1420-7001

웹 프로그래밍

Web Programming

By

Dr. Abdul Majeed (조교수)

1st Semester, 2025

가천대학교
Gachon University

IT융합대학 컴퓨터공학부(컴퓨터공학전공)

Summary of the Previous Lesson

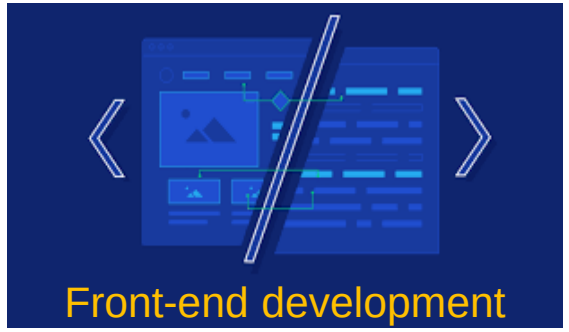
Part I

Exam Solutions
- ❑ MCQs
- ❑ Short Questions
- ❑ Etc.

Part II

Project work
- ❑ Ideas collection
- ❑ Project presentation

Course contents
- ❑ JS
- ❑ PhP
- ❑ Databases.



Part III

Displaying JS Outputs
- ❑ Window alert
- ❑ Inner Html
- ❑ Console.log
- ❑ Print
- ❑ Alert

Part IV

Basics of JS
- ❑ Variables
- ❑ Operators
- ❑ Keywords
- ❑ Comments
- ❑ Etc.

Note: Please execute all codes at least once on your computers.

Part I

Advanced JS Codes

Web Programming

Part I (a)

JS Functions

Web Programming

JS Coding Concepts-JS Function

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).

```html
<html>
<body>
<h1>JavaScript Functions</h1>

<p>Call a function which performs a calculation and returns the result:</p>

<p id="demo"></p>

<script>
function myFunction(p1, p2) {
  return p1 * p2;
}

let result = myFunction(4, 3);
document.getElementById("demo").innerHTML = result;
</script>

</body>
</html>
```

JavaScript Functions

Call a function which performs a calculation and returns the result:

12

JS Coding Concepts-JS Function

- A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: **(*parameter1, parameter2, ...*)**
- The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Important points

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

JS Coding Concepts-JS Function Call

- The code inside the function will execute when "something" **invokes** (calls) the function:
  - ✓ When an event occurs (when a user clicks a button)
  - ✓ When it is invoked (called) from JavaScript code
  - ✓ Automatically (self-invoked) (onPageLoad() or simply onload())

JS Coding Concepts-JS Function Return

- When JavaScript reaches a return statement, the function will stop executing.

- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

- Functions often compute a **return value**. The return value is "returned" back to the "caller":

JS Coding Example-JS Function Return

```html
<html>
<body>
<h1>JavaScript Functions</h1>

<p>Call a function which performs a calculation and returns the result:</p>

<p id="demo"></p>

<script>
function myFunction(p1, p2) {
  return p1 * p2;
}

let result = myFunction(4, 3);
document.getElementById("demo").innerHTML = result;
</script>

</body>
</html>
```

JavaScript Functions

Call a function which performs a calculation and returns the result:

12

JS Coding Concepts-JS Functions

⌘ With functions you can reuse code.

⌘ You can write code that can be used many times.

⌘ You can use the same code with different arguments, to produce different results.

JS Coding Example-JS Function Call: () Operator

- The () operator invokes (calls) the function

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Functions</h1>

<p>Invoke (call) a function that converts from Fahrenheit to Celsius:</p>
<p id="demo"></p>

<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}

let value = toCelsius(77);
document.getElementById("demo").innerHTML = value;
</script>

</body>
</html>
```

JavaScript Functions

Invoke (call) a function that converts from Fahrenheit to Celsius:

25

JS Coding Example-JS Function Call: () Operator

⌘ The () operator invokes (calls) the function.
⌘ Wrong call of the function.

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Functions</h1>

<p>Invoke (call) a function to convert from Fahrenheit to Celsius:</p>
<p id="demo"></p>

<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}

let value = toCelsius();
document.getElementById("demo").innerHTML = value;
</script>

</body>
</html>
```

JavaScript Functions

Invoke (call) a function to convert from Fahrenheit to Celsius:

NaN

JS Coding Example-JS Function Call: () Operator

⌘ Accessing a function without () returns the function and not the function result.

```html
<html>
<body>

<h1>JavaScript Functions</h1>

<p>Accessing a function without () returns the function and not the function result:</p>
<p id="demo"></p>

<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}

let value = toCelsius;
document.getElementById("demo").innerHTML = value;
</script>

</body>
</html>
```

JavaScript Functions

Accessing a function without () returns the function and not the function result:

function toCelsius(f) { return (5/9) * (f-32); }

As you see from the examples above, toCelsius refers to the function object, and toCelsius() refers to the function result.

