Seoul Software ACademy

웹 개발자 부트캠프 과정

SeSAC x CODINGOn

With. 팀 뤼쳐드

귀여운 – 형용사

미니언이 – 명사

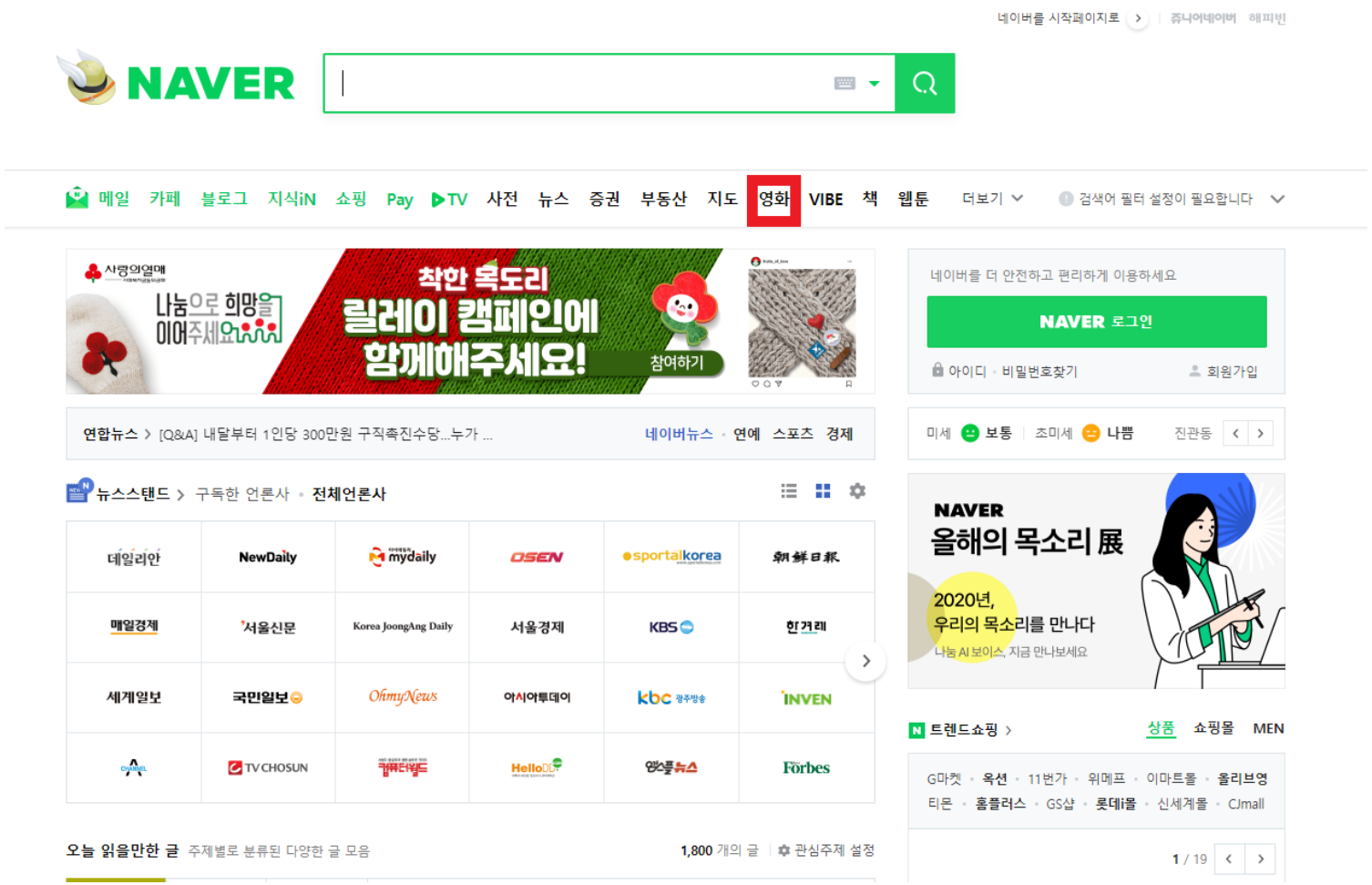
춤춘다 – 동사

JavaScript

JavaScript

웹 페이지에서 복잡한 기능을 구현할 수 있도록 하는
스크립팅 언어 또는 프로그래밍 언어 생동감!!

