Lab 08. JavaScript (1)

인터넷과웹기초

How to write JS code

- Internal JavaScript
  - in HTML, between <script> and </script> tags.

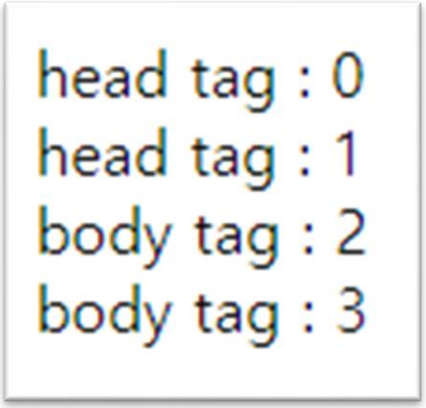
- External JavaScript
  - JavaScript files have the file extension **.js**
  - External Ref. → <script src="/js/myScript.js"></script>

- Precautions
  - It must be written as case-sensitive.
  - Basically, Sentences are separated by semicolons.

JS in <head> and <body>

- You can write JS code in <head> and <body>

- Execution order




```html
<html>
    <head>
        <script>
            var num=0;
            document.write("head tag : " + num + "<br>");
        </script>
        <script>
            var num=1;
            document.write("head tag : " + num + "<br>");
        </script>
    </head>
    <body>
        <script>
            var num=2;
            document.write("body tag : " + num + "<br>");
        </script>
        <script>
            var num=3;
            document.write("body tag : " + num + "<br>");
        </script>
    </body>
</html>
```

