Introduction to Internet and Web

Taewoon Kim

Computer Science and Engineering

Outline

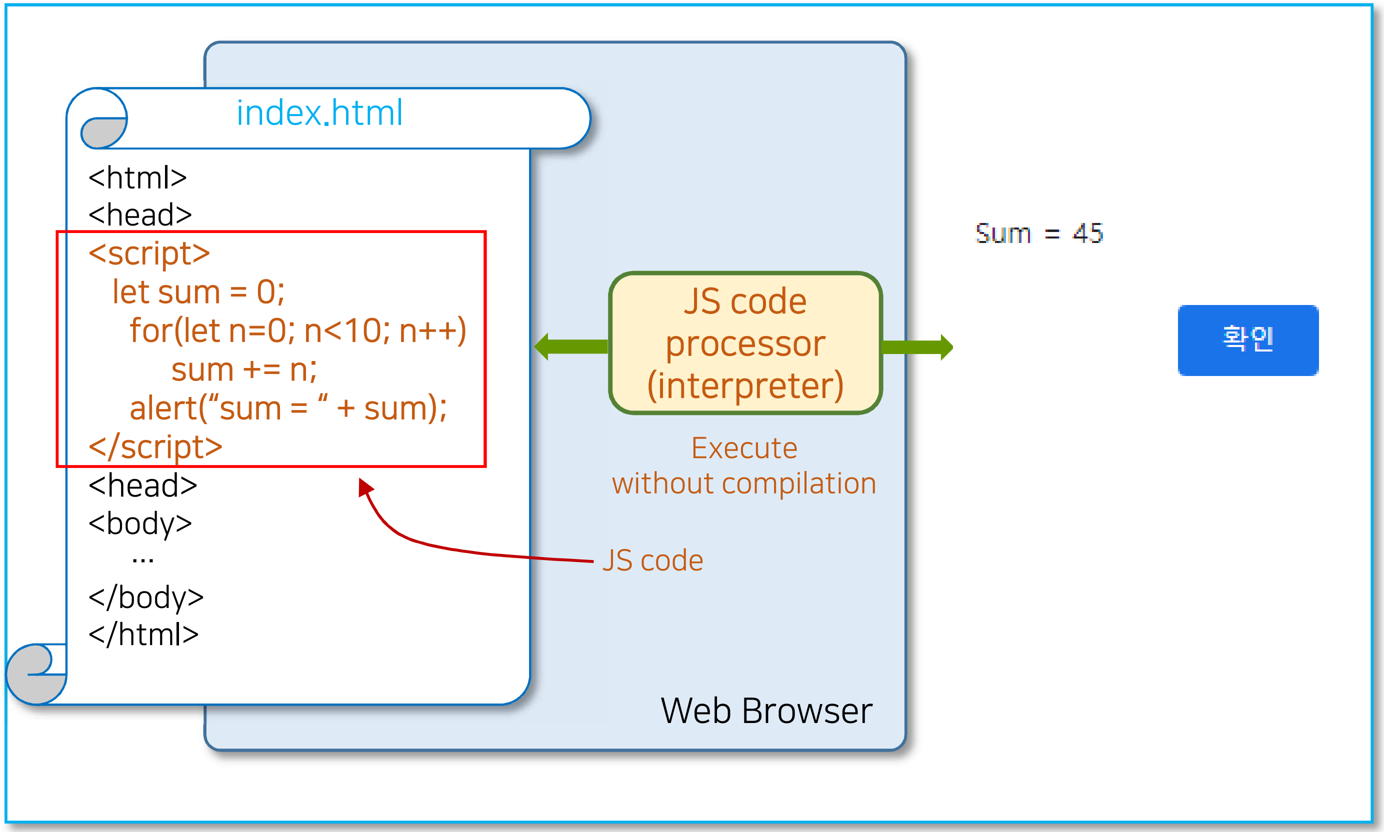
- JavaScript (JS)
    - Introduction
    - Inserting JS code into web page
    - Data types and variables
    - Operators
    - Conditional statements
    - Iteration
    - Function

JavaScript (JS)

- Javascript : History
  - JavaScript was invented by Brendan Eich in 1995.
  - It was developed for Netscape 2, and became the ECMA-262 standard in 1997.
  - After Netscape handed JavaScript over to ECMA, the Mozilla foundation continued to develop JavaScript for the Firefox browser. Mozilla's latest version was 1.8.5. (Identical to ES5).
  - Internet Explorer (IE4) was the first browser to support ECMA-262 Edition 1 (ES1).

JavaScript (JS)

- JavaScript : Features
  - JavaScript is the world's most popular programming language.
  - JavaScript is the programming language of the Web.
  - JavaScript is easy to learn.



```
<html>
<head>
<script>
  let sum = 0;
    for(let n=0; n<10; n++)
        sum += n;
    alert(“sum = “ + sum);
</script>
<head>
<body>
    …
</body>
</html>
```

- JS code can be embedded in HTML code
- JS code runs without compilation : Web browser’s built-in processor (interpreter) directly executes the JS code

JavaScript (JS)

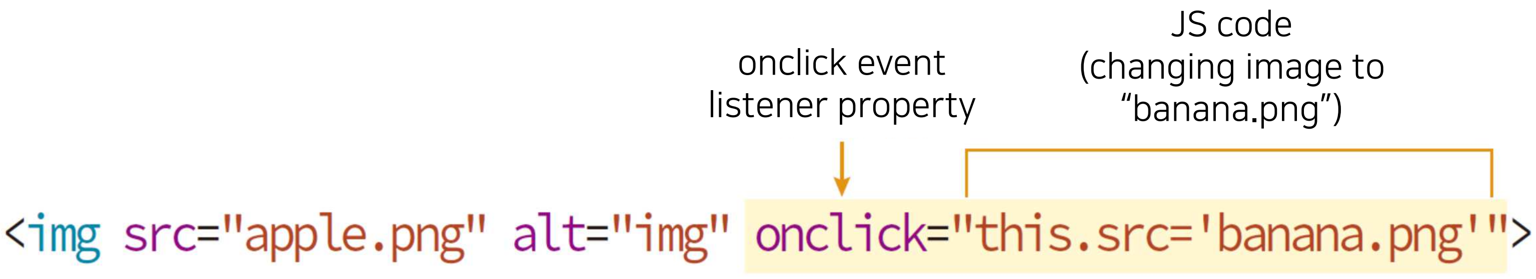
- What JS can do in a web page?
  - Getting user input and computation
    - Only JS can takes input from mouse and keyboard
      - HTML forms provide the input area only, and it cannot take in or process the user input
      - But, with JS we can take in and process the user input
    - Computation/calculation/event handling, etc.
  - Dynamic control of web page contents and style
    - we can dynamically change HTML tag's property, contents, CSS property values, etc.
  - Browser control
    - Changing browser's window size and shape
    - Opening/closing a window
    - Connecting another web site
    - History control, etc.
  - Communicating with web server
  - Making diverse web applications
    - canvas, graphic, local/session storage, location information service, etc.

JavaScript (JS)

- Four different ways of using JS
  - We can write JS code in …
  1. HTML tag’s event listener property
  2. Within <script></script> tag pair
  3. External JS file
  4. The place for URL

JavaScript (JS)

- [1] Writing JS code in HTML tag's event listener property
  - With the event listener property, one can program the behavior for a particular event (e.g., mouse click)
  - To do so, we can write a JS code to the event listener property
  - This is useful when the JS code is short
  - Example:



onclick event listener property

JS code (changing image to "banana.png")

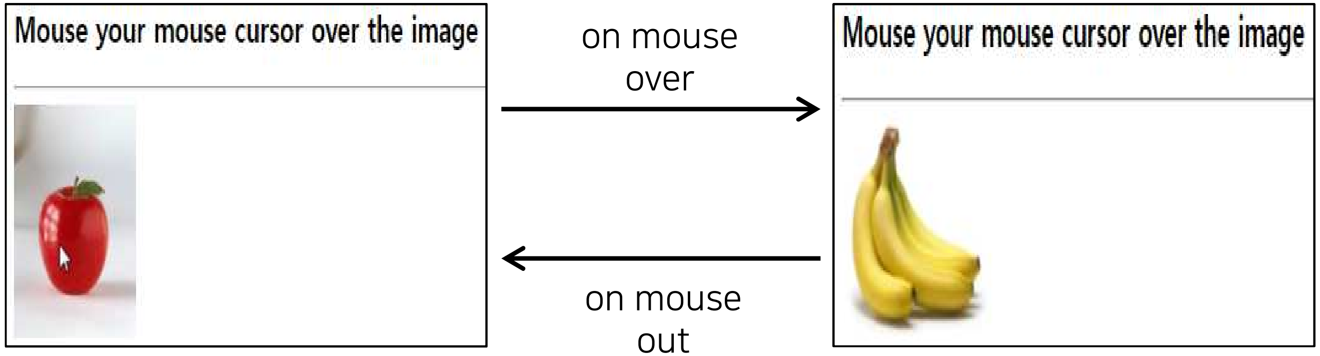
```
<img src="apple.png" alt="img" onclick="this.src='banana.png'">
```

