급

배열 기반의 다중 할당

배열과 비슷한 작성 방법으로 한 번에 여러 개의 변수에 값을 할당하는 다중 할당 기능을 살펴보겠습니다.

```
> let [a,b] = [1,2];

  // a = 1, b = 2 라는 값이 할당되어 집니다.
  console.log(a,b);

  [a, b] = [b, a]; // a에 b가 할당되고, b에 a가 할당되는 교환 성립
  console.log(a,b);
  1 2
  2 1
```

다음 코드의 경우 배열의 길이가 5인 arrayA의 값을 [a,b,c]에 할당합니다. **이렇게 하면 앞의 3개만 할당**됩니다.

```
> let arrayA = [1,2,3,4,5]
<· undefined
> const [a,b,c] = arrayA
<· undefined
> console.log(a,b,c);
  1 2 3
<· undefined
> console.log(a);
  console.log(b);
  console.log(c);
  1
  2
  3
```

6-3 객체와 배열 고급

객체 기반의 다중 할당

다음 코드는 객체 내부의 model_layer 속성과 node 속성을 꺼내 변수에 할당하는 예제입니다.

```
> const object = {
      "model_layer" : "DNN_hidden_layer1",
      "node" : 20,
      "activation" : "relu"
  }

  // object에 들어있는 model_layer, node를 꺼낼 수 있습니다.
  const {model_layer, node} = object;
  console.log(`model_layer : ${model_layer}, node : ${node}`);

  // model_layer의 값은 a로, node의 값은 b로 저장되어 집니다.
  const{a = model_layer, b = node} = object;
  console.log(`model_layer : ${a}, node : ${b}`);
  model_layer : DNN_hidden_layer1, node : 20
  model_layer : DNN_hidden_layer1, node : 20
```

※ 같은 결괏값을 확인할 수 있다.

7-1 문서 객체 조작하기

DOMContentLoaded 이벤트

이 코드는 문서의 **바디(body) 안에 있는 HTML 코드(**innerHTML)를 **자바스크립트로 조작**할 수 있게 하는 코드입니다.

head 태그 내부의 script 태그에서 body 태그에 있는 문서에 접근하려면 **화면에 문서 객체(요소)를 모두 읽어 들일때 까지** 기다려야 합니다.

결괏값

2번째 script 태그

1번째 h1 태그

3번째 script 태그

2번째 h2 태그

HTML 코드를 자바스크립트로 조작하기

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8      <script>
9          // HTML H1 태그를 쉽게 만들 수 있도록 콜백 함수 선언
10         const h1 = (text) => `<h1>${text}</h1>`
11     </script>
12     <script>
13         // 위에서 생성한 h1()를 사용하여 body에 입력을 수행합니다.
14         document.body.innerHTML += h1('1번째 script 태그')
15     </script>
16 </head>
17 <body>
18     <script>
19         // 위에서 만든 h1()를 활용합니다.
20         document.body.innerHTML += h1('2번째 script 태그')
21     </script>
22     <h1>1번째 h1 태그</h1> <!-- 일반적인 h1 테그를 사용하는 과정입니다. -->
23     <script>
24         // 위에서 만든 h1()를 활용합니다.
25         document.body.innerHTML += h1('3번째 script 태그')
26     </script>
27     <h1>2번째 h2 태그</h1> <!-- 일반적인 h1 테그를 사용하는 과정입니다. -->
28 </body>
29 </html>
```

7-1 문서 객체 조작하기

DOMContentLoaded 이벤트는 웹 브라우저가 문서 객체를 모두 읽고 나서 실행하는 이벤트 입니다. 다음과 같이 코드를 구성하면 DOMContentLoaded 상태가 되었을 때 콜백 함수를 호출합니다.

DOMContentLoaded 이벤트

```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>
    <script>
        // document 라는 문서 객체의 DOMContentLoaded
        // (문서 객체를 모두 읽고나서 실행)이벤트가 발생했을 때,
        // 매개변수로 지정한 콜백 함수를 실행하라.
        // 문서객체.addEventListener(이벤트 이름, 콜백함수)
        document.addEventListener('DOMContentLoaded', () => {
            // 입력된 글자 크기를 h1으로 키워준다.
            const h1 = (text) => `<h1> ${text} </h1>`
            // DOMContentLoaded 이벤트 발생 라는 글자 크기 증가
            document.body.innerHTML += h1('DOMContentLoaded 이벤트 발생')
        })
    </script>
</head>
<body>

</body>
</html>
```

결괏값

DOMContentLoaded 이벤트 발생

7-1 문서 객체 조작하기

querySelctor() 메소드 활용예제

문서 객체 **하나를 추출합니다.**

그리고 다음과 같이 **<h1> 태그가 사용**되었을 때 **글자, 글자의 색, 글자의 배경색, 글자 크기**가 **사용자가 원하는 방향**으로 바뀌도록 자바스크립트를 작성할 수 있습니다.

querySelctor() 메소드 활용

```
<!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>
    <script>
        // 문서에 접근하여(document) 이벤트를 발생시켜줍니다.
        document.addEventListener('DOMContentLoaded', () => {
            // h1 태그를 가져와 header 라는 변수에 할당하여 줍니다.
            const header = document.querySelector('h1')

            // 변수 header에 접근하여 스타일 테그를 통해 아래와 같이 적용시켜 줍니다.
            header.textContent = 'HEADERS' // 적용시킬 문자열
            header.style.color = 'white'   // 글자의 색상
            header.style.backgroundColor = 'black' // 배경의 색상
            header.style.padding = '10px' // 글자의 주변 크기
            })
    </script>
</head>
<body>
    <!-- 테그 사이에는 아무 값도 없지만 상단의 <script> 에서 문자부터 css까지 전부 적용 -->
    <h1></h1>
</body>
</html>
```

<h1> 태그가 사용되면 발생하는 명령

<h1>태그

7-1 문서 객체 조작하기

querySelctorAll() 메소드 활용예제

문서 객체 여러 개를 배열로 읽어들이는 함수입니다.

따라서 **내부의 요소에 접근하고 활용**하려면

반복문을 돌려야 합니다.

일반적으로 **forEach() 메소드를 사용**해서

반복문을 돌립니다.

결괏값

HEADERS

HEADERS

HEADERS

HEADERS

querySelctorAll() 메소드 활용

