

## 06. 이벤트와 이벤트 처리기

---

# 이벤트 알아보기

---

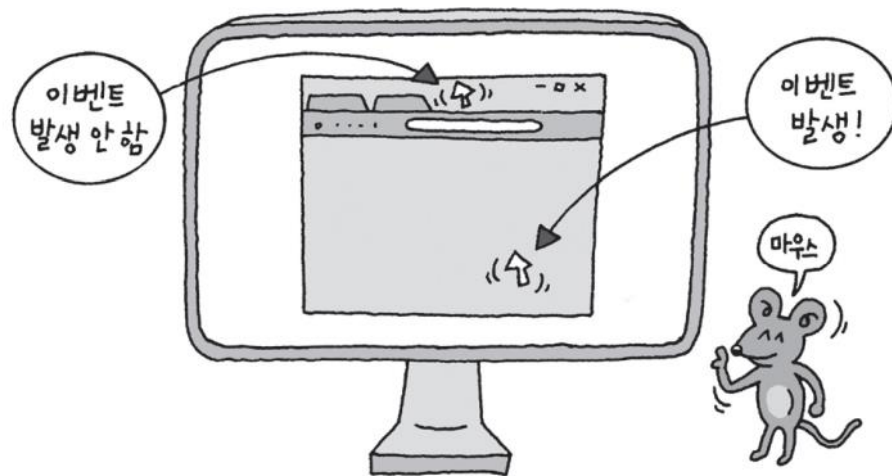
# 이벤트

이벤트(event)란, 웹 브라우저나 사용자가 실행하는 어떤 동작을 말한다.

(예) 웹 문서에서 마우스 버튼을 클릭하는 것, 웹 브라우저가 웹 페이지를 불러오는 것 등

사용자가 웹 문서 영역을 벗어나서 클릭하는 행위는 이벤트가 아니다.

(예) 브라우저 창의 제목 표시줄을 클릭하는 것은 이벤트가 아님



# 문서 로딩과 관련된 이벤트

이벤트	기능
abort	웹 문서가 완전히 로딩되기 전에 불러오기를 멈추었을 때 이벤트가 발생합니다.
error	문서가 정확히 로딩되지 않았을 때 이벤트가 발생합니다.
load	문서 로딩이 끝나면 이벤트가 발생합니다.
resize	문서 화면의 크기가 바뀌었을 때 이벤트가 발생합니다.
scroll	문서 화면이 스크롤되었을 때 이벤트가 발생합니다.
unload	문서를 벗어날 때 이벤트가 발생합니다.

# 마우스와 관련된 이벤트

마우스 버튼이나 휠 버튼을 조작할 때 발생하는 이벤트

이벤트	기능
click	사용자가 HTML 요소를 클릭했을 때 이벤트가 발생합니다.
dblclick	사용자가 HTML 요소를 더블클릭했을 때 이벤트가 발생합니다.
mousedown	사용자가 요소에서 마우스 버튼을 눌렀을 때 이벤트가 발생합니다.
mousemove	사용자가 요소에서 마우스 포인터를 움직일 때 이벤트가 발생합니다.
mouseover	마우스 포인터를 요소 위로 옮길 때 이벤트가 발생합니다.
mouseout	마우스 포인터가 요소를 벗어날 때 이벤트가 발생합니다.
mouseup	요소 위에 놓인 마우스 버튼에서 손을 뗄 때 이벤트가 발생합니다.

# 키보드와 관련된 이벤트

키보드에서 특정 키를 조작할 때 발생하는 이벤트

이벤트	기능
keydown	키를 누르는 동안 이벤트가 발생합니다.
keypress	키를 눌렀을 때 이벤트가 발생합니다.
keyup	키에서 손을 뗄 때 이벤트가 발생합니다.

# 폼과 관련된 이벤트

폼 요소에 내용을 입력하면서 발생할 수 있는 이벤트

이벤트	기능
blur	폼 요소에 포커스를 잃었을 때 이벤트가 발생합니다.
change	목록이나 체크 상태 등이 변경되었을 때 이벤트가 발생합니다(<input>, <select>, <textarea> 태그에서 사용).
focus	폼 요소에 포커스를 놓았을 때 이벤트가 발생합니다(<label>, <select>, <textarea>, <button> 태그에서 사용).
reset	폼이 리셋되었을 때 이벤트가 발생합니다.
submit	[submit] 버튼을 클릭했을 때 이벤트가 발생합니다.

---

# 이벤트 처리하기

---



# 이벤트 처리하기

이벤트가 발생하면 그에 따른 연결 동작이 있어야 한다.

이렇게 이벤트를 처리하는 것을 **이벤트 처리기** 또는 **이벤트 핸들러(event handler)**라고 한다.

## 1) HTML 태그에 연결하기

이벤트가 발생한 HTML 태그에 직접 함수를 연결한다

```
<태그 on 이벤트명 = "함수명">
```

```
<button onclick = "alert('클릭!')">Click</button>
```

HTML 태그에 스크립트를 함께 사용하기 때문에 스크립트 소스에서 함수 이름이 바뀌거나 다른 변경 내용이 있을 경우 HTML 소스도 함께 수정해야 한다

## 2) 웹 요소에 직접 함수 연결하기

스크립트 소스를 변경해도 HTML 마크업에는 영향을 주지 않게 하려면  
이벤트 처리기도 스크립트 소스에서 처리하는 것이 좋다

*요소.on 이벤트명 = 함수*

```
const button = document.querySelector("button");

button.onclick = function() {
  document.body.style.backgroundColor = "green";
}
```

클릭했을 때 (onclick)  
실행할 함수를 표현식으로 할당

함수를 미리 만들어 두었다면 그 함수를 지정해도 된다.  
이때 실행할 함수 이름 뒤에 중괄호(( ))를 사용하지 않는다.

```
function changeBackground() {
  document.body.style.backgroundColor = "green";
}

const button = document.querySelector("button");
button.onclick = changeBackground;
```

