Introduction to JavaScript: Part 1

Introduction to Internet and Web

부산대학교 정보·의생명공학대학
정보컴퓨터공학부

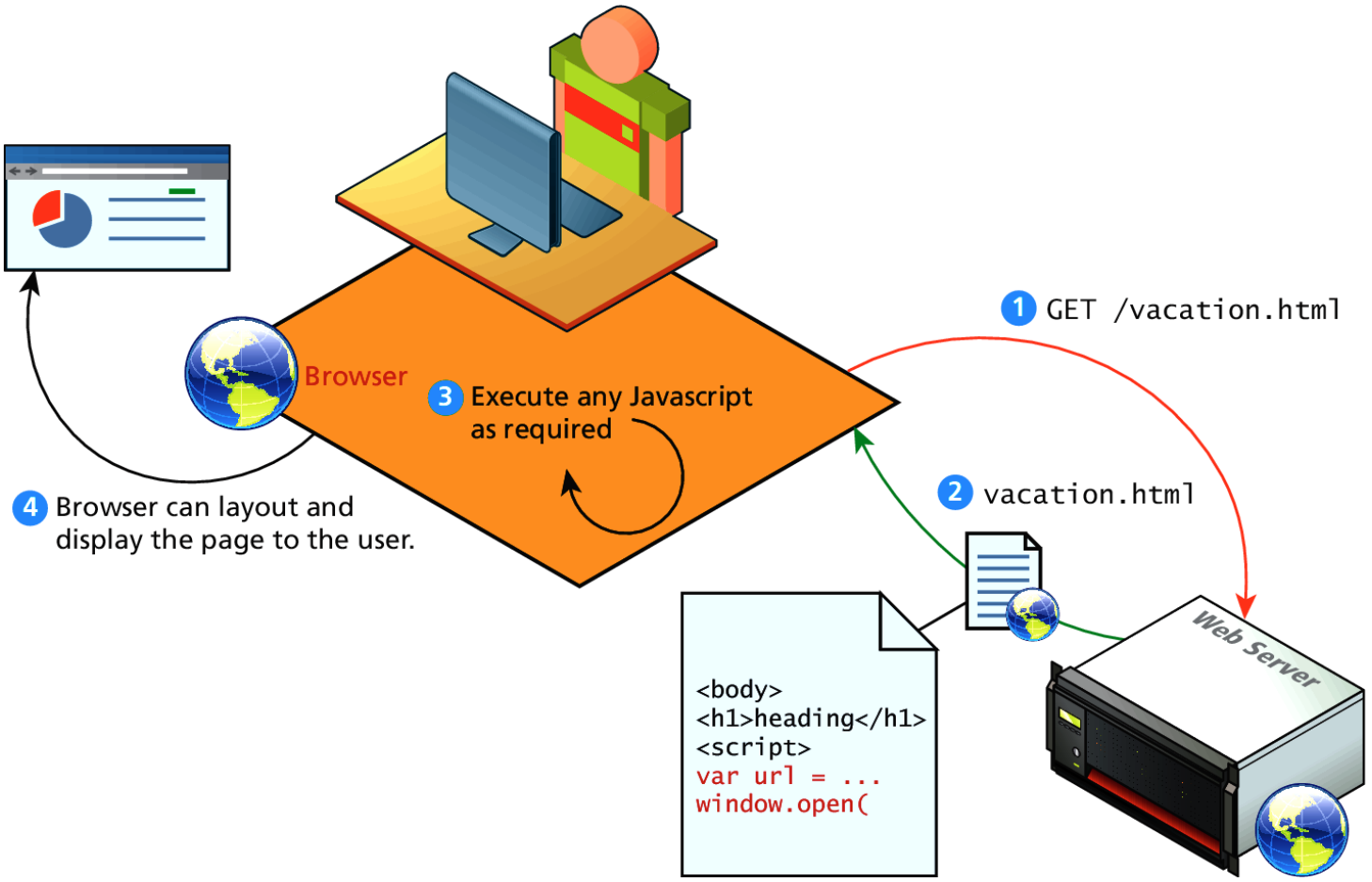
Contents

- ❖ JavaScript Introduction
- ❖ JavaScript Syntax
- ❖ JavaScript Variables
- ❖ JavaScript Functions

JAVASCRIPT INTRODUCTION

Why Study JavaScript?