```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>
    <script>
        // 문서에 접근하여(document) 이벤트를 발생시켜줍니다.
        document.addEventListener('DOMContentLoaded', () => {
            // 문서상의 모든 <h1> 태그를 가져와 headers 라는 변수에 할당하여 줍니다.
            const headers = document.querySelectorAll('h1')
            // headers에 접근하여 값을 하나씩 header 변수에 할당합니다.
            headers.forEach((header) => {
                // 변수의 값을 아래와 같은 스타일로 적용시켜 줍니다.
                header.textContent = 'HEADERS' // 적용시킬 문자열
                header.style.color = 'white'   // 글자의 색상
                header.style.backgroundColor = 'black' // 배경의 색상
                header.style.padding = '10px' // 글자의 주변 크기
            })
        })
    </script>
</head>
<body>
    <!-- 테그 사이에는 아무 값도 없지만 상단의 <script> 에서 문자부터 css까지 전부 적용 -->
    <h1></h1>
    <h1></h1>
    <h1></h1>
    <h1></h1>
</body>
</html>
```

forEach() 메소드를 사용

7-1 문서 객체 조작하기

글자 조작하기

지금까지 살펴본 예제들에서 innerHTML 속성과 textContent 속성을 사용해서 문서 객체 내부의 글자를 조작하였습니다.

textContent 속성은 입력된 문자열을 그대로 넣어주고, **innerHTML 속성은 입력된 문자열을 HTML 형식**으로 넣어줍니다.

글자 조작하기

```
<!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>
    <div id = 'a'></div>
    <div id = 'b'></div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            // 문서상 id a, b를 찾아서 변수화
            const a = document.querySelector('#a');
            const b = document.querySelector('#b');

            // 두 변수 다 h1 태그를 받고 있지만 결국 b만 h1 태그가 적용됩니다.

            a.textContent = '<h1>textcontent 속성</h1>'
            b.innerHTML = '<h1>innerHMTL 속성</h1>'
        })
    </script>
</body>
</html>
```

결괏값

<h1>textcontent 속성</h1>

innerHMTL 속성

첫 번째 문장은 아무런 변화가 보이지 않습니다.

7-1 문서 객체 조작하기

속성 조작하기

문서 객체의 속성을 조작할 때는 다음과 같은 메소드를 사용합니다.

메소드 이름	문서 객체.setAttribute(속성 이름, 값)
설명	특성 속성에 값을 지정합니다.
메소드 이름	문서 객체.getAttribute(속성 이름)
설명	특성 속성을 추출합니다.

결괏값

속성 조작하기(https://codepen.io/AI_R_IM/pen/xxXqzXe?editors=1000)

```
<!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>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            // class가 rect인 모든 요소를 변수 rects에 저장합니다.
            const rects = document.querySelectorAll('.rect')
            // rects에 담긴 것을 하나씩 꺼내 변수 rect에 저장.
            // 인덱스도 할당하여 index에 저장 (0,1,2,3)
            rects.forEach((rect, index)=> {
                // 이미지를 하나씩 꺼낼 때 마다 100, 200, 300, 400의 크기를 갖는
                // width 변수 생성
                const width = (index + 1) * 100
                // 위 변수의 값을 이미지 크기로 할당
                const src   = `https://placekitten.com/${width}/250`
                // rect 변수의 src라는 속성에, 위에서 생성한 변수 src를 할당합니다.
                rect.setAttribute('src',src)
            })
        })
    </script>
    <!-- img 태그지만 src 요소가 없습니다. 이는 js를 통해 생성합니다. -->
    <img class = 'rect'>
    <img class = 'rect'>
    <img class = 'rect'>
    <img class = 'rect'>
</body>
</html>
```

7-1 문서 객체 조작하기

스타일 조작하기

문서 객체의 스타일을 조작할 때는 **style 속성을 사용**합니다. style 속성은 객체이며, 내부에는 속성으로 css를 사용해서 지정할 수 있는 스타일 들이 존재합니다. 이러한 속성에는 css로 입력할 때 사용하는 값과 같은 값을 입력합니다.

다만 css에서 사용할 때와 약간 다릅니다. **자바스크립트에서는 – 기호를 식별자에 사용할 수 없으므로, 두 단어 이상의 속성**은 다음과 같이 **케멜 케이스(Camel Case)**나타냅니다.

Css 속성 이름	자바스크립트 style 속성 이름
background-color	backgroundColor
text-align	textAlign
font-size	fontSize

style 객체는 3가지 방법으로 조정할 수 있습니다. 일반적으로는 첫 번째 방법을 가장 많이 사용합니다.

```
h1.style.backgroundColor
h1.style.['backgroundColor']
h1.style.background-Color
```

결괏값(https://codepen.io/AI_R_IM/pen/bGoqxzE)

※ rgb(적색, 녹색, 청색)

```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>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            // body 태그 하위의 div 태그 선택하여 divs 변수에 할당
            const divs = document.querySelectorAll('body > div');
            // divs에 담긴 전체 div 태그를 하나씩 가져옵니다.
            divs.forEach((div, index) => {
                //console.log(div, index);

                // 화면 배경 색상에 변화를 주기 위해
                // 반복 처리를 수행할 때 마다 10씩 증가하는 변수값을 생성
                const val = index * 10
                // 높이 강제 지정
                div.style.height = `10px`
                // 배경 색상은 rgb()함수에 변수 val를 할당하여
                // 배경 색상을 조금씩 변화를 줍니다.
                div.style.backgroundColor = `rgb(${val}, ${val}, ${val})`
            })
        })
    </script>
    <!-- body > div -->
    <div></div><div></div><div></div><div></div><div></div><div></div>
    <div></div><div></div><div></div><div></div><div></div><div></div>
    <div></div><div></div><div></div><div></div><div></div><div></div>
    <div></div><div></div><div></div><div></div><div></div><div></div>
    <div></div><div></div><div></div><div></div><div></div><div></div>
</body>
</html>
```

7-1 문서 객체 조작하기

문서 객체 생성하기

지금까지는 **body 태그 내부에 있는 특정 문서 객체를 읽어들이고 이를 조작**했습니다. **문서 객체를 생성**하고 싶을 때에는 document.createElement() 메소드를 사용합니다.

그런데 **문서 객체를 만들었다고 문서 객체가 배치되는 것은 아닙니다.** 문서를 **어떤 문서 아래에 추가할지를 지정**해줘야 합니다. 이러한 그림을 프로그래밍에서는 **Tree** 라고 부릅니다. 어떤 문서 객체가 있을 때 **위에 있는 것을 부모(parent)**라고 부르고, **아래에 있는 것을 자식(child)**이라고 부릅니다.

문서 객체에는 **appendChild() 메소드**가 있으며, 이를 활용하면 **어떤 부모 객체 아래에 자식 객체를 추가**할 수 있습니다. 문서 객체 트리 구조를 만드는 방법은 다음과 같습니다.

부모 객체.appendChild(자식객체)

7-1 문서 객체 조작하기

document.createElement() 메소드로 **<h1> 태그를 생성**하고, 이를 **document.body** 태그 아래에 연결하는 코드

