Javascript 기초

작성자: 신행준(hjshin1218@naver.com)

1. 들어가며

본 문서는 ECMAScript 5가 기준이며, ECMAScript 6에 대한 내용은 다루지 않으며, Fiori App개발을 위한 JavaScript 최소한의 수준만은 언급

2. JavaScript 정의

Webpage 개발에 사용되는 언어는 HTML, CSS, JavaScript 세 개로 HTML은 Webpage의 골격을, CSS는 Webpage의 스타일을 담당하고 있으며, JavaScript는 Webpage의 움직임 및 상호작용을 담당하기 위해 만들어짐

3. JavaScript 사용법

JavaScript는 HTML의 script tag내에 기술 또는 별도의 파일로 참조하여 사용이 가능하며 script tag는 JavaScript같은 언어를 HTML내에서 직접 작성이 가능한 공간을 확보하기 위하여 만들어진 tag임

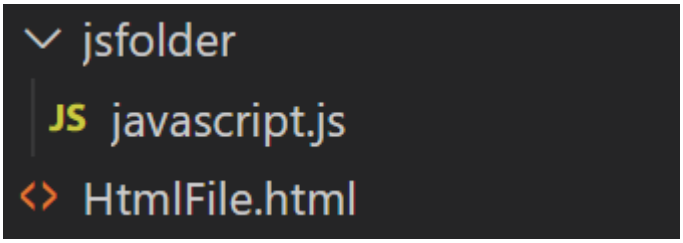
2.1. HTML내에 직접 작성

```
<script>
    //js 작성
</script>
```

2.2. 외부에 존재하는 JavaScript파일을 참조

```
<script src="jsfolder/javascript.js"></script>
```

P.S. JavaScript 위치



4. JavaScript 기초

- **주석(Comment)**

  Logic에 영향이 없으며 설명 또는 이해를 돕기 위해 소스 내에 기술 해놓은 메모

  - Line: `// 내용 기술`
  - Block: `/* 내용 기술 */`

- **세미콜론(;)**

  JavaScript Interpreter에게 하나의 문자이 끝났음을 알려주는 역할을 담당하고 있으며, 없어도 동작하나 100% 보장할 수도 없으며, 예기치 않은 오류 등을 피하기 위하여 습관을 들이는 것을 추천

- **중괄호({})**

  중괄호는 어디서 사용했는지에 따라서 그 의미가 달라짐. 변수 선언 시에 사용하면 객체라는 의미로 사용되지만, **함수(function), 조건문(if, switch), 반복문(for, while, do ~ while)에서 사용되면 해당 구문이 수행하는 Logic의 범위(Scope)**라는 의미로 해석되며 이는 **블록(Block)**이라고 불림

- **줄 바꿈 및 들여쓰기**

  줄 바꿈 및 들여쓰기는 주석과 마찬가지로 Logic에 영향은 없으나 Source를 볼 때 가독성을 높이기 위하여 맞추어 주는 것을 추천하며, IDE별로 자체적으로 제공되거나 Plugin / Extension으로 제공되는 Code formatter를 사용

- **변수(Variable)**

  Program에서 사용할 Data를 담아두는 공간으로, 복잡한 연산 또는 지속적인 사용이 필요한 Data를 담아두는 위해 사용

- **자료형(Data Type)**

  프로그래밍 언어에서 실수, 정수, 문자 등 여러 종류의 데이터를 식별하는 분류로서, 더 나아가면 해당 자료형에서 사용 가능한 값 및 사용 가능한 명령, 데이터의 의미, 값을 저장하는 방식을 결정

- **배열**

  변수와 마찬가지로 Program에서 사용할 Data를 담아두는 공간으로서, 변수와는 달리 일련의 순서로 Data를 담고있는 객체

- **객체**

  물리적으로 존재하거나 또는 추상적으로 생각할 수 있는 것 중에서 자신만의 **속성** 및 동작을 가지고 있고 다른 것과 식별이 가능한 것을 의미

- **함수**

  복잡한 계산 또는 특정한 목적을 수행하는 Logic의 집합으로 Program의 모듈화 및 코드 재 사용성을 높이는 장점이 있음

- **제어문(Control Statement)**

  JavaScript의 Logic을 특정한 조건식을 기준으로 분기시키거나 반복시키는 등 일련의Logic 제어를 수행하는 문장

5. 연산자

- **산술 연산자**
  - ✓ **+**

    좌변과 우변의 값을 더하거나 문자열을 결합한 결과를 반환

    Ex)

