ch07. 자바스크립트(2)

학습 목차

1. DOM(Document Object Model) 기본 개념
2. 요소 노드 선택
3. 노드 탐색
4. 요소 노드의 텍스트 조작
5. attribute 조작
6. 스타일 조작
7. 클래스 조작
8. DOM 조작
9. 이벤트
10. 이벤트 핸들러 등록
11. 이벤트 핸들러 제거
12. 이벤트 활용
13. 이벤트의 디폴트 행동 취소
14. 이벤트 흐름
15. 웹 브라우저 객체 모델

학습 목표

- DOM 기본 개념을 이해할 수 있습니다.
- DOM 트리 구성요소를 이해할 수 있습니다.
- 요소 노드를 탐색하고 조작할 수 있습니다
- 요소 노드를 생성하고 추가할 수 있습니다.
- 이벤트 등록 방법을 이해하고 웹 문서에 활용하여 다양한 동적 효과를 구현할 수 있습니다
- 윈도우 객체를 이해하고 활용할 수 있습니다

1. DOM(Document Object Model) 기본 개념

- DOM
  - HTML 모든 요소와 요소의 어트리뷰트, 텍스트를 각각의 객체로 만들고 이들 객체의 부자 관계를 표현할 수 있는 트리 구조로 구성한 것(DOM tree)
  - 기능
    - HTML 문서에 대한 모델 구성
    - HTML 문서 내의 각 요소에 접근/수정

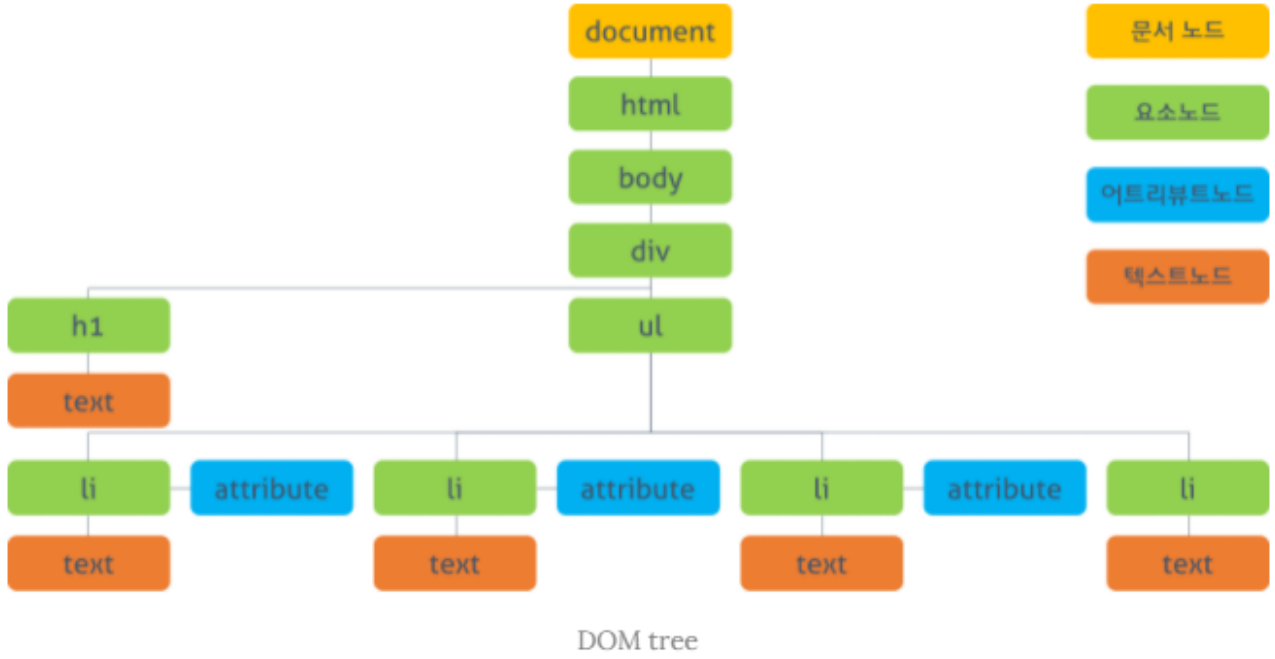
- DOM API
  - DOM에 접근하고 변경하는 프로퍼티와 메소드 집합

- DOM은 자바스크립트를 통해 동적으로 변경 & 렌더링에 반영

1. DOM(Document Object Model) 기본 개념

- DOM 트리

```
<html lang="en">
  <body>
    <div>
      <h1>Cities</h1>
      <ul>
        <li id="one" class="red">hallym</li>
        <li id="two" class="red">software</li>
        <li id="three" class="red">big data</li>
        <li id="four">IoT</li>
      </ul>
    </div>
  </body>
</html>
```



DOM tree

1. DOM(Document Object Model) 기본 개념

- DOM 트리 노드 종류
  - 문서 노드(document node)
    - 트리의 최상위에 존재, DOM tree에 접근하기 위한 시작점(entry point)
  - 요소 노드(element Node)
    - HTML 요소 표현
    - 어트리뷰트, 텍스트 노드에 접근하려면 먼저 요소 노드를 찾아 접근
  - 어트리뷰트 노드(Attribute Node)
    - HTML 요소의 어트리뷰트, 해당 요소 노드를 찾아 접근하면 참조, 수정 가능
  - 텍스트 노드(Text Node)
    - HTML 요소의 텍스트 표현, 요소 노드의 자식이며 자식 노드를 가질 수 없다
- DOM을 통한 웹 페이지 조작(manipulate)
  - 조작하고자하는 요소를 선택 또는 탐색
  - 선택된 요소의 텍스트 또는 어트리뷰트 조작

2. 요소 노드 선택

- HTML 구조나 내용 또는 스타일 등을 동적으로 조작하려면 먼저 요소 노드 선택
  - 문서 노드인 document를 통하여 메소드 호출

document.

- 하나의 요소 노드 선택

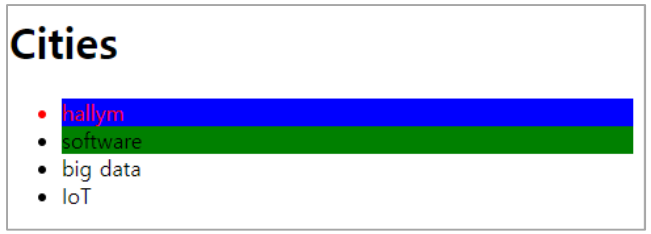
메소드	기능
getElementById(id)	• 태그의 id속성과 매개변수 id속성이 일치하는 한 개의 요소 노드 반환 • 요소 노드가 여러 개 이면 첫 번째 요소만 반환 • 일치하는 요소가 없으면 null 반환
querySelector(cssSelector)	• CSS 셀렉터를 만족하는 한 개의 요소 노드 반환 • 요소 노드가 여러 개 이면 첫 번째 요소만 반환 • 일치하는 요소가 없으면 않으면 null 반환

getElementById(id)
Id 선택자.

querySelector(cssSelector)
CSS 선택자.

2. 요소 노드 선택

- 하나의 요소 노드 선택




```
<head>
  <script>
    window.onload=function(){
      // id 어트리뷰트 값이 'one'인 하나의 요소를 선택한다.
      const elem = document.getElementById('one');
      //스타일 변경
      elem.style.background ='blue';
      elem.style.color ='red';

      //css 셀렉터를 사용하여 하나의 요소를 선택한다
      const demem  = document.querySelector('.red');
      demem.style.background ='green';
    }
  </script>
</head>
<body>
  <div>
    <h1>Cities</h1>
    <ul>
      <li id="one">hallym</li>
      <li id="two" class="red">software</li>
      <li id="three" class="red">big data</li>
      <li id="four">IoT</li>
    </ul>
  </div>
</body>
```

2. 요소 노드 선택

- 여러 개의 요소 노드 선택

메소드	기능
getElementsByName(name)	• name어트리뷰트 값으로 요소 노드를 모두 선택
querySelectorAll(cssSelector)	• 지정된 CSS 선택자를 사용하여 요소 노드를 모두 선택 • NodeList 객체 반환 – non live
getElementsByClassName(class)	• class 어트리뷰트 값으로 요소 노드를 모두 선택 • HTMLCollection 객체 반환 – live
getElementsByTagName(tagname)	• 태그명으로 요소 노드를 모두 선택 • HTMLCollection 객체 반환 - live

- NodeList & HTMLCollection
  - DOM API가 여러 개의 결과값을 반환하기 위한 컬렉션
  - for...of 사용 가능
  - 스프레드 문법을 사용 배열로 변환 가능
- 노드 객체의 상태 변경과 상관 없이 안전하게 DOM 컬렉션 사용 => 배열로 변환

2. 요소 노드 선택

- 여러 개의 요소 노드 선택

```
<head>
  <script>
    window.onload = function () {
      // HTMLCollection 반환(유사 배열), HTMLCollection은 실시간으로 노드 상태 변경
      const elems =document.getElementsByClassName('red');

      for (let i = 0; i < elems.length; i++) {
            elems[i].className = 'blue';  // 클래스 어트리뷰트의 값을 변경한다.
      }
    }
  </script>
</head>
<body>
  <div>
    <h1>Cities</h1>
    <ul>
      <li class="red">hallym</li>
      <li class="red">software</li>
      <li class="red">big data</li>
      <li id="four">IoT</li>
    </ul>
  </div>
</body>
```

```
// 유사 배열 객체인 HTMLCollection을 배열로 변환한다.
const arr=[...elems]; //const arr=Array.from(elems);
arr.forEach(value => value.className = 'blue');
console.log(arr);
```

실시간으로 변경됨

```
▼<div>
   <h1>Cities</h1>
  ▼<ul>
    ▶<li id="one" class="blue">…</li>
    ▶<li id="two" class="red">…</li>
    ▶<li id="three" class="blue">…</li>
    ▶<li id="four">…</li>
   </ul>
 </div>
```

```
▼<div>
   <h1>Cities</h1>
  ▼<ul>
    ▶<li id="one" class="blue">…</li>
    ▶<li id="two" class="blue">…</li>
    ▶<li id="three" class="blue">…</li>
    ▶<li id="four">…</li>
   </ul>
 </div>
```

