11주차 JavaScript

상태	완료 (승인 X)
설명	객체,이벤트

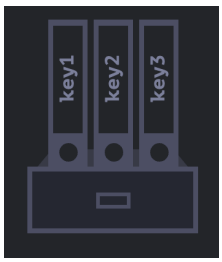
객체

자바스크립트엔 여덟 가지 자료형이 있으며 이 중 일곱 개는 오직 하나의 데이터(문자열, 숫자 등)만 담을 수 있어 '원시형(primitive type)'이라 부른다.

객체형은 원시형과 달리 다양한 데이터를 담을 수 있다. 키로 구분된 데이터 집합이나 복잡한 개체(entity)를 저장할 수 있다. 객체는 자바스크립트 거의 모든 면에 녹아있는 개념이므로 자바스크립트를 잘 다루려면 객체를 잘 이해하고 있어야 한다.

객체는 중괄호 `{...}`를 이용해 만들 수 있다. 중괄호 안에는 **'키(key): 값(value)**' 쌍으로 구성된 *프로퍼티(property)* 를 여러 개 넣을 수 있는데, `키`엔 문자형, `값`엔 모든 자료형이 허용된다. 프로퍼티 키는 '프로퍼티 이름' 이라고도 부른다.

서랍장을 상상하면 객체를 이해하기 쉽다. 서랍장 안 파일은 프로퍼티, 파일 각각에 붙어있는 이름표는 객체의 키라고 생각하면 되며 복잡한 서랍장 안에서 이름표를 보고 원하는 파일을 쉽게 찾을 수 있듯이, 객체에선 키를 이용해 프로퍼티를 쉽게 찾을 수 있다. 추가나 삭제도 마찬가지이다.



> 해당 챕터는 객체를 이리저리 가지고 놀아보면서 alert() 명령어를 직접 활용해서 객체가 어떻게 변하는지 알아보면서 실습을 진행해보자.

빈 객체(빈 서랍장)를 만드는 방법은 두 가지가 있다.

```
let user = new Object(); // '객체 생성자' 문법
let user = {};  // '객체 리터럴' 문법
```

중괄호 `{...}`를 이용해 객체를 선언하는 것을 *객체 리터럴(object literal)* 이라고 부른다. 객체를 선언할 땐 주로 이 방법을 사용하게 된다.

중괄호 `{...}` 안에는 '키: 값' 쌍으로 구성된 프로퍼티가 들어가게 된다.

```
let user = {     // 객체
  name: "John",  // 키: "name",  값: "John"
  age: 30        // 키: "age", 값: 30
};
```

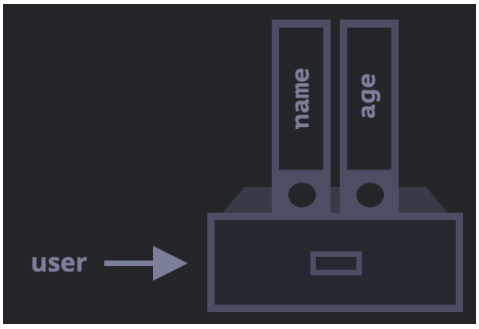
`:`을 기준으로 왼쪽엔 키가, 오른쪽엔 값이 위치한다.

현재 객체 `user`에는 프로퍼티가 두 개 있는 것이다.

1️⃣ 첫 번째 프로퍼티 – `"name"`(이름)과 `"John"`(값)

2️⃣ 두 번째 프로퍼티 – `"age"`(이름)과 `30`(값)

서랍장(객체 `user`) 안에 파일 두 개(프로퍼티 두 개)가 담겨있는데, 각 파일에 "name", "age"라는 이름표가 붙어 있다고 생각하시면 조금더 이해하기 쉽다.



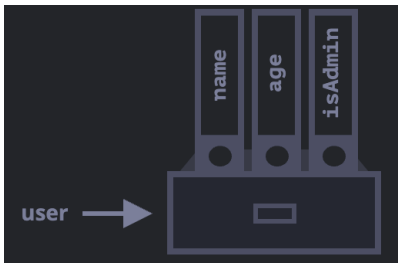
서랍장에 파일을 추가하고 뺄 수 있듯이 개발자는 프로퍼티를 추가, 삭제할 수 있다.

