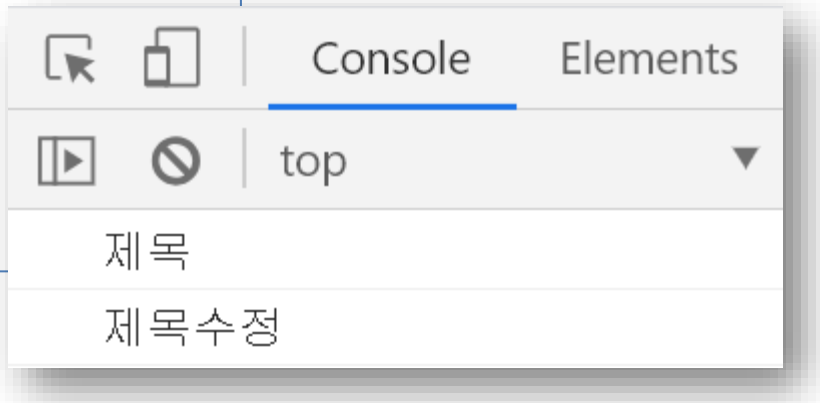
■ Variable (변수)

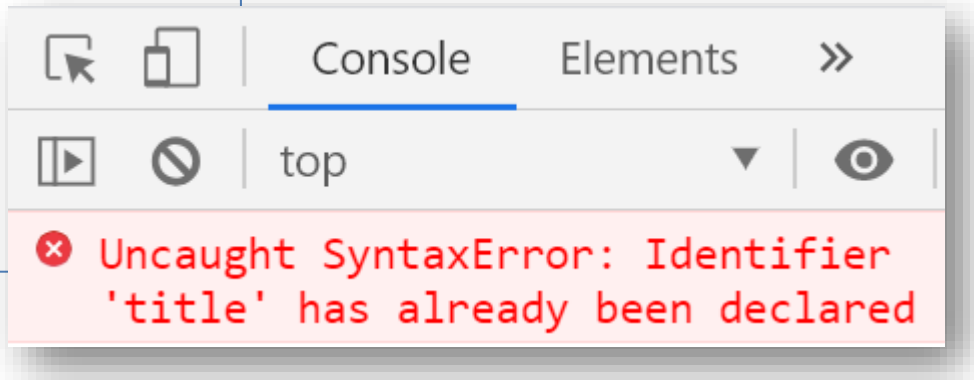
- ECMAScript 6 부터 추가된 문법
- 예전부터 사용되어오던 `var` 와 비슷한 기능

```
let title = '제목';
console.log(title);

title = '제목수정';
console.log(title);
```



- 중복선언 불가

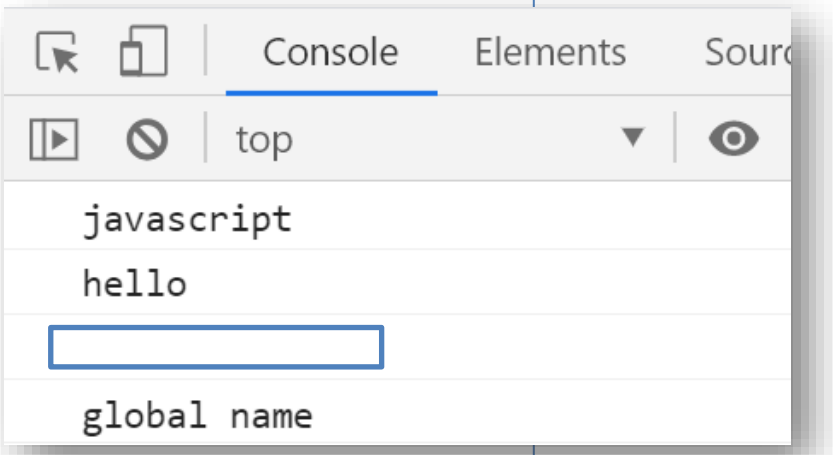
```
let title = '제목';
console.log(title);

let title = '제목수정';
console.log(title);
```



