



웹 프로그래밍

HTML5, CSS3, Javascript



학습 내용

- I. 웹 표준 소개
- II. HTML
- III. CSS
- IV. JavaScript
- V. jQuery

Javascript

- <https://tc39.es>



비영리 기구 Ecma International은 Javascript를 포함한 다양한 기술 표준을 목적으로 하는 단체

그 중 TC39 위원회(committee)는 자바스크립트(ECMAScript) 표준 제정을 담당함.

이 위원회에는 Google, Microsoft, Apple 등 웹 기술과 관계가 깊은 거대 기술 벤더들이 참여하고 있음

대부분의 논의 내용이 Github 등 웹에 공개 되어 있음

Javascript 설정 방법

- Html 문서내 설정 방법

```
<script>
```

```
    const myName = document.getElementById('my-name');  
    console.log(myName.innerText);
```

```
</script>
```

- 외부 파일(스크립트) 참조 방식

```
<script src="/js/main.js"> </script>
```

./js/main.js 파일 저장

```
const myName = document.getElementById('my-name');  
console.log(myName.innerText);
```

<script> 태그 속성 이해

- 스크립트 코드를 문서에 포함하거나 외부 스크립트 참조

속성	의미	값	특징
src	참조할 외부 스크립트 URL	URL	포함된 스크립트 코드는 무시됨
type	MIME 타입	text/javascript(기본값)	생략해도 됨
async	스크립트의 비동기적(Asynchronously) 실행 여부	불린(Boolean)	src 속성 필수
defer	페이지가 모두 로드 된 후에 해당 외부 스크립트가 실행됨을 명시 문서 파싱(구문 분석) 후 작동 여부	불린(Boolean)	src 속성 필수

학습목표

- 자바스크립트의 기본 용어를 이해한다.
- 자바스크립트 기본 자료형과 연산자 사용방법을 익힌다.
- 조건문, 반복문을 사용해 프로그램 흐름을 제어한다.
- 자바스크립트 코딩 방법을 익힌다.

학습내용

내용	설명
자료와 변수	프로그램 개발의 첫걸음. 자료형과 변수 학습
조건문	프로그램의 흐름을 변화시키는 요소. 조건문의 종류를 알아보고 사용 방법을 이해
반복문	배열의 개념과 문법을 익혀 while 반복문과 for 반복문 학습
함수	다양한 형태의 함수를 만들기와 매개변수를 다루는 방법 이해
객체	객체의 속성과 메소드, 생성, 관리하는 기본 문법 학습
문서 객체 모델	DOMContentLoaded 이벤트를 사용한 문서 객체 조작과 다양한 이벤트의 사용 방법 이해
예외 처리	구문 오류와 예외를 구분하고, 예외 처리의 필요성과 예외를 강제로 발생시키는 방법을 이해
클래스	객체 지향을 이해하고 클래스의 개념과 문법 학습
jQuery, Ajaxs	jQuery 사용법 기본 익히기

1. 자바스크립트 기본

HTML 페이지 구조 작성방법

- 내부 자바스크립트

<script> 태그를 사용해 HTML 페이지 내부에 코드 작성

- 외부 자바스크립트

<script> 태그의 src 속성에 파일 경로를 입력해 HTML 페이지로 불러옴

HTML 페이지 구조 작성방법

- 내부 자바스크립트 작성 방법

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <title>HTML Intro</title>
  <script>
    // 경고창 출력
    alert('Hello HTML')
  </script>
</head>
<body>
  <h1>여러분을 환영합니다!!</h1>
</body>
</html>
```

HTML 페이지 구조 작성방법

- 외부 자바스크립트 작성 방법

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <title>HTML Intro</title>
  <script src="./js/test.js"></script>
</head>
<body>
  <h1>여러분을 환영합니다!!</h1>
</body>
</html>
```

자바스크립트 파일

test.js

```
alert('Hello HTML');
```

자바스크립트 개요와 개발환경 설정

Javascript란?

- 자바스크립트(JavaScript)는 웹 브라우저에서 사용하는 프로그래밍 언어
- 웹 클라이언트 애플리케이션 개발
 - 초기의 웹은 변하지 않는 정적인 글자로 이뤄진 커다란 책 →
자바스크립트가 나오며 웹 문서의 내용을 동적으로 바꾸거나
사용자의 마우스 클릭과 같은 이벤트 처리가 가능



Javascript로 할 수 있는 것들

- 웹 서버 애플리케이션 개발
 - 기존 웹 개발에는 2 가지 이상의 프로그래밍 언어가 필요
 - 웹 클라이언트 애플리케이션을 자바스크립트로 개발하고, 웹 서버 애플리케이션은 C#, 자바(Java), 루비(Ruby), 파이썬(Python) 등
 - 2009년에 **Node.js**가 등장하면서 자바스크립트만으로 **웹 서버 애플리케이션 개발**이 가능해짐
 - Node.js의 장단점
 - 웹 서버 애플리케이션을 개발할 때 꼭 필요한 간단한 모듈만 제공하므로 데이터 처리와 예외 처리 등이 조금 복잡
 - 빠른 속도로 서버 구매 비용과 유지 비용이 1/10 수준

Javascript로 할 수 있는 것들

- 모바일 애플리케이션 개발
 - 페이스북의 리액트 네이티브(React Native) : 자바스크립트만으로 모든 운영체제에서 빠르게 작동하는 네이티브 애플리케이션 작성 가능
 - 안드로이드폰은 자바/코틀린(Kotlin), 아이폰은 스위프트(Swift) 프로그래밍 언어로 개발
- 데스크톱 애플리케이션 개발
 - 깃허브(GitHub)에서 자바스크립트 개발 전용 텍스트 에디터인 아톰(Atom) 배포: 일렉트론
 - 일렉트론으로 개발된 애플리케이션: 마이크로소프트의 비주얼 스튜디오 코드(Visual Studio Code), 디스코드 (Discord) 클라이언트, 깃허브 데스크톱 클라이언트, 워드프레스(Wordpress) 데스크톱 클라이언트, 몽고디비 (MongoDB), 데이터 관리 도구 컴파스(Compass) 등
- **데이터베이스 관리**
 - **MongoDB**: 데이터베이스 관리에 자바스크립트를 활용하는 대표적인 NoSQL 데이터베이스

SECTION 1-1 자바스크립트의 활용(3)

- 자바스크립트의 종류
 - 1990년대 중반부터 자바스크립트가 많은 곳에서 사용되자 유럽컴퓨터제조협회(ECMA)는 자바스크립트를 ECMAScript라는 이름으로 표준화
 - 2000년대 중반부터 자바스크립트가 많은 곳에서 널리 사용되며, 자바스크립트의 문법이 급속도로 발전

ECMAScript	버전 표준 발표 시기
ECMAScript 1	1997년 6월
ECMAScript 2	1998년 6월
ECMAScript 3	1999년 12월
ECMAScript 4	2008년 10월
ECMAScript 5	2009년 12월
ECMAScript 2015	2015년 6월
ECMAScript 2020	2020년 6월

▲ ECMAScript 6부터는 발표 연도를 사용해서 ECMAScript 2015와 같이 버전을 부르는 경우가 일반적

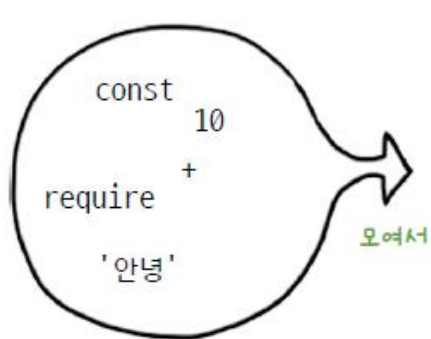
모바일 애플리케이션의 종류

- 네이티브 앱
 - 아이폰: 오브젝티브-C(Objective-C)
 - 안드로이드폰: 자바(Java) 프로그래밍
- 모바일 웹 앱
 - 웹사이트 화면을 애플리케이션으로 감싸기만 해서 보여줌
- 하이브리드 앱
 - 스마트폰의 기능과 웹 페이지를 연결할 수 있는 층을 설치해서 웹사이트가 스마트폰의 기능을 활용
 - 쿠팡, 위메프 등의 쇼핑 애플리케이션
- 리액트 네이티브
 - 하나의 프로그램을 만들어서 여러 프로그램으로 만들어주는 엔진 또는 프레임워크
 - 페이스북, 인스타그램, 핀터레스트, 디스코드, 스카이프 등

