JavaScript Quick Guide

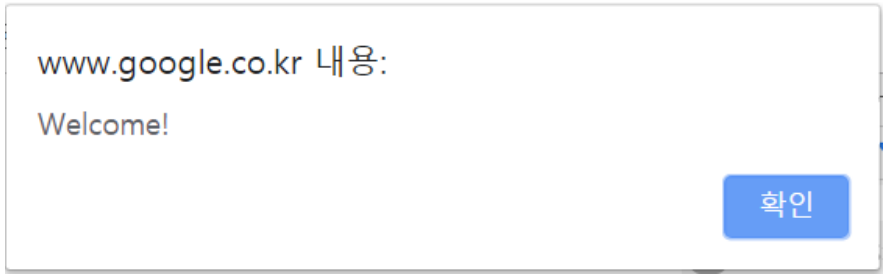
Working with DOM : Web Browser를 위한 JavaScript

웹 브라우저 대화상자

: alert vs prompt vs confirm

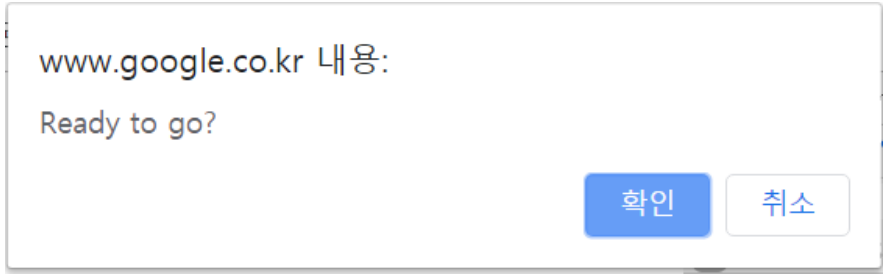
- alert : 가장 기본적인 웹 브라우저 대화상자

```
alert("Welcome!");
```



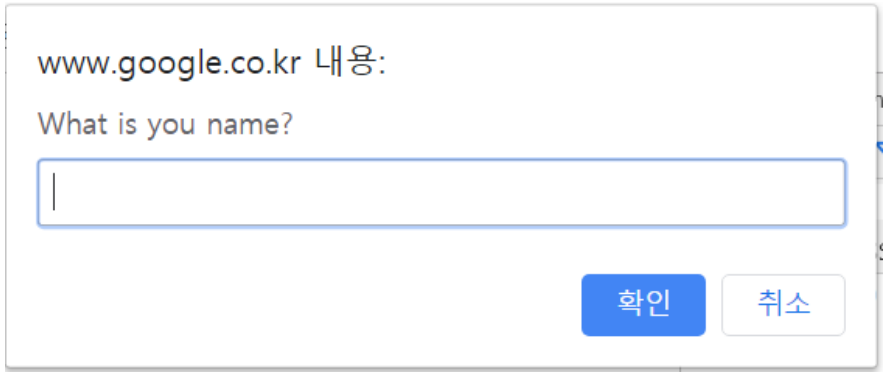
- confirm : 사용자의 선택을 요청
  - 반환값은 Boolean

```
var choice = confirm("Ready to go?");
console.log(choice);
```



- prompt : 사용자 입력 값을 받음
  - 취소시 null

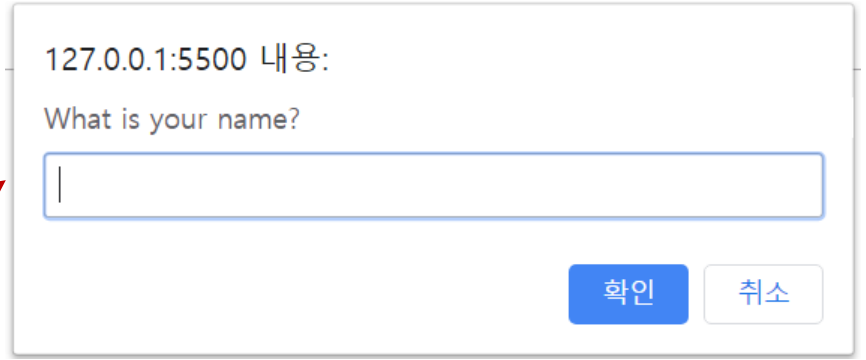
```
var name = prompt("What is you name?");
```