2. 요소 노드 선택

- 여러 개의 요소 노드 선택

Non-live

```
<head>
  <script>
    window.onload = function () {
      // NodeList 객체 반환, 실시간으로 노드 객체 상태 변경 반영하지 않음
      const elems =document.querySelectorAll('.red');
      for (let i = 0; i < elems.length; i++) {
        elems[i].className = 'blue';
      }
    }
  </script>
</head>
<body>
  <div>
    <h1>Cities</h1>
    <ul>
      <li class="red">hallym</li>
      <li class="red">software</li>
      <li class="red">big data</li>
      <li id="four">IoT</li>
    </ul>
  </div>
</body>
```

```
▼<div>
    <h1>Cities</h1>
  ▼<ul>
    ▶<li id="one" class="blue">…</li>
    ▶<li id="two" class="blue">…</li>
    ▶<li id="three" class="blue">…</li>
    ▶<li id="four">…</li>
    </ul>
  </div>
```

```
<script>
    window.onload = function () {
      const elems =document.getElementsByTagName('li');

      for(let value of elems){
        value.style.color ='blue';
      }
    }
</script>
```

Cities

- hallym
- software
- big data
- IoT

3. 노드 탐색

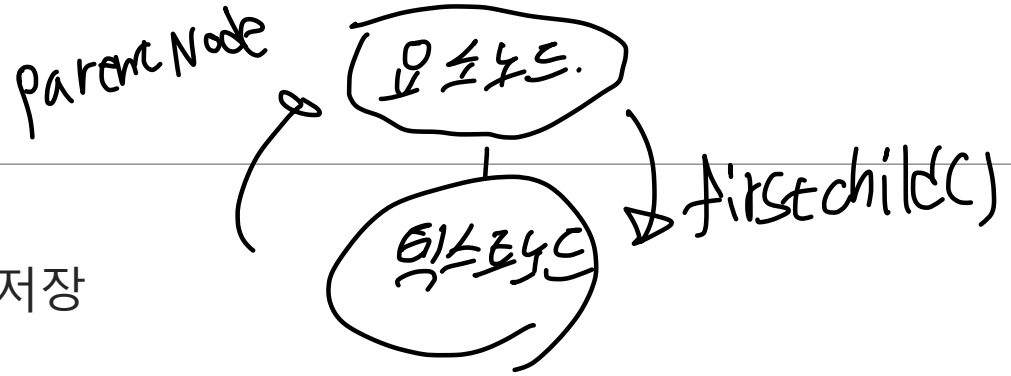
- 노드 탐색 프로퍼티는 읽기 전용
- 공백 텍스트 노드
  - HTML 요소 사이의 스페이스, 탭, 줄 바꿈 등의 공백 문자는 텍스트 노드 생성
  - 노드 탐색 시 공백 문자가 생성한 텍스트 노드에 주의

- 자식 노드 탐색 프로퍼티
  - firstChild, lastChild
    - 첫번째, 마지막 자식 노드 반환 => 반환된 노드는 텍스트 노드이거나 요소 노드
  - childNodes
    - 모든 자식 노드 탐색하여 반환 => 요소 노드, 텍스트 노드, 주석노드까지 접근
  - children
    - 자식 노드 중 요소 노드만 모두 탐색하여 반환 => 텍스트 노드는 포함되지 않는다
  - firstElementChild, lastElementChild,
    - 첫번째 자식 노드와 마지막 자식 노드 반환 => 요소 노드만 반환

3. 노드 탐색

- 자식 노드 존재 확인
  - hasChildNodes() 메소드
    - 자식 노드가 존재하면 true, 존재하지 않으면 false 반환
- 부모 노드 탐색 프로퍼티
  - parentNode
    - 현재 노드의 부모 요소 노드 반환
- 형제 노드 탐색 프로퍼티
  - previousSibling
    - 형제 노드 중에서 자신의 이전 형제 노드 반환 => 요소 노드이거나 텍스트 노드
  - nextSibling
    - 형제 노드 중에서 자신의 다음 형제 노드 반환 => 요소 노드이거나 텍스트 노드
  - previousElementSibling
    - 형제 노드 중에서 자신의 이전 형제 노드 반환 => 요소 노드만 반환
  - nextElementSibling
    - 형제 노드 중에서 자신의 다음 형제 노드 반환 => 요소 노드만 반환

3. 노드 탐색



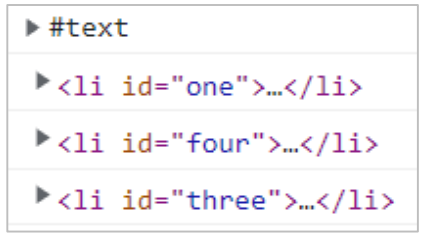
- 요소 노드의 텍스트 탐색
  - 요소의 텍스트는 텍스트 노드에 저장
  - 접근/수정 방법
    - 해당 텍스트노드의 부모 노드 선택
    - firstChild 속성을 사용하여 텍스트 노드 탐색
- 노드 정보 취득
  - nodeType
    - 노드 타입을 나타내는 상수 반환 – 1: element node, 2:attribute, 3:text
  - nodeName
    - 노드 이름을 문자열로 반환

3. 노드 탐색

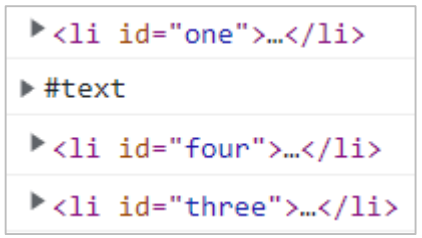
```
<body>
  <div>
    <h1>Cities</h1>
    <ul>
      <li id="one" class="red">hallym</li>
      <li id="two" class="red">software</li>
      <li id="three" class="red">big data</li>
      <li id="four">IoT</li>
    </ul>
  </div>
<script>
    const elem1 = document.querySelector('#two');
    console.log('===부모 노드 이름 ===');
    console.log(elem1.parentNode.nodeName); //nodeName 프로퍼티 : 노드 이름

    const elem2 = document.querySelector('ul');
    if (elem2.hasChildNodes()) {
      console.log("===자식요소 & 텍스트 요소===");
      console.log(elem2.childNodes); // 텍스트 요소를 포함한 모든 자식 요소를 반환한다.

      console.log("===자식요소===");
      console.log(elem2.children);  // 자식 요소 중에서 element type 요소만을 반환한다.

      [...elem2.children].forEach(el => console.log(el.nodeName)); // LI
    }
  </script>
  </body>
```

3. 노드 탐색

```
<div>
    <h1>University</h1>
    <ul>
      <li id="one">hallym</li>
      <li id="two">software</li>
      <li id="three">big data</li>
      <li id="four">content</li>
    </ul>
  </div>
  <script>
    const enode=document.querySelector('ul');
    const {firstChild} = enode;  //첫번째 자식 노드
    const {nextSibling} =firstChild;  //첫번째 자식 노드의 다음 형제 노드
    const {lastElementChild} = enode; //마지막 자식 노드
    const {previousElementSibling} = lastElementChild; //마지막 자식노드의 이전 형제 노드

    console.log(firstChild);
    console.log(nextSibling);
    console.log(lastElementChild);
    console.log(previousElementSibling);
  </script>
</body>
```



firstChid -> firstElementChild로 하면



1. 노드 반환 시 요소 노드만 반환하는지
2. 텍스트 노드 또는 요소 노드를 반환하는지를 구분해야 함

4. 요소 노드의 텍스트 조작

* 텍스트조작

- HTML 텍스트 조작(Manipulation) 프로퍼티
  - getter, setter 모두 존재
  - textContent
    - 요소의 텍스트 콘텐츠를 취득 또는 변경, 마크업 무시
    - 순수한 텍스트만 지정해야 하며 마크업을 포함시키면 문자열로 인식되어 그대로 출력
  - nodeValue
    - 노드의 값 반환
      - 텍스트노드의 경우 문자열, 요소 노드의 경우 null 반환
    - 요소 노드 텍스트 변경
      - 요소 노드 선택 -> firstChild 프로퍼티를 사용하여 텍스트 노드 탐색 -> 텍스트 변경

4. 요소 노드의 텍스트 조작

```
<body>
  <div>
    <h1>Cities</h1>
    <ul>
      <li id="one" class="red">hallym</li>
      <li id="two" class="red">software</li>
      <li id="three" class="red">big data</li>
      <li id="four">IoT</li>
    </ul>
  </div>
  <script>
   /* 요소 노드의 텍스트 변경 */
    const one = document.getElementById('one'); // 1. 해당 텍스트 노드의 부모 요소 노드를 선택한다
    const textNode = one.firstChild; //2. firstChild 프로퍼티를 사용하여 텍스트 노드를 탐색한다.
    textNode.nodeValue = 'University'; //3. nodeValue 프로퍼티를 이용하여 텍스트를 수정한다.


     // nodeValue 프로퍼티를 사용하여 노드의 값을 취득한다.
    console.log(textNode.nodeValue); // University

    // nodeName, nodeType을 통해 노드의 정보를 취득할 수 있다.
    console.log(one.nodeName); // LI
    console.log(one.nodeType); // 1: element node, 2:attribute ,3:text

  </script>
</body>
```

Cities

- University
- software
- big data
- IoT

University
LI
1

4. 요소 노드의 텍스트 조작

```
<head>
  <script>
    window.onload = function () {
      const ul = document.querySelector('ul');
      // 요소의 텍스트 취득
      console.log(ul.textContent);

      const one = document.getElementById('one');
      // 요소의 텍스트 취득
      console.log(one.textContent);

      // 요소의 텍스트 변경
      one.textContent += ', University';
      console.log(one.textContent);

      // 요소의 마크업이 포함된 콘텐츠 변경.
      one.textContent = '<h1>Heading</h1>';
      // 마크업이 문자열로 표시된다.
      console.log(one.textContent);
    }
  </script>
</head>
```