JavaScript 사용



동적기능

동적처리

이벤트 처리

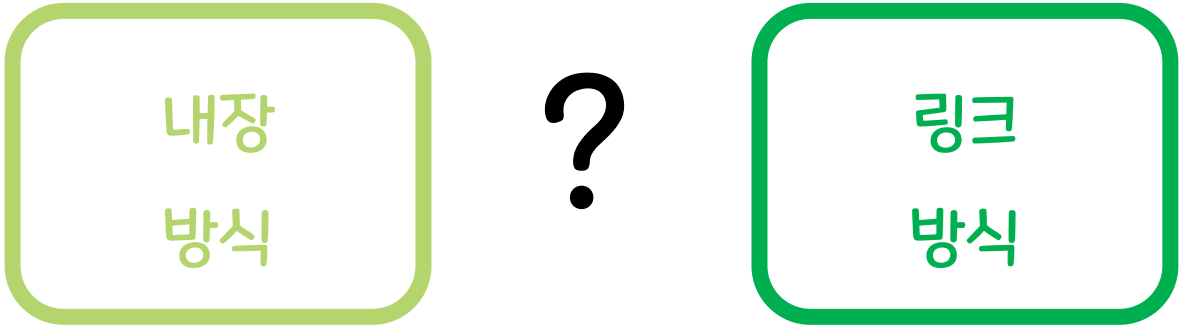
슬라이드 메뉴

...

HTML-CSS-JS

[출처] https://www.deconetwork.com/blog/skinning-your-deconetwork-website/

JavaScript 참조 방식



내장 방식!

```
<script>
    alert("헤드 그런데 js 파일 링크 위");
</script>
```

- 위치는 어디서나 사용이 가능합니다!
  - Head 태그 내부
  - Body 태그 내부
  - Head 와 body 사이
  - Body 아래 등

링크 방식!

- Java Script 파일을 따로 만들어서 링크하는 방식 like CSS

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

- 위치는 어디서나 사용이 가능합니다!
  - Head 태그 내부
  - Body 태그 내부
  - Head 와 body 사이
  - Body 아래 등

각각의 장단점이 존재 하겠죠?

- 내장 방식
  - 간단하게 만들 수 있음
  - 특정 페이지에서만 작동하는 기능일 경우 내장 방식으로만 따로 구현 가능
- 링크 방식
  - JS 코드의 양이 많아지면 파일로 관리하는 편이 편함
  - 같은 기능을 다른 페이지에서 사용하고 싶을 때 JS 파일 링크만 걸어서 사용 가능
  - 유지 보수 용이성이 편리

일단 해봅시다!

- console.log("TEXT");
  - 우리의 디버깅 친구!
  - 물론 이걸 쓰는건 안 좋은 방식입니다!
- alert("TEXT");
  - 화면에 뭐든 떠야 재미 있으니까!
  - 자매품 confirm("TEXT");
    - 얘는 취소 버튼이 있어요! → 값을 리턴한다 & IF문에 사용 가능!

읽기 순서?

- CSS의 방식 또는 선언 위치에 따라서 읽기 순서가 달라졌지요?
- JS는 어떤지 확인해 봅시다!

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        alert("헤드 그런데 js 파일 링크 위")
    </script>
    <script src="./index.js"></script>
    <script>
        alert("헤드 그런데 js 파일 링크 아래")
    </script>
</head>
<script>
    alert("헤드랑 바디 사이");
</script>
<body>
    <h1>hello, Js World!</h1>
    <script>
        alert("바디 안쪽");
    </script>
</body>
<script>
alert("바디 아래");
</script>
</html>
```

index.html

