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 4
JAVASCRIPT STRINGS, NUMBERS ARRAYS AND DATE FORMATS

Introduction

JavaScript is a versatile programming language widely used for web development, offering powerful tools for manipulating strings, handling numbers, working with arrays, and managing dates. This course provides an in-depth exploration of these fundamental data types and their associated methods in JavaScript. Through a series of lessons, you will learn how to work with strings, including string methods for searching and formatting. You'll also delve into the intricacies of JavaScript numbers, covering standard numeric operations, BigInt for handling large integers, and various number methods and properties. Additionally, the course covers arrays, teaching you array manipulation techniques, search methods, sorting, and iteration. Finally, you'll explore the Date object in JavaScript, learning how to work with dates, different date formats, and various methods for retrieving and setting date components. By the end of this course, you'll have a solid understanding of how to effectively utilize JavaScript's string, number, array, and date functionalities in your web development projects.

Learning Outcomes

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

1. Manipulate and utilize strings effectively in your JavaScript code.
2. Develop a solid understanding of numerical operations in JavaScript, including BigInt, number methods, and properties.
3. Gain expertise in working with JavaScript dates, including understanding date formats and utilizing various date methods for retrieval and manipulation.
4. Demonstrate the practical application of string manipulation, numerical operations, array handling, and date formatting through a small JavaScript project.

Lesson 1. JavaScript Strings

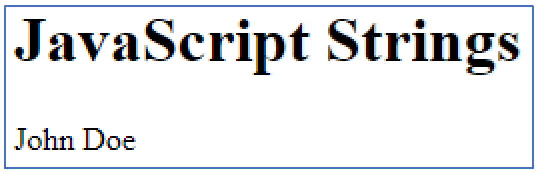
- Strings are for storing text
- Strings are written with quotes

A. Using Quotes

A JavaScript string is zero or more characters written inside quotes.

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Strings</h1>
        <p id="demo"></p>
        <script>
        let text = "John Doe";  // String written inside quotes
        document.getElementById("demo").innerHTML = text;
        </script>
        </body>
</html>
```

Output:

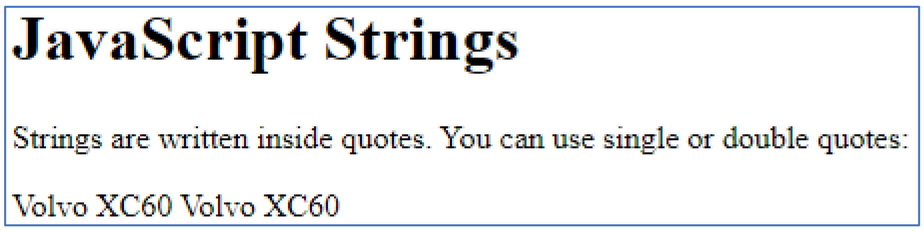


```
JavaScript Strings

John Doe
```

You can use single or double quotes:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Strings</h1>
<p>Strings are written inside quotes. You can use single or double quotes:</p>
<p id="demo"></p>
<script>
let carName1 = "Volvo XC60"; // Double quotes
let carName2 = 'Volvo XC60'; // Single quotes
document.getElementById("demo").innerHTML =
carName1 + " " + carName2;
</script>
</body>
</html>
```

Output:



```
JavaScript Strings

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

Volvo XC60 Volvo XC60
```

Note

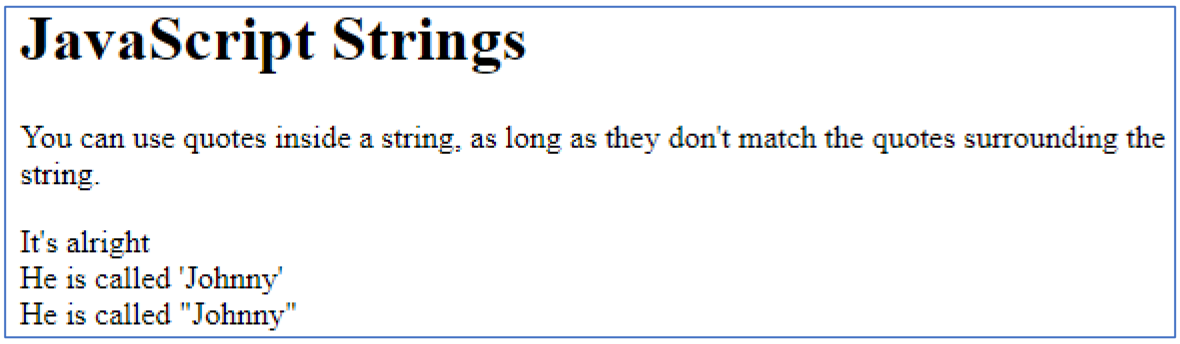
- Strings created with single or double quotes works the same.
- There is no difference between the two.

B. Quotes Inside Quotes

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

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Strings</h1>
<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"
```

C. Template Strings

Templates were introduced with ES6 (JavaScript 2016).

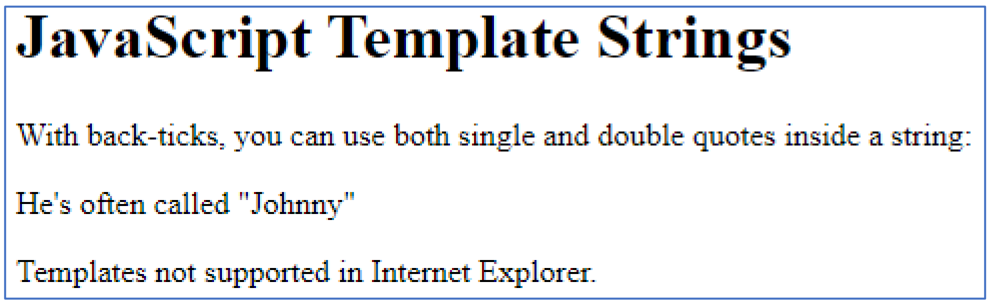
Templates are strings enclosed in backticks (`This is a template string`).

Templates allow single and double quotes inside a string:

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Template Strings</h1>
        <p>With back-ticks, you can use both single and double quotes inside a
        string:</p>
        <p id="demo"></p>
        <p>Templates not supported in Internet Explorer.</p>
```

```
          <script>
          let text = `He's often called "Johnny"`;
          document.getElementById("demo").innerHTML = text;
          </script>
</body>
</html>
```

Output:



JavaScript Template Strings

With back-ticks, you can use both single and double quotes inside a string:

He's often called "Johnny"

Templates not supported in Internet Explorer.

Note:

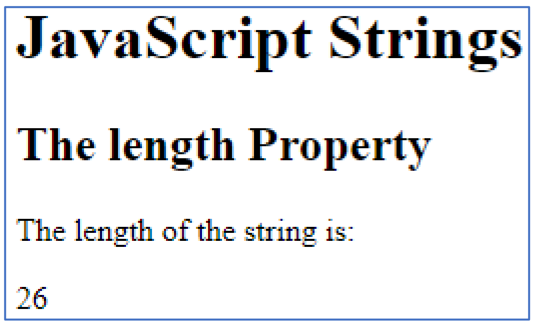
- Templates are not supported in Internet Explorer.

D. String Length

To find the length of a string, use the built-in length property:

```
<!DOCTYPE html>
<html>
<body>
          <h1>JavaScript Strings</h1>
          <h2>The length Property</h2>
          <p>The length of the string is:</p>
          <p id="demo"></p>
          <script>
          let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
          document.getElementById("demo").innerHTML = text.length;
          </script>
</body>
</html>
```

Output:



JavaScript Strings

The length Property

The length of the string is:

26

E. Escape Characters

Because strings must be written within quotes, JavaScript will misunderstand this string:

```
let text = "We are the so-called "Vikings" from the north.";
```

- The string will be chopped to "We are the so-called ".
- To solve this problem, you can use an backslash escape character.
- The backslash escape character (\) turns special characters into string characters:

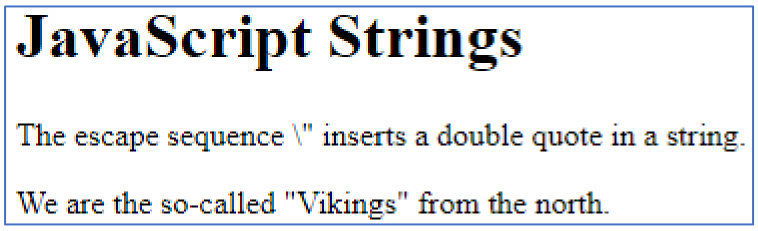
Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Examples:

\" inserts a double quote in a string:

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Strings</h1>
        <p>The escape sequence \" inserts a double quote in a string.</p>
        <p id="demo"></p>
        <script>
        let text = "We are the so-called \"Vikings\" from the north.";
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



JavaScript Strings

The escape sequence \" inserts a double quote in a string.

We are the so-called "Vikings" from the north.

\' inserts a single quote in a string:

```
let text= 'It\'s alright.';
```

\\ inserts a backslash in a string:

```
let text = "The character \\ is called backslash.";
```

Six other escape sequences are valid in JavaScript:

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

Note:

The 6 escape characters above were originally designed to control typewriters, teletypes, and fax machines. They do not make any sense in HTML.

F. Breaking Long Lines

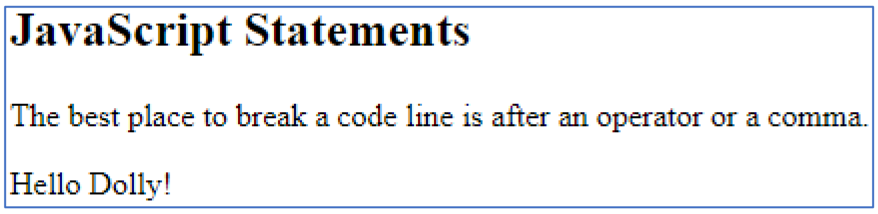
For readability, programmers often like to avoid long code lines.

A safe way to break up a **statement** is after an operator:

Examples:

```
<!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!
```

A safe way to break up a string is by using string addition:

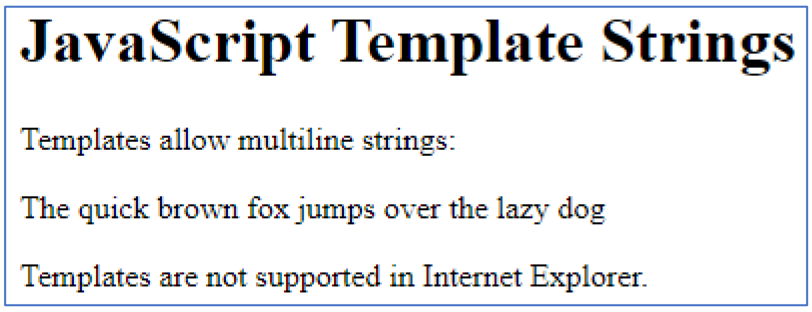
```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Strings</h1>
        <p>The safest way to break a code line in a string is using string
        addition.</p>
        <p id="demo"></p>
        <script>
        document.getElementById("demo").innerHTML = "Hello " +
        "Dolly!";
        </script>
</body>
</html>
```

G. Template Strings

- Templates were introduced with ES6 (JavaScript 2016).
- Templates are strings enclosed in backticks (`This is a template string`).
- Templates allow multiline strings:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Template Strings</h1>
<p>Templates allow multiline strings:</p>
<p id="demo"></p>
<p>Templates are not supported in Internet Explorer.</p>
<script>
let text =
`The quick
brown fox
jumps over
the lazy dog`;
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Output:



```
JavaScript Template Strings

Templates allow multiline strings:

The quick brown fox jumps over the lazy dog

Templates are not supported in Internet Explorer.
```

Note:

Templates are not supported in Internet Explorer.

H. JavaScript Strings as Objects

Normally, JavaScript strings are primitive values, created from literals:

```
let x = "John";
```

But strings can also be defined as objects with the keyword **new**:

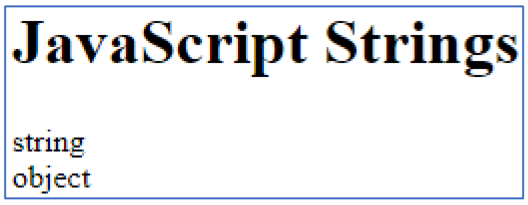
```
let y = new String("John");
```

Example:

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Strings</h1>
        <p id="demo"></p>
        <script>
```

```
          // x is a string
          let x = "John";
          // y is an object
          let y = new String("John");
          document.getElementById("demo").innerHTML =
          typeof x + "<br>" + typeof y;
          </script>
</body>
</html>
```

Output:



JavaScript Strings

```
string
object
```

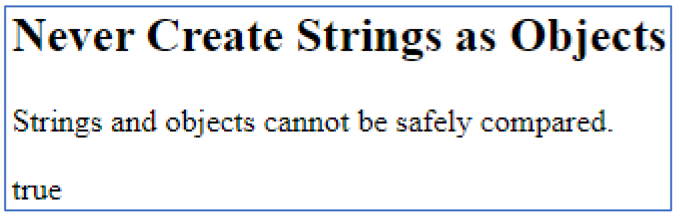
Note:

- Do not create Strings objects.
- The new keyword complicates the code and slows down execution speed.
- String objects can produce unexpected results:

When using the == operator, x and y are equal:

```
<!DOCTYPE html>
<html>
<body>
          <h2>Never Create Strings as Objects</h2>
          <p>Strings and objects cannot be safely compared.</p>
          <p id="demo"></p>
          <script>
          let x = "John";       // x is a string
          let y = new String("John");  // y is an object
          document.getElementById("demo").innerHTML = (x==y);
          </script>
</body>
</html>
```

Output:



Never Create Strings as Objects

Strings and objects cannot be safely compared.

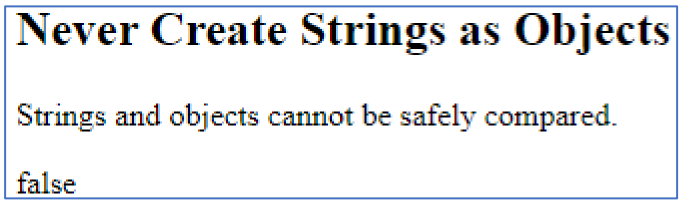
```
true
```

When using the === operator, x and y are not equal:

```
<!DOCTYPE html>
<html>
<body>
          <h2>Never Create Strings as Objects</h2>
```

```
        <p>Strings and objects cannot be safely compared.</p>
        <p id="demo"></p>
        <script>
        let x = "John";        // x is a string
        let y = new String("John");  // y is an object
        document.getElementById("demo").innerHTML = (x===y);
        </script>
</body>
</html>
```

Output:



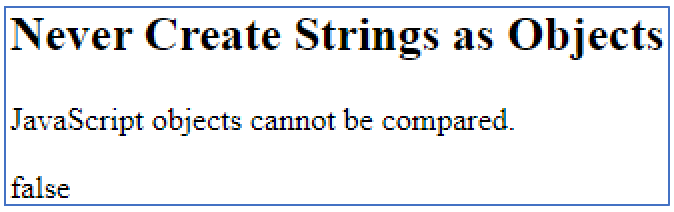
```
Never Create Strings as Objects

Strings and objects cannot be safely compared.

false
```

Note: the difference between (x==y) and (x===y).

(x == y) true or false?

```
<!DOCTYPE html>
<html>
<body>
        <h2>Never Create Strings as Objects</h2>
        <p>JavaScript objects cannot be compared.</p>
        <p id="demo"></p>
        <script>
        let x = new String("John");  // x is an object
        let y = new String("John");  // y is an object
        document.getElementById("demo").innerHTML = (x==y);
        </script>
</body>
</html>
```

Output:

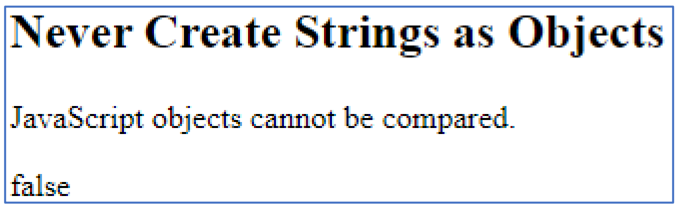


```
Never Create Strings as Objects

JavaScript objects cannot be compared.

false
```

(x === y) true or false?

```
<!DOCTYPE html>
<html>
<body>
        <h2>Never Create Strings as Objects</h2>
        <p>JavaScript objects cannot be compared.</p>
        <p id="demo"></p>
        <script>
        let x = new String("John");  // x is an object
        let y = new String("John");  // y is an object
        document.getElementById("demo").innerHTML = (x===y);
        </script>
</body>
</html>
```

Output:



```
Never Create Strings as Objects

JavaScript objects cannot be compared.

false
```

Note: Comparing two JavaScript objects always returns false.

Lesson 2. JavaScript String Methods

Basic String Methods

Javascript strings are primitive and immutable: All string methods produces a new string without altering the original string.

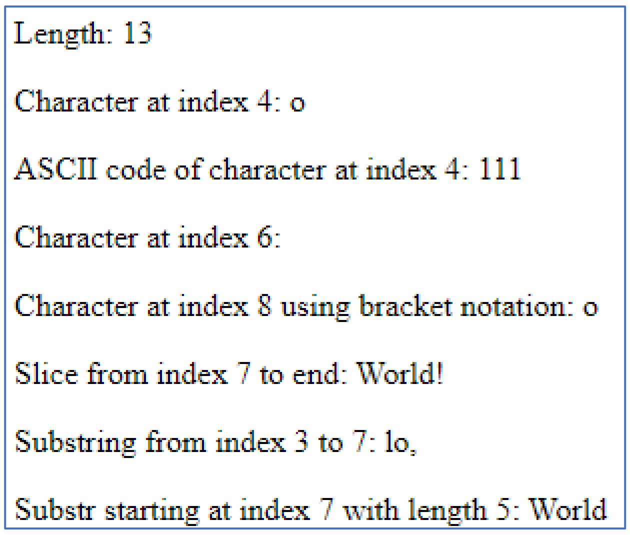
- String length
- String charAt()
- String charCodeAt()
- String at()
- String [ ]
- String slice()
- String substring()
- String substr()
- String toUpperCase()
- String toLowerCase()
- String concat()
- String trim()
- String trimStart()
- String trimEnd()
- String padStart()
- String padEnd()
- String repeat()
- String replace()
- String replaceAll()
- String split()

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>String Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let str = "Hello, World!";
console.log("Original string:", str);

let length = str.length; // String length
document.getElementById("demo").innerHTML += "<p>Length: " + length + "</p>";

let charAtIndex = str.charAt(4); // String charAt()
document.getElementById("demo").innerHTML += "<p>Character at index 4: " +
charAtIndex + "</p>";

let charCodeAtIndex = str.charCodeAt(4); // String charCodeAt()
document.getElementById("demo").innerHTML += "<p>ASCII code of character at
index 4: " + charCodeAtIndex + "</p>";

let atIndex = str.at(6); // String at() (not a built-in method, but similar to charAt())
document.getElementById("demo").innerHTML += "<p>Character at index 6: " +
atIndex + "</p>";

let bracketIndex = str[8]; // String [ ]
document.getElementById("demo").innerHTML += "<p>Character at index 8 using
bracket notation: " + bracketIndex + "</p>";

let sliced = str.slice(7); // String slice()
document.getElementById("demo").innerHTML += "<p>Slice from index 7 to end: " +
sliced + "</p>";

let subString = str.substring(3, 7); // String substring()
```

```
document.getElementById("demo").innerHTML += "<p>Substring from index 3 to 7: "
+ subString + "</p>";

let subStr = str.substr(7, 5); // String substr()
document.getElementById("demo").innerHTML += "<p>Substr starting at index 7 with
length 5: " + subStr + "</p>";
</script>

</body>
</html>
```

Output:



```
Length: 13

Character at index 4: o

ASCII code of character at index 4: 111

Character at index 6:

Character at index 8 using bracket notation: o

Slice from index 7 to end: World!

Substring from index 3 to 7: lo,

Substr starting at index 7 with length 5: World
```

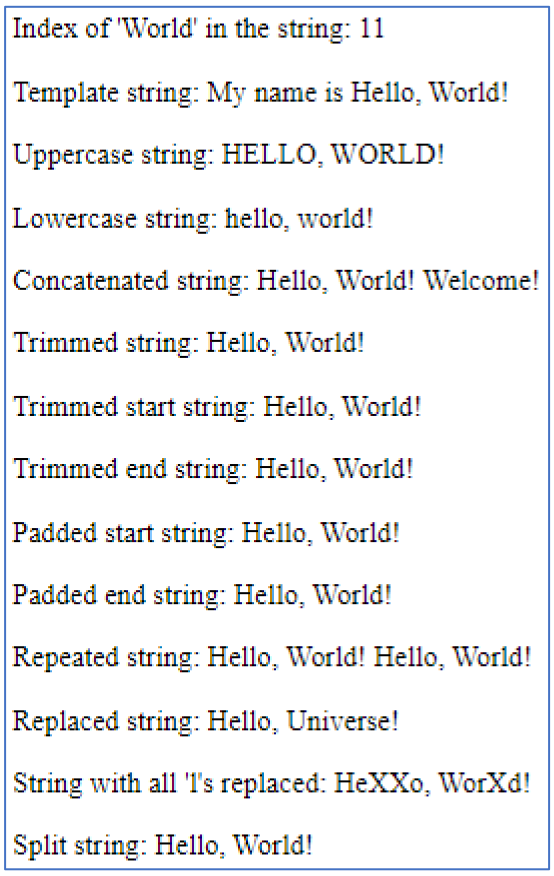
Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>String Methods Demo</title>
</head>
<body>
<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let str = "    Hello, World!    ";
console.log("Original string:", str);

let searchIndex = str.search("World"); // String search()
document.getElementById("demo").innerHTML += "<p>Index of 'World' in the string: "
+ searchIndex + "</p>";
```

```
let templateString = `My name is ${str}`; // String templates
document.getElementById("demo").innerHTML += "<p>Template string: " +
templateString + "</p>";

let upperCaseString = str.toUpperCase(); // String toUpperCase()
document.getElementById("demo").innerHTML += "<p>Uppercase string: " +
upperCaseString + "</p>";

let lowerCaseString = str.toLowerCase(); // String toLowerCase()
document.getElementById("demo").innerHTML += "<p>Lowercase string: " +
lowerCaseString + "</p>";

let concatenatedString = str.concat(" Welcome!"); // String concat()
document.getElementById("demo").innerHTML += "<p>Concatenated string: " +
concatenatedString + "</p>";

let trimmedString = str.trim(); // String trim()
document.getElementById("demo").innerHTML += "<p>Trimmed string: " +
trimmedString + "</p>";

let trimmedStartString = str.trimStart(); // String trimStart()
document.getElementById("demo").innerHTML += "<p>Trimmed start string: " +
trimmedStartString + "</p>";

let trimmedEndString = str.trimEnd(); // String trimEnd()
document.getElementById("demo").innerHTML += "<p>Trimmed end string: " +
trimmedEndString + "</p>";

let paddedStartString = str.padStart(20, "*"); // String padStart()
document.getElementById("demo").innerHTML += "<p>Padded start string: " +
paddedStartString + "</p>";

let paddedEndString = str.padEnd(20, "*"); // String padEnd()
document.getElementById("demo").innerHTML += "<p>Padded end string: " +
paddedEndString + "</p>";

let repeatedString = str.repeat(2); // String repeat()
document.getElementById("demo").innerHTML += "<p>Repeated string: " +
repeatedString + "</p>";

let replacedString = str.replace("World", "Universe"); // String replace()
```

```
document.getElementById("demo").innerHTML += "<p>Replaced string: " + replacedString + "</p>";

let replacedAllString = str.replaceAll("l", "X"); // String replaceAll() - Not supported in some browsers
document.getElementById("demo").innerHTML += "<p>String with all 'l's replaced: " + replacedAllString + "</p>";

let splittedString = str.split(","); // String split()
document.getElementById("demo").innerHTML += "<p>Split string: " + splittedString + "</p>";
</script>

</body>
</html>
```

Output:



```
Index of 'World' in the string: 11

Template string: My name is Hello, World!

Uppercase string: HELLO, WORLD!

Lowercase string: hello, world!

Concatenated string: Hello, World! Welcome!

Trimmed string: Hello, World!

Trimmed start string: Hello, World!

Trimmed end string: Hello, World!

Padded start string: Hello, World!

Padded end string: Hello, World!

Repeated string: Hello, World! Hello, World!

Replaced string: Hello, Universe!

String with all 'l's replaced: HeXXo, WorXd!

Split string: Hello, World!
```

Lesson 3. JavaScript String Search

String Search Methods

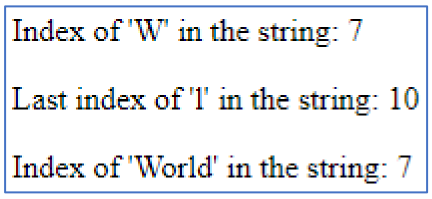
- String indexOf()
- String lastIndexOf()
- String search()
- String match()
- String matchAll()
- String includes()
- String startsWith()
- String endsWith()

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>String Search Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let str = "Hello, World!";
console.log("Original string:", str);

let indexOfW = str.indexOf("W"); // String indexOf()
document.getElementById("demo").innerHTML += "<p>Index of 'W' in the string: " +
indexOfW + "</p>";

let lastIndexOfl = str.lastIndexOf("l"); // String lastIndexOf()
document.getElementById("demo").innerHTML += "<p>Last index of 'l' in the string: "
+ lastIndexOfl + "</p>";

let searchIndex = str.search("World"); // String search()
document.getElementById("demo").innerHTML += "<p>Index of 'World' in the string: "
+ searchIndex + "</p>";
</script>
```

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

Output:



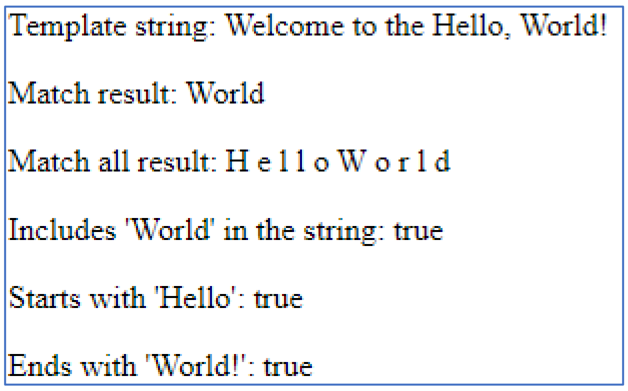
```
Index of 'W' in the string: 7

Last index of 'l' in the string: 10

Index of 'World' in the string: 7
```

- **indexOf()** returns the index of the first occurrence of the specified substring within the string.
- **lastIndexOf()** returns the index of the last occurrence of the specified substring within the string.
- **search()** searches the string for a specified value and returns the position of the first occurrence.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Basic String Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let str = "Hello, World!";
console.log("Original string:", str);

let stringTemplate = `Welcome to the ${str}`; // String templates
document.getElementById("demo").innerHTML += "<p>Template string: " +
stringTemplate + "</p>";

let matchResult = str.match("World"); // String match()
document.getElementById("demo").innerHTML += "<p>Match result: " + matchResult
+ "</p>";

let matchAllResult = str.matchAll(/[a-zA-Z]/g); // String matchAll() - Not supported in
some browsers
let matchAllOutput = "";
for (let match of matchAllResult) {
    matchAllOutput += match[0] + " ";
}
document.getElementById("demo").innerHTML += "<p>Match all result: " +
matchAllOutput + "</p>";

let includesResult = str.includes("World"); // String includes()
document.getElementById("demo").innerHTML += "<p>Includes 'World' in the string:
" + includesResult + "</p>";

let startsWithResult = str.startsWith("Hello"); // String startsWith()
document.getElementById("demo").innerHTML += "<p>Starts with 'Hello': " +
startsWithResult + "</p>";

let endsWithResult = str.endsWith("World!"); // String endsWith()
document.getElementById("demo").innerHTML += "<p>Ends with 'World!': " +
endsWithResult + "</p>";
</script>

</body>
</html>
```

Output:



```
Template string: Welcome to the Hello, World!

Match result: World

Match all result: H e l l o W o r l d

Includes 'World' in the string: true

Starts with 'Hello': true

Ends with 'World!': true
```

Lesson 4. JavaScript Template Strings

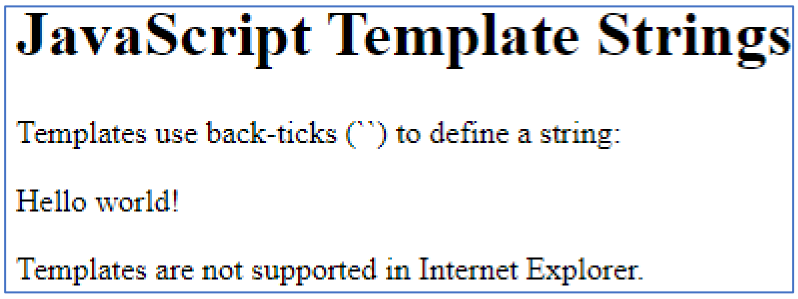
- String Templates
- Template Strings
- Template Literals

A. Back-Tics Syntax

Template Strings use back-ticks (``) rather than the quotes ("") to define a string:

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Template Strings</h1>
        <p>Templates use back-ticks (``) to define a string:</p>
        <p id="demo"></p>
        <p>Templates are not supported in Internet Explorer.</p>
        <script>
        let text = `Hello world!`;
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



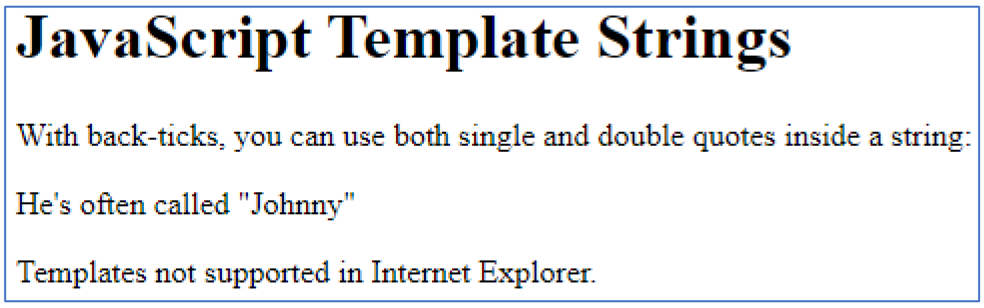
```
JavaScript Template Strings

Templates use back-ticks (``) to define a string:

Hello world!

Templates are not supported in Internet Explorer.
```

B. Quotes Inside Strings

Template Strings allow both single and double quotes inside a string:

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Template Strings</h1>
        <p>With back-ticks, you can use both single and double quotes inside a
        string:</p>
        <p id="demo"></p>
        <p>Templates not supported in Internet Explorer.</p>
        <script>
        let text = `He's often called "Johnny"`;
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



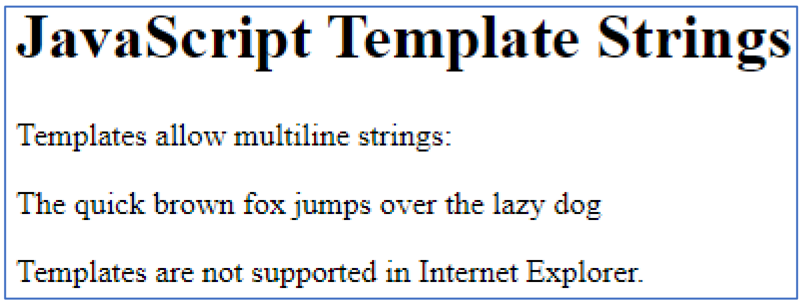
```
JavaScript Template Strings

With back-ticks, you can use both single and double quotes inside a string:

He's often called "Johnny"

Templates not supported in Internet Explorer.
```

C. Multiline Strings

Template Strings allow multiline strings:

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Template Strings</h1>
        <p>Templates allow multiline strings:</p>
        <p id="demo"></p>
        <p>Templates are not supported in Internet Explorer.</p>
        <script>
        let text =

        `The quick
        brown fox
        jumps over
        the lazy dog`;
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



```
JavaScript Template Strings

Templates allow multiline strings:

The quick brown fox jumps over the lazy dog

Templates are not supported in Internet Explorer.
```

D. Interpolation

Template String provide an easy way to interpolate variables and expressions into strings.

The method is called string interpolation.

The syntax is:

```
${...}
```

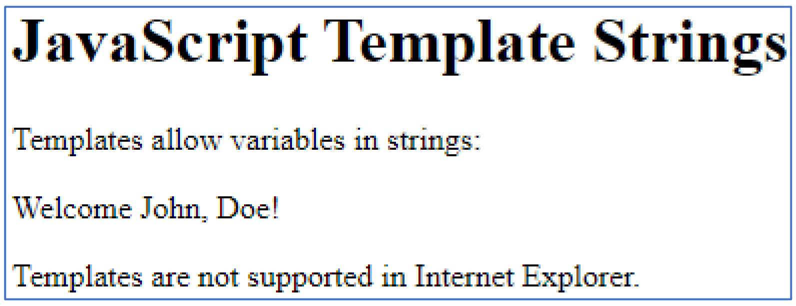
a. Variable Substitutions

Template Strings allow variables in strings:

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Template Strings</h1>
        <p>Templates allow variables in strings:</p>
        <p id="demo"></p>
        <p>Templates are not supported in Internet Explorer.</p>
```

```
          <script>
          let firstName = "John";
          let lastName = "Doe";
          let text = `Welcome ${firstName}, ${lastName}!`;
          document.getElementById("demo").innerHTML = text;
          </script>
</body>
</html>
```

Output:



```
JavaScript Template Strings

Templates allow variables in strings:

Welcome John, Doe!

Templates are not supported in Internet Explorer.
```

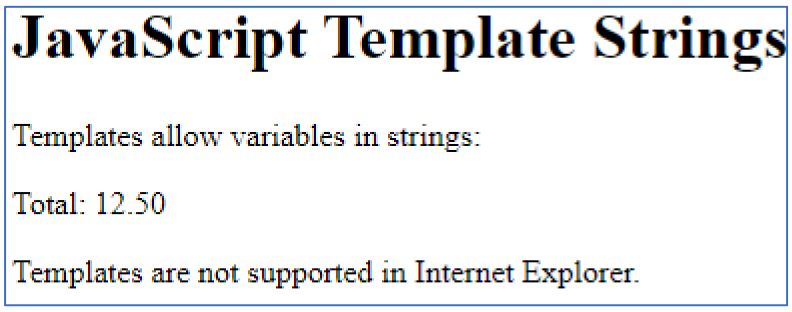
Automatic replacing of variables with real values is called string interpolation.

b. **Expression Substitution**

Template Strings allow expressions in strings:

```
<html>
<body>
          <h1>JavaScript Template Strings</h1>
          <p>Templates allow variables in strings:</p>
          <p id="demo"></p>
          <p>Templates are not supported in Internet Explorer.</p>
          <script>
          let price = 10;
          let VAT = 0.25;
          let total = `Total: ${(price * (1 + VAT)).toFixed(2)}`;
          document.getElementById("demo").innerHTML = total;
          </script>
</body>
</html>
```