이렇게하면 구체적 first child 까지 자식에 접근할 필요 없는 거임.

```
<body>
  <div>
    <h1>Cities</h1>
    <ul>
      <li id="one" class="red">hallym</li>
      <li id="two" class="red">software</li>
      <li id="three" class="red">big data</li>
      <li id="four">IoT</li>
    </ul>
  </div>
</body>
```

Cities

- <h1>Heading</h1>
- software
- big data
- IoT

```
hallym
software
big data
IoT
```

```
hallym
hallym, University
<h1>Heading</h1>
```

5. attribute 조작

- 요소 노드의 모든 attribute는 attributes 프로퍼티로 취득
  - getter 만 존재, 값 변경 불가
- attribute 조작 메소드

메소드	설명
setAttribute(attribute, value)	속성 지정
getAttribute(attribute)	속성 추출
removeAttribute(attribute)	속성 제거
hasAttribute(attribute)	존재 확인

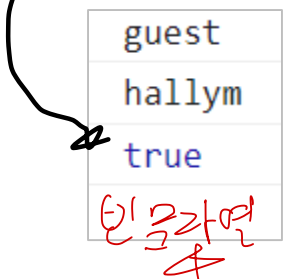
```
<body>
  <input type="text" id="user" value="hallym">
  <script>
    const {attributes} = document.getElementById('user');
    console.log(attributes.id.value);
    console.log(attributes.type.value);
    console.log(attributes.value.value);

    const $input = document.getElementById('user');
    $input.setAttribute('value', 'software');
    console.log($input.getAttribute('value'));
    $input.removeAttribute('value');
    console.log($input.hasAttribute('value'));
  </script>
</body>
```

```
user
text
hallym
software
false
```

5. attribute 조작

- 요소 노드 객체에는 HTML attribute 에 대응하는 DOM 프로퍼티 존재
  - HTML attribute 와 DOM 프로퍼티가 항상 1:1 대응하지 않으며
  - attribute와 프로퍼티 키가 반드시 일치하지 않음

- getAttribute() 로 취득한 값은 언제나 문자열
- DOM 프로퍼티로 취득한 값은 상태에 따라 다르다
  - 예:checkbox 요소의 checked 프로퍼티 값은 불리언 타입




```
<body>
  <input type="text" id="user" value="hallym">
  <br>
  <input type="checkbox" id="cbox" checked> 선택
  <script>
    const $input = document.getElementById('user');
    $input.id="guest";
    console.log($input.id);
    console.log($input.value);

    const $state = document.getElementById("cbox");
    console.log($state.checked);
    console.log($state.getAttribute('checked')); //빈문자열
  </script>
</body>
```

6. 스타일 조작

≠ 스타일 조작.

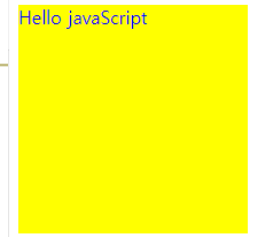
- style 프로퍼티
  - getter setter 모두 존재
  - 요소 노드의 인라인 스타일만 추가, 제거, 변경
  - style 프로퍼티에 값을 할당하면 해당 css 프로퍼티가 인라인 스타일로 html요소에 추가 또는 변경
  - 자바스크립트는 특수 문자 '-'을 식별자에 사용할 수 없으므로 오류 출력
    - - 로 연결된 단어의 첫 글자를 대문자로 변경

스타일 식별자 변환

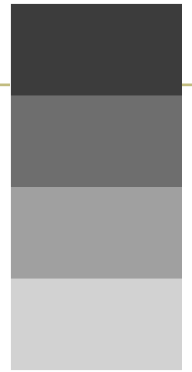
스타일시트의 스타일 속성	자바스크립트의 스타일 식별자
background-image	backgroundImage
background-color	backgroundColor
box-sizing	boxSizing
list-style	listStyle

6. 스타일 조작

```
<body>
  <div style="color:red">Hello javaScript</div>
  <script>
    const $div = document.querySelector('div');
    console.log($div.style);

    $div.style.color='blue';
    $div.style.width='200px';
    $div.style.height='200px';
    $div.style.backgroundColor='yellow';
  </script>
</body>
```




```
<body>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <script>
    const $div = document.querySelectorAll('div');
    let i=10;

    for(let box of $div){
      box.style.height = '100px';
      i+=50;
      box.style.backgroundColor = `rgb( ${i}, ${i}, ${i})`;
    }
  </script>
</body>
```



```
<div style="height: 100px; background-color: rgb(60, 60, 60);"></div>
<div style="height: 100px; background-color: rgb(110, 110, 110);"></div>
<div style="height: 100px; background-color: rgb(160, 160, 160);"></div>
<div style="height: 100px; background-color: rgb(210, 210, 210);"></div>
```

7. 클래스 조작

- class 조작

  - className 프로퍼티
    - class 속성 값을 가져오거나 설정(getter, setter 모두 존재)
    - class 속성의 값이 여러 개일 경우 공백으로 구분된 문자열 반환 – split() 메소드를 사용하여 배열로 변경

  - classList 프로퍼티
    - class 어트리뷰트 정보를 갖는 객체 반환, 메소드를 사용하여 조작
    - 메소드
      - add(... className) : 인수로 전달한 1개 이상의 문자열을 class 값으로 추가
      - remove(... className) : 인수로 전달한 문자열과 일치하는 클래스 삭제
      - item(index) : index에 해당하는 클래스 반환
      - contains(className) : 인수로 전달한 클래스 값이 포함되어 있는지 확인
      - replace(oldClassName, newClassName) : 클래스 변경
      - toggle(className) : 인수로 전달한 문자열과 일치하는 클래스가 없으면 추가, 있으면 제거

7. 클래스 조작

```
<script>
  window.onload = function () {
    const elems = document.querySelectorAll('li');

    // className
    [...elems].forEach(elem => {
        if (elem.className === 'red') { // class 어트리뷰트 값을 취득하여 확인
            elem.className = 'blue'; // class 어트리뷰트 값을 변경한다.
      }
    });

    // classList – class 어트리뷰트 정보를 갖는 객체 반환, 메소드를 사용하여 조작
    [...elems].forEach(elem => {
        if (elem.classList.contains('blue')) { // class 어트리뷰트 값 확인
            elem.classList.replace('blue', 'green'); // class 어트리뷰트 값 변경한다
      }
    });

    // h1 태그 요소 중 첫번째 요소를 취득
    const heading = document.querySelector('h1');

    // id 어트리뷰트의 값을 변경.
    // id 어트리뷰트가 존재하지 않으면 id 어트리뷰트를 생성하고 지정된 값을 설정
    heading.id = 'heading';
    console.log(heading.id);
  }
</script>
```

```
<body>
  <div>
    <h1>Cities</h1>
    <ul>
      <li id="one" class="red">hallym</li>
      <li id="two" class="red">software</li>
      <li id="three" class="red">big data</li>
      <li id="four">IoT</li>
    </ul>
  </div>
</body>
```

```
▼<div>
    <h1 id="heading">Cities</h1>
  ▼<ul>
    ▶<li id="one" class="green">…</li>
    ▶<li id="two" class="green">…</li>
    ▶<li id="three" class="green">…</li>
    ▶<li id="four">…</li>
    </ul>
  </div>
```

8. DOM 조작

* DOM조작.

- innerHTML 프로퍼티
  - setter, getter 모두 존재
  - 요소 노드의 마크업을 문자열로 반환하거나 변경

```
<head>
  <script>
    window.onload = function () {
      const ul = document.querySelector('ul');
      console.log(ul.innerHTML); // 요소 노드의 마크업 취득

      const one = document.getElementById('one');
      console.log(one.innerHTML); // 요소의 텍스트 취득

      // 요소의 텍스트 변경
      one.innerHTML += ', University';
      console.log(one.innerHTML);

      const two = document.getElementById('two');
      // 요소의 마크업이 포함된 콘텐츠 변경
      two.innerHTML = '<h1>Heading</h1>';
      console.log(two.innerHTML); // 마크업이 문자열로 표시
    }
  </script>
</head>
```

Software에서 <h1>의 Heading으로

```
<body>
  <div>
    <h1>Cities</h1>
    <ul>
      <li id="one" class="red">hallym</li>
      <li id="two" class="red">software</li>
      <li id="three" class="red">big data</li>
      <li id="four">IoT</li>
    </ul>
  </div>
</body>
```

Cities

- hallym, University

Heading

- big data
- IoT

```
      <li id="one" class="red">hallym</li>
      <li id="two" class="red">software</li>
      <li id="three" class="red">big data</li>
      <li id="four">IoT</li>
```

```
hallym
hallym, University
<h1>Heading</h1>
```

8. DOM 조작

- innerHTML 프로퍼티를 사용하지 않고, 새로운 콘텐츠를 추가할 수 있는 방법은 DOM을 직접 조작
- 순서
  1. 요소 노드 생성
     - createElement() 메소드를 사용하여 새로운 요소 노드 생성
     - createElement() 메소드의 인자로 태그 이름을 전달
  2. 텍스트 노드 생성
     - createTextNode() 메소드를 사용하여 새로운 텍스트 노드 생성
  3. 생성된 요소를 DOM에 추가
     - appendChild() 메소드를 사용하여 생성된 노드를 DOM tree에 추가
     - 또는 removeChild() 메소드를 사용하여 DOM tree에서 노드 삭제

메소드	**기능**
createElement(tagName)	태그 이름을 인자로 전달하여 요소를 생성
createTextNode(text)	텍스트를 인자로 전달하여 텍스트 노드를 생성
appendChild(node)	인자로 전달한 노드를 마지막 자식 요소로 DOM 트리에 추가
removeChild(node)	인자로 전달한 노드를 DOM 트리에 제거
insertBefore(newNode, childNode)	newNode를 childNode 앞에 삽입 childNode는 insertBefore() 메소드를 호출한 노드의 자식이어야 함

