Javascript

참조 사이트

- 모던 자바스크립트 튜토리얼
  - https://ko.javascript.info/
- MDN (Mozilla Developer Network) JavaScript 레퍼런스
  - https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference
- W3School JavaScript 튜토리얼
  - https://www.w3schools.com/

웹 문서의 구성요소

- HTML
  - 웹 문서 내용의 구조를 정의
- CSS(Cascade Style Sheet)
  - 웹 문서의 디자인을 처리
- JavaScript
  - 웹 문서 내에서 동적인 부분을 처리

JavaScript 개요

- 웹 문서(HTML) 내에서 실행되는 프로그램을 구현하기 위한 언어
  - 웹 문서 내에서 요청 파라미터 검증 등 사용자의 요청에 대한 동적 처리를 위해 만들어짐
    - Node.js 이후 웹브라우저 뿐 아니라 standalone 환경에서도 실행할 수 있게 되어 개발 가능한 application의 범위가 넓어짐.
- JavaScript 엔진
  - JavaScript 프로그램을 실행 하기 위한 실행환경.
  - 웹 문서내에 구현된 javascript는 Web Browser 에 내장 되어 있는 JavaScript 엔진에 의해 실행된다.

주요 JavaScript framework

- Web Frontend framework
  - **jQuery**
  - **React**
  - **Vue.js**
  - **svelte**
- Backend Server application framework
  - **node.js**
- Mobile application framework
  - **React native**
- Desktop application framework
  - **Electron**

HTML 페이지에 JavaScript 코드 추가

- <script> 태그 내에 작성
  - script 태그는 HTML 문서 내 어디든지 올 수 있다.

```
<script>
    document.write("안녕하세요");
</script>
```

- 외부 JavaScript 코드 추가
  - JavaScript 코드를 HTML 외부에 작성 - **.js 로 저장**
  - <script **src="파일 URL"**></script>
  - 태그 내에 JavaScript 코드를 넣어서는 안 된다.

```
<script src="scripts/myscript.js"></script>
<script>
    //코드
</scrip>
```

자바스크립트 구문

- 주석
  - // : 한줄 주석
  - /* 주석 */ : block 주석
    - 여러줄 주석이나 코드 사이에 주석을 넣을 때 사용
- **; (세미콜론)**
  - 각 명령문들의 구분자로 한 명령문이 끝나면 뒤에 붙인다.
  - 한 줄에 여러 명령문을 작성할 경우 반드시 붙인다.
  - 한 줄에 한 명령문씩 작성할 경우 생략 가능하다.
    - 생략 가능하지만 붙여주는 것이 권장된다.
- **{ }**
  - 코드 블록
  - 명령문들을 묶을 때 사용한다.

Data Type

- JavaScript 데이터 타입 – 리터럴 타입과 , 객체(참조)타입이 있다.
- **리터럴(Literal) 타입**
  - **number** : 정수, 실수
  - **string** : 문자열
    - 값은 따옴표로 감싼다. (큰따옴표, 작은 따옴표 상관없음)
    - 여러 줄 문자열의 경우 ` `(백틱)으로 감싼다. 변수의 값을 이용한 format 문자열도 지원한다.
  - **boolean** : 논리값 – **true/false**
  - **null** : 값이 없음
  - **undefined** : 값이 없음
- **객체(Object) 타입**
  - **object : 객체**
  - **array** : 배열
  - **function** : 함수
- **typeof** 연산자
  - 값의 Data Type을 문자열로 알려주는 연산자
  - 구문
    - **typeof 값; typeof 변수;**

```
name="김영수"
age="20"
`이름 : ${name}
나이: ${age}`
```

```
typeof  "hello";   //string
typeof  20;        //number
typeof  true;      //boolean
```

변수

- 변수 선언
  - 변수는 값을 저장하는 저장소 이며 모든 타입의 값을 다 대입할 수 있다.
  - **var** 변수명 [= 값], **let** 변수명 [= 값], **const** 변수명 = 값
    - **var** 변수명 :
      - 선언된 함수의 **전 영역에서** 사용할 수 있다. (function scope)
      - 같은 이름의 변수를 여러 개 선언 할 수 있다. (재정의)
    - **let** 변수명 :
      - **변수가 선언된 block 내에서만 사용할 수 있다.** (block scope)
      - 같은 scope에 같은 이름으로 변수를 재 선언 할 수 없다.
    - **const** 변수명 :
      - **상수** 선언. 한번 값이 할당되면 변경할 수 없다. 같은 scope에 같은 이름으로 변수를 재 선언 할 수 없다.
      - **선언된 block 내에서만 사용할 수 있다.** (block scope)
    - var 보다 **let이나 const를 사용을 권장 한다.**