JS Coding Example-JS Function as Variable

- Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

```html
<html>
<body>

<h1>JavaScript Functions</h1>
<p>Using a function as a variable:</p>

<p id="demo"></p>

<script>
let text = "The temperature is " + toCelsius(77) + " Celsius.";
document.getElementById("demo").innerHTML = text;

function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
</script>

</body>
</html>
```

JavaScript Functions

Using a function as a variable:

The temperature is 25 Celsius.

JS Coding Example-JS Function-Local Variables

- ⌘ Variables declared within a JavaScript function, become **LOCAL** to the function.
- ⌘ Local variables can only be accessed from within the function.

```html
<html>
<body>
<h1>JavaScript Functions</h1>
<p>Outside myFunction() carName is undefined.</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
let text = "Outside: " + typeof carName;
document.getElementById("demo1").innerHTML = text;

function myFunction() {
  let carName = "Volvo";
  let text = "Inside: " + typeof carName + " " + carName;
  document.getElementById("demo2").innerHTML = text;
}
myFunction();
</script>
</body>
</html>
```

JavaScript Functions

Outside myFunction() carName is undefined.

Outside: undefined

Inside: string Volvo

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.
Local variables are created when a function starts, and deleted when the function is completed.

Part I (b)

JS Objects

Web Programming

JS Coding Concepts-JS Objects- Objects in real life

- In real life, a car is an **object**.
- A car has **properties** like weight and color, and **methods** like start and stop

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

- ✓ All cars have the same **properties**, but the property **values** differ from car to car.
- ✓ All cars have the same **methods**, but the methods are performed **at different times**.

JS Coding Concepts-JS Objects- Difference between variables & Objects

- You have already learned that JavaScript variables are containers for data values.
- This code assigns a **simple value** (Fiat) to a **variable** named car

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p id="demo"></p>

<script>
// Create and display a variable:
let car = "Fiat";
document.getElementById("demo").innerHTML = car;
</script>

</body>
</html>
```

JavaScript Variables

Fiat

- ✓ Objects are variables too. But objects can contain many values.
- ✓ This code assigns **many values** (Fiat, 500, white) to a **variable** named car

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
// Create an object:
const car = {type:"Fiat", model:"500", color:"white"};

// Display some data from the object:
document.getElementById("demo").innerHTML = "The car type is " + car.type;
</script>

</body>
</html>
```

JavaScript Objects

The car type is Fiat

The values are written as **name:value** pairs (name and value separated by a colon).

JS Coding Concepts-JS Objects- Objects Definition

⌘ You define (and create) a JavaScript object with an object literal.

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

```
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
// Create an object:
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

JavaScript Objects

John is 50 years old.

JS Coding Concepts-JS Objects- Objects Definition

- Spaces and line breaks are not important. An object definition can span multiple lines:

(1) (2)

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

```html
<html>
<body>

<h2>JavaScript Objects</h2>

<p id="demo"></p>

<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

JavaScript Objects

John is 50 years old.

JS Coding Concepts-JS Objects- Accessing Properties

⌘ The object properties can be accessed in following two ways.

(1)

```
objectName.propertyName
```

(2)

```
objectName["propertyName"]
```

JS Coding Concepts-JS Objects- Accessing Properties (1st way)

⌘ You can access object properties in two ways. (1)

```
objectName.propertyName
```

```html
<html>
<body>
<h2>JavaScript Objects</h2>
<p>There are two different ways to access an object property.</p>
<p>You can use person.property or person["property"].</p>
<p id="demo"></p>
<script>
// Create an object:
const person = {
  firstName: "John",
  lastName : "Doe",
  id       :  5566
};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " " + person.lastName;
</script>

</body>
</html>
```

JavaScript Objects

There are two different ways to access an object property.

You can use person.property or person["property"].

John Doe

JS Coding Concepts-JS Objects- Accessing Properties (2nd way)

⌘ You can access object properties in two ways.

(2)

objectName["propertyName"]

```
<html>
<body>
<h2>JavaScript Objects</h2>
<p>There are two different ways to access an object property.</p>
<p>You can use person.property or person["property"].</p>
<p id="demo"></p>
<script>
// Create an object:
const person = {
  firstName: "John",
  lastName : "Doe",
  id     :  5566
};

// Display some data from the object:
document.getElementById("demo").innerHTML =
person["firstName"] + " " + person["lastName"];
</script>

</body>
</html>
```

JavaScript Objects

There are two different ways to access an object property.