```
alert("링크방식");
```

index.js

읽기 순서!

- 말 그대로 읽히는 순서에 따라서 작동 합니다!!

잠깐!

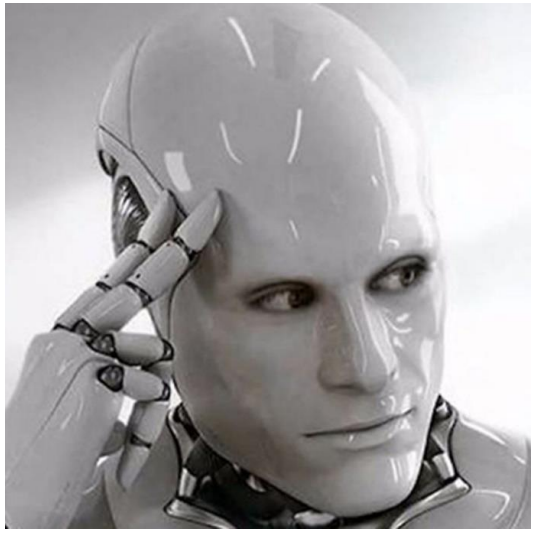
표기법

- dash-case (kebab-case)
- snake_case
- camelCase
- PascalCase

Zero-based Numbering

- 프로그래밍에서는 0 기반으로 번호를 매긴다!
- 특수한 경우를 제외하고는 **0부터 숫자를 시작**함

1 2 3 4 5 6 7 8 9 10 ~



0 1 2 3 4 5 6 7 8 9 ~

```
let fruits = ['Apple', 'Banana', 'Cherry'];

console.log(fruits[0]); // 'Apple'
console.log(fruits[1]); // 'Banana'
console.log(fruits[2]); // 'Cherry'
```

JavaScript 주석

```
// 한 줄 메모
/* 한 줄 메모 */

/**
 * 여러 줄
 * 메모1
 * 메모2
 */
```

JavaScript 변수

JavaScript 변수

JavaScript 변수

1) 선언

```JavaScript
let 변수이름;
var 변수이름;
```

2) 할당

```
const 변수이름 = 값;
let 변수이름 = 값;
var 변수이름 = 값;
```

var & let & const

```
var 변수이름;
```

1) 선언 단계와 초기화가 동시에 이루어지며 아무것도 할당하지 않으면 자동으로 **undefined** 가 할당

2) 중복 선언 가능. 재선언 가능

var & let & const

```
let 변수이름;
```

1) 변수 **중복 선언이 불가능**하지만, **재할당 가능**

2) var 과 마찬가지로 선언을 하지 않으면 자동으로 undefined가 들어간다.

var & let & const

```
const 변수이름 = 값;
```

1) 초반에 선언할 때 반드시 **초기화를 동시에 진행**해야 한다.

2) 재선언 불가능, 재할당 불가능

JavaScript 자료형

언어 타입

강한 타입 언어

타입 검사를 통과하지 못한다면 실행 자체가 안 된다.

String, int, double 등처럼 타입을 1종류로 명확히 지정

약한 타입 언어

런타임에서 타입 오류를 만나더라도 실행을 막지 않는다.

타입이 여러 종류인 값들이 상관없이 지정된다.

언어 타입

강한 타입 언어 (Strong)	약한 타입 언어 (Weak)
Java, C, C++, C#	Javascript, Python .....

JavaScript 는 느슨한 언어!

- Javascript는 데이터 종류와 관계 없이 var, let, const 키워드로 변수를 선언하고 사용함! => 약한 타입 언어

- 강한 타입 언어들은 변수를 선언할 때 명확하게 타입을 1종류만 지정함!
  - ex. JAVA, C, C++

JavaScript 자료형

Primitive 자료형

Object 자료형

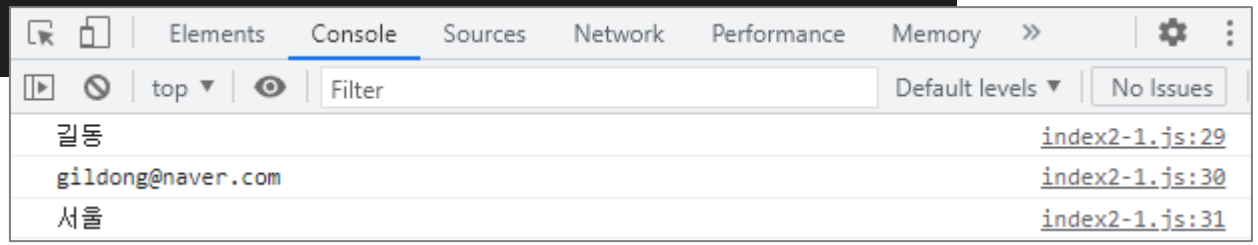
자료형

- 기본형 (Primitive)
  - string
  - number
  - boolean
  - null
  - undefined
- 객체 (Object)
  - 기본형이 아닌 것은 모두 객체

문자열 String

