데이터 시각화(ECMA 6 기술)

Data visualization

ECMAScript

코딩 환경

- HTML 에디터가 있다면, 에디터 사용하면 편리
- 전반적으로 HTML에 대한 기본 지식이 있다는 가정
  (중간 자주 쓰는 태그에 대해서는 추가 설명 예정)
- HTML 파일과 script 파일을 따로 생성

```
<!DOCTYPE html>
<html lang=ko>
   <head>
      <meta charset="utf-8">
      <title>타이틀</title>
      <script src="0_js.js"></script>
   </head>
   <body>
   </body>
</html>
```

```
"use strict";
debugger;
```

- use strict
  > 일부 Object는 strict 모드에서 실행이 기본
- Debugger
  > 소스 코드를 라인 단위로 디버깅 가능

자바스크립트 기초

브라우저에 이미 떠 있는 페이지의 DOM을 조작해서 페이지를 동적으로 만들 수 있는 스크립트 언어

- 특 징
  - 브라우저에 의해 실행되는 클라이언트 사이드 스크립트 언어
  - 명령어를 사용하는(imperative) 절차적(procedural) 언어로 구조적 프로그래밍(structured programming) 가능
  - 객체(object)의 정의와 사용을 지원하는 객체 기반(object-based) 언어
  - 실행 시간에 자료형을 검사하는 동적 타이핑(dynamic typing) 언어
  - 동적인 웹 페이지와 향상된 사용자 인터페이스를 쉽게 구현 가능

자바스크립트 기초

- **기본 구조**

```
형식1 : <script type = "text/javascript">
            JavaScript_코드
        </script>
```

```
형식2 : <script language = "JavaScript">
            JavaScript_코드
        </script>
```

```
형식3 : <script type = "text/javascript" src = "JavaScript 파일의 url">
            JavaScript_코드
        </script>
```

```
형식4 : <script language = "JavaScript" src = "JavaScript 파일의 url">
            JavaScript_코드
        </script>
```

자바스크립트 기초

JavaScript Keywords

break	else	instanceof	true	case	false	new	try
catch	finally	null	typeof	continue		for	return
var	default	function	switch	void	delete	if	this
while	do	in	throw	with			

JavaScript reservation Keywords

abstract	enum	int	short	Boolean	export
interface	static	byte	extends	long	super
char	final	native	synchronized	class	float
package	throws	const	goto	private	transient
dubugger	implements	protected	volatile	double	
import	public				

문장 끝 세미콜론(;)이 없어도 행을 바꾸면 문장으로 취급하기 때문에 실행에 문제는 없음.
하지만 대부분 프로그래밍 언어에서 세미콜론을 입력하므로 관례상 입력

자바스크립트 기초

대소문자 구분

- 자바스크립트는 HTML과 다르게 대소문자를 구분

ex)

함수 : getElementById ≠ getElementbyID

변수 : myVariable ≠ MyVariable

여백 무시

- 자바스크립트는 여백을 무시

ex)

```
var person="Hege";
var person = "Hege";
```

코드 라인 변경 방법

- 백슬러시를 이용하여 문자열 코드라인을 변경 하여 작성 할 수 있음

ex)

```
document.write("Hello World");
document.write \
      ("Hello World");
```

자바스크립트 기초

주석은 자바 스크립트가 실행되지 못하게 하거나 소스 코드 설명 시 사용

싱글 라인 주석

```
// Write to a heading:
document.getElementById("myH1").innerHTML="Welcome";
```

```
//document.getElementById("myH1").innerHTML="Welcome";
```

```
var x=5;    // declare x and assign 5 to it
```

멀티 라인 주석

```
/*
The code below will write to a heading and to a paragraph,
and will represent the start of my homepage:
*/
```

자바스크립트 기초

String, Number, Boolean, Array, Object, Null, Undefined.

Number

```
5
6
5+6
3.14;
```

String

```
"Hello"
" JS"
"Hello JS"
"a"
```

Boolean

```
true;
false;
```

Array

```
[1,2,3]
["a","b","c"]
[1,"a",2,"b"]
```

Object

```
{
   f_name:"john",
   l_name:"doe",
   id     :5566
}
```

Undefined and Null

```
Undefined
: no value
Null
: emptied
```

자바스크립트 기초

자바스크립트 변수들은 정보를 저장하기 위한 “containers“

수학과 마찬가지로 자바스크립트 변수는 변수 값(x=5) 또는 식(z=x+y)를 표현 할 수 있음

변수 값은 짧은 알파벳 또는 설명하는 단어로 가능

- 변수 이름은 문자로 시작해야 함 (name, age) – O, (1name) – X
- 문자 외에 시작 가능한 것으로는 $와 _가 있음 (하지만 사용하지 x)
- 대소문자를 구분함 (y 와 Y는 다른 변수)

```
var x=5;
var y=6;
var z=x+y;
var textVar=”hi”
```

