Javascript

1. Basic Syntax

1.1 자바스크립트의 특징

- 웹 브라우져에서 동작되는 동적으로 웹 페이지를 변경해주는 언어
- javascript와 actionscript가 있었지만 actionscript는 거의 사라지고 javascript만 남음
- 인터프리터, 동적타이핑, 객체지향
- ECMAScript
    - Ecma International의 조직의 TC-39위원회가 표준화 작업을 진행
    - ECMA5, ECMA6, ECMA8의 버전이 있다.
- 코드문장의 끝에는 ;을 붙여 준다.
- References
    - ECMA 표준문서
        - http://www.ecma-international.org/publications/standards/Ecma-262.htm
    - javascript 레퍼런스
        - https://developer.mozilla.org/en-US/docs/Web/javascript/Reference
    - coffee 스크립트
        - http://coffeescript.org/
        - ruby와 python의 영향을 받음
    - type 스크립트
        - https://www.typescriptlang.org/
        - 인터페이스와 추상클래스의 개념이 추가

1.2 주석

- 주석은 코드를 실행하지 않는 코드로 변경하거나 코드에 대한 설명을 작성할때 사용

- 한줄주석 : // code

- 여러줄 주석 : /* code */

```
// one line comment
/*
multiple-line comment 1
multiple-line comment 2
*/
```

1.3 식별자

- 상수 : SNAKE_KASE (대문자 스네이크 케이스)

- 변수 : camelCase (카멜 케이스)

- 함수 : camelCase (카멜 케이스)

- 모듈 : PascalCase (파스칼 케이스)

- 자바스크립트에서 사용가능한 식별자 특수 기호 : $, _

- 숫자를 가장 앞에 사용할수 없다.

- 예약어를 식별자로 사용할수 없다.

- _name : private variable, private function

- $target : selector를 변수로 사용할때

1.4 출력

- 출력할때는 console.log 함수를 이용하여 출력

```
console.log("hello js");
// 여러개의 데이터를 출력할때는 ,로 구분하여 출력
```

console.log("hello", "js");

1.5 변수

- 자바스크립트는 동적타이핑으로 변수를 할당

- 변수 할당시에는 변수 이름 앞에 var 을 넣어줌

---

var data = 1;

console.log(data);

- 기본 데이터 타입으로는 number, string, function, boolean, undefined, object가 있다.

Number

---

var data = 1;

console.log(typeof data, data);

var data = 1.1;

console.log(typeof data, data);

String

---

var data = 'data';

console.log(typeof data, data);

Boolean

---

var data = true;

console.log(typeof data, data);

Function

---

var a = function(){};

console.log(typeof a);

- 없는 데이터의 표현
  - undefined
  - null
  - NaN

---

var data = undefined;

console.log(typeof data, data);

> undefined undefined

---

var data = null;

console.log(typeof data, data);

> object null

---

var data = NaN;

console.log(typeof data, data);

> number NaN

- null, undefined, NaN 의 차이점
    - undefined : 선언은 되었으나 값이 할당되지 않음
    - null : 선언이 되어 값이 없음이 할당됨
    - NaN : 숫자에서의 undefined, 0/0
    - NaN은 비교연산이 되지 않음

// 값 할당이 안된 undefined

var a;

console.log(typeof a);

// undefined를 숫자로 형변환하면 NaN

console.log(Number(a));

// 없음을 의미하는 null

var a = null;

console.log(a);

// NaN 으로 비교연산하면 항상 false

```
console.log(NaN == NaN, NaN === NaN);
```

- ,로 한줄에 여러개의 변수 선언 가능

```
var a = 1, b = 2;
console.log(a, b);

var a = b = 3;
console.log(a, b);

// 보통은 아래와 같이 사용함
var a = 1,
    b = 2;
console.log(a, b);
```

- 형변환
    - Number : 숫자로 형변환
    - String : 문자로 형변환
    - Boolean : boolean으로 형변환

```
// 문자를 숫자로 형변환
var a = "1";
console.log(typeof a);
console.log(typeof Number(a));
```

// 숫자를 문자로 형변환

var b = 1;

console.log(typeof b);

console.log(typeof String(b));

// 숫자를 boolean으로 형변환 1

var c = 1;

console.log(typeof c);

console.log(typeof Boolean(c), Boolean(c));

// 숫자를 boolean으로 형변환 2

var d = 0;

console.log(typeof d);

console.log(typeof Boolean(d), Boolean(d));