- By default, "apple.png" will be shown on the web browser.
- For the click event on the apple image, it will be changed to banana.png

JavaScript (JS)

- [1] Writing JS code in HTML tag's event listener property

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JS</title>
</head>
<body>
<h3>Mouse your mouse cursor over the image</h3>
<hr>
<img src="media/apple.png" alt="image"
        onmouseover="this.src='media/banana.png'"
        onmouseout="this.src='media/apple.png'">
</body>
</html>
```



Mouse your mouse cursor over the image

Mouse your mouse cursor over the image

event listener property

A short JS code

"this" = current tag (i.e., "img tag" in this example)
Thus, "this.src" = "img src"

JavaScript (JS)

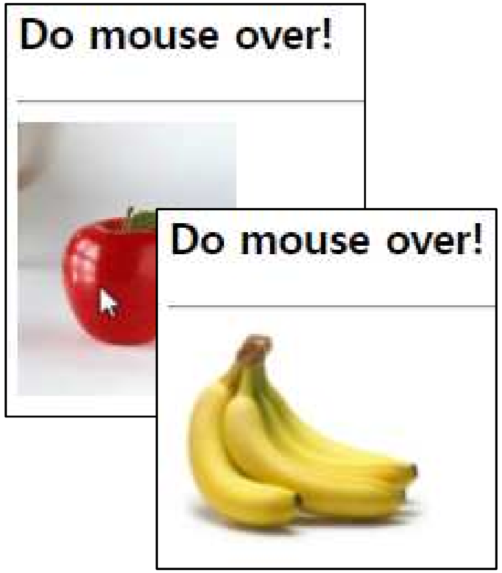
- [2] Writing JS code within <script></script> tag pair
  - The <script></script> tag pair can be placed anywhere
  - There can be multiple <script></script> tag pairs in a single HTML file

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JS</title>
<script>
function over(obj) {
    obj.src="media/banana.png";
}
function out(obj) {
    obj.src="media/apple.png";
}
</script>
</head>
<body>
<h3>Do mouse over!</h3>
<hr>
<img src="media/apple.png" alt="image"
        onmouseover="over(this)"
        onmouseout="out(this)">
</body>
</html>
```

Argument “obj” will become <img> tag in this example

JS code, calling the function named “over” and passing an argument “this” to the function

In this case, “this” = <img> tag



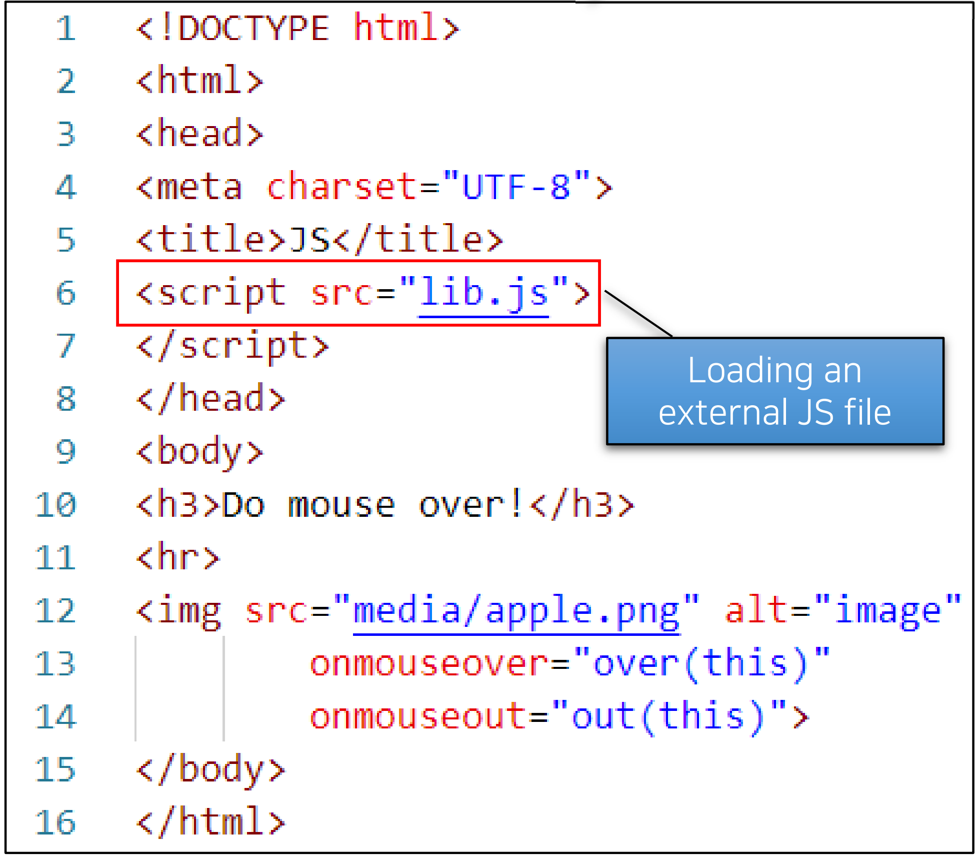
Do mouse over!

Do mouse over!

JavaScript (JS)

- [3] Writing JS code in an external *.js file

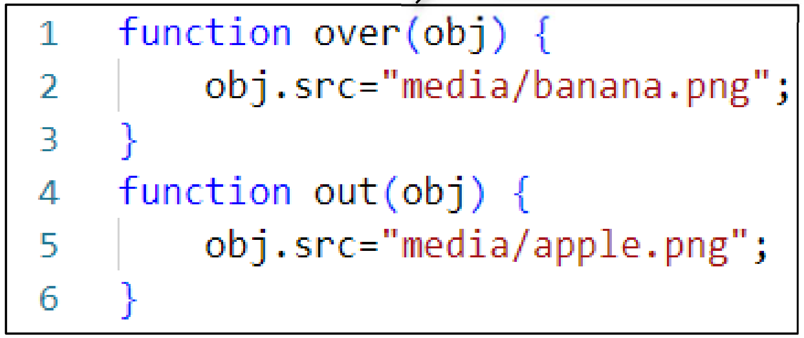
fruit.html



```html
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JS</title>
<script src="lib.js">
</script>
</head>
<body>
<h3>Do mouse over!</h3>
<hr>
<img src="media/apple.png" alt="image"
        onmouseover="over(this)"
        onmouseout="out(this)">
</body>
</html>
```

Loading an external JS file

lib.js

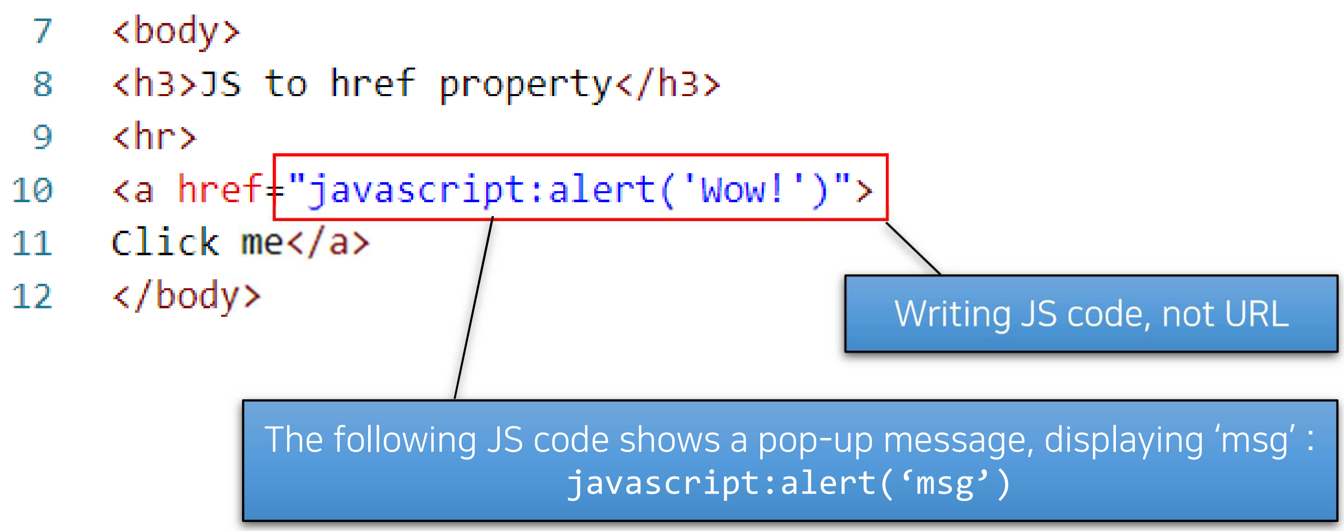


```javascript
function over(obj) {
    obj.src="media/banana.png";
}
function out(obj) {
    obj.src="media/apple.png";
}
```

