Day 04 (2019-06-26)
1. Javascript Variable Scope

(1) Global scope

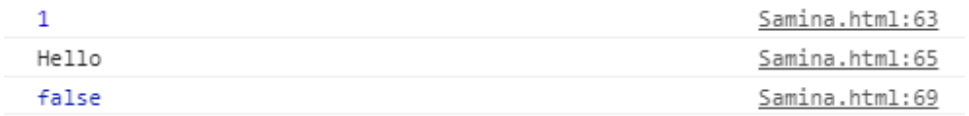
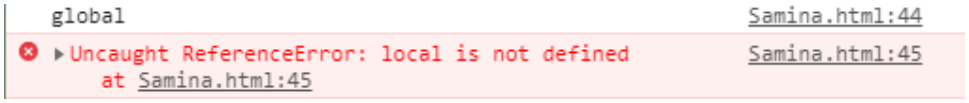
스크립트 전역에서 참조되는 영역.

(2) Local scope

정의된 함수 안에서만 참조되는 영역.

특징

항목	설명	예제	실행 결과
함수단위의 Scope	블록 단위의 Scope를 채택하는 일반적인 언어와 달리 Javascript는 함수단위의 Scope임.	`function varScope() { if (true) { var b = 0; for (var c = 0; c < 5; c++) { console.log ("c=" + c); } console.log ("c=" + c); } console.log("b=" + b); } varScope();`	c=0 Samina.html:53 c=1 Samina.html:53 c=2 Samina.html:53 c=3 Samina.html:53 c=4 Samina.html:53 c=5 Samina.html:55 b=0 Samina.html:57

변수명중복가능	변수명이 중복 선언되어도 에러가 나지 않음. 가장 가까운 범위의 변수를 참조.	`var duplicate = 1;` `console.log(duplicate);` `var duplicate = 'Hello';` `console.log(duplicate);` `(function(){` `duplicate = false;` `})();` `console.log(duplicate);`	
var 키워드 생략가능	var 키워드를 생략할 경우 전역 Scope로 취급됨	`<script>` `var global = 'global';` `function func(){` `var local = 'local';` `}` `console.log(global);` `//global` `console.log(local);` `//error` `</script>`	

렉시컬	Scope는 함수의 실행 시점이 아닌 선언 시점에 생긴다는 특징. → 변수명 중복 가능 항목에서 이름이 동일한 여러 변수가 있을 경우 가장 가까운 범위의 변수를 참조하나, **'선언시'**에 가장 가까운 변수를 찾음	<script> var name = 'zero'; function log() { console.log(name); } function wrapper() { var name = 'nero'; log(); } wrapper(); </script>	zero Samina.html:76

호이스팅	자바스크립트는 코드를 인터프리팅 하기 전에 먼저 컴파일을 수행한다. var var1 = 'javascript';의 구분은 아래와 같이 두 개의 구문으로 나누어 처리됨. var var1; var1 = 'javascript';	``` var var1 = 'Javascript'; if(1 !== 1){ var1 = 'Java'; } console.log ('Hello ', var1); console.log ('Hello hoisting :', hoisting); var hoisting = ''; try { console.log('Hello hoisting in function:', func_hoisting); (function () { var func_hoisting = ''; })(); } catch(e) { console.error(e); } hoisted_func(); function hoisted_func() { console.log("Hello i'm hoisted func"); } ```	Hello Javascript Samina.html:17 Hello hoisting : undefined Samina.html:20 ▶ ReferenceError: func_hoisting is not defined Samina.html:29 at Samina.html:24 Hello i'm hoisted func Samina.html:34 global Samina.html:44

	즉 변수의 선언 단계와 초기화 단계를 구분하며, 선언 단계에서는 그 선언이 소스 코드의 어떤 곳에 위치하던 해당 스코프의 컴파일 단계에서 처리한다.		