변수

- 변수 명 규칙
  - 사용할 수 있는 문자 : 유니코드 일반 문자, 숫자, 특수문자 $, _
  - 숫자는 두 번째 글자부터 사용가능
  - 키워드는 사용할 수 없다.
  - **관례**적으로 카멜 표기법을 사용한다.

```
var num = 20;
var address = "서울시 종로구";
var num1, num2, num3;
var num1, num2=20, num3=40;
var ageList = [10, 20, 30, 40, 50];
```

```
let num = 20;
let address = "서울시 종로구";
let num1, num2, num3;
let num1, num2=20, num3=40;
let ageList = [10, 20, 30, 40, 50];
```

```
const num = 10
num = 20 (X)
```

```
function test() {
    var num = 10;
    for (let idx=1; idx < 10; idx++) {
        console.log(num); //사용가능
        console.log(idx); //사용할 가능.
    }
    console.log(num); //사용가능
    console.log(idx); //사용할 수 없다.
}
```

변수 – 값 대입

- 변수에 값 대입하기
  - Literal 값 대입
    - =
    - x = 10; y = 20;
  - 값 직접 대입: var y = 20;
  - 표현식(연산식)으로 대입
    - 숫자 표현식 : var num = 6+7; var num = (10/2) * 3; var num = 10.22-3.21 ;
    - 논리 표현식 : var flag = x >2; var flag = 10 == 15;
    - 문자열 표현식 : var str = "super"+"computer"; var date = "3월"+21+"일";
  - 다른 변수의 값 대입 : var x = y;
  - 다른 function 의 실행 결과 대입 : var k = abc();

연산자

- 대입 연산자
  - =
  - 변수에 값을 대입(할당) 한다.
  - 변수 연산자= 값
  - 변수가 가진 값에 값을 연산한 결과를 변수에 다시 대입한다.
    - x += 10; y *= x
- 산술연산자
  - +, -, *
  - / : 나누기, %: 나머지 연산자
- 증감 연산자
  - 변수++, ++변수 : 변수의 값을 1 증가 시킨다.
  - 변수--, --변수: 변수의 값을 1 감소 시킨다.
  - 변수 앞에 붙이면 증감연산을 다른 연산보다 먼저 한다.
  - 변수 뒤에 붙이면 다른 연산을 증감 연산보다 먼저 한다.

```
let a = 10
let b = a++
console.log(a, b) // 11, 10
```

```
let a = 10
let b = ++a
console.log(a, b) // 11, 11
```

연산자

- 문자열 연산자
  - +를 이용해 두 문자열 또는 문자열 또는 다른 타입의 값을 붙인다.
  - “a”+“b” : “ab”
  - “결과 : “+10 : “결과 : 10”
- 템플릿 문자열을 이용한 문자열 합치기
  - ` (backtick) 으로 감싼다.
  - 변수가 들어갈 자리를 ${ } 로 묶어준다.
  - 여러 줄 입력도 가능하다.
  - ${ } 안에서 연산식도 사용할 수 있다.

```
let name = "홍길동";
let age = 20
let info = `이름 : ${name}, 나이: ${age}`
```

```
let value = `1번줄
2번줄
3번줄`
```

```
a = 10, b = 20, c=30
expr = `${a}+${b}*${c}=${a+b*c}`
```

연산자

- 비교연산자
  - 크기 비교 : **> , >=, < , <=**
  - 동등 비교
    - **==, !=** (타입과 상관없이 값이 같은 지 여부 비교. 피연산자 타입이 다르면 타입을 맞춘 뒤 비교)
    - **=== , !==** (값 뿐만 아니라 타입도 같은 지 여부까지 비교)
    - 10 == "10" : true, 10 === "10" : false
- 논리연산자
  - **&&**
    - 피연산자 false가 있으면 false. 둘다 true이면 true를 반환
  - **||**
    - 피연산자에 true가 있으면 true, 둘다 false 이면 false를 반환
  - **!**
    - 단항 연산자로 true를 false로 false를 true로 바꾼다.
- 조건 연산자 (삼항 연산자)
  - 조건 **?** 참반환값 **:** 거짓반환값
  - var s = i>=0 ? "양수" : "음수 "

조건문

- if – else 문

```
구문
if (조건){
    구문
} else if (조건){
    구문
} else {
    구문
}
```

조건이 true이 block을 실행한다
조건 : 모든 타입의 값이 다 들어올 수 있다.