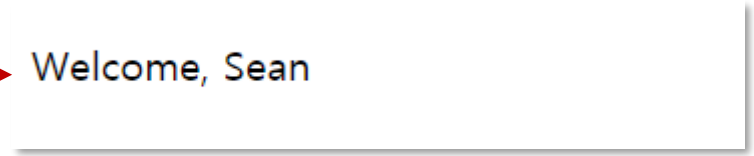


HTML에 JavaScript 삽입하기
: 내부 JavaScript

- script 태그 : HTML 문서 안에 JavaScript를 삽입
  - script 태그는 HTML 문서 어디에서든 사용할 수 있음
  - script 태그는 한 문서 안에 여러 개를 사용해도 상관 없음
  - script 태그가 삽입된 위치에서 소스가 실행(순차 실행)

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Example</title>
    <script>
      var name = prompt("What is your name?");
    </script>
  </head>
  <body>
    <script>
      document.write("<p>Welcome, " + name + "</p>");
    </script>
  </body>
</html>
```





HTML에 JavaScript 삽입하기
: 외부 JavaScript 불러오기

- script 태그의 src 속성을 이용, JavaScript 소스 파일 위치를 지정
  - 복수 개의 HTML에서 공통적인 함수 등을 작성, 별도의 파일로 분리하고 HTML에서 불러오면 개별 HTML 페이지 내에 매번 JavaScript를 작성하지 않고도 동일 코드를 재사용할 수 있다
  - 프로젝트의 규모가 커질 수록 JavaScript를 HTML로부터 분리하는 것이 효율적

```html
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Example</title>
    <script src="ask.js"></script>
  </head>
  <body>
    <script src="print.js"></script>
  </body>
</html>
```

ask.js

```
var name = prompt("What is your name?");
```

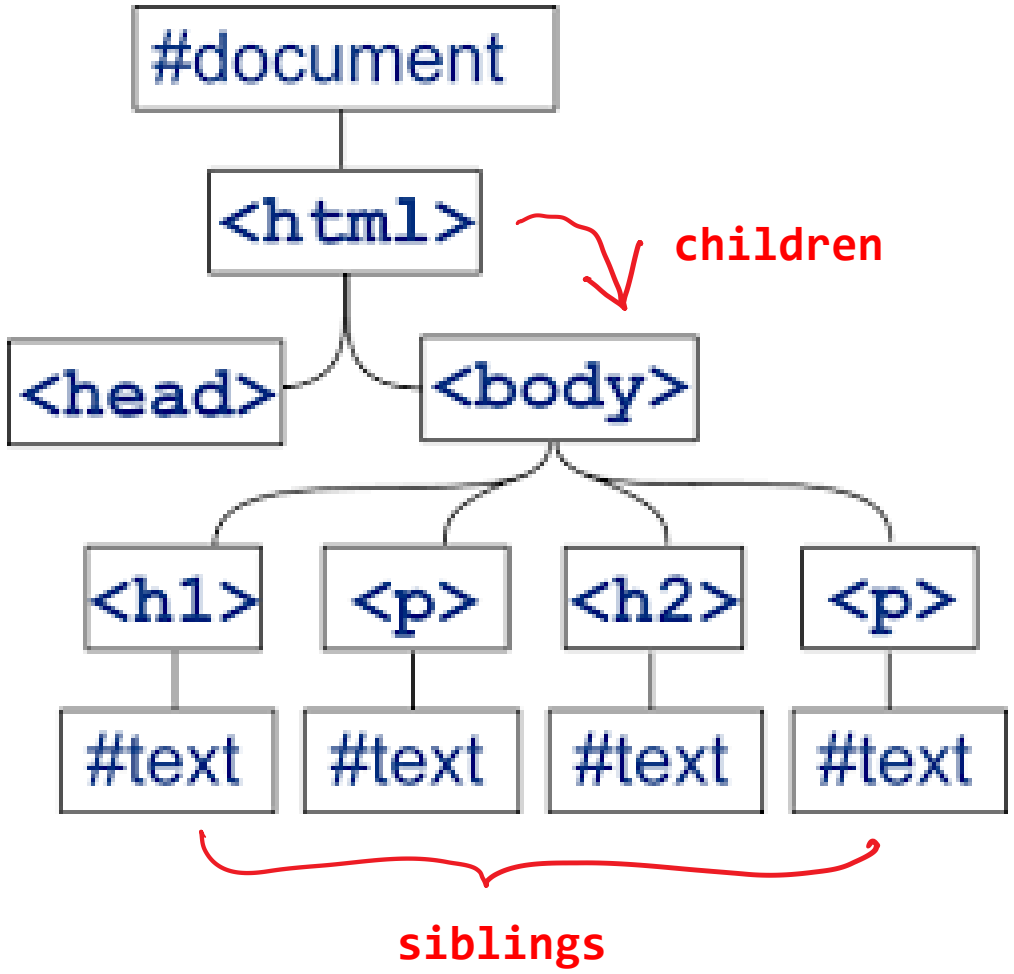
print.js