점 표기법(dot notation)을 이용하면 프로퍼티 값을 읽는 것도 가능하다.

```
// 프로퍼티 값 얻기
alert( user.name ); // John
alert( user.age ); // 30
```

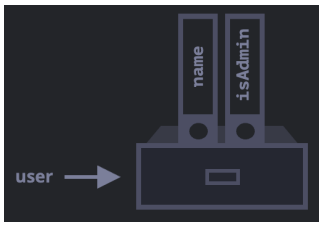
프로퍼티 값엔 모든 자료형이 올 수 있으며 불린(참or거짓)형 프로퍼티를 추가해 보자.

```
user.isAdmin = true;
```



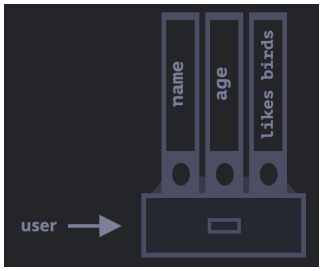
`delete` 연산자를 사용하면 프로퍼티를 삭제할 수 있다.

```
delete user.age;
```



여러 단어를 조합해 프로퍼티 이름을 만든 경우엔 프로퍼티 이름을 따옴표로 묶어줘야 한다.

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true  // 복수의 단어는 따옴표로 묶어야 합니다.
};
```



마지막 프로퍼티 끝은 쉼표로 끝날 수 있다.

```
let user = {
  name: "John",
  age: 30,}
```

굳이 마지막 항목임에도 `,`를 붙일 수 있는 이유는, 모든 프로퍼티의 형태가 똑같아지기 때문이다. 물론 쉼표를 달지 않아도 객체를 사용하는데 문제가 없다.

💡 const로 선언한 객체

`const`로 선언된 객체를 수정할 수 있다.

```
const user = {
  name: "John"
};

user.name = "Pete"; // (*)alert(user.name); // Pete
```

`(*)` 줄에서 오류가 나와야 할 것 같지만 그렇지 않다. 해당 이유는 `const`는 `user`라는 변수의 값 자체를 갈아끼우는 상황에서만 보호를 해주기 때문이다.

우선은 `const`로 객체를 만들어도 그 안의 값을 수정할 수 있다는 사실만 알고 넘어가자.

대괄호 표기법

여러 단어를 조합해 프로퍼티 키를 만든 경우엔, 점 표기법을 사용해 프로퍼티 값을 읽을 수 없다.

```
// 문법 에러가 발생합니다.
user.likes birds = true
```

자바스크립트가 `likes`까지만 이해할 수 있기 때문, 객체에서는 값을 조회할때 `.`을 사용하는 방법 외에도 `배열(리스트)` 에서 값을 가져올때처럼 `[ ]` 를 활용하는 방법이 있다.

```
let user = {};

// set
user["likes birds"] = true;

// get
alert(user["likes birds"]); // true

// delete
delete user["likes birds"];
```

따옴표는 `'` `"` 아무거나 사용이 가능하다.

```
let key = "likes birds";
```

```
// user["likes birds"] = true; 와 같습니다.
user[key] = true;
```

해당 문법을 사용하면 [ ] 안에 변수를 넣는것도 가능하기 때문에 동적으로 객체를 다루는 것이 가능하다.

```
let user = {
  name: "John",
  age: 30
};

let key = prompt("사용자의 어떤 정보를 얻고 싶으신가요?", "name");

// 변수로 접근
alert( user[key] ); // John (프롬프트 창에 "name"을 입력한 경우)
```

그런데 점 표기법은 이런 방식이 불가능하다. 다만 점 표기법을 활용하면 사용할때 보다 간편하니 상황에 맞게 이를 사용하면 된다.

```
let user = {
  name: "John",
  age: 30
};

let key = "name";
alert( user.key ) // undefined
```

단축 프로퍼티

프로퍼티 값을 기존 변수에서 받아와 사용하는 경우가 생각보다 많다.

```
function makeUser(name, age) {
  return {
    name: name,
    age: age,
    // ...등등
  };
}

let user = makeUser("John", 30);
alert(user.name); // John
```

위 예시의 프로퍼티들은 이름과 값이 변수의 이름과 동일한데, 이런 경우 단축 구문을 활용하면 코드를 짧게 줄이는 것이 가능하다.

`name:name` 대신 `name` 만 적어주어도 동작하기 때문이다.