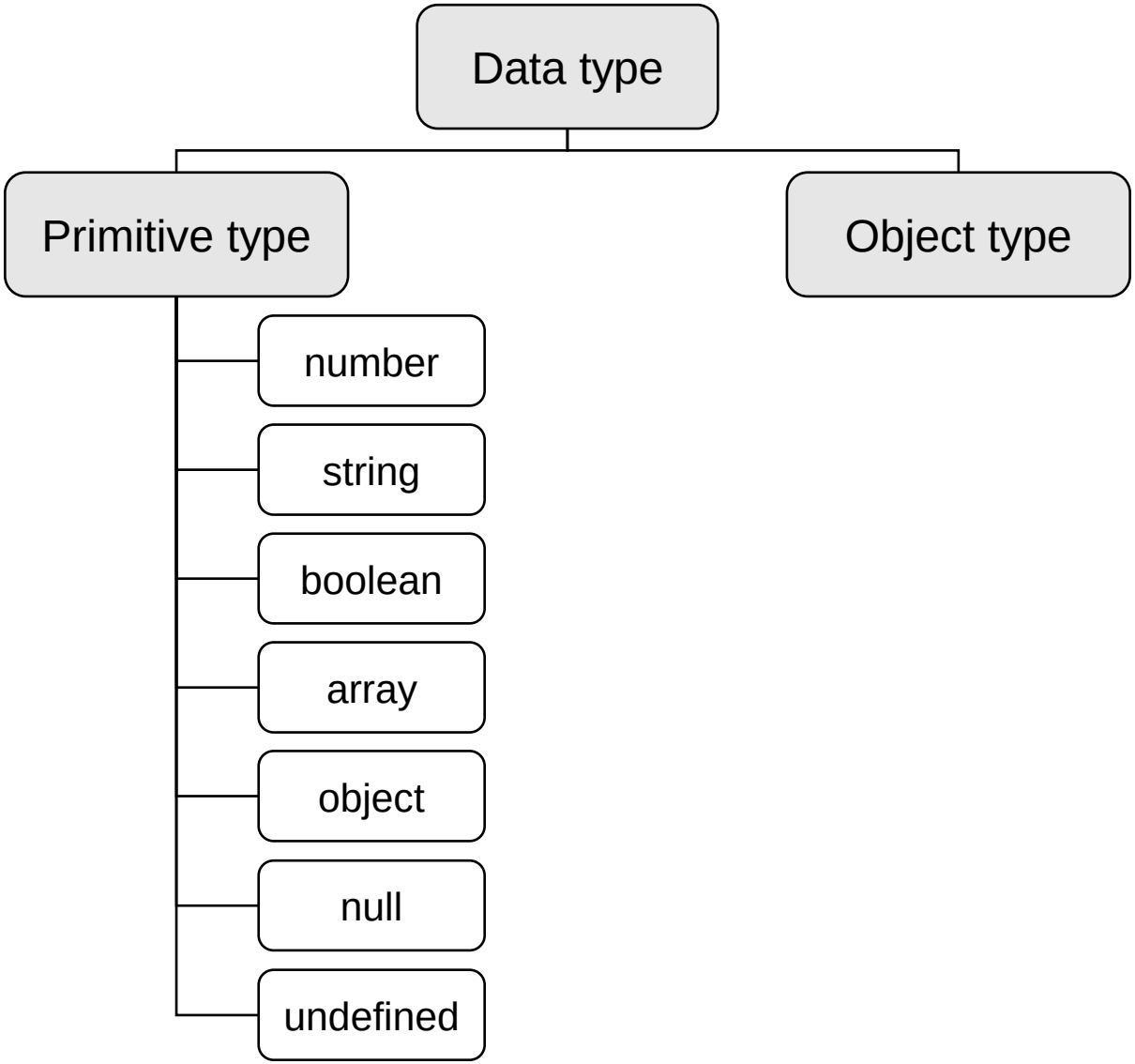
Operators in JS

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Operator	Description
&	AND
|	OR
~	NOT
^	XOR
<<	Zero fill left shift
>>	Signed right shift
>>>	Zero fill right shift

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Data Type in JS



Data Type in JS (cont'd)

- Checking data type using the typeof operator

number
number
string
boolean
object
object
undefined
object

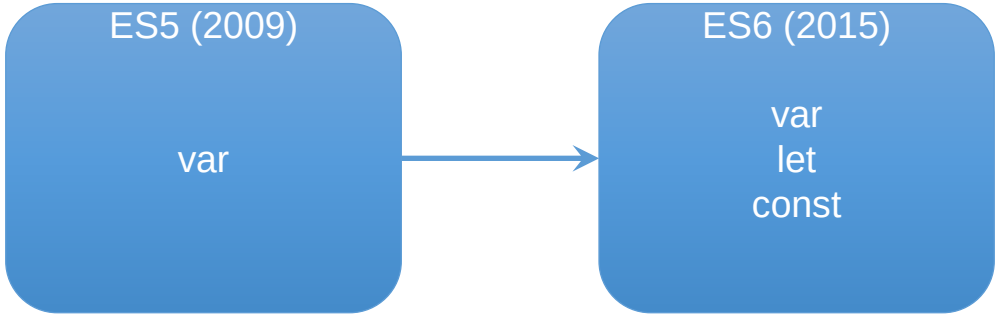
```
<html>
    <head></head>
    <body>
        <script>
            var num;
            var obj=null;
            document.write(typeof 100 + "<br>");
            document.write(typeof 10.5 + "<br>");
            document.write(typeof "name" + "<br>");
            document.write(typeof true + "<br>");
            document.write(typeof [1,2,3] + "<br>");
            document.write(typeof {name:'name'} + "<br>");
            document.write(typeof num + "<br>");
            document.write(typeof obj + "<br>");
        </script>
    </body>
</html>
```

Variables

Variables

- There are 3 ways to declare a JavaScript variable:
  - var
  - let
  - const

- ECMAScript (ES)
  - JavaScript was developed by Netscape Communications, after Microsoft developed JScript.
  - there is a cross-browser issues.
  - To solve this issues, JS was standardized with ECMAScript.

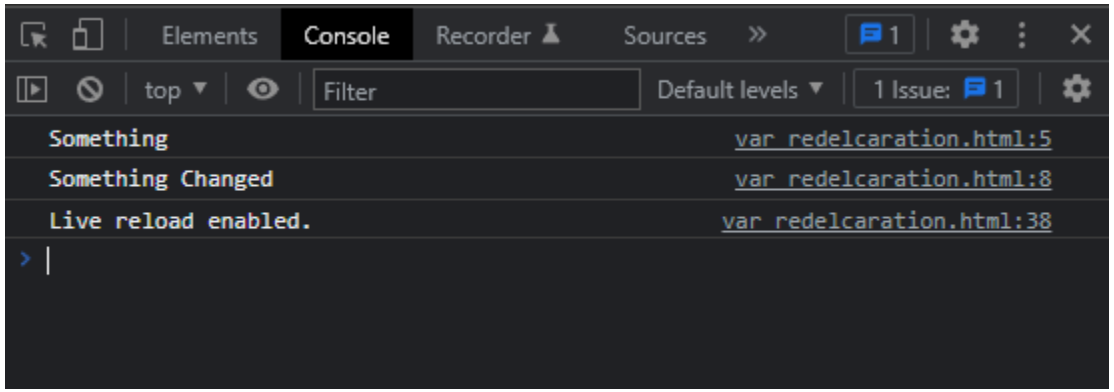


Redeclaration

- var
  - The value assigned at the time of variable declaration can be changed at the time of redeclaration.

```html
<html>
    <body>
        <script>
            var text = "Something";
            console.log(text);

            var text = "Something Changed";
            console.log(text);
        </script>
    </body>
</html>
```



No errors!