```
document.write("<p>Welcome, " + name + "</p>");
```

Document Object Model
: 간단한 이해

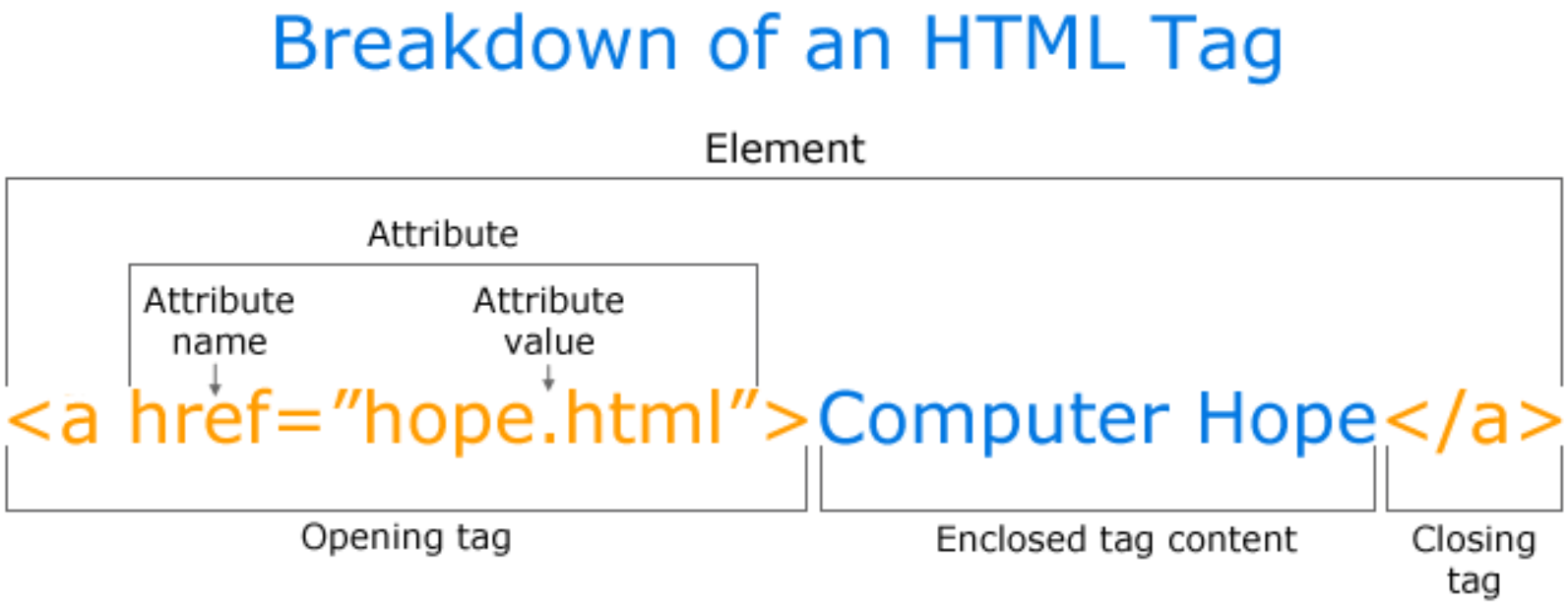
- HTML은 XML의 하위 집합
  - XML과 같이 Contents, Attribute 등을 이용, 내용과 속성에 접근할 수 있다
  - 단, XML처럼 강한 구속력이 작용하지는 않는다
- HTML은 Document Object Model이라는 Tree 구조로 구성

- Document Object Model
  - 객체 지향 모델로서 구조화된 문서를 표현하는 형식 (위키피디아)
  - 플랫폼/언어 중립적으로 구조화된 문서를 표현하는 W3C의 공식 표준



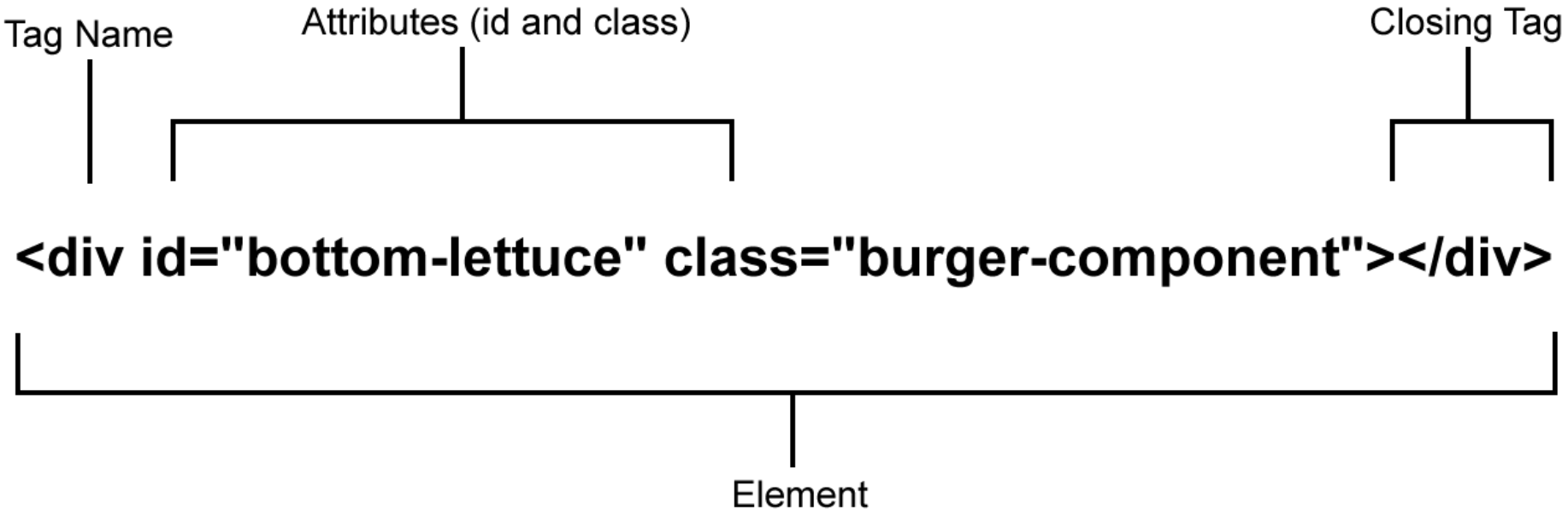
HTML 태그(Tag)와 엘리먼트(Element)
: 간단한 이해

- HTML 엘리먼트는 세 부분(여는 태그, 콘텐트, 닫는 태그)으로 구성
  - 콘텐트가 없는 엘리먼트의 경우, 닫는 태그 없이 단독으로 사용하기도 함 : 예)
  - 대부분 엘리먼트는 속성을 가지며 중첩이 가능함
- 속성(Attribute)
  - 태그의 동작을 제어하기 위해 여는 태그 안에 사용되는 특수 용어
  - 속성은 주로 속성명=속성값의 쌍으로 기술된다



id와 class 속성

- HTML 태그는 id와 class를 가질 수 있다
  - id와 class는 모든 태그가 가질 수 있는 공통 속성
  - id: 페이지 내 유일한 요소 식별 값(페이지 내 유일해야 함)
  - class: 페이지 내 동일 클래스를 가진 요소가 여럿 있을 수 있으며,
    한 요소가 여러 클래스를 가질 수도 있음



document 객체
: DOM을 다루는 브라우저 내장 객체

- document 객체 : 웹 문서(HTML) 자체를 가리키는 DOM 요소