- 묵시적 형변환

  - 여러 문자 타입을 혼합해서 사용하면 묵시적 형변환이 일어남

// 숫자 -> 문자

var b = 1;

```
console.log(typeof b);


var c = "" + b;
console.log(typeof c, c);
```

```
> number
> string 1
```

```
// 문자 -> 숫자
console.log(typeof c)
var d = c - 0;
var e = +c;
console.log(typeof d, d, typeof e, e);
```

```
> string
> number 1 number 1
```

```
// 숫자 -> bool - 논리연산자를 사용
console.log(typeof e)
var f = !e;
console.log(typeof f, f);
```

```
> number
> boolean false
```

1.6 연산자

- 산술연산

- 할당연산

- 비교연산

- 삼항연산

- eval

1.6.1 산술연산

- +,-,*,/,%,++, --

```
console.log(1+2, 1-2, 2*3, 3/2, 5%3);
```

- ++a와 a++의 차이

```
// 대입 후 증가
var a = 1;
var b = a++;
console.log(a, b);
```

```
> 2 1
```

```
// 증가 후 대입
var a = 1;
var b = ++a;
console.log(a, b);
```

> 2 2

- 올림, 반올림, 내림, 제곱, 제곱근, 랜덤등은 Math 객체의 함수를 사용해야함

```
// 반올림, 올림, 내림
console.log(Math.round(10.6), Math.ceil(10.2), Math.floor(10.6));
```

> 11 11 10

```
// 소수 둘째자리까지 반올림해서 출력
// 결과 값이 문자열
var number = 12.3456;
console.log(number.toFixed(2), typeof number.toFixed(2));
```

> 12.35 string

```
// 제곱, 제곱근, 랜덤
console.log(Math.pow(3,2), Math.sqrt(9), Math.random());
```

> 9 3 0.6165747207066672

1.6.2 할당연산

- +=,-=,*=,/=,%=

```
var a = 1;
a -= 2;
console.log(a);
```

1.6.3 비교연산

- ==, ===, !=, !==, >, <, >=, <=
- ==과 ===의 차이
    - == 값을 비교하고, ===는 데이터 타입과 값을 모두 비교

```
console.log(1=='1', 1!='1', 1==='1', 1!=='1')
```

> true false false true

1.6.4 논리연산

- &&, ||, !
- && : and 연산
- || : or 연산
- ! : not 연산

```
console.log(true && true, true && false, false && false)
console.log(true || true, true || false, false || false)
console.log(!true, !false)
```

> true false false
> true true false
> false true

1.6.5 삼항연산

- (조건)?(참):(거짓)

```
var a = 0;
var b = 1;
var result = (a === b) ? true : false;
console.log(result);
```

1.6.6 eval

- 문자열로된 코드를 실행

```
var x = 1;
eval('x++;');
console.log(x);
```

> 2

```
function disp(data){
    console.log(data);
}
var f = "disp";
var x = 1;
eval(f + "(" + x + ")");
```

> 1

- 함수의 이름이나 변수를 문자열로 받은 데이터로 실행하고 싶을때 사용할수 있지만 코드 실행속도가 느리고 악의적으로 사용될수 있기 때문에 사용 자제해야한다.

2. Condition and Loop

- 조건문과 반복문

2.1 Condition - 조건문

- if, else if, else

- switch, case, default

2.1.1 if, else if, else

- if 조건을 확인하고 else if 조건을 확인하고 만족하는 조건이 있으면 만족하는 조건의 구문을 실행하고, 만족하는 조건이 없으면 else 구문의 코드를 실행

```
var data = 20;
if (data > 20) {
    console.log('if');
} else if (data < 20){
    console.log('else if');
} else {
    console.log('else');
}
```

2.1.2 switch, case, default

- case 구문에서 break가 없으면 다음 조건을 확인

- case 구문에서 표현식도 가능하지만 숫자나 문자 리터럴을 사용

```
var n = 2
switch(n){
    case 0: // n === 0
        console.log("zero");
        break;
    case 1: // n === 1
        console.log("one");
        break;
    default:
        console.log("not zero and one");
}
```

```
var n = 1
var m = 0
switch(n){
    case m + 1: // (n ===)이 부분이 생략 m + 1
        console.log("zero");
        break;
    case 1:
        console.log("one");
        break;
    default:
        console.log("not zero and one");
}
```

2.2 Loop - 반복문

2.2.1 while

```
var w = 0;
while (true) {
    w += 1;
    if (w === 5){
        break;
    }
    console.log(w);
}
```