```
let myName = '길동';
var email = 'gildong@naver.com';
let city = '서울';
console.log(myName);
console.log(email);
console.log(city);
```

String 문자형 데이터
따옴표를 사용



문자 + 변수를 동시에 쓰고 싶을 때!

- 메소드의 매개 변수로 넣어서 사용
  - console.log(“문자”, 변수, “문자”);

- + 연산자를 사용해서 변수를 문자로 변환 후 더하여 사용
  - console.log(“문자” + 변수 + “문자”);

- 백틱 문자 사용
  - `문자를 쓰다가 변수를 쓰고 싶으면 ${variable} 처럼 쓰면 됩니다`

숫자 Number

```
// Number 숫자형 데이터
// 정수 및 소수점 숫자를 나타냄
let number = 123;
let opacity = 0.7;

console.log(number);
console.log(opacity);
```

참 거짓 데이터 Boolean

```
// Boolean 참, 거짓 데이터
// true, false 두 가지 값만 가지는 데이터
let checked = true;
let isShow = false;

console.log(checked);
console.log(isShow);
```

```
true                                                    index2-1.js:69
false                                                   index2-1.js:70
```

미할당 데이터 undefined

```
// Undifined
// 값이 할당되지 않은 상태를 표기
let undef;
let obj = {
  abc: 123,
};

console.log(undef);
console.log(obj.abc);
console.log(obj.efg);
```

```
undefined                                   index2-1.js:97
123                                         index2-1.js:98
undefined                                   index2-1.js:99
```

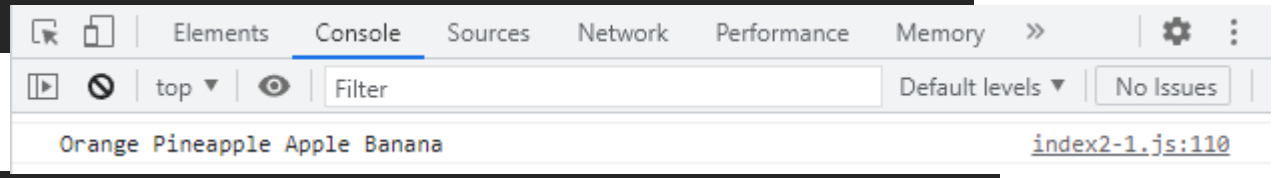
빈 데이터 null

```
// Null
// 어떤 값이 "의도적"으로 비어 있음을 의미할 때 사용
let empty = null;

console.log(empty);
```

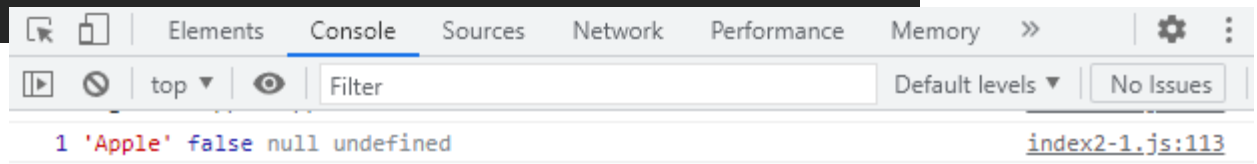
```
null
```

배열 Array

```
// Array 배열 데이터
// 여러 데이터를 순차적으로 저장
let fruits = ["Orange", "Pineapple", "Apple", "Banana"];

console.log(fruits[0], fruits[1], fruits[2], fruits[3]);
```




```
let data = [1, "Apple", false, null, undefined];

console.log(data[0], data[1], data[2], data[3], data[4]);
```



데이터 꾸러미 Object

```
let cat = {
  name: '나비',
  age: 1,
  isCute : true,
  mew: function () {
    return '냐옹';
  },
};
```

```
console.log(cat);
console.log(cat.name);
console.log(cat.age);
console.log(cat.mew());
```

typeof

자료형을 알려주는 typeof

- 해당 자료형이 어떤 것인지 알려주는 착한 친구!