❖ Client-side Scripting



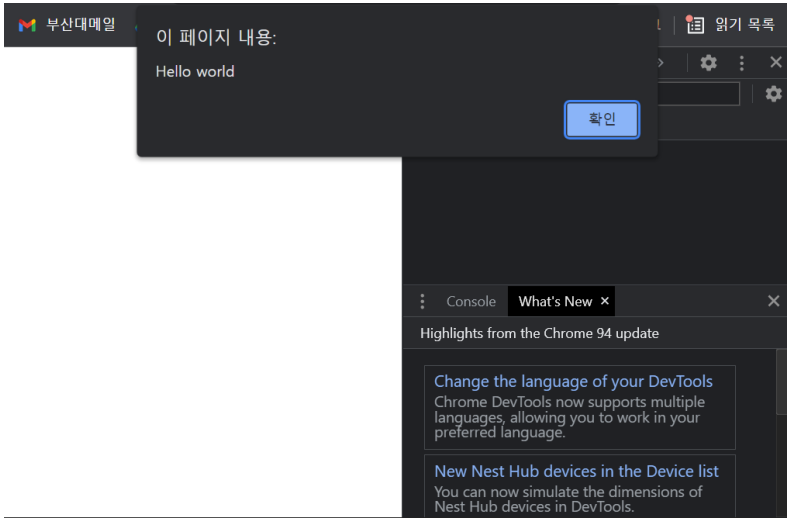
JavaScript Where To

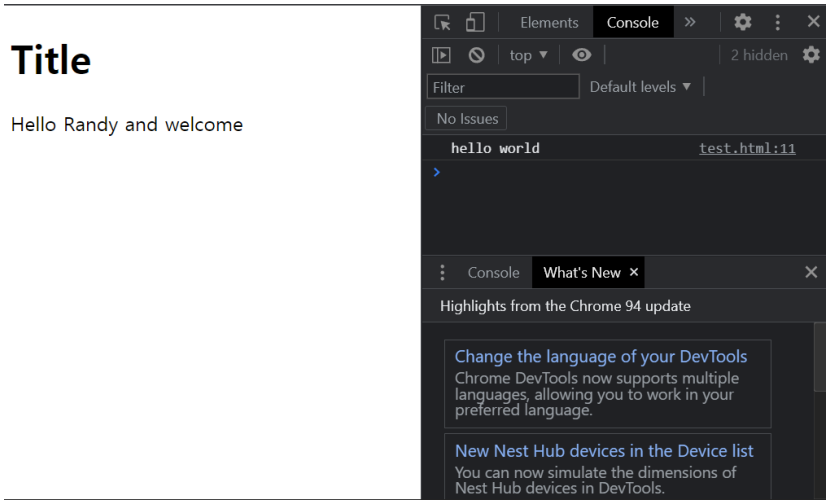
- In HTML, JavaScript code is inserted between <script> and </script> tags.
- You can place any number of scripts in an HTML document.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

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

<h2>JavaScript in Body</h2>

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

<script>
document.getElementById("demo").in
nerHTML = "My First JavaScript";
</script>

</body>
</html>
```

[참조] JavaScript Output

- ❖ alert() Displays content within a pop-up box.
- ❖ console.log() Displays content in the Browser's JavaScript console.
- ❖ document.write() Outputs the content (as markup) directly to the HTML document.
- ❖ document.getElementById("demo").innerHTML Outputs the contents to the specific element.





JavaScript in <head>

- **In this example, a JavaScript function is placed in the <head> section of an HTML page.**
- **The function is invoked when a button is clicked:**
  - A JavaScript function is a block of JavaScript code, that can be executed when "called" for.
  - For example, a function can be called when an event occurs, like when the user clicks a button.

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
  document.getElementById("demo").
innerHTML = "Paragraph changed.";
}
</script>
</head>
<body>

<h2>Demo JavaScript in Head</h2>

<p id="demo">A Paragraph.</p>

<button type="button"
onclick="myFunction()">Try
it</button>

</body>
</html>
```

JavaScript in <body>

- **In this example, a JavaScript function in placed in the <body> section of an HTML page.**
- **The function is invoked when a button is clicked:**
  - A JavaScript function is a block of JavaScript code, that can be executed when "called" for.
  - For example, a function can be called when an event occurs, like when the user clicks a button.