필수 기본 용어

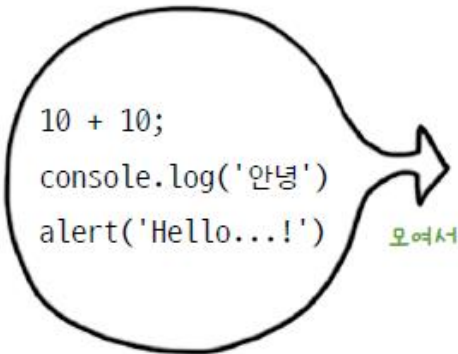
표현식

값을 만들어 내는
간단한 코드



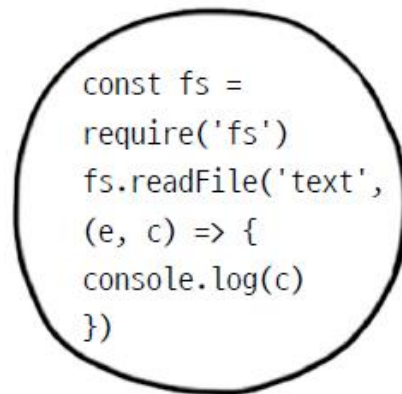
문장

표현식이 하나 이상 모인 것



프로그램

문장이 모인 것



필수 기본 용어

- 표현식: 자바스크립트에서 값을 만들어내는 간단한 코드
- 문장: 하나 이상의 표현식이 모여 문장(statement)을 구성.
- 문장 종결자 세미콜론(;) 또는 줄 바꿈
- 키워드 예약어 : 자바스크립트에 내장되어 있는 명령

await	break	case	catch
class	const	continue	debugger
default	delete	do	else
export	extends	finally	for
function	if	import	in
instanceof	new	return	super
switch	this	throw	try
typeof	var	void	while
with	yield	let	static

필수 기본 용어

- 식별자 : 프로그래밍 언어에서 **이름을 붙일 때** 사용하는 단어
 - 주로 변수명이나 함수명 등으로 사용
 - 예약 키워드는 사용 안됨, 숫자로 시작 불가
 - 특수 문자는 _와 \$만 허용
 - 공백 문자를 포함할 수 없음
- 식별자를 만드는 일반적인 관례
 - 클래스 이름은 항상 대문자로 시작
 - 변수, 인스턴스, 함수, 메소드의 이름은 항상 소문자로 시작
 - 여러 단어로 이루어진 식별자는 각 단어의 첫 글자를 대문자

Javascript 변수명 유효성 검사하는 사이트
<http://mothereff.in/js-variables>

필수 기본 용어

- 식별자의 종류

구분	단독으로 사용	다른 식별자와 사용
식별자 뒤에 괄호 없음	변수	속성
식별자 뒤에 괄호 있음	함수	메소드

필수 기본 용어

- 주석 : 프로그램 코드를 설명할 때 사용, 실행에 영향을 주지 않음
- HTML 태그 주석
 - `<!-- -->`로 문자열을 감싸 생성
- 자바스크립트 주석
 - 한 줄 : `//`
 - 여러 줄 : `/* 주석 내용 */`

Vscode 단축키

`//` : Ctrl + /

`/* */` : Shift + alt + A

```
<script>
// 주석은 코드 실행에 아무 영향을 미치지 않습니다.
/*
alert('Hello JavaScript')
alert('Hello JavaScript')
alert('Hello JavaScript ' )
*/
</script>
```

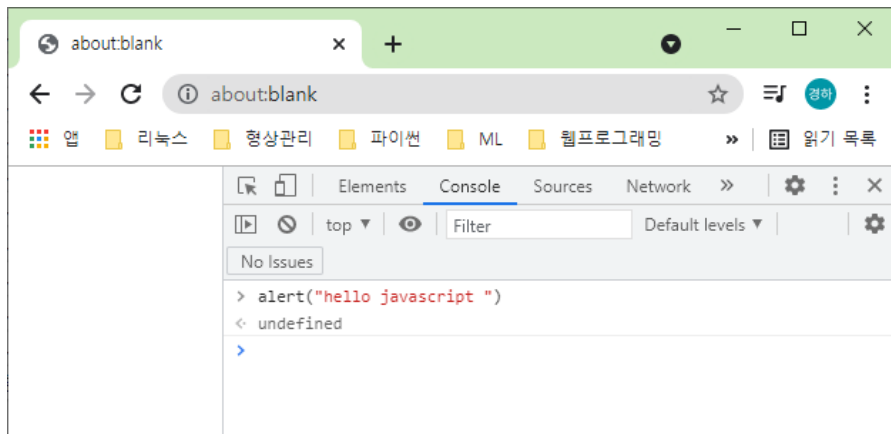
필수 기본 용어

- Javascript 실행 환경

구글 크롬 실행
url: about:blank

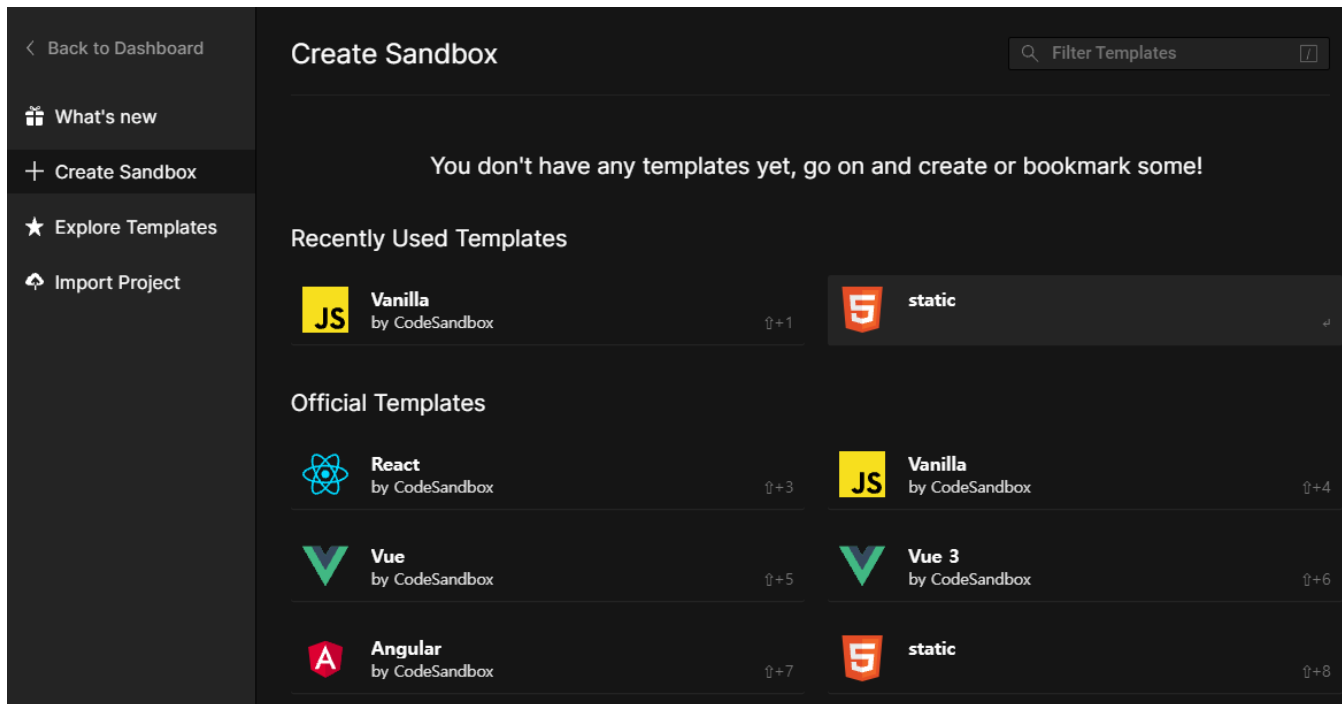
개발자도구 선택
ctrl+shift+i(F12)

Console에서
테스트



```
alert('Hello JavaScript...!')  
console.log('Hello JavaScript...!')
```


- <https://codesandbox.io/>



자료형과 변수 익히기

자료형

- 기본 자료형(Primitive values)
 - Boolean
 - Null
 - Undefined
 - Number
 - String
 - Symbol(ECMAScript6에 추가됨)
- 객체(Objects)

기본 자료형

- 자료(data): 프로그래밍에서 프로그램이 처리할 수 있는 모든 것
- 변수의 고정 타입이 없음, 값을 할당할 때 정해짐.(동적 타이핑)
- 자료형(data type): 자료 형태에 따라 나눠 놓은 것
 - 숫자(number), 문자열(string), 불(Boolean) 자료형
- 자바스크립트는 2가지 방법으로 문자열을 생성
 - 큰따옴표, 작은따옴표를 사용

```
> '안녕하세요'
```

```
"안녕하세요"
```

```
> "안녕하세요"
```

```
"안녕하세요"
```

boolean

```
// Boolean

const isTrue = true;
const isFalse = false;

console.log(isTrue, typeof isTrue);
console.log(isFalse, typeof isFalse);

const a = new Boolean(false);

console.log(a, typeof a);

if (a) {
  console.log('false?');
}

💡
const b = Boolean(false);

console.log(b, typeof b);
```

자료형

- Null

```
// Null

const a = null;

console.log(a, typeof a);

// Undefined

let b;

console.log(b, typeof b);

b = undefined;

console.log(b, typeof b);

if (a == b) {
  console.log(a == b);
}
```

- undefined

```
types > JS example3.js > ...
1 // Null
2
3 const a = null;
4
5 console.log(a, typeof a);
6
7 // Undefined
8
9 let b;
10
11 console.log(b, typeof b);
12
13 b = undefined;
14
15 console.log(b, typeof b);
16
17 if (a == b) {
18   console.log(a == b);
19 }
20
21 if (a === b) {
22   console.log(a === b);
23 }
```

