K-Digital Training 웹 풀스택 과정

JavaScript


JS

귀여운 – 형용사

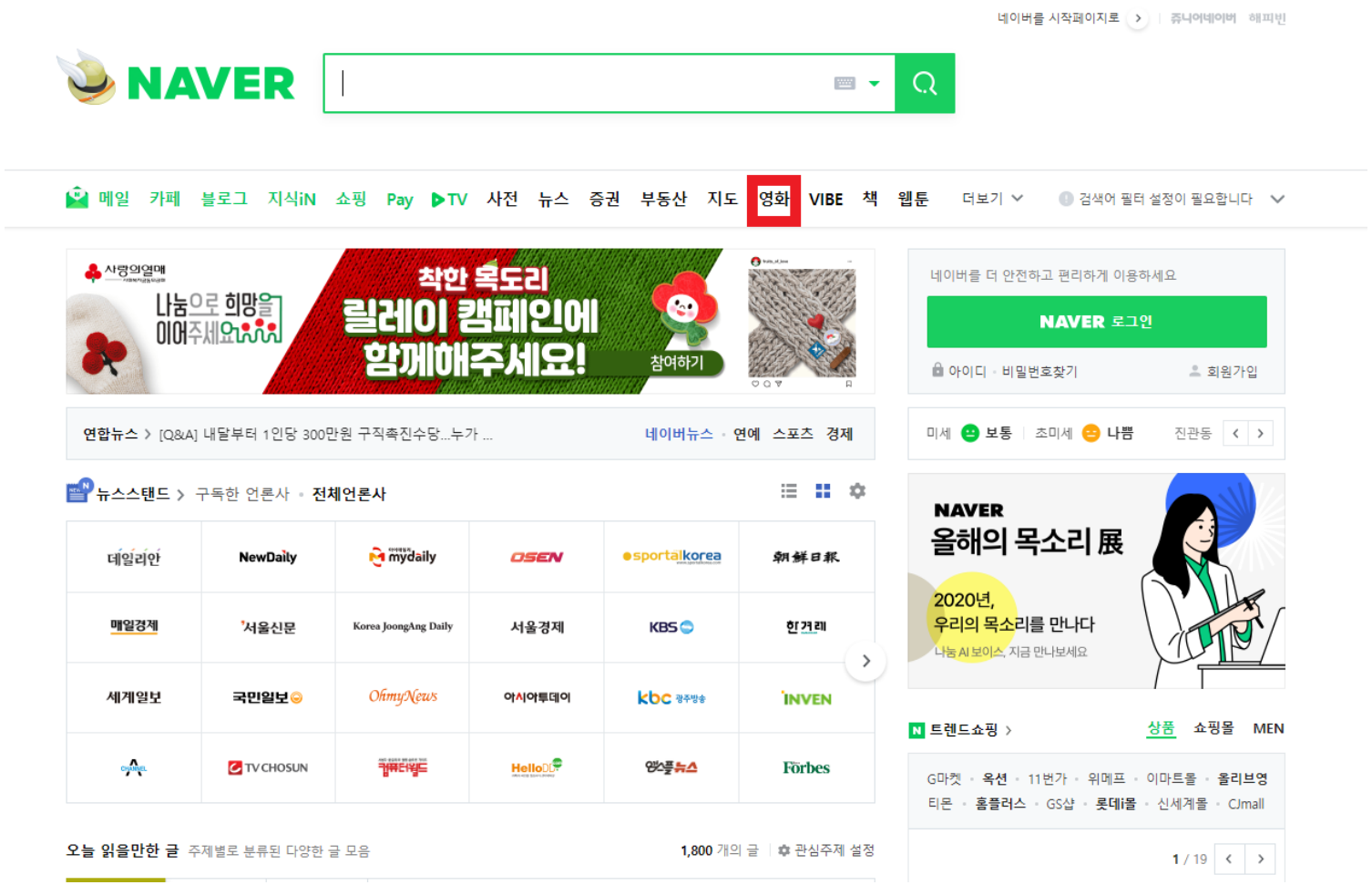
미니언이 – 명사

춤춘다 – 동사

생동감

JavaScript

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



동적기능

동적처리

이벤트 처리

슬라이드 메뉴

...