```
function makeUser(name, age) {
  return {
    name, // name: name 과 같음
    age,  // age: age 와 같음
    // ...
  };}
```

한 객체에서 일반 프로퍼티와 단축 프로퍼티를 함께 사용하는 것도 가능하다.

```
let user = {
  name,  // name: name 과 같음
  age: 30
};
```

프로퍼티 이름 제약

변수 이름(키)엔 ‘for’, ‘let’, ‘return’ 같은 예약어를 사용하면 안된다.

그런데 객체 프로퍼티엔 이런 제약이 없다.

```
// 예약어를 키로 사용해도 괜찮습니다.
let obj = {
  for: 1,
  let: 2,
  return: 3
};

alert( obj.for + obj.let + obj.return );  // 6
```

문자형이 아닌, 어떤 값이 들어가도 되며, 다만 들어간 값이 문자열로 변환되서 `key`로 사용되게 된다.

```
let obj = {
  0: "test" // "0": "test"와 동일합니다.
};

// 숫자 0은 문자열 "0"으로 변환되기 때문에 두 얼럿 창은 같은 프로퍼티에 접근합니다,
alert( obj["0"] ); // test
alert( obj[0] ); // test (동일한 프로퍼티)
```

in 연산자

자바스크립트 객체의 중요한 특징 중 하나는 다른 언어와는 달리, 존재하지 않는 프로퍼티에 접근하려 해도 에러가 발생하지 않고 `undefined`를 반환하는 것이다.

이런 특징을 응용하면 프로퍼티 존재 여부를 쉽게 확인할 수 있다.

```
let user = {};

alert( user.noSuchProperty === undefined ); // true는 '프로퍼티가 존재하지 않음'을 의미합니다.
```

이렇게 `undefined`와 비교하는 것 이외에도 연산자 `in`을 사용하면 조금더 직관적으로 프로퍼티의 존재 유무를 확인할 수 있다.

```
"key" in object
```

```
let user = { name: "John", age: 30 };

alert( "age" in user ); // user.age가 존재하므로 true가 출력됩니다.
alert( "blabla" in user ); // user.blabla는 존재하지 않기 때문에 false가 출력됩니다.
```

💡 **`undefined`**랑 비교하지 않고 **`in`** 연산자를 사용하는 이유 ❓

"`undefined`랑 비교해도 충분한데 왜 `in` 연산자가 있는 거지?"라는 의문이 들 수 있다.

대부분의 경우, 일치 연산자를 사용해서 프로퍼티 존재 여부를 알아내는 방법(`"=== undefined"`)은 꽤 잘 동작한다. 하지만 예외의 상황이 항상 존재한다. 아래 상황을 한번 봐보자.

```
let obj = {
  test: undefined
};

alert( obj.test ); // 값이 `undefined`이므로, 얼럿 창엔 undefined가 출력됩니다. 그런데 프로퍼티 test는 존재합니다.

alert( "test" in obj ); // `in`을 사용하면 프로퍼티 유무를 제대로 확인할 수 있습니다(true가 출력됨).
```

해당 **`키(key)`**의 **`값(value)`** 이 **`undefined`**인경우는 많지 않지만, 위와같은 경우가 나올 수 있어 **`in`** 연산자가 존재하게 된다.

for … in

`for..in` 반복문을 사용하면 객체의 모든 키를 순회할 수 있다. `for..in`은 앞서 학습했던 `for(;;)` 반복문과는 완전히 다르며, `for..of` 반복문과 조금 유사하다.

```
for (key in object) {
  // 각 프로퍼티 키(key)를 이용하여 본문(body)을 실행합니다.
}
```

아래 예시를 실행하면 객체 `user`의 모든 프로퍼티가 출력된다.