Redeclaration

- let, const
  - After ES6 (2015), let and const cannot be redeclared.

let_redeclaration.html

```
let text: string
Cannot redeclare block-scoped variable 'text'. javascript
View Problem    No quick fixes available
```

```html
1   <html>
2       <body>
3           <script>
4               let text = "Something";
5               console.log(text);
6
7               let text = "Something Changed";
8               console.log(text);
9           </script>
10      </body>
```

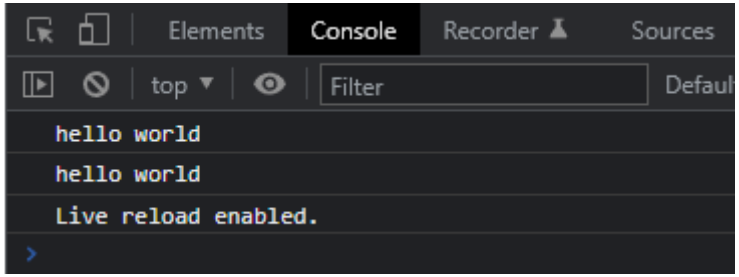
```
Elements  Console  Recorder
top  Filter  Default levels  1 Issue: 1
Uncaught SyntaxError: Identifier 'text' has already been declared   let_redeclaration.html:7
Live reload enabled.   let_redeclaration.html:38
>
```

Scope

- Scope in JavaScript
  - Before ES6, JavaScript had Global Scope and Function Scope.

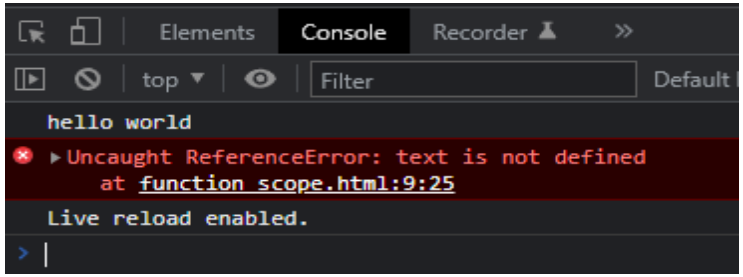
- Global Scope

```
<script>
    var text = "hello world";
    function func() {
        console.log(text);
    }
    func();
    console.log(text);
</script>
```



- Function Scope

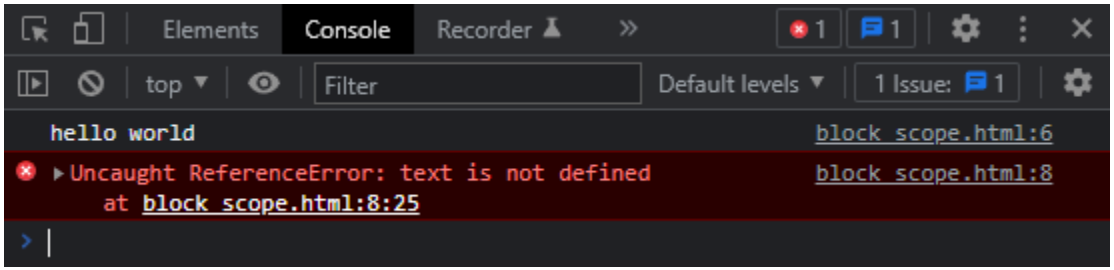
```
<script>
    function func() {
        var text = "hello world";
        console.log(text);
    }
    func();
    console.log(text);
</script>
```



Scope

- Block Scope
  - ES6 introduced two important new JavaScript keywords: let, const
  - These two keywords provide Block Scope in JavaScript.
  - Variables declared inside a { } block cannot be accessed from outside the block

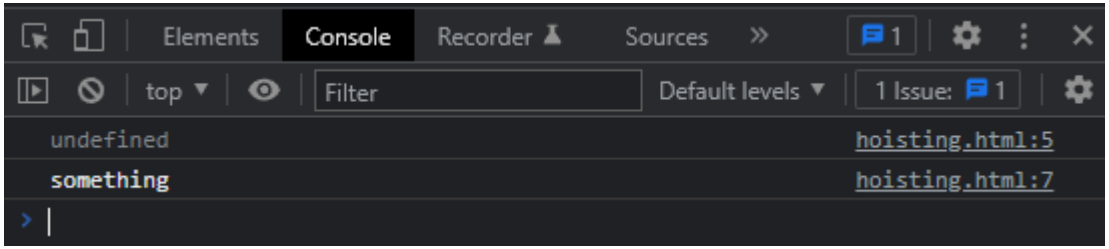
```
<script>
    {
        var text = "hello world";
        console.log(text);
    }
    console.log(text);
</script>
```



