



— ISTANBUL —  
OKAN UNIVERSITY

# CENG374

## Internet Programming

### LESSON 9



The background is white with several colorful abstract shapes and dots. In the top-left corner, there are overlapping yellow and blue circles. In the top-right corner, there are yellow, blue, and orange circles. In the bottom-left corner, there is a large orange cloud-like shape. In the bottom-right corner, there are blue and yellow cloud-like shapes. Scattered throughout the background are small dots in yellow, blue, and orange.

# **Client-Side Technologies: JavaScript and XHTML Documents**

# JavaScript Output

## JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using `innerHTML`.
- Writing into the HTML output using `document.write()`.
- Writing into an alert box, using `window.alert()`.
- Writing into the browser console, using `console.log()`.

# JavaScript Statements

Keyword	Description
var	Declares a variable
let	Declares a block variable
const	Declares a block constant
if	Marks a block of statements to be executed on a condition
switch	Marks a block of statements to be executed in different cases
for	Marks a block of statements to be executed in a loop
function	Declares a function
return	Exits a function
try	Implements error handling to a block of statements

# JavaScript Variables

## Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using `var`
- Using `let`
- Using `const`

# JavaScript Variables

```
x = 5;  
y = 6;  
z = x + y;
```

```
var x = 5;  
var y = 6;  
var z = x + y;
```

# JavaScript Variables

```
x = 5;
```

```
y = 6;
```

```
z = x + y;
```

```
let x = 5;
```

```
let y = 6;
```

```
let z = x + y;
```

# JavaScript Variables

```
x = 5;  
y = 6;  
z = x + y;
```

```
const x = 5;  
const y = 6;  
const z = x + y;
```



# JavaScript Variables

```
const price1 = 5;  
const price2 = 6;  
let total = price1 + price2;
```

# When to Use var, let, or const?

1. Always declare variables
2. Always use `const` if the value should not be changed
3. Always use `const` if the type should not be changed (Arrays and Objects)
4. Only use `let` if you can't use `const`
5. Only use `var` if you MUST support old browsers.

# JavaScript Let

The `let` keyword was introduced in ES6 (2015).

Variables declared with `let` have **Block Scope**

Variables declared with `let` must be **Declared** before use

Variables declared with `let` cannot be **Redeclared** in the same scope

# JavaScript Let

```
{  
  let x = 2;  
}  
// x can NOT be used here
```

# Global Scope

```
{  
  var x = 2;  
}  
// x CAN be used here
```

# JavaScript Const

The `const` keyword was introduced in ES6 (2015).

Variables defined with `const` cannot be **Redeclared**

Variables defined with `const` cannot be **Reassigned**

Variables defined with `const` have **Block Scope**

# JavaScript Const

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

# When to use JavaScript const?

**Always declare a variable with `const` when you know that the value should not be changed.**

Use `const` when you declare:

- A new Array
- A new Object
- A new Function
- A new RegExp



# JavaScript Operators

Javascript operators are used to perform different types of mathematical and logical computations.

Examples:

The **Assignment Operator** `=` assigns values

The **Addition Operator** `+` adds values

The **Multiplication Operator** `*` multiplies values

The **Comparison Operator** `>` compares values

# JavaScript Operators

## Types of JavaScript Operators

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators

# JavaScript Arithmetic

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

# JavaScript Assignment

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

# JavaScript Assignment

Operator	Example	Same As
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y

# JavaScript Assignment

Operator	Example	Same As
<code>&amp;=</code>	<code>x &amp;= y</code>	<code>x = x &amp; y</code>
<code>^=</code>	<code>x ^= y</code>	<code>x = x ^ y</code>
<code> =</code>	<code>x  = y</code>	<code>x = x   y</code>

# JavaScript Assignment

Operator	Example	Same As
&&=	x &&= y	x = x && (x = y)
=	x   = y	x = x    (x = y)
??=	x ??= y	x = x ?? (x = y)

# JavaScript Data Types

## JavaScript has 8 Datatypes

1. String
2. Number
3. BigInt
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

## The Object Datatype

The object data type can contain:

1. An object
2. An array
3. A date



# JavaScript Data Types 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 Data Types Examples

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

<h2>JavaScript</h2>

<p>When adding a number and a string, JavaScript will treat the number as a string.</p>

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

<script>
let x = 16 + "Volvo";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

# JavaScript Data Types Examples

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

<h2>JavaScript</h2>

<p>When adding a number and a string, JavaScript will treat the number as a string.</p>

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

<script>
let x = 16 + "Volvo";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

## Output

### JavaScript

When adding a number and a string, JavaScript will treat the number as a string.

16Volvo

# JavaScript Data Types Examples

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

<h2>JavaScript</h2>

<p>When adding a string and a number, JavaScript will treat the number as a string.</p>

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

<script>
let x = "Volvo" + 16;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

# JavaScript Data Types Examples

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

<h2>JavaScript</h2>

<p>When adding a string and a number, JavaScript will treat the number as a string.</p>

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

<script>
let x = "Volvo" + 16;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

## Output

### JavaScript

When adding a string and a number, JavaScript will treat the number as a string.

Volvo16

# JavaScript Data Types Examples

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

<h2>JavaScript</h2>

<p>JavaScript evaluates expressions from left to right. Different sequences can produce different results:</p>

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

<script>
let x = 16 + 4 + "Volvo";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

# JavaScript Data Types Examples

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

<h2>JavaScript</h2>

<p>JavaScript evaluates expressions from left to right. Different sequences can produce different results:</p>

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

<script>
let x = 16 + 4 + "Volvo";
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Output

## JavaScript

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

20Volvo

# JavaScript Data Types Examples

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

<h2>JavaScript</h2>

<p>JavaScript evaluates expressions from left to right. Different sequences can produce different results:</p>

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

<script>
let x = "Volvo" + 16 + 4;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```



# JavaScript Data Types Examples

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

<h2>JavaScript</h2>

<p>JavaScript evaluates expressions from left to right. Different sequences can produce different results:</p>

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

<script>
let x = "Volvo" + 16 + 4;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

## Output

### JavaScript

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

Volvo164

# JavaScript Types are Dynamic

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

<h2>JavaScript Data Types</h2>

<p>JavaScript has dynamic types. This means that the same variable can be used to hold different data types:</p>

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

<script>
let x;           // Now x is undefined
x = 5;           // Now x is a Number
x = "John";      // Now x is a String

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

</body>
</html>
```

# JavaScript Strings

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

<h2>JavaScript Strings</h2>

<p>Strings are written with quotes. You can use single or double quotes:</p>

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

<script>
let carName1 = "Volvo XC60";
let carName2 = 'Volvo XC60';

document.getElementById("demo").innerHTML =
carName1 + "<br>" +
carName2;
</script>

</body>
</html>
```

# JavaScript Strings

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

<h2>JavaScript Strings</h2>

<p>Strings are written with quotes. You can use single or double quotes:</p>

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

<script>
let carName1 = "Volvo XC60";
let carName2 = 'Volvo XC60';

document.getElementById("demo").innerHTML =
carName1 + "<br>" +
carName2;
</script>

</body>
</html>
```

## Output

### JavaScript Strings

Strings are written with quotes. You can use single or double quotes:

Volvo XC60  
Volvo XC60

# JavaScript Strings

```
// Single quote inside double quotes:  
let answer1 = "It's alright";  
  
// Single quotes inside double quotes:  
let answer2 = "He is called 'Johnny'";  
  
// Double quotes inside single quotes:  
let answer3 = 'He is called "Johnny"';
```

# JavaScript Numbers

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

<h2>JavaScript Numbers</h2>

<p>Numbers can be written with, or without decimals:</p>

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

<script>
let x1 = 34.00;
let x2 = 34;
let x3 = 3.14;

document.getElementById("demo").innerHTML =
x1 + "<br>" + x2 + "<br>" + x3;
</script>

</body>
</html>
```

## Output

### JavaScript Numbers

Numbers can be written with, or without decimals:

34  
34  
3.14

# Exponential Notation

```
let y = 123e5;    // 12300000  
let z = 123e-5;   // 0.00123
```

# Note

Most programming languages have many number types:

Whole numbers (integers):

byte (8-bit), short (16-bit), int (32-bit), long (64-bit)

Real numbers (floating-point):

float (32-bit), double (64-bit).