### 3) addEventListener() 사용하기


- 이벤트 리스너는 어떤 DOM 요소에서도 사용할 수 있다.
- addEventListener 함수를 사용

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

- 요소: 이벤트가 발생한 요소
- 이벤트: 이벤트 유형. 단, 여기에서는 이벤트 이름 앞에 on을 붙이지 않고 click이나 keypress처럼 이벤트 이름을 그대로 사용한다.
- 함수: 이벤트가 발생했을 때 실행할 함수. 기존에 있는 함수를 사용해도 되고 직접 익명 함수를 작성해도 된다. 익명 함수로 실행할 때는 event 객체를 사용해서 다양한 것들을 처리할 수 있다.
- 캡처 여부: 이벤트를 캡처링하는지의 여부. true면 캡처링을, false면 버블링을 한다는 의미. 선택 사항이며 기본값은 false

### 3) addEventListener() 사용하기

```
function changeBackground() {  
    document.body.style.backgroundColor = "green";  
}  
const button = document.querySelector("button");  
button.onclick = changeBackground;
```



```
function changeBackground() {  
    document.body.style.backgroundColor = "green";  
}  
const button = document.querySelector("button");  
button.addEventListener("click", changeBackground);
```

### 3) addEventListener() 사용하기

```
const button = document.querySelector("button");

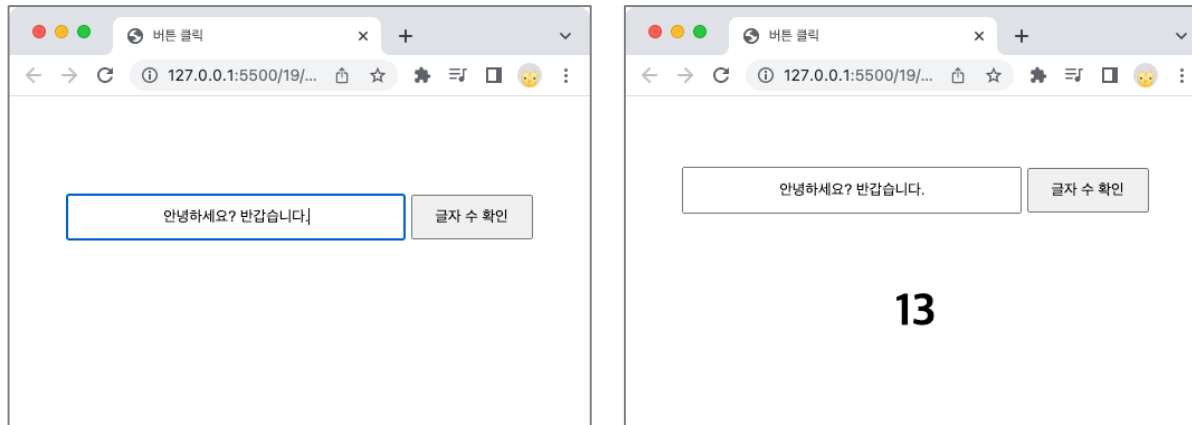
// 익명함수 사용
button.addEventListener("click", function() {
  document.body.style.backgroundColor = "green";
});
```

```
const button = document.querySelector("button");

// 화살표 함수 사용
button.addEventListener("click", () => {
  document.body.style.backgroundColor = "green";
});
```

## (예) 텍스트 필드에 입력한 글자 수 체크

텍스트 필드에 단어를 입력했을 때 단어의 길이를 화면에 표시하기.



### <참고> 문자열 길이는?

문자열의 길이는 `length` 프로퍼티  
에 들어 있다.

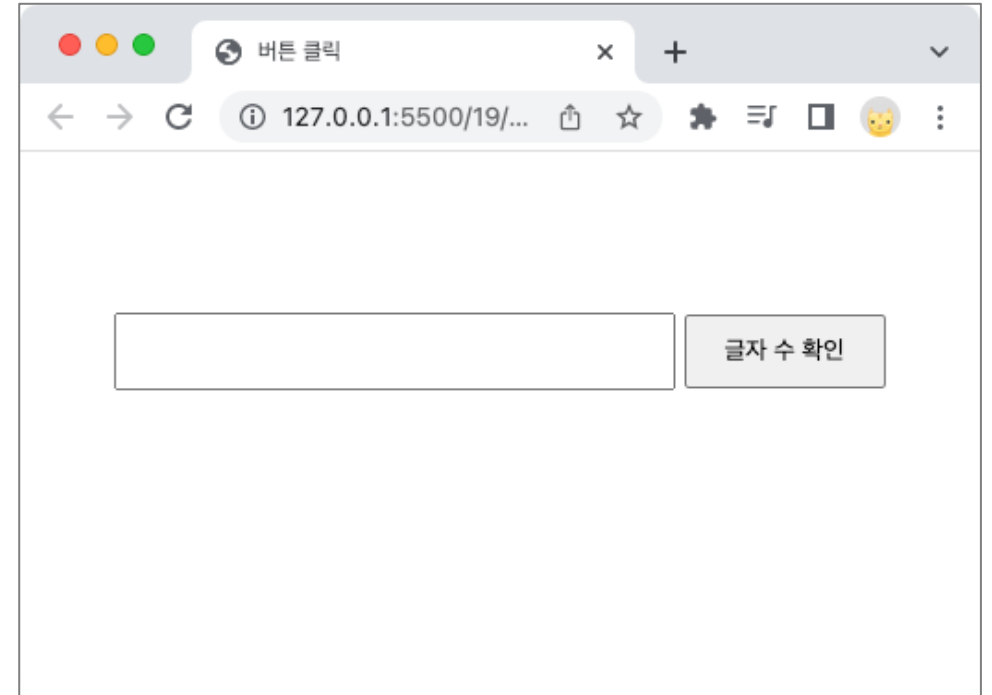
```
> str = "안녕하세요?"  
< '안녕하세요?'  
> str.length  
< 6  
>
```