타입 별 false인 값

- string : 빈 문자열
- number : 0
- null, undefined

- switch case 문

```
switch (표현식) {
  case 값 :
    구문
    [break;]
  case 값 : 구문
  default : 구문
}
```

- 표현식 : 문자열, 숫자, 논리형 모두 올 수 있다
- 표현식 == 값 인 case의 구문 부터 break를 실행할 때 까지 나머지 구문을 모두 실행한다.

반복문

- **for, while, do while, for in**
- for 구문 – 반복횟수가 정해져 있는 경우 사용하는 것이 좋다.

```
for ([초기식]; [조건식]; [증감식]) {
        반복할 구문
}
```

```
let sum = 0;
for(let i = 0; i < 10; i++){
  sum = sum +i;
}
```

- while 구문 – 반복횟수가 정해져 있지 않은 경우 사용하는 것이 좋다.

```
[초기식]
while (조건식) {
    반복구문
    [증감식]
}
```

```
let i =0;
let sum = 0;
while(i < 10){
    sum = sum +i;
    i++;
}
```

반복문

- do while문: 1회이상 반복할 때 사용하는 것이 좋다.

```
[초기식]
do{
      반복구문
      [증감식]
}while(조건식);
```

```
let sum = 0;
let i = 0;
do{
     sum = sum +i;
     i++;
}while(i < 10);
```

- for in - 객체의 **속성명**(instance 변수)이나 배열의 **index**를 반복 조회

```
for(변수 in 객체||배열){
          반복 구문
}
```

```
let obj = {name:"김영수", age:20}
for(let property in obj){
   console.log(obj[property]);
}

let arr =  [1,2,3,4,5]
for (let index in arr) {
     console.log(arr[index])
}
```

반복문

- **break** – 반복문을 종료한다.

```
for (let i = 0; i < 10; i++) {
  if (I == 3) {
     break
  }
  .....
}
```

- **continue** – 현재 반복을 종료하고 다음 반복을 실행한다.

```
for (let i = 0; i < 10; i++) {
  if (i % 3 == 0) {
     continue
  }
  .....
}
```

function - 함수

- 프로그램 수행의 일련 과정을 하나의 단위로 묶어주는 수행 block
  - 구현된 script 실행 시 바로 실행되지 않고 호출 시 실행된다.
  - 반복 호출이 가능.
- **함수는 객체이다. ➔ 함수는 값이다.**
  - **JavaScript에서는 함수가 객체로 취급된다.**
  - **전달인자**나 **리턴 값**으로 사용할 수도 있다.

함수 기본 구문

```
function 함수이름(arg [,arg]){
        //함수 코드
        [return value]
}
```

```
function test(name){
    alert(name);
    return name+"님";
}
```

함수 정의 표현식(함수 리터럴)

```
함수이름 = function(arg [,arg]){
        //함수코드
        [return value]
}
```

```
let test = function(name){
    alert(name);
    return name+"님";
}
```

function - 함수

- Rest Parameter
  - 마지막 매개변수를 **...변수명** 으로 선언하면 호출 시 여러 개의 값을 배열로 받을 수 있다.
    - rest parameter는 반드시 마지막 매개변수로 하나만 선언할 수 있다.

```
function test (num1, num2, ...num3) {
    console.log(num1, num2, num3)
}
```

```
test(10, 20, 30, 40, 50, 60, 70)
```

    - 위 예제에서 10=>num1, 20=>num2 에 대입되며 나머지는 num3에 배열로 대입된다.

function - 함수

- 화살표 함수 (arrow function)
  - 익명함수를 만들기 위한 문법
    - expression을 반환

변수 = (arg1, arg2, ...) => expression	let func = (num1, num2) => num1 + num2

- 매개변수가 없을 경우 빈 괄호 사용(생략하면 안됨). 매개변수가 하나일 경우 ( ) 생략 가능

let func = () => console.log("hello")	let func = num => num * 10

- 구현부가 여러 줄일 경우 { } 중괄호로 묶어준다. 중괄호 사용시 반환값이 있을 경우 return 문을 사용해야 한다.

```
let func = (num1, num2) =>{
    result = num1 + num2;
    return result
}
```

Event 처리

- 자바 스크립트는 사용자의 요청처리를 Event 모델을 통해 처리한다.
  - Event Source : Event가 발생 한 컴포넌트
  - Event
    - Event Source의 상태를 바꾸게 만드는 Action(동작).
    - Ex) 마우스로 버튼 클릭, Text 입력양식에 글 입력, 페이지 로딩 등..
  - Event Handler
    - Event 발생시 처리하는 코드를 등록하는 것.
  - Listener
    - Event Source에서 Event가 발생하는 것을 감시하다 발생시 Handler 호출
    - Web Browser