JavaScript (JS)

- [4] Writing a JS code in the place for URL
  - For example, to the **href** property of **<a>** tag, we can write a JS code

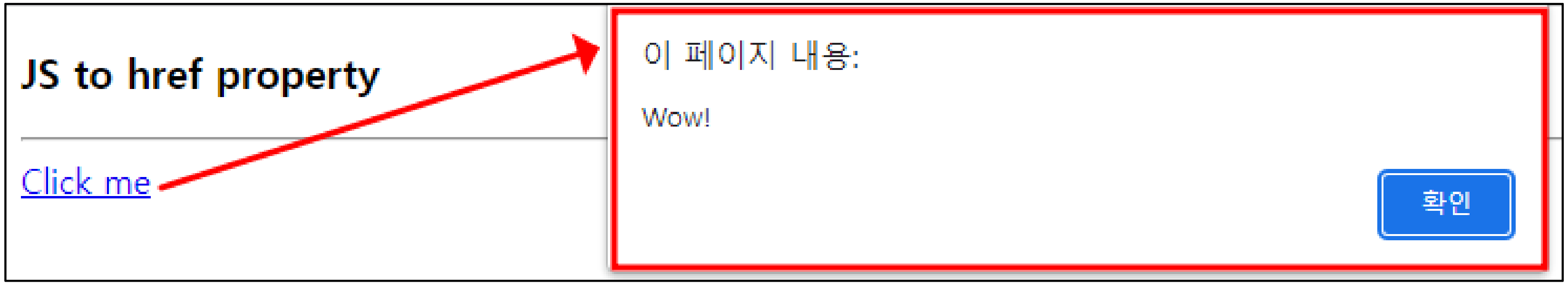
```
<a href=“javascript:your_JS_code_here”>Click Here</a>
```



```
<body>
<h3>JS to href property</h3>
<hr>
<a href="javascript:alert('Wow!')">
Click me</a>
</body>
```

Writing JS code, not URL

The following JS code shows a pop-up message, displaying ‘msg’ :
`javascript:alert(‘msg’)`



JS to href property

Click me

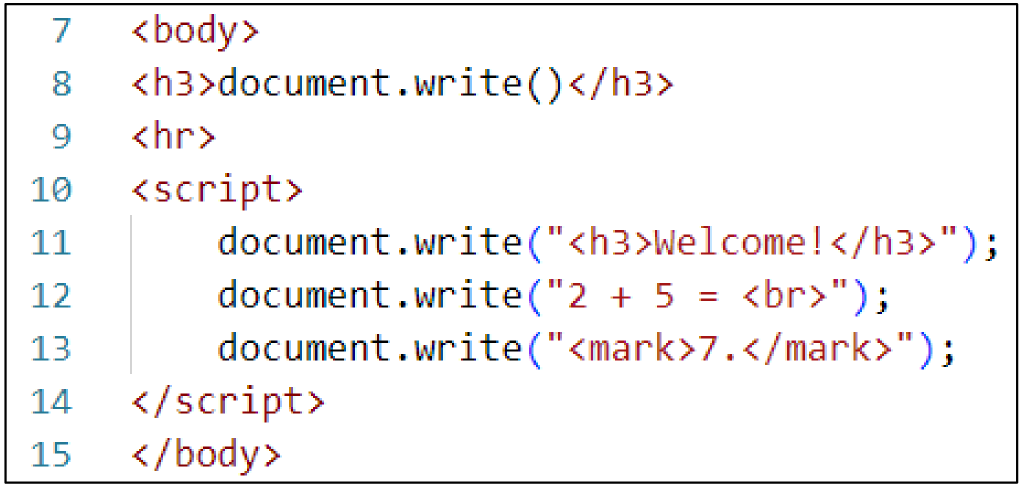
이 페이지 내용:

Wow!

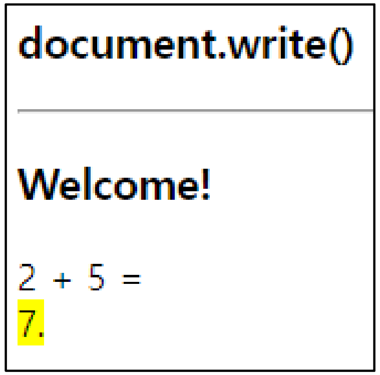
JavaScript (JS)

- Generating HTML contents by using JS

`document.write(“msg”);`	• This writes “msg” to the same line • In other words, the code document.write(“msg”); will be replaced to “msg” when the HTML file is loaded on a web browser.
`document.writeln(“msg”);`	• This writes “msg” + new line character (‘₩n’) to the same line • In other words, the code document.write(“msg”); will be replaced to “msg₩n” when the HTML file is loaded on a web browser.



```
<body>
<h3>document.write()</h3>
<hr>
<script>
    document.write("<h3>Welcome!</h3>");
    document.write("2 + 5 = <br>");
    document.write("<mark>7.</mark>");
</script>
</body>
```



document.write()

Welcome!

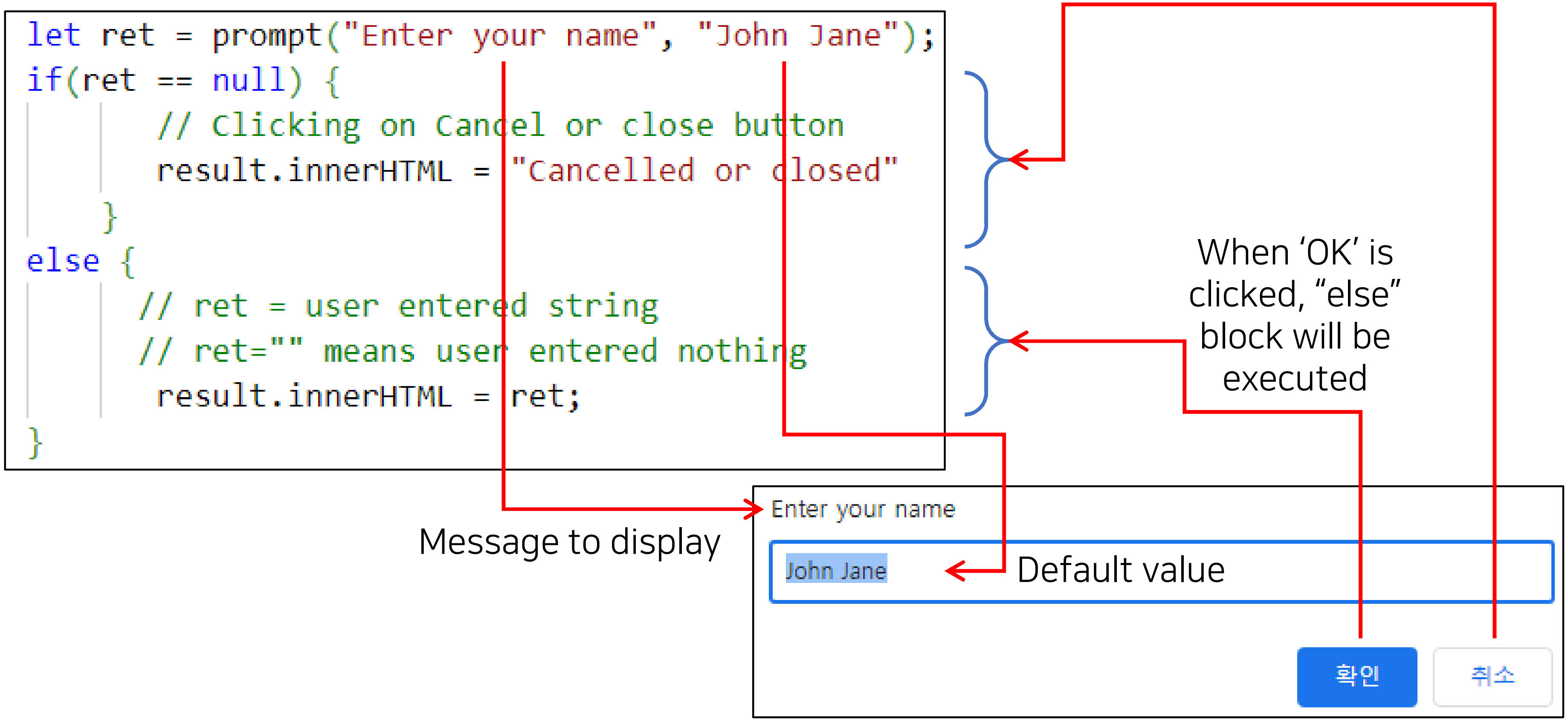
2 + 5 =
7.

JavaScript (JS)

- With JS, we can make three different dialogs to…
  - show a pop-up message or
  - get an input from user

- JS dialog : (1) “prompt” to get user input

When ‘Cancel’ is clicked,
“if” block will be executed
since ret becomes null



```
let ret = prompt("Enter your name", "John Jane");
if(ret == null) {
        // Clicking on Cancel or close button
        result.innerHTML = "Cancelled or closed"
    }
else {
       // ret = user entered string
       // ret="" means user entered nothing
        result.innerHTML = ret;
}
```

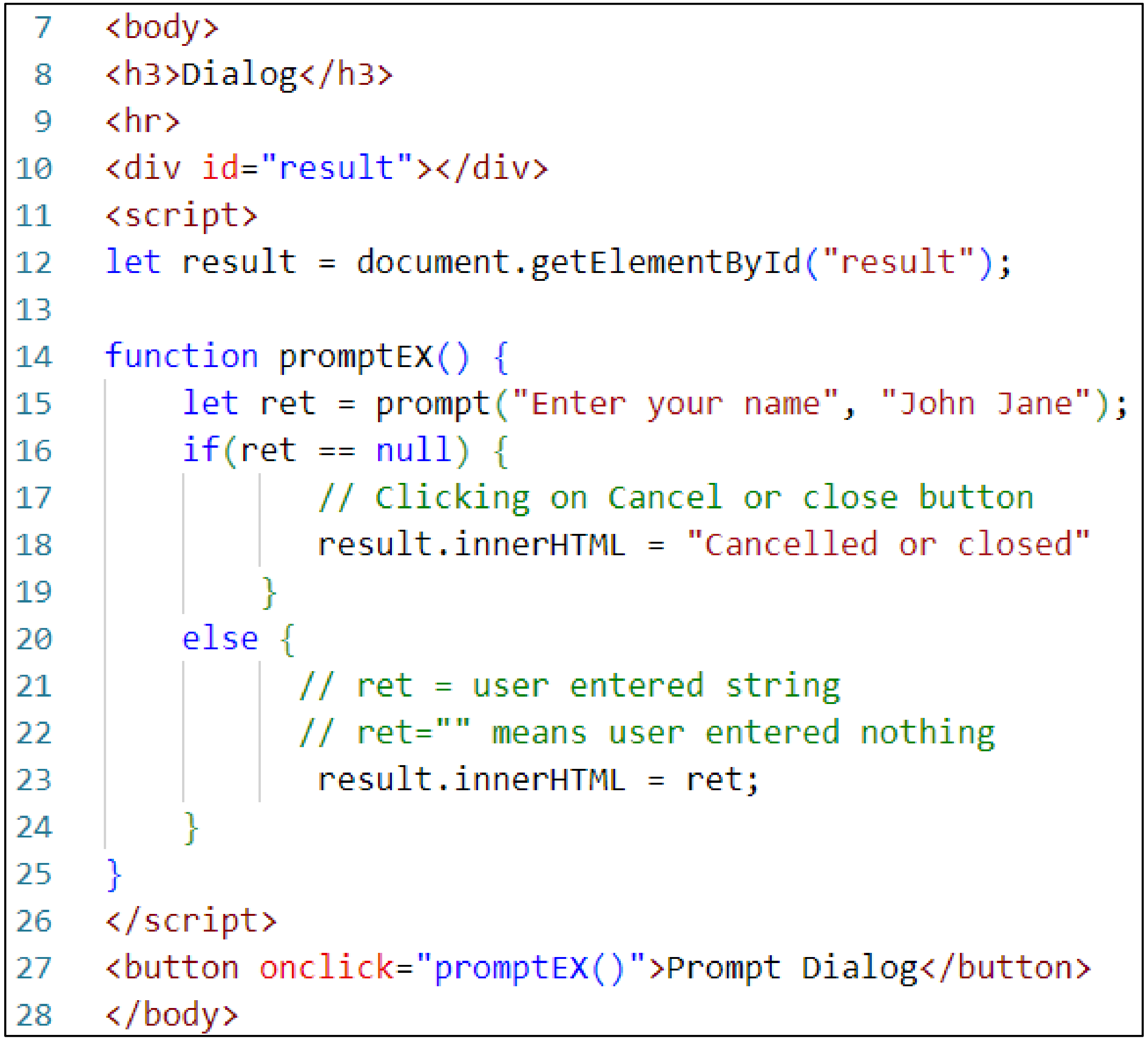
When ‘OK’ is clicked, “else” block will be executed

Message to display

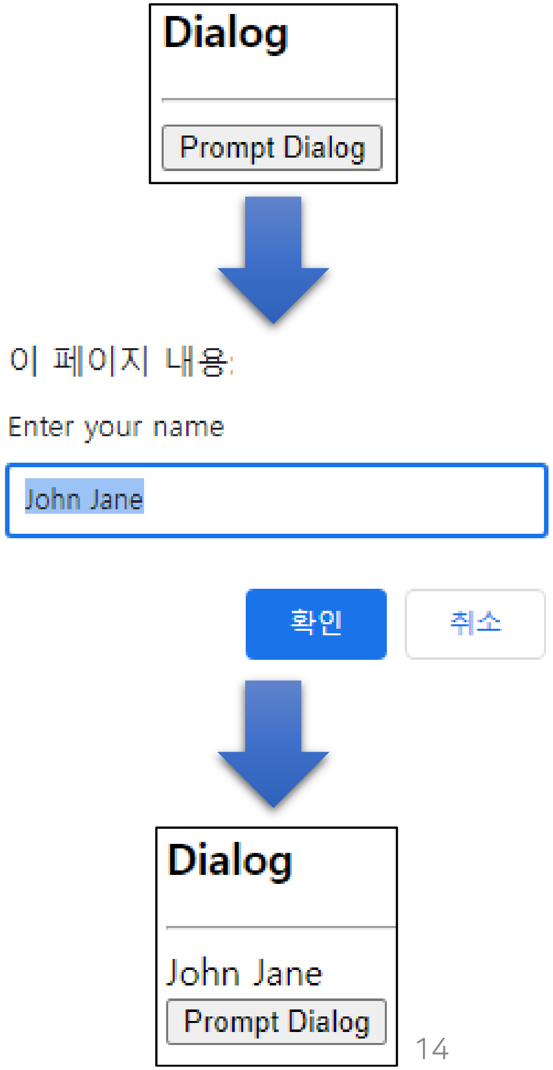
Default value

JavaScript (JS)

- JS dialog : (1) “prompt” to get user input
  - Complete code:



```
<body>
<h3>Dialog</h3>
<hr>
<div id="result"></div>
<script>
let result = document.getElementById("result");

function promptEX() {
    let ret = prompt("Enter your name", "John Jane");
    if(ret == null) {
           // Clicking on Cancel or close button
           result.innerHTML = "Cancelled or closed"
        }
    else {
          // ret = user entered string
          // ret="" means user entered nothing
           result.innerHTML = ret;
    }
}
</script>
<button onclick="promptEX()">Prompt Dialog</button>
</body>
```