```
<!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>
    <script>
        // 문서에 addEventListener() 함수 적용
        document.addEventListener('DOMContentLoaded', () => {
            // <h1> 태그를 생성하는 변수 header를 생성합니다.
            const header = document.createElement('h1')

            // h1 태그를 담는 header 변수에 텍스트를 생성(textContent)합니다.
            header.textContent = '문서 객체 동적으로 생성하기'
            // header에 담긴 글자에 style 태그를 달아줍니다.
            header.style.color = 'white'
            header.style.backgroundColor = 'black'

            // h1 태그의 상위 태그인 body와 연결시켜 주기 위해 appendChild()를 활용
            document.body.appendChild(header)
        })
    </script>
</body>
</html>
```

<h1> 태그 생성

<body> 태그와 <h1> 태그 연결

7-1 문서 객체 조작하기

문서 객체 이동하기

appendChild()메소드는 문서 객체를 이동할 때도 사용할 수 있습니다.

문서 객체의 부모는 언제나 하나여야 합니다. 따라서 문서 객체를 다른 문서 객체에 추가하면 문서 객체가 이동합니다.

다음 코드는 1초마다 <h1> 태그가 이동하게 됩니다.

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

```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>
    <div id = 'first'>
        <h1>첫 번째 div 태그 내부</h1>
    </div>
    <hr>
    <div id = 'second'>
        <h1>두 번째 div 태그 내부</h1>
    </div>
</body>
</html>
```

JS

```js
document.addEventListener('DOMContentLoaded', () => {
  // id 속성이 first인 div 태그를 선택하여 변수화 합니다.
  const divA = document.querySelector('#first')
  // id 속성이 second인 div 태그를 선택하여 변수화 합니다.
  const divB = document.querySelector('#second')
  // h1 태그를 생성하여 줍니다.
  const h1 = document.createElement('h1') //
  // h1 태그를 적용시켜 생성하는 문자열
  h1.textContent = '이동하는 h1 태그'

  // 익명 함수
  const toFirst = () => {
    // h1 태그를 divA 하위에 붙여줍니다.
    divA.appendChild(h1)
    // setTimeout()를 통해 1초뒤 toSecond 함수 실행
    setTimeout(toSecond, 1000) // 1초(1000ms)뒤에 toSecond() 함수 실행
  }

  // toSecond 함수
  const toSecond = () => {
    // H1 태그를 divB에 연결합니다.
    divB.appendChild(h1)
    // 10초 뒤 위에서 정의한 toFirst함수 호출
    setTimeout(toFirst, 3000)

  }
  // toFirst() 함수 동작
  toFirst()
})
```

7-1 문서 객체 조작하기

이벤트 설정하기

지금까지 계속 document.addEventListener('DOMContentLoaded', () => { }) 라는 형태의 코드를 사용하고 있습니다. 이 코드는 "**document라는 문서 객체의 DOMContentLoaded 이벤트가 발생했을 때, 매개변수로 지정한 콜백 함수를 실행**해라" 는 의미입니다.

문서 객체.addEventListener(이벤트 이름, 콜백 함수)

※ 이벤트 리스너 또는 이벤트 핸들러 라고 부릅니다.

모든 문서 객체는 생성되거나 클릭되거나 마우스를 위에 올리거나 할 때 **이벤트(event) 라는 것이 발생**합니다. 그리고 이 이벤트가 발생할 때 실행할 함수는 **addEventListener() 메소드를 사용**합니다.

이벤트가 발생할 때 실행할 함수를 **이벤트 리스너(event listener) 또는 이벤트 핸들러(event handler)**라고 부릅니다.

7-1 문서 객체 조작하기

결괏값

클릭 횟수 : 44

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

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>
    <h1>클릭 횟수 : 0</h1>
</body>
</html>
```

CSS

```css
/* 글자가 중복해서 선택되는 것을 막습니다. 드레그를 막습니다. */
h1 {
  user-select : none;
}
```

JS

```js
// DOMContentLoaded 는 문서를 전부 읽은 후 함수를 실행
document.addEventListener('DOMContentLoaded', () => {
  let counter = 0 // 클릭하는 횟수를 받을 변수를 생성합니다.
  // h1 태그를 찾아 변수화 처리 합니다.
  const h1 = document.querySelector('h1')
  // h1 태그에 click 이벤트가 발생하면 event 함수가 발생
  h1.addEventListener('click', (event) => {
    // counter 변수의 값을 1씩 증가
    counter ++
    // h1 태그에 textContent를 통해 글자 생성 및
    // 증가한 counter 값을 출력합니다.
    h1.textContent = `클릭 횟수 : ${counter}`
  })
})
```

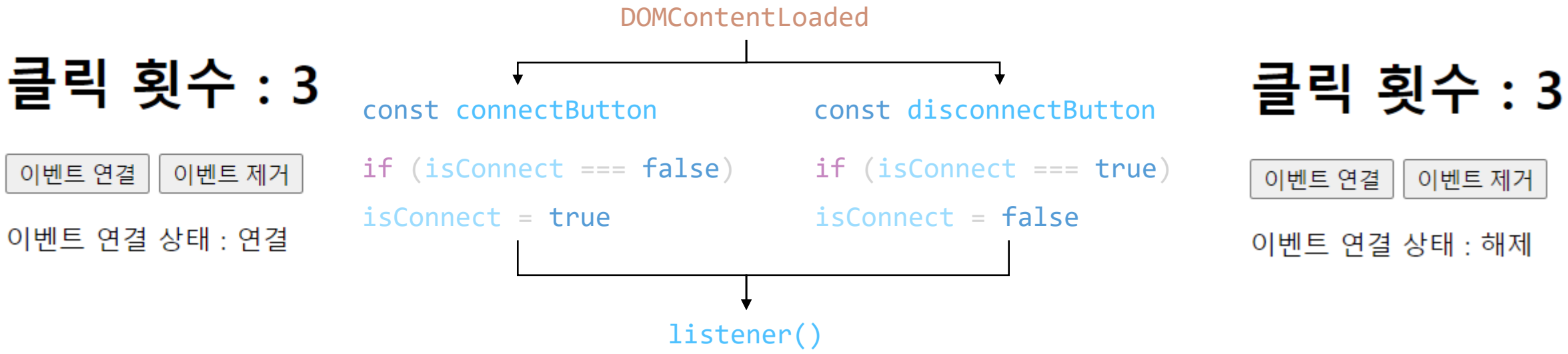
7-1 문서 객체 조작하기

이벤트 제거하기

이벤트를 제거할 때는 다음과 같은 형태로 **removeEventListener() 메소드를 사용**합니다. 이벤트 리스너 부분에는 연결할 때 사용했던 이벤트 리스너를 넣습니다. **변수 또는 상수로 이벤트 리스너**를 미리 만들고, 이를 **이벤트 연결과 연결 제거에 활용**합니다.