자료형

- Symbol()

```
JS example1.js    JS example2.js    JS exam
types > JS example6.js > ...
1  const a = Symbol();
2  const b = Symbol(37);
3  const c = Symbol('Mark');
4  const d = Symbol('Mark');
5
6  console.log(a, typeof a);
7  console.log(c === d);
8
```

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

기본 자료형

- 특수 문자
 - ₩" : 따옴표를 문자 그대로 사용해야 할 때
 - ₩n: 줄바꿈 ₩t: 탭 ₩₩: 역슬래시(₩) 그 자체를 의미
- 문자열 연산자
 - 숫자 자료와 마찬가지로 문자열도 기호를 사용해서 연산 처리

```
> '안녕' + '하세요.'  
"안녕하세요."
```

- 문자 선택 연산자 : 문자열 내부의 문자 하나를 선택

```
> '안녕하세요'[0]  
"안"  
> '안녕하세요'[1]  
"녕"  
> '안녕하세요'[2]  
"하"
```

기본 자료형

- 문자열 자료형

- 문자열 길이 구하기

```
> "안녕하세요".length
```

```
5
```

```
> "자바스크립트".length
```

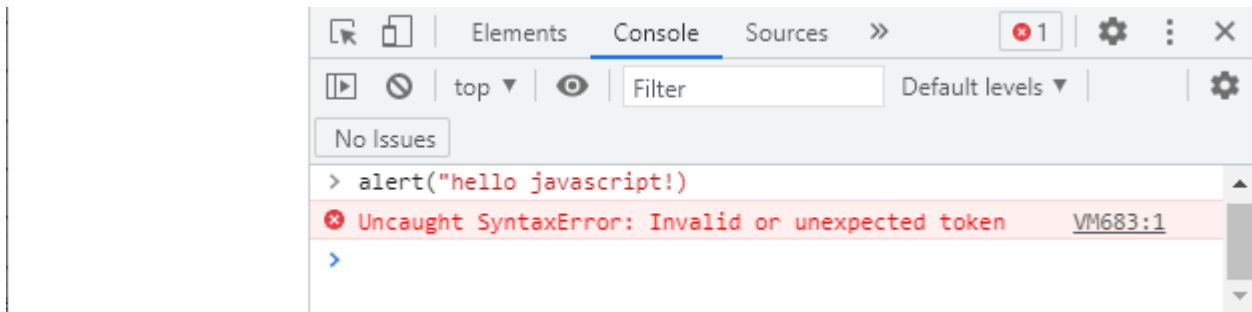
```
6
```

```
> "".length
```

```
0
```

빈 문자열도 문자열이라는 것을 기억!!!

- Uncaught SyntaxError: Invalid or unexpected token(구문 오류)



기본 자료형

- 숫자 자료형
 - 정수, 실수, NaN 모두 같은 자료형으로 인식
 - 숫자 연산자

연산자	설명
+	더하기 연산자
-	빼기 연산자
*	곱하기 연산자
/	나누기 연산자
%	나머지 연산자

```
1 // Number
2
3 const a = 37;
4
5 console.log(a, typeof a);
6
7 const b = 96.7;
8
9 console.log(b, typeof b);
10
11 const c = NaN;
12
13 console.log(c, typeof c);
14
15 const d = Number('Mark');
16
17 console.log(d, typeof d);
18
19 const e = Number('37');
20
21 console.log(e, typeof e);
22
23 |
```

기본 자료형

- Boolean 자료형
 - True(참)과 False(거짓) 값을 표현할 때 사용
- 비교 연산자

연산자	설명
==	양쪽이 같다
!=	양쪽이 다르다
>	왼쪽이 더 크다
<	오른쪽이 더 크다
>=	왼쪽이 더 크거나 같다
<=	오른쪽이 더 크거나 같다

비교연산자

- 비교 표현식 이해하기

```
01 <script>
02  if (262 < 90) {
03    alert('262는 90보다 작습니다.')
04  }
05  if (262 > 50) {
06    alert('262는 50보다 큼니다. ')
07  }
08 </script>
```

이 페이지 내용:

262는 50보다 큼니다

확인

- 논리 연산자

연산자	설명
&&	논리곱 연산자
	논리합 연산자
!	

비교연산 예)

```
if (true || false) {  
  console.log('한개만 참이면 참');  
}  
  
if (false || true) {  
  console.log('한개만 참이면 참');  
}  
  
if (false || false) {  
  console.log('두개 모두 거짓이면 거짓');  
}  
  
// !표현식  
  
if (!true) {  
  console.log('참이면 거짓');  
}  
  
if (!false) {  
  console.log('거짓이면 참');  
}
```

기본 자료형

- 자료형 검사 : typeof 연산자

```
> typeof('문자열')
```

```
"string" ← 문자열을 의미
```

```
typeof(273)
```

```
"number " ← 숫자를 의미
```

```
> typeof(true)
```

```
"boolean"
```

typeof() :

string, number, boolean, undefined, function,
object, symbol, bigint

좀 더 알아보기

- 템플릿 문자열은 백틱(`) 기호로 감싸 만들
 - 문자열 내부에 `\${...}` 기호를 사용하여 표현식을 넣으면 표현식이 문자열 안에서 계산됨

```
> console.log(`260 + 50 = ${260 + 50}입니다...!`)  
260 + 50 = 310입니다...!
```

좀 더 알아보기

- == 연산자와 != 연산자
 - '값이 같은지'를 비교하는 연산자
 - 다음 코드들은 모두 true를 출력

```
> 1 == "1"
```

```
true
```

→ 자료형은 무시하고 값만 비교함. 형변환으로 값이 같으므로 true

```
> false == "0"
```

```
true
```

→ false가 0으로, "0"이 0으로 변환된 뒤에 비교

```
> "" == []
```

```
true
```

→ 빈 문자열과 배열 []이 비어 있으므로 서로 비교

```
> 0 == []
```

```
true
```

→ 0과 [] 비어 있는 것 서로 비교

변수의 유효범위

const, let 의 유효 범위

블록 스코프

```
// 함수
function hello1() {
  const name = 'Mark';
  console.log(name);
}

// console.log(name);

// arrow 함수
const hello2 = () => {
  const age = 37;
  console.log(age);
};

// console.log(age);
```

scope > JS example1.js > ...

```
1  // 블록
2  {
3    //
4    const name = 'Mark';
5    console.log(name);
6  }
7
8  // console.log(name);
9
10 // 밖에서 안으로
11
12 let age = 37;
13
14 {
15   age++;
16   console.log(age);
17 }
18
19 console.log(i);
20
21 // 중첩
22
```

호이스팅(Hoisting)

- 아래 있는 선언을(만) 끌어올린다
- 선언만 끌어올리고 값 할당은 먼저 할당 되어야함.
- 함수, var 아래 선언한 것을 위에서 사용함.

hoisting > JS example3.js > ...

```
1 console.log(name);
2
3 name = 'Mark';
4
5 console.log(name);
6
7 var name;
8
```

hoisting > JS example3.js > [🔗] name

```
1 console.log(name);
2
3 name = 'Mark';
4
5 console.log(name);
6
7 var name = 'Woongjae';
8
```

JS example1.js X

JS example2.js

JS example3.js

hoisting > JS example1.js > 📦 hello1

```
1 // 함수 먼저
2 function hello1() {
3   console.log('hello1');
4 }
5
6 hello1();
7
8 // 함수의 호출을 먼저
9 hello2();
10
11 function hello2() {
12   console.log('hello2');
13 }
14
```

상수와 변수

- 상수 : 항상 같은 수, 값을 할당 하면 수정할 수 없음

`const 변수이름 = 값`

- 예) `pi = 3.141592` 선언

```
> const pi=3.141592
```

```
< undefined
```

```
> pi
```

```
< 3.141592
```

```
> const r = 10
```

```
< undefined
```

```
> 2*pi*r
```

```
< 62.83184
```

```
> pi = 1.2
```

```
✖ ▶ Uncaught TypeError: Assignment to constant variable.  
   at <anonymous>:1:4
```

상수와 변수

- 변수 선언 및 값 할당

let pi

pi = 30

또는

let pi = 30

```
> let pi = 3.141592
```

```
< undefined
```

```
> pi
```

```
< 3.141592
```

```
> let r = 10
```

```
< undefined
```

```
> 2*pi*r
```

```
< 62.83184
```

```
> pi*r*r
```

```
< 314.1592
```

```
> pi = 10
```

```
< 10
```

```
> 2*pi*r
```

```
< 200
```

```
>
```

변수와 상수

- ES5까지 var 사용
- ES6부터 let, const 사용
- var의 변수선언 유효 범위
- 변수 a, b의 유효 범위 확인

```
var a = 0  
function(){  
    a++  
    console.log(a)  
}
```

```
1  // var 함수 스코프  
2  
3  var a = 0;  
4  
5  (function() {  
6      a++;  
7      console.log(a);  
8  })();  
9  
10 console.log(a);  
11  
12 (function() {  
13     var b = 0;  
14     console.log(b);  
15 })();  
16  
17 b++;  
18 console.log(b);  
19
```

