

# 자바스크립트 이벤트핸들러

자바스크립트 BOM

자바스크립트 DOM

자바스크립트 비동기 AJAX

다음 자료는 자바스크립트의 많은 기능 중  
반드시 알아야할 기능을 수록했기 때문에  
전부다 암기하세요



# 이벤트 핸들러

## 이벤트란?

화면에서 **클릭**이나, 동작시 발생하는 **동적인 기능**입니다  
**자바스크립트의 첫번째 강력한 기능**이며, 사용방법만 암기하면 충분합니다

**태그**와 **스크립트의 이벤트**를 연결하는 방식에는  
**인라인 이벤트 모델**, **기본 이벤트 모델**, **표준 이벤트 모델**이 있으며  
이벤트 앞에 **on**을 붙입니다.

다음은 이벤트의 종류들입니다

# 이벤트 핸들러

이벤트 종류	설명
click	마우스를 클릭했을 때 이벤트 발생
dblclick	마우스를 더블클릭했을 때 이벤트 발생
mouseover	마우스를 오버했을 때 이벤트 발생
mouseout	마우스를 아웃했을 때 이벤트 발생
mousedown	마우스를 눌렀을 때 이벤트 발생
mouseup	마우스를 떼었을 때 이벤트 발생
mousemove	마우스를 움직였을 때 이벤트 발생
keydown	키를 눌렀을 때 이벤트 발생
keyup	키를 떼었을 때 이벤트 발생
keypress	키를 누른 상태에서 이벤트 발생
focus	포커스가 이동되었을 때 이벤트 발생
blur	포커스가 벗어났을 때 이벤트 발생
change	값이 변경되었을 때 이벤트 발생
submit	submit 버튼을 눌렀을 때 이벤트 발생
reset	reset 버튼을 눌렀을 때 이벤트 발생
select	input, textarea 요소 텍스트를 드래그해서 선택했을 때 이벤트 발생
load	로딩이 완료되었을 때 이벤트 발생
abort	이미지의 로딩이 중단되었을 때 이벤트 발생
resize	사이즈가 변경되었을 때 이벤트 발생
scroll	스크롤바를 움직였을 때 이벤트가 발생

select

# 인라인 이벤트 모델

인라인 이벤트 모델  
기본 이벤트 모델  
표준 이벤트 모델

인라인 이벤트 모델은 html요소에 직접 이벤트를 연결하는 방식입니다  
태그 안에 이벤트 종류가 들어가며 on이 붙습니다  
onclick같은 이벤트 안에는 **모든 스크립트 코드가 들어갈 수 있습니다.**

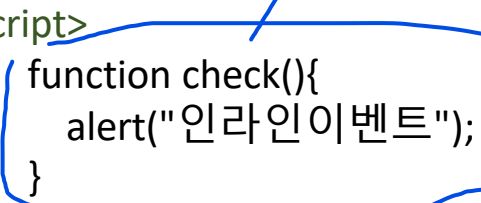
예시

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

예시

```
<button onclick="check()">인라인이벤트</button>
```

```
<script>  
  function check(){  
    alert("인라인이벤트");  
  }  
</script>
```



1. onclick은 일반적으로 가장 많이 사용되는 이벤트 입니다

2. 동일한 함수에 여러 이벤트를 걸어줄 수 있습니다

3. 이때 어떤 이벤트에 대한 동작인지 확인하는 키워드는 **this**입니다

# 기본 이벤트 모델

인라인 이벤트 모델  
기본 이벤트 모델  
표준 이벤트 모델

기본 이벤트 모델은 HTML요소를 취득 후 이벤트를 **익명함수로 연결하는 방식**입니다  
(선호 합니다)

HTML요소를 취득할 때는 순서상 "**취득할 요소**"가 "**요소 취득 명령어**" 이전에 있어야 합니다

`<button id="bt">인라인이벤트</button>` ← **취득할 요소**

`<script>`

`var bt = document.getElementById("bt");` ← **요소 취득 명령**

```
bt.onclick = function() {  
    console.log("출력");  
}
```