```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // 키
  alert( key );  // name, age, isAdmin
  // 키에 해당하는 값
  alert( user[key] ); // John, 30, true
}
```

이전에 배웠던 `for..of` 반복문과 상당히 유사한데, 반복문으로 `key`의 값을 가지고 온다는 것이 다르다.

요약

객체는 몇 가지 특수한 기능을 가진 연관 배열(associative array)이다.

객체는 프로퍼티(키-값 쌍)를 저장한다.

- 프로퍼티 키는 문자열이나 심볼이어야 하며 보통 문자열이다.
- 값은 어떤 자료형도 가능하다.

아래와 같은 방법을 사용하면 프로퍼티에 접근할 수 있다.

- 점 표기법: `obj.property`
- 대괄호 표기법 `obj["property"]`. 대괄호 표기법을 사용하면 `obj[varWithKey]` 같이 변수에서 키를 가져올 수 있다.

객체엔 다음과 같은 추가 연산자를 사용할 수 있다.

- 프로퍼티를 삭제하고 싶을 때: `delete obj.prop`
- 해당 key를 가진 프로퍼티가 객체 내에 있는지 확인하고자 할 때: `"key" in obj`
- 프로퍼티를 나열할 때: `for (let key in obj)`

문제 1

다음 각 동작을 한 줄씩, 코드로 작성해보세요.

1️⃣ 빈 객체 `user`를 만듭니다.

2️⃣ `user`에 키가 `name`, 값이 `John`인 프로퍼티를 추가하세요.

3️⃣ `user`에 키가 `surname`, 값이 `Smith`인 프로퍼티를 추가하세요.

4️⃣ `name`의 값을 `Pete`로 수정해보세요.

5️⃣ `user`에서 프로퍼티 `name`을 삭제하세요.

문제 2

객체에 프로퍼티가 하나도 없는 경우 `true`, 그렇지 않은 경우 `false`를 반환해주는 함수 `isEmpty(obj)`를 만들어 보세요.

아래와 같이 동작해야 합니다.

```
let schedule = {};

alert( isEmpty(schedule) ); // true

schedule["8:30"] = "get up";

alert( isEmpty(schedule) ); // false
```

문제 3

`const`와 함께 선언한 객체를 변경하는 게 가능할까요? 생각해보고 코드를 실행해봅시다.

```
const user = {
  name: "John"
};
// 아래 코드는 에러 없이 실행될까요?
user.name = "Pete";
```

문제 4

모든 팀원의 월급에 대한 정보를 담고 있는 객체가 있다고 해봅시다.

```
let salaries = {
  John: 100,
  Ann: 160,
  Pete: 130
}
```

모든 팀원의 월급을 합한 값을 구하고, 그 값을 변수 `sum`에 저장해주는 코드를 작성해보세요. `sum`엔 `390`이 저장되어야겠죠?

주의: `salaries`가 비어있다면 `sum`에 `0`이 저장되어야 합니다.

문제 5

객체 `obj`의 프로퍼티 값이 숫자인 경우 그 값을 `두 배` 해주는 함수 `multiplyNumeric(obj)`을 만들어보세요.

```
// 함수 호출 전
let menu = {
  width: 200,
  height: 300,
  title: "My menu"
};

multiplyNumeric(menu);

// 함수 호출 후
menu = {
  width: 400,
  height: 600,
  title: "My menu"
};
```

`multiplyNumeric`은 아무것도 반환하지 않아도 괜찮습니다. 객체 자체를 수정해주기만 하면 됩니다.

브라우저 환경과 다양한 명세서

자바스크립트는 본래 웹 브라우저에서 사용하려고 만든 언어이다. 따라서 브라우저 환경에서 지원하는 다양한 객체들이 존재한다.

아래 예시에선 `window` 객체를 전역 객체로 사용하고 있다.

```
function sayHi() {
  alert("안녕하세요.");
}

// 전역 함수는 전역 객체(window)의 메서드임
window.sayHi();
```

아래 예시에선 `window` 객체가 브라우저 창을 대변하고 있으며, 이를 이용해 창의 높이를 출력한다.