HTML-CSS-JS

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

Javascript 사용법

```
<script>
    // Javascript 코드 작성
</script>
```

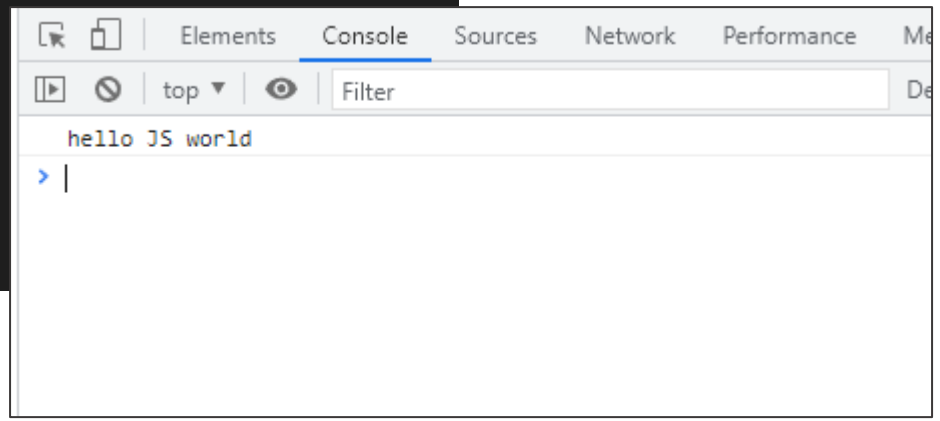
Javascript 사용해보기

- **console.log()**
  - 브라우저의 개발자 도구 > 콘솔 에서 확인 가능

- **alert()**
  - 브라우저가 열렸을 때, 그 내용을 알림창으로 보여줌

```html
<!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>js start</title>
    <script>
      console.log('hello JS world');
    </script>
  </head>
  <body>


  </body>
</html>
```



내장
방식

&

링크
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

읽기 순서?

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

Index.js

```
alert(“링크방식");
```

```
<!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

읽기 순서!

- 말 그대로 읽히는 순서에 따라서 작동 합니다!!

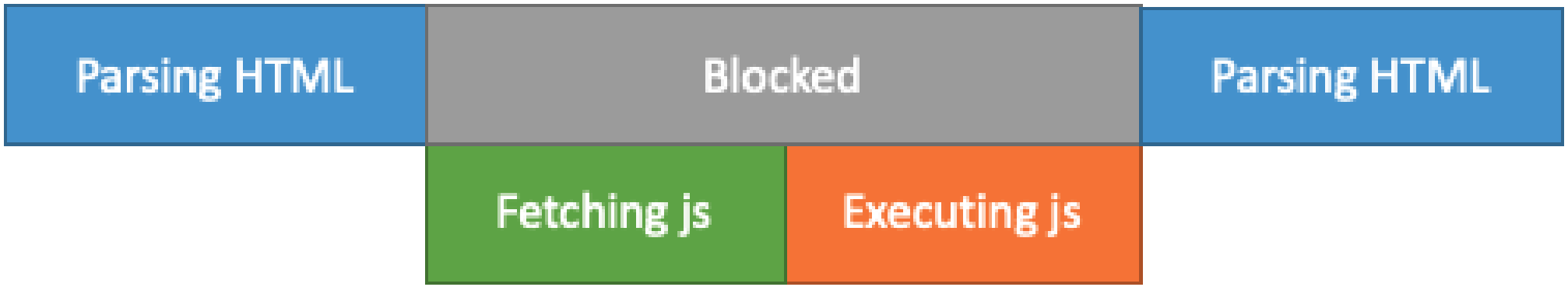
<script>의 위치는 어디가 좋을까?

<head>

?

<body>

<head> 에 포함될 경우


Parsing HTML
Blocked
Parsing HTML
Fetching js
Executing js

〈body〉 에 포함될 경우

JS 기초!

표기법

dash-case(kebab-case)

snake_case

camelCase

ParcelCase

thequickbrownfoxjumpsoverthelazydog

HTML CSS

dash-case(kebab-case)

the-quick-brown-fox-jumps-over-the-lazy-dog

snake_case

the_quick_brown_fox_jumps_over_the_lazy_dog

JS

camelCase

theQuickBrownFoxJumpsOverTheLazyDog

PascalCase

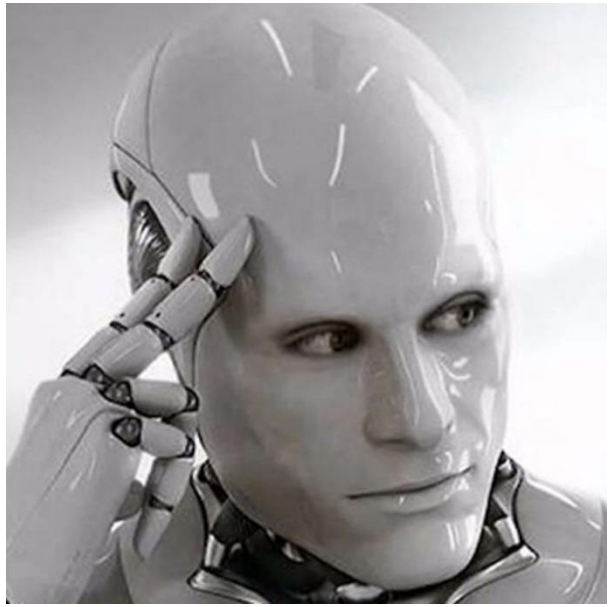
TheQuickBrownFoxJumpsOverTheLazyDog

Zero-based Numbering

0 기반 번호 매기기!

특수한 경우를 제외하고 0부터 숫자를 시작합니다.

1 2 3 4 5 6 7 8 9 10 ~



0 1 2 3 4 5 6 7 8 9 ~

주석

Comments