`</script>`

**익명함수로 연결하는 방식**

`<script>`

`window.onload = function() {`

```
var bt = document.getElementById("bt");  
bt.onclick = function() {  
    console.log("출력");  
}
```

`}`

`</script>`

`<button id="bt">인라인이벤트</button>`

HTML요소를 취득할 때는 순서상  
"**취득할 요소**"가  
"**요소 취득 명령어**" 이후에 오면 반  
드시 load이벤트를 적용해야 합니다

**반드시 외우세요**

# 기본 이벤트 모델

인라인 이벤트 모델  
기본 이벤트 모델  
표준 이벤트 모델

```
<script>
window.onload = function() {

    function test() {
        console.log("출력");
    }
    var bt = document.getElementById("bt"); ← 요소 취득 명령
    bt.onclick = test; ←
}
</script>

<button id="bt">인라인이벤트</button> ← 취득할 요소
```

앞에서 확인 했던 익명함수 방식은  
이렇게도 사용가능 합니다

단 함수 호출시 ()를 붙이지 않습니다

()를 붙이면 코드 실행시 무조건 함수가  
한번 실행되 버립니다.

# 표준 이벤트 모델

인라인 이벤트 모델  
기본 이벤트 모델  
표준 이벤트 모델

on  
↓  
가

표준 이벤트 모델은 객체.addEventListener(이벤트, 함수) 방식으로 연결합니다.

```
<script>  
window.onload = function() {  
  
    var bt = document.getElementById("bt");  
    function view() {  
        console.log("출력");  
    }  
  
    bt.addEventListener('click', view);  
}  
</script>
```

← 취득할 요소

← 이벤트에 on을 붙이지 않습니다

```
<button id="bt">인라인이벤트</button>
```

자바스크립트 이벤트핸들러

자바스크립트 BOM

자바스크립트 DOM

자바스크립트 비동기 AJAX





# BOM and DOM

## 자바스크립트의 두번째 강력한 기능

Browser Object Model **VS** Document Object Model

**BOM**은 브라우저 객체 모델이라 하며, 내장객체들을 의미합니다.  
window, location, history, document 등이 포함됩니다.



**DOM**은 문서 객체 모델이라 하며, document객체를 의미합니다.  
**DOM**은 요소(element)의 선택, 삭제 생성 등을 위해 사용합니다.

**DOM**객체를 이용한 요소의 접근은 자바스크립트의 강력한 기능이며 반드시 외워야 하는 기능입니다.

DOM먼저 살펴봅시다

# Element(태그) 노드 선택

메서드	설명
<code>getElementById()</code>	요소의 id가 xx인 태그를 선택
<code>getElementsByName()</code>	요소의 name이 xx인 태그를 전부 선택
<code>getElementsByClassName()</code>	요소의 class가 xx인 태그를 전부 선택
<code>getElementsByTagName()</code>	요소의 tag명이 xx인 태그를 전부 선택
<code>querySelector()</code>	요소 선택 방법이 css와 동일(첫번째 요소만 선택)
<code>querySelectorAll()</code>	요소 선택 방법이 css와 동일(모든 태그 선택)

`<button id="check1">체크박스요소 확인</button><br>`

```
<script>
  var check1 = document.getElementById("check1");
  check1.onclick = function() {

  }
</script>
```

많이 사용되는 id취득 방법  
요소를 check1에 저장하고 기본 이벤트 모델 사용

## 노드 선택 getElementByName()

`getElementByName()` 는 `name`인 요소들을 전부 선택합니다

```
<button id="check1">체크박스요소 확인</button><br>
<input type="checkbox" name="inter" value="1">JAVA
<input type="checkbox" name="inter" value="2">JSP
<input type="checkbox" name="inter" value="3">JSX
<input type="checkbox" name="inter" value="4">SPRING
<input type="checkbox" name="inter" value="5">DB
<input type="checkbox" name="inter" value="6">JQUERY
```