Dialog

이 페이지 내용:

Enter your name

John Jane

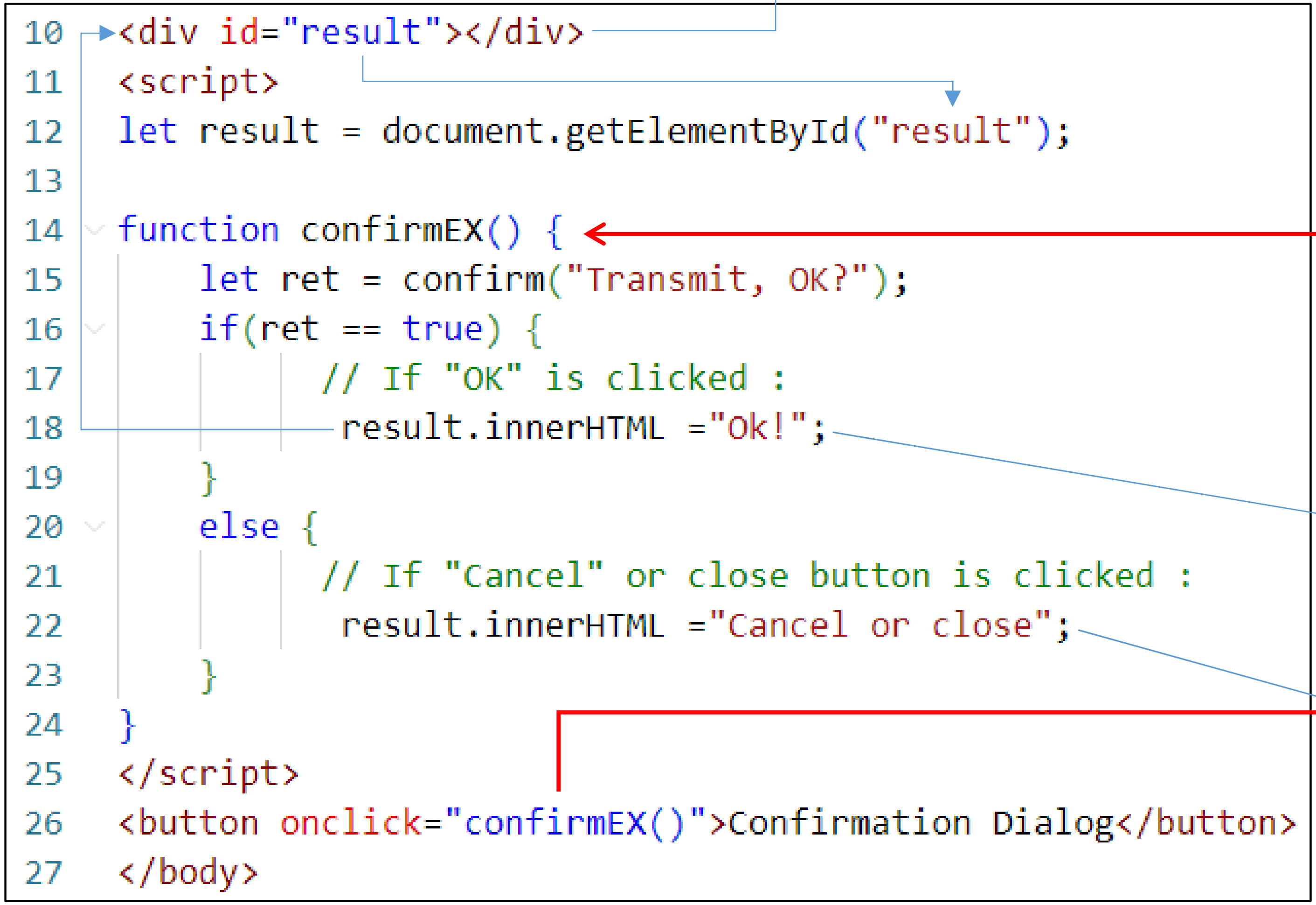
Dialog

John Jane

JavaScript (JS)

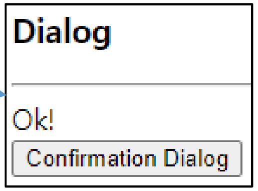
- JS dialog : (2) “confirm” dialog to get YES/NO-type user’s input

Eventually, this part of the code will be replaced by the value assigned to result.innerHTML



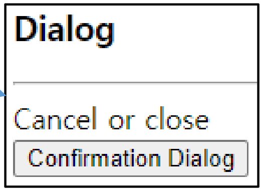
```
<div id="result"></div>
<script>
let result = document.getElementById("result");

function confirmEX() {
    let ret = confirm("Transmit, OK?");
    if(ret == true) {
          // If "OK" is clicked :
           result.innerHTML ="Ok!";
    }
    else {
          // If "Cancel" or close button is clicked :
           result.innerHTML ="Cancel or close";
    }
}
</script>
<button onclick="confirmEX()">Confirmation Dialog</button>
</body>
```

When the button is clicked, confirmEX() function will be executed



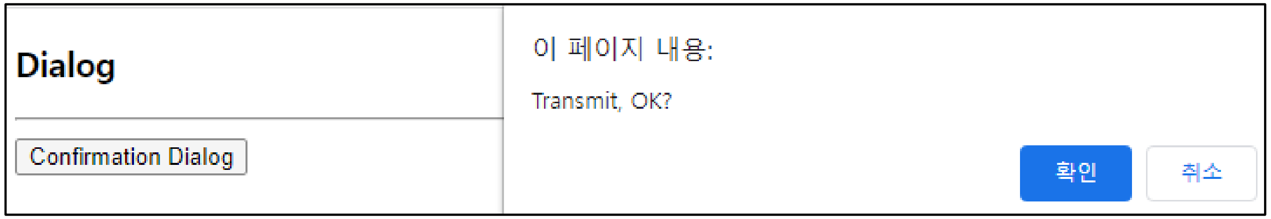
Dialog

Ok!



Dialog

Cancel or close



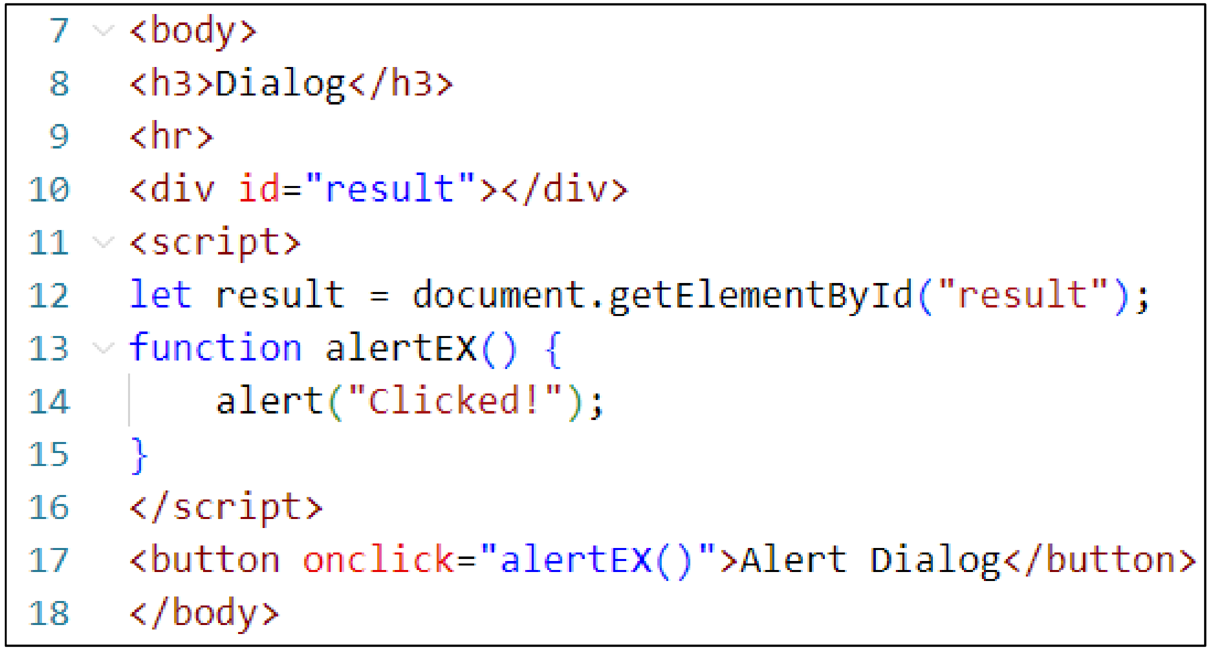
Dialog

이 페이지 내용:

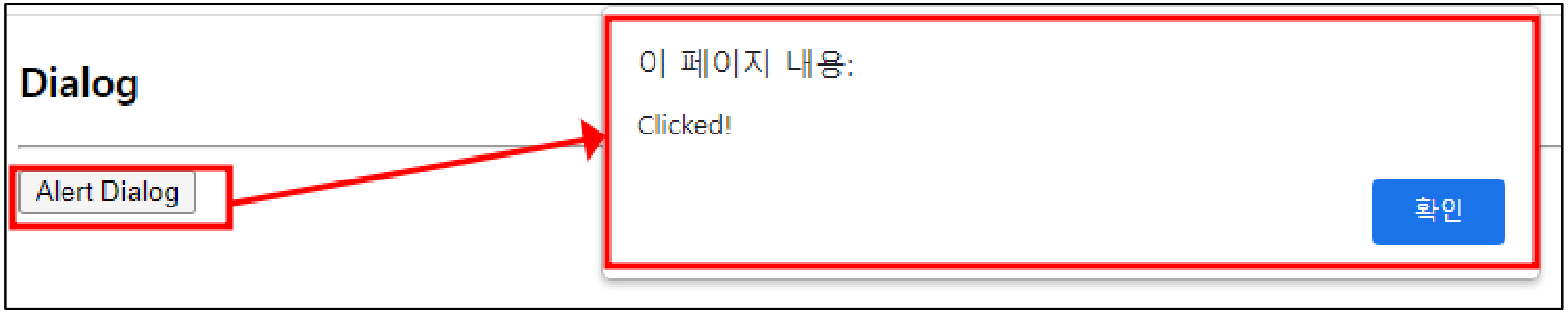
Transmit, OK?

JavaScript (JS)

- JS dialog : (3) “alert” dialog to show a message



```
<body>
<h3>Dialog</h3>
<hr>
<div id="result"></div>
<script>
let result = document.getElementById("result");
function alertEX() {
    alert("Clicked!");
}
</script>
<button onclick="alertEX()">Alert Dialog</button>
</body>
```



Dialog

이 페이지 내용:

Clicked!

JavaScript (JS)

- JS identifier
  - In JS, we can declare and use variables, constants and functions.
  - To correctly distinguish one particular variable/constant/function from the rest, a unique identifier (= name) should be given
  - Rules for making identifiers
    - First letter should be alphabet (A-Z, a-z), underscore(_) or $
    - Second to the last letters should be alphabet (A-Z, a-z), underscore(_), $ or 0-9
    - Case-sensitive! That is, myHome and myhome are different
    - JS reserved keywords cannot be used for identifiers such as if, for, null, false, etc
  - Example

```
6variable;      // Invalid : cannot begin with a number
student_ID;     // Valid
_code;          // Valid, but not recommended style
if;             // Invalid : cannot use the reserved keyword
%calc           // Invalid : % is not allowed for identifier
bar, Bar;       // Balid and two identifiers are differently handled
```

JavaScript (JS)

- Statement
  - In a computer programming language, a statement is a line of code commanding a task
  - A statement can be a single line, or takes up multiple lines

a statement, single line	a statement, multiline

`var mySum = x + y;`	`var mySum = x`
	`            + y;`

  - Each statement is terminated by semicolon (;)
  - There can be multiple statements in a single line, but not recommended

```
var x = 10; var y = 20; var mySum = x + y;
```

```
i = i + 1                       // (O) if there is a single statement on a single line,
                                   semicolon can be omitted, but not recommended
j = j + 1;                      // (O)
k = k + 1; m = m + 1;           // (O) if properly separated by semicolon between statements,
                                   there can be multiple statements on a single line
n = n + 1 p = p + 1;            // (x) a semicolon is missing between the two statements
```

JavaScript (JS)

- Comments
  - a programmer-readable explanation or annotation in the source code of a computer program
  - added with the purpose of making the source code easier for humans to understand, and is ignored by interpreter
  - In short, comments do not do anything. They're only for those who read the code

- Comments : example

```
// a single-line comment, taking up to the end of this line
```

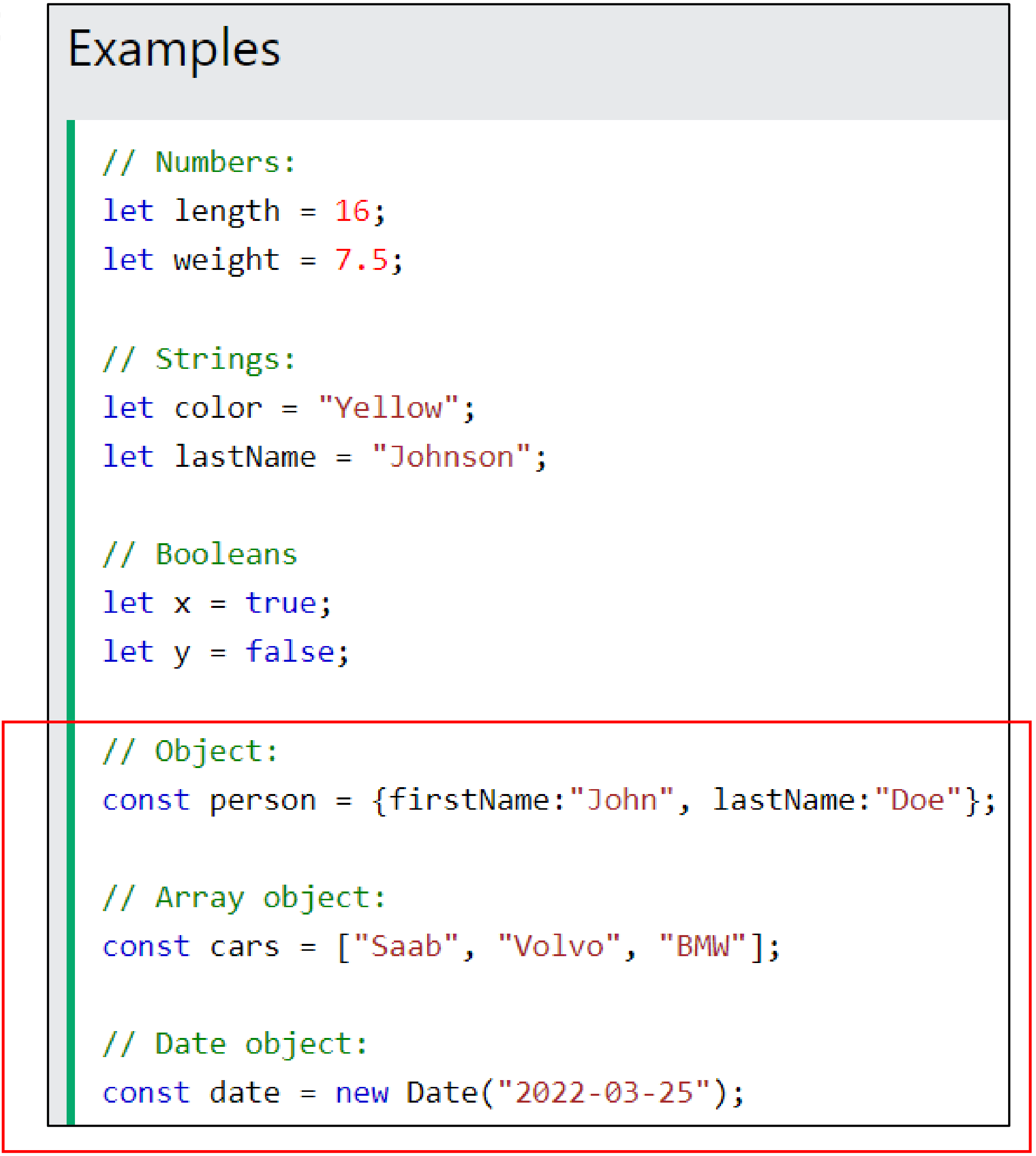
```
/*
  - multi-line comment
  - useful when the comment gets long..
  - good luck
*/
```

JavaScript (JS)