Use stric

오래 된 Javascript 에서 묵인되어 안전하지 않은 동작들을 사전에 잡을 수 있도록 하는 가이드라인

Javascript 엔진의 최적화 작업을 어렵게 하는 실수들을 바로 잡음

대표적 특징 (상세 참조)

글로벌 변수 생성 방지	`"use strict";` `//   mistypedVariable` `mistypedVaraible = 17; //     ReferenceError`
확장 불가 객체에 새로운 프로퍼티 할당을 에러로 처리	`"use strict";` `var fixed = {};` `Object.preventExtensions(fixed);` `fixed.newProp = "ohai"; // TypeError`
읽기전용 프로퍼티에 값 대입을 불가 처리	`var obj2 = { get x() { return 17; } };` `obj2.x = 5; // TypeError`
사용할 수 없는 프로퍼티에 할당	`//` `var undefined = 5; // TypeError` `var Infinity = 5; // TypeError` `//` `var obj1 = {};` `Object.defineProperty(obj1, "x", { value: 42, writable: false });` `obj1.x = 9; // TypeError`

2. ES6 (ES2015)

let, const 키워드 추가

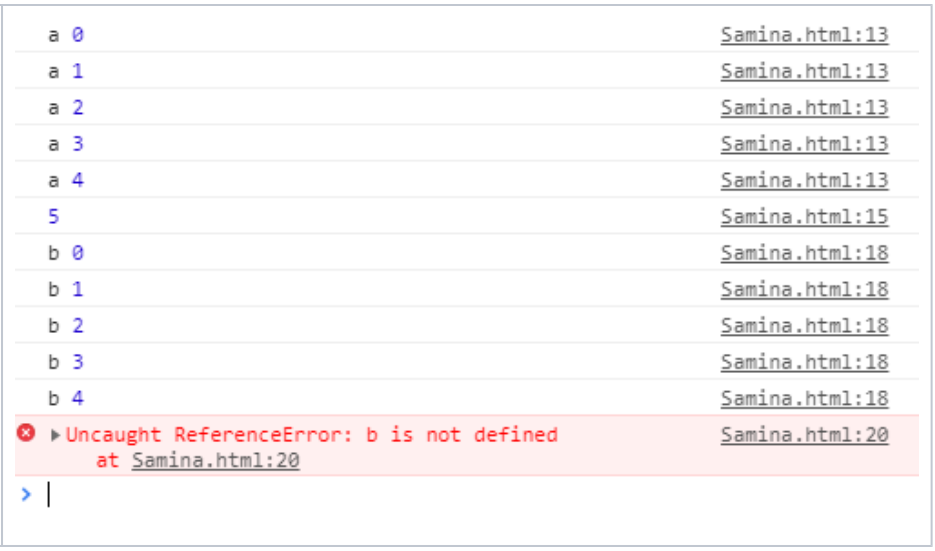
var를 이용할 경우 hoisting, 전역 변수, function scope 등의 특징 때문에 작성한 코드가 우아할 수 없었고 다른 Third party library를 쉽게 가져다 쓸 수 없었다.
→ 전역 변수가 중복 사용되어 오동작을 일으킬 수 있는 가능성이 큼
→ 다양한 디자인 패턴으로 커버 해왔음 (노출식 모듈 패턴 등)

상수를 위한 키워드가 없어 CONST_NUMBER와 같이 대문자를 이용하고 상수로 취급하는 **'약속'**을 통해 처리 해왔음.

ES6에선 블록 스코프를 가지고 변수의 중복선언이 불가능한 let과 상수를 위한 const 키워드가 추가 되었다.

단 호이스팅은 발생한다. 만약 선언부 이전에 해당 변수를 사용할 경우 undefined로 취급되는 var와 달리 Uncaught ReferenceError가 발생한다
→ let이나 const로 생성된 변수는 TDZ(Temporal Dead Zone)에 들어가기 때문이다.
→ TDZ란 선언이 되어있지만 아직 초기화가 되지 않아 메모리가 할당되지 않은 상태이다.

```
    for(var a=0; a<5; a++){
        console.log('a', a);
    }
    console.log(a);

    for(let b=0; b<5; b++){
        console.log('b', b);
    }
    console.log(b);

    const PROP_A = 1;
    PROP_A = 2; //compile error
```



테스트

실행 예측

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title></title>
  <style type="text/css">
    div {
      border: solid 1px black;
      width: 100px;
      height: 100px;
    }
  </style>
</head>

<body>
  <div id="a1"></div>
  <div id="a2"></div>
  <div id="a3"></div>
  <div id="a4"></div>
  <div id="a5"></div>
  <hr />
  <div id="b1"></div>
  <div id="b2"></div>
  <div id="b3"></div>
  <div id="b4"></div>
  <div id="b5"></div>
</body>
<script>
  for (var a = 1; a <= 5; a++) {
     document.getElementById('a' + a).onclick = function (e) {
        alert('Hello event '+ a); ///???
     }
  }

  for (let b = 1; b <= 5; b++) {
     document.getElementById('b' + b).onclick = function (e) {
        alert('Hello event '+ b);  //???
     }
  }
</script>
</html>
</html>
```

Arrow문법

함수형 프로그래밍 패러다임의 지원을 위해 람다 지원

새로운 this 바인딩의 scope를 만들어 내지 않음 (자바스크립트 this 바인딩 참조)

```
const obj = {
  name: 'Lee',
  sayHi() { // === sayHi: function() {
    console.log(`Hi ${this.name}`);
  }
};
obj.sayHi(); // Hi Lee
```

```
const obj1 = {
  name : 'Lee',
  sayName : () => {console.log(`hi + ${this.name}`);}
};
obj1.sayName();
```

→ this가 객체가 아닌 전역에 바인딩 됨

```
var user = {
  name: 'lee',
  hello: function(){
      var self = this;
      setTimeout(function(){
        console.log("self -> "+ self.name);
        console.log("this -> "+ this.name);
      }, 1000)
    }
}
user.hello();
```

```
var user = {
  name: 'lee',
  hello: function(){
      setTimeout(() => {
          console.log("this -> "+ this.name);
      }, 1000)
    }
}
user.hello();
```

```
let target;
for(let i=0, length = this.users.length; i<
length; i++){
  if(this.users[i].id === targetId){
    target = this.users[i];
  }
)
```

filter, map, reduce 등의 함수형 프로그래밍을 지원하는 함수 사용시 유용

```
const target = this.users.filte(user => user.id  ===
targetId);
```

Template Strings

문자열 기반 템플릿을 위해 concat 또는 + 연산으로 선언하는 불편함...

```
var template = {
                'auditTrailWrap': ''.concat(
                        '<div class="audit-wrap">',
                        '<div class="list-title
bright-gray">',
                        '<div class="left">#{TITLE}<
/div>',
                        '<div class="right">',
                        '<a href="#" onclick="return
false;" class="_audit-trail-close">',
                        '<div><span class="sprite
icon-cancel"></span></div>',
                        '</a>',
                        '</div>',
                        '</div>',
                        '<div class="audit-list">',
                        '<table class="list-table01"
>',
                        '<colgroup>#{COL_HTML}<
/colgroup>',
                        '<thead><tr>#{HEADER_HTML}<
/tr></thead>',
                        '<tbody>#{AUDIT_TRAILS_HTML}<
/tbody>',
                        '</table>',
                        '</div>',
                        '</div>'
                ),
                'entryDataJson': '"#{NAME}":"#
{VALUE}"'
        };
```

→ 별도의 정규표현식을 통해 replace 작업이 필요함

```
 var contentHTML = helper.templateSubstitute
(template.auditTrailWrap, {
            'TITLE': messages['AUDIT_TRAIL.TITLE'],
            'COL_HTML': colHTMLPieces.join(''),
            'HEADER_HTML': headerHTMLPieces.join(''),
            'AUDIT_TRAILS_HTML': bodyHTML
        });
```

→ 파라미터를 실수로 넘기지 않았다면?

```
const title = '';
const colGroupHtml = '';
const headerHtml = '';
const auditTrailHtml = '';
//....

const template = {
        'auditTrailWrap':
            `<div class="audit-wrap">
                        <div class="list-title
bright-gray">
                    <div class="left">${title}<
/div>
                    <div class="right">
                        <a href="#" onclick="
return false;" class="_audit-trail-close">
                        <div><span class="
sprite icon-cancel"></span></div>
                        </a>
                    </div>
                </div>
                <div class="audit-list">
                    <table class="list-table01">
                        <colgroup>${colGroupHtml}<
/colgroup>
                        <thead><tr>${headerHtml}<
/tr></thead>
                        <tbody>${auditTrailHtml}<
/tbody>
                    </table>
                </div>
                        </div>`
        ,
        'entryDataJson': `"${name}":"${value}"`
    };
```

→ 실제 변수를 바인딩 할 수 있음

→ 컴파일 시점에 파라미터를 넘기지 않은 것을 알 수 있음 (함수의 바인딩처럼 동작)

Spread 연산자

```
//Array Spread//     var parts = ['shoulders', 'knees'];
const lyrics = ['head', ...parts, 'and', 'toes'];
// ["head", "shoulders", "knees", "and", "toes"]

// Object  Spread
// Object assigne
//  Object.assign    (setters   )
const obj1 = { foo: 'bar', x: 42 };
const obj2 = { foo: 'baz', y: 13 };

const clonedObj = { ...obj1 };
// Object { foo: "bar", x: 42 }

const mergedObj = { ...obj1, ...obj2 };
// Object { foo: "baz", x: 42, y: 13 }

const user = {
  name : 'lee',
  phone : '010~~~'
  ...
}
//
const {name, phone, email, address} = user;
```

Class

기존 포로토타입 기반 객체지향 패턴을 더 쉽게 사용할 수 있는 대체재

constructor, static 등의 지원과 명확한 상속 구조 구현 가능

```
class SkinnedMesh extends THREE.Mesh {
  constructor(geometry, materials) {
   super(geometry, materials);

   this.idMatrix = SkinnedMesh.defaultMatrix();
   this.bones = [];
   this.boneMatrices = [];
   //...
  }
  update(camera) {
    //...
    super.update();
  }
  get boneCount() {
    return this.bones.length;
  }
  set matrixType(matrixType) {
    this.idMatrix = SkinnedMesh[matrixType]();
  }
  static defaultMatrix() {
    return new THREE.Matrix4();
  }
}
```

그 외 Feature

향상 된 정규표현 식, Property Shortand, Generator, Typed arrays... (매우 유용한 것들이 많습니다)

3. Attribute, Property, Data

HTML의 파싱 과정에서 DOM이 생성되고 HTML tag가 가진 표준 속성(Attribute)이 DOM 객체의 프로퍼티(Property)가 된다.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title></title>
  <style type="text/css">
</head>

<body>
  <div id="target" attr1="attr1" data-attr2="attr2" data-dash-attr="
dash attr"></div>
</body>
<script>
  var target = document.getElementById('target');
  console.log(target.id);
  console.log(target.attr1);
  console.log(target.getAttribute('attr1'));
  console.log(target.dataset.attr2);
  console.log(target.dataset.dashAttr);

  target.attr1 = "attr1 changed";
  console.log(target.attr1);

  target.id = 'newId';
  console.log(target.id);
  console.log(target.getAttribute('id'));

  target.setAttribute('id', 'another changed')
  console.log(target.id);
</script>
</html>
```

target DOM은 id와 dataset 프로퍼티를 가지게 된다

→ data-로 시작하는 attribute는 dataset 프로퍼티로 들어 감
단, dash로 구분된 것은 camel case로 들어감을 주의

attr1 attribute는 property화 되지 않음

→ 표준 Attribute가 아님

DOM에 자신만의 Property를 만들 수 있음

→ target.attr1 = '~~~';

표준 속성이 아닌 속성을 가져오려면 getAttribute를 사용해야 함

→ Attribute control

- hasAttribute(name)
- getAttribute(name)
- setAttribute(name, value)
- removeAttribute(name)

표준 속성이 변경되면 이에 대응하는 프로퍼티도 수정 됨

또한 대응하는 프로퍼티가 수정되면 표준 속성도 수정 됨

비표준 속성과 dataset

HTML 태그를 이용할 때 표준 속성 외에 개발자가 정의한 커스텀 속성의 사용은 반드시 필요 (언제까지 input hidden으로...)
→ 해당 HTML tag가 DOM으로 변경될 때 DOM에 필요 데이터를 넣어줄 경우가 많기 때문 (주로 key 값)

data- 안붙이고 그냥 내가 원하는 형태로 하면 안되나?

```
<div show-info="name”></div>
<div show-info="age”></div>
<script>
 let user = {
 name: “Pete”,
 age: 25
 };

 for(let div of document.querySelectorAll(‘[show-info]’)) {
 //
 let field = div.getAttribute(‘show-info’);
 div.innerHTML = user[field]; // Pete, then age
 }
</script>
```

HTML6에서 show-info 속성이 표준 속성이 되면....?