문서 객체.removeEventListener(이벤트 이름, 이벤트 리스너)

결괏값(버튼을 클릭함에 따라 동작하는 함수가 달라지게 됩니다.)
https://codepen.io/AI_R_IM/pen/VwMbmoe?editors=0010



7-2 이벤트 활용

이번에는 입력 양식에 관련된 내용을 많이 살펴보겠습니다.

이벤트 모델

이벤트를 연결하는 방법을 **이벤트 모델** 이라고 부릅니다. 이전에는 이벤트를 연결할 때 addEventListener() 메소드를 사용했습니다. 이 방법은 현재 표준으로 사용하고 있는 방법이므로 **표준 이벤트 모델** 이라고 부릅니다.

```
1 document.body.addEventListener('keyup', () => {
2
3 })
```

과거에는 다음과 같이 문서 객체가 갖고 있는 **on● ●으로 시작하는 속성**에 **함수를 할당해서 이벤트를 연결**했습니다. 이와 같은 이벤트 연결 방법을 고전 이벤트 모델 이라고 합니다.

```
1 document.body.onkeyup = (event) => {
2
3 }
```

그리고 고전 이벤트 모델처럼 on ● ● 으로 시작하는 속성을 **HTML 요소에 직접 넣어서 이벤트를 연결하는 것**을 **인라인 이벤트 모델** 이라 합니다.

7-2 이벤트 활용

인라인 이벤트 모델은 HTML 요소의 on● ● **속성에 자바스크립트 코드를 넣는것**입니다. 현재 코드에서는 listener() 라는 함수를 호출하고 있습니다. 이때 on ● ● 속성 내부에서 변수 event를 활용할 수 있습니다. 이 변수를 **listener() 함수의 매개변수**로 전달합니다.

```
<script>
    const listener = (event) => {

    }
</script>
<body onkeyup="listener(event)">
</body>
```

모든 이벤트 모델의 이벤트 리스너는 **첫 번째 매개변수로 이벤트 객체**를 받습니다. 이벤트 객체에는 이벤트와 관련된 정보가 들어있습니다. 웹 브라우저에는 많은 이벤트가 있으므로 이벤트를 모두 설명하는 것은 불가능합니다. 또한 모든 이벤트와 이벤트 객체의 속성을 다 공부하는 것 또한 추천하지 않습니다.

하지만 최근 프론트엔드 프레임워크들이 인라인 이벤트 모델을 활용하는 형태로 코드를 작성해서 현재에는 **인라인 이벤트 모델과 표준 이벤트 모델을 많이 사용**하고 있습니다. 현대에 인라인 이벤트 모델이 어떻게 활용되는지는 이후에 **리엑트 프레임워크**를 통해 살펴보면서 공부해 보시기를 권해드립니다.

7-2 이벤트 활용

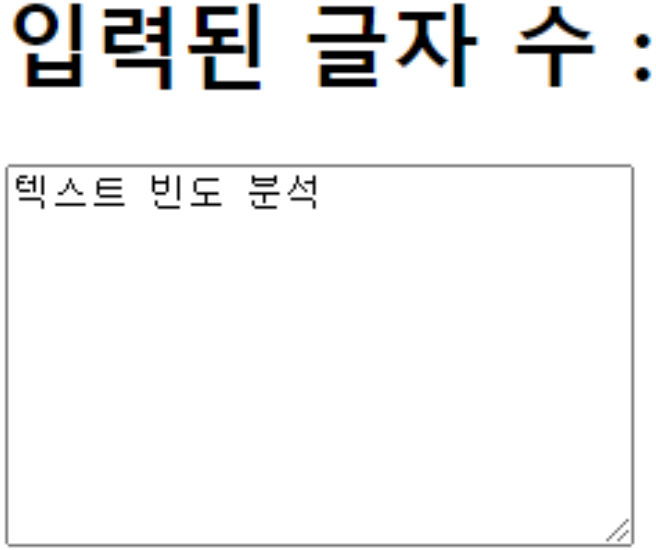
키보드 이벤트

키보드 이벤트는 다음과 같이 3가지 이벤트가 있습니다.

이벤트	설명
keydown	키가 눌릴 때 실행됩니다. 키보드를 꾹 누르고 있을 때도, 입력될 때도 실행됩니다.
keypress	키가 입력되었을 때 실행됩니다. 하지만 웹 브라우저에 따라서 아시아권의 문자를 제대로 처리하지 못하는 문제가 있습니다.
keyup	키보드에서 키가 떨어질 때 실행됩니다.

위와같은 문제가 있기에 주로 keyup 이벤트를 사용하곤 합니다.

그럼 간단하게 **키보드 이벤트로 입력 양식의 글자 수를 세는 프로그램**을 만들어 보겠습니다. 다음 프로그램은 textarea에 keyup 이벤트를 적용해서 입력한 글자 수를 세는 프로그램입니다. textarea 처럼 텍스트를 입력하는 **입력 양식의 값은 value 속성**으로 읽어들입니다.



7-2 이벤트 활용

```
<body>
    <script>
        // 문서를 모두 읽고나서 동작합니다.
        document.addEventListener('DOMContentLoaded', () =>{
            // 문서상의 textarea,h1 태그를 찾아 변수화 합니다.
            const textarea = document.querySelector('textarea');
            const h1       = document.querySelector('h1');

            // textarea 변수에 대하여 이벤트를 줍니다.
            // 키보드에서 키가 떨어질때(keyup)
            textarea.addEventListener('keyup', (event) => {

                //console.log(textarea.value);
                //textarea에 저장된 값(value)의 길이(length)를 변수화 합니다.
                const length = textarea.value.length
                // h1변수에 아래와 같은 안내글과 입력된 글자 수를 변수화 합니다.
                // 그리고 h1태그에 출력되어집니다.
                h1.textContent = `입력된 글자 수 : ${length}`
            })
        })
    </script>
    <h1></h1>

    <!-- 글자를 입력받는 창을 생성합니다. -->
    <textarea></textarea>
</body>
```

1. html 상의 태그를 찾아 변수화 합니다.

2. 입력 태그(textarea)에서 키보드가 눌리는 이벤트가 발생하면 수행되는 기능 정의

3. 자바스크립트에서 수행된 명령에 따라 출력되어지는 HTML 소스

7-2 이벤트 활용

키보드 키 코드 사용하기

키보드 이벤트가 발생할 때는 **이벤트 객체로 어떤 키를 눌렀는지와 관련된 속성들**이 따라옵니다.
여러 속성들이 있지만, 여기서는 다음과 같은 속성만 살펴보겠습니다.