```
console.log(typeof  " 안뇽");
console.log(typeof 3);
console.log(typeof null);
```

```
string
number
object
```

자료형 확인하기

- typeof()

```
typeof('문자')
typeof(245)
typeof(function() { } );
typeof(true)
typeof({})
typeof([])
typeof(NaN)
typeof(null)
```

형변환

성적 구하는 프로그램 만들기

- 그런데 결과값이 좀 이상하죠?
- Prompt 로 입력 받은 값은 "문자"로 저장이 됩니다!
  - "80" + "50" = "8050"
  - "8050" / 2 → 4025

```
let mathScore = prompt("수학 점수를 입력 하세요");
let engScore = prompt("영어 점수를 입력 하세요");

let avg = (mathScore + engScore) / 2;

console.log(avg);
```

JS의 자동 형변환!

- 처음에는 편할 수도 있지만 큰 문제를 일으키게 됩니다.
- 지금은 작은 프로그램이라 문제가 없었지만, 프로그램이 더 커진다면 의도하지 않은 중대한 문제를 일으킬 수도 있습니다.
  - 만약, 비트코인 거래 사이트라면?

명시적 형변환

- 자동 형변환에 의존 하지 않고 개발자가 직접 형 변환을 시키는 것!
- 문자로 변환 → String();
- 숫자로 변환 → Number();

```
// 문자 데이터로 변환
let num = 123;
num = String(num);
console.log(typeof num);

// 숫자 데이터로 변환
num = Number(num);
console.log(typeof num);

console.log(Number("abcdefg"));
```

```
string
number
NaN
```

형 변환 - 문자열로

1. String()

```
String(true);
```

2. toString()

```
a.toString();
(false).toString();
```

형 변환 - 정수로

1. Number()

```
Number(true)    // 1
Number('10')
```

2. parseInt()

```
parseInt('10', 10);
```

Javascript 연산자

연산자

- 대입 연산자: =
- 비교 연산자: ==, !=, ===, !===, >, >=, <, <=
- 산술 연산자: +, -, *, /, %(나머지), **(거듭제곱)
- 논리 연산자: !(not), &&(and), ||(or)

연산자 줄여쓰기

- num = num + 5 → num += 5
- num = num - 5 → num -= 5
- num = num * 5 → num *= 5
- num = num / 5 → num /= 5

증가, 감소 연산자

- ++
- --

```
let result1, result2;
let num = 10, num2 = 10;

result = num++;
console.log(result);