```
alert(window.innerHeight); // 창 내부(inner window) 높이
```

문서객체모델 (DOM)

문서 객체 모델(Document Object Model, DOM)은 웹 페이지 내의 모든 콘텐츠를 객체로 나타내준다. 조금 쉽게 설명하자면, DOM이 웹페이지 자체라고 생각해도 괜찮다.

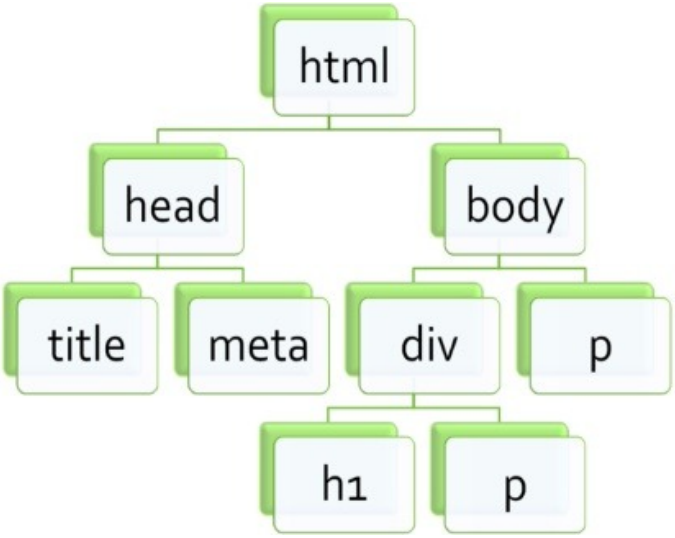
`document` 객체는 페이지의 기본 '진입점' 역할을 하며 `document` 객체를 이용해 페이지 내 그 무엇이든 변경할 수 있고, 원하는 것을 만들 수도 있다.

```
// 배경을 붉은색으로 변경하기
document.body.style.background = "red";

// 1초 후 원상태로 복구하기
setTimeout(() => document.body.style.background = "", 1000);
```

문서 객체 모델은 예시에서 소개한 `document.body.style` 외에도 수많은 기능을 제공한다.

자세히 파고들어가면 어려운 내용이니 브라우저 구조가 모두 객체로 이루어져 있고, 아래와 같이 트리 구조를 가진다는 것만 생각하고 넘어가자.



id 속성을 사용해서 요소 검색하기

요소에 `id` 속성이 있으면 위치에 상관없이 메서드 `document.getElementById(id)`를 이용해 접근할 수 있다.

```
<div id="elem">
  <div id="elem-content">Element</div>
</div>
<script>
  // 요소 얻기
  let elem = document.getElementById('elem');// 배경색 변경하기
  elem.style.background = 'red';
</script>
```

id는 중복되면 안 된다.

`id`는 유일무이해야 한다. 문서 내 요소의 `id` 속성값은 중복되어선 안 된다.

같은 `id`를 가진 요소가 여러 개 있으면 `document.getElementById`같이 `id`를 이용해 요소를 검색하는 메서드의 동작이 예측 불가능해지게 된다.

querySelectorAll

`elem.querySelectorAll(css)`은 다재다능한 요소 검색 메서드이다. 이 메서드는 `elem`의 자식 요소 중 주어진 CSS 선택자에 대응하는 요소 모두를 반환한다.

```
<ul>
  <li>1-1</li>
  <li>1-2</li>
</ul>
<ul>
  <li>2-1</li>
  <li>2-2</li>
</ul>
<script>
  let elements = document.querySelectorAll("ul > li:last-child");
  for (let elem of elements) {
    alert(elem.innerHTML); // "1-2", "2-2"
  }
</script>
```

`querySelectorAll`은 CSS 선택자를 활용할 수 있다는 점에서 아주 유용하다.

querySelector

`elem.querySelector(css)`는 주어진 CSS 선택자에 대응하는 요소 중 첫 번째 요소를 반환한다.

`elem.querySelectorAll(css)[0]`과 동일하지만 처음 요소만 찾을때 바로 반환하기 때문에 속도가 더빠르다는 장점이 존재한다.

브라우저 이벤트

이벤트(event) 는 무언가 일어났다는 신호이며 모든 DOM 노드는 이런 신호를 만들어 낼 수 있다.

자주 사용되는 유용한 DOM 이벤트는 무엇이 있는지 잠시 살펴보자.

마우스 이벤트:

- `click` – 요소 위에서 마우스 왼쪽 버튼을 눌렀을 때(터치스크린이 있는 장치에선 탭 했을 때) 발생
- `contextmenu` – 요소 위에서 마우스 오른쪽 버튼을 눌렀을 때 발생
- `mouseover`와 `mouseout` – 마우스 커서를 요소 위로 움직였을 때, 커서가 요소 밖으로 움직였을 때 발생
- `mousedown`과 `mouseup` – 요소 위에서 마우스 왼쪽 버튼을 누르고 있을 때, 마우스 버튼을 뗄 때 발생
- `mousemove` – 마우스를 움직일 때 발생

폼 요소 이벤트:

- `submit` – 사용자가 `<form>`을 제출할 때 발생
- `focus` – 사용자가 `<input>`과 같은 요소에 포커스 할 때 발생

키보드 이벤트:

- `keydown`과 `keyup` – 사용자가 키보드 버튼을 누르거나 뗄 때 발생

문서 이벤트:

- `DOMContentLoaded` – HTML이 전부 로드 및 처리되어 DOM 생성이 완료되었을 때 발생

CSS 이벤트:

- `transitionend` – CSS 애니메이션이 종료되었을 때 발생

이벤트 핸들러

이벤트에 반응하려면 이벤트가 발생했을 때 실행되는 함수인 *핸들러(handler)* 를 할당해야 한다.

HTML 속성

HTML 안의 `on<event>` 속성에 핸들러를 할당할 수 있으며 이벤트가 발생하는 경우 핸들러가 실행된다.

아래는 `input` 태그의 `onclick` 속성에 `click` 핸들러를 할당하는 예시이다.