예

```
<input type="button" value="확인" onclick="alert('button클릭')"/>
<form action="a.jsp" onsubmit="alert('전송합니다.')">
```

Event 처리

- 주요 Event와 Handler

Event	Handler	설명
load	**onload**	해당 페이지가 로딩 되었을 때(처음 읽힐 때) 발생
focus	**onfocus**	입력 양식를 선택해서 포커스가 주어졌을 때
blur	**onblur**	포커스가 폼의 입력 양식을 벗어났을 때
change	**onchange**	입력 양식 select에서 선택 item이 바뀌었을때
mousemove	**onmousemove**	해당영역에 마우스를 움직였을 때 발생
mouseover	**onmouseover**	해당 영역에 마우스가 올라갔을 때 발생
mouseout	**onmouseout**	해당 영역에서 마우스가 나갔을 때 발생
mousedown	**onmousedown**	해당 영역에서 마우스 버튼을 눌렀을 때 발생
mouseup	**onmouseup**	해당 영역에서 누르던 마우스 버튼을 떼었을 때 발생
click	**onclick**	해당 영역에서 마우스를 클릭 했을 때 발생
keydown	**onkeydown**	해당 영역에서 키보드를 눌렀을 때 발생
keyup	**onkeyup**	해당 영역에서 누르고 있던 키보드를 떼었을 때 발생
keypress	**onkeypress**	해당 영역에서 키보드를 계속 누르고 있을 때 발생
submit	**onsubmit**	폼의 내용을 전송 할 때 발생
reset	**onreset**	폼의 내용을 초기화 시킬 때

JavaScript와 객체

- 자바스크립트는 객체지향언어(Object Oriented Language)
  - Java Script가 지원하는 객체만 사용할 수 있었으나 1.2 버전부터는 개발자가 직접 객체를 만들 수 있다.
- 객체의 종류
  - 내장 객체(Built-In 객체) : 웹 브라우저에서 제공하는 객체
    - **자바스크립트 내장객체** : JavaScript 코드 작성시 유용한 기능을 제공해 주는 객체들
      - Array, String, Date, Math 등
    - **DOM 객체** (브라우저 내장객체) : 웹 브라우저의 각 요소들을 객체로 제공
  - 사용자 정의 객체 : 사용자가 만드는 객체

Array

- 배열 객체
  - 생성 구문
    - 변수 = [value, value, value…]
  - 값 접근
    - 변수[index]
  - 속성
    - length : 배열의 크기

```
let arr = [10, 20, 30, 40, 50];

for(let idx = 0; idx<arr.length; idx++){
    alert(arr[idx]);
}
```

Array

- 주요 메소드
  - push(value[,value,...])
    - 배열에 값을 추가
  - pop()
    - 배열의 마지막 index의 값을 return하고 삭제한다.
  - shift()
    - 배열의 첫번째 index의 값을 return하고 삭제한다.
  - splice(start_idx [,length])
    - 배열의 일부값들을 삭제. 시작 idx와 삭제할 개수(개수 생략하면 나머지 다 삭제)
  - slice(start_idx[, end_idx]);
    - 배열의 일부 값들을 조회하여 새로운 배열로 return 한다.(시작 idx와 끝 idx. 끝 idx는 포함 안됨)
  - concat(배열객체[,배열객체,..])
    - 배열을 합쳐 새로운 배열을 생성
  - join([구분자])
    - 배열의 element들을 구분자로 이어진 문자열로 만들어 return. 기본구분자는 ' , ' 임.

사용자 정의 객체

- 자바스크립트에서 객체 구현은 파이썬의 dictionary와 비슷하다.
- 객체 리터럴 구문

변수 = {
property명 : value
[, properyt명: value]
}
	- value에 함수객체가 할당 되면 메소드가 된다.
	- property 명은 변수로 표현하거나 문자열로 표현할 수 있다.

```
let person1 = {
    name: "홍길동",
    age: 20,
    address: "서울",
    go: function(place) {
        console.log(place+"에 간다");
    }
}
```

```
let person2 = {
    "name": "이순신",
    "age": 25,
    "address": "부산",
    "go": function(place) {
        console.log(place+"에 간다");
    }
}
```

사용자 정의 객체

- 객체 property 호출
  - dot 표기법과 [ ] 표기법을 제공한다.
  - 변수
    - 객체.property, 객체["property"]
      - console.log(person1.name)
      - console.log(person1["name"])
      - person1.age = 20
      - person1["age"] = 30
  - 메소드
    - 객체.메소드명([args,...]), 객체[“메소드명“]([args, ...])
      - person1.go("학교")
      - person1["go"]('학교')

배열, 객체 구조분해 할당

- 배열의 원소들을 각각 다른 변수에 대입

[변수1, 변수2, ...] = 배열 [변수1, 변수2=value, ..] = 배열 변수에 기본값을 대입하면 대입할 원소가 없으면 기본값이 대입된다.	arr1 = [1, 2, 3] [v1, v2, v3=-1] = arr1//v1=1, v2=2, v3=3 arr2 = [1,2] [v1, v2, v3=0] = arr2 //v3은 기본값 0이 대입된다. arr3 = [1,2,3,4,5,6] [v1, v2**, ...**v3] = arr3 //v1=1, v2=2, v3=[4,5,6]

- 객체의 property 들을 각각 다른 변수에 대입

var { key1, key2, key3 = 기본값 } = 객체 객체의 key를 변수로 선언한다. var {key1: newKey, key2} = 객체 key1의 값을 변수 newKey에 대입	obj = {name:'홍길동', age:20} {name, age}= obj //name=>홍길동, age=> 20이 대입 var {name:new_name, age} = obj //new_name=>홍길동, age=20

DOM(Document Object Model) Tree 구조

- 문서의 구성요소들(elements들)을 웹브라우저가 객체화 해서 관리하는 방식
  - 문서 구성요소를 객체화 한 것을 DOM이라 한다.
  - DOM 객체들을 Tree 구조로 계층화 해서 관리
- 동적으로 문서의 내용, 구조, 스타일에 접근하여 변경, 생성
- 구성
  - 노드 (Node) : Tree 구조를 구성하는 요소
    - Element(요소) 노드 : 태그
    - Text 노드
    - Attribute(속성) 노드 : 태그 내 속성
- Web browser 가 HTML **문서를 로딩 시점 생성** 및 구성 한다.

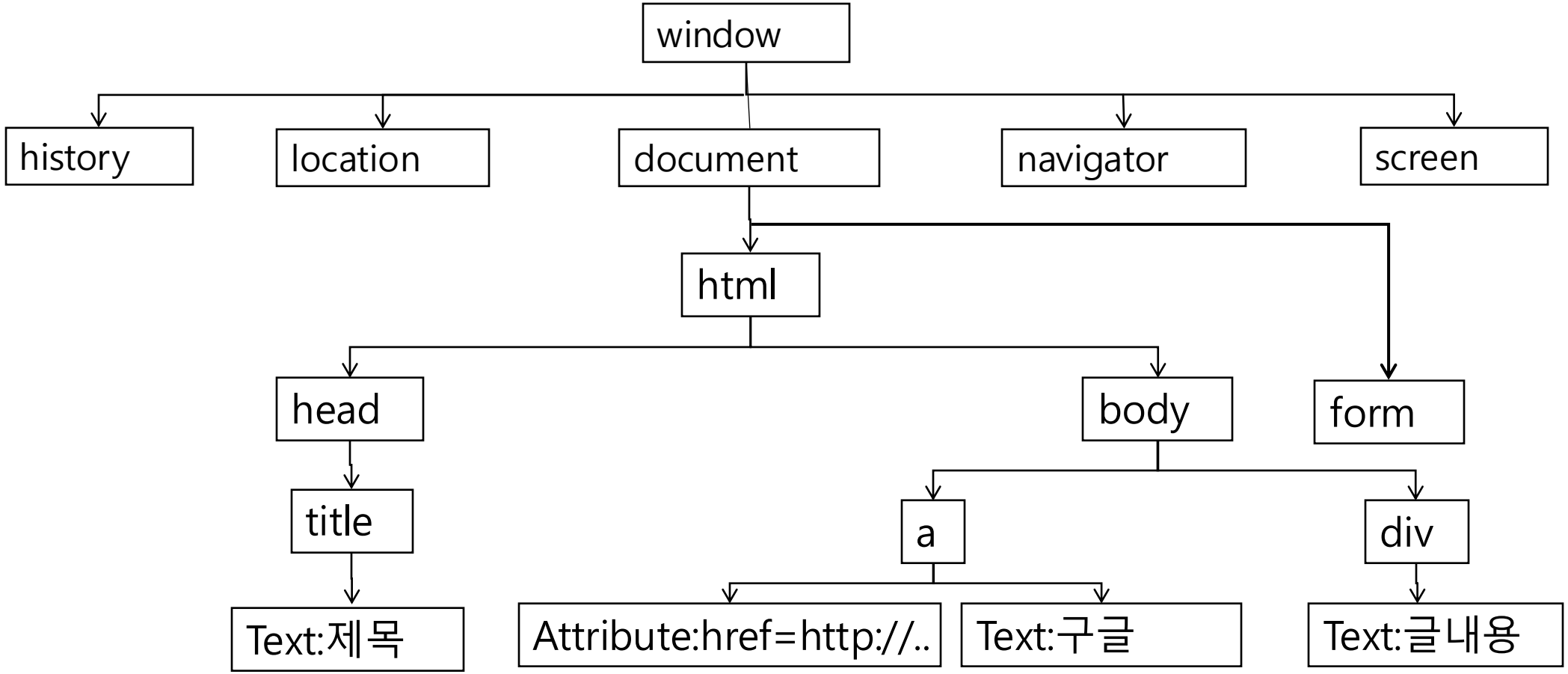
DOM(Document Object Model)