- Data types are the types of values that JS can handle
  - Numeric data type : integers, real numbers, … (e.g., 11, 22.5, …)
  - Logical/Boolean data type : logical, binary values (e.g., true, false)
  - String data type : “Hello world”, “A”
    - JS has no character type, and thus, both a single character and a sequence of characters are treated as a string
  - Object reference data type : referencing or pointing to an object
    - We’ll cover the topic soon
  - null : special, reserved keyword indicating an absence of a value

JavaScript (JS)

- To be precise, JS has 8 data types:
  - 1. String
    2. Number
    3. Bigint
    4. Boolean
    5. Undefined
    6. Null
    7. Symbol
    8. Object
- The object datatype can contain:
  - 1. An object
    2. An array
    3. A date

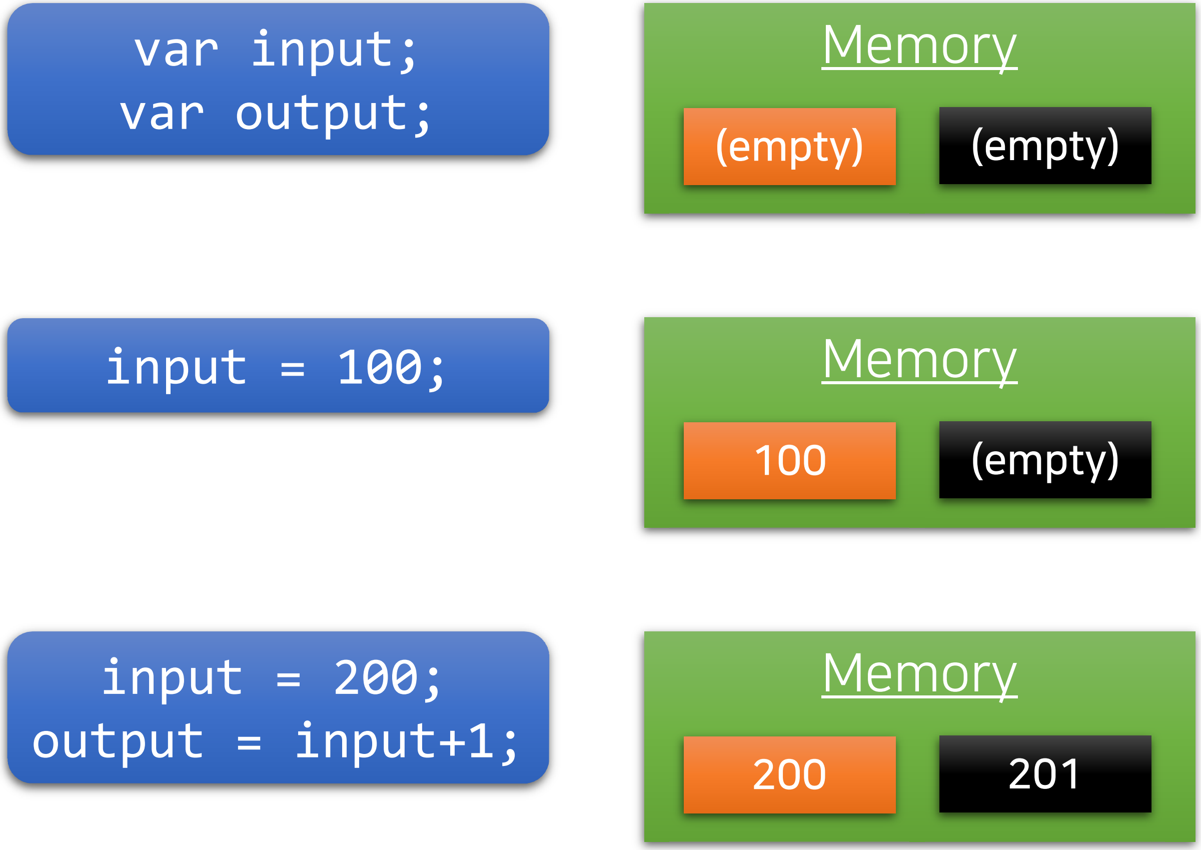


Examples

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

JavaScript (JS)

- Variables are containers, used to store data during the runtime of the JS code
- When a variable is declared (= created), a certain space is allocated in memory, and the value therein can be changed during the runtime
- Each variable has a unique identifier (= name)



```
var input;
var output;
```

Memory

(empty) (empty)

```
input = 100;
```

Memory

100 (empty)

```
input = 200;
output = input+1;
```

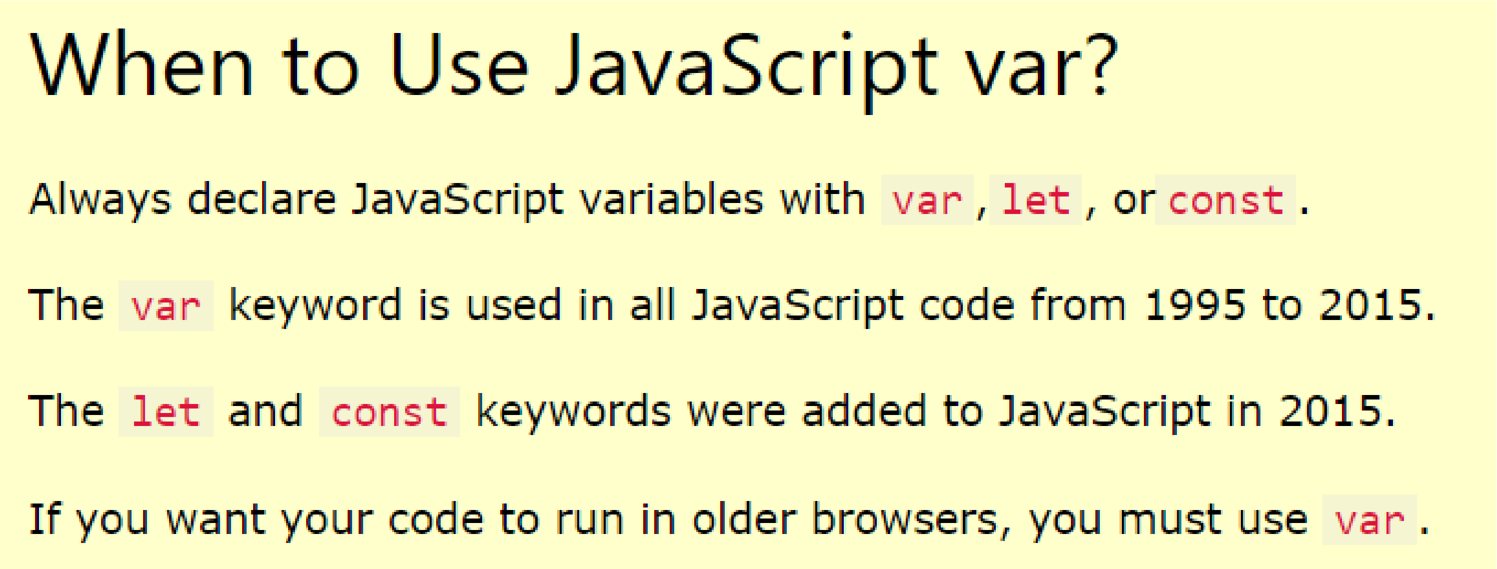
Memory

200 201

JavaScript (JS)

- Four ways to declare (= create) variables

var	Declare variables, and the stored values can be changed during runtime of the JS code
let	
const	Declare constant variable, and the stored value cannot be changed.
(nothing)	Without var, let, const, a variable can still be created



When to Use JavaScript var?

Always declare JavaScript variables with `var`, `let`, or `const`.

The `var` keyword is used in all JavaScript code from 1995 to 2015.

The `let` and `const` keywords were added to JavaScript in 2015.

If you want your code to run in older browsers, you must use `var`.

JavaScript (JS)

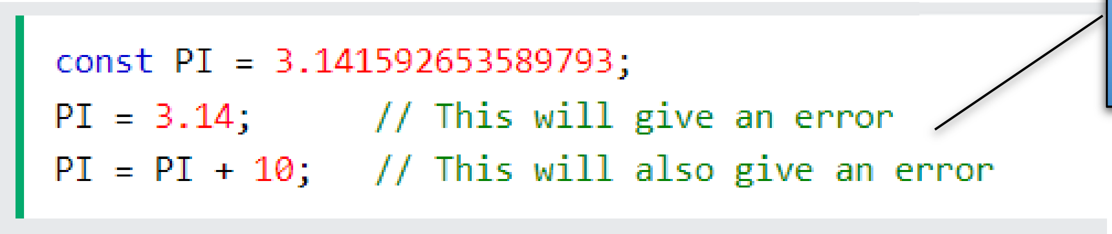
- Declaring variables using var : var identifier [= initial value];

```
var score;                 // declare variable “score”
var year, month, day;      // declare three variables: year, month, day
var address = “Busan”;     // declare variable “address”, and initialize with “Busan”
```

- Declaring variables using let : let identifier [= initial value];

```
let score;                 // declare variable “score”
let year, month, day;      // declare three variables: year, month, day
let address = “Busan”;     // declare variable “address”, and initialize with “Busan”
```

- Declaring constant variables using const



```
const PI = 3.141592653589793;
PI = 3.14;       // This will give an error
PI = PI + 10;    // This will also give an error
```

“const” type variables do not allow value changes

- Declaring variables without using any of above

```
age = 21;        // a variable is declared and initialized without var/let
```

JavaScript (JS)

- More about variables and assigning values to variables
  - When declaring a variable, one do not need to consider the data type of the variable

JavaScript	General programming language (specifying data type)
`let score = 5;`	`int score = 5;`
`let pi = 3.14;`	`float pi = 3.14;`
`let name = “CSE”;`	`String name = “CSE”;`

  - JS variables can be used to store different data types dynamically

JavaScript	General programming language

`let value = 66.8; // save real number`
`value = "hello world“; // save string`
	`double value = 66.8;`
	`value = "hello world“; // not allowed`

JavaScript (JS)

- Life and scope of variables
  - life : creating and deletion of variables
  - scope : the area of the program where the variables can be accessed after its declaration

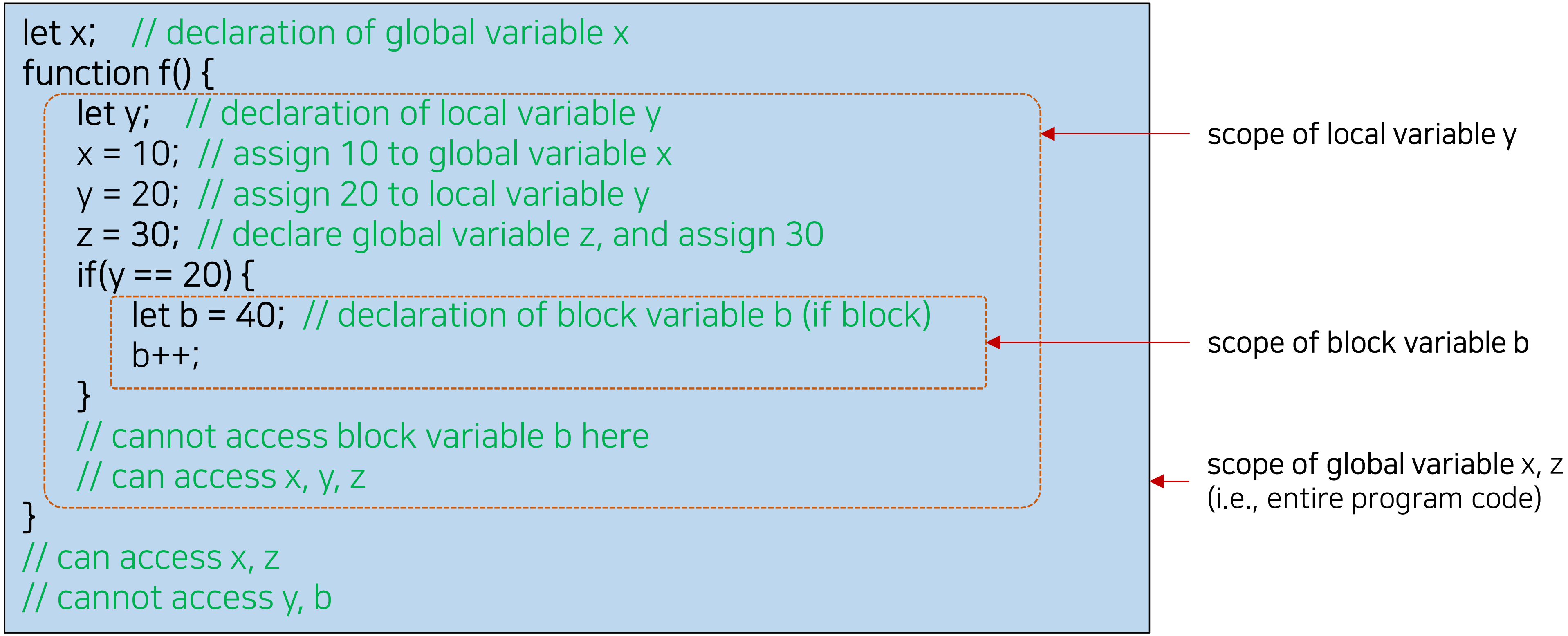
JavaScript (JS)

- Three scopes of variables
  - Variables are categorized into the following three

	Declaration	Scope	Life
Global variable	• Declared outside function, or • Declared anywhere without var/let	Can be accessed anywhere	• Created when the program begins (i.e., web page loaded) • Deleted when the program terminates (i.e., web page closes)
Local variable	• Declared with let, inside a function	Can be accessed within the function	• Created when the function begins • Deleted when the function ends
Block variable	• Declared with let, inside a block (e.g., if, while, for, etc.)	Can be accessed within the block	• Created when the block begins • Deleted when the block ends

JavaScript (JS)

- Three scopes of variables
  - Example:



```
let x;    // declaration of global variable x
function f() {
    let y;    // declaration of local variable y
    x = 10;  // assign 10 to global variable x
    y = 20;  // assign 20 to local variable y
    z = 30;  // declare global variable z, and assign 30
    if(y == 20) {
        let b = 40;  // declaration of block variable b (if block)
        b++;
    }
    // cannot access block variable b here
    // can access x, y, z
}
// can access x, z
// cannot access y, b
```

scope of local variable y

scope of block variable b

scope of global variable x, z
(i.e., entire program code)

JavaScript (JS)

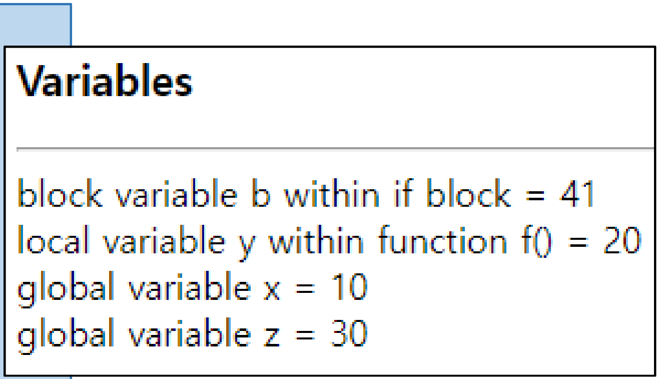
- When the same identifier is used for global and local variable
  - accessing global variable : use prefix “this.”
  - accessing local variable : use its identifier as it is
  - Example:

```
var x;          // global variable
function f() {
   var x;       // local variable
   x = 1;       // assign 1 to local variable x
   this.x = 100; // assign 100 to global variable x
}
```

  - Note
    - “this” cannot be used for a global variable declared with let

JavaScript (JS)

```
<!DOCTYPE html>
<html>
<head><meta charset="utf-8"><title>Variable</title></head>
<body>
<h3> Variable </h3>
<hr>
<script>
let x; // declaring global variable x
function f() {
    let y; // declaring local variable y within f()
    x = 10; // assigning 10 to global variable x
    y = 20; // assigning 20 to local variable y
    z = 30; // declaring global variable z, and assign 30
    if(y == 20) {
        let b = 40; // declaration of block variable b (if-block)
        b++;
        document.write(“block variable b within if block = " + b + "<br>");
    }
    // cannot access block variable b here
    // can access x, y, z
    document.write(“local variable y within function f() y = " + y + "<br>");
}

f(); // calling function f()
document.write(“global variable x = " + x + "<br>");
document.write(“local variable z = " + z);
// can access x, z here, and cannot access local variable y and block variable b
</script>
</body></html>
```



Variables

block variable b within if block = 41
local variable y within function f() = 20
global variable x = 10
global variable z = 30

scope of block variable b

scope of local variable y

scope of global variable x, z
(i.e., entire program)

THE END