You can use person.property or person["property"].

John Doe

JavaScript objects are containers for **named values** called properties.

JS Coding Concepts-JS Objects- Object Methods

- ⌘ Objects can also have **methods**.
- ⌘ Methods are **actions** that can be performed on objects.
- ⌘ Methods are stored in properties as **function definitions**

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

A method is a function stored as a property.

JS Coding Concepts-JS Objects- Object Methods

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

JavaScript Objects

An object method is a function definition, stored as a property value.

John Doe

```html
<html>
<body>
<h2>JavaScript Objects</h2>
<p>An object method is a function definition, stored as a property value.</p>
<p id="demo"></p>

<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};

// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>

</body>
</html>
```

JS Coding Concepts-JS Objects- Object Methods

In the example on previous slide, this refers to the **person object**:

- **this.firstName** means the **firstName** property of **person**.
- **this.lastName** means the **lastName** property of **person**.

Part I (c)

JS Events

Web Programming

JS Coding Concepts-JS Events [HTML+JS]

- ⌘ HTML events are "**things**" that happen to HTML elements.
- ⌘ When JavaScript is used in HTML pages, JavaScript can "**react**" on these events.

⌘ An HTML event can be something the browser does, or something a user does.
⌘ Here are some examples of HTML events:
- √ An HTML web page has finished loading
- √ An HTML input field was changed
- √ An HTML button was clicked

⌘ Often, when events happen, you may want to do something.
⌘ JavaScript lets you execute code when events are detected.
⌘ HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

JS Coding Example-JS Events [HTML+JS]

⌘ In the following example, an onclick attribute (with code), is added to a <button> element:

```
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Attribute</h2>

<button onclick="document.getElementById('demo').innerHTML=Date()">The time is?</button>

<p id="demo"></p>

</body>
</html>
```

In the example above, the JavaScript code changes the content of the element with id="demo".

JavaScript HTML Events

The onclick Attribute

The time is?

JavaScript HTML Events

The onclick Attribute

The time is?

Sat May 04 2024 14:58:32 GMT+0900 (Korean Standard Time)

JS Coding Example-JS Events [HTML+JS]

In this example, the code changes the content of its own element (using **this**.innerHTML):

```html
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Attribute</h2>

<button onclick="this.innerHTML=Date()">The time is?</button>

</body>
</html>
```

JavaScript HTML Events

The onclick Attribute

The time is?

JavaScript HTML Events

The onclick Attribute

Sat May 04 2024 15:01:09 GMT+0900 (Korean Standard Time)

JS Coding Example-JS Events [Calling Functions]

✓ JavaScript code is often several lines long. It is more common to see event attributes calling functions.

```html
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Attribute</h2>

<p>Click the button to display the date.</p>
<button onclick="displayDate()">The time is?</button>

<script>
function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>

<p id="demo"></p>

</body>
</html>
```

JavaScript HTML Events

The onclick Attribute

Click the button to display the date.

The time is?

JavaScript HTML Events

The onclick Attribute

Click the button to display the date.

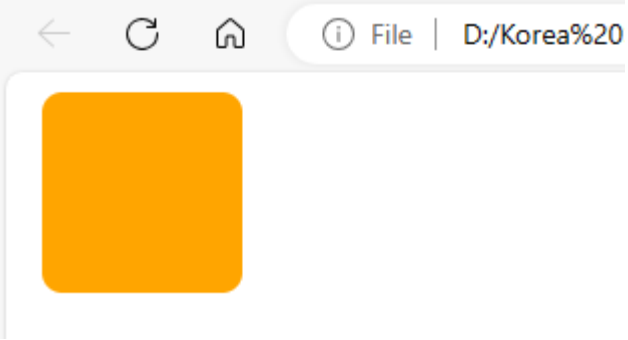
The time is?