- HTML 문서 구조

```
<html>
   <head>
        <title>제목</title>
   </head>
   <body>
        <a href="http://google.com">구글</a>
        <div>글내용</div>
        <form> .. </form>
   <body>
</html>
```

DOM(Document Object Model)

- DOM Tree 구조



DOM 객체 (브라우저 내장객체)

- window 객체
  - 브라우저 내장객체 중 최상위 Node에 위치한 객체
    - 자바스크립트 동작에 필요한 전역 객체로 동작
    - 사용자가 보는 브라우저 창을 나타낸다.
    - 브라우저 창의 HTML 문서에 접근하는 시작객체로 사용된다.
  - 접근방법
    - window.속성, window.메소드
    - window 는 생략 가능
  - 주요 메소드

메소드	설 명
open()	새로운 윈도우(창) 열기. 열린 창에서 연 창을 **opener** 통해 접근할 수 있다.
close()	열려있는 브라우저(윈도우) 닫기
alert()	간단한 메시지를 보여주기 위한 dialog box
confirm()	사용자로부터 어떠한 작업을 확인 받기 위한 dialog box
parseInt(String) **parseFloat(String)**	인수의 String을 정수, 실수 형태로 변환
isNaN(String)	인수의 String이 숫자 형태가 아니면 true, 숫자형태이면 false리턴
eval(String)	String을 JavaScript 코드로 변환 해서 실행

DOM 객체 (브라우저 내장객체)

- document객체
  - 브라우저에 보여지는 HTML 문서를 가리키는 객체
    - 문서의 구성요소에 접근
    - 문서의 내용을 변경
  - window의 하위 객체
  - 접근 방법
    - window.document 또는 document
  - 주요 메소드

메소드	설명
write("문자열")	문자열을 문서에 출력한다.
DOM 객체 접근 및 생성 메소드	

DOM 객체 (브라우저 내장객체)를 통한 Form 처리

- form 객체
  - <form> 객체
  - document의 하위 계층 객체
  - 접근 방법
    - window.document.form의 name속성값
  - 주요 속성

속성	설명
action	<form>의 action 속성
method	<form>의 method 속성
enctype	<form>의 encoding 속성
name	<form>의 name 속성. form 객체접근시사용

  - 주요 메소드

메소드	설명
reset()	입력양식 초기화. reset버튼 의 동작 처리
submit()	입력된 값을 action의 url로 전송. submit 버튼의 동작 처리

DOM 객체 (브라우저 내장객체)를 통한 Form 처리

- text, password객체
  - <input type="text">, <input type="password"> 객체
  - form의 하위 객체
  - 접근 방법
    - window.document.form_name.태그의name속성값
  - 주요 속성

속성	설명
name	태그의 name 속성
value	태그의 value 속성
기타 input type='text||password' 가지는 속성들	

  - 주요 메소드

메소드	설명
focus()	객체에 포커스를 준다.

DOM 객체 (브라우저 내장객체)를 통한 Form 처리

- textarea객체
  - <textarea> 객체
  - form의 하위 객체
  - 접근 방법
    - window.document.form_name.textarea_name 속성값
  - 주요 속성

속성	설명
name	태그의 name 속성
value	textarea에 입력된 값(value)
cols	열 속성값 – 한줄에 입력될 수 있는 최대 글자수
rows	행 속성값 – 보이는 총 행수

  - 주요 메소드

메소드	설명
focus()	객체에 포커스를 준다.