2.2.2 do/while

- do 구문의 코드를 실행하고 while 구문의 표현식에서 true, false를 확인해서 do 구문을 실행할지 판단

```
    do {
        코드
    } while(조건);
```

```
var k = 0;
do {
    k++;
```

```
    console.log(k);
} while( k < 3 );
```

2.2.3 for

- 초기값을 설정하고 조건이 true이면 코드를 실행하고 반복구문을 실행하고 다시 조건을 확인

```
    javascript
    for(초기값; 조건; 반복){
        코드
    }
```

```
for(var k = 0; k < 3; k++){
    console.log(k);
}
```

```
for(var k = 0; k < 5; k++){
    if (k === 3){
        continue;
    }
    console.log(k);
}
```

Quiz 1. 랜덤 함수를 이용하여 Warrior와 Wizard의 데미지를 출력하세요.

랜덤함수 - Math.random() - 0 이상 10 미만의 실수가 출력

Warrior의 데미지 범위는 10 ~ 20

Wizard의 데미지 범위는 5 ~ 35

- for문을 사용하여 한번 셀을 실행할때 5번의 데미지가 출력되도록 하세요.

- 데미지는 정수로 출력하세요.

```
// 전사 데미지
for(var i=0; i<5; i++){
    console.log(Math.round(Math.random() * 10 + 10))
}
```

```
// 마법사 데미지
for(var i=0; i<5; i++){
    console.log(Math.round(Math.random() * 30 + 5))
}
```

---

2. 위의 데미지로 전사의 체력이 500이고 마법사의 체력이 400일때 둘이 싸우면 누가 이기는지 코드로 작성하세요.

- 동시에 공격하여 체력이 동시에 0이되면 같이 죽는걸로 합니다.

```
// 전사와 마법사 전투
var warrior = 500, wizard = 400;

while (true) {
```

```
    warrior_attack = Math.round(Math.random() * 10 + 10);
    wizard_attack = Math.round(Math.random() * 30 + 5);

    warrior -= wizard_attack;
    wizard -= warrior_attack;

    if (warrior <= 0 || wizard <= 0){
        break;
    }
}

if(warrior <= 0 && wizard <= 0) {
    result = "무승부";
} else if(warrior <= 0) {
    result = "마법사 승리";
} else {
    result = "전사 승리";
}

console.log(warrior, wizard);
result
```

---

3. Array and Object

- 배열과 객체

3.1 Array - 배열

- 순서가 있는 데이터의 집합

- []를 사용하여 선언

- 배열의 데이터 타입은 object

```
var ls = ['a','b','c','d'];
typeof ls;
```

```
> 'object'
```

3.1.1 push

- 데이터를 뒤에 추가할때 사용

```
var ls = ['a','b','c','d'];
ls.push('e');
console.log(ls);
```

3.1.2 unshift

- 데이터를 앞에 추가할때 사용

```
var ls = ['a','b','c','d'];
ls.unshift('e');
console.log(ls);
```

3.1.3 pop

- 가장 마지막에 있는 데이터 리턴 후 제거

```
var ls = ['a','b','c','d'];
var result = ls.pop();
console.log(ls, result);
```

> ['a', 'b', 'c'] 'd'

3.1.4 shift

- 가장 앞에 있는 데이터 리턴 후 제거

```
var ls = ['a','b','c','d'];
var result = ls.shift();
console.log(ls, result);
```

3.1.5 indexOf

- 데이터의 인덱스 위치 확인
- 찾는 데이터가 없으면 -1 을 리턴

```
var ls = ['a','b','c','d'];
console.log(ls.indexOf('c'));
console.log(ls.indexOf('e'));
```

> 2

> -1

3.1.6 splice

- splice(start index, range length)

- 리스트 데이터를 인덱스로 잘라 리턴함, 잘려진 데이터는 리스트 변수에서 제거됨

```
var ls = ['a','b','c','d','e'];
console.log(ls.splice(2, 3));
console.log(ls);
```

3.1.7 slice

- slice(start index, end index)

- 리스트 데이터를 인덱스로 자르지만 잘려진 데이터가 리스트 변수에서 제거되지는 않는다.