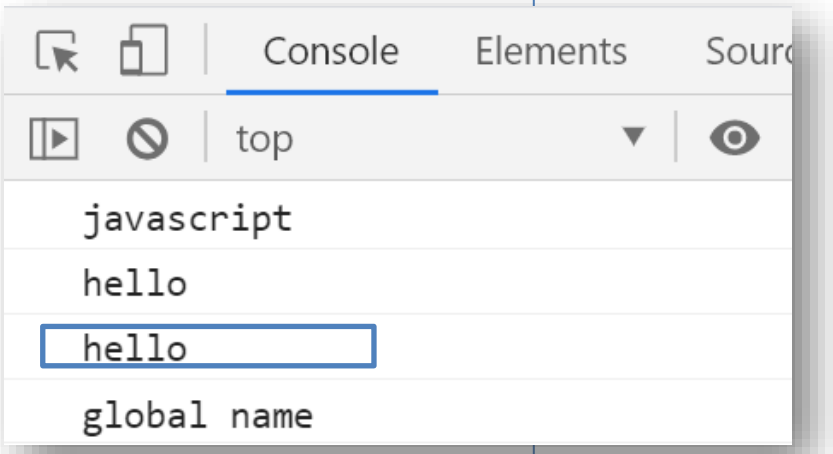
■ Variable (변수)

● 중괄호 사용 시 범위가 한정적 (지역변수)

```
let globalScope = 'global name';
{
  let name = 'javascript';
  console.log(name);
  name = 'hello';
  console.log(name);
}
console.log(name);
console.log(globalScope);
```




```
let globalScope = 'global name';
{
  var name = 'javascript';
  console.log(name);
  name = 'hello';
  console.log(name);
}
console.log(name);
console.log(globalScope);
```

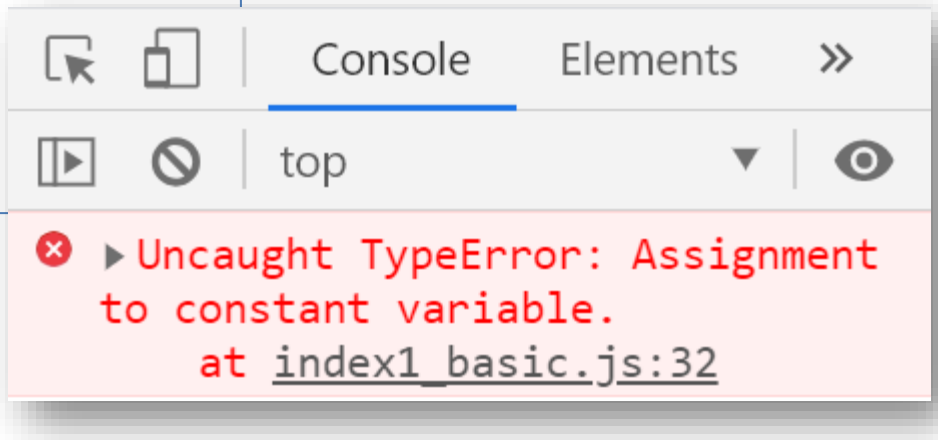


■ Constants (상수)

- ● `let` 변수와 사용법이 비슷하지만 입력값 수정 불가 (immutable)

  - 보안 / 멀티 스레드 환경에서 안전 / 실수 방지

```
const daysInWeek = 7;
const maxNumber = 5;
const title = '제목';
title = '제목수정';
```

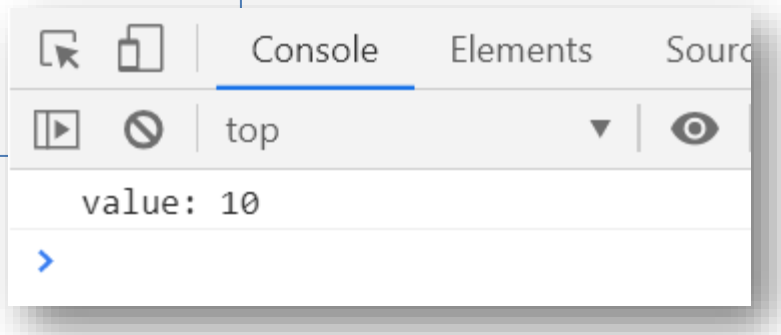


■ Hoisting

- 실제 선언 위치와 관계없이 코드의 위로 끌어올려서 선언해주는 것
- 값을 할당하는 것은 실제 위치에서 동작

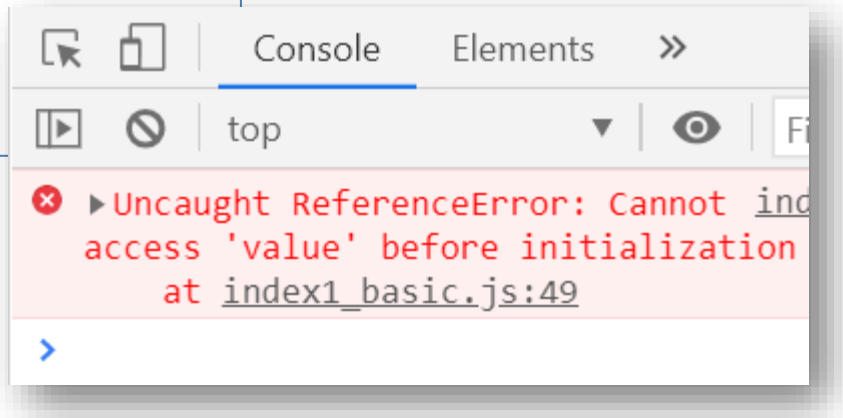
- var

```
value = 10;
console.log(value);
var value;
```