8. DOM 조작

```
<body>
  <div>
    <h1>Cities</h1>
    <ul>
      <li id="one" class="red">hallym</li>
      <li id="two" class="red">software</li>
      <li id="three" class="red">big data</li>
      <li id="four">IoT</li>
    </ul>
  </div>
  <script>
      const newElem = document.createElement('li');  // 태그 이름을 인자로 전달하여 요소를 생성
      const newText = document.createTextNode('Beijing'); // 텍스트 노드를 생성
      newElem.appendChild(newText); // 텍스트 노드를 newElem 자식으로 DOM 트리에 추가

      const container = document.querySelector('ul');
      // newElem을 container의 자식으로 DOM 트리에 추가. 마지막 요소로 추가된다.
      container.appendChild(newElem);

      const removeElem = document.getElementById('one');
      container.removeChild(removeElem);  // container의 자식인 removeElem 요소를 DOM 트리에서 제거
  </script>
</body>
```

8. DOM 조작

- 복수의 노드 생성과 추가


```
<body>
  <ul id="major"></ul>
  <script>
    const $major = document.getElementById('major');

    //DocumentFragment 노드(노드 객체의 일종으로 부모 노드가 없음) 생성
    const $fragment = document.createDocumentFragment();

    ['hallym', 'software', 'information'].forEach(text => {
      //1. 요소노드 생성
      const $li = document.createElement('li');

      //2. 텍스트 노드 생성
      const textNode = document.createTextNode(text);

      //3. 텍스트 노드를 $li 요소노드의 자식노드로 추가
      $li.appendChild(textNode);

      //4. $li 요소 노드를 DocumentFragment 노드의 마지막 자식 노드로 추가
      $fragment.appendChild($li);
    });

    //5. DocumentFragment 노드를 $major 요소 노드의 마지막 자식 노드로 추가
    $major.appendChild($fragment);
  </script>
</body>
```

9. 이벤트

- 브라우저는 클릭이나 키 입력, 마우스 이동 등이 발생하면 이를 감지하여 특정한 타입의 이벤트를 발생
- 이벤트가 발생하면 브라우저에 위임 된 함수를 호출하여 처리
  - 이벤트 핸들러(이벤트 리스너)
    - 이벤트가 발생했을 때 호출될 함수
  - 이벤트 핸들러 등록
    - 브라우저에게 이벤트 핸들러의 호출을 위임
  - 이벤트 타입
    - 이벤트의 종류를 나타내는 문자열

인라인 이벤트 핸들러
이벤트 핸들러 프로퍼티
addEventListener() 메소드 방식

```
<body>
  <button>onClick</button>
  <script>
    const $btn = document.querySelector('button');

    //버튼 요소에 클릭이벤트가 발생하면 이벤트 핸들러를 호출하도록 브라우저에 위임
    $btn.onclick = () => { alert('Button Click');};
  </script>
</body>
```

10. 이벤트 핸들러 등록 - 3가지 방식 숙지.

- **인라인 이벤트 핸들러 방식(이벤트 핸들러 어트리뷰트방식)**
  - HTML 요소의 이벤트 핸들러 속성에 이벤트 핸들러를 등록하는 방법
  - 더 이상 사용되지 않지만 간혹 사용한 경우가 있어 알아둘 필요 있음
  - 이벤트 핸들러 어트리뷰트 이름
    - on+이벤트 타입으로 구성
    - 함수 호출문 할당

```
<head>
  <meta charset="UTF-8">
  <script>
    function divClick(elem) {
      elem.style.backgroundColor = 'green';
      alert(this); //window
    }
  </script>
</head>
<body>
  <h1>사각형 박스를 클릭하세요</h1>
  //인라인 이벤트 핸들러 방식
  <div style="width: 100px;height: 100px;background-color: coral;" onclick="divClick(this)"></div>
</body>
```

인라인 이벤트 핸들러 내부의 this ->window 객체
단, 호출할 때 this -> 이벤트를 바인딩한 DOM 요소

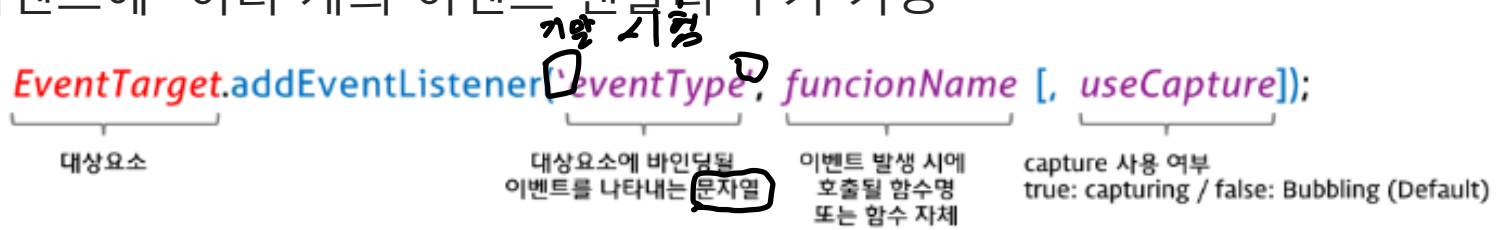
10. 이벤트 핸들러 등록

- **이벤트 핸들러 프로퍼티 방식**
  - 요소 노드의 이벤트 속성으로 이벤트를 연결
  - 자바스크립트와 HTML이 분리
  - 한번에 하나의 이벤트 리스너만 가질 수 있음
  - 이벤트 타깃
    - 이벤트를 발생 시킬 객체
  - 이벤트 속성
    - on + 이벤트 타입
    - 함수 호출이 아닌 참조를 할당

```
<div id="box" style="width: 100px;height: 100px;background-color: coral;"></div>
<script>
    const $box = document.querySelector('#box');

    //이벤트 핸들러 프로퍼티 방식
    //$box:이벤트 타깃, onclick:이벤트 속성,  function(){ } : 이벤트핸들러
    $box.onclick = function() {
      $box.style.backgroundColor = 'green';
    };
</script>
```

10. 이벤트 핸들러 등록

- **addEventListener 메소드 방식**
  - addEventListener 메소드를 이용하여 대상 DOM 요소에 이벤트를 바인딩하고 해당 이벤트가 발생했을 때 실행될 콜백 함수(이벤트 핸들러)를 지정
  - 하나의 이벤트에 여러 개의 이벤트 핸들러 추가 가능




```
<div id="box" style="width: 100px;height: 100px;background-color: coral;"></div>
<script>
    const $box = document.querySelector('#box');
    const mleave = ()=>{
      $box.style.backgroundColor = 'coral';
    };

    //addEventListener 메소드 방식
    $box.addEventListener('mouseover', function(){
      $box.style.backgroundColor = 'green';
    });

    $box.addEventListener('mouseleave', mleave);
</script>
```

11. 이벤트 핸들러 제거

이벤트 핸들러 제거.

- removeEventListener() 메소드
  - addEventListener() 메소드로 등록한 이벤트 핸들러 제거
  - addEventListener() 메소드 인수와 일치해야 함
- 이벤트 핸들러 프로퍼티 방식
  - 이벤트 핸들러 프로퍼티에 null 할당

```
<div id="box" style="width: 100px;height: 100px;background-color: coral;"></div>
<script>
    const $box = document.querySelector('#box');
    const boxclick = () =>{  $box.style.backgroundColor = 'green';  };

    //addEventListener 메소드 방식
    $box.addEventListener('click', boxclick);
    $box.removeEventListener('click', boxclick); //이벤트 핸들러 제거

   //이벤트 핸들러 프로퍼티 방식
   $box.onclick = boxclick;
   $box.onclick = null; //이벤트 핸들러 제거
</script>
```

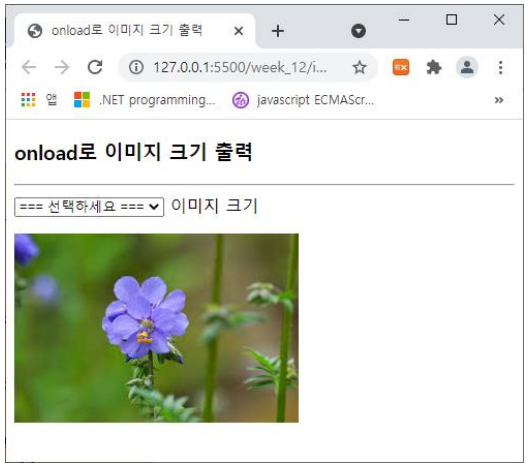
결과는?

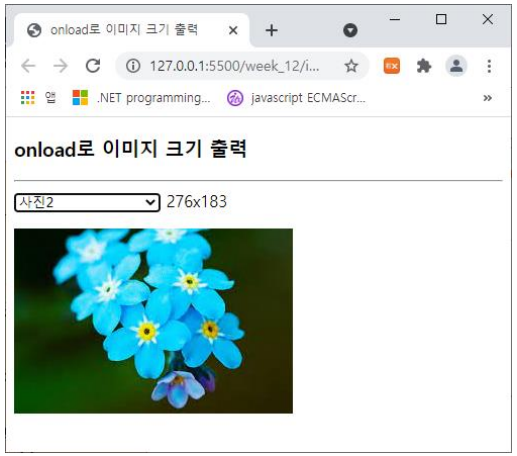
```
<script>
    const $box = document.querySelector('#box');
    const boxclick = ()=>{
      $box.style.backgroundColor = 'green';
      $box.removeEventListener('click', boxclick);
    };

    $box.addEventListener('click', boxclick);
</script>
```

12. 이벤트 활용

- load & change 이벤트 – 이미지 크기 출력

```
<head>
  <title>onload로 이미지 크기 출력</title>
  <script>
    function changeImage() {
      const sel = document.getElementById("sel");
      const img = document.getElementById("myImg");
      img.onload = function () {  // 이미지 로딩 완료 후 이미지 크기 출력
        const mySpan = document.getElementById( " mySpan " );
        mySpan.innerHTML = img.width +  " x "  + img.height;
      }
      const index = sel.selectedIndex; // 선택된 옵션 인덱스
      img.src = sel.options[index].value; // <option>의 value 속성
    }
  </script>
</head>
```

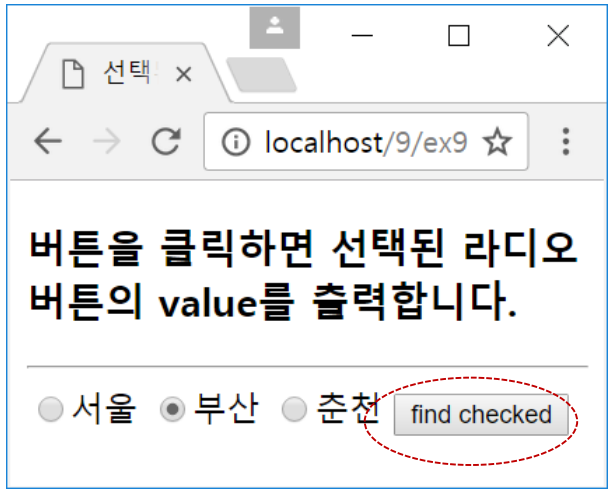


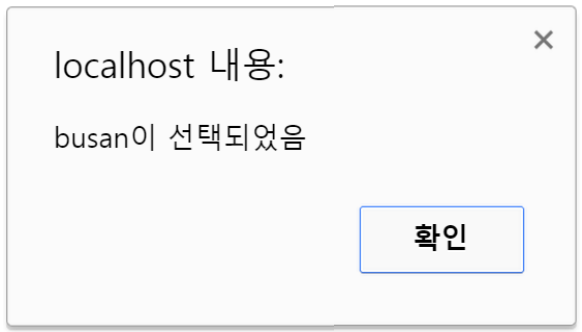



```
<body>
  <form>
    <select id="sel" onchange="changeImage()">
      <option value=" "> === 선택하세요 ===</option>
      <option value="image/flow2.jpg">사진1</option>
      <option value="image/flow3.jpg">사진2</option>
      <option value="image/flow4.jpg">사진3</option>
      <option value="image/flow4.jpg">사진4</option>
    </select>
    <span id="mySpan">이미지 크기</span>
  </form>
  <p><img id="myImg" src="image/flow7.jpg"></p>
</body>
```

12. 이벤트 활용

- click 이벤트 - 선택된 라디오 버튼 알아내기





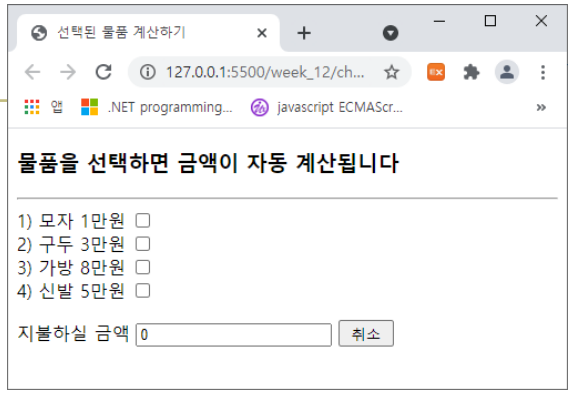

```
<head>
  <title>선택된 라디오 버튼 알아내기</title>
  <script>
   function findChecked() {
     let found = null;
     let kcity = document.getElementsByName("city");
     for (let i = 0; i < kcity.length; i++) {
       if (kcity[i].checked == true)    found = kcity[i];
     }
     if (found != null)   alert(found.value + "이 선택되었음");
     else  alert("선택된 것이 없음");
   }
   window.onload = function(){
     let btn = document.querySelector('input[type=button]');
     btn.addEventListener('click', findChecked);
   }
  </script>
</head>
<body>
   <form>
    <input type="radio" name="city" value="seoul" checked>서울
    <input type="radio" name="city" value="busan">부산
    <input type="radio" name="city" value="chunchen">춘천
    <input type="button" value="find checked" >
  </form>
</body>
```

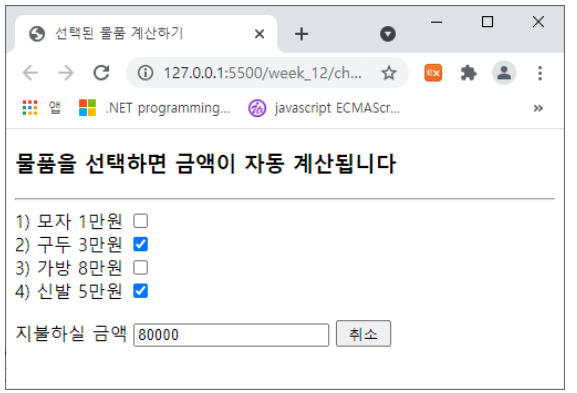
12. 이벤트 활용

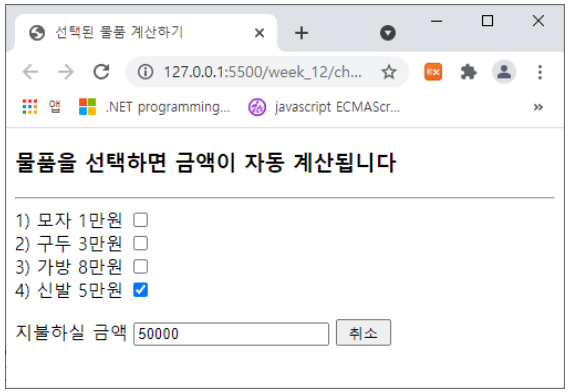
- 체크 박스 click 이벤트 - 체크박스로 선택한 물품 계산

이벤트 핸들러 프로퍼티 방식 & addEnentListener 메소드 방식의 이벤트핸들러 내부의 this -> 이벤트를 바인딩한 DOM 요소

```
<head>
  <script>
    let calc = (function(){
      let sum = 0;
      return function() {
        if (this.checked)   sum += parseInt(this.value);
        else sum -= parseInt(this.value);
        document.getElementById("sumtext").value = sum;
      }
    })();
   window.onload = function () {
      const query = 'input[type=checkbox]';
      const chbox = document.querySelectorAll(query);
      chbox.forEach((el) => { el.addEventListener('click', calc);  });
    }
  </script>
</head>
<body>
   <form>  1) 모자 1만원 <input type="checkbox"  value="10000"><br>
           2) 구두 3만원 <input type="checkbox" value="30000"><br>
           3) 가방 8만원 <input type="checkbox"  value="80000"><br>
           4) 신발 5만원 <input type="checkbox"  value="50000"><br>
          지불하실 금액 <input type="text" id="sumtext" value="0"> <input type="reset" value=" 취소 ">
  </form>
</body>
```

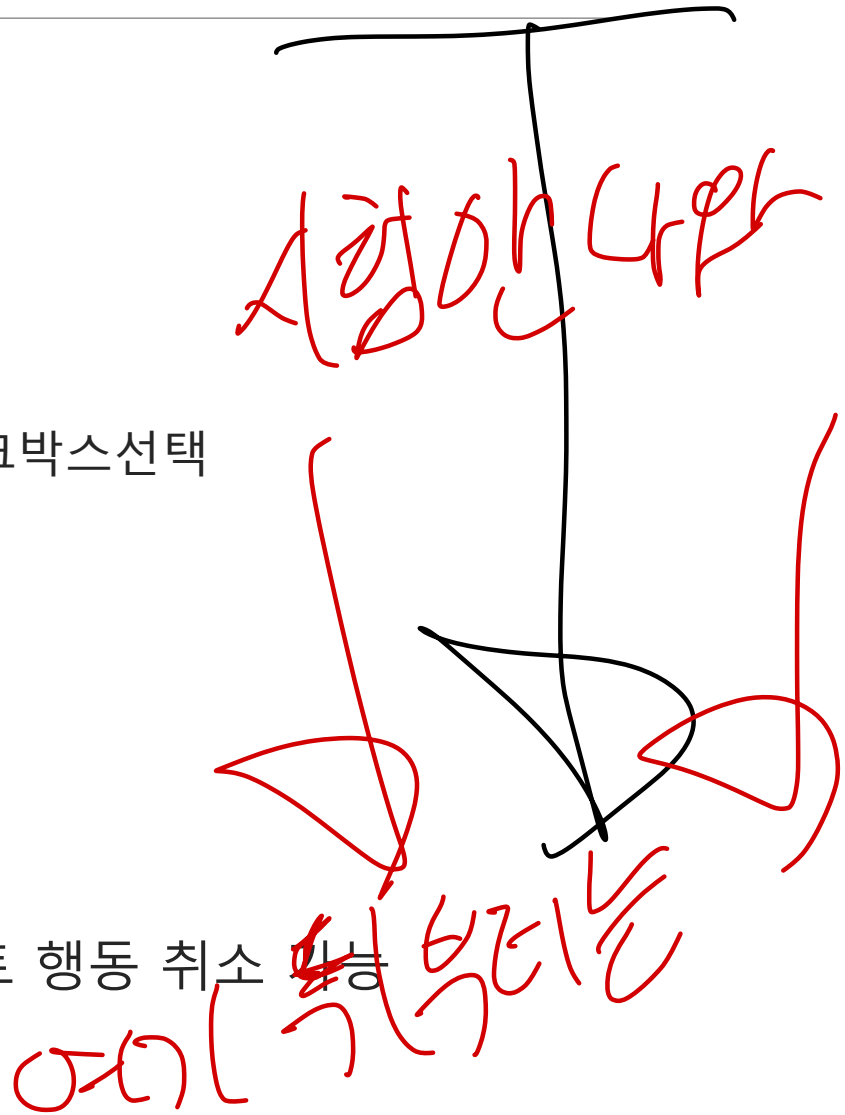






13. 이벤트의 디폴트 행동 취소

- 이벤트의 디폴트 행동이란?
  - 특정 이벤트에 대한 HTML 태그의 기본 행동
  - 사례
    - <a>의 click 이벤트의 디폴트 행동 : 웹 페이지 이동
    - submit 버튼의 click 이벤트의 디폴트 행동 : 폼 데이터 전송
    - <input type="checkbox">의 click 이벤트의 디폴트 행동 : 체크박스선택

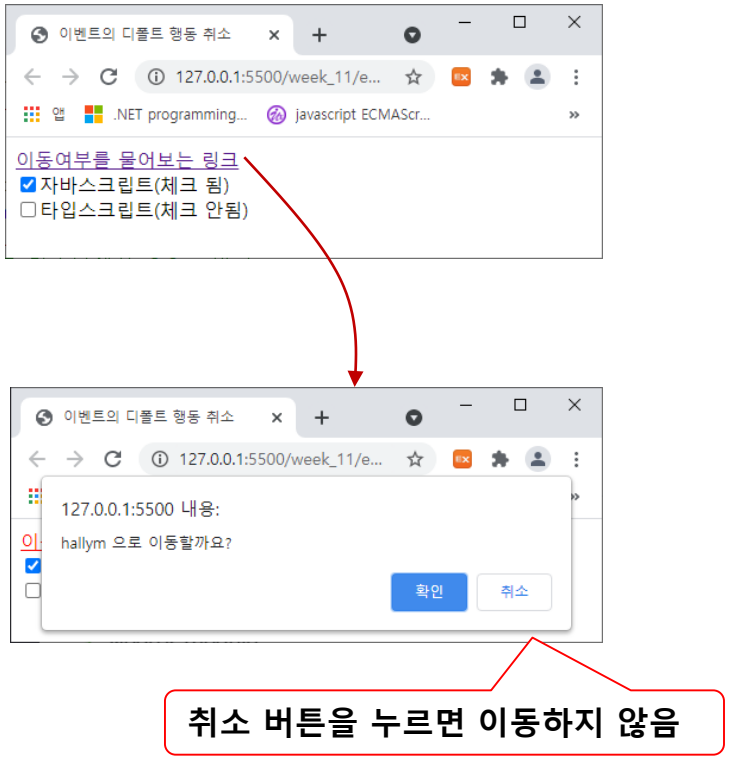


- 이벤트의 디폴트 행동을 막는 방법
  - 이벤트 리스너에서 false 리턴
  - 이벤트 객체의 preventDefault() 메소드 호출

- 이벤트 객체의 cancelable 프로퍼티가 true인 경우만 디폴트 행동 취소 가능

13. 이벤트의 디폴트 행동 취소

```
<head>
  <meta charset="UTF-8">
  <title>addEventListener 메소드 </title>
  <script>
    window.onload = function () {
      const elem = document.getElementById('event');
      elem.addEventListener('click', function (e) {
        console.log(e.cancelable);
        e.preventDefault();  //이벤트 객체의preventDefault() 호출
      });

      const button = document.getElementById('button');
      button.onclick = function () {
        let answer=confirm('hallym 으로 이동할까요?')
        return answer;  //이벤트 리스너에서 false 반환
      };
    }
  </script>
</head>
<body>
  <a id="button" href="http://www.hallym.ac.kr">이동 여부를 물어보는 링크</a>
  <form>
    <input type="checkbox">자바스크립트(체크 됨)<br>
    <input id="event" type="checkbox">타입스크립트(체크 안됨)
  </form>
</body>
```



14. 이벤트 흐름

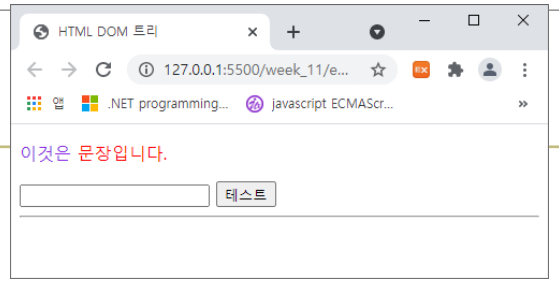
- 이벤트 흐름이란?
  - 이벤트가 발생하면 window 객체에 먼저 도달하고, DOM 트리를 따라 이벤트 타겟에 도착하고, 다시 반대 방향으로 흘러 window 객체에 도달한 다음 사라지는 과정

- 이벤트가 흘러가는 과정
  - 캡쳐 단계(capturing phase)
    - 이벤트가 window 객체에서 중간의 모든 DOM 객체를 거쳐 타겟 객체에 전달되는 과정
    - 이벤트가 거쳐가는 모든 DOM 객체(window포함)의 이벤트 핸들러 실행
  - 버블 단계(bubbling phase)
    - 이벤트가 타겟에서 중간의 모든 DOM 객체를 거쳐 window 객체에 전달되는 과정
    - 이벤트가 거쳐가는 모든 DOM 객체(window포함)의 이벤트 핸들러 실행

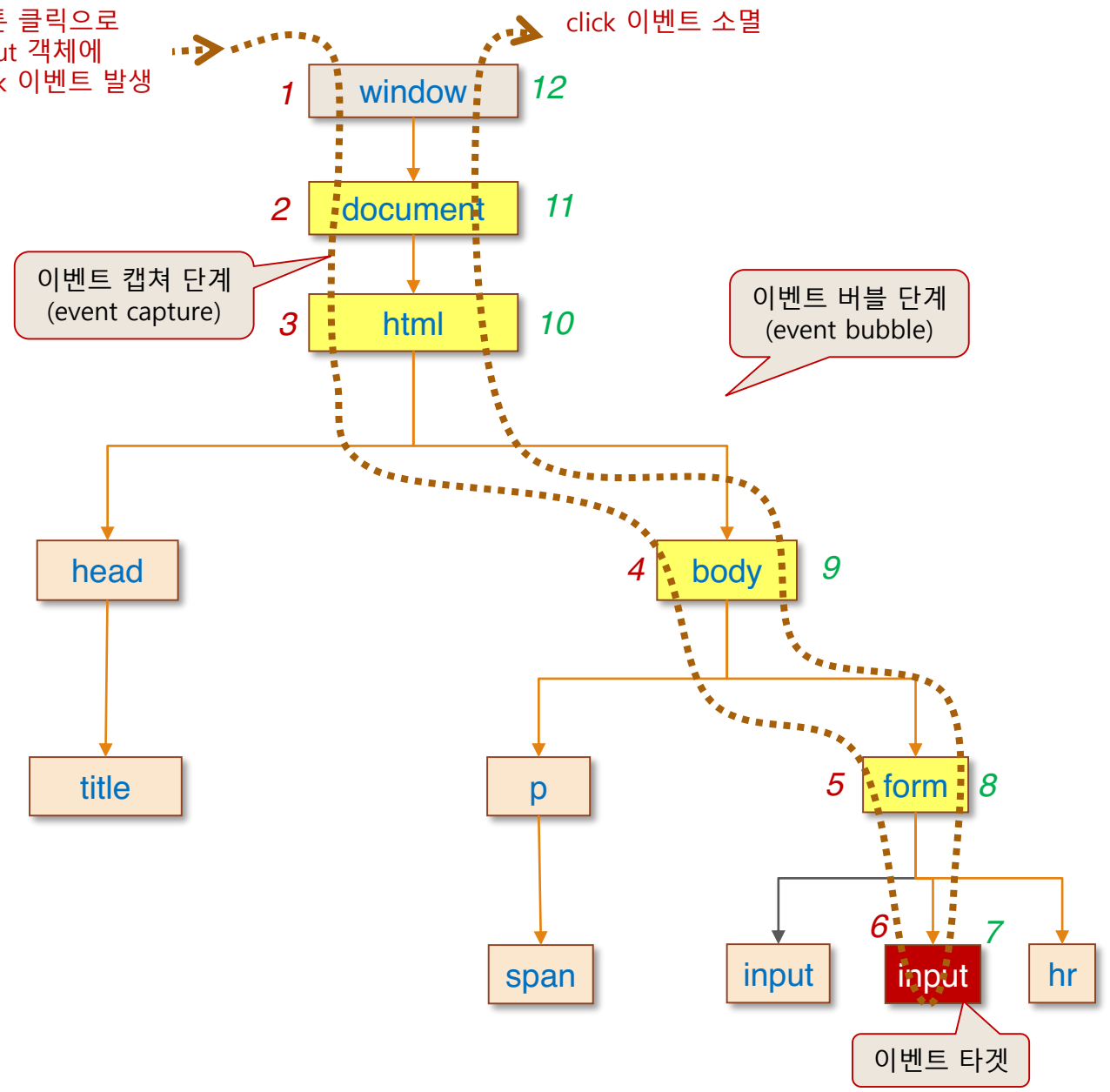
- DOM 객체에는 캡쳐 리스너와 버블 리스너 두 개 모두 작성할 수 있음
  - 이벤트 핸들러 등록 시, 캡쳐 리스너인지 버블 리스너인지 구분

14. 이벤트 흐름

- 샘플 웹 페이지




```
<!DOCTYPE html>
<html>
<head>
  <title>HTML DOM 트리</title>
  <style>
    p {     color: blueviolet;    }
    span {      color: darkgreen;    }
    #div{       color:darkorange;    }
  </style>
</head>
<body>
  <p>이것은   <span>문장입니다.</span>  </p>
  <form>
    <input type="text">
    <input type="button" value="테스트" id="button">
    <hr>
  </form>
  <div id="div"></div>
</body>
</html>
```



14. 이벤트 흐름

- 캡쳐 리스너와 버블 리스너 등록
  - addEventListener()의 3 번째 매개 변수 이용
    - true이면 캡쳐 리스너, false또는 생략 이면 버블 리스너

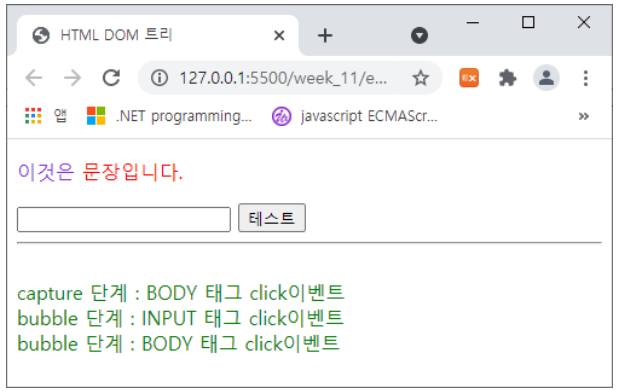
```
let b = document.getElementById("button");
b.addEventListener("click", capFunc, true); // 캡쳐 단계에서 capFunc() 실행
b.addEventListener("click", bubbleFunc, false); // 버블 단계에서 bubbleFunc() 실행
```

  - 다른 방법의 이벤트 리스너 등록의 경우
    - 버블 리스너로 자동 등록
    - 예)

```
obj.onclick = function(e) { // 버블 리스너로 작동
        ...
}
```

14. 이벤트 흐름

```
<script>
    let div = document.getElementById("div"); // 이벤트 메시지 출력 공간
    let button = document.getElementById("button");

    // body 객체에 캡쳐 리스너 등록
    document.body.addEventListener("click", capture, true); // 캡쳐 단계(1)

    // 타겟 객체에 버블 리스너 등록
    button.addEventListener("click", bubble, false); // 버블 단계(2)

    // body 객체에 버블 리스너 등록
    document.body.addEventListener("click", bubble); // 버블 단계(3)

    function capture(e) { // e는 이벤트 객체
      let obj = e.currentTarget; // 현재 이벤트를 받은  DOM 객체
      let tagName = obj.tagName; // 태그 이름
      div.innerHTML += "<br>capture 단계 : " + tagName + " 태그 " + e.type + "이벤트";
    }

    function bubble(e) { // e는 이벤트 객체
      let obj = e.currentTarget; // 현재 이벤트를 받은  DOM 객체
      let tagName = obj.tagName; // 태그 이름
      div.innerHTML += "<br>bubble 단계 : " + tagName + " 태그 " + e.type + "이벤트";
    }
  </script>
```



14. 이벤트 흐름 - 중단

- 이벤트 흐름 중단
  - stopPropagation() 호출

```
<body>
  <p>버튼을 클릭하면 이벤트 전파를 중단한다. <button>버튼</button></p>
  <script>
    const body = document.querySelector('body');
    const para = document.querySelector('p');
    const button = document.querySelector('button');

    body.addEventListener('click', function () { // 버블링
      console.log('Handler for body.');
    });

    para.addEventListener('click', function () { // 버블링
      console.log('Handler for paragraph.');
    });

    button.addEventListener('click', function (e) { // 버블링
      console.log('Handler for button.');
          e.stopPropagation();  // 이벤트 전파를 중단한다.
    });
  </script>
</body>
```

이벤트 흐름 중단

```
Handler for button.
```

이벤트 흐름 중단하지 않음

```
Handler for button.
Handler for paragraph.
Handler for body.
```

주석으로 처리하면 이벤트 흐름이 중단되지 않음

15. 웹 브라우저 객체 모델(BOM)

- 웹 브라우저 객체 모델(BOM : Browser Object Model)
  - 웹 브라우저와 관련된 객체의 집합

- 종류

(1) document 객체

- 웹 브라우저에서 보여주는 문서에 관련된 정보를 접근할 때 사용하는 객체.
- 자바스크립트를 이용하여 웹 브라우저에 출력할 때에도 사용
- document에는 문자열, 이미지, 폼, 링크 등 모든 페이지 상위 객체가 포함

(2) window 객체

- 웹 브라우저 기반 자바스크립트의 객체 계층 구조에서 최상위에 존재하며, 웹 브라우저 윈도우를 나타내는 객체
- 창 열기, 창 닫기, 창 크기 조절 등의 창을 제어하는 다양한 작업을 할 수 있다.

(3) screen 객체

- screen 객체는 사용자의 디스플레이 화면에 대한 다양한 정보를 저장하는 객체

15. 웹 브라우저 객체 모델(BOM)

(4) history 객체

- 웹 브라우저의 URL에 대한 히스토리 정보를 문서와 문서 상태 목록으로 나타내는 객체

(5) location 객체

- 현재 문서의 URL(웹 브라우저 주소 표시줄)과 관련된 정보를 가지고 있는 객체
- window 객체의 하위 객체지만 window 객체를 생략하고 사용할 수 있다.
- 프로토콜의 종류, 호스트 이름, 문서 위치 등의 정보를 가지고 있는 객체

(4) navigator 객체

- 웹 브라우저 공급자 및 버전 정보 등을 포함한 웹 브라우저에 대한 다양한 정보를 저장하는 객체

16. window 객체

- window 객체
  - 열려 있는 브라우저 윈도우나 탭 윈도우의 속성을 나타내는 객체
  - 브라우저 윈도우나 탭 윈도우마다 별도의 window 객체 생성
- window 객체의 생성
  - 3 가지 경우
    - 브라우저가 새로운 웹 페이지를 로드할 때
    - <iframe> 태그 당 하나의 window 객체 생성
    - 자바스크립트 코드로 윈도우 열기 시 window 객체 생성
      - window.open("웹페이지 URL", "윈도우이름", "윈도우속성"),

16. window 객체

- 윈도우 열기
  - window.open()
    - 윈도우를 새로 열고 웹 페이지 출력
    - 3개의 매개변수를 가진 함수

```
window.open(sURL, sWindowName, sFeature)
```

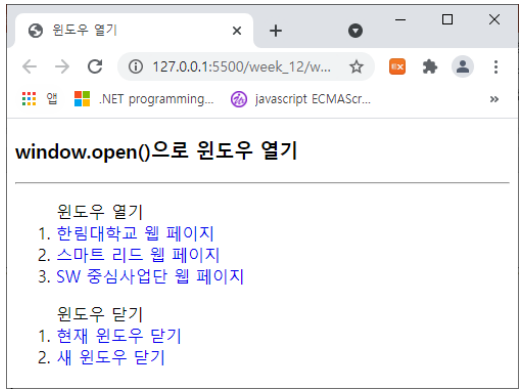
- sURL : 윈도우에 출력할 웹 페이지 주소 문자열
- sWindowName : 새로 여는 윈도우의 이름 문자열로서 생략 가능
- sFeature : 윈도우의 모양, 크기 등의 속성들을 표현하는 문자열. 속성들은 빈칸 없이 콤마(‘,’)로 분리하여 작성하며 생략 가능

  - 윈도우 이름(sWindowName)

_blank : 이름 없는 새 윈도우를 열고, 웹 페이지 로드
_parent : 현재 윈도우(혹은 프레임)의 부모 윈도우에 웹 페이지 로드
_self : 현재 윈도우에 웹 페이지 로드
_top : 브라우저 윈도우에 웹 페이지 로드

16. window 객체

```
<head>
  <script> //윈도우 열기 & 닫기
   let newWin = null;  //새로 오픈한 윈도우 저장
   function load(URL) {
     newWin = window.open(URL, 'myWin', 'left=300,top=300,width=400,height=300');
   }
   function closeNewWindow() {
     if (newWin == null || newWin.closed) // 윈도우가 열리지 않았거나 닫힌 경우
       return; // 윈도우가 없는 경우 그냥 리턴
     else
       newWin.close(); // 열어 놓은 윈도우 닫기
   }
  </script>
</head>
<body>
   <ol> 윈도우 열기
   <li><a href="javascript:load('http://www.hallym.ac.kr')"> 한림대학교 웹 페이지</a></li>
   <li><a href="javascript:load('https://smartlead.hallym.ac.kr')">스마트 리드 웹 페이지</a></li>
   <li><a href="javascript:load('https://hlsw.hallym.ac.kr')"> SW 중심사업단 웹 페이지</a></li>
  </ol>
  <ol> 윈도우 닫기
   <li><a href="javascript:window.close()"> 현재 윈도우 닫기</a></li>
   <li><a href="javascript:closeNewWindow()"> 새 윈도우 닫기</a></li>
  </ol>
</body>
```



16. window 객체

- 타이머 활용
  - window 객체의 타이머 기능 2 가지
    - 타임아웃 코드 1회 호출
      - setTimeout()/clearTimeout() 메소드

```
var timerID = setTimeout("timeOutCode", msec)
clearTimeout(timerID)
```

- timeOutCode : 타임아웃 자바스크립트 코드
- msec : 밀리초 단위의 정수로서, 타임아웃 지연 시간

setTimeout()은 msec 후에 timeOutCode를 1회 실행하도록 타이머를 설정하고, 타이머 ID를 리턴한다.
clearTimeout()은 작동 중인 timerID의 타이머를 해제한다.

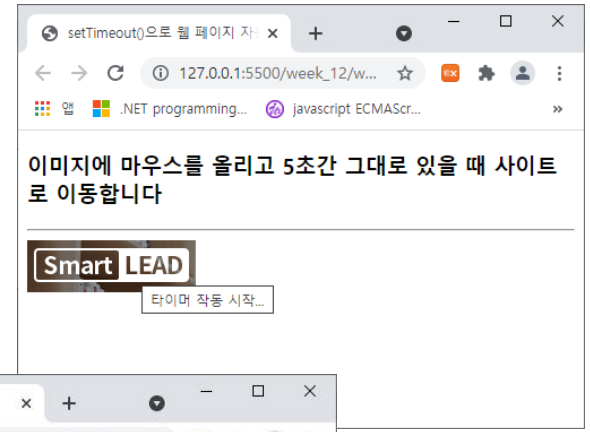
    - 타임아웃 코드 반복 호출
      - setInterval()/clearInterval() 메소드
      - setTimeout()과 동일한 문법 사용

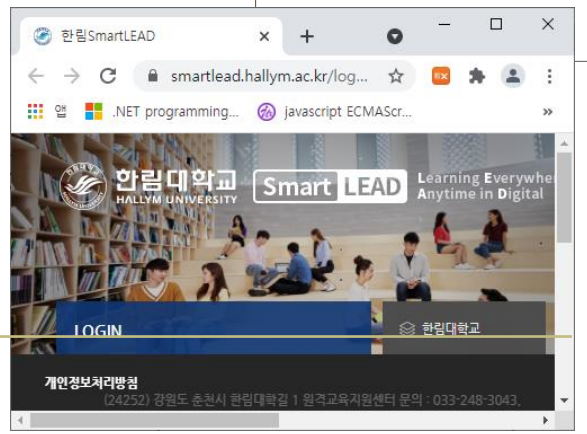
16. window 객체

- 웹 페이지 자동 연결

이미지 위에 마우스를 올린 상태로 5초가 지나면 스마트리드로 연결하며, 5초 전에 이미지를 벗어나면 타이머를 해제한다.

```
<body>
  <h3>이미지에 마우스를 올리고 5초간 그대로 있을 때 사이트로 이동합니다</h3>
  <hr>
  <img id="img" src="image/smart.PNG" onmouseover="startTimer(5000)" onmouseout="cancelTimer()">
  <script>
    var timerID = null;
    function startTimer(time) {
      // 타이머 시작
      timerID = setTimeout("load('https://smartlead.hallym.ac.kr')", time);

      // 이미지에 마우스 올리면 나타내는 툴팁 메시지
      document.getElementById("img").title = "타이머 작동 시작...";
    }
    function cancelTimer() {
      if (timerID != null)    clearTimeout(timerID); // 타이머 중단
    }
  </script>
</body>
```

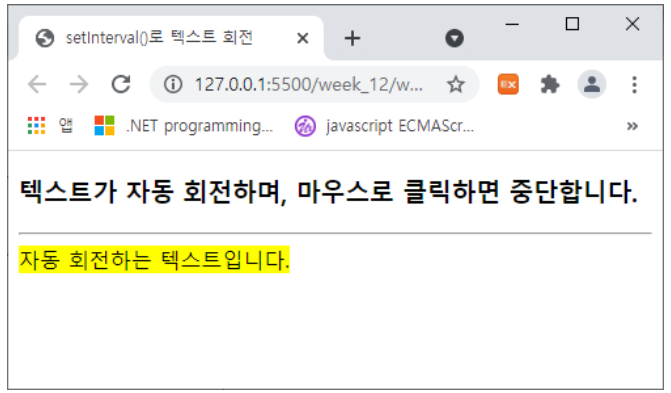




16. window 객체

setInterval()을 이용하여 텍스트를 옆으로 반복 회전. 텍스트 위에 마우스를 클릭하면 회전 중단.

```
<body>
  <h3>텍스트가 자동 회전하며, 마우스로 클릭하면 중단합니다.</h3>
  <hr>
  <div><span id="span" style="background-color:yellow">
      자동 회전하는 텍스트입니다.</span>
  </div>
  <script>
    let span = document.getElementById("span");
    let timerID = setInterval("doRotate()", 200); // 200밀리초 주기로 doRotate() 호출
    span.onclick = function (e) { // 마우스 클릭 이벤트 리스너
      clearInterval(timerID); // 타이머 해제. 문자열 회전 중단
    }
    function doRotate() {
      let str = span.innerHTML;
      let firstChar = str.substr(0, 1);
      let remains = str.substr(1, str.length - 1);
      str = remains + firstChar;
      span.innerHTML = str;
    }
  </script>
</body>
```

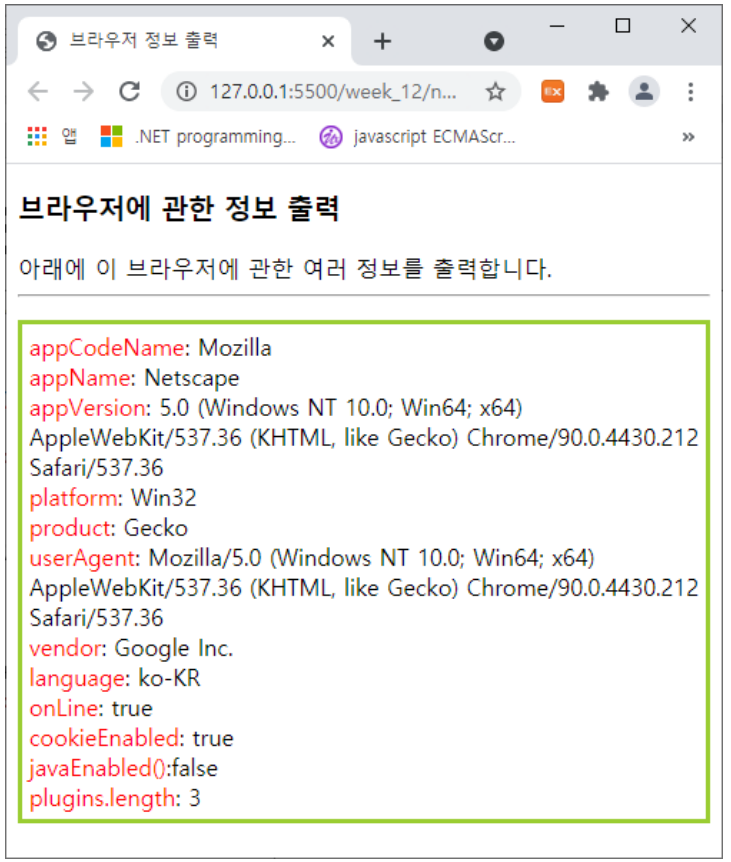


16. location 객체

```
<head>
  <script>
    function LoInfo() {
      let message = `현재 웹 문서 주소 : ${location.href} ₩n`;
      message += `현재 웹 문서 경로 : ${location.pathname} ₩n`;
      alert(message);
    }
    function LoAssign() { location.assign('https://library.hallym.ac.kr'); }
    function LoHref() { location.href = 'https://smartlead.hallym.ac.kr'; }
  </script>
</head>
<body>
  <h3>location 객체의 속성 값</h3>
  <hr>
  <form>
    <input type="button" value="객체 속성 확인" onclick="LoInfo()">
    <p></p>
    <input type="button" value="학교도서관 연결하기" onclick="LoAssign()">
    <p></p>
    <input type="button" value="스마트리드 연결하기" onclick="LoHref()">
    <p></p>
  </form>
</body>
```

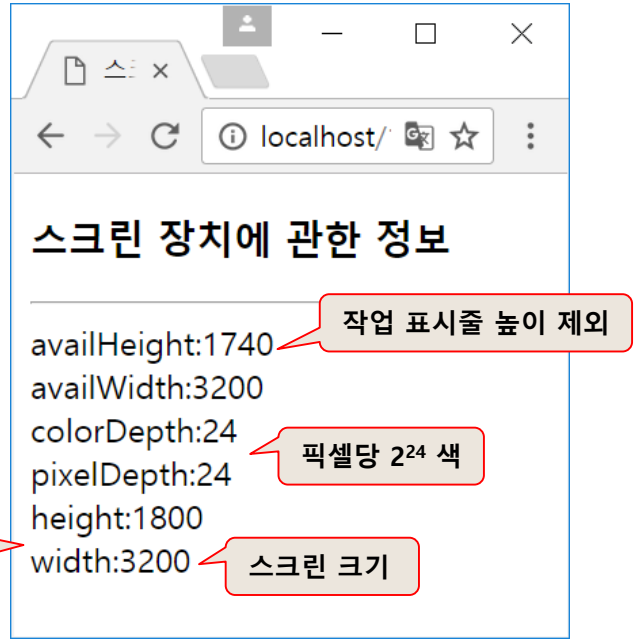
17. navigator 객체

```
<script> //브라우저 정보 출력
    function printNavigator() {
      var text = "<span>appCodeName</span>: " + navigator.appCodeName + "<br>";
      text += "<span>appName</span>: " + navigator.appName + "<br>";
      text += "<span>appVersion</span>: " + navigator.appVersion + "<br>";
      text += "<span>platform</span>: " + navigator.platform + "<br>";
      text += "<span>product</span>: " + navigator.product + "<br>";
      text += "<span>userAgent</span>: " + navigator.userAgent + "<br>";
      text += "<span>vendor</span>: " + navigator.vendor + "<br>";
      text += "<span>language</span>: " + navigator.language + "<br>";
      text += "<span>onLine</span>: " + navigator.onLine + "<br>";
      text += "<span>cookieEnabled</span>: " + navigator.cookieEnabled + "<br>";
      text += "<span>javaEnabled()</span>:" + navigator.javaEnabled() + "<br>";
      text += "<span>plugins.length</span>: " + navigator.plugins.length + "<br>";
      const div = document.getElementById("div");
      div.innerHTML = text;
    }
    window.onload = printNavigator;
  </script>
</head>
<body>
  <h3>브라우저에 관한 정보 출력</h3>
  아래에 이 브라우저에 관한 여러 정보를 출력합니다.
  <hr>
  <p>
  <div id="div"></div>
</body>
```



18. screen 객체

```
<head>
  <title>스크린 장치에 관한 정보 출력</title>
  <script>
    window.onload = function(){
      let text = "availHeight:".fontcolor('blue') + screen.availHeight + "<br>";
      text += "availWidth:".fontcolor('blue') + screen.availWidth + "<br>";
      text += "colorDepth:".fontcolor('blue') + screen.colorDepth + "<br>";
      text += "pixelDepth:".fontcolor('blue') + screen.pixelDepth + "<br>";
      text += "height:".fontcolor('blue') + screen.height + "<br>";
      text += "width:".fontcolor('blue') + screen.width + "<br>";
      document.getElementById("div").innerHTML = text;
    }
  </script>
</head>
<body>
  <h3>스크린 장치에 관한 정보</h3>
  <hr>
  <div id="div"></div>
</body>
```



height와 width는 브라우저의 설정에서 확대/축소 값을 100%로 해야 정확한 값으로 출력됨

19. history 객체

- history 객체
  - 윈도우에서 방문한 웹 페이지 리스트(히스토리)를 나타내는 객체

프로퍼티	설명	r/w
length	히스토리 리스트에 있는 엔트리 수	r

메소드	설명
back()	히스토리에 있는 이전 웹 페이지로 이동. 브라우저의 <back> 버튼과 동일
forward()	히스토리에 있는 다음 웹 페이지로 이동. 브라우저의 <forward> 버튼과 동일
go(n)	히스토리에서 현재 웹 페이지에서 n 만큼 상대적인 웹 페이지로 이동

  - history 객체를 이용하여 웹 페이지를 이동하는 코드 사례

```
history.back();      // 이전 페이지로 이동
history.go(-1);      // 이전 페이지로 이동
history.forward();   // 다음 페이지로 이동
history.go(1);       // 다음 페이지로 이동
```

Q & A

- 자바스크립트에 대한 학습이 모두 끝났습니다.

- 모든 내용을 이해 하셨나요?
- 아직 이해가 안되는 내용이 있다면 다시 한번 복습하시기 바랍니다.
- 질문은 한림 SmartLEAD 쪽지 또는 e-mail 또는 전화상담을 이용하시기 바랍니다.

- 다음 시간에는 jQuery에 대하여 알아보도록 하겠습니다

- 수고하셨습니다.^^