```
console.log(document.__proto__); // HTMLDocument
```

  - 모든 자식 노드의 확인

```
console.log(document.children); // HTMLCollection
```

  - 개별 자식 노드의 확인

```
console.log(document.childElementCount)
console.log(document.children[0]); // HTML Element
```

  - document에 내용 출력: document.write()

```
document.write("Welcome!");
```

document 객체
: DOM 요소에 접근하기

- document 객체를 이용, DOM 요소에 접근할 수 있음
  - 태그명을 이용한 DOM 요소 접근 : .getElementsByTagName()

```
document.getElementsByTagName("p");
// 문서 내 모든 p 태그를 검색 - HTMLCollection
```

  - id 속성을 이용한 DOM 요소 접근 : .getElementById()

```
document.getElementById("search");
// 문서 태그 중 id가 search인 요소를 검색
```

  - class 속성을 이용한 DOM 요소 접근 : .getElementsByClassName()

```
document.getElementsByClassName("section_footer");
// 문서 태그 중 section_footer 클래스를 포함한 요소를 검색
```

document 객체
: DOM 요소에 접근하기

- document 객체를 이용, DOM 요소에 접근할 수 있음
  - Selector를 이용한 DOM 요소 탐색 : .querySelector(), .querySelectorAll()
  - .querySelector() 메서드는 셀렉터 조건을 만족하는 첫 번째 HTMLElement를 반환
  - .querySelectorAll() 메서드는 셀렉터 조건을 만족하는 HTMLCollection을 반환

```
document.querySelectorAll("div.words");
// 문서 내 words 클래스가 포함된 모든 div 노드 리스트를 반환
```

  - .querySelectorAll() 메서드는 NodeList를 반환 -> HTMLCollection과 거의 동일한 방식으로 사용 가능

태그 속성의 제어
: getAttribute, setAttribute

- HTMLElement 객체에서 속성을 가져오거나 수정하는 메서드

```
var obj = document.getElementById(…);
          document.getElementsByTagName(…);
          document.getElementsByClassName(…);
          document.querySelector(…);
          document.querySelectorAll(…)
```

```
<tag attr1="val1" attr2="val2"…>Content</tag>
```

```
obj.setAttribute("attr2", "new value");
```

속성 수정하기

속성 가져오기

```
obj.getAttribute("attr1");
```

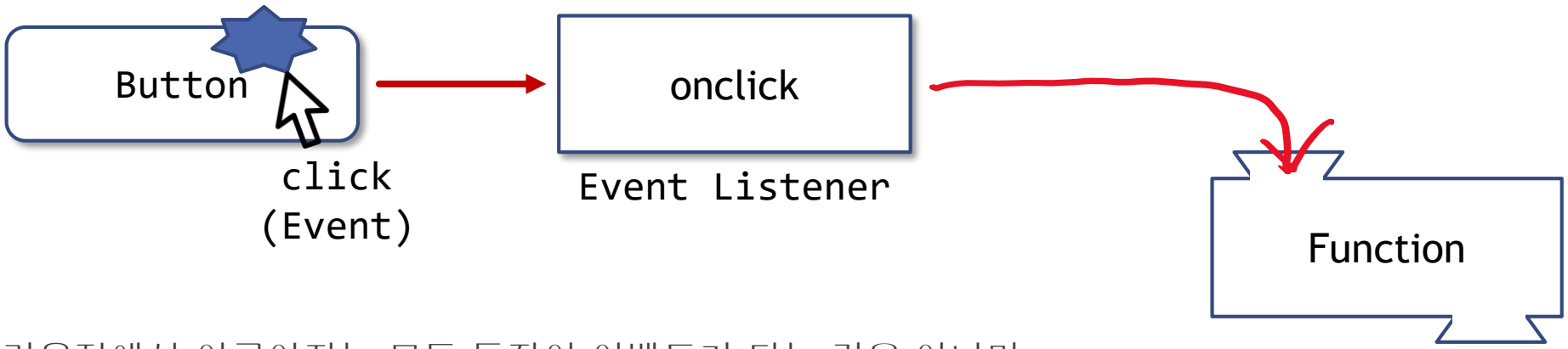
"val1"

속성 이름 조회