상수와 변수

- 변수 적용 연산자

복합 대입 연산자	설명	사용 예	의미
<code>+=</code>	기존 변수의 값에 값을 더하기	<code>a += 1</code>	<code>a = a+1</code>
<code>-=</code>	기존 변수의 값에 값을 빼기	<code>a -= 1</code>	<code>a = a-1</code>
<code>*=</code>	기존 변수의 값에 값을 곱하기	<code>a *= 1</code>	<code>a = a*1</code>
<code>/=</code>	기존 변수의 값에 값을 나누기	<code>a /= 1</code>	<code>a = a/1</code>
<code>%=</code>	기존 변수의 값에 나머지를 구하기	<code>a %= 1</code>	<code>a = a%1</code>

```
> let value = 10
undefined
> value += 10
20
> value
20
```


상수와 변수

- 변수에 적용할 수 있는 연산자

```
> let num = 10
```

```
< undefined
```

```
> num++
```

```
< 10
```

```
> num
```

```
< 11
```

```
> ++num
```

```
< 12
```

```
> num++
```

```
< 12
```

```
> num
```

```
< 13
```

증감 연산자	설명
변수++	기존의 변수 값에 1을 더하기(후위)
++변수	기존의 변수 값에 1을 더하기(전위)
변수--	기존의 변수 값에 1을 빼기(후위)
--변수	기존의 변수 값에 1을 빼기(전위)

문자열 입력

prompt(메시지 문자열, 기본 입력 문자열)

```
const msg = prompt('message', '_default')  
msg
```

```
> const msg = prompt('메시지를 남기세요.', 'hello')
```

```
< undefined
```

```
> msg
```

```
< "hello"
```

```
> |
```

이 페이지 내용:

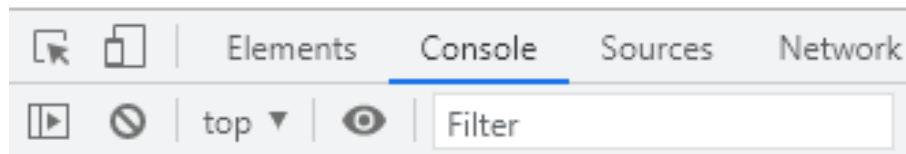
메시지를 남기세요.

확인

취소

확인 요구 메시지 창

- `confirm()` 함수
- [확인] 버튼 : `true` 리턴, [취소] 버튼 `false` 리턴



```
> const input = confirm("회원가입을 하겠습니까")
< undefined
> input
< true
```

이 페이지 내용:

회원가입을 하겠습니까

확인

취소

숫자 자료형으로 변환

- Number() 함수 사용

```
> Number("273")
```

```
273
```

```
> typeof(Number("273"))
```

```
"number"
```



자료형은 숫자

- 숫자로 변환할 수 없는 문자열인 경우, NaN(Not a Number) 값 출력

```
> Number("abc")
```

```
NaN
```

문자열 자료형 변환

- String() 함수 사용

> String(52.273) → 숫자 자료형이 문자열 자료형으로 변환
"52.273"
> String(true) → 불 자료형이 문자열 자료형으로 변환
"true"
> String(false)
"false"

- 문자열 연산자(+)와 빈 문자열("") 사용

> 273 + "" → 빈 문자열을 연결해 문자열 자료형으로 변환
"273"
> true + ""
"true"
> NaN + ""
"NaN"

Boolean 자료형으로 변환

- Boolean() 함수 사용

```
> Boolean(0)
false
> Boolean(NaN)
false
> Boolean("")
false
> Boolean(null)
false
> !null
true
> !!NaN
false
```

- 0, NaN, '...' 혹은 ""(빈 문자열), null, undefined 5개의 자료형은 false로 변환됨.
- !, !! 형 변환 가능

제어문(control statement) 조건문

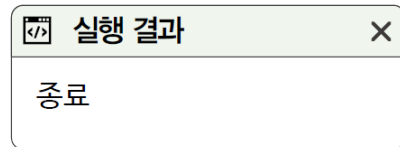
프로그램의 흐름을 변화시키는 요소.
조건문, 반복문의 종류를 알아보고 사용 방법 익히기

조건문

- if 조건문

```
if(불 값이 나오는 표현식) {  
    불 값이 참일 때 실행할 문장  
}
```

```
<script>  
    if (273 < 100) {  
  
        alert('273 < 100 => true')  
    }  
  
    alert('종료')  
</script>
```




```
8   if (false) console.log(false);
9   if (0) console.log(false);
10  if (') console.log(');
11  if (null) console.log(null);
12  if (undefined) console.log(undefined);
13  if (NaN) console.log(NaN);
14
15  // true
16  // 37
17  // -37
18  // 'Mark'
19  // {}
20  // []
21
22  if (true) console.log(true);
23  if (37) console.log(37);
24  if (-37) console.log(-37);
25  if ('Mark') console.log('Mark');
26  if ({}) console.log({});
27  if ([]) console.log([]);
28
```

if 조건문

현재 시각에 따라 오전과 오후를 구분하는 프로그램

```
> const date = new Date()   // 날짜 객체 선언
undefined
> date.getFullYear()
2020
> date.getMonth() + 1   // January is 0, February is 1
6
> date.getDate()
4
> date.getHours()
15
> date.getMinutes()
5
> date.getSeconds()
7
> date.getDay()          // Sunday is 0, Monday is 1
6
```

if 조건문 [예제 3-1]

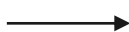
현재 시각에 따라 오전과 오후를 알려주는 HTML 문서 만들기

```
<script>
```

```
  // 객체 변수를 선언
```

```
  const date = new Date()
```

```
  const hour = date.getHours()
```



현재시간을 0~23 사이의 값으로 출력

```
  if (hour < 12) {  
    alert('오전입니다.');
```



이 페이지 내용:
오전입니다.

확인

```
  }  
  
  if (hour >= 12) {  
    alert('오후입니다.')
```

```
</script>
```

if 조건문

if else 문

```
if(불 값이 나오는 표현식) {  
    불 값이 참일 때 실행할 문장  
} else {  
    불 값이 거짓일 때 실행할 문장  
}
```

if 조건문 [예제 3-2]

if else 조건문을 사용해 현재 시간 구하기

```
<script>
  // 변수를 선언
  const date = new Date()
  const hour = date.getHours()
  if (hour < 12) {
    // 표현식 hour < 12가 참일 때 실행
    alert('오전입니다.')
  } else {
    // 표현식 hour < 12가 거짓일 때 실행
    alert('오후입니다.')
  }
</script>
```

이 페이지 내용:

오전입니다.

확인

if 조건문

중첩 조건문 : 조건문 안에 조건문을 중첩해 사용

```
if (조건 표현식 1) {  
    if (조건 표현식 2) {  
        표현식 2가 참일 때 실행할 문장  
    } else {  
        표현식 2가 거짓일 때 실행할 문장  
    }  
} else {  
    if (조건 표현식 3) {  
        표현식 3이 참일 때 실행할 문장  
    } else {  
        표현식 3이 거짓일 때 실행할 문장  
    }  
}
```

표현식 1이 참이면 실행

표현식 1이 거짓이면 실행

if 조건문 [예제 3-3]

중첩 조건문 예제

시간 < 11시

아침 먹을 시간

11 =< 시간 < 15시

점심 먹을 시간

시간 > 15시

저녁 먹을 시간

```
<script>
  const date = new Date()
  const hour = date.getHours()
  if (hour < 11) {
    // 표현식 hour < 11이 참일 때 실행
    alert( ' 아침 먹을 시간입니다. ' )
  } else {
    // 표현식 hour < 11이 거짓일 때 실행
    if (hour < 15) {
      // 표현식 hour < 15가 참일 때 실행
      alert('점심 먹을 시간입니다.')
    } else {
      // 표현식 hour < 15가 거짓일 때 실행
      alert('저녁 먹을 시간입니다.')
    }
  }
}</script>
```

if 조건문 [예제 3-4]

if else if 조건문 : 중첩 조건문에서 중괄호를 생략한 형태

```
if (불 표현식) {  
    문장  
} else if (불 표현식) {  
    문장  
} else if (불 표현식) {  
    문장  
} else {  
    문장  
}
```

```
<script>  
    const date = new Date()  
    const hour = date.getHours()  
    if (hour < 11) {  
        // 조건 값이 hour < 11이 참일 때 실행  
        alert( ' 아침 먹을 시간입니다. ' )  
    } else if (hour < 15) {  
        // 조건 값이 11<= hour < 15일 때 실행  
        alert( ' 점심 먹을 시간입니다. ' )  
    } else {  
        // 조건 값이 hour >= 15일 때 실행  
        alert('저녁 먹을 시간입니다.')    }  
</script>
```

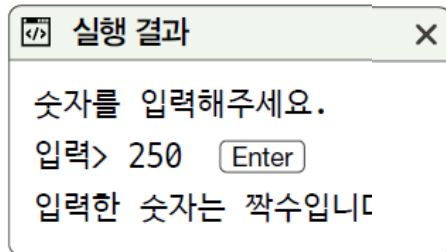

문제 [예제 3-5]

- 숫자를 입력 받아 홀수와 짝수를 구분하는 프로그램

```
<script>  
const a = Number(prompt('숫자를 입력해주세요.',  
"
```



```
</script>
```



[힌트] 홀수와 짝수를 어떻게 구분해야 할까?