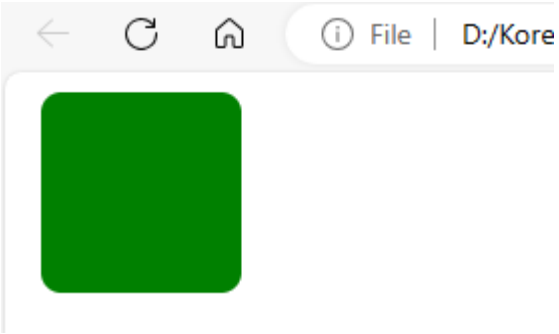
Sat May 04 2024 15:05:02 GMT+0900 (Korean Standard Time)

JS Coding Example-JS Events [OnMouseOver]

✓ In the below example, the colour of the object will change when mouse if over it.

```html
<html>
  <head>
  <style>
  div {
  width: 100px;
  height: 100px;
  background-color: orange;
  margin: 10px;
  border-radius: 10px;
}
  </style>
  </head>
  <body>
    <script>
      function changeColor() {
        document.getElementById("my-div").style.backgroundColor = "green";
      }
    </script>
    <div id="my-div" onmouseover="changeColor()"></div>
  </body>
</html>
```





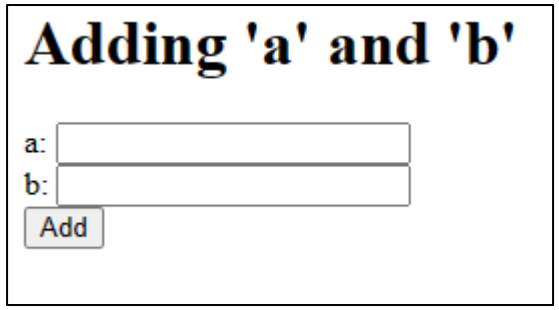
JS Coding Example-JS Events [HTML and JS]

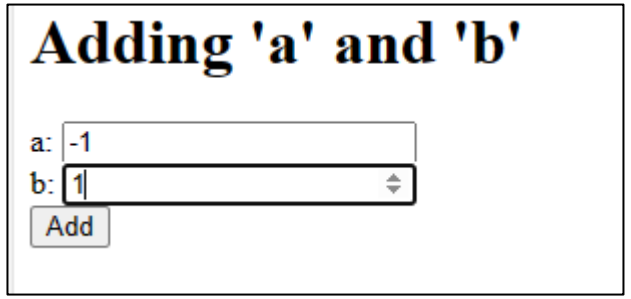
- ✓ Passing HTML input value as a JavaScript Function Parameter.

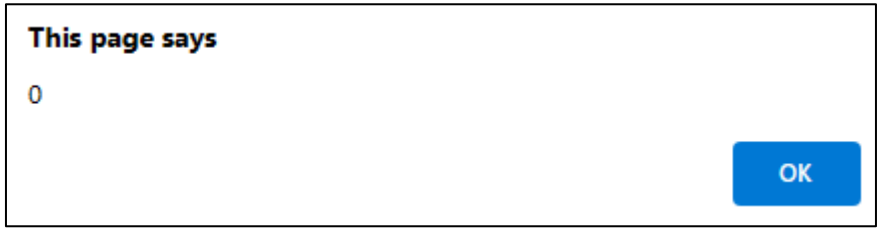
```html
<html>
<body>
  <h1>Adding 'a' and 'b'</h1>

  a: <input type="number" name="a" id="a"><br> b: <input type="number" name="b" id="b"><br>
  <button onclick="add(document.getElementById('a').value,document.getElementById('b').value)">Add</button>

  <script>
    function add(a, b) {
      var sum = parseInt(a, 10) + parseInt(b, 10);
      alert(sum);
    }
  </script>
</body>
</html>
```







JS Coding Concepts-JS Events

⌘ Here is a list of some common HTML events.

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JS Coding Concepts-JS Events Handler

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- ✓ Things that should be done every time a page loads
- ✓ Things that should be done when the page is closed
- ✓ Action that should be performed when a user clicks a button
- ✓ Content that should be verified when a user inputs data
- ✓ And more …

Many different methods can be used to let JavaScript work with events:

- ✓ HTML event attributes can execute JavaScript code directly
- ✓ HTML event attributes can call JavaScript functions
- ✓ You can assign your own event handler functions to HTML elements
- ✓ You can prevent events from being sent or being handled
- ✓ And more …

Please go through events and event handlers in the HTML DOM chapters.

Summary of the Today's Lesson

- JS Functions
  - Without parameter
  - With parameter
  - Function Calls
- JS Objects
  - Creation of objects
  - Accessing object properties
- JS Events
  - onClick()
  - onMouseOver()
  - onMouseout()

```
> response
<· ▼Object
    ▶config: Object
    ▼data: Object
      ▼assignToMap: Object
        ▼123: Array[4]
            0: 1
            1: 2
            2: 3
            3: 4
            length: 4
          ▶__proto__: Array[0]
        ▶345: Array[4]
        ▶678: Array[4]
          ▶__proto__: Object
      ▶__proto__: Object
    ▶headers: (d)
      status: 200
      statusText: "OK"
    ▶__proto__: Object
> |
```