`<script>`

```
var check1 = document.getElementById("check1");
check1.onclick = function() {
```

```
    var arr = document.getElementsByName("inter");
    for(var i = 0; i < arr.length; i++) {
        console.log(arr[i]);
    }
}
```

`</script>`

`name`이 `inter`인 요소를 모두 취득  
여러 개이기 때문에 배열에 담킨다

# 노드 선택 querySelector()

querySelector() 요소 선택방법이 css와 동일(첫번째 요소만 선택)  
-css문법으로 요소를 선택한다는 강점이 있다

```
<div class="box2">
  <ul>
    <li>안녕</li>
    <li>헬로</li>
    <li>니하오</li>
  </ul>
</div>
```

```
<script>
  var test1 = document.querySelector(".box2 > ul > li");
  console.log(test1);
</script>
```

css문법으로 요소를 취득

단 첫번째 요소 <li>안녕<li>만 선택됨

# 노드 선택 querySelectorAll()

querySelectorAll() 요소 선택방법이 css와 동일(모든 요소 선택)  
-css문법으로 요소를 선택한다는 강점이 있다

```
<div class="box2">
  <ul>
    <li>안녕</li>
    <li>헬로</li>
    <li>니하오</li>
  </ul>
</div>
```

```
<script>
  var test2 = document.querySelector(".box2 > ul > li");
  console.log(test2);
</script>
```

css문법으로 요소를 취득

모든 <li>가 배열로 선택됨

# 노드css 변경하기

1. 노드의 css속성을 바꿀 때는 style속성을 이용합니다
2. css의 속성은 카멜표기법을 따릅니다.
3. 노드의 style에 전달되는 값은 문자열로 작성합니다

규칙

**노드.style.css속성 = 값**

```
<div class="box2">
  <ul>
    <li>안녕</li>
  </ul>
</div>
```

**<script>**

```
var test = document.querySelector(".box2 > ul > li");
test.style.backgroundColor = "red";
```

**</script>**

# 노드 생성, 추가

요소를 생성하는 방법입니다

함수	설명
createElement()	요소를 생성
createTextNode()	텍스트를 생성
appendChild()	요소를 부모 자식 관계로 넣어줌
innerHTML =	요소를 문자방식으로 생성
insertBefore(삽입노드, 기준노드)	기준노드 앞에 삽입노드 추가

```
<button type="button" id="add">추가</button>
```

```
<script>
```

```
function test() {  
    var div = document.createElement("div");  
    var p = document.createElement("p");  
    var text = document.createTextNode("생성!!!!");  
  
    p.appendChild(text);  
    div.appendChild(p);  
    document.body.appendChild(div);  
}  
var add = document.getElementById("add");  
add.onclick = test;
```

```
</script>
```

추가

생성!!!!

# 노드 생성, 추가

가장 기본적인 방법입니다 (문자 방식으로 태그 생성)

innerHTML = 문자열

```
<div id="inner1"></div>

<button id="btn1" >버튼1</button>
<script>
  var btn1 = document.getElementById("btn1");
  btn1.onclick = function() {
    let inner1 = document.getElementById("inner1");
    inner1.innerHTML = "<b>버튼클릭시 문자방식으로 요소를 생성합니다</b>";
  }
</script>
```

버튼클릭시 문자방식으로 요소를 생성합니다

버튼1





# 노드 삭제

요소를 생성하는 방법입니다

함수	설명
remove()	노드 삭제
removeChild(삭제 할 자식 노드)	자식 노드 삭제



# 부모 노드, 자식 노드 선택

노드 간 부모자식을 선택하는 방법입니다

기능	설명
childNodes	모든 자식 노드 선택 (단, 노드에 생략된 text도 포함된다
children	모든 자식 노드 선택
parentElement	부모노드 선택
nextElementSibling	다음 형제 노드 선택
previousElementSibling	이전 형제 노드 선택
firstChild	첫번째 자식 노드 선택
lastChild	마지막 자식 노드 선택