```
<input value="클릭해 주세요." onclick="alert('클릭!')" type="button">
```

버튼을 클릭하면 `onclick` 안의 코드가 실행된다.

여기서 주의해야 할 것은 속성값 내에서 사용된 따옴표이며. 속성값 전체가 큰따옴표로 둘러싸여 있기 때문에 작은 따옴표로 둘러쌓아야한다. `onclick="alert("클릭!")"`과 같이 속성값 내부에 또 큰따옴표를 쓰면 코드가 작동하지 않게 되니 주의하자.

긴 코드를 HTML 속성값으로 사용하는 것은 추천하지 않는다.. 만약 코드가 길다면, 함수를 만들어서 이를 호출하는 방법을 사용해야 한다.

아래 버튼을 클릭하면 함수 `countRabbits()`이 호출된다.

```
<script>
  function countRabbits() {
    for(let i=1; i<=3; i++) {
      alert(`토끼 ${i}마리`);
    }
  }
</script>
<input type="button" onclick="countRabbits()" value="토끼를 세봅시다!">
```

this 사용해보기

핸들러 내부에 쓰인 `this`의 값은 핸들러가 할당된 요소이다.

아래 예시의 `this.innerHTML`에서 `this`는 `button`이므로 버튼을 클릭하면 버튼 안의 콘텐츠가 `alert`창에 출력되게 된다.

```
<button onclick="alert(this.innerHTML)">클릭해 주세요.</button>
```

이벤트 객체

`click` 이벤트가 발생했다면 마우스 포인터가 어디에 있는지, `keydown` 이벤트가 발생했다면 어떤 키가 눌렸는지 등에 대한 상세한 정보가 필요할때가 있다.

이벤트가 발생하면 브라우저는 `이벤트 객체(event object)`라는 것을 만드는데 여기에 이벤트에 관한 상세한 정보를 넣은 다음, 핸들러에 인수 형태로 전달하기에 위에서 말한 정보들을 얻어낼 수 있다.

아래는 이벤트 객체로부터 포인터 좌표 정보를 얻어내는 예시이다.

```
<input type="button" value="클릭해 주세요." id="elem"><script>
  elem.onclick = function(event) {
    // 이벤트 타입과 요소, 클릭 이벤트가 발생한 좌표를 보여줌
    alert(event.type + " 이벤트가 " + event.currentTarget + "에서 발생했습니다.");
    alert("이벤트가 발생한 곳의 좌표는 " + event.clientX + ":" + event.clientY +"입니다.");
  };
</script>
```

💡 문제 1

`button`을 클릭하면 `<div id="text">`가 사라지도록 `button`에 자바스크립트를 추가해봅시다.

```
<!DOCTYPE HTML>
<html>

<head>
  <meta charset="utf-8">
</head>

<body>

  <input type="button" id="hider" value="Text를 숨기려면 클릭하세요." />

  <div id="text">Text</div>

  <script>
    /* 코드를 여기에 작성하세요 */
  </script>

</body>
</html>
```

브라우저와 이벤트 부분은 자세히 이해하려면 굉장히 복잡하기에 아주 단순하게만 알아보고, 다음 시간부터 진행될 컴세바 페이지 제작수업을 통해 천천히 이해해보자.