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

<h2>Demo JavaScript in Body</h2>

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

<button type="button"
onclick="myFunction()">Try
it</button>

<script>
function myFunction() {
  document.getElementById("demo").
innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

External JavaScript

❖ Script can also be placed in external files

- External scripts are practical when the same code is used in many different web pages
- JavaScript files have the file extension .js
- To use an external script, put the name of the script file in the src attribute of a <script> tag

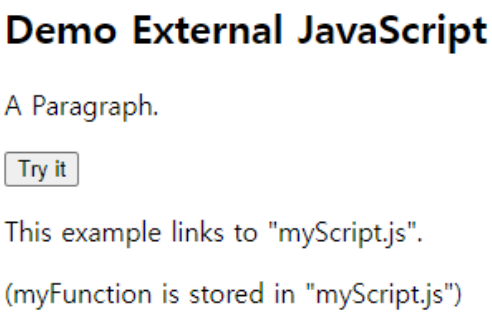
❖ Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

External JavaScript

External file: myScript.js

```
function myFunction() {
 document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```




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

<h2>Demo External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button"
onclick="myFunction()">Try
it</button>

<p>This example links to
"myScript.js".</p>
<p>(myFunction is stored in
"myScript.js")</p>

<script
src="myScript.js"></script>

</body>
</html>
```

External References

- External scripts can be referenced with a full URL or with a path relative to the current web page.
- This example uses a full URL to link to a script:

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

- This example uses a script located in a specified folder on the current web sites:

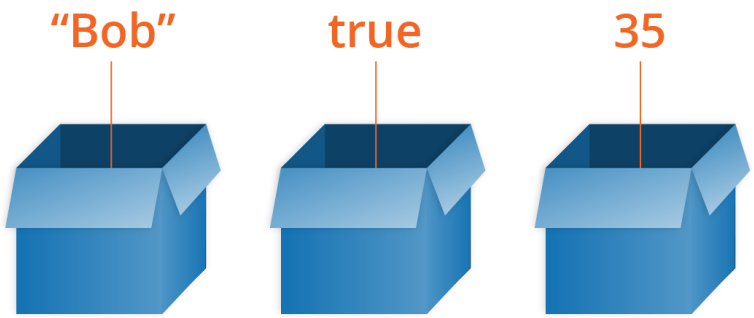
```
<script src="/js/myScript1.js"></script>
```

JAVASCRIPT SYNTAX

JavaScript Values

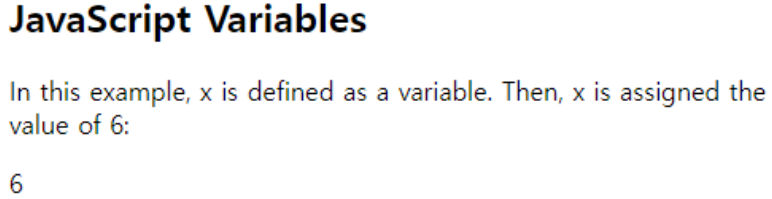
- The JavaScript syntax defines two types of values: Fixed values and variable values.
- Fixed values are called literals.
  - **Numbers** are written with or without decimals: 10.5, 1001
  - **Strings** are text, written within double or single quotes: "John Doe", 'John Deo'
- Variables values are called variables.
  - JavaScript uses the **var, let, const** keyword to declare variables.
  - An equal sign is used to assign values to variables

```
var x;
x = 6;
```



Declaring (Creating) JavaScript Variables

❖ In this example, x is defined as a variable. Then, x is assigned (given) the value 6:




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

<h2>JavaScript Variables</h2>

<p>In this example, x is defined
as a variable.
Then, x is assigned the value of
6:</p>