    ```
    1 + 2            // 결과는 숫자(3)
    “Hi, ” + “hello”        // 결과는 문자열(Hi, hello)
    “123” + 456      // 결과는 문자열(123456)
    ```

  - ✓ **-**

    좌변의 값에서 우변의 값을 뺀 결과 값을 반환

    Ex)

    ```
    5 – 2            // 결과는 숫자(3)
    ```

  - ✓ *****

    좌변의 값과 우변의 값을 곱한 결과 값을 반환

    Ex)

    ```
    4 * 2            // 결과는 숫자(8)
    ```

  - ✓ **/**

    좌변의 값을 우변의 값으로 나눈 결과 값을 반환

    Ex)

    ```
    6 / 2            // 결과는 숫자(3)
    ```

  - ✓ **%**

    좌변의 값을 우변의 값으로 나눈 나머지 값을 반환

    Ex)

    ```
    7 % 4            // 결과는 숫자(3)
    ```

- **대입 연산자**
  - ✓ =

    우변의 값을 좌변의 변수에 할당

    Ex)

    ```
    iAge = 10;      // 변수(iAge)에서 숫자(10)을 할당
    ```

- **비교 연산자**
  - ✓ <

    좌변이 우변보다 작으면 true를 반환하고 아닌 경우에는 false를 반환
  - ✓ >

    좌변이 우변보다 크면 true를 반환하고 아닌 경우에는 false를 반환
  - ✓ <=

    좌변이 우변보다 작거나 같으면 true를 반환하고 아닌 경우에는 false를 반환
  - ✓ >=

    좌변이 우변보다 크거나 같으면 true를 반환하고 아닌 경우에는 false를 반환
  - ✓ !=

    좌변과 우변의 값이 다르면 true를 반환하고 아닌 경우에는 false를 반환
  - ✓ !==

    좌변과 우변이 다르면 true를 반환하고 아닌 경우에는 false를 반환
  - ✓ ==

    좌변과 우변의 값이 **등등한지** 판단

    판단기준)

    - 값 하나가 null이고 다른 하나가 undefined이면 두 값은 동등하다
    - 값 하나가 숫자이고 다른 하나가 문자열이면 문자열을 숫자로 변환 후 다시 비교한다
    - 두 값 중 하나가 true 이면 그것을 1 로 변환한 후 다시 비교한다. 두 값 중 하나가 false 이면 그 값을 0 으로 변환한 후 다시 비교
    - 값 하나가 객체이고 다른 하나가 숫자나 문자열이면 객체를 원시형 값으로 변환 후 다시 비교한다. 객체는 toString()이나 valueOf() 메서드를 통해 원시형 값으로 변환된다.

- 그 밖의 다른 값의 조합은 동등하지 않다.

Ex)

```
4 == 4          // 값이 동등하므로 결과는 논리 값(true)

4 == “4”                // 값이 동등하므로 결과는 논리 값(true)
```

✓ **===**

좌변과 우변의 값이 **같은지** 판단

판단기준)

- 두 값이 다른 타입이면 두 값은 일치하지 않는다.
- 두 값이 숫자이고 값이 같으면 두 값은 일치한다. 두 값이 모두 NaN 이거나 둘 중 하나가 NaN 이면 두 값은 일치하지 않는다.
- 두 값이 문자열이고 정확히 동일한 문자를 같은 위치에 가지고 있으면 두 값은 일치한다.
- 두 값이 모두 true 거나 false 이면 두 값은 일치한다.
- 두 값이 모두 동일한 객체나 배열, 함수를 가리킬 경우 두 값은 일치한다. 두 값이 다른 객체(또는 배열이나 함수)를 참조하면 두 값은 일치하지 않는다.
- 두 값이 모두 null 이거나 undefined 이면 두 값은 일치한다.

Ex)

```
4 === 4         // 값이 같으므로 결과는 논리 값(true)

4 === “4”       // 값이 다르므로 결과는 논리 값(false)
```

- **논리 연산자**
  - ✓ &&

    좌변과 우변이 모두 true일 경우 true를 반환하고 하나라도 false인 경우에는 false를 반환

    Ex)

    ```
    true && true    // 결과는 논리 값(true)

    true && false   // 결과는 논리 값(false)
    ```

  - ✓ ||

    좌변과 우변 중 하나라도 true일 경우 true를 반환하고 모두 false일 경우에는 false를 반환

    Ex)

    ```
    true || false   // 결과는 논리 값(true)

    false || false  // 결과는 논리 값(false)
    ```

  - ✓ !