# 노드의 속성 추가 및 제거

메서드	설명
getAttribute()	요소의 속성 취득
setAttribute()	요소의 속성 저장
removeAttribute()	요소의 속성 제거

```
<form action="##" id="form">  
  <input type="hidden" value="11" id="num">  
</form>
```

단 input으로 사용자가 변경하는 값은 처리하지 못합니다

```
<script>  
  var num = document.getElementById("num").getAttribute("value");  
  
  if(num == "") {  
    document.getElementById("form").setAttribute("action", "https://www.naver.com");  
  } else {  
    document.getElementById("form").setAttribute("action", "https://www.google.com");  
  }  
</script>
```

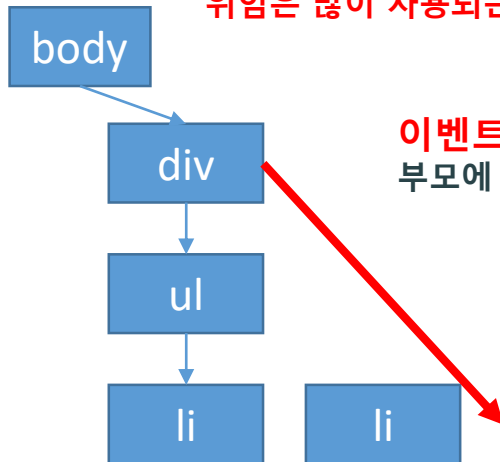
# 이벤트 객체

이벤트 발생시 실행되는 함수의 (인자값) 으로 현재 실행되는 event객체를 넣어주게 되어있다.

이벤트 객체의 기능	설명
stopPropagation()	이벤트 전파를 막는다. (버블링 중단하기)
target	이벤트를 적용한 타겟 속성
currentTarget	실제 이벤트가 걸려있는 타겟 속성
preventDefault()	고유특성을 가진 태그의 이벤트를 제거

이벤트 전파는 부모에 하나의 이벤트만 걸어 놓으면, 모든 이벤트가 동일하게 자식으로 위임되는 특징입니다.

**위임은 많이 사용되는 강력한 이벤트 핸들링 패턴입니다**



## 이벤트 위임

부모에 이벤트를 걸어 두면 자식이 동일한 이벤트를 위임 받는 특징.

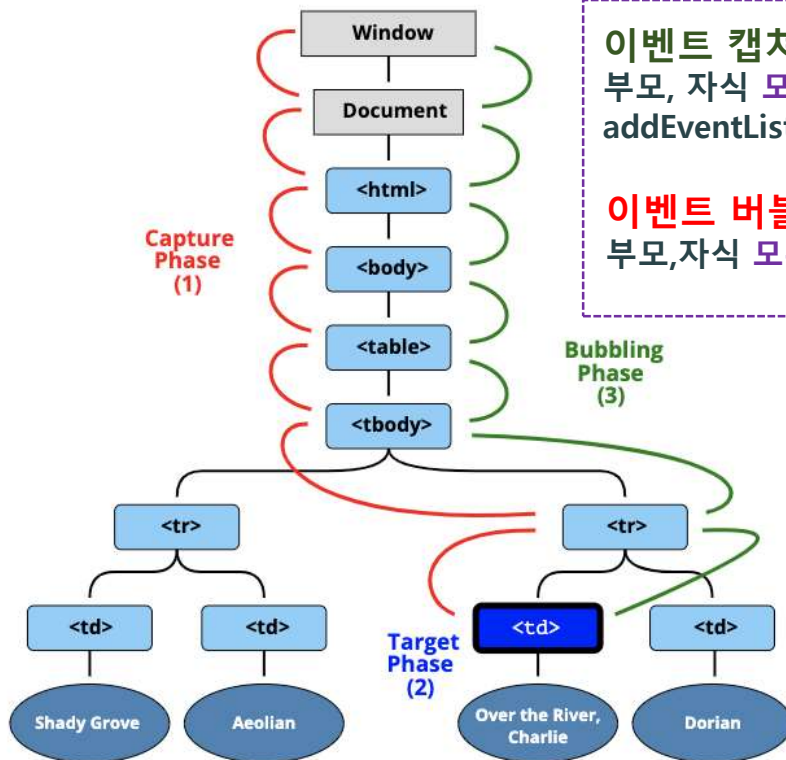
div에 이벤트를 걸어 놓으면  
ul, li에 동일 이벤트를 위임 받게 된다

**이벤트 위임을 잘 이용하면 모든 li에 일일이 이벤트를 등록하는 작업을 없앨 수 있다.**

Ex) li에 동일한 이벤트 1000개를 걸어야 할 경우, ul에 이벤트 하나만 등록하면 동일 이벤트를 위임 받는다

# 이벤트 객체(이벤트 위임의 원리가 되는 버블링, 캡처링)

DOM의 이벤트 실행 방식 - DOM의 이벤트 동작 방식은 버블링이다



## 이벤트 캡처링

부모, 자식 모두 이벤트가 걸려있을 때 실행순서가 부모 -> 자식으로 실행되는 특징  
addEventListener()로만 구현이 가능하다

## 이벤트 버블링

부모, 자식 모두 이벤트가 걸려있을 때 실행순서가 자식 -> 부모로 실행되는 특징

## 이벤트 실행방식

1. 캡처링 -> 2. target -> 3. 버블링

기본적으로 선언하는 이벤트함수는 버블링으로 실행된다

이벤트 버블링과 캡처링은 단순히 이해만 하면 됩니다!

버블링은 거의 모든 상황에 적용됩니다. 반대로 캡처링은 구현할 일이 거의 없습니다.

특정한 경우가 아니라면 버블링을 막지 마세요. (문제가 되는 경우가 많을 수 있습니다)

# form객체

form객체는 **document 객체의 하위 객체** 중 하나입니다.  
form을 이용하면 form 유효성 검사 등을 할 수 있습니다.

form객체는 document의 하위 객체이므로 유일하게 **name속성으로 선택이 가능합니다**

ex) document.폼명.요소명

속성 값	설명
value	input, textarea 요소의 value값을 반환합니다.
checked	checkbox나 radio가 체크되어 있으면 true, 체크되어 있지 않으면 false를 반환합니다
disabled	요소가 활성화 상태이면 false, 비활성 상태이면 true를 반환
length	요소의 개수를 반환
focus()	요소의 포커스를 맞춥니다
blur()	요소의 포커스를 없애 줍니다
submit()	form을 값을 전송합니다
reset()	form의 값을 초기화 합니다

전부 암기

# 사용 예시

```
<form name="regForm">
  아이디: <input type="text" name="id"><br>
  비밀번호: <input type="password" name="pw"><br>
  <button type="button" id="regist" onclick="check()">가입</button>
</form>
```

```
<script>
function check() {
  if(document.regForm.id.value == "") {
    alert("아이디를 입력하세요");
    return;
  } else if(document.regForm.id.value.length < 4) {
    alert("아이디는 4글자 이상으로 입력하세요");
    return;
  }
  .....생략
}
</script>
```

# BOM

## 자바스크립트의 두번째 강력한 기능

Browser Object Model VS Document Object Model

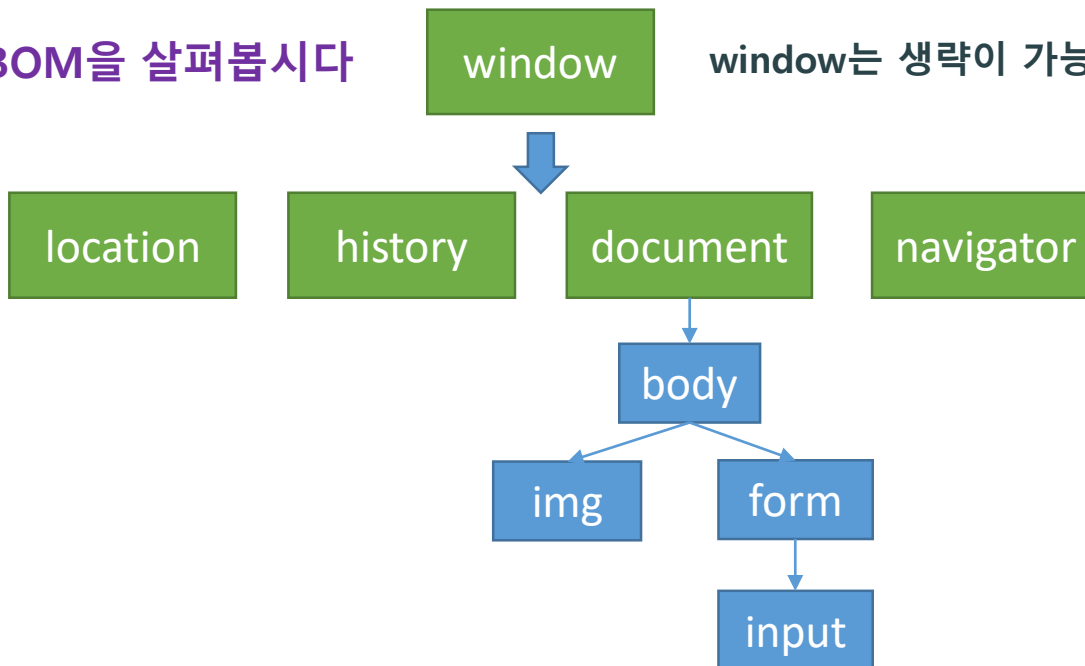
**BOM**은 브라우저 객체 모델이라 하며, 내장객체들을 의미합니다.  
window, location, history, document 등이 포함됩니다.

window 객체는 웹브라우저에 대한 전반적인 정보 취득이나 제어에 관련된 객체이며  
하위 객체에는 location, history, document 등이 있습니다

BOM을 살펴봅시다

window

window는 생략이 가능합니다





# window객체, location객체, history객체

window.open() – 새창을 띄워주는 메서드 입니다.

## 사용방법

window.open(문서 주소, 이름, “옵션=값, 옵션=값, 옵션=값” );

옵션	설명
width = 픽셀값	가로 넓이
height = 픽셀값	세로 넓이
left = 픽셀값	위치
top = 픽셀값	위치
location = yes 또는 no	윈도우 주소창 show/hide
scrollbars = yes 또는 no	윈도우 스크롤바 show/hide
menubar = yes 또는 no	윈도우 메뉴바 show/hide
toolbar = yes 또는 no	윈도우 툴바 show/hide
status = yes 또는 no	윈도우 상태줄 show/hide

하나로 묶는 거에 주의 하세요

# window객체, location객체, history객체

## window객체-기본함수

함수	설명
alert()	경고창
confirm()	확인창
setTimeout()	일정 시간이 지난 후 함수 실행
clearTimeout()	setTimeout중지 메서드
setInterval()	일정 시간마다 함수 반복 실행
clearInterval()	setInterval중지 메서드

## location객체

자바스크립트 페이지 이동

location.href = 주소;

자바스크립트 새로그침

location.reload()

## history객체

history.go(-1); //기록이동

history.back(); //뒤로가기

history.replaceState(저장할데이터 , 바꿀제목 , 바뀐주소 ); //새로운기록추가

history.state //페이지데이터



# navigator객체

## navigator객체 – 주요 함수

함수	설명
appName()	브라우저 이름을 얻어옵니다
geolocation.getCurrentPosition()	현재 위치 정보를 얻어옵니다

자바스크립트 이벤트핸들러

자바스크립트 BOM

자바스크립트 DOM

자바스크립트 비동기 AJAX



# AJAX(비동기 통신)

AJAX ( Asynchronous Javascript and XML) 은 웹 페이지의 이동없이 필요한 데이터만 전송하는 기술입니다.

일반적인 경우 데이터 처리는 요청 순서대로 진행하지만 **AJAX는 순차적으로 진행하지 않습니다.** 이런 방식을 **비동기 방식** 이라고 합니다.

자바스크립트의 비동기 방식은 상당히 까다로운데, 이를 간단히 사용할 수 있게 해주는 최신 자바스크립트 API인 **fetch API**를 이용하도록 하겠습니다.

참고문헌: [https://developer.mozilla.org/ko/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/ko/docs/Web/API/Fetch_API)

```
fetch('test').then(function(response) {  
    response.text().then(function(text) {  
        alert(text);  
    })  
})
```

fetch API 일반적 구조

## API가 뭔가요?

우리 기능을 사용 할려면 너는  
x1, x2, x3, x4, x5 .... x100을 처리해야해

하지만 너무 어렵고 많은 것을 해야 하잖아?  
그럼 우리가 사용방법을 정의해서 줄게  
너는 이렇게 사용해!!!

Application Programming Interface(API)

# fetch API - AJAX

```
fetch('test').then(function(response) {  
    response.text().then(function(text) {  
        alert(text);  
    })  
})
```

상당히 어려워 보이는 코드지만 아주 단순한 코드입니다



fetch('test')는 서버에 요청하는 코드예요 -> 코드로 확인  
then()은 요청 후에 1시간 동안 응답이 오지 않으면 1시간을 기다려야 겠죠?  
그러면 응답이 오면 then이라는 함수를 실행 시킬게! 스크립트야 너는 다른 일을 하고 있어! 라는 **promise**입니다



```
var callback = function() {  
    내용..  
}  
fetch('test').then(callback);
```

**promise**는 자바스크립트 비동기처리에  
이용되는 객체 입니다. 즉, 응답 후에 실행시켜 줄게  
라는 약속인 거예요

fetch(url)로 요청이 실행되고, 끝나고 나면 .then 함수를 실행 할게!  
그리고 끝나면 callback 이라는 메서드를 실행해줘!

응답이 왔을 때 실행 시켜주는 함수를  
프로그래머들은 **callback** 함수라고 불러요

다만 callback이라는 함수가 익명 함수로 들어온 것 뿐이죠!

# fetch API - AJAX

```
fetch('test').then(function(response) {  
    response.text().then(function(text) {  
        alert(text);  
    })  
})
```

**response**는 then() 을 실행시킬 때, 콜백함수에 response라는 응답 객체를 넣어 줄게!  
그냥 response로 선언하고 받아서 쓰면 되!



```
fetch('test').then(function(response) {  
    response.text().then(function(text) {  
        alert(text);  
    })  
})
```

매개변수로 response를 **text()** 텍스트로 바꾸면 **text**라는 매개변수로 넣어 줄게!  
여기서 **text**는 바뀔 수 있어요!

나머지 기능은 코드로 확인해봅시다!

# JSON이란

JSON(JavaScript Object Notation)은 자바스크립트 객체로 구성된 데이터입니다  
자바스크립트 객체 형태의 문자열인 셈입니다

서블릿(서버)에서 AJAX로 응답할 때 **객체**를 어떻게 줄 것인가?  
방법이 있는가???

그렇다면 JSON형태의 문자열로 응답해 줄테니 받아서 쓰세요!

ex)

{키 : 값, 키 : 값, 키 : [배열] }

그렇다면 서버에서는 JSON을 이용해서 어떻게 응답할 수 있는가?

1. JSON객체를 지원해주는 라이브러리가 필요합니다. (JSON-SIMPLE 라이브러리)
2. response객체의 설정을 JSON형식으로 지정합니다

**코드**

```
JSONObject json = new JSONObject();  
json.put("키" : " 값");  
response.setContentType("application/json");  
response.setCharacterEncoding("UTF-8");  
response.getWriter().write(json.toString());
```





## Chapter 2

# 수고하셨습니다