- let / const

```
value = 10;
console.log(value);
let value;
```



■ Variable Type

● Primitive (기본) : 더 이상 나눠지지 않는 하나의 값 (Single item)

- number, string, boolean, null, undefined, symbol

● Reference (참조) : Single item 들을 하나의 단위로 관리

- { }, [ ], class

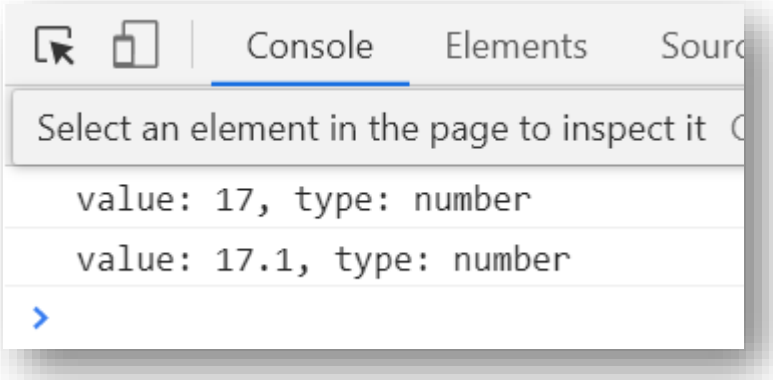
● Function (함수)

- First-class Function
- 함수를 다른 변수와 동일하게 다루는 것

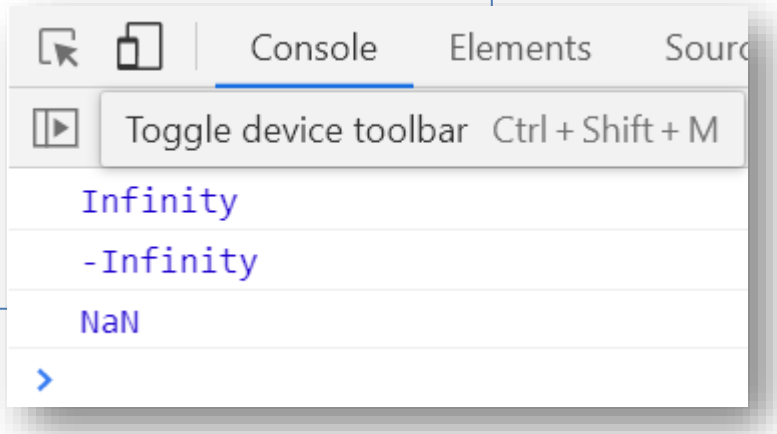
■ Primitive

● number

```
const count = 17; // integer
const size = 17.1; // decimal number
console.log(`value: ${count}, type: ${typeof count}`);
console.log(`value: ${size}, type: ${typeof size}`);
```



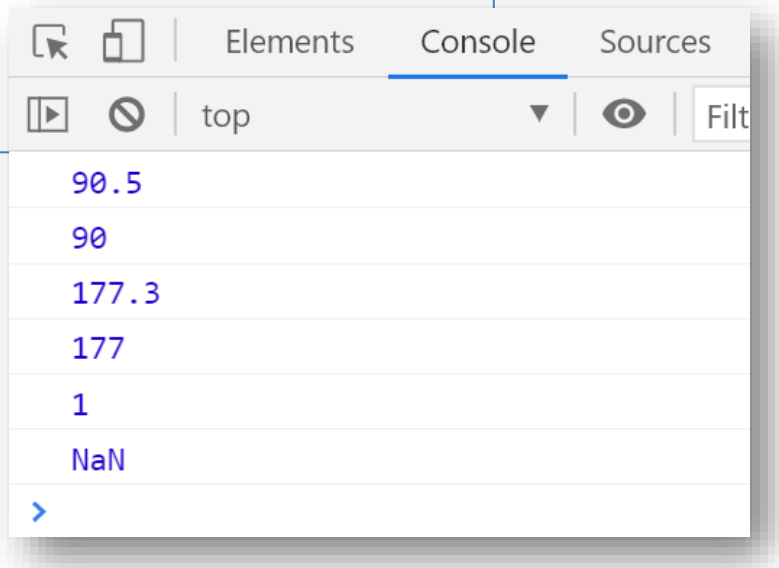

```
const infinity = 1 / 0;
const negativeInfinaty = -1 / 0;
const nan = 'not a number' / 2;
console.log(infinity);
console.log(negativeInfinaty);
console.log(nan);
```