    값이 false일 경우에는 true를 반환하고 true일 경우에는 false를 반환

    Ex)

    ```
    !true           // 결과는 논리 값(false)

    !false          // 결과는 논리 값(true)
    ```

6. 변수(Variable)

- **명명규칙(Naming Rule)**
  - **숫자, 영문자, 특수 기호($, _)만 사용 가능**
  - **첫 글자로 숫자는 사용 불가**
  - **예약어 사용 불가**

- **선언**

  JavaScript Interpreter에게 변수를 선언하겠다고 알려주는 명령어(var)를 이용

  Ex)

  - iTotalAmount라는 이름의 변수를 선언

    ```
    var iTotalAmount;
    ```

  - sName이라는 이름의 변수를 선언하고 문자열(홍길동) 값으로 초기값을 설정

    ```
    var sName = "홍길동";
    ```

- **유효범위(Scope)**

  Scope는 선언된 변수가 유효한 범위를 의미

  Ex-1)

```
var fnLog = function() {
    // 선언된 변수(iMax)가 영향을 미칠 수 있는 유효한 범위는 { ~ } 임
    var iMax = 999;
    console.log(iMax);              // 변수(iMax)값 정상 출력
};
fnLog();                    // 함수 실행
console.log(iMax);          // 구문 오류 발생
```

Ex-2)

```
var iMax = 999;

var fnLog = function() {

    // 자신의 Scope내에는 변수(iMax)가 선언되어 있지 않지만 상위 Scope에 존재

    console.log(iMax);              // 변수(iMax) 값 정상 출력

};

fnLog();                    // 함수 실행

console.log(iMax);          // 변수(iMax) 값 정상 출력
```

- **참고사항**
  - ✓ **예약어**

    JavaScript에서 사용하기 위하여 예약되어 있는 단어로 var, function, for등이 있음

  - ✓ **use strict**

    이전 JavaScript에서 허용되던 잘못된 구문들을 오류로 변경(엄격 모드)하여 보다안정적인 코드를 만들 수 작성할 수 있도록 유도

7. 자료형(Data Type)

- **원시 타입(Primitive Type)**

  number, string, boolean, null, undefined**으로** 정의되어 있으며, 원시 타입은 **할당된 값(Data)을 메모리(변수)에 그대로 저장**

  ✓ **숫자/number**

  음수는 앞에 부호(-)를 넣어서 셋팅

  Ex)

  ```
  var iMax = 100;

  var iMinusValue = -9;
  ```

  ✓ **문자열/string**

  문자의 집합을 의미하며, 따옴표(‘) 혹은 쌍 따옴표(“)로 감싸서 표현

  Ex)

  ```
  var sHello = ‘Hello!!;

  var sName = “Jun’’;
  ```

  ✓ **논리값/boolean**

  참(true), 거짓(false) 값으로 표현

  Ex)

  ```
  var bExists = false;

  var bContinum = true;
  ```

  ✓ **null**

  아직 값이 정해지지 않았음을 의미

  Ex)

  ```
  var oValue = null;
  ```

✓ **undefined**

아직 자료형이 정해지지 않았음을 의미

Ex)

```
var iValue;
```

- **참고사항**

✓ **falsy value 종류**

true/false와 같은 Boolean이 아니지만 Interpreter가 문맥상 false로 인식하는 값

1. false
2. 숫자 0
3. “” 또는 ‘’
4. null
5. undefined
6. NaN

✓ **null 과 undefined 비교**

```
var Value1 = null;
var Value2;

console.log(Value1 == Value2)            // 판단 결과는 논리 값(true)
```

설명

null과 undefined는 비교 연산 시에 falsy value로 취급되므로 궁극적으로는 false == false을 비교하는 것이 되므로 결과는 true가 출력

```
console.log(Value1 === Value2)          // 판단 결과는 논리 값(false)
```

설명

null과 undefined는 비교 연산 시에 falsy value로 취급되나 Value1의 자료형은 object이고 Value2의 자료형이 정의되지 않은 undefined이므로 false가 출력

- **참조 타입(Reference Type)**

  array, object, function가 있으며, 원시 타입과는 달리 **할당된 값(Data)은 별도의 메모리에 저장되고 메모리(변수)에는 값(Data)이 저장된 메모리의 주소 값이 저장되고, 출력 시에는 주소 값에 저장되어 있는 값(Data)을 찾아서 출력**