result2 = ++num2;
console.log(result2);
```

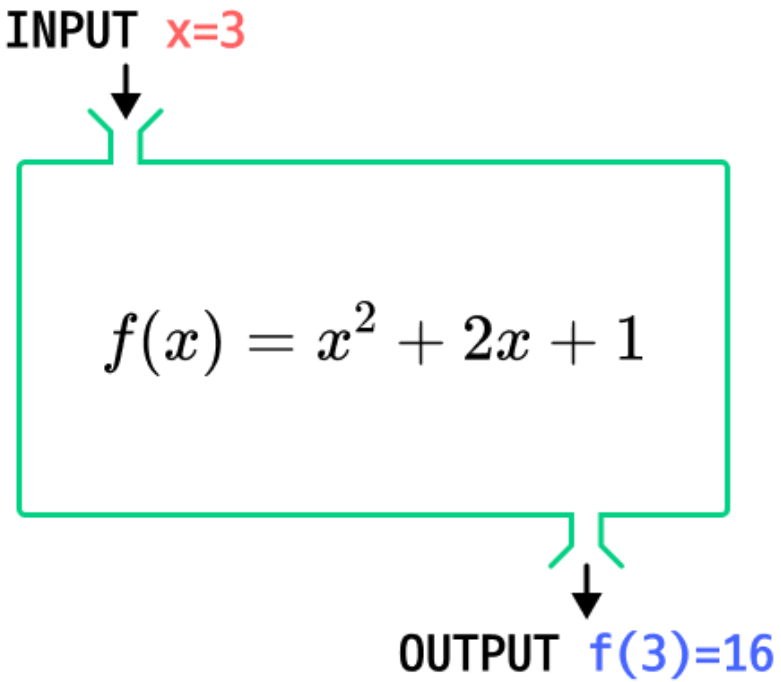
```
10
11
```

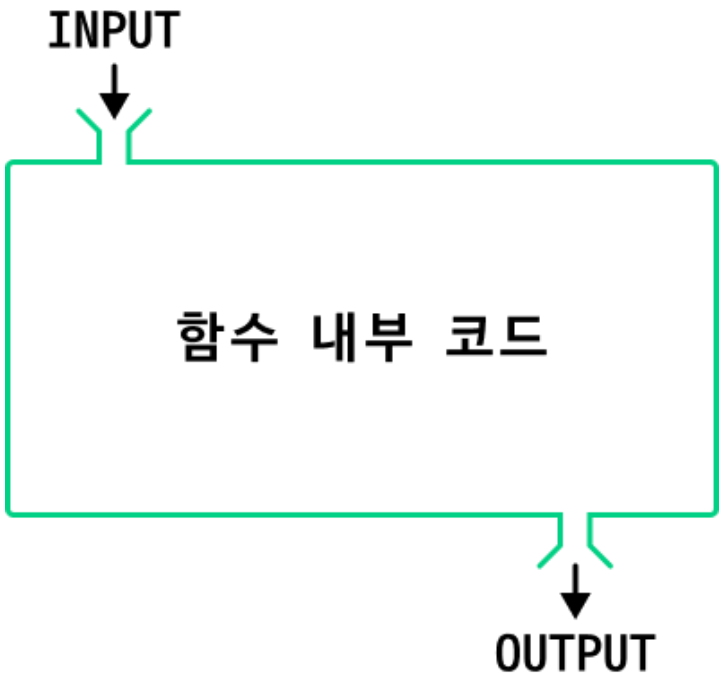
논리 연산자

- || (or) : 여러 개 중 하나라도 true → true
- && (and) : 모든 값이 true → true
- ! (not) : true → false, false → true

JavaScript 함수

함수





JavaScript 함수

```
function hello() { }
```

특정 작업을 수행하기 위해 독립적으로 설계된 코드 집합!

JavaScript 함수

```
function hello() { }
```

→ Keyword

JavaScript 함수

```
function hello() { }
```

→ **Name (함수 이름)**

: 일반적으로 camelCase 이용

JavaScript 함수

```
function hello() { }
```

→ **Parameter**

: 함수를 선언할 때 **매개변수(인자)**를 받을 것

JavaScript 함수

```
function hello() { }
```

→ **Body (Block)**

: 함수가 실행되는 Scope 라고도 한다.

함수 선언 방식

명시적 함수 선언

```
function hello() { ... }
```

```
// 함수 선언문
function sayHello(name) {
    console.log(`Hello, ${name}`);
}
```

함수 표현식

```
const hello = function () { ... }
```

```
// 함수 표현식
let sayHello = function (name) {
    console.log(`Hello, ${name}`);
}

// 함수 표현식 (화살표 함수 사용)
let sayHello = (name) => {
    console.log(`Hello, ${name}`);
}
```

함수 형태

// 1. 매개변수 X, 반환값 X

```
function sayHello() {
    console.log(`Hello`);
}
```

// 2. 매개변수 X, 반환값 O

```
let sayHello = () => {
    return `Hello`
}
```

// 3. 매개변수 O, 반환값 X

```
let sayHello = function (name) {
    console.log(`Hello, ${name}`);
}
```

// 4. 매개변수 O, 반환값 O

```
let sayHello = function (name) {
    return `Hello, ${name}`
}
```