이벤트 속성 이름	설명
Code	입력한 키
keyCode	입력한 키를 나타내는 숫자
altKey	[Alt] 키를 눌렀는지
ctrlKey	[Ctrl] 키를 눌렀는지
shiftKey	[Shift] 키를 눌렀는지

code 속성은 입력한 키를 나타내는 문자열이 들어있고, altKey, ctrlKey, shiftKey 속성은 해당키를 눌렀는지 불 자료형 값이 들어있습니다. 어떤 의미인지 간단하게 속성을 출력하는 프로그램을 만들어 살펴보겠습니다.

alt : true
ctrl : true
shift : true
code : KeyX

7-2 이벤트 활용

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

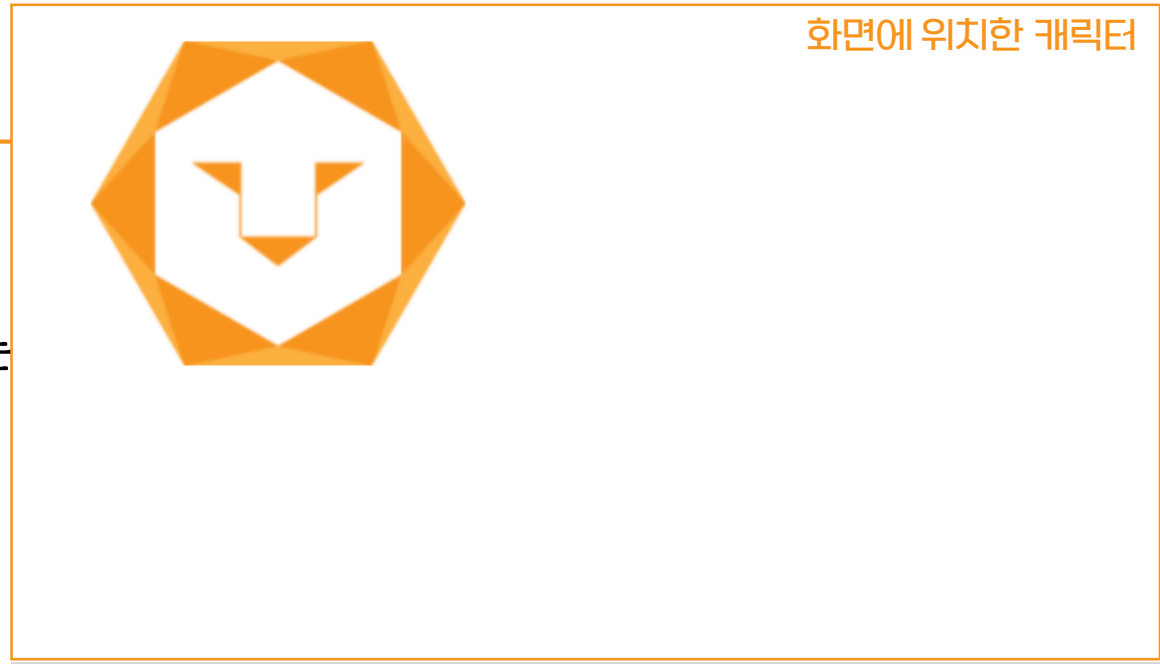
```
<body>
    <h1></h1>
    <script>
        // html 코드를 다 읽고 나서 스크립트를 수행하도록 합니다.
        document.addEventListener('DOMContentLoaded', () => {
            // 문서상에서 h1 태그를 가져와 변수화 합니다.
            const h1 = document.querySelector('h1')
            // print라는 함수를 생성하는데,
            // 이 함수는 다음과 같은 키보드 이벤트를 제어합니다.
            const print = (event) => {
                // 키보드가 눌리면 이벤트가 저장되는 변수 생성
                let output = ''

                output += `alt : ${event.altKey}<br>`    // alt가 눌렸는지 여부를 담습니다.

                output += `ctrl : ${event.ctrlKey}<br>`  // ctrl이 눌렸는지 여부를 담습니다.
                output += `shift : ${event.shiftKey}<br>`// shift가 눌렸는지 여부를 담습니다.

                // event.code의 타입(string)을 확인해서 그 값이 undefined 이 아닌지 확인합니다.
                // 이는 무조건 true를 반환하게 됩니다.
                // 이유는 인터넷 익스플로러와 구 버전 엣지 웹 브라우저는 code 속성을 지원하지
                // 않기에 다음과 같이 keyCode 속성을 추가하여 놓았습니다.
                output += `code : ${typeof(event.code) !== 'undefined' ?
                    event.code : event.keyCode}<br>`

                // 결과인 output은 h1 태그에 innerHTML로 전달하여 줍니다.
                h1.innerHTML = output
            }
            // 키보드가 눌릴때 동작하는 함수 print 입니다.
            document.addEventListener('keydown', print)
            // 키보드에서 키가 떨어질 때 실행되는 함수 print 입니다.
            document.addEventListener('keyup', print)
        })
    </script>
</body>
```

7-2 이벤트 활용

키로 캐릭터 움직이기

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

한국어 처리 때문에 keyup 이벤트를 많이 사용한다고 했는데, **방향키는 처리 문제가 없으므로** 다른 이벤트를 활용해도 됩니다. **방향키를 사용하는 게임 등을 할 때는 방향키를 꾹 누르고 있을 가능성**이 많으므로 **keydown 이벤트를 활용**하여 주겠습니다.



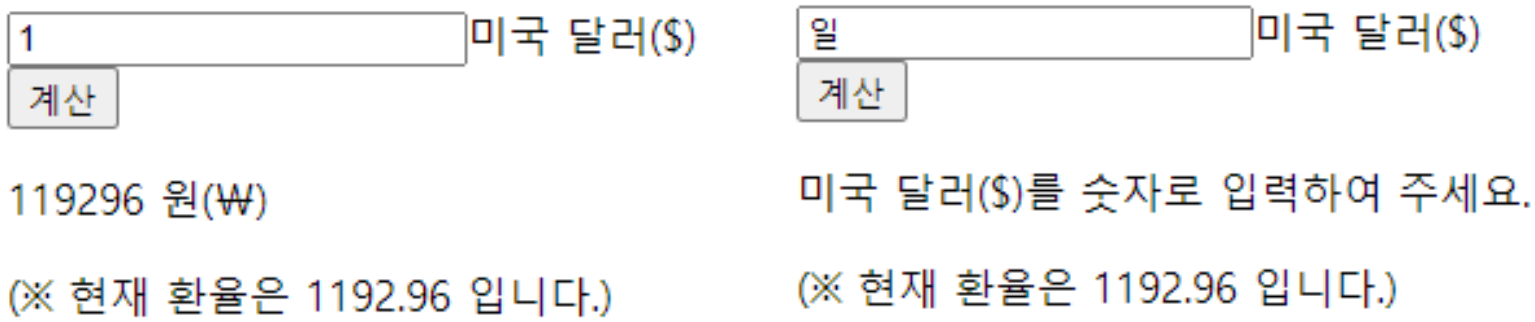
7-2 이벤트 활용

글자 입력 양식 이벤트

사용자로부터 어떠한 입력을 받을 때 사용하는 요소를 **입력 양식(form)**이라고 부릅니다. HTML 에서는 input 태그, textarea 태그, button 태그, select 태그 등이 모두 입력 양식입니다. 웹에서 어떤 사이트에 회원 가입하거나 댓글을 입력하거나 음식을 주문할 때 활용해본 적이 있을 것입니다.

https://codepen.io/AI_R_IM/pen/xxXLdvy?editors=1100

입력 양식을 사용하는 간단한 예를 살펴보겠습니다. **원화(₩) 단위를 달러($) 단위로 변환하는 프로그램**입니다.

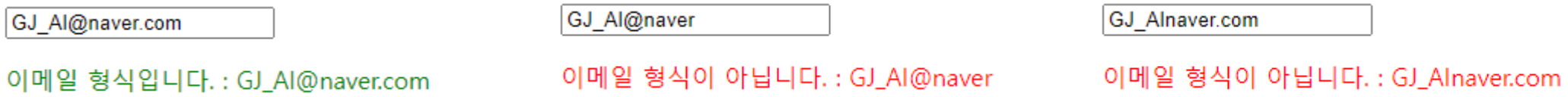


7-2 이벤트 활용

이메일 형식 확인하기

이번에는 인터넷에서 **특정 사이트에 가입**할 때 **이메일과 전화번호 유효성 등을 검사**하는 것을 구현해 보겠습니다. 이는 사용자가 **잘못된 이메일 또는 전화번호를 실수로 입력하는 것을 막는 역할**을 해줍니다.

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



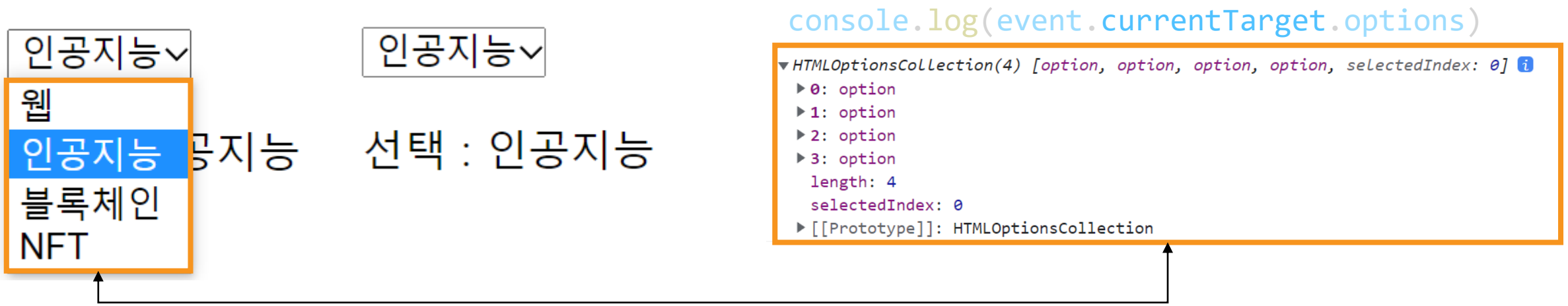
위와 같이 **이메일 형식이 아닐 때는 경고 메시지**를, **이메일 형식이 올바르게 입력되었다면 그에 맞는 내용을 출력**합니다.

7-2 이벤트 활용

드롭다운 목록 활용하기

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

드롭다운 목록은 기본적으로 **select 태그로 구현**합니다. **select 태그**는 사용 방법이 조금 특이하므로 함께 사용 방법을 알아보겠습니다.



코드 중간에 **change 이벤트가 발생**하면 **event.currentTarget.options** 의 값을 호출하여 변수화 합니다.

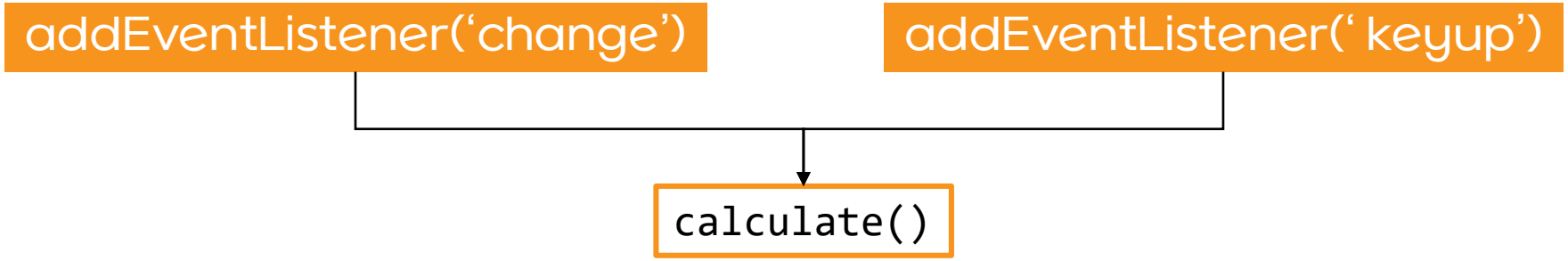
이 값이 가지고 있는 내용은 위와 다음과 같습니다.