- **원시 타입과 참조 타입 비교**

  ✓ **원시 타입의 경우**

```
var iVal1 = 100;

var iVal2 = iVal1;

iVal1 = 200;

console.log(iVal1);             // 결과는 숫자(200)

console.log(iVal2);             // 결과는 숫자(100)
```

  ✓ **참조 타입의 경우**

```
var oNumber1 = { seq: 100 };

var oNumber2 = oNumber1;

oNumber1.seq = 200;

console.log(oNumber1.seq);      // 결과는 숫자(200)

console.log(oNumber2.seq);      // 결과는 숫자(200)
```

위 소스에서 oNumber1의 값(seq)만 변경했는데 oNumber2의 값(seq)이 같이 변경된 이유는 var oNumber2 = oNumber1이라는 구문으로 새로운 oNumber2에 oNumber1의 값을 넣었다고 생각할 수 있지만, 참조 타입의 특성으로 인해 oNumber2는 oNumber1의 메모리 주소만을 복사한 것이고, 결국 oNumber1, oNumber2는 동일한 데이터를 바라보고 있는 것으로 JavaScript에서 이를 shallow copy(얕은 복사)라고 하며, 참조가 끊어진 형태로 데이터를 복사하는 것을 deep copy(깊은 복사)라고 한다.

8. 배열/array

- **선언**

  대괄호[]를 사용해서 선언

  Ex)

  ```
  var aEmtpy = [];                // 빈 배열
  var aNumber = [ 1, 2, 3, 4 ];   // 숫자 1, 2, 3, 4가 순서대로 들어있는 배열
  ```

- **순번/Index**

  JavaScript의 배열의 순번은 1이 아닌 **0부터 시작**

- **값 할당**

  형식)

  배열변수명[순번] = 값;

  Ex)

  ```
  var aCharacter = [];
  aCharacter[0] = “가”;              // 배열변수(aCharacter)의 위치(0)에 “가”를 할당
  ```

- **값 접근**

  배열변수명[순번]

  Ex)

  ```
  var aCharacter = [ “가” ];
  console.log(aCharacter[0]);        // 결과는 문자열(가)
  ```

- **속성(Property)와 행위(Method)**
  - ✓ **속성 - length**

    배열의 길이를 반환

    Ex)

    ```
    var aList = [ 1, 2, 3 ];
    console.log(aList.length);              // 결과는 숫자(3)
    ```

  - ✓ **행위 - unshift**

    배열 처음에 값을 추가한 후 배열의 길이를 반환

    Ex)

    ```
    var aList = [ “b”, “c” ];
    console.log(aList.unshift(“a”));        // 결과는 숫자(3)
    console.log(aList);                     // 결과는 배열([ “a”, “b”, “c” ])
    ```

  - ✓ **행위 - shift**

    배열 첫번째 값을 삭제한 후 삭제된 값을 반한

    Ex)

    ```
    var aList = [ “a”, “b”, “c” ];
    console.log(aList.shift(“d”));          // 결과는 문자(a)
    console.log(aList);                     // 결과는 배열([ “b”, “c” ])
    ```

  - ✓ **행위 – push**

    배열 끝에 값을 추가한 후 배열의 길이를 반환

    Ex)

    ```
    var aList = [ “a”, “b” ];
    console.log(aList.push(“c”));           // 결과는 숫자(3)
    console.log(aList);                     // 결과는 배열([ “a”, “b”, “c” ])
    ```

✓ **행위 – pop**

배열 끝의 값을 삭제한 후 삭제된 값을 반환

Ex)

```
var aList = [ “a”, “b”, “c” ];

console.log(aList.pop());       // 결과는 문자(c)

console.log(aList);             // 결과는 배열([ “a”, “b” ])
```

9. 객체/object

- **선언**

  중괄호{}를 사용하여 선언되며 key: value 구조로 되어 있음

  Ex)

```
var oEmtpy = {};    // 빈 객체
var oPerson = {
    name: “홍길동”,
    age: 15
};                  // 3개(name, age))의 속성(Property)을 가진 객체
```

- **속성(Property)와 행위(Method)**

  ✓ **속성/Property**

  객체의 특징으로 value에 함수 이외의 값을 넣어서 생성하고, 이때 key는 속성명이라고 부름

  형식)