■ Primitive

● number

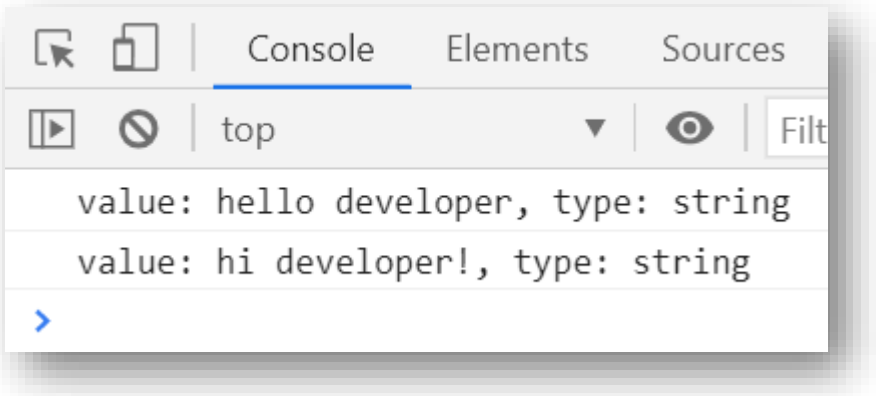
```
let score = 90.5;
console.log(Number(score));
console.log(parseInt(score));

let height = '177.3';
console.log(Number(height));
console.log(parseInt(height));

let bool = true;
console.log(Number(bool));
console.log(parseInt(bool));
```



■ Primitive

● string

```
const char = 'c';
const username = 'developer’;

const greeting = 'hello ' + username;
console.log(`value: ${greeting}, type: ${typeof greeting}`);

const template = `hi ${username}!`; // template literals
console.log(`value: ${template}, type: ${typeof template}`);
```

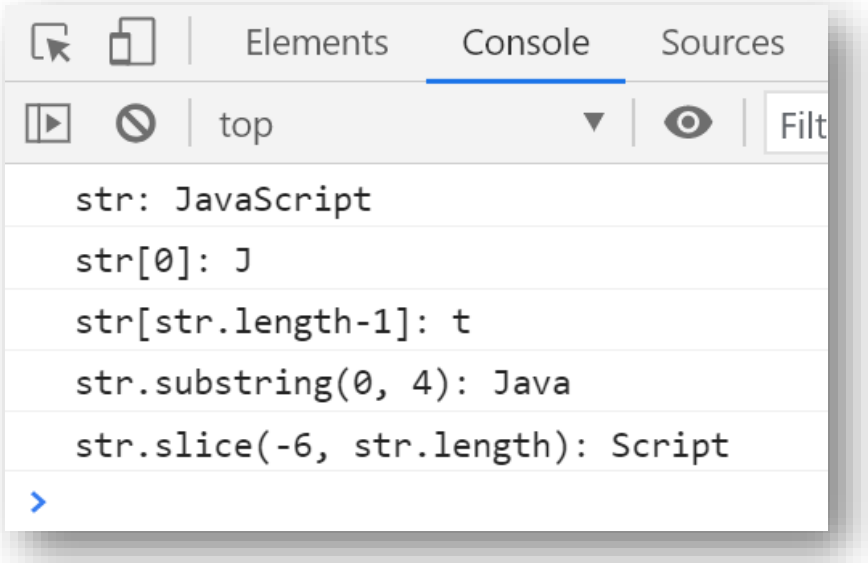



```
console.log('value: ' + template + ', type: ' + typeof template);
```

■ Primitive

● string (indexing / 함수)