Hoisting

- Hoisting in JavaScript
  - Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

```
<script>
    function hoisting () {
        console.log(text);
        var text = "something";
        console.log(text);
    }
</script>
```



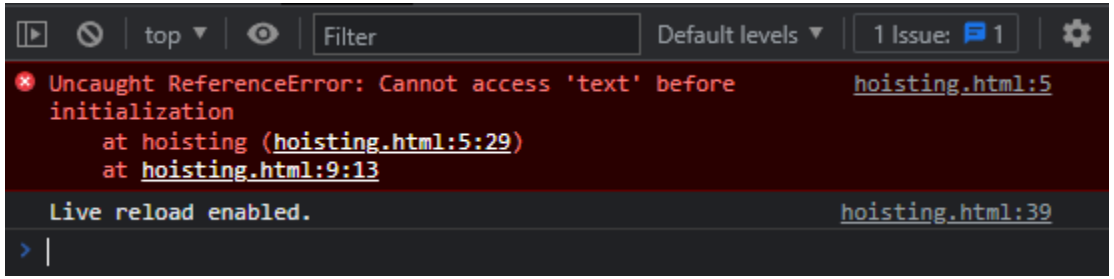
hoisting

```
<script>
    function hoisting () {
        var text;
        console.log(text);
        text = "something";
        console.log(text);
    }
    hoisting();
</script>
```

Hoisting

- Are let and const not hoisted?

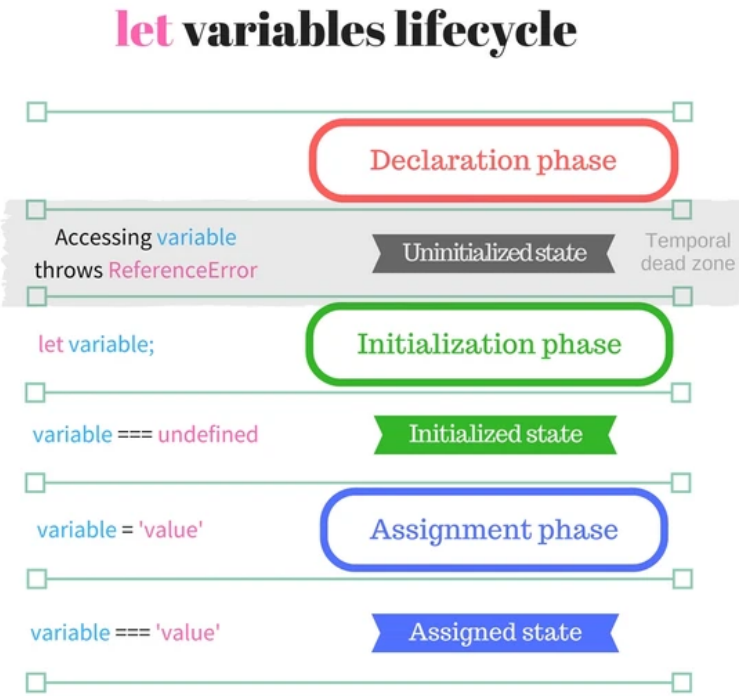
```
<script>
    function hoisting () {
        console.log(text);
        let text = "something";
        console.log(text);
    }
    hoisting();
</script>
```



ReferenceError!
➔ because **TDZ**

Temporal Dead Zone

- TDZ
  - Temporal Dead Zone
  - When entering the scope a variable is created and also a Temporal Dead Zone (TDZ) is created.
  - cannot accessible until code execution reaches where the variable actually resides.