  속성명: 값 또는 객체변수명.**속성명 = 값;**

  Ex-1) 할당

```
// 객체변수(oPerson) 생성 시에 key(name) 생성 및 value(홍길동) 할당
var oPerson = { name: “홍길동” };

// 객체변수(oPerson)에 key(name)을 생성 및 value(홍길동) 할당
var oPerson = {};
oPerson.name = “홍길동”;
```

  Ex-2) 접근

```
var oPerson = { name: “홍길동” };
console.log(oPerson.name);              // 결과는 문자열(홍길동)
```

  ✓ **행위/Method**

  객체가 수행할 수 있는 Logic으로 value에 함수를 넣을 때 생성되며, 이때 key는 행위명이라고 부름

형식)

행위명: function() { ~ } 또는 객체변수명**.행위명 = function() { ~ }**

Ex-1) 선언

```
// 객체변수(oJS) 생성 시에 key(fnHello) 생성 및 value(function() { ~ })을 할당
var oJS = {
        fnHello: function() {
                console.log("Welcome to JavaScript World!");
        }
};

// 객체변수(oJS)에 key(fnHello)을 생성 및 value(function() { ~ })을 할당
var oJS = {};
oJS.fnHello = function() {
        console.log("Welcome to JavaScript World!");
}
```

Ex-2) 호출

```
var oJS = {
        fnHello: function() {
                console.log("Welcome to JavaScript");
        }
};
console.log(oJS.fnHello());             // 결과는 문자열(Welcome to JavaScript)
```

10. 함수/function

- **선언**

  매개변수(Parameter), Logic, 반환 값(Return value)으로 구성된 일련의 목적을 가진 Logic의 모음

  ✓ **매개변수(Parameter)**

  함수 Logic에서 사용되는 값을 전달받을 변수 명으로 콤마(,)로 구분하며, 필요한 만큼 선언이 가능하고 불필요한 경우 정의하지 않을 수 있음

  ✓ **반환 값(Return value)**

  함수의 결과 값을 호출한 부분으로 반환하기 위해서 사용하며, 반환해야 하는 값이 없는 경우에는 삭제 가능

  Ex-1) 익명(Anonymous)함수 선언

```
function(매개변수) {
    Logic 기술;
    return 반환 값;
}
```

  Ex-2) 기명(Named)함수 선언

```
function 함수이름(매개변수) {
    Logic 기술;
    return 반환 값;
}
```

- **호출**

  미리 만들어둔 함수를 실행하는 것을 의미하는 것으로 아래와 같이 기술, 파라미터 및 반환 값은 호출하고자 하는 함수에 정의 된 것을 기준으로 작성

  형식)

  함수변수명(파라미터) 또는 **함수이름(파라미터)**

  Ex-1)

```
// 기명함수가 아니므로 호출하기 위해 변수(fnAdd)에 할당
```

```
var fnAdd = function(val1, val2) {
      var sum;
      console.log(“val1: “ + val1);  // 첫번째 파라미터(val1) 값인 10을 출력
      console.log(“val2: “ + val2);  // 두번째 파라미터(val2) 값인 20을 출력

      sum = val1 + val2;             // va1, val2 값을 합산하여 sum에 할당
      return sum;                    // sum값을 반환
   }

   // 익명함수가 할당된 변수명(fnAdd)에 함수에 선언된 파라미터명(val1, val2)에
   // 각각 들어갈 10, 20을 순차적으로 기술한 후 함수를 호출, 함수에서 돌려준
   // 반환 값은 변수(iSum)로 받는다
   var iSum = fnAdd(10, 20);
   console.log(iSum);                        // 결과는 숫자(30)
```

11. 제어문(Control Statement)

- **조건문**

  조건식의 판단 결과(true / false)에 따라서 Logic을 분기 처리하는 것을 의미

  ✓ **if**

  형식)

  If (조건식1) {

  조건식1의 결과 값이 true이면 수행되는 Logic

  } else if (조건식2) {

  조건식1의 결과 값이 false이고 조건식2의 결과 값이 true이면 수행되는 Logic

  } else {

  조건식1, 조건식2의 결과 값이 모두 false이면 수행되는 Logic

  }

  참고사항)

  - else if인 경우에는 필요한 만큼 추가하여 기술
  - else는 없거나 한 개만 기술
  - **블록을 사용하여 수행할 Logic의 범위를 기술하지 않은 경우 JavaScript Interpreter는 조건식 아래 한 문장만을 수행할 Logic으로 인식**