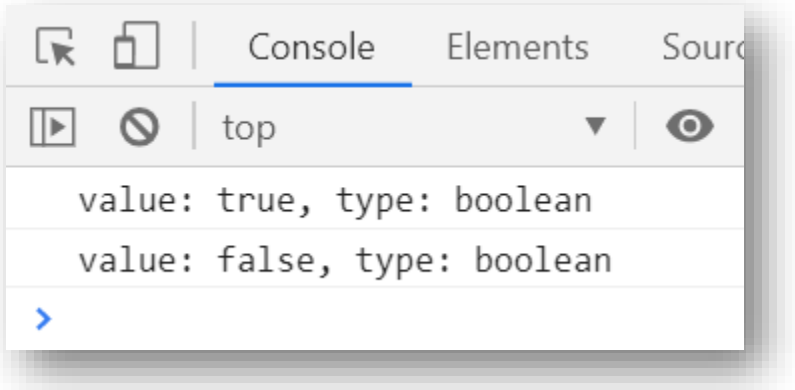
```
const str = 'JavaScript';

console.log(`${str}`)
console.log(`${str[0]}`)
console.log(`${str[str.length-1]}`)
console.log(`${str.substring(0, 4)}`)
console.log(`${str.slice(-6, str.length)}`)
```



■ Primitive

● boolean

```
const canRead = true;
console.log(`value: ${canRead}, type: ${typeof canRead}`);

const test = 3 < 1;
console.log(`value: ${test}, type: ${typeof test}`);

console.log([] ? true : false);
console.log({} ? true : false);
```

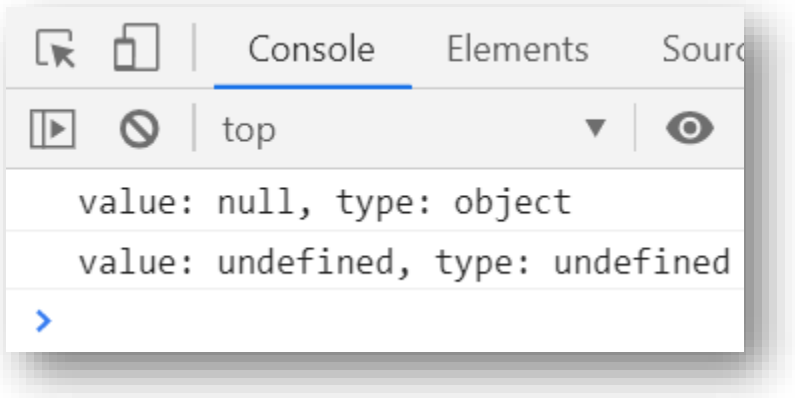


false : 0, null, undefind, NaN, ''
true : false가 아닌 값

■ Primitive

● null / undefined

```
let nothing = null;
console.log(`value: ${nothing}, type: ${typeof nothing}`);

let x = undefined; // let x;
console.log(`value: ${x}, type: ${typeof x}`);
```

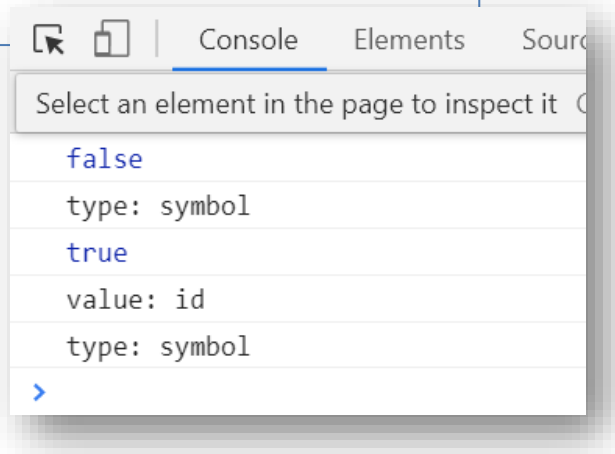


null : 값을 할당하지 않았음을 명시
undefined : 선언만 하고 값에 대해 언급하지 않음

■ Primitive

● symbol

```
const symbol1 = Symbol('id');
const symbol2 = Symbol('id');
console.log(symbol1 === symbol2);
console.log(`type: ${typeof symbol1}`);

const gSymbol1 = Symbol.for('id');
const gSymbol2 = Symbol.for('id');
console.log(gSymbol1 === gSymbol2);
console.log(`value: ${symbol1.description}`);
console.log(`type: ${typeof symbol1}`);
```



Map 등의 다른 자료구조에서 고유한 식별자가 필요하거나
동시다발적으로 작업이 일어나는 코드에서 우선 순위를 주는 등
고유한 식별자가 필요한 경우에 사용

Symbol : 완전히 고유한 값을 부여할 때 사용
Symbol.for : 주어진 문자열이 같으면 같은 값으로 사용

■ Reference

● object

```
const user = { name: 'developer', age: 30 };
user.age = 29;
console.log(user);

const arr = [ 1, 2, 3 ];
arr[0] = 10;
console.log(arr);
```