출처: https://velog.io/@yogongman/TDZ-Temporal-Dead-Zone

Type safety

- what type-safety is?
  - type safety is the extent to which a programming language discourages or prevents type errors. (Wikipedia)
- Arithmetic operators in JavaScript
- The + operator can also be used to concatenate strings
  - Adding a number and a string will return a string

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Example

```
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;
```

The result of *x*, *y*, and *z* will be:

```
10
55
Hello5
```

Type safety

```
<script>
    function addition (a, b) {
        return a + b;
    }
    document.write(("the answer is " +  addition(true,2));
</script>
```

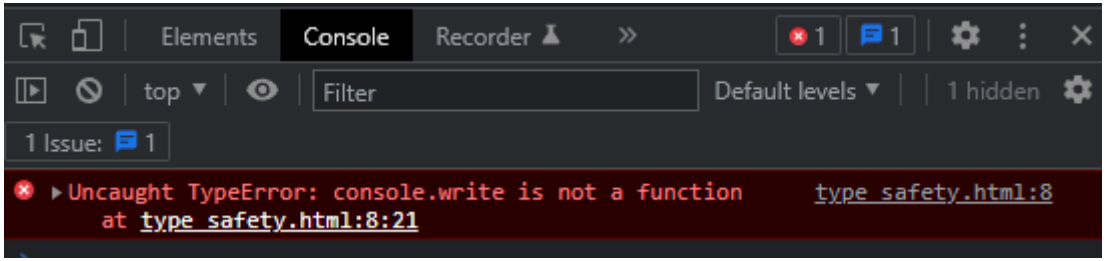
the answer is 3

```
<script>
    function addition (a, b) {
        return a + b;
    }
    document.write("the answer is " + addition("hello",2));
    console.write(addition("hello", 2));
</script>
```

the answer is hello2

There is no error!!
but

```
console.write(addition(true, 2));
```



Type safety

- TypeScript
  - in TypeScript, Developer must specify the type of the variable.
  - Type-safe JavaScript

```
<script>
    function addition (a, b) {
        return a + b;
    }
</script>
```

```
<script>
    function addition (a:number, b:number) {
        return a + b;
    }
</script>
```

Function

Function

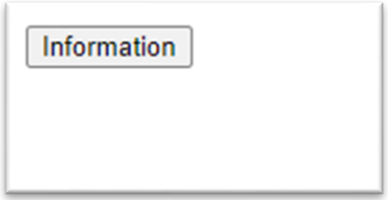
- Function
  - function is a block of code designed to perform a particular task.
  - it is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

```
<script>
    function name(parameter1, parameter2) { //function declaration
        //code to be executed
        return value;
    }
    name(param1, param2); //function invocation
</script>
```

Function: Example

- Example
  - Invoking a function with the onclick attribute.

```
<script>
    function msg(name, age) {
        document.write("name : " + name + "</b><br>");
        document.write("age : " + age + "</b><br>");
    }
</script>
<button type="button" onclick="msg('John',
20)">Information</button>
```



Anonymous Function

- Anonymous function
  - 익명 함수 or 무명 함수
  - Declare a function expression and assign it to a variable.
  - Using a variable as a function name.
  - **Not hoisted

```
<script>
    var something = function(param) {
    document.write("invoking message : " + param + "<br>");
    }
    something("test message");
</script>
```

invoking message : test message

Return value

- Return value
  - use ‘return’ keyword
  - if there is no return value, it can be omitted.

```
<script>
    function addition (a, b) {
        return a + b;
    }
    document.write(("the answer is " +  addition(true,2));
</script>
```

the answer is 3

- Arguments and parameters

```
<script>
    function func(parameter1, parameter2) {
        //code execution
        return value;
    }
    result = func(argument1, argument2);
</script>
```

Case: the number of arguments

- When there are few arguments when invoking a function
  - → NaN

```
<script>
    function add(x, y, z) {
        var sum = x+y+z;
        return sum;
    }
    document.write("first : " + add(1) + "<br>");
    document.write("second : " + add(1, 3) + "<br>");
</script>
```

first : NaN
second : NaN

NaN : Not a Number

- When there are many arguments when calling a function
  - → Receive up to the number of parameters

```
<script>
    function add(x, y, z) {
        var sum = x+y+z;
        return sum;
    }
    document.write("first : " + add(1, 2, 3, 4) + "<br>");
    document.write("second : " + add(1, 3, 5, 7, 9) + "<br>");
</script>
```

first : 6
second : 9

Overriding

- Overriding
  - Using the same function name while writing different number of arguments and different data types so that they can be distinguished when invoking a function.

```
<script>
    function add(x, y, z) {
        var sum;
        if((y===undefined) && (z===undefined))) {
            sum = x;
        }
        else if(z===undefined) {
            sum = x + y;
        }
        else {
            sum = x + y + z;
        }
        return sum;
    }
    document.write("case 1 : " + add(1) + "<br>");
    document.write("case 2 : " + add(1, 2) + "<br>");
    document.write("case 3 : " + add(1, 2, 3) + "<br>");
</script>
```

```
case 1 : 1
case 2 : 3
case 3 : 6
```