  Ex)

```
var iVal = 7;
if (iVal === 7)
        // 조건식의 판단 결과가 true일 때 수행되는 문장
        console.log("iVal의 값은 7입니다");
else
        // 조건식의 판단 결과가 false일 때 수행되는 문장
        console.log("iVal의 값은 7이 아닙니다");
```

✓ **switch case문 형식**

형식)

```
switch (변수) {
        case 값1:
                // 변수 값과 값1이 동일할 때 수행되는 Logic
                break;
        case 값2:
                // 변수 값과 값2가 동일할 때 수행되는 Logic
                break;
        case 값n:
                // 변수 값과 값n이 동일할 때 수행되는 Logic
                break;
        // 변수의 값이 값1 ~ 값n에 해당되지 않는 경우 수행되는 Logic
        default:
}
```

참고사항)

- **Logic 수행 후 break가 없을 시 break를 만날 때까지 아래에 모든 Logic이 수행**되므로 의도적으로 break를 제거하는 경우도 있음

Ex)

```
var iVal;


switch (iVal) {
        case 1:
                console.log("iVal의 값은 1");
                break;
        case 2:
                console.log("iVal의 값은 2");
        default::
```

```
            console.log("iVal값이 유효하지 않습니다");
    }
```

설명)

- iVal이 1인 경우 문자열(iVal의 값은 1)이 출력되고 break문이 존재하므로 switch case 구문을 벗어남
- iVal이 2인 경우 문자열(iVal의 값은 2, iVal값이 유효하지 않습니다)이 출력
- iVal값이 1 또는 2가 아닌 경우 문자열(iVal값이 유효하지 않습니다)이 출력

- **반복문**

  조건식의 판단 결과가 true일 동안 Logic을 반복하여 수행

  ✓ **for**

  형식)

```
for (초기화식; 조건식; 증감식) {
        반복적으로 수행할 Logic
        break;          // Logic 수행을 중단하고 블록을 벗어남
        continue;       // 증감식 수행 후 조건식으로 수행 여부를 판단
};
```

  - **초기화식(선택사항)**

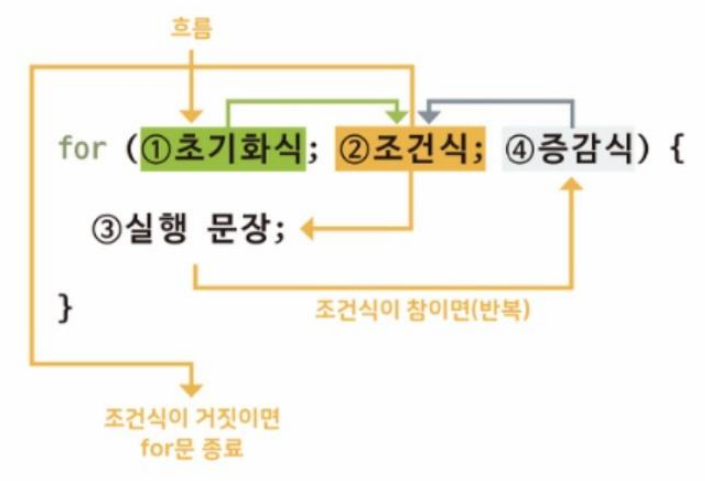
    for문을 위한 변수를 초기화하는 공간

  - **조건식(필수사항)**

    블록 안의 Logic을 계속 실행할지 여부를 판단

  - **증감식(선택사항)**

    블록 안의 Logic이 한번 수행이 끝났을 때 실행되는 구문



✓ **while**

형식)

```
while (조건식) {
        반복적으로 수행할 Logic
}
```

- 조건식의 결과가 false인 경우 반복 Logic은 한번도 수행하지 않을 수 있음
- 조건에 따라서 한번도 수행하지 않을 수도 있는 반복 Logic인 경우 사용

✓ **do while**

형식)

```
do {
        반복적으로 수행할 Logic
} while (조건식)
```

- 무조건 한번은 반복 Logic은 수행됨
- 최소한 한번은 무조건 수행되어야 하는 반복 Logic인 경우 사용

- **참고사항**

  반복적으로 처리해야 하는 데이터 개수가 명확한 경우에는 for를 사용하고, 가변적인 경우에 최소 한번은 실행해야 여부에 따라서 while 또는 do ~ while을 사용

12. 참고 사이트

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference