



# **INTEGRATIVE PROGRAMMING AND TECHNOLOGIES 2**

Bea May M. Belarmino



Bachelor of Science in Information Technology  
College of Computing Studies

# **LAGUNA UNIVERSITY**

## **Vision**

Laguna University shall be a socially responsive educational institution of choice providing holistically developed individuals in the Asia-Pacific Region.

## **Mission**

Laguna University is committed to produce academically prepared and technically skilled individuals who are socially and morally upright.

# Table of Contents

## **Module 1: Javascript Introduction**

Introduction	1
Learning Outcomes	1
Lesson 1. JavaScript Tutorial	2
Lesson 2. JavaScript Can Change HTML Content	2
Lesson 3. JavaScript Can Change HTML Attribute Values	3
Lesson 4. JavaScript Can Change HTML Styles (CSS)	4
Lesson 5. JavaScript Can Hide HTML Elements	4
Lesson 6. JavaScript Can Show HTML Elements	5
Assessment Task	6
Summary	7
References	7

## **Module 2: Javascript Basics - Part 1: Foundation And Syntax**

Introduction	8
Learning Outcomes	8
Lesson 1. JavaScript Where To	8
Lesson 2. JavaScript Output	14
Lesson 3. JavaScript Statements	19
Lesson 4. JavaScript Syntax	24
Lesson 5. JavaScript Comments	30
Lesson 6. JavaScript Variables	32
Lesson 7. JavaScript Let	43
Lesson 8. JavaScript Const	47
Assessment Task	53
Summary	54
References	55

### **Module 3: Javascript Basics - Part 2: Operators, Functions, Objects, And Events**

Introduction	55
Learning Outcomes	56
Lesson 1. JavaScript Operators	56
Lesson 2. JavaScript Data Types	60
Lesson 3. JavaScript Function	66
Lesson 4. JavaScript Objects	71
Lesson 5. JavaScript Events	78
Assessment Task	81
Summary	82
References	83

## **Course Code: PC-2210 – Integrative Programming and Technologies 2**

**Course Description:** This course is about programming with the purpose of combining and coordinating separate elements to construct an interrelated whole; Designing individual modules to function cooperatively as an entire system; Incorporating modules coded in different languages to achieve unified tasks. Students will be introduced to design and build systems and integrate them into an organization. This knowledge area develops the skills to gather requirements, then source, evaluate and integrate components into a single system, and finally validate the system.

### **Course Intended Learning Outcomes (CILO):**

At the end of this course, the students should be able to:

1. Id

### **Course Requirements:**

▪ <b>Class Standing</b>	<b>- 60%</b>
▪ <b>Attendance/ Recitation/ Assignment</b>	<b>- 20%</b>
▪ <b>Quiz</b>	<b>- 20%</b>
▪ <b>Activities</b>	<b>- 20%</b>
▪ <b>Major Exams</b>	<b>- 40%</b>

Periodic Grade	<hr/> 100%
----------------	------------

**Final Grade** = Total CS + Final Exam x 70% + 30% of the Midterm

# MODULE 1

## JAVASCRIPT INTRODUCTION



### Introduction

Welcome to the dynamic world of JavaScript, a versatile scripting language that empowers web developers to create interactive and engaging websites. JavaScript, often abbreviated as JS, is a client-side programming language that plays a crucial role in enhancing the functionality and interactivity of web pages.

One of the key strengths of JavaScript lies in its ability to dynamically manipulate HTML content. With JavaScript, you can seamlessly alter the text, structure, and elements of an HTML page, providing a dynamic and responsive user experience. Whether you want to update a paragraph of text, modify the attributes of an HTML element, or even change the overall styling using CSS, JavaScript is the go-to language for achieving such feats.

In this introductory journey, we will explore some fundamental capabilities of JavaScript. From changing HTML content to manipulating attribute values, styling with CSS, hiding elements, and revealing hidden gems on your webpage – JavaScript opens a world of possibilities for creating modern, interactive, and user-friendly web applications. Get ready to dive into the exciting realm of JavaScript, where you'll learn how to breathe life into static web pages and transform them into dynamic, responsive, and visually appealing experiences for users.



### Learning Outcomes

At the end of this module, students should be able to:

1. Learn the fundamental syntax and structure of JavaScript.
2. Learn how JavaScript can dynamically change text within HTML elements.
3. Implement dynamic content changes in a sample web page using JavaScript.

# Lesson 1. JavaScript Tutorial

JavaScript is the world's most popular programming language. JavaScript is the programming language of the Web. JavaScript is easy to learn. This module will teach you JavaScript from basic to advanced.

## Why Study JavaScript?

JavaScript is one of the 3 languages all web developers must learn:

- 1. HTML to define the content of web pages
- 2. CSS to specify the layout of web pages
- 3. JavaScript to program the behavior of web pages

# Lesson 2. JavaScript Can Change HTML Content

One of many JavaScript HTML methods is getElementById(). The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

Code:

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can change HTML content.</p>
<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello
JavaScript!'">Click Me!</button>
</body>
</html>
```

Output:



### Lesson 3. JavaScript Can Change HTML Attribute Values

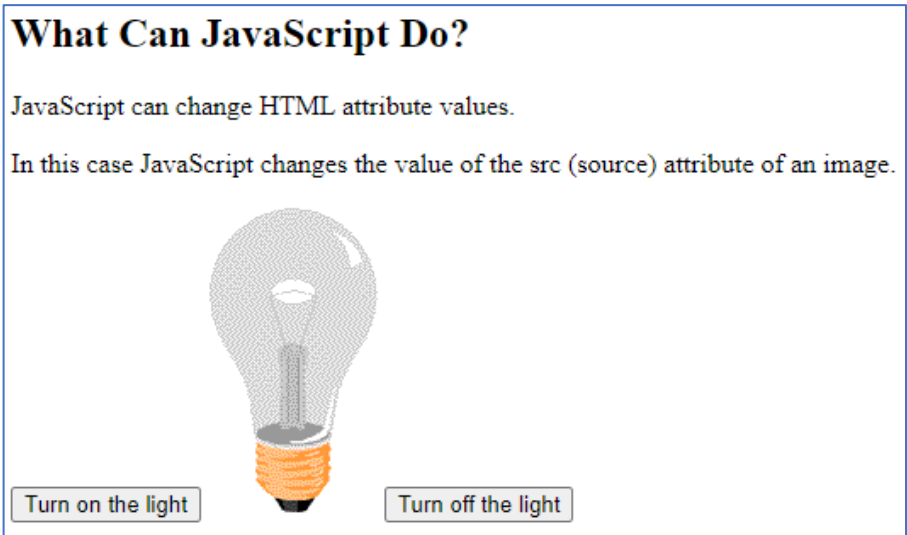
In this example JavaScript changes the value of the src (source) attribute of an <img> tag:**Philosoph**

Code:

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p>JavaScript can change HTML attribute values.</p>
<p>In this case JavaScript changes the value of the src (source) attribute of an image.</p>
<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the
light</button>

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the
light</button>
</body>
</html>
```

Output:





## What Can JavaScript Do?

JavaScript can change HTML attribute values.

In this case JavaScript changes the value of the src (source) attribute of an image.



Turn on the light

Turn off the light

## Lesson 4. JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Code:

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can change the style of an HTML element.</p>
<button                                     type="button"
onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>
</body>
</html>
```

Output:

## What Can JavaScript Do?

JavaScript can change the style of an HTML element.

Click Me!

## What Can JavaScript Do?

JavaScript can change the style of an HTML element.

Click Me!

## Lesson 5. JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the display style:

Code:

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can hide HTML elements.</p>
<button type="button"
onclick="document.getElementById('demo').style.display='none'">Click Me!</button>
</body>
</html>
```

Output:



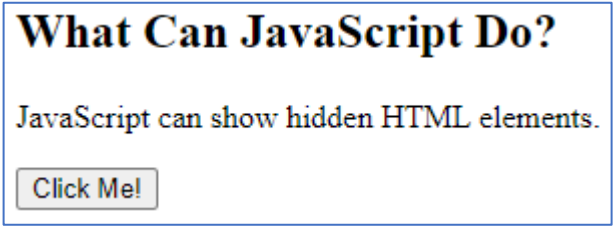
## Lesson 6. JavaScript Can Show HTML Elements

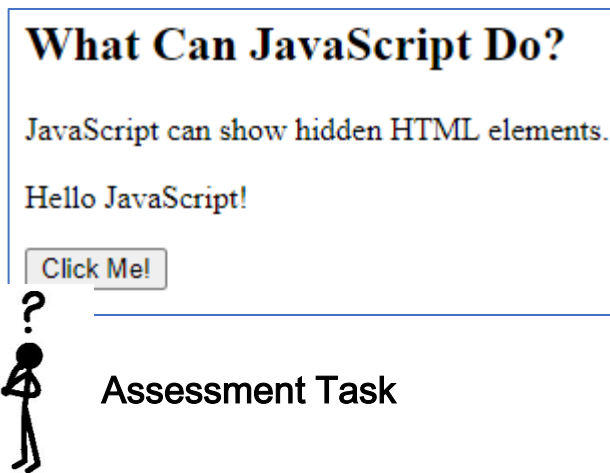
Showing hidden HTML elements can also be done by changing the display style:

Code:

```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<p>JavaScript can show hidden HTML elements.</p>
<p id="demo" style="display:none">Hello JavaScript!</p>
<button type="button"
onclick="document.getElementById('demo').style.display='block'">Click Me!</button>
</body>
</html>
```

Output:





## Assessment Task

### Lab Setup:

- Open a text editor (e.g., Visual Studio Code, Sublime Text).
- Create a new HTML file and save it with an appropriate name (e.g., lastname\_module1\_lab.js).