Output:



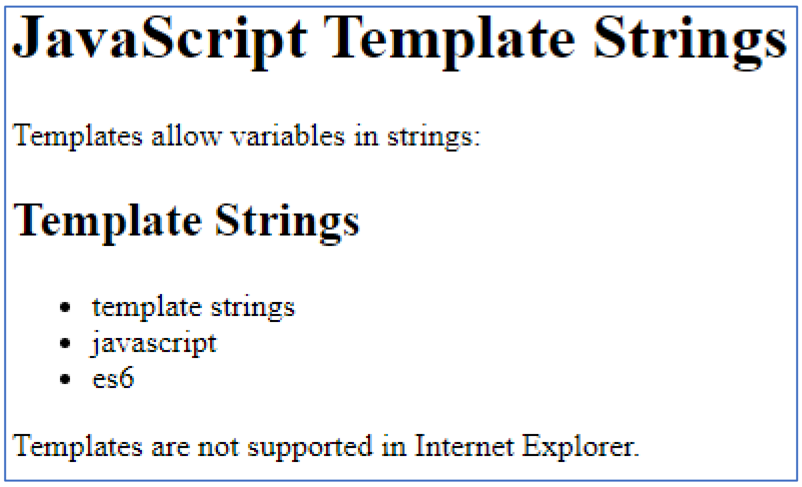
```
JavaScript Template Strings

Templates allow variables in strings:

Total: 12.50

Templates are not supported in Internet Explorer.
```

Automatic replacing of expressions with real values is called string interpolation.

c. HTML Templates

```
<!DOCTYPE html>
<html>
<body>
        <h1>JavaScript Template Strings</h1>
        <p>Templates allow variables in strings:</p>
        <p id="demo"></p>
        <p>Templates are not supported in Internet Explorer.</p>
        <script>
        let header = "Template Strings";
        let tags = ["template strings", "javascript", "es6"];
        let html = `<h2>${header}</h2><ul>`;
        for (const x of tags) {
          html += `<li>${x}</li>`;
        }
        html += `</ul>`;
        document.getElementById("demo").innerHTML = html;
        </script>
</body>
</html>
```

Output:



```
JavaScript Template Strings

Templates allow variables in strings:

Template Strings

  • template strings
  • javascript
  • es6

Templates are not supported in Internet Explorer.
```

Browser Support

- Template Strings is an ES6 feature (JavaScript 2015).
- ES6 is fully supported in all modern browsers since June 2017:

Chrome 51	Edge 15	Firefox 54	Safari 10	Opera 38
May 2016	Apr 2017	Jun 2017	Sep 2016	Jun 2016

Template Strings is not supported in Internet Explorer.

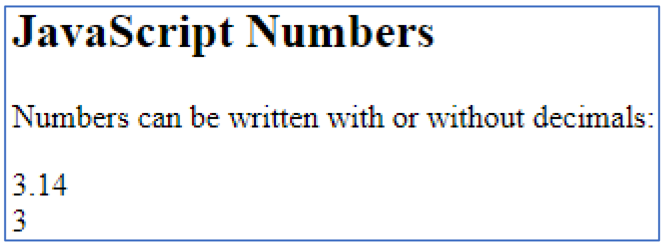
Lesson 5. JavaScript Numbers

JavaScript has only one type of number. Numbers can be written with or without decimals.

Code:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Numbers</h2>
<p>Numbers can be written with or without decimals:</p>
<p id="demo"></p>
<script>
let x = 3.14;
let y = 3;
document.getElementById("demo").innerHTML = x + "<br>" + y;
</script>
</body>
</html>
```

Output:



```
JavaScript Numbers

Numbers can be written with or without decimals:

3.14
3
```

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

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

A. **JavaScript Numbers are Always 64-bit Floating Point**

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.

This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

Value (aka Fraction/Mantissa)	Exponent	Sign
52 bits (0 - 51)	11 bits (52 - 62)	1 bit (63)

B. **Integer Precision**

Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

```
let x = 999999999999999;   // x will be 999999999999999
let y = 9999999999999999;  // y will be 10000000000000000
```

C. Floating Precision

Floating point arithmetic is not always 100% accurate:

```
let x = 0.2 + 0.1; // x will be 0.30000000000000004
```

To solve the problem above, it helps to multiply and divide:

```
let x = (0.2 * 10 + 0.1 * 10) / 10; // x will be 0.3
```

D. Adding Numbers and Strings

WARNING !!

- JavaScript uses the + operator for both addition and concatenation.
- Numbers are added. Strings are concatenated.

```
let x = 10;
let y = 20;
let z = x + y; // z will be 30
```

If you add two strings, the result will be a string concatenation:

```
let x = "10";
let y = "20";
let z = x + y; // z will be 1020
```

If you add a number and a string, the result will be a string concatenation:

```
let x = 10;
let y = "20";
let z = x + y; // z will be 1020
```

If you add a string and a number, the result will be a string concatenation:

```
let x = "10";
let y = 20;
let z = x + y; // z will be 1020
```

A common mistake is to expect this result to be 30:

```
let x = 10;
let y = 20;
let z = "The result is: " + x + y; // z will be 1020
```

A common mistake is to expect this result to be 102030:

```
let x = 10;
let y = 20;
let z = "30";
let result = x + y + z; // z will be 3030
```

- The JavaScript interpreter works from left to right.
- First 10 + 20 is added because x and y are both numbers.
- Then 30 + "30" is concatenated because z is a string.

E. Numeric Strings

JavaScript strings can have numeric content:

```
let x = 100;         // x is a number
let y = "100";       // y is a string
```

JavaScript will try to convert strings to numbers in all numeric operations:

This will work:

```
let x = "100";
let y = "10";
let z = x / y; // z will be 10
let a = x * y; // a will be 1000
let b = x - y; // b will be 90
```

But this will not work:

```
let x = "100";
let y = "10";
let z = x + y; // z will be 10010
```

In the last example JavaScript uses the + operator to concatenate the strings.

F. NaN - Not a Number

- **NaN** is a JavaScript reserved word indicating that a number is not a legal number.
- Trying to do arithmetic with a non-numeric string will result in **NaN** (Not a Number):

```
let x = 100 / "Apple"; // x will be NaN
```

However, if the string is numeric, the result will be a number:

```
let x = 100 / "10"; // x will be 10
```

You can use the global JavaScript function isNaN() to find out if a value is a not a number:

```
let x = 100 / "Apple";
isNaN(x); //output was true
```

Watch out for NaN. If you use NaN in a mathematical operation, the result will also be NaN:

```
let x = NaN;
let y = 5;
let z = x + y; // output was NaN
```

Or the result might be a concatenation like NaN5:

```
let x = NaN;
let y = "5";
let z = x + y; // output was NaN5
```

NaN is a number: typeof NaN returns number:

```
let x = NaN;
document.getElementById("demo").innerHTML = typeof x;
// output was number
```

G. JavaScript Numbers as Objects

Normally JavaScript numbers are primitive values created from literals:

```
let x = 123;
```

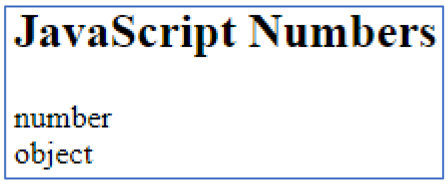
But numbers can also be defined as objects with the keyword **new:**

```
let y = new Number(123);
```

Example:

```
<html>
<body>
<h2>JavaScript Numbers</h2>
<p id="demo"></p>
<script>
// x is a number
let x = 123;
// y is a Number object
let y = new Number(123);
document.getElementById("demo").innerHTML = typeof x + "<br>" + typeof y;
</script>
</body>
</html>
```

Output:



```
JavaScript Numbers

number
object
```

Note:

- Do not create Number objects.
- The new keyword complicates the code and slows down execution speed.
- Number Objects can produce unexpected results:

When using the == operator, x and y are equal:

```
let x = 500;
let y = new Number(500);
//Numbers and Number objects cannot be safely compared:
// the output was true
```

When using the === operator, x and y are not equal.

```
let x = 500;
let y = new Number(500);
//Numbers and Number objects cannot be safely compared:
// the output was false
```

Note: the difference between (x==y) and (x===y).

(x == y) true or false?

```
let x = new Number(500);
let y = new Number(500);
//JavaScript objects cannot be compared:
// the output was false
```

(x === y) true or false?

```
let x = new Number(500);
let y = new Number(500);
//JavaScript objects cannot be compared:
// the output was false
```

Note: Comparing two JavaScript objects always returns false.

Lesson 6. JavaScript BigInt

JavaScript BigInt variables are used to store big integer values that are too big to be represented by a normal JavaScript Number.

A. JavaScript Integer Accuracy

In JavaScript, all numbers are stored in a 64-bit floating-point format (IEEE 754 standard).

With this standard, large integer cannot be exactly represented and will be rounded.

Because of this, JavaScript can only safely represent integers:

Up to 9007199254740991 +(253-1)

and

Down to -9007199254740991 -(253-1).

Integer values outside this range lose precision.

B. How to Create a BigInt

To create a BigInt, append n to the end of an integer or call BigInt():

```
let x = 9999999999999999; // result 10000000000000000
let y = 9999999999999999n; // result 9999999999999999
```

```
let x = 1234567890123456789012345n; // 1234567890123456789012345678890
let y = BigInt(1234567890123456789012345)
// 1234567890123456789012345678890
```

C. BigInt: A new JavaScript Datatype

The JavaScript typeof a BigInt is "bigint":

```
let x = BigInt(999999999999999);
let type = typeof x; // The typeof a BigInt is: bigint
```

BigInt is the second numeric data type in JavaScript (after Number).

With BigInt the total number of supported data types in JavaScript is 8:

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

Lesson 7. JavaScript Number Methods

These number methods can be used on all JavaScript numbers:

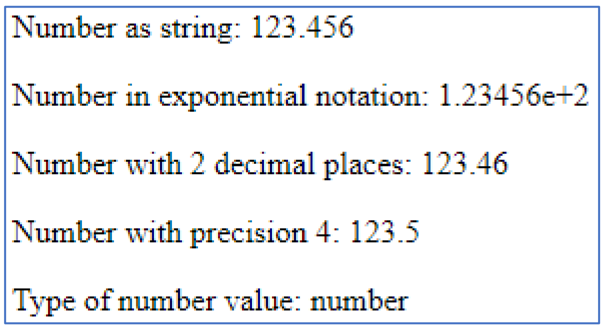
Method	Description
toString()	Returns a number as a string
toExponential()	Returns a number written in exponential notation
toFixed()	Returns a number written with a number of decimals
toPrecision()	Returns a number written with a specified length
ValueOf()	Returns a number as a number

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Number Methods Demo</title>
</head>
<body>
<div id="demo">
  <!-- Results will be displayed here -->
</div>
<script>
let num = 123.456;
console.log("Original number:", num);
let numToString = num.toString(); // toString()
document.getElementById("demo").innerHTML += "<p>Number as string: " +
numToString + "</p>";

let numToExponential = num.toExponential(); // toExponential()
document.getElementById("demo").innerHTML += "<p>Number in exponential
notation: " + numToExponential + "</p>";

let numToFixed = num.toFixed(2); // toFixed()
document.getElementById("demo").innerHTML += "<p>Number with 2 decimal
places: " + numToFixed + "</p>";

let numToPrecision = num.toPrecision(4); // toPrecision()
document.getElementById("demo").innerHTML += "<p>Number with precision 4: " +
numToPrecision + "</p>";
```

```
let numValueOf = typeof num.valueOf(); // valueOf()
document.getElementById("demo").innerHTML += "<p>Type of number value: " +
numValueOf + "</p>";
</script>

</body>
</html>
```