- 값을 복사 할때 사용한다.

```
var ls = ['a','b','c','d','e'];
console.log(ls.slice(2, 5));
console.log(ls);
```

```
> [ 'c', 'd', 'e' ]
> [ 'a', 'b', 'c', 'd', 'e' ]
```

3.1.8 concat

- 두개의 배열을 합칠때 사용

```
var array1 = ['a', 'b', 'c'];
var array2 = ['d', 'e', 'f'];
console.log(array1.concat(array2));
```

3.1.9 forEach

- 리스트 데이터를 인덱스와 값으로 출력

```
// python의 enumulate와 비슷
var ls = ['a','b','c','d'];
ls.forEach(function(value, index){
    console.log(index, value);
})
```

3.2 Object - 객체

- built in object
    - 자바스크립트 내장 객체 (String, Number, Date, Symbol, Math ...)
- native object
    - 브라우져 내장 객체 (Window, Location, Document ...)
    - Document 객체는 jQuery를 이용
- host object
    - 사용자 정의 객체

3.2.1 객체 선언

- 객체 생성 1
    - 객체 리터럴을 이용
    - 단일 객체로만 활용

```
var obj = {
    name : 'andy'
};
```

```
console.log(obj);
```

```
> { name: 'andy' }
```

- 객체 생성 2
  - constructor를 이용
  - 동일한 구성을 가진 객체를 여러개 만들수 있음

```
// this는 python의 self와 동일함
// 자바스크립트의 ES5 문법까지는 class가 없기때문에 function으로 클래스와 동일하게 구현
function Person(name) {
    this.name = name;
}
var person = new Person('andy');
console.log(person);
```

```
> Person { name: 'andy' }
```

3.2.2 객체 속성

- 객체의 접근
  - 프로퍼티를 이용
  - 아래의 코드에서 name과 같이 값에 접근하기위한 값을 프로퍼티라고 함

```
var obj = {};
obj.name = 'andy';
console.log(obj);
```

> { name: 'andy' }

```
var obj = {};
obj['name'] = 'andy';
console.log(obj);
```

> { name: 'andy' }

- in 을 사용해서 키값을 가져오거나 키값이 포함되었는지 확인

```
var obj = {
    name: 'andy',
    email: 'andy@gmail.com',
    addr: 'seoul'
};

// 키값 확인
console.log("name" in obj);
console.log("phone" in obj);

// 키값 가져오기
for(key in obj){
    console.log(key, ':', obj[key]);
}
```

> true

> false

> name : andy

> email : andy@gmail.com

> addr : seoul

- 객체에 함수 담기

```
var obj = {
    plus: function(a, b){
        return a + b
    }
}
console.log(obj.plus(1, 2), obj['plus'](1, 2));
```

> 3 3

- 속성 삭제
    - delete 키워드를 이용하여 객체의 속성을 삭제

```
var obj = {
    name: 'andy',
    addr: 'seoul'
}
delete obj.addr;
console.log(obj);
```

> { name: 'andy' }

3.2.3 객체 복사

- javascript의 기능을 보완해주기 위해 사용되는 라이브러리

- https://lodash.com/

- 얕은 복사

- 주소 값만 참조되었기 때문에 동일한 메모리 사용

- 값은 같은 주소를 참조 하고 있기 때문에 하나의 변수의 값을 변경하면 복사한 다른 변수도 같이 변경

```
// 얕은 복사를 통해 주소값 복사
var obj = { name: 'andy' };
var data = obj;
console.log(obj, data);

// 이름을 바꾸면 다른 객체도 이름의 값이 바뀜
obj.name = 'park';
console.log(obj, data);
```

```
> { name: 'andy' } { name: 'andy' }
> { name: 'park' } { name: 'park' }
```

- 깊은 복사

- 값과 주소를 모두 복사

- 변수 값을 변경해도 다른 변수의 값이 변경되지 않음

```
// 깊은 복사를 하는 함수를 구현
function clone(obj){
    var result = {};
    for(var key in obj) {
        result[key] = obj[key]
```

```
    }
    return result;
}

// clone 함수를 이용하여 객체 복사
var obj = { name: 'andy' };
var data = clone(obj);
console.log(obj, data);

// name값을 바꿔도 다른 객체에 영향을 주지 않음
obj.name = 'park';
console.log(obj, data);
```

```
> { name: 'andy' } { name: 'andy' }
> { name: 'park' } { name: 'andy' }
```

3.2.4 json 객체

```
// 객체 -> 문자열
var obj = { name:"kim", addr:"seoul"};
var str_obj = JSON.stringify(obj);
console.log(typeof obj, obj, typeof str_obj, str_obj);
```

```
> object { name: 'kim', addr: 'seoul' } string {"name":"kim","addr":"seoul"}
```

```
// 문자열 -> 객체
```

```
var parse_obj = JSON.parse(str_obj);
console.log(typeof str_obj, str_obj, typeof parse_obj, parse_obj);
```

```
> string {"name":"kim","addr":"seoul"} object { name: 'kim', addr: 'seoul' }
```

4. Function

- 함수

4.1 함수의 선언과 호출

- 함수 선언 1 (함수 선언식 : Function Declarations)

```
function plus(a, b){
    return a + b;
}
plus(1, 2);
```

```
> 3
```

- 함수 선언 2 (함수 표현식 : Function Expressions)

```
var minus = function(a, b){
    return a - b;
}
minus(1, 2);
```

- 함수 선언식과 함수 표현식의 차이

```
-----------------------------------------------------------------------------------------------
// 함수 선언식은 함수 호출 후에 선언이 되어도 사용이 가능
funcDeclear();

function funcDeclear(){
    console.log("funcDeclear");
}
-----------------------------------------------------------------------------------------------
> funcDeclear

-----------------------------------------------------------------------------------------------
// 함수 표현식은 함수의 호출 전에 선언되어야 사용이 가능
funcExp();

var funcExp = function(){
    console.log("funcExp");
};
-----------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------
// 호이스팅
console.log(funcExp2);

var funcExp2 = function(){
    console.log("funcExp");
};
-----------------------------------------------------------------------------------------------
```

4.2 호이스팅

- var를 이용하여 변수를 선언하면 호이스팅에 의해 변수의 선언이 최상단으로 코드가 올라가는 기능이 있다.(최상단에서 값이 할당되는것은 아님)

```
// 아래의 hoisting1 함수는 hoisting2 함수와 같이 동작
// python에서는 변수가 선언되어 있지 않다고 에러가 발생함

function hoisting1(){
    console.log("first", data);
    var data = "data";
    console.log("second", data);
}

function hoisting2(){
    var data; // 호이스팅에 의해 최상단에 변수가 선언
    console.log("first", data);
    data = "data";
    console.log("second", data);
}

hoisting1();
hoisting2();
```

```
> first undefined
> second data
> first undefined
> second data
```

4.3 스코프

- var를 생략하고 변수를 선언할수 있지만 var를 사용하지 않고 변수를 선언하게 되면 전역변수로 선언된다.

```
var a = 10;
function test(){
    a = 20; // var를 안 사용하면 전역변수로 선언
    console.log(a);
}
test();
console.log(a);
```

> 20

> 20

```
var a = 10;
function test(){
    var a = 20; // var를 사용해야 지역변수로 선언
    console.log(a);
}
test();
console.log(a);
```

> 20

> 10

4.4 익명함수

- 선언과 동시에 호출

- 전역변수를 사용할수 없도록 익명함수로 선언. 외부에서 내부의 함수나 변수를 사용할수 없다.

- 해당 함수를 콘솔창에서 사용이 불가

```
---------------------------------------------------------------------------------------------------
function disp1(data){
    console.log('disp function', data);
}
disp1(1);
---------------------------------------------------------------------------------------------------


---------------------------------------------------------------------------------------------------
(function disp2(data){
    console.log('disp function', data);
}(2));
---------------------------------------------------------------------------------------------------


---------------------------------------------------------------------------------------------------
(function disp3(data){
    console.log('disp function', data);
})(3);
---------------------------------------------------------------------------------------------------


---------------------------------------------------------------------------------------------------
disp1(1);
---------------------------------------------------------------------------------------------------


---------------------------------------------------------------------------------------------------
// 전역 영역에서 호출이 불가능
disp2(2);
disp3(3);
```

4.5 함수의 인수 설정

- 함수를 호출할때 아규먼트의 수와 파라미터의 수가 맞지 않아도 함수가 호출됨

- 부족한 아규먼트는 파라미터에서 undefined로 들어감

- 함수의 초기값을 설정

```
function plus(a, b){
    console.log("b:", b);
    b = b || 1; // b의 초기값을 1로 선언: b가 undefined 이면 || 뒤의 데이터가 b에 할당됨
    return a + b;
}
console.log(plus(1, 2));
console.log(plus(1));
```

---

5. Module Pattern

- 모듈 패턴을 이용한 객체 생성

```
// Module이름의 객체가 있으면 Module객체를 Module 변수에 넣고 없으면 새로운 객체 생성
var Module = Module || {};

// 즉시 실행 함수
(function(_Module){

    // 변수 선언
    var name = "module";
```

```
    // 함수 선언
    function getName(){
        return name;
    }

    // getter
    _Module.getName = function(){
        return getName()
    };

    // setter
    _Module.setName = function(data){
        name = data;
    };

})(Module);

console.log(Module.getName());

Module.setName("javascript");
console.log(Module.getName());
```