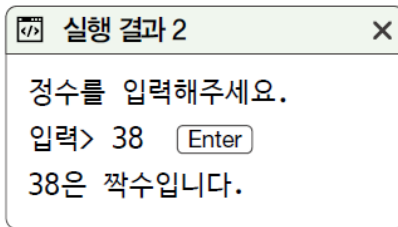
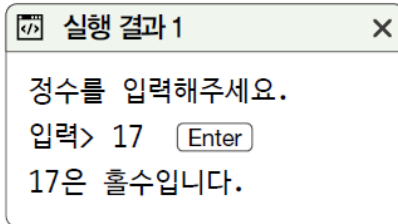
직접 만들어본 뒤에 「짝수와 홀수 구분하기 누적 예제」를 참조

문제 : 짝수, 홀수 구분기 [예제:3-5]

[문제] 짝수, 홀수 구분기 - switch문으로 구현

```
<script>
  const input = prompt('정수를 입력해주세요.');"
  const end = input[input.length - 1]
  // 끝자리를 비교
  alert(`${end}은 짝수입니다.`)
  if (end === "0" ||
      end === "2" ||
      end === "4" ||

      end === "6" ||
      end === "8") {
    alert(`${input}은 짝수입니다.`)
  } else {
    alert(`${input}은 홀수입니다.`)
  }
</script>
```



switch 조건문

기본 형태, default는 생략 가능

```
switch (자료) {  
    case 조건 A:  
        break  
    case 조건 B:  
        break  
    default:  
        break  
}
```

생략할 수 있음

switch 조건문 [예제 3-6]

if 조건문을 switch문으로 바꾸기

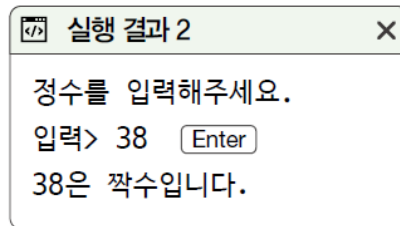
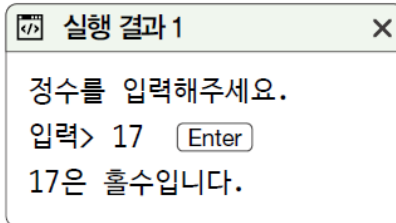
```
<script>
  const date = new Date()
  const hour = date.getHours()
  switch (true) {
    case hour < 11: // 조건이 hour < 11일 때 실행
      alert( ' 아침 먹을 시간입니다. ' )
      break
    case hour < 15: // 조건이 11 <= hour < 15일 때 실행
      alert( ' 점심 먹을 시간입니다. ' )
      break
    default: // 위의 모든 것이 거짓일 때 실행
      alert('저녁 먹을 시간입니다.')
      break
  }
</script>
```

문제 : 짝수, 홀수 구분기 [예제3-5]

[문제] 짝수, 홀수 구분기 - switch문으로 구현

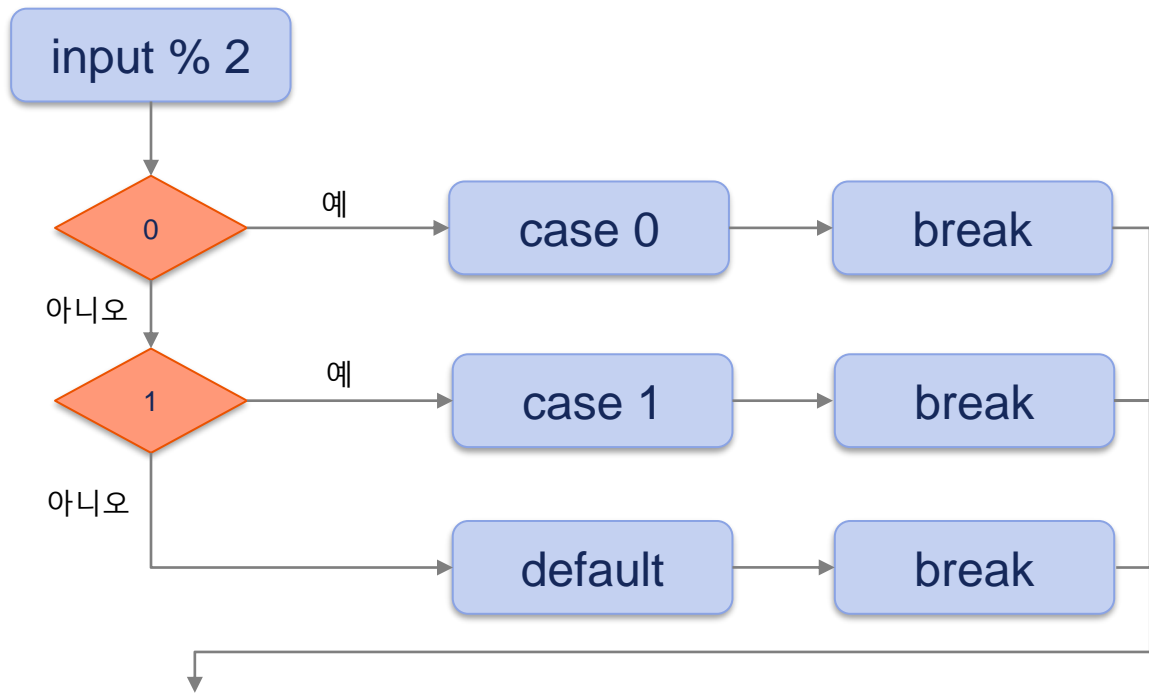
```
<script>
  const input = prompt('정수를 입력해주세요.');"
  const end = input[input.length - 1]
  // 끝자리를 비교
  alert(`${end}은 짝수입니다.`)
  if (end === "0" ||
      end === "2" ||
      end === "4" ||

      end === "6" ||
      end === "8") {
    alert(`${input}은 짝수입니다.`)
  } else {
    alert(`${input}은 홀수입니다.`)
  }
</script>
```



switch 조건문

- break : switch 조건문이나 반복문을 빠져나가기 위해 사용하는 키워드



switch 조건문 [예제 3-7]

switch 조건문 사용하기

```
<script>
  // 변수를 선언
  const input = Number(prompt('숫자를 입력하세요.', '숫자'))
  // 조건문
  switch (input % 2) {    → 나머지 연산자를 사용하여 홀수와 짝수를 구분
    case 0:
      alert('짝수입니다.')
      break
    case 1:
      alert('홀수입니다.')
      break
    default:
      alert('숫자가 아닙니다.')
      break
  }
</script>
```

조건문 [예제 3-8]

[문제] 짝수, 홀수 구분기 - if문으로 구현

```
<script>
  const input = prompt('정수를 입력해주세요.','')
  const num = Number(input)

  if (num % 2 === 0) {
    alert(`${input}은 짝수입니다.`)
  } else {
    alert(`${input}은 홀수입니다.`)
  }
</script>
```

* 주의 하기

```
(`${input}은 짝수입니다.`)
```


조건문 [예제 3-9]

3항 연산자

불 표현식 ? 참일 때의 결과 : 거짓일 때의 결과

```
<script>
```

```
// 변수를 선언
```

```
const input = prompt('숫자를 입력해주세요.', " ")
```

```
const number = Number(input)
```

```
// 조건문
```

```
const result = (number >= 0) ? '0 이상의 숫자입니다.' : '0보다 작은 숫자입니다.'
```

```
alert(result)
```

```
</script>
```

(number >= 0)이 true면 이 값 할당

(number >= 0)이 false면 이 값 할당

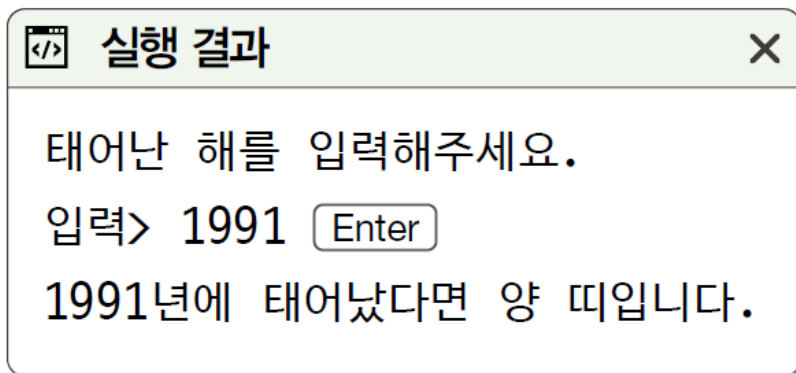
연습문제1 [예제 3-10]

- 학점을 기반으로 별명 붙여주기(누적 예제)
 - 인터넷에서 학점을 학생들이 재미있게 표현한 유머를 이를 조건문으로 구현하고 출력

조건	설명(학생 평가)
4.5	신
4.2~4.5	교수님의 사랑
3.5~4.2	현 체제의 수호자
2.8~3.5	일반인
2.3~2.8	일탈을 꿈꾸는 소시민
1.75~2.3	오락문화의 선구자
1.0~1.75	불가촉천민
0.5~1.0	자벌레
0~0.5	플랑크톤
0	시대를 앞서가는 혁명의 씨앗

연습문제2 [예제 3-11]

- 태어난 연도 입력받아 띠 출력하기
if else if 조건문 사용해보기



연습문제2 [예제 3-11-2]

- 태어난 연도를 입력받아 띠 출력하기

```
<script>
  const rawInput = prompt('태어난 해를 입력해주세요.');"
  const year = Number(rawInput)
  const tti = '원숭이,닭,개,돼지,쥐,소,호랑이,토끼,용,뱀,말,양'.split(',')
  alert(`${year}년에 태어났다면 ${tti[year % 12]} 띠입니다.`)
</script>
```

Tip.