→ 앞으로는 show-info에 넘긴 값에 따라 DOM이 보이는 구조가 갈립니다

→ 관련된 코드 전부 수정 필요

비표준에 대한 표준

data-* 속성은 표준이 아닌, 개발자가 등록한 속성들.

→ W3C에서 약속된 비표준을 위한 표준

→ 커스템 데이터를 전달하기 위해 안전하고 유요한 방법

해당 속성은 dataset 프로퍼티를 통해 접근할 수 있음

언제사용하나?

(1) 비표준 속성이 필요한 경우

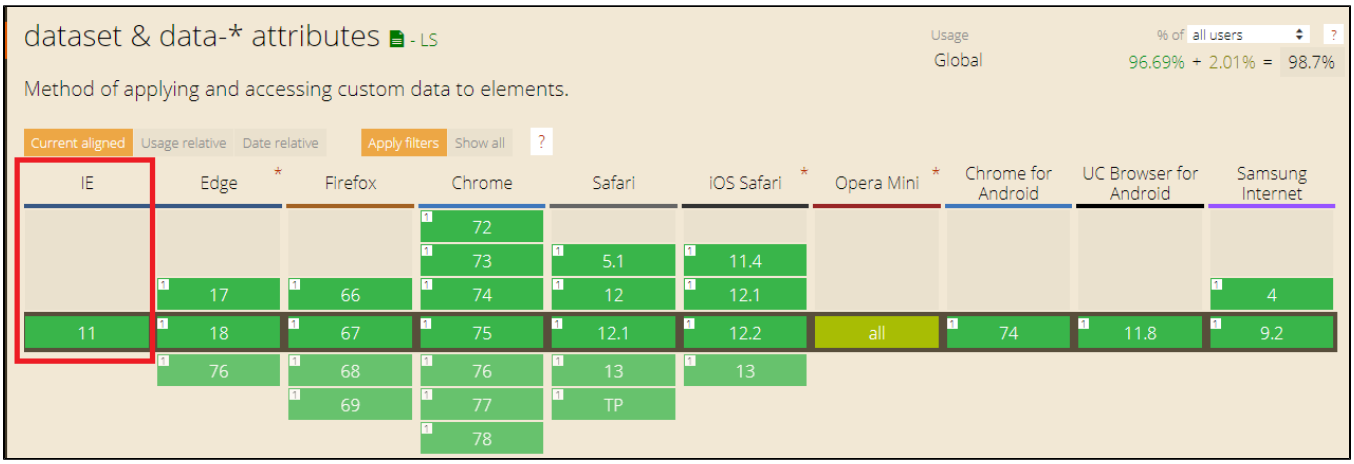
(2) HTML에 적힌 문자 그대로의 값을 읽고싶은 경우 (???)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title></title>
  <style type="text/css">
</head>

<body>
  <a id="href" href="/test">test</a>
  <input id="radio" type="radio" checked="checked" />
</body>
<script>
  var hrefTarget = document.getElementById("href");
  console.log(hrefTarget.href);
  console.log(hrefTarget.getAttribute("href"));