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

<script>
let x;
x = 6;
document.getElementById("demo").in
nerHTML = x;
</script>

</body>
</html>
```

JavaScript Identifier

- In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels)
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).
- The general rules for constructing names for variables (unique identifiers) are:
  - Names can contain letters, digits, underscores, and dollar signs.
  - Names must begin with a letter
  - Names can also begin with $ and _ (but we will not use it in this tutorial)
  - Names are case sensitive (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names

JavaScript Operators

❖ **Arithmetic operators are used to perform arithmetic on numbers**

❖ **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
```

```
var txt1 = "John";
var txt2 = "Doe";
var txt3 = txt1 + " " + txt2;
```

The result of txt3 will be:

```
John Doe
```

JavaScript Operators

❖ **Assignment operators** assign values to JavaScript variables.

❖ Bit operators work on 32 bits numbers.

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

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

JavaScript Expressions

- **An expression is a combination of values, variables, and operators, which computes to a value.**
- **The computation is called an evaluation.**
  - For example, *5 * 10* evaluates to 50:
- **Expressions can also contain variable values: *x * 10***
- **The values can be of various types, such as numbers and strings.**
  - For example, "John" + " " + "Doe", evaluates to "John Doe":

JavaScript Keywords

❖ JavaScript keywords are used to identify actions to be performed

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

JavaScript Comments

- Not all JavaScript statements are “executed”.
- Code after double slashes // or written /* and */ is treated as a comment
- Comments are ignored, and will not be executed.

```
var x = 5;   // I will be executed

// var x = 6;   I will NOT be executed
```

JavaScript Programs

- **JavaScript program is a list of programming statements.**
  - A computer program is a list of "instructions" to be "executed" by a computer.
  - In a programming language, these programming instructions are called statements.
- **JavaScript programs (and JavaScript statements) are often called JavaScript code**
- **JavaScript statements are composed of:**
  - Values, Operators, Expressions, Keywords, and Comments.
- **The statements are executed, one by one, in the same order as they are written.**
- **Semicolons separate JavaScript statements.**

JAVASCRIPT VARIABLES

Variables

❖ There are 3 ways to declare a JavaScript variable:

- Using var
- Using let
- Using const

❖ Variables in JavaScript are dynamically typed.

- This simplifies variable declarations, since we do not require the familiar data-type identifiers

```
var x = "John Doe";
x = 0;
```

Variables

- Creating a variable in JavaScript is called "declaring" a variable.
- You declare a JavaScript variable with the *var/let/const* keyword:

  var carName;

- After the declaration, the variable has no value (technically it has the value of undefined).
- To assign a value to the variable, use the equal sign:

  carName = "Volvo";

- You can also assign a value to the variable when you declare it:

  var carName = "Volvo";

var

- ❖ Variables defined with var can be Redeclared.
- ❖ Variables defined with var can be used before the declaration (JavaScript Hoisting)
  - Hoisting is JavaScript's default behavior of moving declarations to the top.
- ❖ Variables defined with var doesn't have Block Scope.
  - i.e., global scope

```
var x = "John Doe";
var x = 0;

carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
var carName;
```

let

- Variables defined with let cannot be Redeclared.
- Variables defined with let must be Declared before use.
- Variables defined with let have Block Scope.

```
let y = "John Doe";
let y = 0;
// SyntaxError: 'y' has already been declared

carName = "Saab";
let carName = "Volvo";
// Reference Error: Cannot access 'carName' before initialization
```

Scope

❖ global scope / block scope

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

<h2>Redeclaring a Variable Using var</h2>

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

<script>
var  x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

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

<h2>Redeclaring a Variable Using let</h2>

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

<script>
let  x = 10;
// Here x is 10

{
  let x = 2;
  // Here x is 2
}

// Here x is 10
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Const

- Variables defined with const cannot be Redeclared.
- Variables defined with const cannot be Reassigned.
- Variables defined with const have Block Scope.

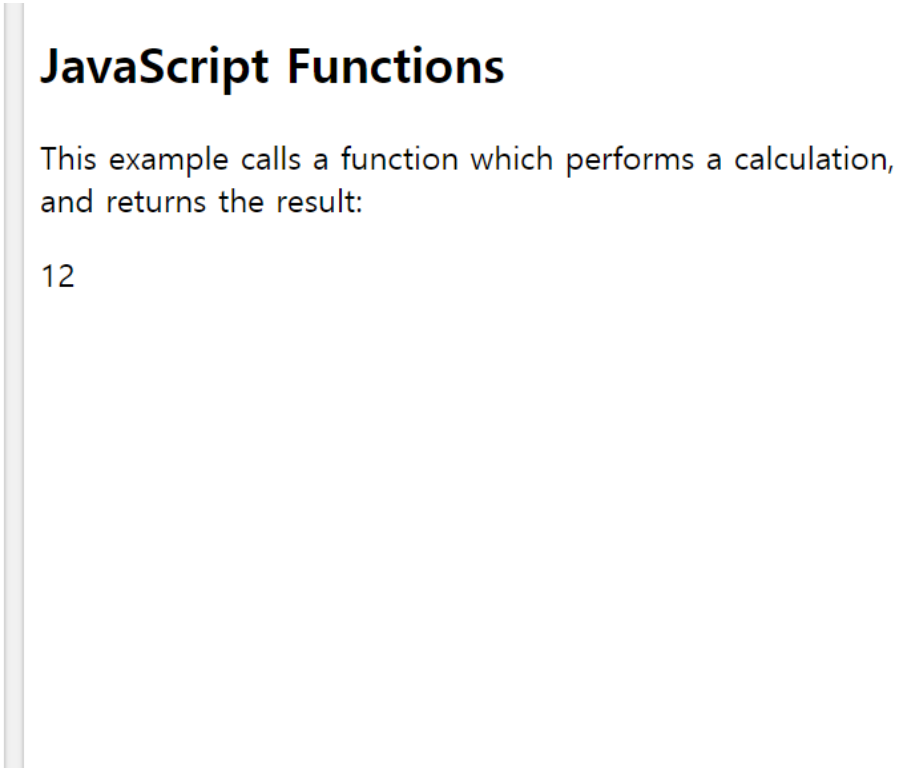
```
const x = "John Doe";   // Allowed
const x = 2;   // Not allowed


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


alert (carName);
const carName = "Volvo";
// ReferenceError: Cannot access 'carName' before initialization
```

JAVASCRIPT FUNCTIONS

JavaScript Functions

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

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

<h2>JavaScript Functions</h2>

<p>This example calls a function which performs a
calculation, and returns the result:</p>

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

<script>
function myFunction(p1, p2) {
  return p1 * p2;
}
document.getElementById("demo").innerHTML =
myFunction(4, 3);
</script>

</body>
</html>
```



JavaScript Function Syntax

❖ **A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().**

- Function name 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
}
```

Function Invocation / Return

- The code inside the function will execute when "something" invokes (calls) the function:
  - e.g., Events, JavaScript code
- 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"

```
<script>
function myFunction(p1, p2) {
  return p1 * p2;
}
document.getElementById("demo").innerHTML =
myFunction(4, 3);
</script>
```

```
<button onclick="displayDate()">The time is?</button>

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

Function Invocation / Return

- Function parameters are listed inside the parentheses () in the fuction definition
- Function arguments are the values receives by the function when it is invoked

Calculate the product of two numbers, and return the result:

```
var x = myFunction(4, 3);   // Function is called, return value will end up in x

function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

Why Functions?

- ❖ You can reuse code: Define the code once, and use it many times
- ❖ You can use the same code many times with different arguments, to produce different results.

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

Local variables

- Variables declared within a JavaScript function, become LOCAL to the function. (Function Scope)
- Local variables can only be accessed from within the function
- 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.
  - The arguments behave as local variables.