```
// 한 줄 메모
/* 한 줄 메모 */

/**
 * 여러 줄
 * 메모1
 * 메모2
 */
```

JavaScript

변수

변수

데이터를 저장하고 참조(사용)하는 데이터의 이름

var, let, const

변수 선언

1) 선언

```JavaScript
let 변수이름;
var 변수이름;
```

2) 할당

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

var & let & const

```
var 변수이름;
```

```
// var
var name = "홍길동";
var name = "나비";

console.log(name);
```

나비

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

2) 중복 선언 가능. 재선언 가능

var 의 문제점

- 중간의 같은 이름의 변수를 다시 선언해도 기존의 변수에 덮어 씌움
- 변수를 선언 했다는 건 분명이 다른 데이터를 넣으려는 것인데, 그것을 기존의 데이터에 덮어 씌우면!? → 문제 발생!
- 그리고 변수가 {블록 단위} 에서 끝나는 것이 아니라, 자기 맘대로 전역으로 돌아다니고 영향력을 행사함 → 의도치 않은 문제 발생!
- 따라서 ES6 문법 부터는 var 대신 let 사용을 권장

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

```
// let
let name = "홍길동";
let name = "나비";

console.log(name);
```

```
Uncaught SyntaxError: Identifier 'name' has already been declared (at index.js:78:5)
```
index.js:78

var & let & const

```
const 변수이름 = 값;
```

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

2) 재선언 불가능, 재할당 불가능

```
// const
const name = "홍길동";
const name = "나비";

console.log(name);
```

```
Uncaught SyntaxError: Identifier 'name' has already been declared (at index.js:78:5)
```
index.js:78

변수 기본 규칙 1

- 변수 이름으로는 문자 / 숫자 / $ / _ 만 사용 가능

```
// 변수 이름으로는 문자 / 숫자 / $ / _ 만 사용 가능

let myName =  " 유진형";       // O
let my$ = "null";              // O
let my_dream= “rich";         // O
let my-house = undefined;     // X
let my* = "KTH"               // X
```

변수 기본 규칙 2

- 첫 글자는 숫자가 될 수 없어요

```
// 첫 글자는 숫자가 될 수 없어요

let 1stName = "진형"          // X
Let firstName = "진형"       // O
```

변수 기본 규칙 3

- 예약어도 사용할 수 없어요

```
// 예약어도 사용할 수 없어요

let let = "let me use this!";
let if = "if i can use this...";
```

```
let this = 'Hello!'; // SyntaxError
let if = 123; // SyntaxError
let break = true; // SyntaxError
```

```
Uncaught SyntaxError: let is disallowed as a lexically bound name (at dom.js: dom.js:15
15:5)
```

예약어

특별한 의미를 가지고 있어, 변수나 함수 이름 등으로 사용할 수 없는 단어

Reserved Word

break, case, catch, continue, default, delete, do, else, false, fina
lly, for, function, if, in, instanceof, new, null, return, switch, this,
throw, true, try, typeof, var, void, while, with,
abstract, boolean, byte, char, class, const, debugger, double, e
num, export, extends, final, float, goto, implements, import, int,
interface, long, native, package, private, protected, public, shor
t, static, super, synchronized, throws, transient, volatile, as, is,
namespace, use, arguments, Array, Boolean, Date, decodeURI,
decodeURIComponent, encodeURI, Error, escape, eval,
EvalError, Function, Infinity, isFinite, isNaN, Math, NaN,
Number, Object, parseFloat, parseInt, RangeError,
ReferenceError, RegExp, String, SyntaxError, TypeError,
undefined, unescape, URIError ...

변수 기본 규칙 4

- 변수 이름은 읽기 쉽도록 센스 있게!