  var radio = document.getElementById("radio");
  console.log(radio.checked);
  console.log(radio.getAttribute("checked"));
</script>
</html>
```

HTML 태그에 attribute가 DOM의 프로퍼티로 들어가면서 일부 값이 바뀌기 때문

주의점

~~(1) 익스플로러~~



IE10 이하 지원을 위해서는 getAttribute를 통해 접근해야 함.

(2) 성능

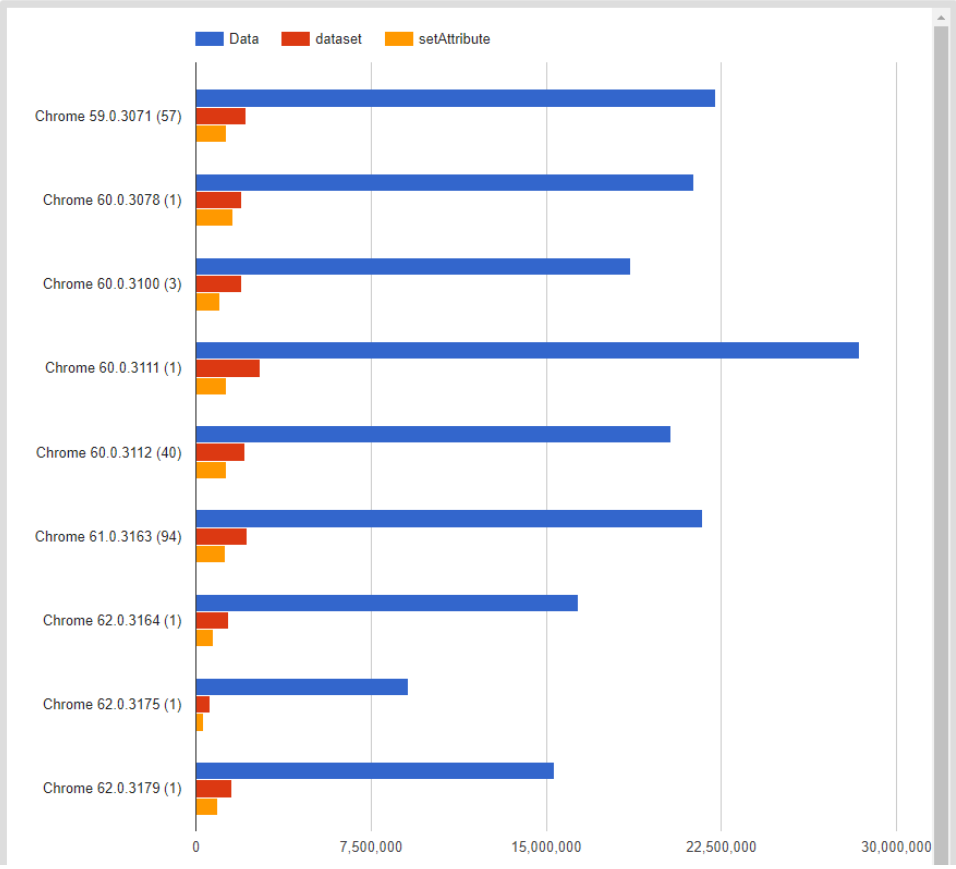
JS Data 저장소에 저장하는것과의 비교

Data 〉 dataset 〉 setAttribute의 순서대로 동일 시간 대비 처리량이 많음

```
<!DOCTYPE
HTML
PUBLIC "-
//W3C
//DTD
HTML 4.01
```

```
Transitional//EN"
"http://www.w3.org
/TR/html4
/loose.
dtd">
<html>
<head>
  <meta
http-
equiv="
Content-
Type"
content="
text
/html;
charset=ut
f-8">
  <title><
/title>
  <style
type="text
/css">
</head>

<body>
  <div
id="
target"><
/div>
</body>
<script>

  var
Data =
function()
{
    var
warehouse
= {};
    var
count = 1;

return {

reset:
function()
{

count = 1;

warehouse
= {};
       },
    set:
function
(dom,
data) {
      if
(!dom.
__data) {

dom.
__data =
"hello" +
count++;
       }

warehouse
[dom.
```

Browserscope



```
__data] =
data;
     },
     get:
function
(dom) {

return
warehouse
[dom.
__data];
     }
    };
  }();

  var div
=
document.
getElement
ById
('target')
;
  Data.set
(div,
{yo:
'yo', ma:
'ma', la:
'la'});
  var
data =
Data.get
(div);
  var a =
data.yo;
  var b =
data.ma;
  var c =
data.la;