Output:



```
Number as string: 123.456

Number in exponential notation: 1.23456e+2

Number with 2 decimal places: 123.46

Number with precision 4: 123.5

Type of number value: number
```

Lesson 8. JavaScript Number Properties

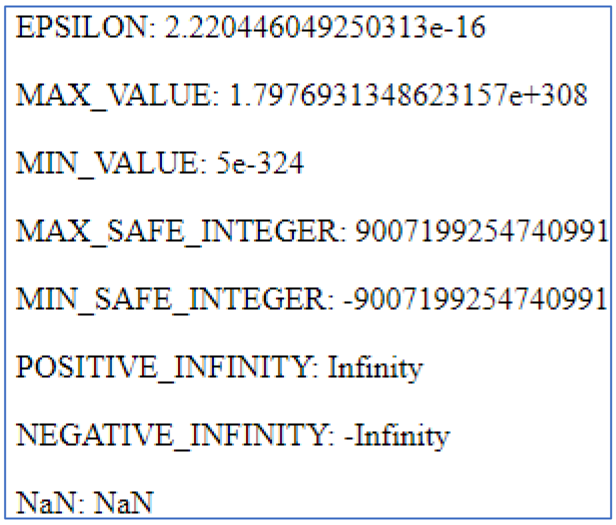
Property	Description
EPSILON	The difference between 1 and the smallest number > 1.
MAX_VALUE	The largest number possible in JavaScript
MIN_VALUE	The smallest number possible in JavaScript
MAX_SAFE_INTEGER	The maximum safe integer (253 - 1)
MIN_SAFE_INTEGER	The minimum safe integer -(253 - 1)
POSITIVE_INFINITY	Infinity (returned on overflow)
NEGATIVE_INFINITY	Negative infinity (returned on overflow)
NaN	A "Not-a-Number" value

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Number Properties Demo</title>
</head>
<body>
<div id="demo">
  <!-- Results will be displayed here -->
</div>
<script>
// EPSILON
let epsilon = Number.EPSILON;
```

```
document.getElementById("demo").innerHTML += "<p>EPSILON: " + epsilon +
"</p>";

// MAX_VALUE
let maxValue = Number.MAX_VALUE;
document.getElementById("demo").innerHTML += "<p>MAX_VALUE: " + maxValue +
"</p>";

// MIN_VALUE
let minValue = Number.MIN_VALUE;
document.getElementById("demo").innerHTML += "<p>MIN_VALUE: " + minValue +
"</p>";

// MAX_SAFE_INTEGER
let maxSafeInteger = Number.MAX_SAFE_INTEGER;
document.getElementById("demo").innerHTML += "<p>MAX_SAFE_INTEGER: " +
maxSafeInteger + "</p>";

// MIN_SAFE_INTEGER
let minSafeInteger = Number.MIN_SAFE_INTEGER;
document.getElementById("demo").innerHTML += "<p>MIN_SAFE_INTEGER: " +
minSafeInteger + "</p>";

// POSITIVE_INFINITY
let positiveInfinity = Number.POSITIVE_INFINITY;
document.getElementById("demo").innerHTML += "<p>POSITIVE_INFINITY: " +
positiveInfinity + "</p>";

// NEGATIVE_INFINITY
let negativeInfinity = Number.NEGATIVE_INFINITY;
document.getElementById("demo").innerHTML += "<p>NEGATIVE_INFINITY: " +
negativeInfinity + "</p>";

// NaN
let nan = Number.NaN;
document.getElementById("demo").innerHTML += "<p>NaN: " + nan + "</p>";
</script>
</body>
</html>
```

Output:

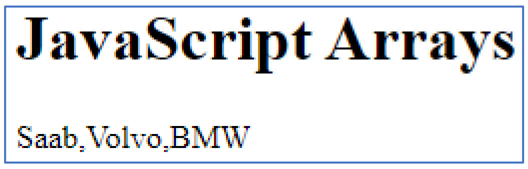

```
EPSILON: 2.220446049250313e-16

MAX_VALUE: 1.7976931348623157e+308

MIN_VALUE: 5e-324

MAX_SAFE_INTEGER: 9007199254740991

MIN_SAFE_INTEGER: -9007199254740991

POSITIVE_INFINITY: Infinity

NEGATIVE_INFINITY: -Infinity

NaN: NaN
```

Lesson 9. JavaScript Arrays

An array is a special variable, which can hold more than one value:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
<p id="demo"></p>
<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
</body>
</html>
```

Output:


```
JavaScript Arrays

Saab,Volvo,BMW
```

A. Why Use Arrays?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

B. Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```

It is a common practice to declare arrays with the const keyword.

Example:

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

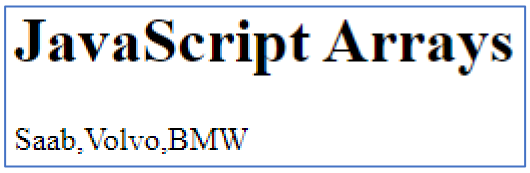
Spaces and line breaks are not important. A declaration can span multiple lines:

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

You can also create an array, and then provide the elements:

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Output:



```
JavaScript Arrays

Saab,Volvo,BMW
```

Note:

- The two examples above do exactly the same.
- There is no need to use new Array().
- For simplicity, readability and execution speed, use the array literal method.

C. Accessing Array Elements

You access an array element by referring to the index number:

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0]; // the result was Saab
```

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

D. Changing an Array Element

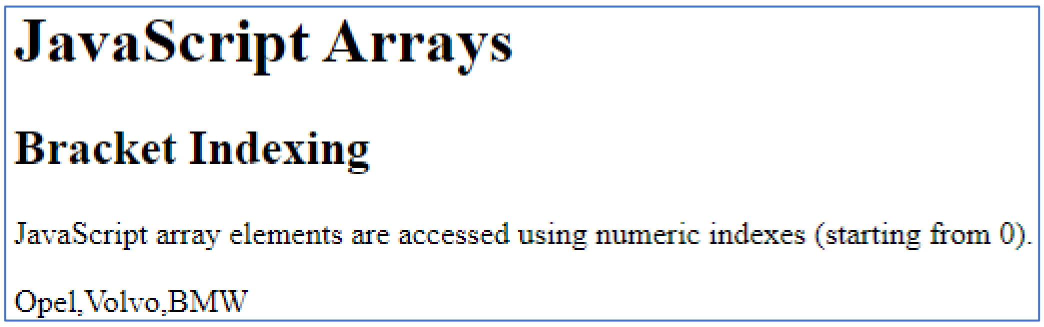
This statement changes the value of the first element in cars:

```
cars[0] = "Opel";
```

Example:

```
<p id="demo"></p>
<script>
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars;
</script>
```

Output:


JavaScript Arrays

Bracket Indexing

JavaScript array elements are accessed using numeric indexes (starting from 0).

Opel,Volvo,BMW

E. Converting an Array to a String

The JavaScript method toString() converts an array to a string of (comma separated) array values.

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result:

```
Banana,Orange,Apple,Mango
```

F. Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

```
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

G. Arrays are Objects

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

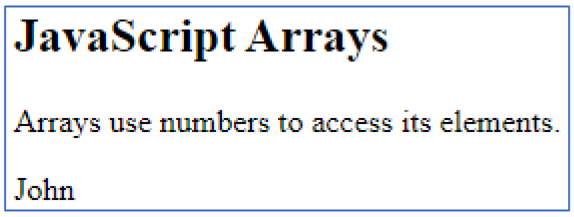
But, JavaScript arrays are best described as arrays.

Arrays use numbers to access its "elements". In this example, person[0] returns John:

Array:

```
<p id="demo"></p>
<script>
const person = ["John", "Doe", 46];
document.getElementById("demo").innerHTML = person[0];
</script>
```

Output:


JavaScript Arrays

Arrays use numbers to access its elements.

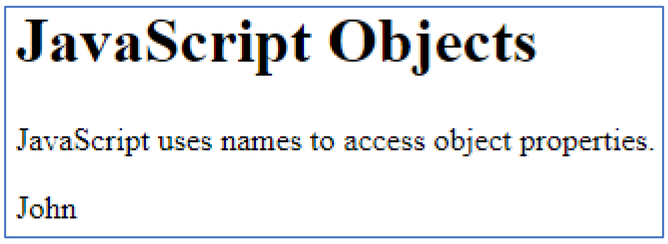
John

Objects use names to access its "members". In this example, person.firstName returns John:

Object:

```
<p id="demo"></p>
<script>
const person = {firstName:"John", lastName:"Doe", age:46};
document.getElementById("demo").innerHTML = person.firstName;
</script>
```

Output:



```
JavaScript Objects

JavaScript uses names to access object properties.

John
```

H. Array Elements Can Be Objects

- JavaScript variables can be objects. Arrays are special kinds of objects.
- Because of this, you can have variables of different types in the same Array.
- You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Lesson 10. JavaScript Arrays Methods

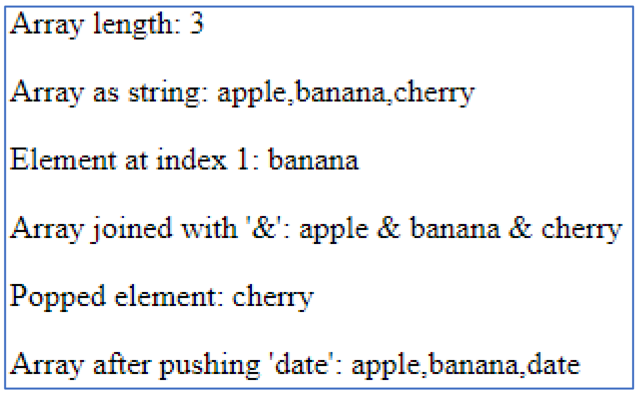
Basic Array Methods

- Array length
- Array toString()
- Array at()
- Array join()
- Array pop()
- Array push()
- Array shift()
- Array unshift()
- Array delete()
- Array concat()
- Array copyWithin()
- Array flat()
- Array splice()
- Array toSpliced()
- Array slice()

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Array Methods Demo</title>
</head>
<body>
<div id="demo">
  <!-- Results will be displayed here -->
</div>
<script>
let arr = ["apple", "banana", "cherry"];
console.log("Original array:", arr);

let arrayLength = arr.length; // Array length
document.getElementById("demo").innerHTML += "<p>Array length: " + arrayLength
+ "</p>";

let arrayToString = arr.toString(); // Array toString()
document.getElementById("demo").innerHTML += "<p>Array as string: " +
arrayToString + "</p>";

let atIndex = arr[1]; // Array at()
document.getElementById("demo").innerHTML += "<p>Element at index 1: " +
atIndex + "</p>";

let arrayJoin = arr.join(" & "); // Array join()
document.getElementById("demo").innerHTML += "<p>Array joined with '&': " +
arrayJoin + "</p>";

let poppedElement = arr.pop(); // Array pop()
document.getElementById("demo").innerHTML += "<p>Popped element: " +
poppedElement + "</p>";

arr.push("date"); // Array push()
document.getElementById("demo").innerHTML += "<p>Array after pushing 'date': " +
arr + "</p>";
</script>
</body>
</html>
```

Output:



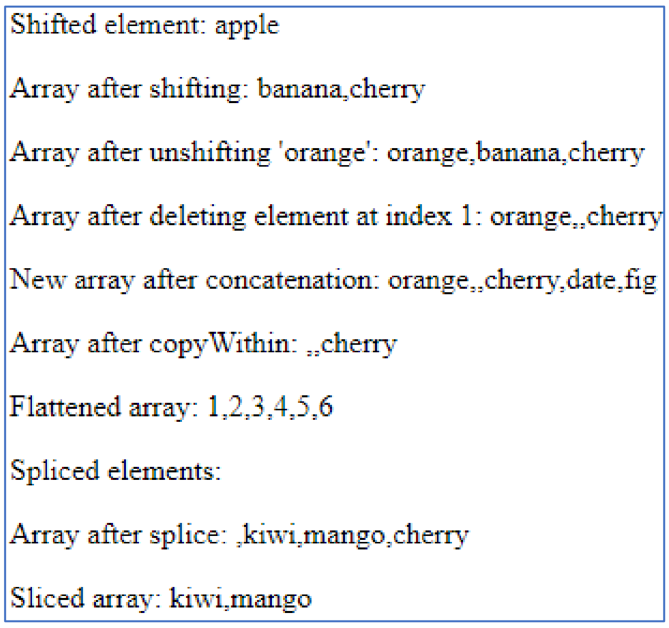
```
Array length: 3