DOM 객체 (브라우저 내장객체)를 통한 Form 처리

- checkbox 객체
  - <input type="checkbox"> 객체
  - form의 하위 객체
  - 접근 방법
    - window.document.form_name.checkbox_name 속성값
  - 주요 속성

속성	설명
name	checkbox name 속성
value	checkbox value 속성
checked	checkbox checked 속성으로 선택상태. true/false

  - 주요 메소드

메소드	설명
click()	체크박스 클릭 처리

DOM 객체 (브라우저 내장객체)를 통한 Form 처리

- radio 객체
  - <input type="radio"> 객체
  - form의 하위 객체
  - 접근 방법
    - window.document.form_name.radio_name속성값
  - 주요 속성

속성	설명
name	radio name 속성
value	radio value 속성
checked	radio checked 속성으로 선택상태. true/false

  - 주요 메소드

메소드	설명
click()	라디오버튼 클릭 처리

DOM 객체 (브라우저 내장객체)를 통한 Form 처리

- select 객체
  - <select> 객체
  - form의 하위 객체
  - select 태그의 value(사용자가 선택하는 값)은 <option> 태그에 있다.
    값에 접근하기 위해서는 select를 통해 option 객체로 접근한다.
  - 접근 방법
    - window.document.form_name.select_name 속성값
      선택 Item(option) 접근 : window.document.form.select_name.**options - 배열**
  - 주요 속성

속성	설명
name	<select> name 속성
selectedIndex	선택된 item(option)의 index. (option의 index는 0부터 시작)
length	option의 총 개수
options	select내의 option객체들을 가진 배열(Node List)

DOM 객체 (브라우저 내장객체)를 통한 Form 처리

- option 객체
  - select 내의 <option> 태그 객체
  - select 객체의 options 속성을 통해 배열로 받아 접근한다.
  - 주요 속성

속성	설명
text	option의 내용
value	value 속성의 값 (value속성이 없으면 내용이 value의 역할을 한다.)
selected	option의 선택 여부. true/false

DOM 조회 및 동적 변경

- JavaScript에서 DOM Node객체를 조회할 수 있다.
- JavaScript에서 동적으로 DOM Node를 생성, 삭제, 추가할 수 있다.
- DOM Tree 구조에 Node 가 추가 되거나 삭제 되면 그것이 화면에 바로 적용 된다.
  - 동적으로 화면 구성 요소를 바꿀 수 있다.

DOM Node 조회 - document 객체의 메소드를 이용한 조회

- getElementById("태그ID속성명") : Node
  - tag의 ID 속성으로 조회

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


var divLayer = document.getElementById("layer")
```

- getElementsByTagName("태그이름") : Node []
  - 태그 이름으로 조회
  - Node List (배열) 이 리턴된다.

```
<a href=".."> ... </a>
<a href=".."> ... </a>
<a href=".."> ... </a>

var aList = document.getElementsByTagName("a");  // => 페이지 내의 모든 A 객체를 배열로 리턴
```

DOM Node 조회 - document 객체의 메소드를 이용한 조회

- getElementsByName("입력태그의 name속성값") : Node []
  - 입력 태그의 name 속성으로 조회
  - Node List (배열) 이 리턴된다.

```
<input type="checkbox" name="hobby" value="...">
<input type="checkbox" name="hobby" value="...">
<input type="checkbox" name="hobby" value="...">

var chkList= document.getElementsByName("hobby");
```

- getElementsByClassName("class속성 이름") : Node []
  - 태그의 class 속성 값으로 조회
  - Node List (배열) 이 리턴된다.

```
<span class="cls"></span>
<span class="cls"></span>
<span class="cls"></span>

var clsList= document.getElementsByClassName("cls");
```

DOM Node 조회 - document 객체의 메소드를 이용한 조회

- querySelector('selector') : Node
  - selector를 만족하는 **첫번째** element 객체를 반환

```
<div id='num1'>..</div>
<div id='num2'>..</div>
<div id='num3'>..</div>

var node = document.querySelector('div#num1')
```

- querySelectorAll ('selector') : Node list
  - selector를 만족하는 **모든** element 객체를 반환

```
<span class="cls"></span>
<span class="cls"></span>
<span class="cls"></span>

var nodeList = document.querySelectorAll('span.cls')
```

DOM Node 조회 – DOM Element 객체의 속성 이용

- DOM Element 객체를 기준으로 조회한다.
  - 모든 DOM 객체는 다음 속성을 제공하며 자식을 기준으로 찾는다.
- parentNode
  - 부모 노드 return
- firstChild/lastChild
  - 첫 번째/ 마지막 자식 노드 return
- children/childNodes
  - 자식노드를 Node 리스트(배열)로 return
  - **Children** : Element 노드들만 리턴, **ChildNodes** : 모든 자식 노드(Text/Element..)들 리턴
  - 예
    - 모든 자식 노드의 개수 조회 : 부모노드.children.length
    - 두 번째 자식 노드 : 부모 노드. children[1]

DOM Node 조회 – DOM Element 객체의 속성 이용

- previousSibling, nextSibling
  - 앞/뒤 위치한 노드 return
  - 모든 타입의 노드 다 리턴 (text노드, elements 노드 등)
- previousElementSibling, nextElementSibling
  - 앞/뒤 위치한 element 노드(태그) return

DOM Tree 노드 구성 변경 – Node 생성

- **document.createElement**(태그명)
  - Element 노드 생성 => 태그 생성 개념
  - var x = document.createElement('p'); // <p> 태그 생성
- **document.createTextNode**("문자열")
  - 텍스트 노드 생성 => 태그 내에 Text 쓰는 개념
  - var txt = document.createTextNode("hello");

DOM Tree 노드 구성 변경 – DOM Tree의 Node 변경

- DOM Element 객체의 메소드를 이용해 Node를 추가/수정/삭제 한다.
- appendChild() : 마지막 자식노드 추가
  - 하위 노드 추가. 마지막 자식 노드로 추가한다.
  - 부모노드.**appendChild**(추가할 자식노드)
- insertBefore() : 자식 노드 삽입
  - 하위 노드들 사이에 Node를 삽입한다.
  - 부모노드.**insertBefore**(삽입할 노드, 기존 자식노드)
- hasChildNodes() : 자식노드 유무 조회
  - true/false 리턴
- removeChild() : 자식노드 삭제
  - 자식 노드를 제거한다.
  - 부모노드.removeChild(삭제할 자식노드)

DOM Tree 노드 구성 변경 – DOM Tree의 Node 변경

- replaceChild()
  - 자식 노드를 다른 노드로 교체 한다.
  - 부모 노드.replaceChild(새 노드, 교체할 자식노드)
- innerHTML, innerText 속성 (메소드 아님)
  - Node객체 하위의 DOM을 만들때 사용한 HTML구문을 가지고 있는 속성
    - innerText는 Text 노드만 가지고 있다.
    - innerHTML은 element 노드, text 노드 모두 가지고 있다.
  - 구문이 바뀌면 DOM 구조도 변경된다.

```
<div id="layer"> <b>hello</b> </div>
```

```
var txt = document.getElementById("layer").innerHTML; //조회

document.getElementById("layer").innerHTML = "<i>안녕</i>"; //변경
```

DOM Tree 노드 구성 변경 – 속성 변경

- 특정 Node(노드)의 속성값 변경
  - Node객체.setAttriute(“속성명”, “속성값”)
  - Node객체.속성 = “값”;
- 특정 Node(노드) 속성값 조회
  - Node객체.getAttribute(“속성명”) : 속성값
  - Node객체.속성

fetch() 를 이용한 network 요청

- fetch()는 네트워크 요청(Network Request)을 보내고 응답(Response)을 받기 위한 JavaScript 함수
  - 기존의 XMLHttpRequest보다 더 간결하고 Promise 기반으로 설계되어 비동기 코드 흐름과 잘 맞는다.
- 호출
  - fetch(url, options): Promise<Response>
    - url (문자열) - 요청을 보낼 주소
    - options (객체) - 요청과 관련된 다양한 설정들. 요청방식, header, body 등
  - 반환값: Promise<Response>
    - 요청을 보내면 Promise가 바로 반환된다. 서버에서 응답이 오면 Promise가 Response객체로 이행된다.
    - Promise: 자바스크립트의 비동기 처리 제어를 위한 표준으로 비동기 작업의 미래 완료 결과(성공 또는 실패)를 나타내는 객체이다.

fetch() 를 이용한 network 요청

- Response
  - fetch()의 최종 반환 값으로 서버 응답 정보를 담아 제공한다.
  - 주요 Property
    - ok (bool) - 정상 응답인지 오류응답지. (200번 범위면 true 반환)
    - status (number): HTTP 상태 코드(200, 404, 405)
    - headers (headers object): 응답 Header 객체
    - body: (ReadableStream): 응답 본문(body) 응답 본문 데이터 스트림
  - Body 내용 읽기 메소드
    - 응답 본문(body)을 다양한 형식으로 변환해서 읽는 함수를 제공. Stream을 이용해 읽기 때문에 한번 읽으면 다시 읽을 수 없다.
    - Response.json(): Promise<object>: JSON 텍스트를 파싱해서 JS객체로 반환
    - Response.text(): Promise<string>: 본문의 내용을 문자열로 그대로 반환
    - Response.blob(): Promise<Blob>: 이미지/바이너리 파일
    - Response.formData(): Promise<FormData>: 폼데이터 파싱