```
obj.getAttributeNames()
```

이벤트 리스너

- Event와 EventListener
  - Event : 웹 브라우저나 사용자가 행하는 어떤 동작.
    예) 웹 페이지의 링크 누르기, 웹 브라우저가 페이지를 로드
  - EventListener (이벤트 처리기) : Event가 발생했을 때 처리할 함수로 연결해주는 것



  - 브라우저에서 이루어지는 모든 동작이 이벤트가 되는 것은 아니며
  - 이벤트 리스너를 통해 처리 함수가 연결되어 있지 않으면 이벤트가 발생하더라도 아무 일도 일어나지 않음
  - 요소의 종류에 따라 발생시킬 수 있는 이벤트는 다를 수 있다.

이벤트 리스너
: DOM Event의 종류

이벤트 리스너의 이름 : on + 이벤트명

예: **click -> onclick**

- 마우스 이벤트

이벤트	설명
click	사용자가 요소를 마우스로 클릭했을 때
dblclick	사용자가 요소를 마우스로 두 번 클릭했을 때
mousedown	사용자가 요소를 마우스 버튼을 누를 때
mousemove	사용자가 요소 위에서 마우스 포인터를 움직일 때
mouseover	마우스 포인터가 요소 위로 올라올 때
mouseout	마우스 포인터가 요소 밖으로 벗어날 때
mouseup	사용자가 누르고 있던 마우스 버튼에서 손을 뗄 때

- 태그에 이벤트 리스너를 붙이는 방법 1: Inline 방식 - 추천하지 않음

```
<img src="Penguins.jpg" alt="펭귄" onclick="alert(this.alt);">
```

Listener JavaScript

이벤트 리스너
: DOM Event의 종류

이벤트 리스너의 이름 : on + 이벤트명

예: **load -> onload**

- 키보드 이벤트

이벤트	설명
keypress	사용자가 키보드를 누를 때
keydown	사용자가 키보드를 눌렀을 때(누른 상태)
keyup	사용자가 키보드를 눌렀다 뗄 때(해제)

- 문서 로딩 이벤트(브라우저가 발생시키는 이벤트)

이벤트	설명
abort	웹 문서가 완전히 로딩되기 전, 불러오기를 멈췄을 때
error	문서가 정확히 로딩되지 않았을 때
load	문서 로딩이 완료되었을 때. 주로 초기화 작업에 활용
resize	문서 화면 크기가 바뀌었을 때
scroll	문서 화면에 스크롤이 발생했을 때
unload	문서를 벗어날 때

이벤트 리스너
: DOM Event의 종류

이벤트 리스너의 이름 : on + 이벤트명

예: **submit -> onsubmit**

- 폼 이벤트 : 폼 입력 양식에서 발생하는 이벤트

이벤트	설명
blur	폼 요소가 포커스를 잃었을 때
change	목록, 체크 상태 등이 변경되었을 때
focus	폼 요소에 포커스가 놓였을 때
reset	폼이 리셋되었을 때
submit	submit 버튼을 눌렀을 때

- 이벤트 리스너를 붙이는 방법 2:

```
<img id="pic" src="Penguins.jpg" alt="펭귄">
<script>
    var obj = document.getElementById("pic");
    obj.onclick = function(evt) {
        alert(evt.target.alt);
    }
</script>
```

이벤트 리스너
: DOM Event의 종류

- 이벤트 리스너를 붙이는 방법 3: 가장 추천
  - .addEventListener 메서드를 이용

```
<img id="pic" src="Penguins.jpg" alt="펭귄">
<script>
function introduce(evt) {
    alert(evt.target.alt);
}
window.addEventListener("load", function() {
    var obj = document.getElementById("pic");
    obj.addEventListener("click", introduce, false);
}, false);
</script>
```

이벤트명 JavaScript

Capture 여부

true – capturing
false - bubbling

동적으로 스타일 변경하기

- HTMLElement의 style 객체를 이용하면 동적으로 요소의 스타일을 추가, 변경할 수 있다
  - 단, CSS의 속성명과 JavaScript DOM style 객체의 변수명은 다를 수 있다(JavaScript 변수 명명 규칙)