' 문자열A '.split(', ') 문자열A를 ,로 잘라서 배열을 만들어내는 메소드.

'원숭이,닭,개,돼지,쥐,소,호랑이,토끼,용,뱀,말,양'.split(',')
=> ['원숭이', '닭', '개', '돼지', '쥐', '소', '호랑이', '토끼', '용', '뱀', '말', '양'] 배열 생성

배열 자료형 알아보기

- 여러 개의 변수를 한 번에 선언해 다룰 수 있는 자료형
- []를 사용해서 생성함
- 내부 요소는 , 로 구분함.
- [요소1, 요소2, 요소3, 요소4]

배열(array)

배열(array): 여러 자료를 묶어서 활용할 수 있는 특수한 자료

```
> const str = '안녕하세요'  
> str[2]  
> str[str.length - 1]
```

str

안	녕	하	세	요
[0]	[1]	[2]	[3]	[4]

str[2]

str[str.length-1]

str.length : 5

배열(array)

배열 만들기

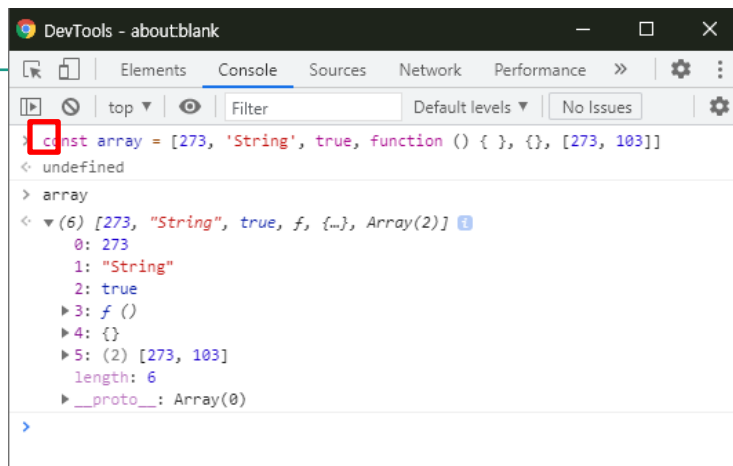
[요소, 요소, 요소, ... ,요소]

```
> const array = [273, 'String', true, function () { }, {}, [273, 103]]
```

```
undefined
```

```
> array
```

```
(6) [273, "String", true, f, {...}, Array(2)]
```



배열(array)

배열 요소에 접근하기

배열[인덱스]

인덱스는 계산식으로
표현해도 됨

```
> const numbers = [273, 52, 103, 32]
```

```
undefined
```

```
> numbers[0]
```

```
273
```

```
> numbers[1]
```

```
52
```

```
> numbers[1 + 1]
```

```
103
```

```
> numbers[1 * 3]
```

```
32
```


배열(array)

배열 요소 개수 확인하기

배열.length

```
> const fruits = [ ' 딸기 ' , ' 사과 ' , ' 포도' , '바  
  나나']
```

undefined

fruits

```
> fruits.length
```

4

딸기	사과	포도	바나나
[0]	[1]	[2]	[3]

```
> fruits[fruits.length - 1]
```

"바나나"

배열(array)

- 배열 끝에 요소 추가하기

배열.push(요소)

```
> const fruitsA = ['사과', '배', '바나나']
```

```
< Undefined
```

```
> fruitsA.push('딸기')
```

```
< 4
```

```
> fruitsA
```

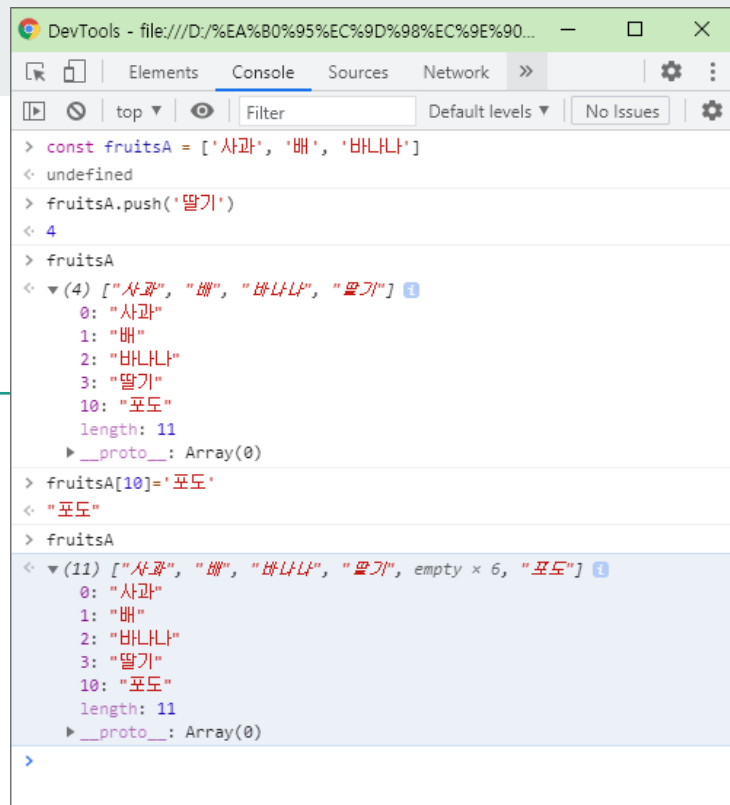
```
< (4) ["사과", "배", "바나나", "딸기"]
```

```
> fruitsA[10]='포도'
```

```
< "포도"
```

```
> fruitsA
```

```
< (11) ["사과", "배", "바나나", "딸기", empty × 6, "포도"]
```



여기서 잠깐!!

- Const로 배열을 선언 했는데 값 변경이 가능하네요?

```
const fruitsA = ['사과', '배', '바나나']
```

```
Undefined
```

```
fruitsA.push('딸기')
```

```
4
```

```
fruitsA
```

```
(4) ["사과", "배", "바나나", "딸기"]
```

```
fruitsA[10]='포도'
```

```
"포도"
```

```
fruitsA
```

```
(11) ["사과", "배", "바나나", "딸기", empty × 6, "포도"]
```

- const를 사용하더라도 **배열과 객체의 값을 변경하는 것은 가능**
- const는 값을 재할당하는 것만 불가능함.

배열(array)

- 배열 요소 제거하기

배열.splice(인덱스, 제거할 요소 개수)

```
fruitsA
fruitsA(11) ["사과","배","바나나","딸기",empty × 6,"포도"]
fruitsA.splice(10,1)
["포도"]
fruitsA
(10) ["사과", "배", "바나나", "딸기", empty × 6]
fruitsA[10] = '포도'
"포도"
fruitsA.splice(4,6)
(6) [empty × 6]
fruitsA
(5) ["사과", "배", "바나나", "딸기", "포도"]
```

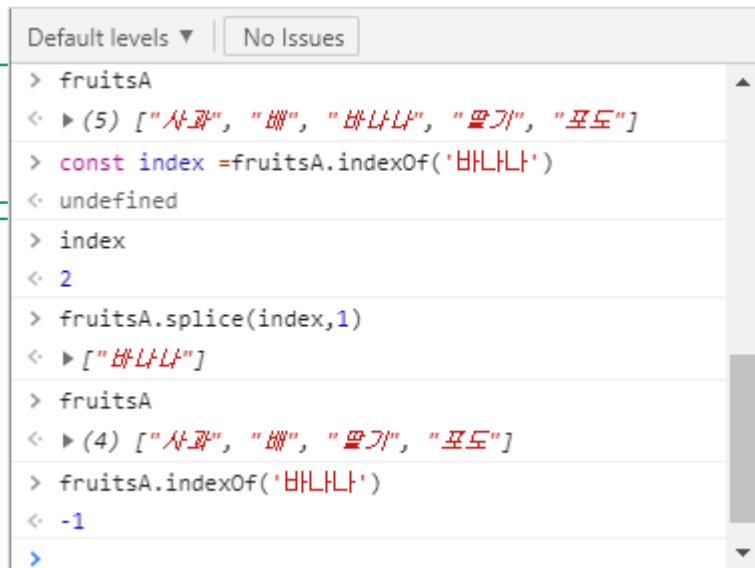


배열(array)

