03. Data Type

1. Use strict

```
// add in ES 5
// use this for valina JavaScript
'use strict';
```

2. Variable, rw(read/write)

☞ let (added in ES6)

```
let globalName = 'global name';
{
  let name = 'SmartIT';
  comsole.log(name);
  name = 'hello';
  console.log(name);
  console.log(globalName);
}
console.log(name);  // {} 밖에서는 더 이상 볼 수 가 없다.
console.log(globalName);
```

☞ var (don't ever use this!)

☞ var hoisting (move declaration from bottom to top)

☞ has no block scope

3. Constant, r(read only)

☞ use const whenever possible.

☞ only use let if variable needs to change.

```
const daysInWeek = 7;
const maxNumber = 5;
```

☞ Note~

☞ Immutable data types: primitive types, frozen objects (i.e. object.freezen())

☞ Mutable data types: all objects by default are mutable in JS

☞ favor immutable data type always for a few reasons.

✓ security

✓ thread safety

✓ reduce human mistake

4. Variable types

☞ primitive, single item: number, string, boolean, null, undefined, symbol

☞ object, box container

☞ function, first-class function

❖ C data types for number

```
int main() {
        short a = 12;           // 2 bytes
        int a = 12;             // 4 bytes
        long b = 1234;          // 8 bytes
        float d = 1.2f          // 4 bytes
        double e = 8.2; // 16 bytes
        return 0;
}
```

❖ Java data types for number

```
class Main {
   public static void main(String[] args) {
        byte a = 12;            // 1 bytes
        short b =12;            // 2 bytes
        int a = 12;             // 4 bytes
        long b = 12;            // 8 bytes
        float d = 1.2f          // 4 bytes
        double e = 8.2; // 16 bytes
   }
}
```

☑ JavaScript data types for number

⊙ number

```
let a = 12;
let b = 1.2;
```

☞ TypeScript

```
let a: number = 12;
let b: number = 1.2;
```

☞ number - special number values: infinity, -infinity, NaN

```
const infinity = 1 / 0;
const negativeInfinity = -1 / 0;
const nAn = 'not a number' /2;

console.log(infinity);
console.log(negativeInfinity);
console.log(nAn);
```

☞ bigInt (fairly new, don't use it yet)

```
const bitInt = 12345678901234567890123456789012345678901234567890n; // over (-2**53) ~ 2**53
console.log(`value: ${bigInt}, type: ${typeof bigInt}`);
Number.MAX_SAFE_INTEGER;
```

⊙ string

```
const char = 'c';
const brendan = 'brandan';
console.log(`value: ${greetong}, type: ${typeof greeting}`);

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

⊙ boolean

```
// false : 0, null, undefined, NaN, ''
// true : any orther value
const canRead = 'true';
const test = 3 < 1; // false
console.log(`value: ${canRead}, type: ${typeof canRead}`);
console.log(`value: ${test}, type: ${typeof test}`);
```

⊙ null

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

⊙ undefined

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

⊙ symbol, create unique identifiers for objects

```
const symbol1 = Symbol('id');
const symbol2 = Symbol('id');
console.log(symbol1 === symbol2);       // false

const gSymbol1 = Symbol.for('id');
const gSymbol2 = Symbol.for('id');
console.log(gSymbol1 === gSymbol2);     // true

console.log(`value: ${symbol1}, type: ${typeof symbol1}`); // error 발생
console.log(`value: ${symbol1.description}, type: ${typeof symbol1}`);
```

5. Dynamic typing: dynamically typed language

```
let text = 'hello';
// console.log(text.charAt(0));  // h
console.log(`value: ${text}, type: ${typeof text}`);

text = 1;
console.log(`value: ${text}, type: ${typeof text}`);

text = '7' + 5;
console.log(`value: ${text}, type: ${typeof text}`);

text = '8' / '2';
console.log(`value: ${text}, type: ${typeof text}`);
// console.log(text(charAt(0));         // error 발생
```

☞ JavaScript는 Runtime에서 타입이 정해지기 때문에 error가 발생한다.

❖ TypeScript

☞ Dynamic typing 때문에 위와 같은 현상을 방지하기 위하여 TypeScript가 나오게 되었다.

☞ 이 TypeScript는 JavaScript 위에 타입이 더 올려진 언어이다.

☞ 그러나 JavaScript를 이용하면 바로 브라우저가 이해할 수 있기 때문에 실시간으로 연동해서 볼 수가 있지만, TypeScript는 결국 브라우저가 이해할 수 있는 JavaScript로 transcompilation을 이용해야 되기 때문에 실시간으로 보기가 어렵다.

```
// object, real-life object, data structure

const smu = { name: 'smart', age: 25 };
smu.age = 21;
```