자바스크립트 기초

Local JavaScript 변수

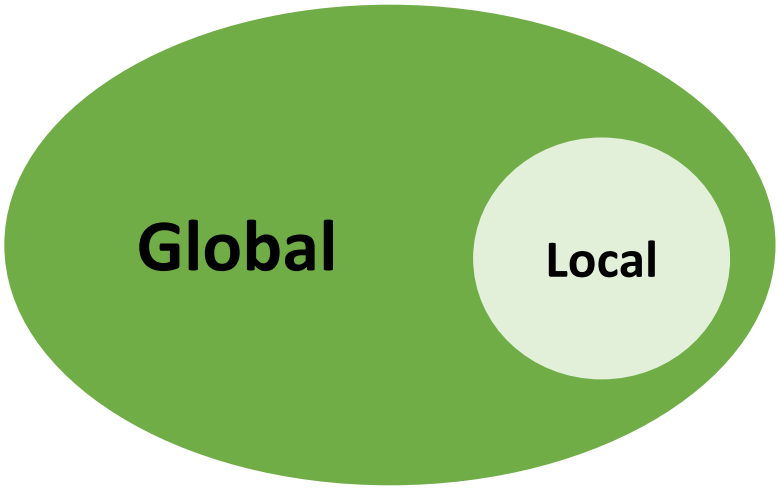
JavaScript 함수 내에서 선언 된 변수로 함수 내에서 활용 가능

함수 내에서 선언 되므로 각 함수마다 같은 이름을 가진 변수들을 선언 할 수 있음

Local 변수는 함수가 끝남과 동시에 삭제됨

Global JavaScript Variables

모든 스크립트 함수에 공유하는 변수




```
function f1(){
        var a = 1;
        alert(a);
}
function f2(){
        var a = 2;
        alert(a);
}
```

```
var a = 1;
function f1(){
        alert(a);
}
function f2(){
        alert(a);
}
```

ECMAScript

var, let

- var 키워드 : 변수 선언
  > 로컬 변수(Local Variable) / 글로벌 변수(Global Variable) 구분

- let 키워드 : 변수 선언
  > var 키워드의 문제점을 해결하기 위한 것

  무엇을??? 일단, 총 6가지 특징이 있음

  1. 함수 안에 작성한 let 변수는 함수가 스코프
  2. 함수 안에 if(a=b) { let sports = "축구"} 형태로 코드를 작성했을 때, sports 변수는 함수가 스코프가 아니라 if 문 블록이 스코프
  3. 블록 밖에 같은 이름 변수가 있어도 스코프가 다르므로 변수 각 각에 값을 설정할 수 있음, 변수 값이 유지
  4. 블록 안에 블록을 계층적으로 작성하면 각 각 블록이 스코프
  5. 같은 스코프에서 같은 이름의 let 변수를 선언할 수 없음
  6. let 변수는 호이스팅 되지 않음

ECMAScript

var, let

- var

```
//"use strict";
one = 100;
function get(){
   one = 300;
   console.log("함수:", one);
}
get();
console.log("글로벌:", one);
```

- let

```
'use strict';
let book;
let sports = '축구';
sports = '농구';

let one = 1, two = 2, three;
//let four = 4, let five =5;          //error
//let six = 6, var seven = 7;          //error
```

- 콤마(,) 를 통해 연속으로 선언 가능
  var a, b, c, d; let a, b, c, d;

- 둘은 혼용해서 쓸 수 없음
  ex) let a, var t;

블록 스코프

- let 변수를 선언하는 가장 큰 목적은 스코프
- 그 중 블록 스코프에서 상당히 돋보일 수 있음

```
let sports = '축구';
if (sports){
    let sports = '농구';
    console.log("블록:" , sports);
}
console.log("글로벌:" , sports);
```

ECMAScript

var 및 let을 사용했을 때 this 차이점

- this 의 경우 현재 스코프 상태를 판단해 해당 변수의 값을 보여주는 역할을 함

```
var music = "음악";
console.log(this.music);

let sports = "축구";
console.log(this.sports);
```

- this.music 의 경우 “음악” 이라는 결과가 나옴
- this.sports는 undefined 결과가 나옴

왜?

자바스크립트 함수

호출을 당할 때 실행 되는 코드 블록 단위 모음

호출 되는 시기 :

- 이벤트가 발생 할 때
- 자바스크립트 코드에서 호출 할 경우
- 자동으로 호출 되는 경우

JavaScript 기본 함수 구문

```
function functionName(){
  some code to be executed
}
```

JavaScript 인수를 사용한 함수 구문

```
function functionName(var1, var2){
  some code to be executed
}
```

자바스크립트 함수

JavaScript 반환 값을 가지는 함수

값을 반환하여 사용하고자 할 때 사용

```
function myFunction(){
        var x = 5;
        return x;
}
```

```
var myVar = myFunction();
```

자바스크립트 함수

익명 함수 : function(…){ … };

```
var function1 = function(){
        alert("function1");
};
```

function 뒤에 이름이 없고 변수에 넣어서 사용 하는 것이 특징

선언적 함수 : function 함수이름(…){ … };

```
function function1(){
        alert("function1");
}
```

function 뒤에 이름을 작성하여 사용 하는 것이 특징

이 둘의 차이점은??

```
function1(); (X)

var function1 = function(){
        alert("function1");
};

function1();
```

```
function1();

function function1(){
        alert("function1");
}

function1();
```

ECMAScript

function

- ECMAScript 는 함수도 스코프를 가지므로 하나의 블록 스코프

```
let sports = "축구", music = "재즈";
function get(){
    let music = "클래식";
    console.log(music);
    console.log(sports);
}
get();
```

ECMAScript

function

- 아래 소스 코드에 대한 결과는???

```
"use strict";
var sports = "축구";
let music = "재즈";

function get(){
    var sports = "농구";
    let music = "클래식";
    console.log("1: ", sports);
    console.log("2: ", this.sports);
    console.log("3: ", this.music);
}

window.get();
get();
```

ECMAScript

try-catch / switch-case

- try catch 문에서도 블록 스코프를 갖음
- Switch –case 문의 경우 case는 별도 스코프를 갖지 않으며 switch 스코프에 속함

```
let sports = "축구";
try{
    let sports = "농구";
    console.log(sports);
} catch (e){

};
console.log(sports);
```

```
var count = 1;
let sports = '축구';
switch (count){
    case 1:
        let sports = '농구';
        console.log(sports;
};
```

ECMAScript

호이스팅

- 스크립트는 소스 코드를 위에서 아래로 순차적으로 실행
- 따라서 호출될 함수를 작성한 후, 아래에서 함수를 호출해야함
- 하지만 함수 선언문은 함수를 호출하는 코드를 위에 작성하고 호출될 함수를 아래 작성해도 함수가 호출
- var 변수의 경우 사용하는 코드를 위에 작성하고 아래 변수를 선언해도 에러가 나지는 않음, 하지만 undefined (함수도 마찬가지)
- 하지만, let 변수는 호이스팅이 되지 않음

```
console.log(sports);
var sports = "스포츠";

console.log(music);
let music = "음악";
```

ECMAScript

for

- for 문도 var 와 let 변수로 작성하는 것에 대해 차이가 있음
- 1
- let 변수의 경우 반복할 때마다 스코프를 갖는 반면, var 변수는 스코프를 갖지 않음

```
var nodes = document.querySelector("ul");
for (var k=0; k<nodes.children.length; k++){
   var el = nodes1.children[k];
   el.onclick = function(event){
      event.target.style.backgroundColor = "yellow";
      console.log(k);
   }
}
```

```
var nodes = document.querySelector("ul");
for (let k=0; k<nodes.children.length; k++){
   var el = nodes.children[k];
   el.onclick = function(event){
      event.target.style.backgroundColor = "yellow";
      console.log(k);
   }
}
```

ECMAScript

const

- const 키워드는 변수에 할당된 값을 변경할 수 없음
- “상수”

```
const SPORTS = "축구";
try{
   SPORTS = "농구";
} catch (e){
   console.log("const 재할당 불가");
}
```

const

- 하지만, obj 프로퍼티에는 값을 할당 할 수 있음

```
const obj = {language: "한글"};
try{
    obj = {};
} catch (e){
    console.log("const 재할당 불가");
}
obj.language = "영어";
console.log(obj.language);
```

자바스크립트 객체

자바스크립트 객체는 “Everything”
a String, a Number, an Array, a Date....
속성과 메소드(properties and methods).

Properties and Methods

Properties are values associated with an object.
Methods are actions that can be performed on objects.

Properties:

car.name=Fiat

car.model=500

car.weight=850kg

car.color=white

Methods:

car.start()

car.drive()

car.brake()

자바스크립트 객체

자바스크립트 객체 선언 (key : value)

```
first name : John
last  name : Doe
age        : 50
eyecolor   : blue
```

```
<!DOCTYPE html>
<html>
<body>
<script>
        var person=new Object();
        person.firstname="John";
        person.lastname="Doe";
        person.age=50;
        person.eyecolor="blue";
        document.write(person.firstname + " is " + person.age + " years old.");
</script>
</body>
</html>
```