Array as string: apple,banana,cherry

Element at index 1: banana

Array joined with '&': apple & banana & cherry

Popped element: cherry

Array after pushing 'date': apple,banana,date
```

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Array Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let arr = ["apple", "banana", "cherry"];
console.log("Original array:", arr);

// Array shift()
let shiftedElement = arr.shift();
document.getElementById("demo").innerHTML += "<p>Shifted element: " +
shiftedElement + "</p>";
document.getElementById("demo").innerHTML += "<p>Array after shifting: " + arr +
"</p>";

// Array unshift()
arr.unshift("orange");
document.getElementById("demo").innerHTML += "<p>Array after unshifting 'orange':
" + arr + "</p>";

// Array delete()
delete arr[1];
document.getElementById("demo").innerHTML += "<p>Array after deleting element at
index 1: " + arr + "</p>";
```

```
// Array concat()
let newArr = arr.concat(["date", "fig"]);
document.getElementById("demo").innerHTML += "<p>New array after
concatenation: " + newArr + "</p>";
// Array copyWithin()
let copiedArray = arr.copyWithin(0, 1, 2);
document.getElementById("demo").innerHTML += "<p>Array after copyWithin: " +
copiedArray + "</p>";
// Array flat()
let flatArray = [[1, 2], [3, 4, [5, 6]]];
let flattenedArray = flatArray.flat();
document.getElementById("demo").innerHTML += "<p>Flattened array: " +
flattenedArray + "</p>";
// Array splice()
let splicedArray = arr.splice(1, 1, "kiwi", "mango");
document.getElementById("demo").innerHTML += "<p>Spliced elements: " +
splicedArray + "</p>";
document.getElementById("demo").innerHTML += "<p>Array after splice: " + arr +
"</p>";
// Array slice()
let slicedArray = arr.slice(1, 3);
document.getElementById("demo").innerHTML += "<p>Sliced array: " + slicedArray +
"</p>";
</script>
</body>
</html>
```

Output:



```
Shifted element: apple

Array after shifting: banana,cherry

Array after unshifting 'orange': orange,banana,cherry

Array after deleting element at index 1: orange,,cherry

New array after concatenation: orange,,cherry,date,fig

Array after copyWithin: ,,cherry

Flattened array: 1,2,3,4,5,6

Spliced elements:

Array after splice: ,kiwi,mango,cherry

Sliced array: kiwi,mango
```

Lesson 11. JavaScript Array Search

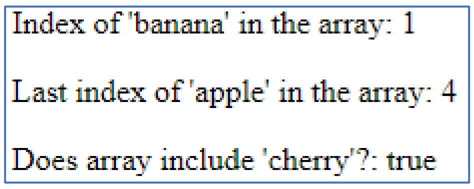
Array Find and Search Methods

- Array indexOf()
- Array lastIndexOf()
- Array includes()
- Array find()
- Array findIndex()
- Array findLast()
- Array findLastIndex()

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Array Find and Search Methods Demo</title>
</head>
<body>
<div id="demo">
  <!-- Results will be displayed here -->
</div>
<script>
let arr = ["apple", "banana", "cherry", "banana", "apple"];
console.log("Original array:", arr);

// Array indexOf()
let indexOfBanana = arr.indexOf("banana");
document.getElementById("demo").innerHTML += "<p>Index of 'banana' in the array:
" + indexOfBanana + "</p>";

// Array lastIndexOf()
let lastIndexOfApple = arr.lastIndexOf("apple");
document.getElementById("demo").innerHTML += "<p>Last index of 'apple' in the
array: " + lastIndexOfApple + "</p>";

// Array includes()
let includesCherry = arr.includes("cherry");
document.getElementById("demo").innerHTML += "<p>Does array include 'cherry'?: "
+ includesCherry + "</p>";
</script>
</body>
</html>
```

Output:

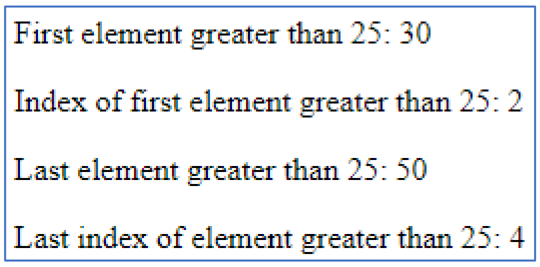

```
Index of 'banana' in the array: 1

Last index of 'apple' in the array: 4

Does array include 'cherry'?: true
```

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Array Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let arr = [10, 20, 30, 40, 50];
console.log("Original array:", arr);

// Array find()
let foundElement = arr.find(element => element > 25);
document.getElementById("demo").innerHTML += "<p>First element greater than 25:
" + foundElement + "</p>";

// Array findIndex()
let foundIndex = arr.findIndex(element => element > 25);
document.getElementById("demo").innerHTML += "<p>Index of first element greater
than 25: " + foundIndex + "</p>";

// Array findLast() - Custom function
Array.prototype.findLast = function(callback) {
    for (let i = this.length - 1; i >= 0; i--) {
        if (callback(this[i])) {
            return this[i];
        }
    }
    return undefined;
};
let foundLastElement = arr.findLast(element => element > 25);
```

```
document.getElementById("demo").innerHTML += "<p>Last element greater than 25:
" + foundLastElement + "</p>";

// Array findLastIndex() - Custom function
Array.prototype.findLastIndex = function(callback) {
    for (let i = this.length - 1; i >= 0; i--) {
        if (callback(this[i])) {
            return i;
        }
    }
    return -1;
};
let foundLastIndex = arr.findLastIndex(element => element > 25);
document.getElementById("demo").innerHTML += "<p>Last index of element greater
than 25: " + foundLastIndex + "</p>";
</script>

</body>
</html>
```

Output:



```
First element greater than 25: 30

Index of first element greater than 25: 2

Last element greater than 25: 50

Last index of element greater than 25: 4
```

Lesson 12. JavaScript Array Sort

Alpabetic Sort

- Array sort()
- Array reverse()
- Array toSorted()
- Array toReversed()
- Sorting Objects

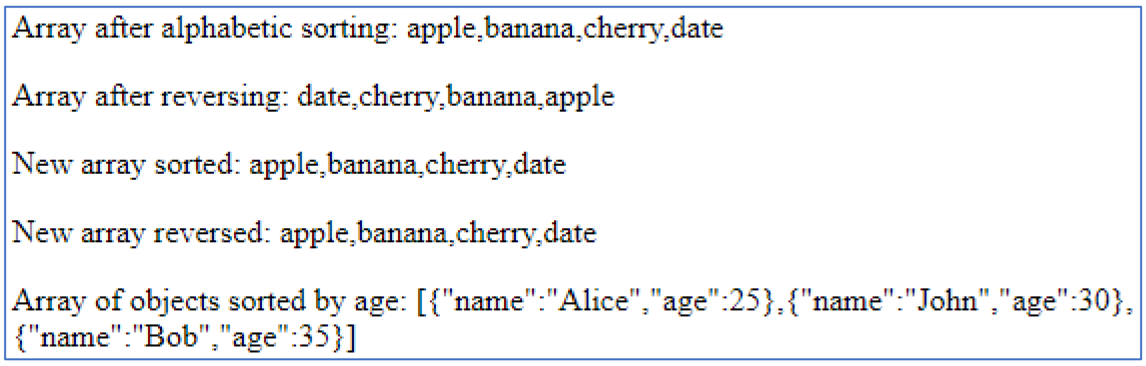
Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Array Sorting Methods Demo</title>
</head>
<body>
```

```
<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let arr = ["banana", "apple", "cherry", "date"];
console.log("Original array:", arr);

// Array sort() - Alphabetic sort
arr.sort();
document.getElementById("demo").innerHTML += "<p>Array after alphabetic sorting: " + arr + "</p>";

// Array reverse() - Reverse the order of elements
arr.reverse();
document.getElementById("demo").innerHTML += "<p>Array after reversing: " + arr + "</p>";

// Array toSorted() - Custom function to sort and return a new array
function toSorted(array) {
   return [...array].sort();
}
let sortedArray = toSorted(arr);
document.getElementById("demo").innerHTML += "<p>New array sorted: " + sortedArray + "</p>";

// Array toReversed() - Custom function to reverse and return a new array
function toReversed(array) {
   return [...array].reverse();
}
let reversedArray = toReversed(arr);
document.getElementById("demo").innerHTML += "<p>New array reversed: " + reversedArray + "</p>";

// Sorting Objects
let objArr = [
   { name: "John", age: 30 },
   { name: "Alice", age: 25 },
   { name: "Bob", age: 35 }
];
console.log("Original object array:", objArr);
```

```
objArr.sort((a, b) => a.age - b.age); // Sort objects based on age
document.getElementById("demo").innerHTML += "<p>Array of objects sorted by age: " + JSON.stringify(objArr) + "</p>";
</script>

</body>
</html>
```

Output:



```
Array after alphabetic sorting: apple,banana,cherry,date

Array after reversing: date,cherry,banana,apple

New array sorted: apple,banana,cherry,date

New array reversed: apple,banana,cherry,date

Array of objects sorted by age: [{"name":"Alice","age":25},{"name":"John","age":30},
{"name":"Bob","age":35}]
```

Numeric Sort

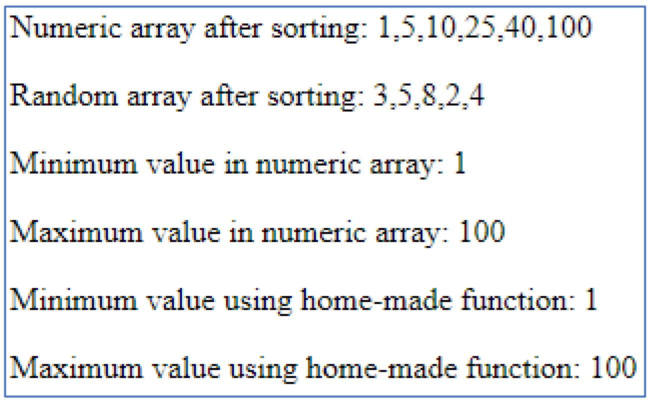
- Numeric Sort
- Random Sort
- Math.min()
- Math.max()
- Home made Min()
- Home made Max()

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Numeric Sort and Math Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let numericArr = [40, 100, 1, 5, 25, 10];
console.log("Original numeric array:", numericArr);

// Numeric Sort
```

```
numericArr.sort((a, b) => a - b);
document.getElementById("demo").innerHTML += "<p>Numeric array after sorting: "
+ numericArr + "</p>";

// Random Sort
let randomArr = [4, 2, 8, 5, 3];
console.log("Original random array:", randomArr);
randomArr.sort(() => Math.random() - 0.5);
document.getElementById("demo").innerHTML += "<p>Random array after sorting: "
+ randomArr + "</p>";

// Math.min() and Math.max()
let minValue = Math.min(...numericArr);
let maxValue = Math.max(...numericArr);
document.getElementById("demo").innerHTML += "<p>Minimum value in numeric
array: " + minValue + "</p>";
document.getElementById("demo").innerHTML += "<p>Maximum value in numeric
array: " + maxValue + "</p>";

// Home-made Min() and Max() functions
function homeMadeMin(array) {
  return Math.min.apply(null, array);
}

function homeMadeMax(array) {
  return Math.max.apply(null, array);
}

let homeMadeMinValue = homeMadeMin(numericArr);
let homeMadeMaxValue = homeMadeMax(numericArr);
document.getElementById("demo").innerHTML += "<p>Minimum value using home-
made function: " + homeMadeMinValue + "</p>";
document.getElementById("demo").innerHTML += "<p>Maximum value using home-
made function: " + homeMadeMaxValue + "</p>";
</script>

</body>
</html>
```

Output:



```
Numeric array after sorting: 1,5,10,25,40,100

Random array after sorting: 3,5,8,2,4

Minimum value in numeric array: 1

Maximum value in numeric array: 100

Minimum value using home-made function: 1

Maximum value using home-made function: 100
```

Lesson 13. JavaScript Array Iteration

Array iteration methods operate on every array item:

- Array forEach
- Array map()
- Array flatMap()
- Array filter()
- Array reduce()
- Array reduceRight()
- Array every()
- Array some()
- Array from()
- Array keys()
- Array entries()
- Array with()
- Array Spread (...)

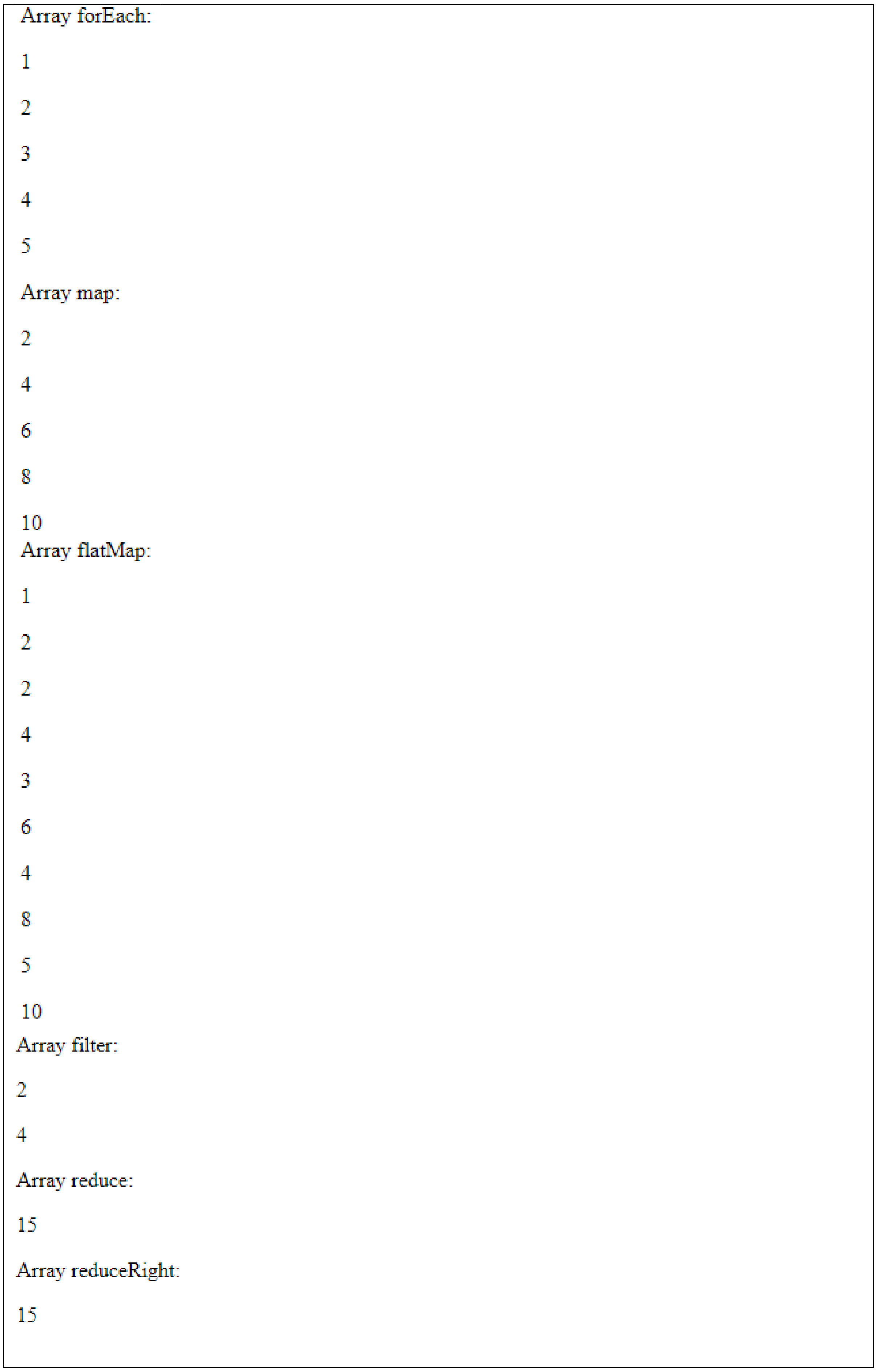
Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Array Iteration Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let arr = [1, 2, 3, 4, 5];
console.log("Original array:", arr);
```

```
// Array forEach
document.getElementById("demo").innerHTML += "<p>Array forEach:</p>";
arr.forEach(item => {
   document.getElementById("demo").innerHTML += "<p>" + item + "</p>";
});

// Array map()
let mappedArray = arr.map(item => item * 2);
document.getElementById("demo").innerHTML += "<p>Array map:</p>";
mappedArray.forEach(item => {
   document.getElementById("demo").innerHTML += "<p>" + item + "</p>";
});

// Array flatMap()
let flatMappedArray = arr.flatMap(item => [item, item * 2]);
document.getElementById("demo").innerHTML += "<p>Array flatMap:</p>";
flatMappedArray.forEach(item => {
   document.getElementById("demo").innerHTML += "<p>" + item + "</p>";
});

// Array filter()
let filteredArray = arr.filter(item => item % 2 === 0);
document.getElementById("demo").innerHTML += "<p>Array filter:</p>";
filteredArray.forEach(item => {
   document.getElementById("demo").innerHTML += "<p>" + item + "</p>";
});

// Array reduce()
let reducedValue = arr.reduce((accumulator, currentValue) => accumulator +
currentValue, 0);
document.getElementById("demo").innerHTML += "<p>Array reduce:</p>";
document.getElementById("demo").innerHTML += "<p>" + reducedValue + "</p>";

// Array reduceRight()
let reduceRightValue = arr.reduceRight((accumulator, currentValue) => accumulator +
currentValue, 0);
document.getElementById("demo").innerHTML += "<p>Array reduceRight:</p>";
document.getElementById("demo").innerHTML += "<p>" + reduceRightValue +
"</p>";
</script>
```

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

Output:



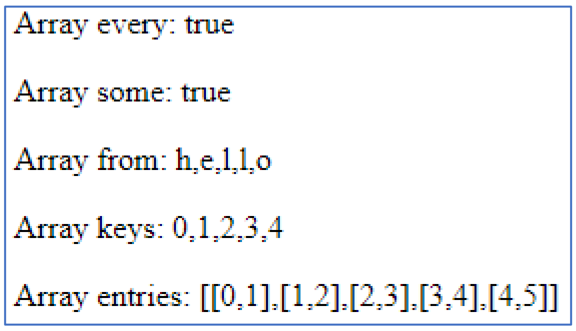
```
Array forEach:

1

2

3

4

5

Array map:

2

4

6

8

10
Array flatMap:

1

2

2

4

3

6

4

8

5

10
Array filter:

2

4

Array reduce:

15

Array reduceRight:

15
```

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Array Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let arr = [1, 2, 3, 4, 5];
console.log("Original array:", arr);

// Array every()
let allGreaterThanZero = arr.every(item => item > 0);
document.getElementById("demo").innerHTML += "<p>Array every: " +
allGreaterThanZero + "</p>";

// Array some()
let hasEvenNumber = arr.some(item => item % 2 === 0);
document.getElementById("demo").innerHTML += "<p>Array some: " +
hasEvenNumber + "</p>";

// Array from()
let fromArray = Array.from("hello");
document.getElementById("demo").innerHTML += "<p>Array from: " + fromArray +
"</p>";

// Array keys()
let keys = Array.from(arr.keys());
document.getElementById("demo").innerHTML += "<p>Array keys: " + keys + "</p>";

// Array entries()
let entries = Array.from(arr.entries());
document.getElementById("demo").innerHTML += "<p>Array entries: " +
JSON.stringify(entries) + "</p>";
```

```
// Array with()
let arrayWith = Array.with(3, "a");
document.getElementById("demo").innerHTML += "<p>Array with: " + arrayWith +
"</p>";

// Array Spread (...)
let spreadArray = [...arr];
document.getElementById("demo").innerHTML += "<p>Array spread: " + spreadArray
+ "</p>";
</script>

</body>
</html>
```

Output:



```
Array every: true

Array some: true

Array from: h,e,l,l,o

Array keys: 0,1,2,3,4

Array entries: [[0,1],[1,2],[2,3],[3,4],[4,5]]
```

Lesson 14. JavaScript Date

- JavaScript Date Objects let us work with dates:
- Mon Mar 11 2024 01:02:19 GMT+0800 (Philippine Standard Time)

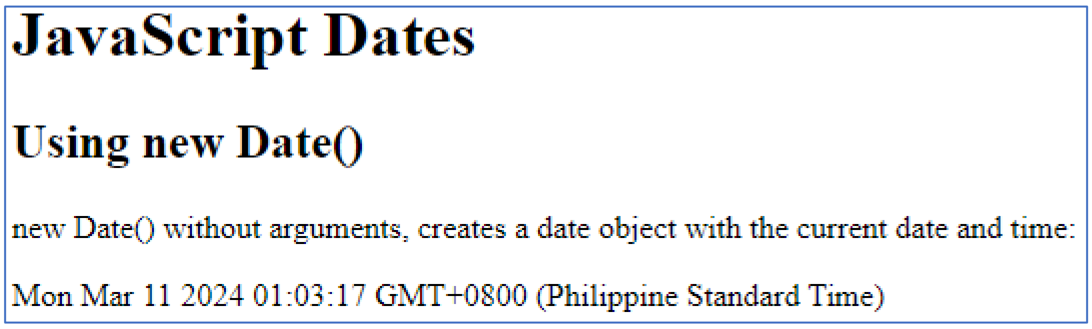
Year: 2024 Month: 3 Day: 11 Hours: 1 Minutes: 2 Seconds: 19

Examples:

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

<h1>JavaScript Dates</h1>
<h2>Using new Date()</h2>
<p>new Date() without arguments, creates a date object with the current date and
time:</p>

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

<script>
const d = new Date();
```

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

</body>
</html>
```

Output:



```
JavaScript Dates

Using new Date()

new Date() without arguments, creates a date object with the current date and time:

Mon Mar 11 2024 01:03:17 GMT+0800 (Philippine Standard Time)
```

Note

- Date objects are static. The "clock" is not "running".
- The computer clock is ticking, date objects are not.