**Javascript numbers are always one type:  
double (64-bit floating point).**



# JavaScript BigInt

```
let x = BigInt("123456789012345678901234567890");
```

# JavaScript Booleans

```
let x = 5;  
let y = 5;  
let z = 6;  
(x == y)           // Returns true  
(x == z)           // Returns false
```

# JavaScript Arrays

```
const cars = ["Saab", "Volvo", "BMW"];
```

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

# JavaScript Objects

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

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

# The typeof Operator

```
typeof ""           // Returns "string"  
typeof "John"       // Returns "string"  
typeof "John Doe"   // Returns "string"
```

```
typeof 0            // Returns "number"  
typeof 314          // Returns "number"  
typeof 3.14         // Returns "number"  
typeof (3)          // Returns "number"  
typeof (3 + 4)      // Returns "number"
```

# Undefined

```
let car;    // Value is undefined, type is undefined
```

```
car = undefined;    // Value is undefined, type is undefined
```

# Empty Values

# JavaScript Functions

```
<!DOCTYPE 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

```
<!DOCTYPE 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>
```

## Output

### JavaScript Functions

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

12

# JavaScript Function Syntax

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

# Function Invocation

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)

# Function Return

```
<!DOCTYPE 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>
let x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;

function myFunction(a, b) {
  return a * b;
}
</script>

</body>
</html>
```

# Function Return

```
<!DOCTYPE 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>
let x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;

function myFunction(a, b) {
  return a * b;
}
</script>

</body>
</html>
```

Output

## JavaScript Functions

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

# Why 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.

# The () Operator

```
<!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>
```

# The () Operator

## Output

### JavaScript Functions

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

<h1>JavaScript Functions</h1>

<p>Invoke (call) a function that converts from Fahrenheit to Celsius:
<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>
```

Invoke (call) a function that converts from Fahrenheit to Celsius:  
25



# The () Operator

Accessing a function with incorrect parameters can return an incorrect answer:

```
<!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>
```

# The () Operator

Accessing a function with incorrect parameters can return an incorrect answer:

```
<!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>
```

Output

## JavaScript Functions

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

NaN

# The () Operator

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

```
<!DOCTYPE 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>
```

# The () Operator

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

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

<h1>JavaScript Functions</h1>

<p>Accessing a function without () returns
<p id="demo"></p>

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

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

</body>
</html>
```

## Output

### JavaScript Functions

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

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

# Functions Used as Variable Values

```
<!DOCTYPE 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>
```

# Local Variables

```
<!DOCTYPE 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>
```

# Local Variables

```
<!DOCTYPE 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>
```

## Output

### JavaScript Functions

Outside myFunction() carName is undefined.

Outside: undefined

Inside: string Volvo

# Questions



# Question 1

Execute the function named `myFunction`.

```
function myFunction() {  
    alert("Hello World!");  
}  
 ;
```

# Answer 1

```
function myFunction() {  
    alert("Hello World!");  
}  
myFunction();
```

## Question 2

Use comments to describe the correct data type of the following variables:

```
let length = 16;           //   
let lastName = "Johnson"; //   
const x = {  
  firstName: "John",  
  lastName: "Doe"  
};                          // 
```

# Answer 2

```
let length = 16;           // Number
let lastName = "Johnson"; // String
const x = {
  firstName: "John",
  lastName: "Doe"
};                          // Object
```

## Question 3

Use the correct assignment operator that will result in `x` being 15 (same as `x = x + y`).

```
x = 10;
```

```
y = 5;
```

```
x  y;
```

## Answer 3

```
x = 10;
```

```
y = 5;
```

```
x += y;
```

## Question 4

What will be the output of the following code snippet?

```
function greet(name) {  
  console.log("Hello, " + name);  
}  
  
var person = {  
  name: "John",  
  greet: function() {  
    greet(this.name);  
  }  
};  
  
var greetFn = person.greet;  
greetFn();
```

## Question 5

What will be the output of the following code snippet?

```
var a = 10;

function outer() {
  var b = 20;

  function inner() {
    var c = 30;
    console.log(a + b + c);
  }

  return inner;
}

var innerFunction = outer();
innerFunction();
```



The background is white with decorative elements in the corners. Top-left: A cluster of overlapping blue and yellow circles. Top-right: A cluster of overlapping yellow, blue, and orange circles. Bottom-left: A large, irregular orange shape. Bottom-right: A cluster of overlapping blue and yellow circles. Scattered throughout are small dots in yellow, blue, and orange.

Thanks for listening...