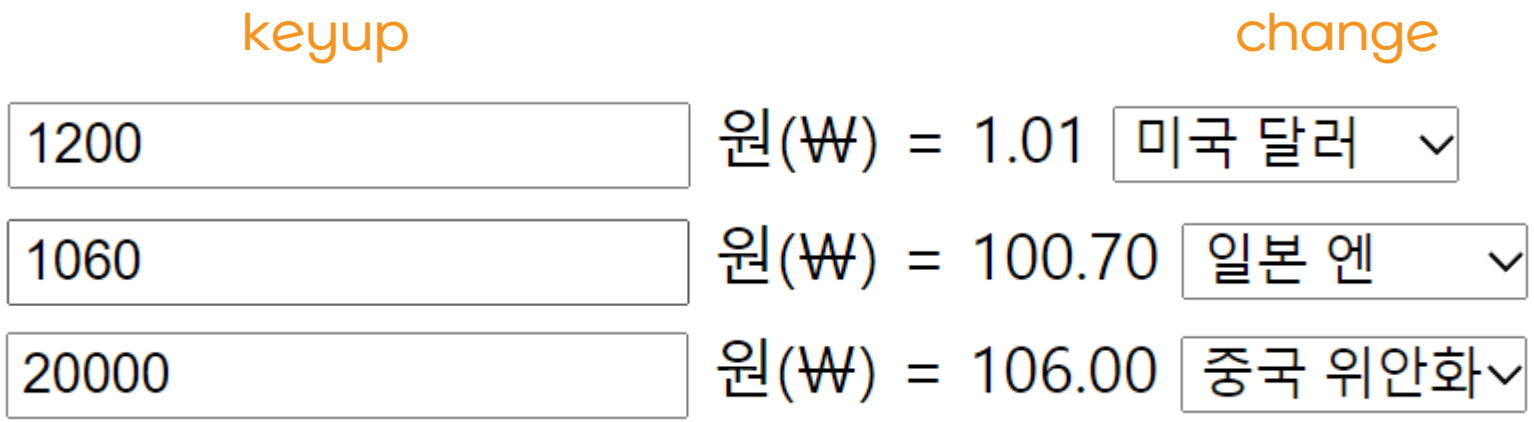
7-2 이벤트 활용

단위를 여러 단위로 변환하는 프로그램

이전에는 하나의 단일 단위를 변환하는 과정을 살펴보았다면 이번에는 **드롭다운 방식을 적용**하여 원하는 보기에 맞게 단위를 변환하는 프로그램을 만들어 보겠습니다. 본 프로그램은 **change와 keyup 이벤트가 발생하면 구현해둔 단위 변환 함수를 동작**시켜 주도록 합니다.

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





7-2 이벤트 활용

라디오 버튼 활용하기

체크 박스와 비슷한 입력 양식 요소로 **라디오 버튼**이 있습니다. 라디오 버튼은 다음과 같이 **하나의 보기만을 선택해야 하는 상황**에서 주로 사용되어 집니다.

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

사용하고자 하는 인공지능 기술을 선택하여 주세요.

◉ 이미지 분류 ○ 텍스트 감정분석 ○ 음성 분류 ○ 영상 합성

선택된 기술은 이미지 분류입니다.

사용하고자 하는 인공지능 기술을 선택하여 주세요.

○ 이미지 분류 ○ 텍스트 감정분석 ◉ 음성 분류 ○ 영상 합성

선택된 기술은 음성 분류입니다.

8-1 Danfo.js

danfo.js 활용하여 데이터 생성하기

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

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- danfojs를 불러오는 과정입니다. -->
    <script src="https://cdn.jsdelivr.net/npm/danfojs@0.3.3/lib/bundle.min.js"></script>
</head>
<body>
    <script>
        // dfd의 Series()를 사용해 인스턴스(s)를 생성합니다.
        s = new dfd.Series([1,2,3,4,5])

        // s를 다양한 방식으로 출력하여 줍니다.
        //console.log(s);
        console.log(s.$data);
        // 원하는 특정 범위를 선택하여 출력합니다.
        console.log(s.$data.slice(0,3));
        // 변수 s를 출력하는 함수 print() 입니다.
        s.print()
    </script>
</body>
</html>
```

danfo.js를 불러오는 코드

데이터(시리즈) 생성 및 출력

결괏값

```
▶ (5) [1, 2, 3, 4, 5]
▶ (3) [1, 2, 3]
```

0	1
1	2
2	3
3	4
4	5

https://danfo.jsdata.org/

8-1 Danfo.js

텐서(tensor)를 활용하여 시리즈 데이터 생성

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

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://cdn.jsdelivr.net/npm/danfojs@0.3.3/lib/bundle.min.js"></script>
    <title>Document</title>
</head>
<body>
    <script>
        const tf = dfd.tf //danfo 라이브러리로 부터 tensorflow를 불러올 수 있습니다.
        // 텐서플로우에서 사용되어지는 데이터 타입으로 tensor()를 활용합니다.
        let tensor_arr = tf.tensor([12,34,56,2])
        tensor_arr.print()

        // dfd.Series() 함수를 사용해 생성된 변수를 인스턴스화 할 수 있습니다.
        let s = new dfd.Series(tensor_arr)
        s.print()
    </script>
</body>
</html>
```

결괏값

```
Tensor
    [12, 34, 56, 2]
```

0	12
1	34
2	56
3	2

8-1 Danfo.js

danfo.js 활용하여 데이터 생성하기

JSON 데이터를 인위로 생성하여 이를 dfd.DataFrame( ) 함수를 사용해 데이터 프레임(DataFrame)으로 전환하여 줍니다.

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

```
<body>
    <script>
        // JSON 형태의 2차원 텐서 데이터를 인위적으로 생성하여 주었습니다.
        json_data =
        [{ A: 0.4612, B: 4.28283, C: -1.509, D: -1.1352 },
        { A: 0.5112, B: -0.22863, C: -3.39059, D: 1.1632 },
        { A: 0.6911, B: -0.82863, C: -1.5059, D: 2.1352 },
        { A: 0.4692, B: -1.28863, C: 4.5059, D: 4.1632 }]

        // dfd.DataFrame()를 사용해 데이터 프레임 인스턴스를 생성하여 줍니다.
        df = new dfd.DataFrame(json_data)
        df.print()

        // 데이터 프레임 내에 원하는 값을 선택하여 가져오는 방법입니다.
        console.log(df.$data[0]);
        console.log(df.$data[1][0]);
        console.log(df.$data[3][3]);
    </script>
</body>
```

결괏값

	A	B	C
0	0.4612	4.28283	-1.50
1	0.5112	-0.22863	-3.39
2	0.6911	-0.82863	-1.50
3	0.4692	-1.28863	4.505

```
▶ (4) [0.4612, 4.28283, -1.509, -1.1352]
0.5112
4.1632
```

8-1 Danfo.js

dfd.data_range(), df.head(), df.tail()

범위를 갖는 데이터 생성 및 데이터 일부를 출력하는 함수입니다.