- 배열 요소 제거하기

```
const 인덱스 = 배열.indexOf(요소)  
배열.splice(인덱스, 1)
```

```
fruitsA  
(5) ["사과", "배", "바나나", "딸기", "포도"]  
const index = fruitsA.indexOf('바나나')  
Undefined  
Index  
2  
fruitsA.splice(index,1)  
["바나나"]  
fruitsA  
(4) ["사과", "배", "딸기", "포도"]  
fruitsA.indexOf('바나나')  
-1
```



```
Default levels ▾ No Issues  
> fruitsA  
< ▶ (5) ["사과", "배", "바나나", "딸기", "포도"]  
> const index = fruitsA.indexOf('바나나')  
< undefined  
> index  
< 2  
> fruitsA.splice(index,1)  
< ▶ ["바나나"]  
> fruitsA  
< ▶ (4) ["사과", "배", "딸기", "포도"]  
> fruitsA.indexOf('바나나')  
< -1  
>
```

배열(array)

- 배열의 특정 위치에 요소 추가하기

배열.splice(인덱스, 0 또는 1, 요소)
0 : 인덱스에 추가, 1 : 인덱스에 덮어쓰기

fruitsA

(4) ["사과", "배", "딸기", "포도"]

fruitsA.splice(1,0,'오렌지')

[]

fruitsA

(5) ["사과", "오렌지", "배", "딸기", "포도"]

fruitsA.splice(1,1,'귤')

["오렌지"]

fruitsA

(5) ["사과", "귤", "배", "딸기", "포도"]

```
Default levels ▾ | No Issues
< -1
> fruitsA
< ▶ (4) ["사과", "배", "딸기", "포도"]
> fruitsA.splice(1,0,'오렌지')
< ▶ []
> fruitsA
< ▶ (5) ["사과", "오렌지", "배", "딸기", "포도"]
> fruitsA.splice(1,1,'귤')
< ▶ ["오렌지"]
> fruitsA
< ▶ (5) ["사과", "귤", "배", "딸기", "포도"]
> |
```

더 알아보기

- 자료 처리 연산자, 함수, 메소드의 처리 구분
 - 비 파괴적 처리 수행 : 처리 후에 원본 내용이 변경되지 않음

```
> const a = '안녕'
> const b = '하세요'
> const c = a + b
> c
"안녕하세요"
> a
"안녕"
> b
"하세요"
```

메모리가 여유로운 현대의
프로그래밍 언어와 라이브러리는
자료 보호를 위해서 대부분
비파괴적 처리를 수행

더 알아보기

- 자료 처리 연산자, 함수, 메소드의 처리 구분
 - 파괴적 처리 수행 : 처리 후에 원본 내용이 변경됨

const array = ["사과", "배", "바나나"]	→	변수를 선언
array.push("귤")	→	배열 뒷부분에 요소를 추가하는 처리
4		
array		
(4) ["사과", "배", "바나나", "귤"]	→	원본(array) 내용이 변경됨

제어문(control statement) 반목문

반복문(for in) [예제 3-12]

- for in 반복문(배열 요소 반복 참조 실행)

```
for (const 반복 변수 in 배열 또는 객체) {  
  문장  
}
```

top ▾

Filter

Default levels ▾

No Issues

0번째 할일 : 출근

4-12 for in.html:11

1번째 할일 : 고객미팅

4-12 for in.html:11

2번째 할일 : js수강

4-12 for in.html:11

3번째 할일 : 탁구

4-12 for in.html:11

>

```
<script>  
  const todos = ['출근', '고객미팅', 'js수강', '탁구']  
  for (const i in todos){  
    console.log(`${i}번째 할일 : ${todos[i]}`)  
  }  
</script>
```

`\${}`

반복문(for of) [예제 3-13]

- for of 반복문

```
for (const 반복 변수 of 배열 또는 객체) {  
    문장  
}
```

Default levels ▾		No Issues
오늘 할 일 : 출근	4-13 for of.html:11	
오늘 할 일 : 고객미팅	4-13 for of.html:11	
오늘 할 일 : js수강	4-13 for of.html:11	
오늘 할 일 : 탁구	4-13 for of.html:11	
>		

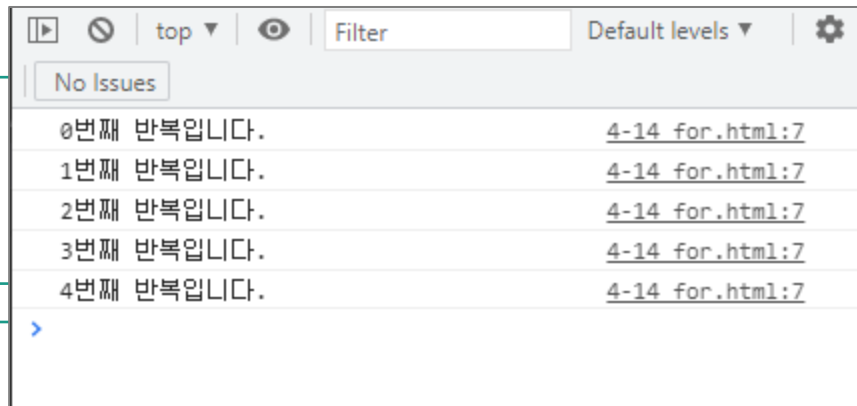
```
<script>  
  const todos = ['출근', '고객미팅', 'js수강', '탁구']  
  for (const todo of todos){  
    console.log(`오늘 할 일 : ${todo}`)  
  }  
</script>
```

반복문(for) [예제 3-14]

- for 반복문 : 범용적인 반복문

```
for (let i = 0; i < 반복 횟수; i++) {  
    문장  
}
```

```
<script>  
    for (let i = 0; i < 5; i++) {  
        console.log(`${i}번째 반복입니다.`)  
    }  
</script>
```



0번째 반복입니다.	4-14 for.html:7
1번째 반복입니다.	4-14 for.html:7
2번째 반복입니다.	4-14 for.html:7
3번째 반복입니다.	4-14 for.html:7
4번째 반복입니다.	4-14 for.html:7

let i=0; 대신 const = 0;
가능할까?

반복문(for) [예제 3-15]

- for 반복문
 - 1부터 N까지 더하기

```
01 <script>
02  let output = 0
03  for (let i = 1; i <= 100; i++) {
04    output += i
05  }
06  console.log(`1~100까지 합 : ${output}`)
07 </script>
```

반복문(for) [예제 3-16]

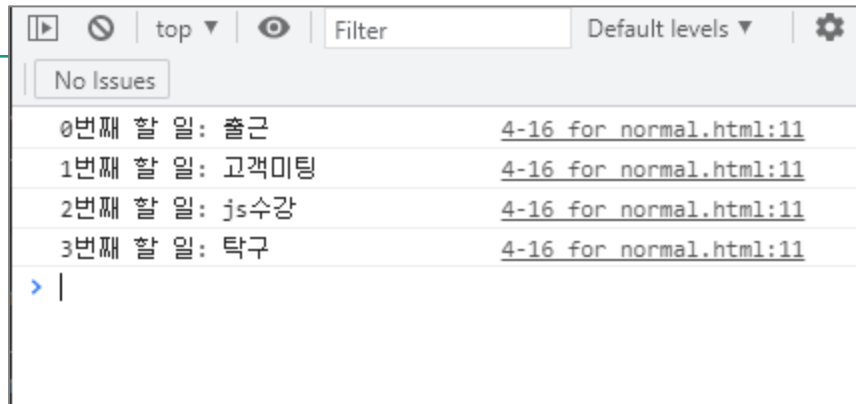
- 일반 for 반복문과 배열 사용하기

```
<script>
```

```
  const todos = ['출근', '고객미팅', 'js수강', '탁구']
```

```
  for (let i = 0; i < todos.length; i++) {  
    console.log(`${i}번째 할 일: ${todos[i]}`)  
  }
```

```
</script>
```



반복문 (while)

- while 반복문

- While 조건이 true면 계속 반복함
- 반복문에서 빠져나올 장치가 필요함(조건 false, break)
- 무한 루프에 빠지지 않도록 주의할 것

```
while (불 표현식) {  
    문장  
}
```

```
<script>  
    let i = 0  
    while (true) {  
        alert(`${i}번째 반복입니다.`)  
        i = i + 1  
    }  
</script>
```

