**문서 객체모델(DOM)**

-HTML 문서를 내부적으로 표현하는 표준모델

-HTML 문서의 계층적인 구조를 트리로 표현함

-노드의 종류

DOCUMENT\_NODE – DOM 트리의 루트 노드이며 HTML 문서를 나타냄.

ex) window.document

ELEMENT\_NODE – HTML 요소를 나타냄

ex) <body>, <a>, <p>, <script>, <style>, <html>, <h1> 등

ATTRIBUTE\_NODE – 속성을 나타냄.

ex) class=”myClass” 와 같은 속성을 나타냄

TEXT\_NODE – 요소 안에 들어있는 텍스트

**HTML 요소찾기**

-자바스크립트를 이용해 HTML 요소를 조작하여 동적 웹페이지를 작성하려면 원하는 요소를 찾아야함

-id로 찾기

1. document.getElementById(“main”); 문서 안에서 id가 main인 요소를 찾아서 반환함

요소자체를 반환함.

2. document.getElementById(“main”).innerHTML;

요소의 내용을 반환함.

-입력 양식 찾기

1. HTML 문서 안에 있는 모든 입력양식(form)은 forms 배열안에 전부 들어있음

2. 기술한 순서대로 들어있음. 따라서 순서대로 forms[0] 로 접근 가능

3. form 안에 input요소들은 document.forms[0].elements[0] … 로 접근 가능

4. 만약 입력 양식과 입력 필드에 name 속성이 지정돼있으면 찾기가 더욱 쉽다.

<form name=”myFrom”> // document.myFrom

<input type=”text” id=”target1” name=”text1”> // document.myform.text1

</form>

-태그 이름으로 찾기 : getElementsByTagName(태그명문자열)

1. var eleArray = document.getElementsByTagName(“div”); //<div> 태그를 모두 찾아서 반환

2. 반환된 모든 요소는 배열 형태로 들어가 있음

3. var list = document.getElementsByTagName(‘ul’)[0];

var allItems = list.getElementsByTagName(‘li’);

-DOM 트리 순회

1. 자식 노드와 부모 노드 관계를 이용하여 한노드씩 방문하는 방법

2. childNodes[], nextSibling[], parentNode[] 를 이용

3. childNodes : 한 요소의 모든 자식요소에 접근 가능. 배열로 반환됨.

firstChild : ‘childNodes’ 배열의 첫번째 자식노드가 반환됨. == childNodes[0]

lastChild : 마지막 자식노드 반환 == childNodes[childNodes.length-1]

parentNode : 현재 노드의 부모 노드를 반환한다.

nextSibling : 현재 노드의 다음 형제 노드를 반환함

previousSibling : 현재 노드의 이전 형제 노드를 반환한다.

**HTML 변경하기**

-자바스크립트를 통해 HTML 문서의 DOM을 변경할 수 있음. 단, HTML 문서가 직접 변경되는 것은

아니고 DOM만 변경됨.

-요소의 내용 변경하기

-.innerHTML : 요소의 시작태그와 종료태그 사이에 놓여진 모든 HTML 코드와 텍스트.

ex) document.getElementById(“main”).innerHTML = “변경”;

<div id=”main”>초기 내용</div>

-<strong> 같은 것도 중간에 있으면 변경 가능

-요소의 속성 변경하기

-.src : 요소에 있는 속성을 변경할 수 있음

ex) document.getElementById(“image”).src = “poodle.png”

<img id=”image” src=”pome.png” width=”120” height=”100”> 여기서 src 부분이 바뀜

* 이미지 주소가 바뀌는 것

-요소의 스타일 변경

-.style 로 바꿀 수 있음.

-.style.color / .style.fontFamily / .style.fontStyle / .style.visibility 등

**DOM 노드 삭제와 추가**

-새로운 HTML 요소 생성

1. 추가하기를 원하는 노드를 생성함

var node = document.createTextNode(“동적으로 추가된 노드”);

2. 문서 내에서 추가할 위치를 찾는다.

var parent = document.getElementById(“target”);

3. 새로운 노드를 기존의 노드에 연결한다.

parent.appendChild(node);

* 제일 뒤에 추가됨. 제일 앞에 추가하고 싶으면 밑 코드 사용

insertBefore(node, document.getElementById(‘target’).firstChild);

-기존 HTML 요소 삭제

1. 부모노드를 찾음

var parent = document.getElementById(“target”);

2. 자식 노드를 찾는다.

var child = document.getElementById(“p1”) //target안에 p1를 갖는 요소 있음

3. 삭제

parent.removeChild(child);

**브라우저 객체모델**

-웹 브라우저가 가지고 있는 모든 객체를 의미

-최상위 객체는 window고 그 아래로 navigator, location, history, screen, document, frames 가 있음

-종류

window 메인 브라우저 윈도우

window.navigator 브라우저에 대한 정보(버전 정보 등)

window.screen 사용자 화면

window.history 사용자가 방문한 URL 저장

window.location 현재 URL

window.frames 브라우저 윈도우를 차지하고 있는 프레임들

window.document 메인브라우저에 표시된 HTML 문서

-window 객체

-모든 전역 자바스크립트 객체, 함수, 변수는 자동적으로 window 객체의 멤버가 됨

-alert(), prompt() 등

-open() : 새 브라우저 윈도우를 오픈함

window.open(URL, name, specs, replace);

-setTimeout() : 일정한 시간이 지난 후에, 인수로 전달된 함수를 딱 한번만 호출함

setTimeout(호출되는 함수 이름, millise)

-setInterval() : 일정한 시간마다 주기적으로 함수를 호출함. 반드시 개발자가 종료시켜야함.

id = setInterval(함수이름, millisec);

clearInterval(id); //setInterval() 종료

//setInterval()이 반환한 값을 전역으로 저장해서 clearInterval()에 집어넣으면 종료됨

-moveTo() : 윈도우 절대적 이동, moveBy() : 윈도우 상대적 이동

-resizeTo() : 윈도우 절대적 크기 변경, resizeBy() : 윈도우 상대적 크기 변경

-focus() : 특정 윈도우로 키보드 포커스 이동

-scrollTo() : 스크롤 위치 절대적 변경 / scrollBy() : 스크롤 위치 상대적 변경

-Screen 객체

-availHeight 화면의 높이를 반환(테스크 바 제외)

-availWidth 화면의 너비를 반환(태스크 바 제외)

-colorDepth 컬러 팔레트의 비트 깊이를 반환

-height 화면의 전체 높이를 반환

-pixelDepth 화면의 컬러 해상도를 반환

-width 화면의 전체 너비를 반환

-Location 객체

-hash URL 중에서 앵커부분을 반환(#section1과 같은 부분)

-host URL 중에서 hostname과 port를 반환

-hostname URL 중에서 hostname을 반환

-href 전체 URL을 반환

-pathname URL 중에서 경로(path)를 반환

-port URL 중에서 port를 반환

-protocol URL 중에서 protocol 부분을 반환

-search URL 중에서 쿼리부분을 반환

-assign() 새로운 문서를 로드함

-reload() 현재문서를 다시 로드함

-replace() 현재 문서를 새로운 문서로 대체함.

-navigator 객체

-appCodeName 브라우저의 코드 네임

-appName 브라우저의 이름

-appVersion 브라우저의 버전 정보

-cookieEnabled 브라우저에서 쿠키가 활성화되어 있는지 여부

-onLine 브라우저가 인터넷에 연결되어 있으면 true

-platform 브라우저가 컴파일 된 플랫폼

-userAgent 브라우저에서 서버로가는 user-agent 헤더

-JavaEnabled() 자바가 사용가능한지 여부

-taintEnabled() 브라우저에서 data tainting이 가능한지 여부

**이벤트 처리**

-이벤트 : 웹 페이지에서 발생한 상호작용

ex)마우스 클릭, 웹페이지 로딩, 호버링, 키보드 입력

-이벤트 발생 -> 등록된 함수가 실행 -> 페이지에 적용

-onclick : 클릭하면 발생

<h1 onclick=”함수명”>~~~</h1>

-onload / onunload : 각각 웹페이지에 진입할 때, 떠날때 발생 <body> 에 넣는다.

<body onload = “함수명”>

-onchange : 요소의 내용이 변경되면 호출됨. 입력 필드를 검증할 때 종종 사용됨

-onmouseover / onmouseout : 요소 위에 마우스를 올리거나 뺄 때 발생됨

-onmousedown / onmouseup / onclick : 마우스 누름 / 마우스 뗌 / 마우스 클릭 완료

-onfocus /onblur : 마우스로 필드를 선택했을 때 / 필드에서 나갔을 때

-이때 필드에서 나갈 때, 들어올 때랑 달리 요소의 내용이 변경되면 onchange도 발생

**폼의 유효성 검증**

-사용자가 폼의 내용을 잘못 입력했는지 자바스크립트에서 확인하고 보낼 수 있다.

-필수적인 필드를 채웠는가?

-유효한 길이의 텍스트를 입력하였는가

-유효한 날짜를 입력하였는가

-숫자 필드에 텍스트를 입력하지 않았는가

-폼 데이터 접근

-폼 안에 잇는 필드의 id나 name 속성을 이용해야함

-id는 페이지의 요소를 식별함 / name은 폼 내부에서 필드를 식별함

-id면 getElementById() 사용, name이면 form[“이름”]으로 접근

-데이터 검증을 할 때는 onblur로 검증 : 필드를 비워놓는 경우에도 검사하므로

-공백 검증

function checkNotEmpty(field) {

if(field.value.length == 0)

….

…

}

….

이름 : <input type=’text’ id=’user’>

<input type=’button’ onclick=”checkNotEmpty(document.getElementById(‘user’))” value=’확인’>

-데이터 길이 검증

-위처럼 비슷하게 하면 됨. check함수는 field와 min, max 세개를 받음

-정규식으로 데이터 형식 검증

-정규식 표현 – 책보고 확인

var exp = /^\d{10}$/; //수가 10번 나와야함

if(!exp.test(field.value)) // 정규식과 일치하면 true반환

-선택 검증

-다음처럼 첫번째 옵션을 도움말로 두고, 첫번째가 선택되면 실패 발생

<select id=”fruits” class=”required”>

<option value=”0”>선택하세요</option>

<option value=”1”>사과</option>

<option value=”2”>배</option>

</select>

<input type=’button’ onclick=”check(document.getElementById(‘fruits’), ‘메시지’”) value=”확인”>

들어온 fruits의 value가 선택하세요 이면, 들어온 메시지 출력