```
// 변수 이름은 읽기 쉽도록 센스 있게!
let a = 1;
let b = “allie";

let userNumber = 1;
let userName = “allie";
```

변수 기본 규칙 5

- 상수는 대문자로 선언해서 다른 개발자도 알 수 있게 해주세요

```
// 상수는 대문자로 선언해서 다른 개발자도 알 수 있게 해주세요

const WIDTH = 1100;
const LOL_TIER = "silver";
```

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 .....

Javascript는 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++

데이터 종류(자료형)

String

Number

Boolean

Undefined

Null

Object

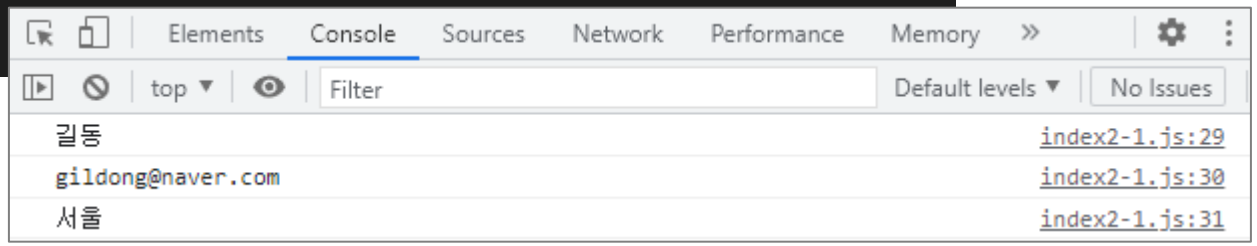
Array

String

문자형 데이터

```
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



문자와

변수를 동시에!

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`

Number

숫자형 데이터

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

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

Boolean

참, 거짓 데이터

```
// 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
```

Undefined

미할당 데이터

```
// 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
```

Null

의도된 빈 데이터

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

console.log(empty);
```

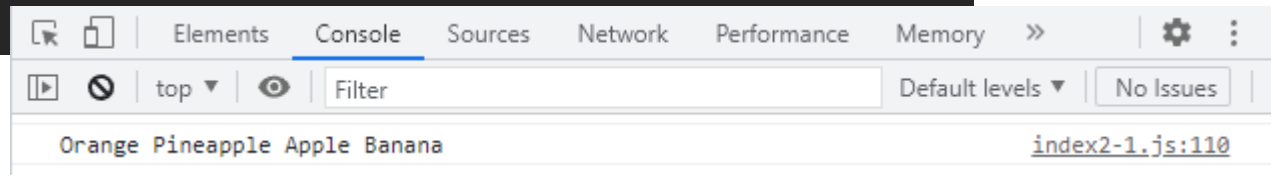
```
null
```

Array

배열 데이터

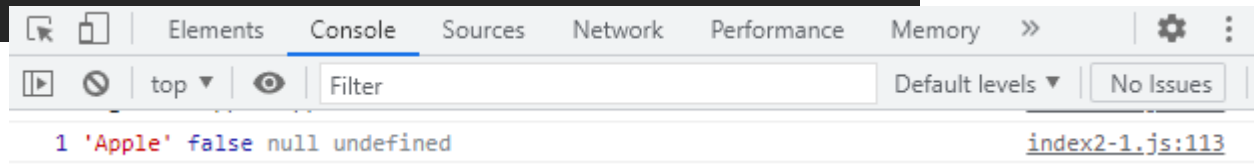
```
// Array 배열 데이터
// 여러 데이터를 순차적으로 저장
let fruits = ["Orange", "Pineapple", "Apple", "Banana"];

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]);
```



배열에서 사용 가능한 속성, 함수

- 변수명.length

- 변수명.push(추가할 값)
- 변수명.pop()
- 변수명.unshift(추가할 값)
- 변수명.shift()
- 변수명.indexOf(찾을 값)
- 변수명.includes(찾을 값)

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

Object

여러 데이터 꾸러미

```
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 친구입니다!