```
<style>
    #box {
        width: 100px;
        height: 100px;
        background-color: red;
    }
</style>
<div id="box"></div>
```

CSS: kebab-case

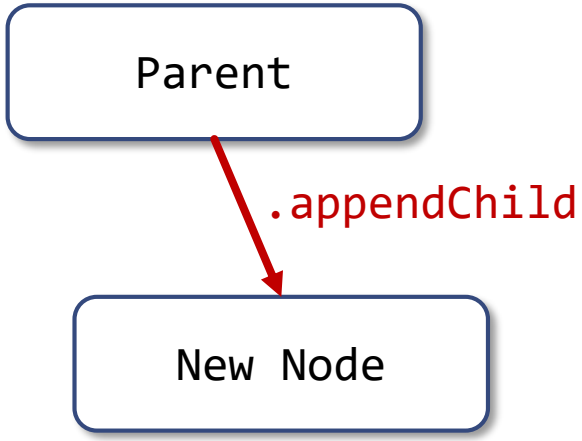
```
<div id="box"></div>

<script>
    var box =
        document.getElementById("box");
    box.style.backgroundColor = "red";
    box.style.width = "100px";
    box.style.height = "100px";
</script>
```

JavaScript: camelCase

동적으로 요소 생성하기

- DOM에 새로운 노드 추가하기
  - STEP 1. createElement로 새로운 요소 만들기
  - STEP 2. 새 요소에 속성과 내용 설정
  - STEP 3. 부모 요소에 Child로 추가(.appendChild)




```
<ul id="cart">
</ul>
<script>
var items = ["김", "단무지", "햄"];
var parent = document.getElementById("cart");
for (var i = 0; i < items.length; i++) {
    var child = document.createElement("li");
    child.innerText = items[i];
    parent.appendChild(child);
}
</script>
```

STEP 1.

STEP 2.

STEP 3.

동적으로 요소 삭제하기

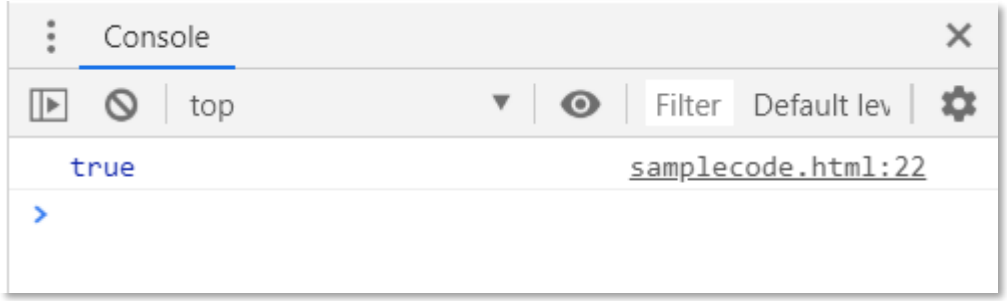
- DOM에서 기존 요소 삭제하기
  - parent에서 .removeChild() 메서드로 child 삭제