### Lab Tasks:

#### Task 1: Basic JavaScript Integration

1. Set up the HTML structure with a heading, a paragraph, and a button.
2. Integrate a `<script>` tag within the HTML to link to an external JavaScript file (create a new file, e.g., script.js).
3. Write a simple JavaScript function that displays an alert when the button is clicked.
4. Verify that the basic JavaScript integration works by testing the button click.

#### Task 2: Dynamic Text Manipulation

1. Create a new section in the HTML to demonstrate dynamic text manipulation.
2. Add an element with a unique ID (e.g., `<p id="dynamicText">This is dynamic text.</p>`).
3. In the script.js file, write a JavaScript function that changes the text content of the element to a predefined message when called.
4. Test the dynamic text manipulation by triggering the function, possibly on a button click.

#### Task 3: HTML Attribute Modification

1. Introduce an image in the HTML with an ID (e.g., ``).
2. In the script.js file, write a function that changes the src attribute of the image to a different image file.
3. Test the HTML attribute modification by triggering the function, possibly on another button click.

#### Task 4: CSS Style Modification

1. Add an HTML element (e.g., `<div id="styledDiv">This is a styled div.</div>`).
2. In the script.js file, write a function that changes the style of the element (e.g., font size, color).

3. Test the CSS style modification by triggering the function, possibly on another button click.

#### **Task 5: Element Visibility Control**

1. Create two buttons and a paragraph with an ID (e.g., `<p id="toggleParagraph">Toggle me!</p>`).
2. In the script.js file, write functions to hide and show the paragraph by changing its display style property.
3. Test the element visibility control by triggering the functions using the respective buttons.



#### **Summary**

- JavaScript is a versatile scripting language crucial for web development, functioning as a client-side programming language to enhance the interactivity of web pages. It dynamically manipulates HTML content, allowing developers to alter text, structure, and elements seamlessly. This introductory journey explores fundamental JavaScript capabilities, from changing HTML content to manipulating attributes, styling with CSS, hiding elements, and creating dynamic and user-friendly web applications.
- The module provides an overview of JavaScript, emphasizing its popularity as the programming language of the web. Students are encouraged to study JavaScript alongside HTML and CSS, as it plays a crucial role in programming the behavior of web pages.
- Several lessons are covered, demonstrating practical applications of JavaScript. These lessons include changing HTML content using methods like `getElementById()`, altering HTML attribute values, changing HTML styles (CSS), hiding and showing HTML elements. Each lesson is accompanied by code examples and outputs, allowing students to grasp the practical implementation of JavaScript concepts.



#### **References**

##### **Book**

- Osmani, A. (2023). Learning JavaScript Design Patterns.(A JavaScript and React Developer's Guide) Second Edition.United States of America, O'Reilly Media, Inc. 105 Gravenstein Highway North, Sebastopol, CA 95472.

##### **Website**

- W3Schools.com (n.d.). Retrieved from [https://www.w3schools.com/js/js\\_output.asp](https://www.w3schools.com/js/js_output.asp)

## **MODULE 2**

# **JAVASCRIPT BASICS - PART 1: FOUNDATION AND SYNTAX**



### **Introduction**

The JavaScript basic serves as an essential primer on JavaScript basics, laying the groundwork for effective scripting. Delving into the fundamentals, this section draws insights from w3schools.com to guide learners through the strategic placement of JavaScript in HTML documents. It explores diverse methods of outputting information, introduces the nature of JavaScript statements, and unfolds the concept of variables. Syntax rules, including naming conventions and whitespace usage, are thoroughly covered. Learners are introduced to comments as valuable annotations, and specific keywords like `let` and `const` are explored, setting the stage for a comprehensive understanding of foundational JavaScript concepts.



### **Learning Outcomes**

At the end of this module, students should be able to:

1. Develop a strong foundation in JavaScript basics, encompassing variables, statements, operators, and functions.
2. Attain proficiency in JavaScript operators, functions, and object-oriented concepts.
3. Develop and showcase a small JavaScript project that integrates learned fundamentals and practices values integration.

## **Lesson 1. JavaScript Where To**

### **1. The `<script>` Tag**

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

**Code:**

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript in Body</h2>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
</body>
</html>
```

**Output:**



Old JavaScript examples may use a type attribute: `<script type="text/javascript">`.  
The type attribute is not required. JavaScript is the default scripting language in HTML.

**2. JavaScript Functions and Events**

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.

**3. JavaScript in <head> or <body>**

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.

**a. JavaScript in <head>**

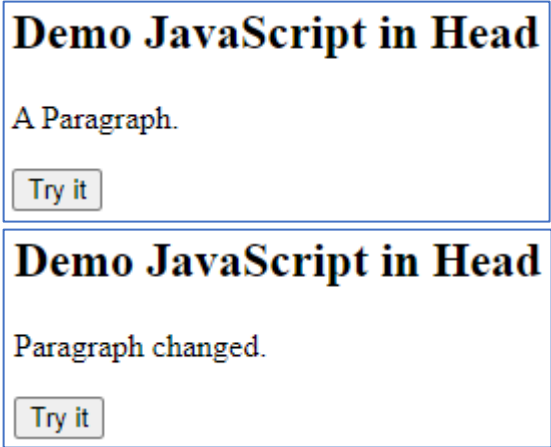
In this example, a JavaScript function is placed in the `<head>` section of an HTML page.  
The function is invoked (called) when a button is clicked:

**Code:**

```
<!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>
```

Output:



**b. JavaScript in <body>**

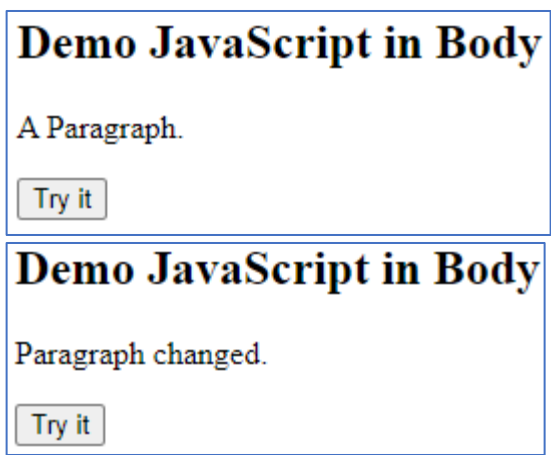
In this example, a JavaScript function is placed in the <body> section of an HTML page.

The function is invoked (called) when a button is clicked:

**Code:**

```
<!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>
```

Output:



Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

4. External JavaScript

Scripts can also be placed in external files:

External file: *myScript.js*

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

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 (source) attribute of a <script> tag:

```
<!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>
```

Output:



### Demo External JavaScript

A Paragraph.

Try it

This example links to "myScript.js".

(myFunction is stored in "myScript.js")

### Demo External JavaScript

Paragraph changed.

Try it

This example links to "myScript.js".

(myFunction is stored in "myScript.js")

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

**Note:** External scripts cannot contain <script> tags.

**a. External JavaScript Advantages**

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

To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

**b. External References**

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
- With a file path (like /js/)
- Without any path

This example uses a full URL to link to myScript.js:

```
<!DOCTYPE html>
<html>
<body>
<h2>External JavaScript</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Click Me</button>
<p>This example uses a full web URL to link to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>
```

```
<script src="https://www.w3schools.com/js/myScript.js"></script>
</body>
</html>
```

Output:

## External JavaScript

A Paragraph.

Click Me

This example uses a full web URL to link to "myScript.js".

(myFunction is stored in "myScript.js")

## External JavaScript

Paragraph changed.

Click Me

This example uses a full web URL to link to "myScript.js".

(myFunction is stored in "myScript.js")

This example uses a file path to link to myScript.js:

```
<!DOCTYPE html>
<html>
<body>
<h2>External JavaScript</h2>
<p id="demo">A Paragraph.</p>
<button type="button" onclick="myFunction()">Try it</button>
<p>This example uses a file path to link to "myScript.js".</p>
<p>(myFunction is stored in "myScript.js")</p>
<script src="/js/myScript.js"></script>
</body>
</html>
```

Output:

## External JavaScript

A Paragraph.

Try it

This example uses a file path to link to "myScript.js".

(myFunction is stored in "myScript.js")

## External JavaScript

Paragraph changed.

Try it

This example uses a file path to link to "myScript.js".

(myFunction is stored in "myScript.js")

This example uses no path to link to myScript.js:

```
<!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>
```

Output:

## Demo External JavaScript

A Paragraph.

Try it

This example links to "myScript.js".

(myFunction is stored in "myScript.js")

## Demo External JavaScript

Paragraph changed.

Try it

This example links to "myScript.js".

(myFunction is stored in "myScript.js")

## Lesson 2. 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().

1. Using innerHTML

To access an HTML element, JavaScript can use the document.getElementById(id) method.

The id attribute defines the HTML element. The innerHTML property defines the HTML content:

```
<!DOCTYPE html>
<html>
<body>
<h2>My First Web Page</h2>
<p>My First Paragraph.</p>
  <p id="demo"></p>
  <script>
    document.getElementById("demo").innerHTML = 5 + 6;
  </script>
</body>
</html>
```

Output:



Changing the innerHTML property of an HTML element is a common way to display data in HTML.

2. Using document.write()

For testing purposes, it is convenient to use document.write():

```
<!DOCTYPE html>
<html>
<body>
  <h2>My First Web Page</h2>
  <p>My first paragraph.</p>
  <p>Never call document.write after the document has finished loading.
  It will overwrite the whole document.</p>
  <script>
```

```
        document.write(5 + 6);
    </script>
</body>
</html>
```

Output:

## My First Web Page

My first paragraph.

Never call `document.write` after the document has finished loading. It will overwrite the whole document.

11

Using `document.write()` after an HTML document is loaded, will delete all existing HTML:

```
<!DOCTYPE html>
<html>
<body>
    <h2>My First Web Page</h2>
    <p>My first paragraph.</p>
    <button type="button" onclick="document.write(5 + 6)">Try it</button>
</body>
</html>
```

Output:

## My First Web Page

My first paragraph.

Try it

The `document.write()` method should only be used for testing.

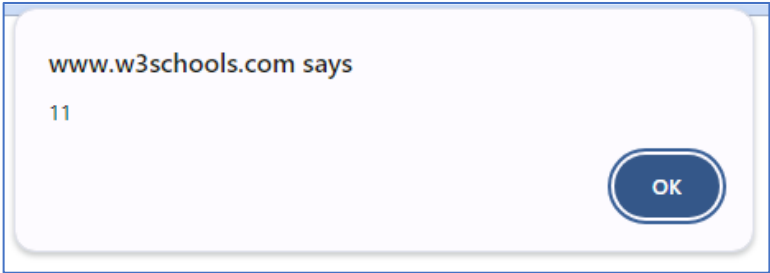
3. Using `window.alert()`

You can use an alert box to display data:

```
<!DOCTYPE html>
<html>
<body>
    <h2>My First Web Page</h2>
    <p>My first paragraph.</p>
    <script>
        window.alert(5 + 6);
    </script>
```

```
</body>
</html>
```

Output:



You can skip the window keyword.  
In JavaScript, the window object is the global scope object. This means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional:

```
<!DOCTYPE html>
<html>
<body>
  <h2>My First Web Page</h2>
  <p>My first paragraph.</p>
  <script>
    alert(5 + 6);
  </script>
</body>
</html>
```

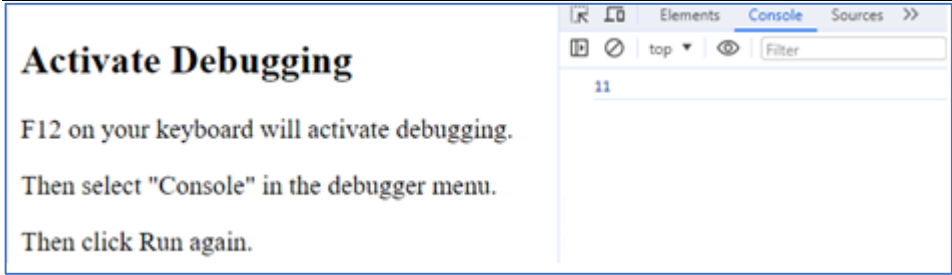
Output:



4. Using console.log()
- For debugging purposes, you can call the console.log() method in the browser to display data.

```
<!DOCTYPE html>
```

```
<html>
<body>
  <h2>Activate Debugging</h2>
  <p>F12 on your keyboard will activate debugging.</p>
  <p>Then select "Console" in the debugger menu.</p>
  <p>Then click Run again.</p>
  <script>
    console.log(5 + 6);
  </script>
</body>
</html>
```

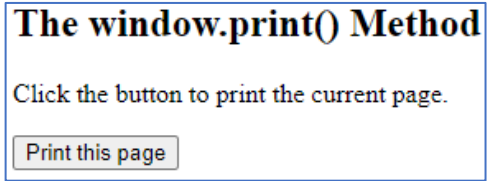


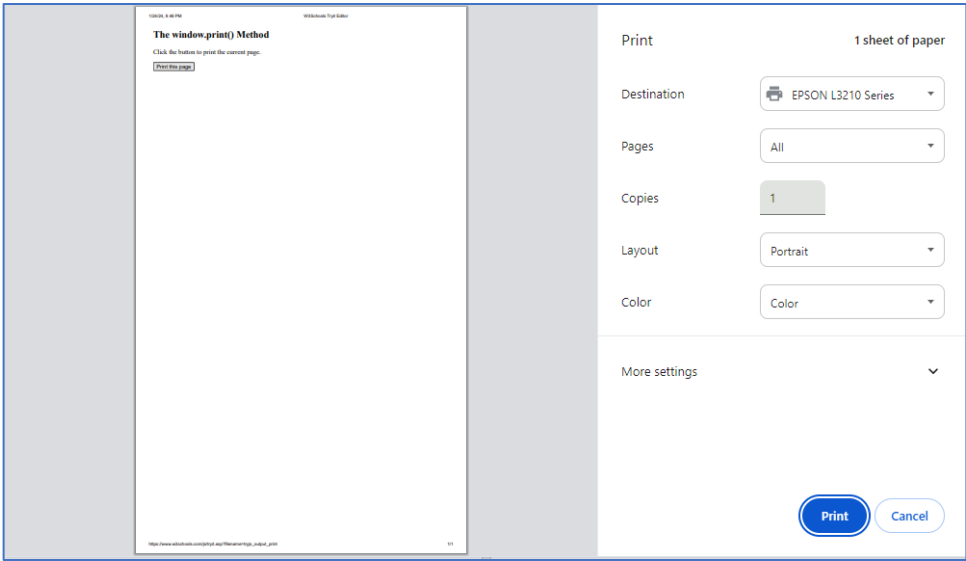
5. JavaScript Print

JavaScript does not have any print object or print methods.  
You cannot access output devices from JavaScript.  
The only exception is that you can call the window.print() method in the browser to print the content of the current window.

```
<!DOCTYPE html>
<html>
<body>
  <h2>The window.print() Method</h2>
  <p>Click the button to print the current page.</p>
  <button onclick="window.print()">Print this page</button>
</body>
</html>
```

Output:





### Lesson 3. JavaScript Statements

Sample Code:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Statements</h2>
<p>A <b>JavaScript program</b> is a list of <b>statements</b> to be executed by a
computer.</p>
<p id="demo"></p>
<script>
let x, y, z; // Statement 1
x = 5;      // Statement 2
y = 6;      // Statement 3
z = x + y;  // Statement 4
document.getElementById("demo").innerHTML =
"The value of z is " + z + ".";
</script>
</body>
</html>
```

Output:

### JavaScript Statements

A **JavaScript program** is a list of **statements** to be executed by a computer.

The value of z is 11.

#### 1. JavaScript Programs



A computer program is a list of "instructions" to be "executed" by a computer.  
In a programming language, these programming instructions are called statements.  
A JavaScript program is a list of programming statements.  
**Note:** In HTML, JavaScript programs are executed by the web browser.

2. **JavaScript Statements**

JavaScript statements are composed of:  
Values, Operators, Expressions, Keywords, and Comments.  
This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Statements</h2>
    <p>In HTML, JavaScript statements are executed by the browser.</p>
    <p id="demo"></p>
    <script>
        document.getElementById("demo").innerHTML = "Hello Dolly.";
    </script>
</body>
</html>
```

Output:

**JavaScript Statements**  
  
In HTML, JavaScript statements are executed by the browser.  
  
Hello Dolly.

Most JavaScript programs contain many JavaScript statements.  
The statements are executed, one by one, in the same order as they are written.  
**Note:** JavaScript programs (and JavaScript statements) are often called JavaScript code.

3. **Semicolons ;**

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Statements</h2>
```

```
<p>JavaScript statements are separated by semicolons.</p>
<p id="demo1"></p>
<script>
let a, b, c;
a = 5;
b = 6;
c = a + b;
document.getElementById("demo1").innerHTML = c;
</script>
</body>
</html>
```

Output:

JavaScript Statements

JavaScript statements are separated by semicolons.