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

```
<body>
    <script>
        // 범위 형식의 날짜를 생성합니다. freq는 연도("Y"), 월("M"), 일("D") 등을 기준으로 나눠줄 수 있습니다.
        dates = new dfd.date_range({ start: '2021-09-01', end: "2021-12-01", freq: "M" })
        console.log(dates);

        // 위에서 생성한 변수를 가지고 다시한번 변수를 생성할 수 있습니다.
        obj_data = {'A': dates,
                    'B': ["bval1", "bval2", "bval3", "bval4"],
                    'C': [10, 20, 30, 40],
                    'D': [1.2, 3.45, 60.1, 45],
                    'E': ["test", "train", "test", "train"]
                    }
        // 이를 데이터 프레임화 합니다.
        df = new dfd.DataFrame(obj_data)
        df.print()

        // head()와 tail()를 사용하여 데이터 일부를 출력합니다.
        df.head(2).print()
        df.tail(2).print()
    </script>
</body>
```

8-1 Danfo.js

describe()를 활용하여 기초통계정보를 확인

https://codepen.io/ALRJM/pen/eYGEVgg?editors=1011

```
<body>
    <script>
        json_data = [{ A: 0.4612, B: 4.28283, C: -1.509, D: -1.1352 },
                     { A: 0.5112, B: -0.22863, C: -3.39059, D: 1.1632 },
                     { A: 0.6911, B: -0.82863, C: -1.5059, D: 2.1352 },
                     { A: 0.4692, B: -1.28863, C: 4.5059, D: 4.1632 }]
        df = new dfd.DataFrame(json_data)
        // describe()를 사용하여 기초통계 정보를 확인할 수 있습니다.
        df.describe().print()
    </script>
</body>
```

기초 통계 정보를 확인할 수 있습니다.

	A	B	C	D
count	4	4	4	4
mean	0.533175	0.4842349999999…	-0.474897500000…	1.5816
std	0.1075428712963…	2.5693167249095…	3.4371471031498…	2.2005448052698…
min	0.4612	-1.28863	-3.39059	-1.1352
median	0.4901999999999…	-0.528629999999…	-1.50745	1.6492
max	0.6911	4.28283	4.5059	4.1632
variance	0.0115654691666…	6.6013884328999…	11.813980208691…	4.84239744

8-1 Danfo.js

sort_values() 활용하여 데이터 정렬하기

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

```
<script>
    let data = {"A": [-20, 30, 47.3, NaN],
         "B": [34, -4, 5, 6] ,
         "C": [20, 2, 3, 30] }
    let df = new dfd.DataFrame(data)
    // sort_values()는 정렬을 수행하여 줍니다.
    // 원하는 id값을 선택(by:"C")하고
    // inplace : true 는 데이터 프레임 자체 내에서
    // 정렬된 상태로 다시 저장
    df.sort_values({by: "C", inplace: true})
    df.print()
</script>
```

	A	B	C
1	30	-4	2
2	47.3	5	3
0	-20	34	20
3	NaN	6	30

8-1 Danfo.js

read_csv()를 활용하여 csv 데이터 불러오기

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

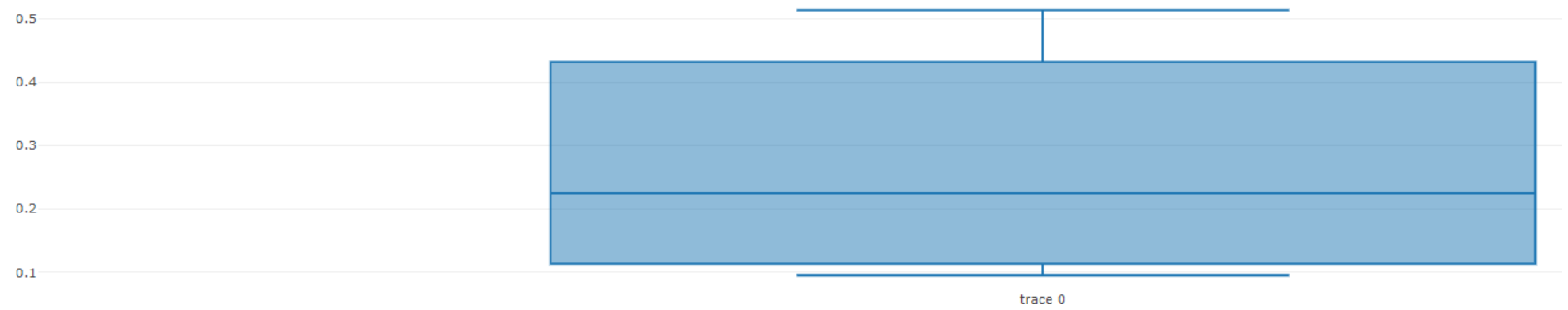
```
<script>
    // read_csv()를 사용해 csv 데이터를 읽어들이는 과정입니다.
     dfd.read_csv('/CODE/data/abalone_mini.csv')
            .then(df => {     // 변수 이름은 df로 지정하였습니다.
            console.log(`df : ${df}`)
            console.log(`df.$columns : ${df.$columns}`)
            //console.log(`df.$data : ${df.$data}`)
            console.log(`df.$data[0] : ${df.$data[0]}`)
            console.log(`df.$data[0].slice(1,9) : ${df.$data[0].slice(1,9)}`)
            console.log(`df.$dtypes : ${df.$dtypes}`)

            // box plot을 그리는 과정입니다.
            data_0 = new dfd.Series(df.$data[0].slice(1,8))
            // 데이터를 변수화하고 box()함수를 plot_div에 출력하여 줍니다.
            data_0.plot('plot_div').box()
        })
</script>
```

데이터 부분 출력

데이터 그래프

8-1 Danfo.js

웹 화면



콘솔창

df :

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.514	0.2245	0.101	0.15	15
1	M	0.35	0.265	0.09	0.2255	0.0995	0.0485	0.07	7
2	F	0.53	0.42	0.135	0.677	0.2565	0.1415	0.21	9
3	M	0.44	0.365	0.125	0.516	0.2155	0.114	0.155	10
4	I	0.33	0.255	0.08	0.205	0.0895	0.0395	0.055	7
5	M	0.5	0.4	0.13	0.6645	0.258	0.133	0.24	12
6	I	0.355	0.28	0.085	0.2905	0.095	0.0395	0.115	7
7	F	0.44	0.34	0.1	0.451	0.188	0.087	0.13	10
8	M	0.365	0.295	0.08	0.2555	0.097	0.043	0.1	7
9	M	0.45	0.32	0.1	0.381	0.1705	0.075	0.115	9

df.$columns : Sex,Length,Diameter,Height,Whole weight,Shucked weight,Viscera weight,Shell weight,Rings

df.$data[0] : M,0.455,0.365,0.095,0.514,0.2245,0.101,0.15,15

df.$data[0].slice(1,9) : 0.455,0.365,0.095,0.514,0.2245,0.101,0.15,15

df.$dtypes : string,float32,float32,float32,float32,float32,float32,float32,int32

8-1 Danfo.js

Danfo JS와 TF JS를 활용하여 타이타닉 생존자 분석 예제

https://danfo.jsdata.org/examples/titanic-survival-prediction-using-danfo.js-and-tensorflow.js