console.
log(a, b,
c);
</script>
</html>
```

※ jQuery의 data

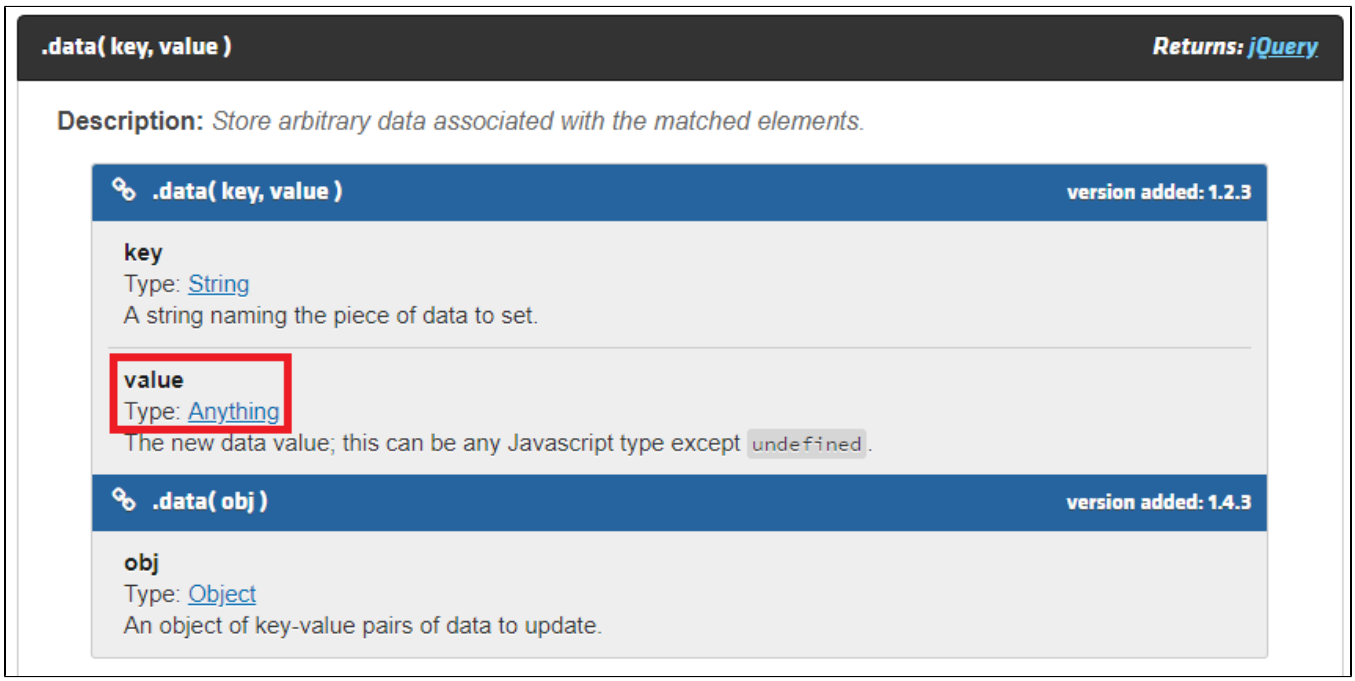
만약 dataset 프로퍼티에 객체를 넣고 싶으면?

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title></title>
  <style type="text/css">
</head>

<body>
  <div id="target"></div>
</body>
    <script src="https://code.jquery.com/jquery-3.1.0.js"></script>
<script>
    var obj = {
      name: 'lee',
      age: 24
    };

    $('#target').data('dat', obj);
    $('#target').attr('atr', obj);
    var target = document.getElementById('target');
    target.dataset.jsData = obj;

    console.log(target.dataset.jsData);                          //"[object Object]"
    console.log(target.dataset.jsData.name);                //undefined

    console.log($('#target').data('dat'));
        /* [object Object] {
           age: 24,
           name: "lee"
      }
        */
    console.log($('#target').data('dat').name);        //"lee"

    console.log($('#target').attr('atr'));                //"[object Object]"
    console.log($('#target').attr('atr').name);        //undefined
</script>
</html>
```

Javascript dataset엔 객체를 넣을 수 없다.

→ 죽어도 넣어야 하겠으면 JSON.stringfy, JSON.parse를 이용....

하지만 jQuery는 가능



참고자료

https://developer.mozilla.org/ko/docs/Learn/HTML/Howto/%EB%8D%B0%EC%9D%B4%ED%84%B0_%EC%86%8D%EC%84%B1_%EC%82%AC%EC%9A%A9%ED%95%98%EA%B8%B0

https://jsperf.com/data-dataset

https://medium.com/@violetboralee/%EC%9E%90%EB%B0%94%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8-%EC%86%8D%EC%84%B1-attribute-%EA%B3%BC-%ED%94%84%EB%A1%9C%ED%8D%BC%ED%8B%B0-property-d2f9b772addf

https://api.jquery.com/data/