11

4. When separated by semicolons, multiple statements on one line are allowed:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript Statements</h2>
  <p>Multiple statements on one line are allowed.</p>
  <p id="demo1"></p>
  <script>
let a, b, c;
a = 5; b = 6; c = a + b;
document.getElementById("demo1").innerHTML = c;
</script>
</body>
</html>
```

Output:

JavaScript Statements

Multiple statements on one line are allowed.

11

**Note:** On the web, you might see examples without semicolons. Ending statements with semicolon is not required, but highly recommended.

5. JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
let person = "Hege";  
let person="Hege";
```

A good practice is to put spaces around operators ( = + - \* / ):

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

6. JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

```
<!DOCTYPE html>  
<html>  
<body>  
    <h2>JavaScript Statements</h2>  
    <p>  
        The best place to break a code line is after an operator or a comma.  
    </p>  
    <p id="demo"></p>  
    <script>  
        document.getElementById("demo").innerHTML =  
        "Hello Dolly!";  
    </script>  
</body>  
</html>
```

Output:

```
JavaScript Statements  
  
The best place to break a code line is after an operator or a comma.  
  
Hello Dolly!
```

7. JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions:

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

```
<body>
  <h2>JavaScript Statements</h2>
  <p>JavaScript code blocks are written between { and }</p>
  <button type="button" onclick="myFunction()">Click Me!</button>
  <p id="demo1"></p>
  <p id="demo2"></p>
  <script>
    function myFunction() {
      document.getElementById("demo1").innerHTML = "Hello Dolly!";
      document.getElementById("demo2").innerHTML = "How are you?";
    }
  </script>
</body>
</html>
```

Output:



8. JavaScript Keywords

JavaScript statements often start with a keyword to identify the JavaScript action to be performed.

Our Reserved Words Reference lists all JavaScript keywords.

Here is a list of some of the keywords you will learn about in this tutorial:

Table 2.1 Reserved Words Reference lists

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

**Note:** JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.

## Lesson 4. JavaScript Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
// How to create variables:
var x;
let y;

// How to use variables:
x = 5;
y = 6;
let z = x + y;
```

### 1. JavaScript Values

The JavaScript syntax defines two types of values:

- Fixed values
- Variable values

Fixed values are called Literals.

Variable values are called Variables.

### 2. JavaScript Literals

The two most important syntax rules for fixed values are:

#### 9.1. Numbers are written with or without decimals:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Numbers</h2>
    <p>Number can be written with or without decimals.</p>
    <p id="demo"></p>
    <script>
        document.getElementById("demo").innerHTML = 10.50;
    </script>
</body>
</html>
```

JavaScript Numbers

Number can be written with or without decimals.

10.5

#### 9.2. Strings are text, written within double or single quotes:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Strings</h2>
```

```
<p>Strings can be written with double or single quotes.</p>
<p id="demo"></p>
<script>
  document.getElementById("demo").innerHTML = 'John Doe';
</script>
</body>
</html>
```

JavaScript Strings

Strings can be written with double or single quotes.

John Doe

3. JavaScript Variables

In a programming language, variables are used to store data values. JavaScript uses the keywords var, let and const to declare variables. An equal sign is used to assign values to 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").innerHTML = x;
  </script>
</body>
</html>
```

Output:

JavaScript Variables

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

6

4. JavaScript Operators

JavaScript uses arithmetic operators ( + - \* / ) to compute values:

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

```
<h2>JavaScript Operators</h2>
<p>JavaScript uses arithmetic operators to compute values (just like algebra).</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = (5 + 6) * 10;
</script>
</body>
</html>
```

Output:

```
JavaScript Operators

JavaScript uses arithmetic operators to compute values (just like algebra).

110
```

JavaScript uses an assignment operator ( = ) to assign values to variables:

```
<!DOCTYPE html>
<html>
<body>
  <h2>Assigning JavaScript Values</h2>
  <p>In JavaScript the = operator is used to assign values to variables.</p>
  <p id="demo"></p>
  <script>
let x, y;
x = 5;
y = 6;
document.getElementById("demo").innerHTML = x + y;
</script>
</body>
</html>
```

Output:

```
Assigning JavaScript Values

In JavaScript the = operator is used to assign values to variables.

11
```

5. 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:

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

```
<body>
  <h2>JavaScript Expressions</h2>
  <p>Expressions compute to values.</p>
  <p id="demo"></p>
  <script>
    document.getElementById("demo").innerHTML = 5 * 10;
  </script>
</body>
</html>
```

Output:

**JavaScript Expressions**

Expressions compute to values.

50

Expressions can also contain variable values:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript Expressions</h2>
  <p>Expressions compute to values.</p>
  <p id="demo"></p>
  <script>
    var x;
    x = 5;
    document.getElementById("demo").innerHTML = x * 10;
  </script>
</body>
</html>
```

Output:

**JavaScript Expressions**

Expressions compute to values.

50

The values can be of various types, such as numbers and strings.

For example, "John" + " " + "Doe", evaluates to "John Doe":

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript Expressions</h2>
  <p>Expressions compute to values.</p>
  <p id="demo"></p>
```



```
<script>
  document.getElementById("demo").innerHTML = "John" + " " + "Doe";
</script>
</body>
</html>
```

Output:

## JavaScript Expressions

Expressions compute to values.

John Doe

### 6. JavaScript Keywords

JavaScript keywords are used to identify actions to be performed.

The let keyword tells the browser to create variables:

```
<!DOCTYPE html>
<html>
<body>
  <h2>The <b>let</b> Keyword Creates Variables</h2>
  <p id="demo"></p>
  <script>
    let x, y;
    x = 5 + 6;
    y = x * 10;
    document.getElementById("demo").innerHTML = y;
  </script>
</body>
</html>
```

Output:

## The let Keyword Creates Variables

110

The var keyword also tells the browser to create variables:

```
<!DOCTYPE html>
<html>
<body>
  <h2>The var Keyword Creates Variables</h2>
  <p id="demo"></p>
  <script>
    var x, y;
    x = 5 + 6;
    y = x * 10;
    document.getElementById("demo").innerHTML = y;
```

```
</script>
</body>
</html>
```

## The var Keyword Creates Variables

110

**Note:** In these examples, using var or let will produce the same result.

You will learn more about var and let later in this tutorial.

### 7. JavaScript Identifiers / Names

Identifiers are JavaScript names.

Identifiers are used to name variables and keywords, and functions.

The rules for legal names are the same in most programming languages.

A JavaScript name must begin with:

- A letter (A-Z or a-z)
- A dollar sign (\$)
- Or an underscore (\_)

Subsequent characters may be letters, digits, underscores, or dollar signs.

**Note:** Numbers are not allowed as the first character in names.

This way JavaScript can easily distinguish identifiers from numbers.

### 8. JavaScript is Case Sensitive

All JavaScript identifiers are case sensitive.

The variables lastName and lastname, are two different variables:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript is Case Sensitive</h2>
  <p>Try to change lastName to lastname.</p>
  <p id="demo"></p>
  <script>
    let lastname, lastName;
    lastName = "Doe";
    lastname = "Peterson";
    document.getElementById("demo").innerHTML = lastName;
  </script>
</body>
</html>
```

Output:

## JavaScript is Case Sensitive

Try to change **lastName** to **lastname**.

Doe

JavaScript does not interpret LET or Let as the keyword let.

## 9. JavaScript and Camel Case

Historically, programmers have used different ways of joining multiple words into one variable name:

### 9.1. Hyphens:

first-name, last-name, master-card, inter-city.

**Note:** Hyphens are not allowed in JavaScript. They are reserved for subtractions.

### 9.2. Underscore:

first\_name, last\_name, master\_card, inter\_city.

### 9.3. Upper Camel Case (Pascal Case):

FirstName, LastName, MasterCard, InterCity.

### 9.4. Lower Camel Case:

JavaScript programmers tend to use camel case that starts with a lowercase letter:

firstName, lastName, masterCard, interCity.

## 10. JavaScript Character Set

JavaScript uses the Unicode character set.

Unicode covers (almost) all the characters, punctuations, and symbols in the world.

For a closer look, please study our Complete Unicode Reference.

## Lesson 5. JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

### 1. Single Line Comments

Single line comments start with //.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

```
<!DOCTYPE html>
<html>
<body>
    <h1 id="myH"></h1>
    <p id="myP"></p>
    <script>
        // Change heading:
        document.getElementById("myH").innerHTML = "JavaScript Comments";
        // Change paragraph:
```

```
        document.getElementById("myP").innerHTML = "My first paragraph.";
    </script>
</body>
</html>
```

Output:

# JavaScript Comments

My first paragraph.

This example uses a single line comment at the end of each line to explain the code:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Comments</h2>
    <p id="demo"></p>
    <script>
        let x = 5;    // Declare x, give it the value of 5
        let y = x + 2; // Declare y, give it the value of x + 2
        // Write y to demo:
        document.getElementById("demo").innerHTML = y;
    </script>
</body>
</html>
```

Output:

# JavaScript Comments

7

2. Multi-line Comments

Multi-line comments start with `/*` and end with `*/`.  
Any text between `/*` and `*/` will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

```
<!DOCTYPE html>
<html>
<body>
    <h1 id="myH"></h1>
    <p id="myP"></p>
    <script>
        /*
        The code below will change
        the heading with id = "myH"
        and the paragraph with id = "myP"
        */
    </script>
</body>
</html>
```

```
*/
document.getElementById("myH").innerHTML = "JavaScript Comments";
document.getElementById("myP").innerHTML = "My first paragraph.";
</script>
</body>
</html>
```

Output:

**JavaScript Comments**  
My first paragraph.

**Note:** It is most common to use single line comments.

Block comments are often used for formal documentation.

## Lesson 6. JavaScript Variables

Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using var
- Using let
- Using const

In this first example, x, y, and z are undeclared variables.