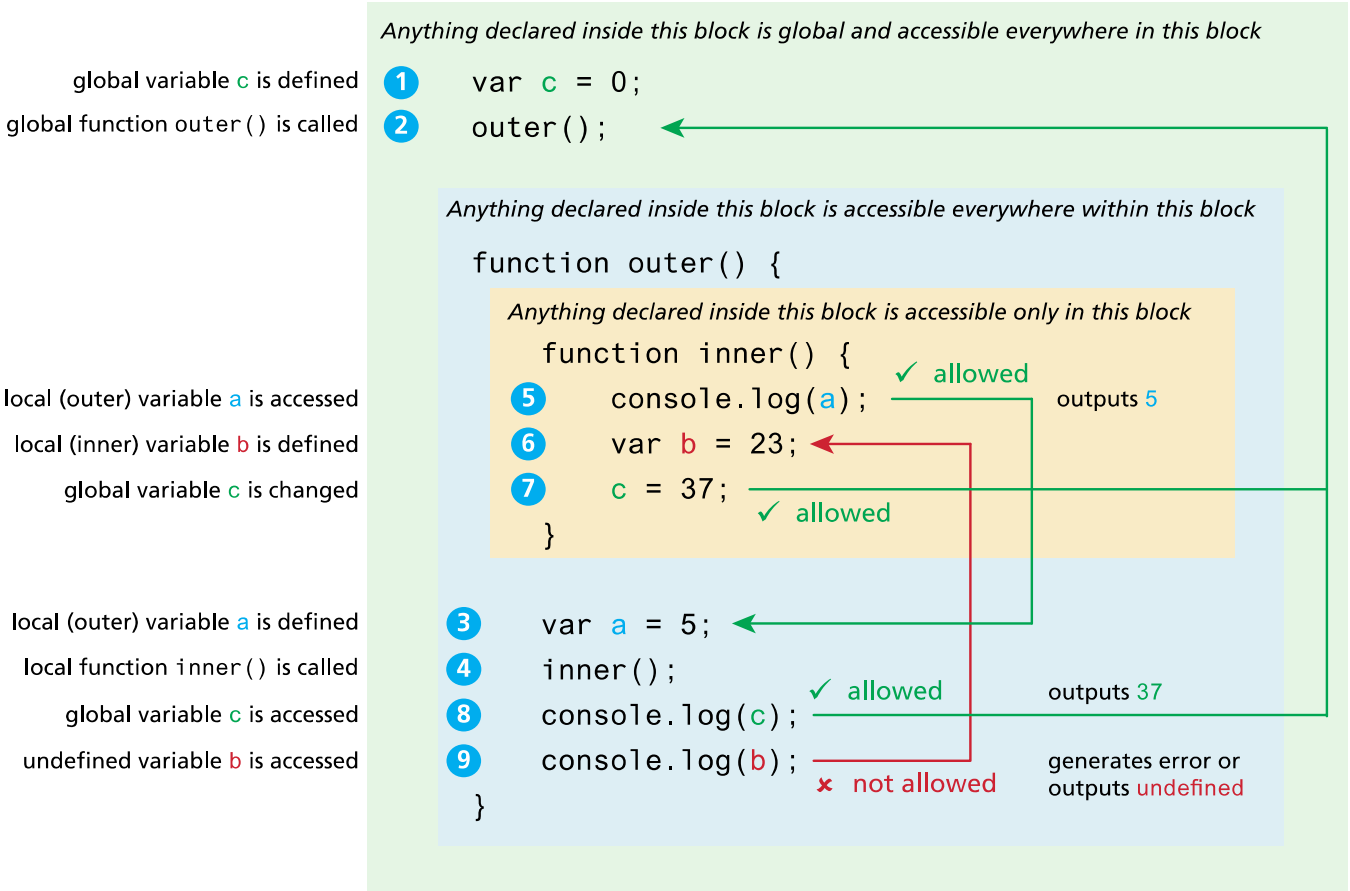
```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Nested Function

- ❖ All functions have access to the global scope.
- ❖ In fact, in JavaScript, all functions have access to the scope "above" them.
- ❖ JavaScript supports nested functions. Nested functions have access to the scope "above" them.
  - In this example, the inner function **plus**() has access to the **counter** variable in the parent function:

```
function add() {
  let counter = 0;
  function plus() {counter += 1;}
  plus();
  return counter;
}
```

Nested Function



요 약

➢ JavaScript Introduction

- How to import JavaScript into HTML document

➢ JavaScript Syntax

- Values, Variables, Keywords, Identifier

➢ JavaScript Variables

- var / let/ const

➢ JavaScript Functions

- How to declare function
- How to invoke function