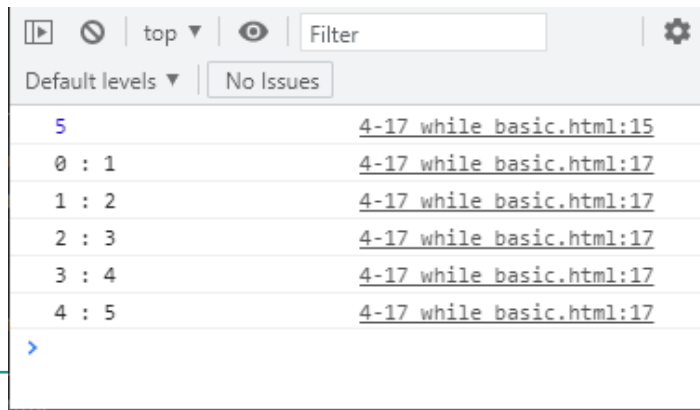
반복문 (while) [예제 3-18]

- while 반복문
 - while 반복문과 함께 배열 사용하기

```
<script>
  let i = 0
  const array = [1, 2, 3, 4, 5]
  console.log(`array.length`)
  while (i < array.length) {
    console.log(`${i} : ${array[i]}`)
    i++
  }
</script>
```

※ 반복 횟수가 정해져 있을 때는 for문으로 구현 하는게 효과적

while 반복문은 조건을 기반으로 반복할 때 효과적



top ▼		Filter	⚙
Default levels ▼		No Issues	
5	4-17 while basic.html:15		
0 : 1	4-17 while basic.html:17		
1 : 2	4-17 while basic.html:17		
2 : 3	4-17 while basic.html:17		
3 : 4	4-17 while basic.html:17		
4 : 5	4-17 while basic.html:17		
>			

반복문 (break) [예제 3-18]

- break 키워드 : switch 조건문이나 반복문을 벗어날 때 사용

```
while (true) {  
    break  
}
```

```
<script>  
    for (let i = 0; true; i++) {  
        alert(i + '번째 반복문입니다.')  
        const isContinue = confirm('계속 하시겠습니까?')  
        if (!isContinue) {  
            break  
        }  
    }  
    alert('프로그램 종료')  
</script>
```

반복문 (continue) [예제 3-19]

- continue 키워드
 - 반복 작업을 멈추고 반복문의 처음으로 프로세스 제어가 넘어 감
 - continue 키워드 활용

```
<script>
  for (let i = 0; i < 5; i++) {
    continue
    alert(i)
  }
  console.log("프로그램 종료")
</script>
```

반복문 (continue) [예제 3-19]

- continue 키워드

```
<script>
  let output = 0
  const nums = []
  for (let i = 1; i <= 10; i++) {
    if (i % 2 === 1) {
      continue
    }
    nums.push(i)
    output += i
  }
  alert(nums + " : " + output)
</script>
```

다음은 어떤 결과를
출력할까요?

반복문[예제 3-20-1]

- 중첩 반복문을 사용하는 피라미드(누적 예제)

- 중첩 반복문은 일반적으로 n-차원 처리를 할 때 사용
- 중첩 반복문 사용하기(1)

```
*  
**  
***  
****  
*****  
*****  
*****  
*****  
*****
```

①

외부의 반복문
: 줄생성(Wn)

i = 1

출력

i = 2

출력

i = 3

출력

i = 4

출력

②

내부의 반복문 : 별 생성(*)

* j = 0 → j < i(1) = 1번 반복해서 *

** j = 0 → j < i(2) = 2번 반복해서 *

*** j = 0 → j < i(3) = 3번 반복해서 *

**** j = 0 → j < i(4) = 4번 반복해서 *

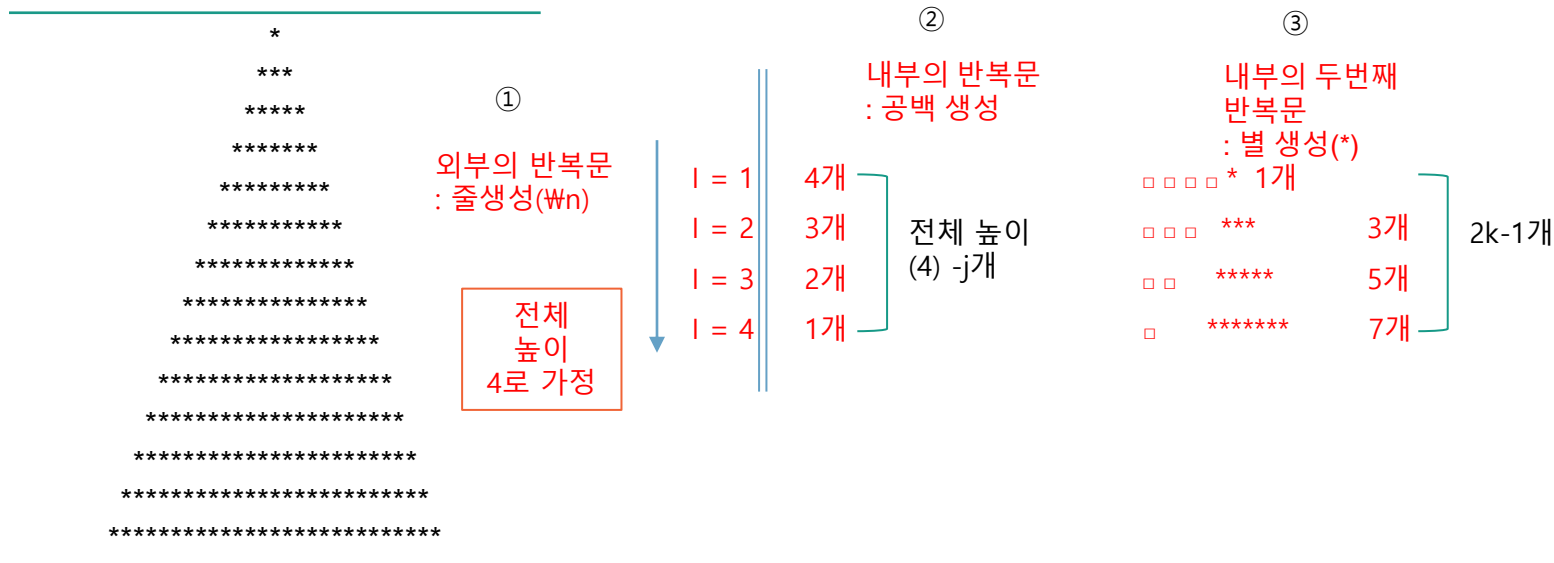
반복문 [예제 3-20-1]

- 중첩 반복문을 사용하는 피라미드(누적 예제)

```
01 <script>
02 // 변수를 선언합니다.
03 let output = ''
04
05 // 중첩 반복문
06 for (let i = 1; i < 10; i++) {
07   for (let j = 0; j < i; j++) {
08     output += '*'
09   }
10   output += '\n'
11 }
12
13 // 출력합니다.
14 console.log(output)
15 </script>
```

반복문 [예제 3-21-2]

- 중첩 반복문을 사용하는 피라미드(누적 예제)



반복문 [예제 3-21-2]

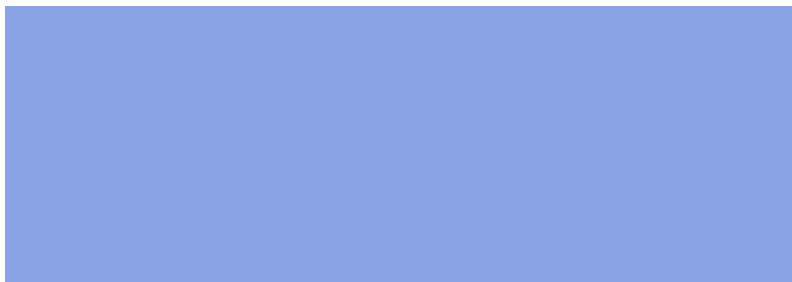
- 중첩 반복문을 사용하는 피라미드(누적 예제)

```
01 <script>
02 // 변수를 선언합니다.
03 let output = ''
04
05 // 반복문
06 for (let i = 1; i < 15; i++) {
07   for (let j = 15; j > i; j--) {
08     output += ' '
09   }
10   for (let k = 0; k < 2 * i - 1; k++) {
11     output += '*'
12   }
13   output += '\n'
14 }
15
16 // 출력합니다.
17 console.log(output)
18 </script>
```

연습문제 [예제 3-21-3]

처음에는 조금 어려울 수 있겠지만, 활용 예제의 피라미드를 활용해서
다음과 같은 피라미드를 만들어 보기

```
<script>  
  // 변수를 선언합니다.  
  let output = "  
  const size = 5  
  // 반복합니다.
```



```
  // 출력합니다.  
  console.log(output)  
</script>
```

```
  *  
  ***  
  *****  
  *****  
  *****  
  *****  
  *****  
  *****  
  ***  
  *
```


마무리

- 자바스크립트의 자료형
- 변수선언(let, const) 키워드
- for in 반복문은 배열의 인덱스를 기반으로 반복할 때 사용
- for of 반복문은 배열의 값을 기반으로 반복할 때 사용
- for 반복문은 횟수를 기반으로 반복할 때 사용
- while 반복문은 조건을 기반으로 반복할 때 사용
- break 키워드는 switch 조건문이나 반복문을 벗어날 때 사용
- continue 키워드는 반복문 안의 반복 작업을 멈추고 반복문의 처음으로 돌아가 다음 반복 작업을 진행