They are automatically declared when first used:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, x, y, and z are undeclared.</p>
<p>They are automatically declared when first used.</p>
<p id="demo"></p>
<script>
x = 5;
y = 6;
z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>
</body>
</html>
```

Output:

# JavaScript Variables

In this example, x, y, and z are undeclared.

They are automatically declared when first used.

The value of z is: 11

**Note:** It is considered good programming practice to always declare variables before use.

From the examples you can guess:

- x stores the value 5
- y stores the value 6
- z stores the value 11

Example using var

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, x, y, and z are variables.</p>
<p id="demo"></p>
<script>
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>
</body>
</html>
```

Output:

# JavaScript Variables

In this example, x, y, and z are variables.

The value of z is: 11

**Note:**

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

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

The var keyword should only be used in code written for older browsers.

Example using let

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

```
<h1>JavaScript Variables</h1>
<p>In this example, x, y, and z are variables.</p>
<p id="demo"></p>
<script>
let x = 5;
let y = 6;
let z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>
</body>
</html>
```

Output:

JavaScript Variables

In this example, x, y, and z are variables.

The value of z is: 11

Example using const

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, x, y, and z are variables.</p>
<p id="demo"></p>
<script>
const x = 5;
const y = 6;
const z = x + y;
document.getElementById("demo").innerHTML =
"The value of z is: " + z;
</script>
</body>
</html>
```

Output:

JavaScript Variables

In this example, x, y, and z are variables.

The value of z is: 11

Mixed Example

```
<!DOCTYPE html>
```

```
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, price1, price2, and total are variables.</p>
<p id="demo"></p>
<script>
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
document.getElementById("demo").innerHTML =
"The total is: " + total;
</script>
</body>
</html>
```

Output:

# JavaScript Variables

In this example, price1, price2, and total are variables.

The total is: 11

The two variables price1 and price2 are declared with the const keyword.

- These are constant values and cannot be changed.
- The variable total is declared with the let keyword.
- The value total can be changed.

Notes:

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.

1. Just Like Algebra

Just like in algebra, variables hold values:

```
let x = 5;
let y = 6;
```

Just like in algebra, variables are used in expressions:

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

From the example above, you can guess that the total is calculated to be 11.

**Note:** Variables are containers for storing values.

2. JavaScript Identifiers

All JavaScript variables must be identified with unique names.



These unique names are called identifiers.  
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.

**Note:** JavaScript identifiers are case-sensitive.

### 3. The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x. (It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

**Note:** The "equal to" operator is written like == in JavaScript.

### 4. JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".  
In programming, text values are called text strings.  
JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.  
If you put a number in quotes, it will be treated as a text string.

```
<!DOCTYPE html>
<html>
<body>
  <h1>JavaScript Variables</h1>
  <p>Strings are written with quotes.</p>
  <p>Numbers are written without quotes.</p>
  <p id="demo"></p>
  <script>
    const pi = 3.14;
    let person = "John Doe";
    let answer = 'Yes I am!';
    document.getElementById("demo").innerHTML =
    pi + "<br>" + person + "<br>" + answer;
```

```
        </script>
    </body>
</html>
```

Output:

JavaScript Variables

Strings are written with quotes.

Numbers are written without quotes.

3.14

John Doe

Yes I am!

5. Declaring a JavaScript Variable

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the var or the let keyword:

```
var carName;
```

or:

```
let carName;
```

After the declaration, the variable has no value (technically it is 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:

```
let carName = "Volvo";
```

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Variables</h1>
    <p>Create a variable, assign a value to it, and display it:</p>
    <p id="demo"></p>
    <script>
        let carName = "Volvo";
        document.getElementById("demo").innerHTML = carName;
    </script>
</body>
</html>
```

Output:

# JavaScript Variables

Create a variable, assign a value to it, and display it:

Volvo

**Note:** It's a good programming practice to declare all variables at the beginning of a script.

## 6. One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with `let` and separate the variables by comma:

```
<!DOCTYPE html>
<html>
<body>
  <h1>JavaScript Variables</h1>
  <p>You can declare many variables in one statement.</p>
  <p id="demo"></p>
  <script>
    let person = "John Doe", carName = "Volvo", price = 200;
    document.getElementById("demo").innerHTML = carName;
  </script>
</body>
</html>
```

Output:

# JavaScript Variables

You can declare many variables in one statement.

Volvo

A declaration can span multiple lines:

```
<!DOCTYPE html>
<html>
<body>
  <h1>JavaScript Variables</h1>
  <p>You can declare many variables in one statement.</p>
  <p id="demo"></p>
  <script>
    let person = "John Doe",
    carName = "Volvo",
    price = 200;
    document.getElementById("demo").innerHTML = carName;
  </script>
</body>
```

```
</html>
```

Output:

# JavaScript Variables

You can declare many variables in one statement.

Volvo

## 7. Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value undefined.

The variable carName will have the value undefined after the execution of this statement:

```
<!DOCTYPE html>
<html>
<body>
  <h1>JavaScript Variables</h1>
  <p>A variable without a value has the value of:</p>
  <p id="demo"></p>
  <script>
    let carName;
    document.getElementById("demo").innerHTML = carName;
  </script>
</body>
</html>
```

Output:

# JavaScript Variables

A variable without a value has the value of:

undefined

## 8. Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable declared with var, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

```
<!DOCTYPE html>
<html>
<body>
  <h1>JavaScript Variables</h1>
  <p>If you re-declare a JavaScript variable, it will not lose its value.</p>
  <p id="demo"></p>
```

```
<script>
  var carName = "Volvo";
  var carName;
  document.getElementById("demo").innerHTML = carName;
</script>
</body>
</html>
```

# JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

Volvo

**Note:**

You cannot re-declare a variable declared with let or const.

This will not work:

```
let carName = "Volvo";
let carName;
```

9. JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

```
<!DOCTYPE html>
<html>
<body>
  <h1>JavaScript Variables</h1>
  <p>The result of adding 5 + 2 + 3 is:</p>
  <p id="demo"></p>
  <script>
    let x = 5 + 2 + 3;
    document.getElementById("demo").innerHTML = x;
  </script>
</body>
</html>
```

**Output:**

# JavaScript Variables

The result of adding 5 + 2 + 3 is:

10

You can also add strings, but strings will be concatenated:

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

```
<body>
  <h1>JavaScript Variables</h1>
  <p>The result of adding "John" + " " + "Doe" is:</p>
  <p id="demo"></p>
  <script>
    let x = "John" + " " + "Doe";
    document.getElementById("demo").innerHTML = x;
  </script>
</body>
</html>
```

## JavaScript Variables

The result of adding "John" + " " + "Doe" is:

John Doe

Also try this:

```
<!DOCTYPE html>
<html>
<body>
  <h1>JavaScript Variables</h1>
  <p>The result of adding "5" + 2 + 3 is:</p>
  <p id="demo"></p>
  <script>
    let x = "5" + 2 + 3;
    document.getElementById("demo").innerHTML = x;
  </script>
</body>
</html>
```

## JavaScript Variables

The result of adding "5" + 2 + 3 is:

523

**Note:** If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Now try this:

```
<!DOCTYPE html>
<html>
<body>
  <h1>JavaScript Variables</h1>
  <p>The result of adding 2 + 3 + "5" is:</p>
  <p id="demo"></p>
  <script>
```

```
        let x = 2 + 3 + "5";
        document.getElementById("demo").innerHTML = x;
    </script>
</body>
</html>
```

Output:

JavaScript Variables

The result of adding 2 + 3 + "5" is:

55

10. JavaScript Dollar Sign \$

Since JavaScript treats a dollar sign as a letter, identifiers containing \$ are valid variable names:

```
<!DOCTYPE html>
<html>
<body>
    <h1>JavaScript Variables</h1>
    <p>The dollar sign is treated as a letter in JavaScript names.</p>
    <p id="demo"></p>
    <script>
        let $$$ = 2;
        let $myMoney = 5;
        document.getElementById("demo").innerHTML = $$$ + $myMoney;
    </script>
</body>
</html>
```

Output:

JavaScript Variables

The dollar sign is treated as a letter in JavaScript names.

7

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function \$ is used to select HTML elements. In jQuery \$("p"); means "select all p elements".

11. JavaScript Underscore (\_)

Since JavaScript treats underscore as a letter, identifiers containing \_ are valid variable names:

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

```
<body>
  <h1>JavaScript Variables</h1>
  <p>The underscore is treated as a letter in JavaScript names.</p>
  <p id="demo"></p>
  <script>
    let _x = 2;
    let _100 = 5;
    document.getElementById("demo").innerHTML = _x + _100;
  </script>
</body>
</html>
```

Output:

**JavaScript Variables**

The underscore is treated as a letter in JavaScript names.

7

Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (hidden)" variables.

## Lesson 7. 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

### 1. Block Scope

Before ES6 (2015), JavaScript did not have Block Scope.

JavaScript had Global Scope and Function Scope.

ES6 introduced the two new JavaScript keywords: let and const.