## (예) 텍스트 필드에 입력한 글자 수 체크

16Wevent-5.html

```
<div id="contents">
  <input type="text" id="word">
  <button id="btn">글자 수 확인</button>
</div>

<div id="result"> </div>
```



- 1) 버튼 클릭했을 때 실행
- 2) 텍스트 필드에 있는 내용을 가져와서, 그 내용의 length 확인
- 3) length값을 결과 영역에 표시

## (예) 텍스트 필드에 입력한 글자 수 체크

06WjsWevent-5.js

```
const button = document.querySelector("#btn");

button.addEventListener("click", () => {
  const word = document.querySelector("#word").value; // 텍스트 상자의 내용
  const result = document.querySelector("#result"); // 결과값 표시할 영역
  let count = word.length; // 문자열의 길이

  result.innerText = `${count}`; // 결과값 표시
});
```



# [실습] 모달 박스 만들기

**모달 박스(modal box)** : 화면에 내용이 팝업되면서 기타 내용은 블러 처리되어 팝업된 내용에만 집중할 수 있게 해 주는 창입니다. (라이트 박스라고도 함)

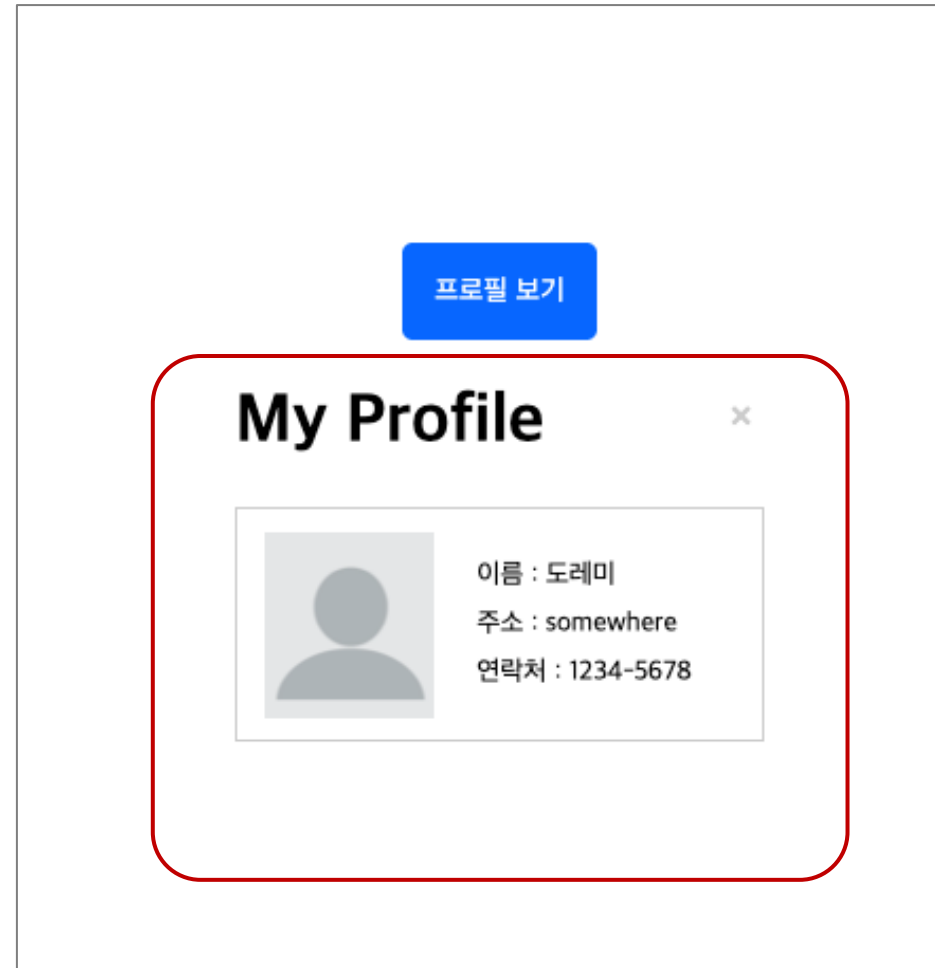
- 모달 박스에 표시되는 내용은 웹 문서 안에 미리 만들어져 있어야 한다.
- CSS와 자바스크립트를 사용해서, 처음에는 모달 박스 부분을 화면에 감춰 두었다가 버튼을 클릭하거나 특정 이벤트가 발생했을 때 모달 박스를 표시한다.

# [실습] 모달 박스 만들기

06\modal.html, 06\css\modal.css 파일 사용

```
<body>
  <button id="open">프로필 보기</button>

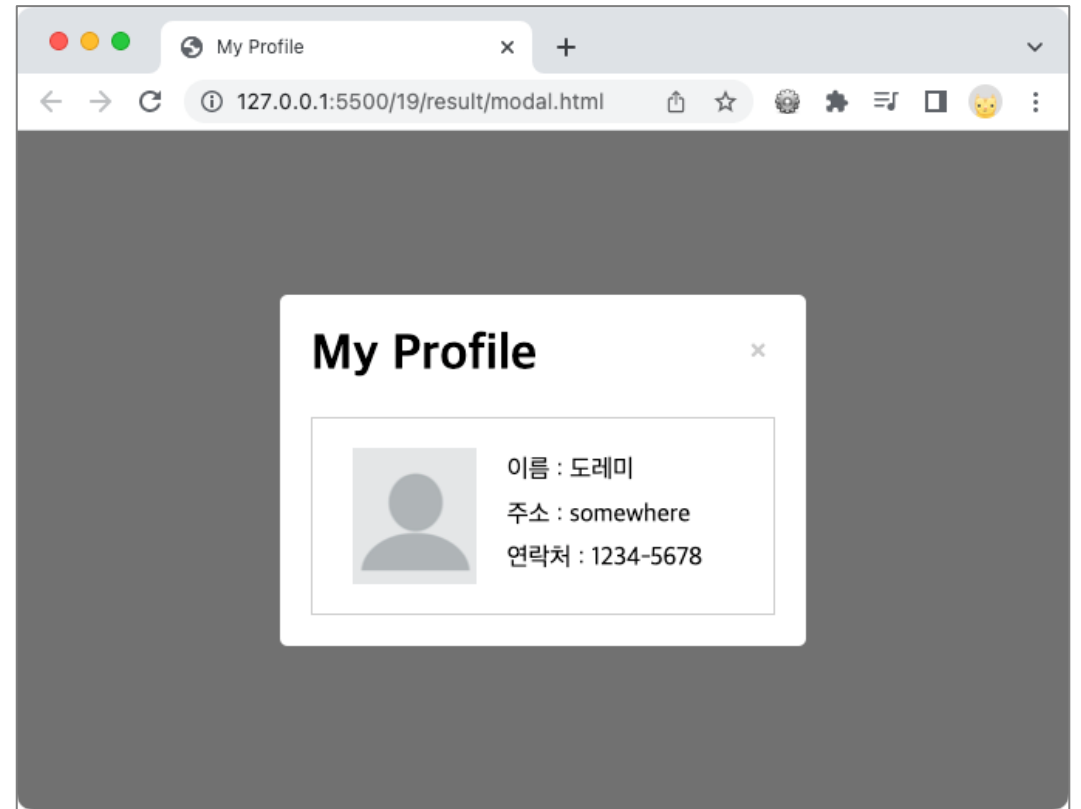
  <div id="modal-box">
    <div id="modal-contents">
      <button id="close">&times;</button>
      <h1 id="title">My Profile</h1>
      <div id="profile">
        
        <div id="desc">
          <p class="user">이름 : 도레미</p>
          <p class="user">주소 : somewhere</p>
          <p class="user">연락처 : 1234-5678</p>
        </div>
      </div>
    </div>
  </div>
</body>
```



## 1) 프로필 내용을 모달 박스 형태로 꾸며보자

06\css\modal.css 소스에 추가

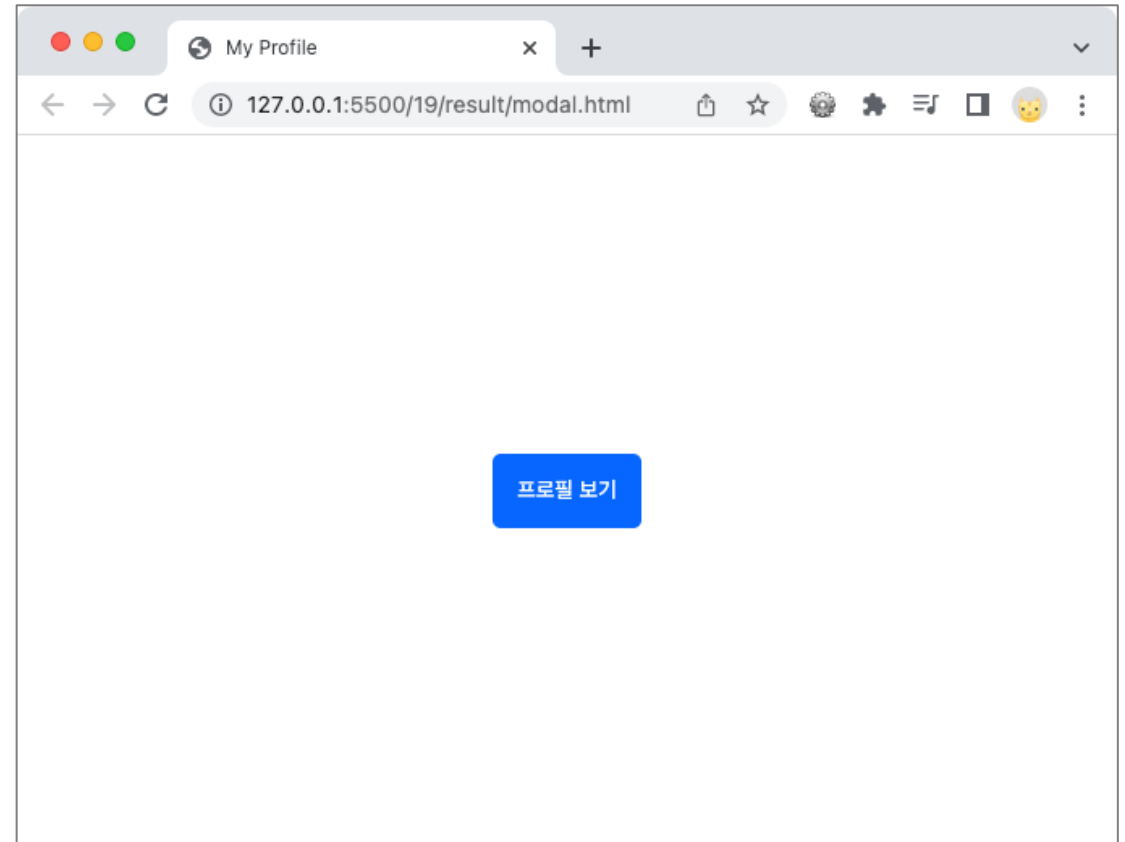
```
button { ..... }  
#modal-box{  
    position:fixed;  
    top:0;  
    left:0;  
    bottom:0;  
    right:0;  
    background-color: rgba(0,0,0,0.6);  
    display: flex;  
    justify-content: center;  
    align-items: center;  
}
```



2) 제대로 나오는 걸 확인했다면, 모달 박스는 화면에서 감췄다가 버튼을 클릭하면 나타나도록 해야 한다.

06\css\modal.css 소스에 추가

```
#modal-box{  
  .....  
  background-color: rgba(0,0,0,0.6);  
  display: none;  
  justify-content: center;  
  align-items: center;  
}  
#modal-box.active {  
  display: flex;  
}
```



- 3) modal.js 파일을 만들고 HTML 문서에 연결한다.
- 4) 파란색 버튼을 클릭하면 모달 박스 열기
- 5) 모달 박스에 있는 X 클릭하면 모달 박스 닫기

06WjsWmodal.js 에 작성

```
const open = document.querySelector("#open");
const modalBox = document.querySelector("#modal-box");
const close = document.querySelector("#close");

open.addEventListener("click", () => {
  modalBox.classList.add("active");
});
close.addEventListener("click", () => {
  modalBox.classList.remove("active");
})
```

---

**event 객체**

---

# event 객체

이벤트가 발생하면 자동으로 만들어지는 객체

event 객체의 메서드

메서드	기능
preventDefault	(취소할 수 있을 경우) 기본 동작을 취소합니다.

event 객체의 프로퍼티

프로퍼티	기능
altKey	이벤트가 발생했을 때 <b>Alt</b> 를 누르고 있었는지의 여부를 확인하고 Boolean 값을 반환합니다.
button	마우스 키를 반환합니다.
charCode	keypress 이벤트가 발생했을 때 어떤 키가 눌렸는지 유니코드값으로 반환합니다.
clientX	이벤트가 발생한 가로 위치를 반환합니다.
clientY	이벤트 발생한 세로 위치를 반환합니다.
ctrlKey	이벤트가 발생했을 때 <b>Ctrl</b> 을 누르고 있었는지의 여부를 확인하고 Boolean 값을 반환합니다.
pageX	현재 문서를 기준으로 이벤트가 발생한 가로 위치를 반환합니다.
pageY	현재 문서를 기준으로 이벤트가 발생한 세로 위치를 반환합니다.
screenX	현재 화면을 기준으로 이벤트가 발생한 가로 위치를 반환합니다.
screenY	현재 화면을 기준으로 이벤트가 발생한 세로 위치를 반환합니다.
shiftKey	이벤트가 발생했을 때 <b>Shift</b> 를 누르고 있었는지의 여부를 확인하고 Boolean 값을 반환합니다.
target	이벤트가 발생한 대상을 반환합니다.
timeStamp	이벤트가 발생한 시간을 밀리초 단위로 반환합니다.
type	발생한 이벤트 이름을 반환합니다.
which	키보드와 관련된 이벤트가 발생했을 때 키의 유니코드값을 반환합니다.

## (예) 마우스 클릭 위치 알아내기

06\event-6.html, 06\js\event-6.js

```
<p>사각형 내부를 클릭해 보세요</p>  
<div id="box"> </div>
```

- 1) #box 부분을 클릭했을 때
- 2) event 객체에서 pageX, pageY 프로퍼티 가져와서 알림창에 표시

```
const box = document.querySelector("#box");  
  
box.addEventListener("click", (e) => {  
    alert(`이벤트 발생 위치 : ${e.pageX}, ${e.pageY}`);  
});
```





## (예) 키보드 키를 눌렀을 때 키값 알아내기

키보드와 관련된 프로퍼티

- event.code : 키코드, event.key : 키 이름 (예전에는 event.keyCode를 사용했지만 지금은 폐기됨)

06₩keycode.html, 06₩js₩keycode.js

```
const body = document.body;
const result = document.querySelector("#result");

body.addEventListener("keydown", (e) => {
  result.innerText = `
    code : ${e.code},
    key : ${e.key}
  `;
});
```

**키보드의 아무 키나 눌러보세요**

code : KeyA,  
key : a

**키보드의 아무 키나 눌러보세요**

code : Digit6,  
key : 6

# [실습] 캐러셀 만들기

캐러셀(carousel) : 이미지나 콘텐츠를 슬라이드 쇼처럼 보여주는 요소

19wcarousel.html 에 작성

```
<body>
  <p>좌우 화살표를 눌러 보세요</p>

  <div id="container">
    <div class="arrow" id="left">&lang;</div>
    <div class="arrow" id="right">&rang;</div>
  </div>
</body>
```

좌우 화살표를 눌러 보세요

<  
>

# 캐러셀 스타일 지정하기

css\carousel.css 파일을 만든 후 carousel.html에 연결

19\css\carousel.css

```
* {  
  margin:0;  
  padding:0;  
  box-sizing: border-box;  
}  
body {  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
  align-items: center;  
  min-height: 100vh;  
}
```

```
p {  
  margin:20px;  
  font-size:2em;  
}  
#container{  
  position:relative;  
  width:600px;  
  height:300px;  
  border:2px solid #ccc;  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
}
```

```
.arrow {  
  z-index:100;  
  font-size:2em;  
  padding:10px;  
  background:#ddd;  
  color:#222;  
  opacity:0.2;  
}  
  
.arrow:hover {  
  opacity:1;  
}
```

# [실습] 캐러셀 만들기

js\carousel.js 파일을 만든 후 carousel.html에 연결

- 1) 캐러셀 영역의 크기(600×300)에 맞는 이미지 준비

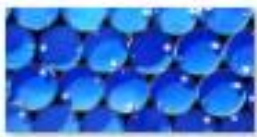
여기에서는 images 폴더에 pic-1.jpg부터 pic-5.jpg까지 모두 다섯 개의 이미지 준비



pic-1



pic-2



pic-3



pic-4



pic-5

- 2) 마크업과 분리해서 이미지를 언제든지 변경할 수 있도록 스크립트 파일에서 배열 형태로 저장한다.
- 3) 기본적으로 배열의 첫 번째 이미지를 캐러셀 영역에 보이도록 지정한다.

# [실습] 캐러셀 만들기

```
const container = document.querySelector("#container"); // 캐러셀 영역
```

```
const pics = [`pic-1.jpg`, "pic-2.jpg", "pic-3.jpg", "pic-4.jpg", "pic-5.jpg"]; // 이미지 배열
```

```
container.style.backgroundImage = `url(images/${pics[0]})`; // 첫 번째 이미지를 기본으로 표시
```

좌우 화살표를 눌러 보세요



# [실습] 캐러셀 만들기

- 1) 화살표 요소를 가져와서 `arrows`로 저장한 후
- 2) `click` 이벤트가 발생하면 왼쪽 화살표인지, 오른쪽 화살표인지 확인한다.  
`event.target`을 사용해서 `id`값이 `left`인지, `right`인지 확인
- 3) `left`라면 이전 이미지로, `right`라면 다음 이미지로 이동
- 4) 첫 번째 이미지에서 `left`를 클릭하면 마지막 이미지로 넘어가고,  
마지막 이미지에서 `right`를 클릭하면 첫 번째 이미지로 이동

# [실습] 캐러셀 만들기

```
const arrows = document.querySelectorAll(".arrow");
let i = 0;

arrows.forEach( arrow => {
  arrow.addEventListener("click", (e) => {
    if(e.target.id === "left") {
      // 이전 이미지 표시
    }
    else if (e.target.id === "right") {
      // 다음 이미지 표시
    }
    container.style.backgroundImage = `url(images/${pics[i]})`;
  });
});
```

# [실습] 캐러셀 만들기

```
arrows.forEach( arrow => {  
  arrow.addEventListener("click", (e) => {  
    if(e.target.id === "left") {  
      i--;  
      if (i < 0) {  
        i = pics.length - 1;  
      }  
    } else if (e.target.id === "right") {  
      i++;  
      if ( i >= pics.length ) {  
        i = 0;  
      }  
    }  
    container.style.backgroundImage = `url(images/${pics[i]})`;  
  });  
});
```



# [실습] 캐러셀 만들기

좌우 화살표를 눌러 보세요



---

# 이벤트 전파

---

# 이벤트가 전파된다?

- 웹 요소에서 이벤트가 발생했을 때 해당 요소에서만 이벤트가 처리되는 것이 아니라 해당 요소를 감싸고 있는 부모 요소, 그리고 그 요소의 부모 요소에서도 똑같이 이벤트가 처리된다.
- 이것을 **이벤트 전파**라고 한다.
- ‘이벤트 버블링’과 ‘이벤트 캡처링’, 두 가지 형태가 있다.

이벤트 리스너를 사용할 때 이벤트 전파 방식을 지정할 수 있다.

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

**캡처 여부:** 이벤트를 캡처링하는지의 여부를 지정한다.

# 이벤트 버블링

- 이름에서 알 수 있듯이, 위로 버블버블~
- 이벤트가 발생한 요소에서부터 부모 요소로, 다시 그 요소의 부모 요소로 이벤트가 전달된다.
- 모든 브라우저에서 대부분의 이벤트는 버블링된다.

06Wbubbling.html

BODY

```
<div onclick = "console.log('div')">
```

DIV

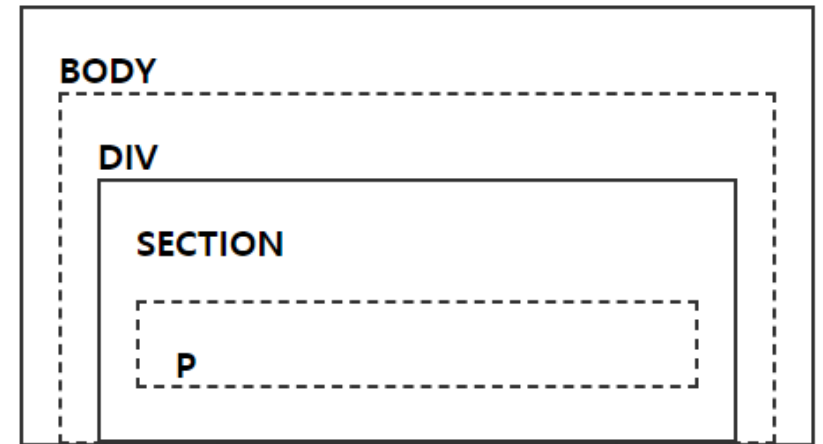
```
<section onclick = "console.log('section')">
```

SECTION

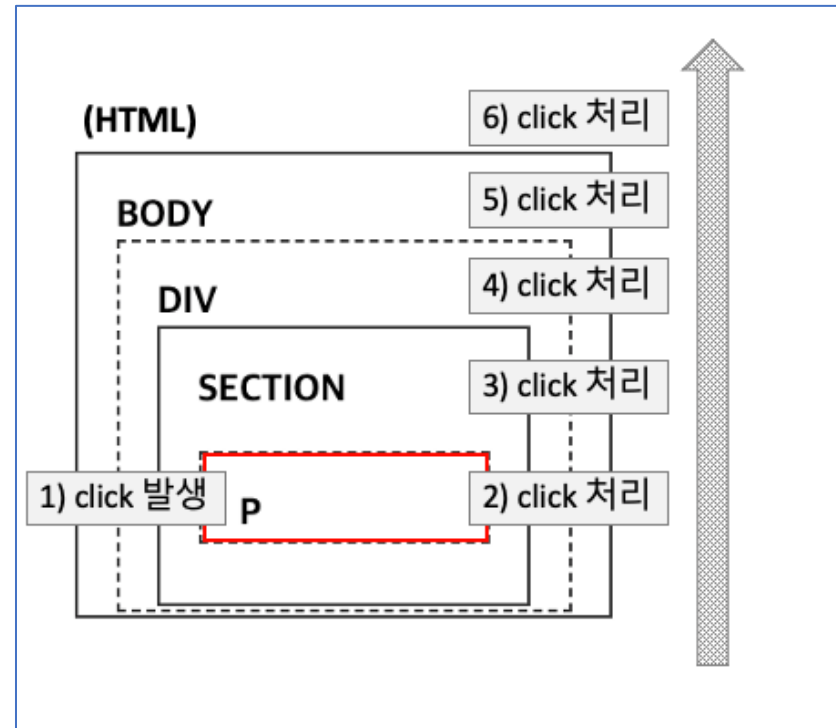
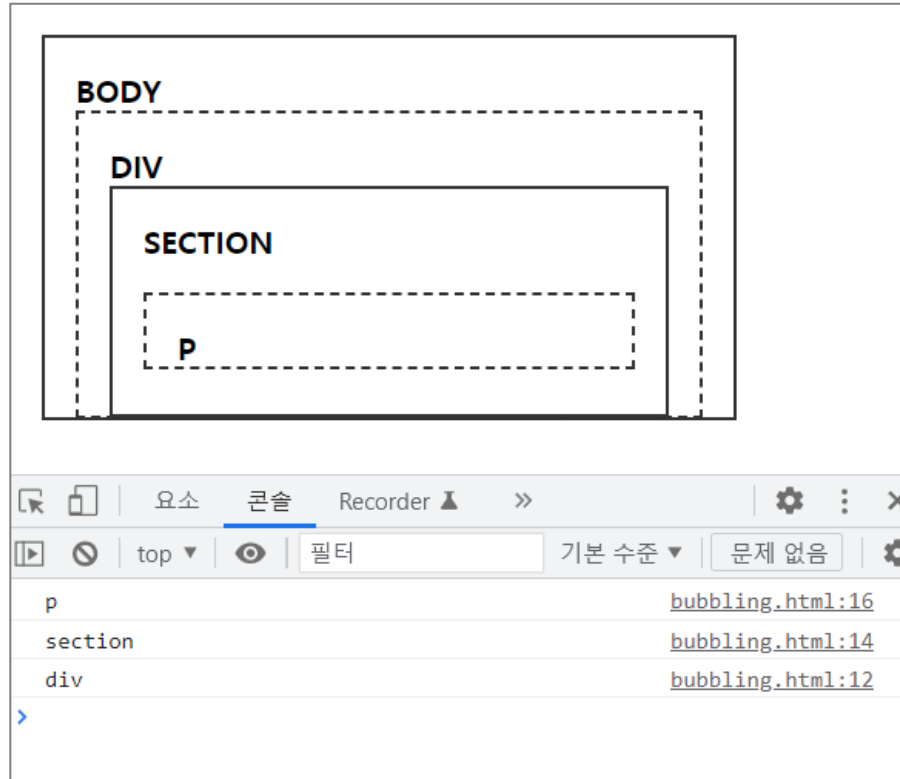
```
<p onclick = "console.log('p')">P</p>
```

```
</section>
```

```
</div>
```



- ① 웹 브라우저에서 06Wbubbling.html 문서를 열고
- ② 가장 안쪽에 있는 ' P ' 를 클릭하고 콘솔 창을 확인해 보자



# event.target과 event.currentTarget

- 이벤트가 발생하면 해당 이벤트와 관련된 정보는 event 객체에 담겨진다.
- 이벤트 전파와 관련이 있는 프로퍼티는 target 프로퍼티와 currentTarget 프로퍼티
  - target**: 이벤트가 처음 발생한 요소, **currentTarget** : 이벤트가 전파된 현재 요소

20WjsWpropagation.js

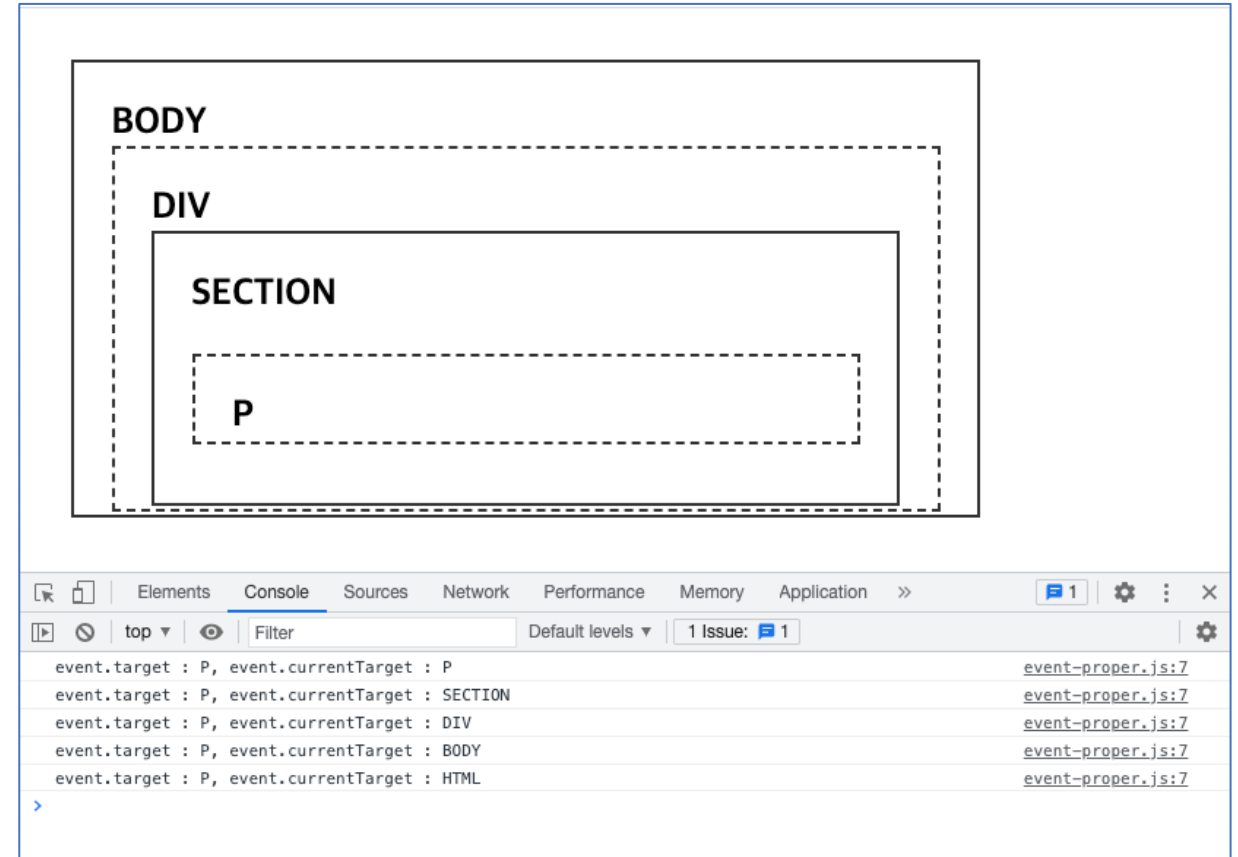
```
const elements = document.querySelectorAll('*');

for (let element of elements) {
  element.addEventListener("click", e =>
    console.log(`
      event.target : ${e.target.tagName},
      event.currentTarget : ${e.currentTarget.tagName}`
    ),
  },
}
```

이벤트 리스너에 전파 관련 옵션이 없으므로 기본값 false 사용 (이벤트 버블링)

웹 브라우저에서 06wpropagation.html을 열고 가장 안쪽의 p 요소를 클릭한다.

- 이벤트 버블링을 기본으로 하기 때문에
- p 요소에서부터 최상위 요소까지 이벤트가 전파됨.
- event.target은 계속 'P'인데,
- event.currentTarget은 'P'부터 시작해서 'HTML'까지 차례로 버블링된다.



# 이벤트 캡처링

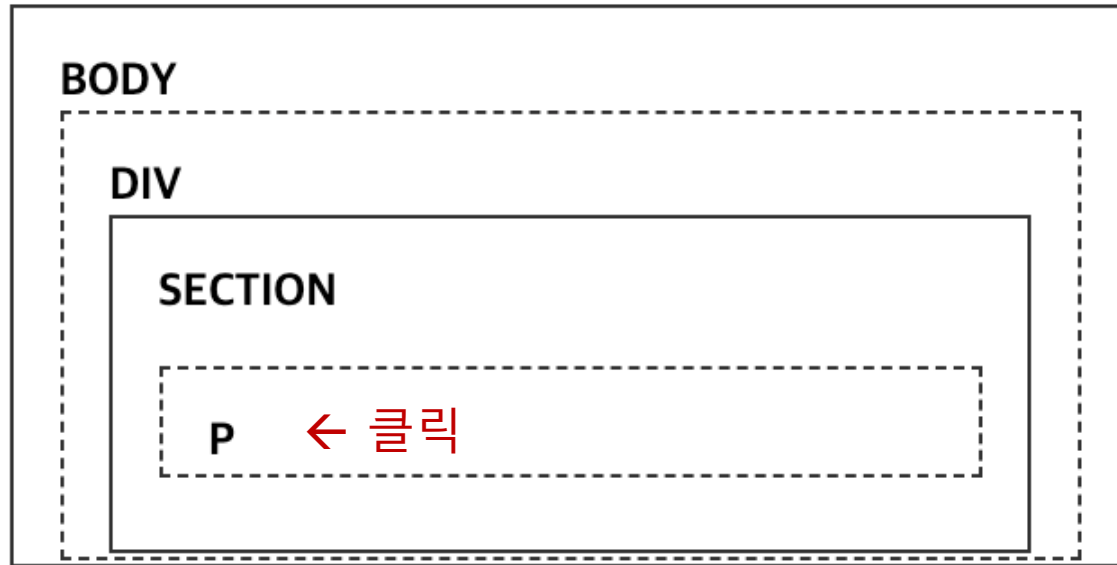
- 이벤트 캡처링은 웹 요소에서 이벤트가 발생하면 일단 최상위 요소에서 시작해서
- 이벤트가 발생한 요소까지 차례로 이벤트가 전파되는 방식
- 이벤트 캡처링을 사용하려면 이벤트 리스너 함수에서 세 번째 옵션을 'true'로 지정해야 한다.

20WjsWcapturing.js

```
const elements = document.querySelectorAll('*');

for (let element of elements) {
  element.addEventListener("click", e =>
    console.log(`
      event.target : ${e.target.tagName},
      event.currentTarget : ${e.currentTarget.tagName}`
    ),
    true);
}
```





```
event.target : P, event.currentTarget : HTML capturing.js:7
event.target : P, event.currentTarget : BODY capturing.js:7
event.target : P, event.currentTarget : DIV capturing.js:7
event.target : P, event.currentTarget : SECTION capturing.js:7
event.target : P, event.currentTarget : P capturing.js:7
```

target과 currentTarget  
값이 어떻게 달라지는지 확인