```
<ul id="cart">
    <li>김</li>
    <li>단무지</li>
    <li class="unwanted">시금치</li>
    <li>햄</li>
</ul>
<script>
    var parent = document.getElementById("cart");
    var items = document.getElementsByClassName("unwanted");
    for (var i = 0; i < items.length; i++) {
        var unwanted = items[i];
        parent.removeChild(unwanted);
    }
</script>
```

동적으로 요소 검사하기

- .hasChildNodes() : 자식 노드가 있는지 확인

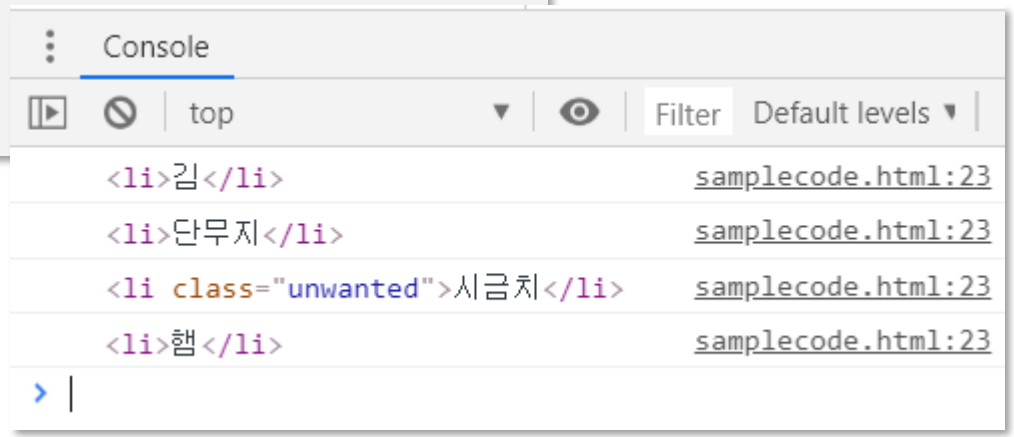
```
<ul id="cart">
    <li>김</li>
    <li>단무지</li>
    <li class="unwanted">시금치</li>
    <li>햄</li>
</ul>
<script>
    var parent = document.getElementById("cart");
    console.log(parent.hasChildNodes()); // true
</script>
```



동적으로 요소 검사하기

- .children : 자식 노드 요소 리스트에 접근

```
<ul id="cart">
    <li>김</li>
    <li>단무지</li>
    <li class="unwanted">시금치</li>
    <li>햄</li>
</ul>
<script>
    var parent = document.getElementById("cart");
    for (var i = 0; i < parent.children.length; i++) {
        console.log(parent.children[i]);
    }
</script>
```



동적으로 요소 추가하기

- insertBefore() :
  - 부모 노드에 자식 노드를 추가할 때 기준이 되는 노드를 지정하고 그 앞에 자식 노드를 추가

```
<ul id="cart">
    <li>김</li>
    <li>단무지</li>
    <li>햄</li>
</ul>
<script>
    var parent = document.getElementById("cart");
    var newItem = document.createElement("li");
    newItem.innerText = "시금치";
    parent.insertBefore(newItem, parent.children[0]);
</script>
```

새 요소를

parent의 첫 번째 child 앞에

- 시금치
- 김
- 단무지
- 햄

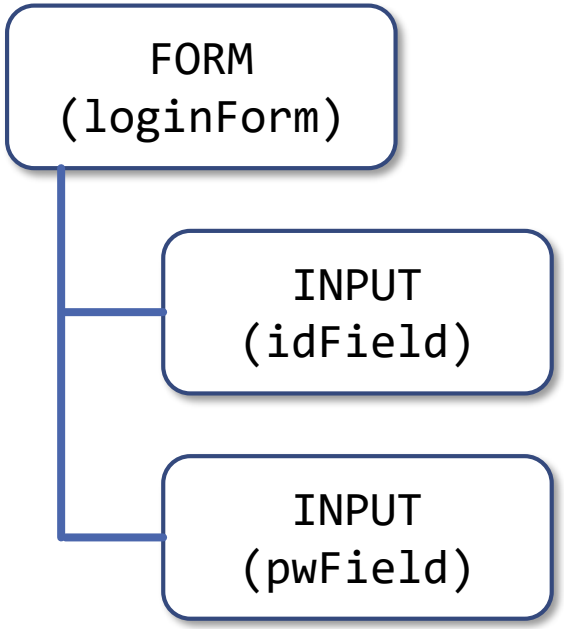
폼 요소 검증
: 폼 요소에 접근하기

- 폼 요소에 접근하기 1. id 값이나 class 값을 사용

```
<input id="nameField" type="text">
<button onclick="sayHello()">Say Hi</button>
<script>
function sayHello() {
    var nameField = document.getElementById("nameField");
    alert("Hello, " + nameField.value);
}
</script>
```

폼 요소 검증
: 폼 요소에 접근하기

- 폼 요소에 접근하기 2. name 속성 이용




```
<form name="loginForm">
    <input name="idField" type="text" />
    <input name="pwField" type="password" />
    <button onclick="checkForm();">로그인</button>
</form>
<script>
function checkForm() {
    alert("ID:" + document.loginForm.idField.value);
    alert("PW:" + document.loginForm.pwField.value);
}
</script>
```