> module
> javascript

5. 웹 브라우저 객체

5.1 window 객체

- 클라이언트 측 자바스크립트에서 가장 중요한 객체

- 전역 객체이며, 전역 변수는 window 객체의 프로퍼티

%%javascript

// console 창에서 결과를 확인할수 있다.

```
var data = 10;
var plus = function(a, b){
    return a + b;
}

console.log(window.data);
console.log(data);

console.log(window.plus(1, 2));
console.log(plus(1, 2));
```

5.2 location 객체

- 창에 표시되는 문서의 URL 관리

- location.href : 현재 URL

- location.origin : 현재 URL의 프로토콜도메인

- location.pathname : 현재 URL의 경로와 파일

- location.port : 현재 URL의 포트

- 이외에도 여러가지 정보를 가지고 있음

%%javascript

console.log(window.location);

console.log(location);

console.log("href :", location.href);

console.log("origin :", location.origin);

console.log("pathname :", location.pathname);

console.log("port :", location.port);

5.3 document 객체

- document.title = "페이지의 타이틀"

%%javascript

document.title = "타이틀 변경";

- document.getElementById("엘리먼트의 아이디이름")

%%html

<input id="id-txt" type="text" value="mail_1">

<script>

  var obj = document.getElementById("id-txt");

  console.log(obj.value);

</script>

- document.getElementsByClassName("엘리먼트의 클래스이름")

- 아이디를 제외한 다른 엘리먼트의 객체는 여러개가 있을수 있음으로 리스트로 반환됨

```
%%html
<input class="class-txt" type="text" value="mail_2">
<script>
    var obj = document.getElementsByClassName("class-txt");
    console.log(obj[0].value);
</script>
```

- document.getElementsByName("name 속성 이름")

```
%%html
<input type="text" name="email" value="mail_3">
<script>
    var obj = document.getElementsByName("email");
    console.log(obj[0].value);
</script>
```

- document.getElementsByTagName("태그 이름")

```
%%html
<input type="text" name="email" value="mail_4">
<script>
    var obj = document.getElementsByTagName("input");
    console.log(obj[3].value);
</script>
```

- document.querySelector("CSS 셀렉터")

```
%%html
<p class="txt">데이터 사이언스</p>
<script>
    var obj = document.querySelector(".txt");
    console.log(obj);
</script>
```

> 데이터 사이언스

6. jQuery

- javascript에서 어렵게 사용하는 웹브라우져 객체들을 쉽게 사용할수 있게 만들어주는 자바스크립트 라이브러리

- https://jquery.com/

- 크게 세가지를 좋은 기능이 있다.

    - 이벤트 핸들링이 편하다.

    - Dom 변경이 편하다.

    - Ajax 통신이 편하다.

- javascript로 클릭 이벤트 만들기

```
%%html
<button class="test-btn">Click</button>
<script>
```

```
    var obj = document.querySelector(".test-btn");
    obj.addEventListener("click", function(){
        alert("TEST");
    });
</script>
```

- jQuery로 클릭 이벤트 만들기

```
%%html
<button class="test2-btn">Click</button>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></
script>
<script>
    $('.test2-btn').on("click", function(){
        alert("jQuery TEST");
    });
</script>
```

- Dom 변경

```
%%html
<p class="data">Data</p>
<button class="test3-btn">Click</button>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></
script>
<script>
```

```
    $('.test3-btn').on("click", function(){
        $('.data').text("Science");
    });
</script>
```

```
%%html
<p class="data2">Science</p>
<button class="test4-btn">Hide</button>
<button class="test5-btn">Show</button>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></
script>
<script>
    $('.test4-btn').on("click", function(){
        $('.data2').hide(1000);
    });
    $('.test5-btn').on("click", function(){
        $('.data2').show(1000);
    });
</script>
```

- Ajax(비동기) 통신으로 데이터 가져오기
    - 직방 사이트의 API로 활용하여 데이터 가져오기

```
%%html
<input type="text" class="addr" value="망원동">
```

```
<button class="get-data">Info</button>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></
script>

<script>
    $('.get-data').on("click", function(){
        var addr = $('.addr').val();
        var url = "https://apis.zigbang.com/search/all?q=" + addr + "&type=oneroom";
        $.getJSON(url, function(result){
            console.log(result.items[0]);
        })
    });
</script>
```