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

```
string
number
object
```

형 변환

성적을 구하는 프로그램 만들기

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

let avg = (mathScore + engScore) / 2;

console.log(avg);
```

성적을 구하는 프로그램 만들기

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

let avg = (mathScore + engScore) / 2;

console.log(avg);
```

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

JS의 자동 형변환!

- 처음에는 편할 수도 있지만 큰 문제를 일으키게 됩니다.
- 방금도 Error 가 떴으면 바로 문제를 수정 했겠지만, Error 가 뜨지 않고 프로그램이 구동이 되었죠!
- 지금은 작은 프로그램이라 문제가 없었지만, 프로그램이 더 커진다면 의도하지 않았지만 정말 중대한 문제를 일으킬 수도 있습니다.
  - 만약, 비트코인 거래 사이트라면?

문자형변환

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

```
// 문자 데이터로 변환
let num = 123;

let a = String(num);
console.log(typeof a);


let b = num.toString();
console.log(typeof b);
```

```
string
string
```

String() 과 toString() 차이

- String()
  - null 과 undefined 에도 문제 X
- toString()
  - null과 undefined 에서 문제 발생

```
let a = null;

let b = String(a);
console.log(typeof b);

let c = a.toString();
console.log(typeof c);
```

```
string
Uncaught TypeError: Cannot read properties of null (reading 'toString')
    at type.html:13:17
```

숫자형변환

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

```
// 숫자 데이터로 변환
console.log(Number("123"));


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

```
123
NaN
```

기본 연산자

연산자

- 대입 연산자: =
- 비교 연산자: ==, !=, ===, !===, >, >=, <, <=
- 산술 연산자: +, -, *, /
- 논리 연산자: !, &&, ||

기본 연산자

- % 나머지 연산자
  - 홀수 판단 : num % 2 == 1 이면 홀수
  - 짝수 판단 : num % 2 == 0 이면 짝수
- ** 거듭 제곱
  - ** 를 사용
  - 2 ** 3 = 8
  - 3 ** 3 = 27

연산자 줄여서 쓰기

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

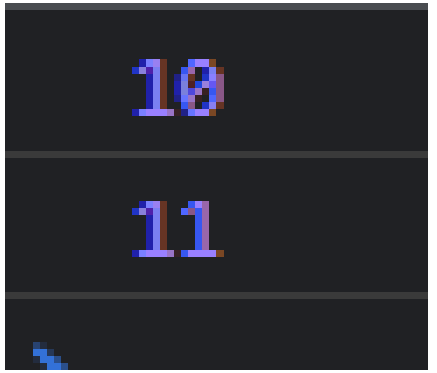
증가, 감소 연산자

- **Num++;**
- **Num--;**

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

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

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



비교 연산자

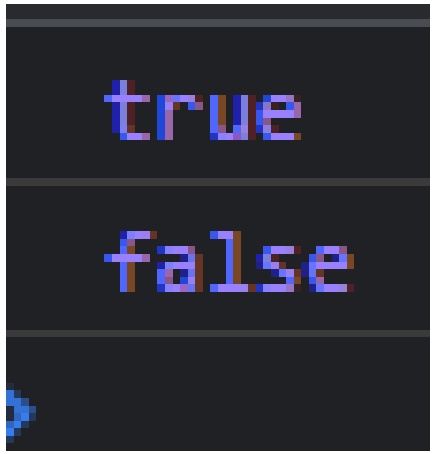
일치 연산자(===)

- **변수의 값 뿐만 아니라 자료형 까지도 비교!**

```
let a = 1;
let b = "1";

// 비교 연산자
console.log(a == b);

// 일치 연산자
console.log(a === b);
```



논리 연산자

논리 연산자

|| (OR)

&& (AND)

! (NOT)

OR 연산자, ||

```
// OR 연산자
// 이름이 뽀로로 이거나, 성인이면 통과

let name = "뽀로로";
let age = 18;

if(name === "뽀로로" || age > 19) {
    console.log("통과");
} else {
    console.log("돌아가");
}
```

```
통과
```

AND 연산자, &&

```
// AND 연산자
// 이름이 뽀로로 이고, 성인이면 통과

let name = "뽀로로";
let age = 18;

if(name === "뽀로로" && age > 19) {
    console.log("통과");
} else {
    console.log("돌아가");
}
```

```
돌아가
```

NOT 연산자, !

```
// NOT 연산자
let age = 16;
let isAdult = age > 19;

if(!isAdult) {
    console.log("돌아가");
} else {
    console.log("통과");
}
```