These two keywords provided Block Scope in JavaScript:

Example

Variables declared inside a { } block cannot be accessed from outside the block:

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

### 2. Global Scope



Variables declared with the var always have Global Scope.  
Variables declared with the var keyword can NOT have block scope:  
Example

```
Variables declared with varinside a { } block can be accessed from outside the block:
{
  var x = 2;
}
// x CAN be used here
```

3. Cannot be Redeclared

Variables defined with let can not be redeclared.  
You can not accidentally redeclare a variable declared with let.  
With let you can not do this:

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

Variables defined with var can be redeclared.  
With var you can do this:

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

4. Redeclaring Variables

Redeclaring a variable using the var keyword can impose problems.  
Redeclaring a variable inside a block will also redeclare the variable outside the block:

```
var x = 10;
// Here x is 10
{
  var x = 2;
  // Here x is 2
}
// Here x is 2
```

Redeclaring a variable using the let keyword can solve this problem.  
Redeclaring a variable inside a block will not redeclare the variable outside the block:  
Example

```
let x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

5. Difference Between var, let and const

Table 2.2 Difference Between var, let and const (w3school.com, n.d.)

	Scope	Redeclare	Reassign	Hoisted	Binds this
--	-------	-----------	----------	---------	------------

var	No	Yes	Yes	Yes	Yes
let	Yes	No	Yes	No	No
const	Yes	No	No	No	No

### 6. What is Good?

- let and const have block scope.
- let and const can not be redeclared.
- let and const must be declared before use.
- let and const does not bind to this.
- let and const are not hoisted.

### 7. What is Not Good?

- var does not have to be declared.
- var is hoisted.
- var binds to this.

### 8. Browser Support

The let and const keywords are not supported in Internet Explorer 11 or earlier.  
The following table defines the first browser versions with full support:

				
Chrome 49	Edge 12	Firefox 36	Safari 11	Opera 36
Mar, 2016	Jul, 2015	Jan, 2015	Sep, 2017	Mar, 2016

Figure 2.1 JavaScript Browser Support

Source: w3schools.com (n.d.)

### 9. Redeclaring

Redeclaring a JavaScript variable with var is allowed anywhere in a program:

```
var x = 2;
// Now x is 2
var x = 3;
// Now x is 3
```

With let, redeclaring a variable in the same block is NOT allowed:

```
var x = 2; // Allowed
let x = 3; // Not allowed
{
  let x = 2; // Allowed
  let x = 3; // Not allowed
}
{
  let x = 2; // Allowed
  var x = 3; // Not allowed
}
```

Redeclaring a variable with let, in another block, IS allowed:

```
let x = 2; // Allowed
```

```
{
let x = 3; // Allowed
}

{
let x = 4; // Allowed
}

//Result value of x = 2
```

10. Let Hoisting

Variables defined with var are hoisted to the top and can be initialized at any time.  
Meaning: You can use the variable before it is declared:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Hoisting</h2>
    <p>With <b>var</b>, you can use a variable before it is declared:</p>
    <p id="demo"></p>
    <script>
        carName = "Volvo";
        document.getElementById("demo").innerHTML = carName;
        var carName;
    </script>
</body>
</html>
```

Output:

JavaScript Hoisting

With var, you can use a variable before it is declared:

Volvo

If you want to learn more about hoisting, study the chapter JavaScript Hoisting.  
Variables defined with let are also hoisted to the top of the block, but not initialized.  
Meaning: Using a let variable before it is declared will result in a ReferenceError:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript Hoisting</h2>
    <p>With <b>let</b>, you cannot use a variable before it is declared.</p>
    <p id="demo"></p>
    <script>
        try {
            carName = "Saab";
```

```
        let carName = "Volvo";
    }
    catch(err) {
        document.getElementById("demo").innerHTML = err;
    }
</script>
</body>
</html>
```

## Lesson 8. 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

### 1. Cannot be Reassigned

A variable defined with the const keyword cannot be reassigned:

```
<!DOCTYPE html>
<html>
<body>
    <h2>JavaScript const</h2>
    <p id="demo"></p>
    <script>
        try {
            const PI = 3.141592653589793;
            PI = 3.14;
        }
        catch (err) {
            document.getElementById("demo").innerHTML = err;
        }
    </script>
</body>
</html>
```

Output:

JavaScript const

TypeError: Assignment to constant variable.

### 2. Must be Assigned

JavaScript const variables must be assigned a value when they are declared:

Correct

```
const PI = 3.14159265359;
```

Incorrect

```
const PI;  
PI = 3.14159265359;
```

**Note:**

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

### 3. Constant Objects and Arrays

The keyword const is a little misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

- Reassign a constant value
- Reassign a constant array
- Reassign a constant object

But you CAN:

- Change the elements of constant array
- Change the properties of constant object

### 4. Constant Arrays

You can change the elements of a constant array:

```
<!DOCTYPE html>  
<html>  
<body>  
  <h2>JavaScript const</h2>  
  <p>Declaring a constant array does NOT make the elements  
  unchangeable:</p>  
  <p id="demo"></p>  
  <script>  
    // Create an Array:  
    const cars = ["Saab", "Volvo", "BMW"];  
    // Change an element:  
    cars[0] = "Toyota";  
    // Add an element:
```

```

cars.push("Audi");
// Display the Array:
document.getElementById("demo").innerHTML = cars;
</script>
</body>
</html>

```

Output:

## JavaScript const

Declaring a constant array does NOT make the elements unchangeable:

Toyota, Volvo, BMW, Audi

But you can NOT reassign the array:

```

<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript const</h2>
  <p>You can NOT reassign a constant array:</p>
  <p id="demo"></p>
  <script>
    try {
      const cars = ["Saab", "Volvo", "BMW"];
      cars = ["Toyota", "Volvo", "Audi"];
    }
    catch (err) {
      document.getElementById("demo").innerHTML = err;
    }
  </script>
</body>
</html>

```

Output:

## JavaScript const

You can NOT reassign a constant array:

TypeError: Assignment to constant variable.

### 5. You can change the properties of a constant object:

```

<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript const</h2>

```

```
<p>Declaring a constant object does NOT make the objects properties
unchangeable:</p>
<p id="demo"></p>
<script>
// Create an object:
const car = {type:"Fiat", model:"500", color:"white"};
// Change a property:
car.color = "red";
// Add a property:
car.owner = "Johnson";
// Display the property:
document.getElementById("demo").innerHTML = "Car owner is " +
car.owner;
</script>
</body>
</html>
```

Output:

JavaScript const

Declaring a constant object does NOT make the objects properties unchangeable:

Car owner is Johnson

But you can NOT reassign the object:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript const</h2>
  <p>You can NOT reassign a constant object:</p>
  <p id="demo"></p>
  <script>
    try {
      const car = {type:"Fiat", model:"500", color:"white"};
      car = {type:"Volvo", model:"EX60", color:"red"};
    }
    catch (err) {
      document.getElementById("demo").innerHTML = err;
    }
  </script>
</body>
</html>
```

Output:

## JavaScript const

You can NOT reassign a constant object:

`TypeError: Assignment to constant variable.`

### 6. Block Scope

Declaring a variable with `const` is similar to `let` when it comes to Block Scope.

The `x` declared in the block, in this example, is not the same as the `x` declared outside the block:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript const variables has block scope</h2>
  <p id="demo"></p>
  <script>
    const x = 10;
    // Here x is 10
    {
      const x = 2;
      // Here x is 2
    }
    // Here x is 10
    document.getElementById("demo").innerHTML = "x is " + x;
  </script>
</body>
</html>
```

Output:

**JavaScript const variables has block scope**  
x is 10

You can learn more about block scope in the chapter JavaScript Scope.

### 7. Redeclaring

Redeclaring a JavaScript `var` variable is allowed anywhere in a program:

Example

```
var x = 2;    // Allowed
var x = 3;    // Allowed
x = 4;        // Allowed
```

Redeclaring an existing `var` or `let` variable to `const`, in the same scope, is not allowed:

Example

```
var x = 2;    // Allowed
```



```
const x = 2; // Not allowed
{
let x = 2;    // Allowed
const x = 2; // Not allowed
}
{
const x = 2; // Allowed
const x = 2; // Not allowed
}
```

Reassigning an existing const variable, in the same scope, is not allowed:

Example

```
const x = 2; // Allowed
x = 2;       // Not allowed
var x = 2;   // Not allowed
let x = 2;   // Not allowed
const x = 2; // Not allowed
{
const x = 2; // Allowed
x = 2;       // Not allowed
var x = 2;   // Not allowed
let x = 2;   // Not allowed
const x = 2; // Not allowed
}
```

Redeclaring a variable with const, in another scope, or in another block, is allowed:

```
const x = 2; // Allowed
{
const x = 3; // Allowed
}
{
const x = 4; // Allowed
}
```

## 8. Hoisting

Variables defined with var are hoisted to the top and can be initialized at any time.

Meaning: You can use the variable before it is declared:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript Hoisting</h2>
  <p>With <b>var</b>, you can use a variable before it is declared:</p>
  <p id="demo"></p>
```

```
<script>
  carName = "Volvo";
  document.getElementById("demo").innerHTML = carName;
  var carName;
</script>
</body>
</html>
```

Output:

JavaScript Hoisting

With **var**, you can use a variable before it is declared:

Volvo

If you want to learn more about hoisting, study the chapter JavaScript Hoisting.

Variables defined with `const` are also hoisted to the top, but not initialized.

Meaning: Using a `const` variable before it is declared will result in a `ReferenceError`:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript Hoisting</h2>
  <p>With const, you cannot use a variable before it is declared:</p>
  <p id="demo"></p>
  <script>
    try {
      alert(carName);
      const carName = "Volvo";
    }
    catch (err) {
      document.getElementById("demo").innerHTML = err;
    }
  </script>
</body>
</html>
```

Output:

JavaScript Hoisting

With **const**, you cannot use a variable before it is declared:

ReferenceError: Cannot access 'carName' before initialization



Assessment Task

### Lab Setup:

- Open a text editor (e.g., Visual Studio Code, Sublime Text).
- Create a new HTML file and save it with an appropriate name (e.g., lastname\_module2\_lab.js).
- Create a new JavaScript file (e.g., script.js) linked to the HTML file.

### Lab Tasks:

#### Task 1: Understanding JavaScript Placement (2.1)

1. Explore the placement of JavaScript code within an HTML document.
2. Create a `<script>` tag in the HTML file and place it in the `<head>` section.
3. Write a simple JavaScript code inside the script tag, such as `console.log("Hello, JavaScript!");`.
4. Open the browser's developer console to observe the output.

#### Task 2: Exploring JS Output (2.2)

1. Implement a JavaScript function that uses `document.write()` to output a message to the HTML document.
2. Create an HTML button, and on button click, call the function to display the message dynamically.
3. Explore an alternative output method using `console.log()` for debugging purposes.

#### Task 3: Grasping JS Statements (2.3)

1. Create a JavaScript function with multiple statements.
2. Include variables, mathematical operations, and conditional statements within the function.
3. Test the function by calling it and observe the results in the console.

#### Task 4: Working with JS Variables (2.4)

1. Declare variables of different types (string, number, boolean) and assign values to them.
2. Display the values using `console.log()`.
3. Experiment with updating variable values and observe the changes.

#### Task 5: Utilizing JS Let and Const (2.5, 2.6)

1. Explore the use of `let` for variable declaration within a block scope.
2. Declare a variable using `const` and attempt to reassign a value to it.
3. Compare and contrast the behavior of `let` and `const` in different scenarios.



### Summary

- Module 2 delves into the foundational aspects of JavaScript, leveraging insights from w3schools.com. The module initiates with an exploration of JavaScript's strategic placement within HTML documents, elucidating the use of `<script>` tags for effective code execution. Learners are introduced to various methods of outputting information,

including `document.write()`, `console.log()`, and techniques for modifying HTML elements.

- The module systematically unfolds the nature of JavaScript statements, emphasizing their role as individual instructions within a program and highlighting the significance of semicolons as statement terminators. It progresses to unravel the concept of variables in JavaScript, providing a comprehensive understanding of variable declaration, assignment, and the handling of different variable types.
- Syntax rules become a focal point, covering naming conventions, case sensitivity, and the importance of whitespace to facilitate the creation of clean and readable code. The inclusion of comments is explored as a means of annotating code, enhancing readability, and providing explanatory notes to ensure code comprehension.
- Specific keywords like `let` and `const` are introduced, with a dedicated focus on their roles in variable declaration, block scope, and immutability. The module comprehensively addresses various operators in JavaScript, spanning arithmetic, assignment, comparison, logical, and bitwise operators, serving as foundational tools for executing variable operations.



## References

### Book

- Osmani, A. (2023). Learning JavaScript Design Patterns.(A JavaScript and React Developer’s Guide) Second Edition.United States of America, O’Reilly Media, Inc. 105 Gravenstein Highway North, Sebastopol, CA 95472.

### Website

- W3Schools.com (n.d.). Retrieved from [https://www.w3schools.com/js/js\\_output.asp](https://www.w3schools.com/js/js_output.asp)

## **MODULE 3**

# **JAVASCRIPT BASICS - PART 2: OPERATORS, FUNCTIONS, OBJECTS, AND EVENTS**



## Introduction

The focus shifts towards advanced JavaScript concepts, building upon the foundational knowledge established in Part 1. Learners delve into the intricacies of operators, covering arithmetic, assignment, comparison, logical, and bitwise operators. The module progresses to elucidate the concept of functions, showcasing their role in code organization and reuse. JavaScript objects take center stage, emphasizing their significance in grouping related data and functions, with a focus on properties and methods. The module concludes with a comprehensive exploration of JavaScript events, providing insights into user and browser-generated occurrences and setting the stage for the exploration of more advanced JavaScript concepts.



At the end of this module, students should be able to:

1. Understand and apply various operators in JavaScript, including arithmetic, assignment, comparison, logical, and bitwise operators.
2. Attain proficiency in JavaScript operators, functions, and object-oriented concepts.
3. Develop and showcase a small JavaScript project that integrates learned fundamentals and practices values integration.

### Lesson 1. JavaScript Operators

The Addition Operator + adds numbers:

The Assignment Operator = assigns a value to a variable.

↓       ↓  
**x + y = z**

#### 1. JavaScript Assignment

The Assignment Operator (=) assigns a value to a variable:

##### Assignment Examples

```
let x = 10;  
  
// Assign the value 5 to x  
let x = 5;  
  
// Assign the value 2 to y  
let y = 2;  
  
// Assign the value x + y to z:  
let z = x + y;
```

#### 2. JavaScript Addition

The Addition Operator (+) adds numbers:

Adding

```
let x = 5;
let y = 2;
let z = x + y;
```

3. JavaScript Multiplication

The Multiplication Operator (\*) multiplies numbers:

```
let x = 5;
let y = 2;
let z = x * y;
```

4. Types of JavaScript Operators

There are different types of JavaScript operators:

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

5. JavaScript Arithmetic Operators

Arithmetic Operators are used to perform arithmetic on numbers:

Arithmetic Operators Example

```
let a = 3;
let x = (100 + 50) * a;
```

Table 3.1 JavaScript Arithmetic Operators (w3school.com, n.d.)

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

**Note:** Arithmetic operators are fully described in the JS Arithmetic chapter.

6. JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

The Addition Assignment Operator (+=) adds a value to a variable.

Table 3.2 JavaScript Assignment Operators (w3school.com, n.d.)

Operator	Example
----------	---------

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

**Note:** Comparison operators are fully described in the JS Comparisons module.

7. JavaScript String Comparison

All the comparison operators above can also be used on strings:

Example

```
let text1 = "A";
let text2 = "B";
let result = text1 < text2;
```

Note that strings are compared alphabetically:

Example

```
let text1 = "20";
let text2 = "5";
let result = text1 < text2;
```

8. JavaScript String Addition

The + can also be used to add (concatenate) strings:

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

The += assignment operator can also be used to add (concatenate) strings:

Example

```
let text1 = "What a very ";
text1 += "nice day";
```

The result of text1 will be:

```
What a very nice day
```

**Note:** When used on strings, the + operator is called the concatenation operator.

9. Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
let x = 5 + 5;
let y = "5" + 5;
```

let z = "Hello" + 5;

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

10  
55  
Hello5

**Note:** If you add a number and a string, the result will be a string!

10. JavaScript Logical Operators

Table 3.3 JavaScript Logical Operators (w3school.com, n.d.)

Operator	Description
&&	logical and
	logical or
!	logical not

**Note:** Logical operators are fully described in the JS Comparisons module.

11. JavaScript Type Operators

Table 3.4 JavaScript Type Operators (w3school.com, n.d.)

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

**Note:** Type operators are fully described in the JS Type Conversion module.

12. JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.  
Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Table 3.5 JavaScript Type Operators (w3school.com, n.d.)

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	1	1
	OR	5   1	0101   0001	101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	10	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	10	2

**Note:** The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.  
Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.



1111111111111111111111111111111111010

## Lesson 2. JavaScript Data Types

- String
- Number
- BigInt
- Boolean
- Undefined
- Null
- Symbol
- Object

The object data type can contain:

- An object
- An array
- A date

```
// 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");
```

## 1. The Concept of Data Types

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

```
let x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

```
let x = "16" + "Volvo";
```

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

Example

```
<!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

Example

```
<!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 evaluates expressions from left to right. Different sequences can produce different results:

JavaScript:

```
<!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>
```

Result

## JavaScript

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

20Volvo

JavaScript:

```
<!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

2. JavaScript Types are Dynamic

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

```
<!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>
```

Output:

## JavaScript Data Types

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

John

3. JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

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

```
<!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

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript Strings</h2>
  <p>You can use quotes inside a string, as long as they don't match the
quotes surrounding the string:</p>
  <p id="demo"></p>
  <script>
let answer1 = "It's alright";
let answer2 = "He is called 'Johnny'";
let answer3 = 'He is called "Johnny"';
document.getElementById("demo").innerHTML =
answer1 + "<br>" +
answer2 + "<br>" +
answer3;
  </script>
</body>
</html>
```

Output:

## JavaScript Strings

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
It's alright
He is called 'Johnny'
He is called "Johnny"
```

### 4. 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
```

### 5. Exponential Notation

Extra large or extra small numbers can be written with scientific (exponential) notation:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript Numbers</h2>
  <p>Extra large or extra small numbers can be written with scientific
  (exponential) notation:</p>
  <p id="demo"></p>
  <script>
```

```
let y = 123e5;
let z = 123e-5;
document.getElementById("demo").innerHTML =
y + "<br>" + z;
</script>
</body>
</html>
```

Output:

JavaScript Numbers

Extra large or extra small numbers can be written with scientific (exponential) notation:

123000000  
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).

You will learn more about numbers later in this tutorial.

### Lesson 3. JavaScript 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).

Example

```
// Function to compute the product of p1 and p2
function myFunction(p1, p2) {
  return p1 * p2;
}
```

#### 1. JavaScript Function Syntax

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
}
```

```
}

```

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.

2. **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)

3. **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":

Example

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

```
<!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:

12

4. **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.

5. The () Operator

The () operator invokes (calls) the function:

Example

Convert Fahrenheit to Celsius:

```
<!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>
```

Output:

JavaScript Functions

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

25

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

Example

```
<!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

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>
```

Output:

## JavaScript Functions

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

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

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

### 6. Functions Used as Variable Values

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

Example

Instead of using a variable to store the return value of a function:

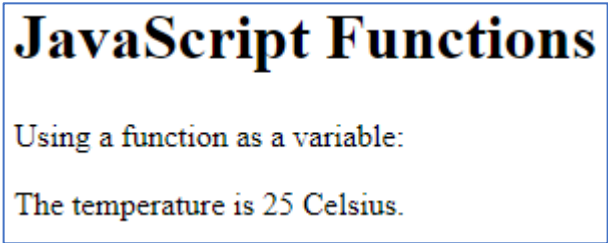
```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
<!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>
```

Output:



7. Local Variables

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

```
<!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

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.

Lesson 4. JavaScript Objects

1. Real Life Objects, Properties, and Methods

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()

Figure 3.1 Real Life Objects, Properties, and Methods

Source: w3schools.com (n.d.)

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.

2. **JavaScript 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>
```

Output:

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>
```

Output:

JavaScript Objects

The car type is Fiat

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

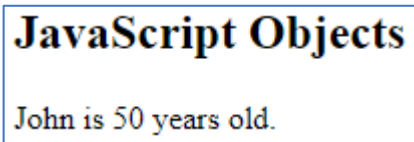
**Note:** It is a common practice to declare objects with the const keyword.  
Learn more about using const with objects in the chapter: JS Const.

3. Object Definition

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

```
<!DOCTYPE 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>
```

Output:



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

```
<!DOCTYPE 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>
```

Output:

JavaScript Objects

John is 50 years old.

4. Object Properties

The name:values pairs in JavaScript objects are called properties:

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

5. Accessing Object Properties

You can access object properties in two ways:

```
objectName.propertyName
```

Or

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

Example1

```
<!DOCTYPE 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>
```

Output:

## JavaScript Objects

There are two different ways to access an object property.

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

John Doe

Example2

```
<!DOCTYPE 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>
```

Output:

## 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.

6. Object Methods

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

Table 3.7 Object Methods (w3school.com, n.d.)

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.

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

In the example above, this refers to the person object:  
this.firstName means the firstName property of person.  
this.lastName means the lastName property of person.

7. What is this?

In JavaScript, the this keyword refers to an object.  
Which object depends on how this is being invoked (used or called).  
The this keyword refers to different objects depending on how it is used:

- In an object method, this refers to the object.
- Alone, this refers to the global object.
- In a function, this refers to the global object.
- In a function, in strict mode, this is undefined.
- In an event, this refers to the element that received the event.
- Methods like call(), apply(), and bind() can refer this to any object.

**Note:** this is not a variable. It is a keyword. You cannot change the value of this.

8. The this Keyword

In a function definition, this refers to the "owner" of the function.  
In the example above, this is the person object that "owns" the fullName function.  
In other words, this.firstName means the firstName property of this object.

9. Accessing Object Methods

You access an object method with the following syntax:

```
objectName.methodName()
```

Example

```
<!DOCTYPE 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>
```

Output:

**JavaScript Objects**

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

John Doe

If you access a method without the () parentheses, it will return the function definition:

```
<!DOCTYPE html>
<html>
<body>
  <h2>JavaScript Objects</h2>
  <p>If you access an object method without (), it will return the function
  definition:</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>
```

Output:

## JavaScript Objects

If you access an object method without (), it will return the function definition:

```
function() { return this.firstName + " " + this.lastName; }
```

### 10. Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

```
x = new String();    // Declares x as a String object
y = new Number();    // Declares y as a Number object
z = new Boolean();    // Declares z as a Boolean object
```

Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

## Lesson 5. JavaScript Events

HTML events are "things" that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can "react" on these events.

### 1. HTML 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'>
```

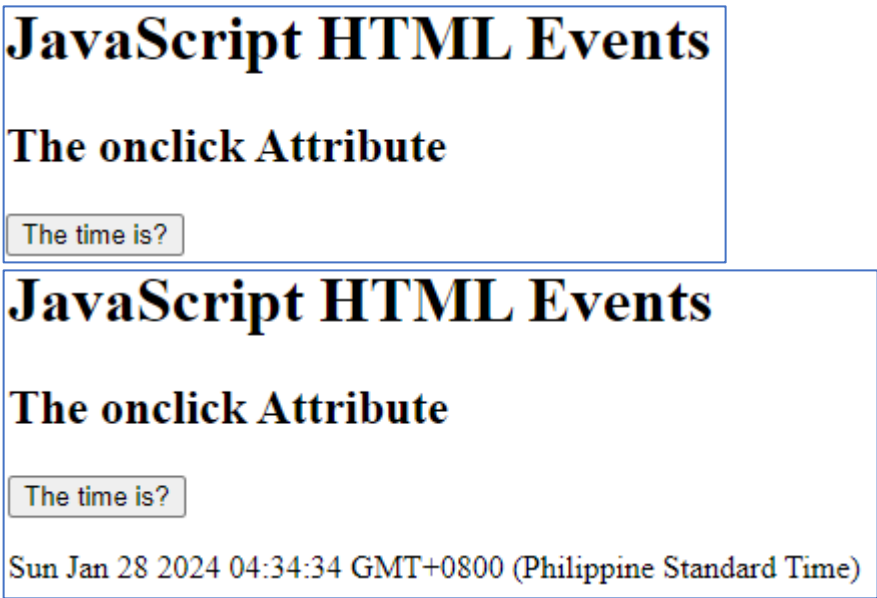
With double quotes:

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

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

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

Output:



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

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

```
<!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>
```

Output

JavaScript HTML Events

The onclick Attribute

The time is?

JavaScript HTML Events

The onclick Attribute

Sun Jan 28 2024 04:36:13 GMT+0800 (Philippine Standard Time)

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

Example

```
<!DOCTYPE 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>
```

Output

# 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?

Sun Jan 28 2024 04:38:35 GMT+0800 (Philippine Standard Time)

### 2. Common HTML 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

### 3. JavaScript Event Handlers

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



### Assessment Task

**Lab Setup:**

- Open a text editor (e.g., Visual Studio Code, Sublime Text).
- Create a new HTML file and save it with an appropriate name (e.g., lastname\_module3\_lab.html).
- Create a new JavaScript file (e.g., script.js) linked to the HTML file.

**Lab Tasks:****Task 1: Exploring JS Operators (3.1)**

1. Declare two variables, num1 and num2, and assign numeric values to them.
2. Utilize various operators (arithmetic, assignment, comparison, logical) to perform operations on these variables.
3. Display the results using console.log().

**Task 2: Understanding JS Functions (3.2)**

1. Create a function named calculateArea that calculates the area of a rectangle.
2. The function should take parameters for length and width, perform the calculation, and return the result.
3. Call the function with different sets of values and display the results.

**Task 3: Working with JS Objects (3.3)**

1. Define an object named person with properties such as name, age, and occupation.
2. Display the values of these properties using console.log().
3. Modify one or more properties of the object and display the updated values.

**Task 4: Handling JS Events (3.4)**

1. Create an HTML button with an ID (e.g., <button id="myButton">Click me!</button>).
2. In the JavaScript file, add an event listener to the button that triggers an alert saying "Button Clicked!" when the button is clicked.
3. Explore additional events (e.g., onmouseover, onmouseout) and implement corresponding event handlers.

**Summary**

- The focus shifts towards more advanced aspects of JavaScript. The concept of functions is elucidated, covering declarations, parameters, return values, and the organizational benefits of utilizing functions for code structure and reusability. Learners gain proficiency in JavaScript objects, understanding their significance as fundamental data types that allow the grouping of related data and functions, with emphasis on object properties and methods.

- The module concludes with a comprehensive examination of JavaScript events, providing insights into user and browser-generated occurrences. It expounds on event handling through event listeners and the mechanisms for responding to user interactions. This conclusion marks the end of the foundational chapter, setting the stage for learners to explore advanced JavaScript concepts with a solid understanding of syntax, operators, functions, objects, and events.



## References

### Book

- Osmani, A. (2023). Learning JavaScript Design Patterns.(A JavaScript and React Developer’s Guide) Second Edition.United States of America, O’Reilly Media, Inc. 105 Gravenstein Highway North, Sebastopol, CA 95472.

### Website

- W3Schools.com (n.d.). Retrieved from [https://www.w3schools.com/js/js\\_output.asp](https://www.w3schools.com/js/js_output.asp)

-                   **END OF MODULE FOR PRELIMINARY TERM PERIOD –**  
SUBMISSION OF THE ASSESSMENT AND EXAMINATION FOR PRELIM PERIOD WILL  
  BE ANNOUNCED VIA iLearnU or GC  
MAKE SURE TO CHECK THE DATES AND NOT FORGET TO TAKE IT AS SCHEDULED