JavaScript Date Output

By default, JavaScript will use the browser's time zone and display a date as a full text string:

```
Mon Mar 11 2024 01:02:19 GMT+0800 (Philippine Standard Time)
```

Creating Date Objects

- Date objects are created with the new Date() constructor.
- There are 9 ways to create a new date object:

```
new Date()
new Date(date string)
new Date(year,month)
new Date(year,month,day)
new Date(year,month,day,hours)
new Date(year,month,day,hours,minutes)
new Date(year,month,day,hours,minutes,seconds)
new Date(year,month,day,hours,minutes,seconds,ms)
new Date(milliseconds)
```

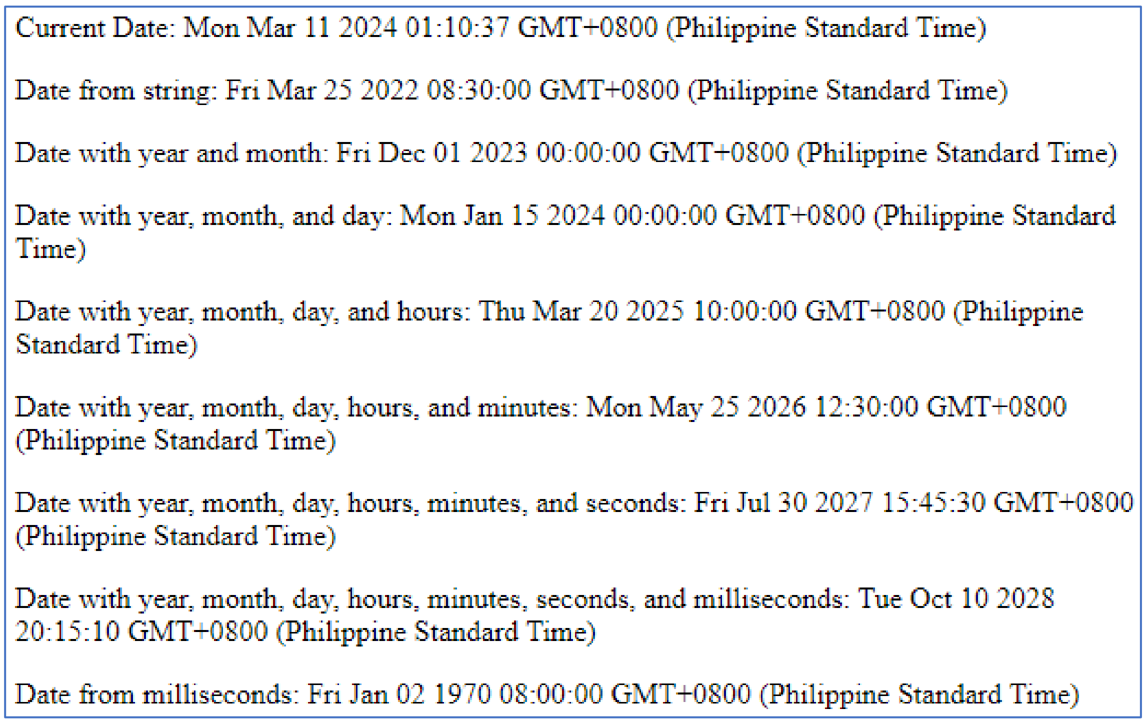
Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Date Creation Demo</title>
</head>
<body>
<div id="demo">
  <!-- Results will be displayed here -->
```

```
</div>
<script>
// Create a new Date object with the current date and time
let currentDate = new Date();
document.getElementById("demo").innerHTML += "<p>Current Date: " + currentDate
+ "</p>";

// Create a new Date object with a specified date string
let dateString = new Date("March 25, 2022 08:30:00");
document.getElementById("demo").innerHTML += "<p>Date from string: " +
dateString + "</p>";

// Create a new Date object with specified year and month
let yearMonthDate = new Date(2023, 11); // Note: Month is zero-based (0-11)
document.getElementById("demo").innerHTML += "<p>Date with year and month: " +
yearMonthDate + "</p>";

// Create a new Date object with specified year, month, and day
let yearMonthDay = new Date(2024, 0, 15); // January 15, 2024
document.getElementById("demo").innerHTML += "<p>Date with year, month, and
day: " + yearMonthDay + "</p>";

// Create a new Date object with specified year, month, day, and hours
let yearMonthDayHours = new Date(2025, 2, 20, 10); // March 20, 2025, 10:00:00
document.getElementById("demo").innerHTML += "<p>Date with year, month, day,
and hours: " + yearMonthDayHours + "</p>";
// Create a new Date object with specified year, month, day, hours, and minutes
let yearMonthDayHoursMinutes = new Date(2026, 4, 25, 12, 30); // May 25, 2026,
12:30:00
document.getElementById("demo").innerHTML += "<p>Date with year, month, day,
hours, and minutes: " + yearMonthDayHoursMinutes + "</p>";

// Create a new Date object with specified year, month, day, hours, minutes, and
seconds
let yearMonthDayHoursMinutesSeconds = new Date(2027, 6, 30, 15, 45, 30); // July 30,
2027, 15:45:30
document.getElementById("demo").innerHTML += "<p>Date with year, month, day,
hours, minutes, and seconds: " + yearMonthDayHoursMinutesSeconds + "</p>";

// Create a new Date object with specified year, month, day, hours, minutes, seconds,
and milliseconds
```

```
let yearMonthDayHoursMinutesSecondsMs = new Date(2028, 9, 10, 20, 15, 10, 500); //
October 10, 2028, 20:15:10.500
document.getElementById("demo").innerHTML += "<p>Date with year, month, day,
hours, minutes, seconds, and milliseconds: " + yearMonthDayHoursMinutesSecondsMs
+ "</p>";

// Create a new Date object with milliseconds since January 1, 1970
let millisecondsDate = new Date(86400000); // 86400000 milliseconds = 1 day
document.getElementById("demo").innerHTML += "<p>Date from milliseconds: " +
millisecondsDate + "</p>";
</script>
</body>
</html>
```

Output:



```
Current Date: Mon Mar 11 2024 01:10:37 GMT+0800 (Philippine Standard Time)

Date from string: Fri Mar 25 2022 08:30:00 GMT+0800 (Philippine Standard Time)

Date with year and month: Fri Dec 01 2023 00:00:00 GMT+0800 (Philippine Standard Time)

Date with year, month, and day: Mon Jan 15 2024 00:00:00 GMT+0800 (Philippine Standard
Time)

Date with year, month, day, and hours: Thu Mar 20 2025 10:00:00 GMT+0800 (Philippine
Standard Time)

Date with year, month, day, hours, and minutes: Mon May 25 2026 12:30:00 GMT+0800
(Philippine Standard Time)

Date with year, month, day, hours, minutes, and seconds: Fri Jul 30 2027 15:45:30 GMT+0800
(Philippine Standard Time)

Date with year, month, day, hours, minutes, seconds, and milliseconds: Tue Oct 10 2028
20:15:10 GMT+0800 (Philippine Standard Time)

Date from milliseconds: Fri Jan 02 1970 08:00:00 GMT+0800 (Philippine Standard Time)
```

Lesson 15. JavaScript Date Formats

JavaScript Date Input

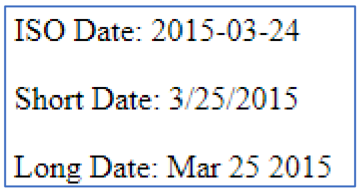
There are generally 3 types of JavaScript date input formats:

Type	Example
ISO Date	"2015-03-25" (The International Standard)
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" or "25 Mar 2015"

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Date Formatting Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let date = new Date("March 25, 2015");
console.log("Original date:", date);

// ISO Date
let isoDate = date.toISOString().split('T')[0];
document.getElementById("demo").innerHTML += "<p>ISO Date: " + isoDate +
"</p>";

// Short Date
let shortDate = (date.getMonth() + 1) + '/' + date.getDate() + '/' + date.getFullYear();
document.getElementById("demo").innerHTML += "<p>Short Date: " + shortDate +
"</p>";

// Long Date
let longMonthNames = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
"Oct", "Nov", "Dec"];
let longDate = longMonthNames[date.getMonth()] + " " + date.getDate() + " " +
date.getFullYear();
document.getElementById("demo").innerHTML += "<p>Long Date: " + longDate +
"</p>";
</script>
</body>
</html>
```

Output:



```
ISO Date: 2015-03-24

Short Date: 3/25/2015

Long Date: Mar 25 2015
```

Lesson 16. JavaScript Date Get Methods

The new Date() Constructor

In JavaScript, date objects are created with new Date().

new Date() returns a date object with the current date and time

Get the Current Time

```
const date = new Date();
```

Date Get Methods

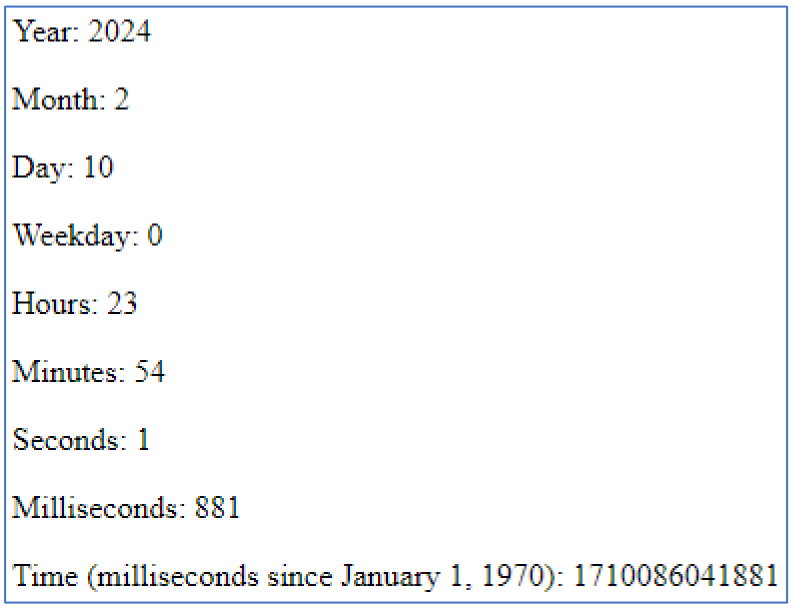
Method	**Description**
getFullYear()	Get **year** as a four digit number (yyyy)
getMonth()	Get **month** as a number (0-11)
getDate()	Get **day** as a number (1-31)
getDay()	Get weekday as a number (0-6)
getHours()	Get **hour** (0-23)
getMinutes()	Get **minute** (0-59)
getSeconds()	Get **second** (0-59)
getMilliseconds()	Get **millisecond** (0-999)

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Date Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>

<script>
let date = new Date();
console.log("Current date:", date);

// Get year as a four digit number (yyyy)
let year = date.getFullYear();
document.getElementById("demo").innerHTML += "<p>Year: " + year + "</p>";

// Get month as a number (0-11)
let month = date.getMonth();
document.getElementById("demo").innerHTML += "<p>Month: " + month + "</p>";
```

```
// Get day as a number (1-31)
let day = date.getDate();
document.getElementById("demo").innerHTML += "<p>Day: " + day + "</p>";

// Get weekday as a number (0-6)
let weekday = date.getDay();
document.getElementById("demo").innerHTML += "<p>Weekday: " + weekday +
"</p>";

// Get hour (0-23)
let hours = date.getHours();
document.getElementById("demo").innerHTML += "<p>Hours: " + hours + "</p>";

// Get minute (0-59)
let minutes = date.getMinutes();
document.getElementById("demo").innerHTML += "<p>Minutes: " + minutes +
"</p>";

// Get second (0-59)
let seconds = date.getSeconds();
document.getElementById("demo").innerHTML += "<p>Seconds: " + seconds +
"</p>";

// Get millisecond (0-999)
let milliseconds = date.getMilliseconds();
document.getElementById("demo").innerHTML += "<p>Milliseconds: " + milliseconds
+ "</p>";

// Get time (milliseconds since January 1, 1970)
let time = date.getTime();
document.getElementById("demo").innerHTML += "<p>Time (milliseconds since
January 1, 1970): " + time + "</p>";
</script>

</body>
</html>
```

Output:



```
Year: 2024

Month: 2

Day: 10

Weekday: 0

Hours: 23

Minutes: 54

Seconds: 1

Milliseconds: 881

Time (milliseconds since January 1, 1970): 1710086041881
```

Lesson 17. JavaScript Set Date Methods

Set Date methods let you set date values (years, months, days, hours, minutes, seconds, milliseconds) for a Date Object.

Set Date Methods

Set Date methods are used for setting a part of a date:

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

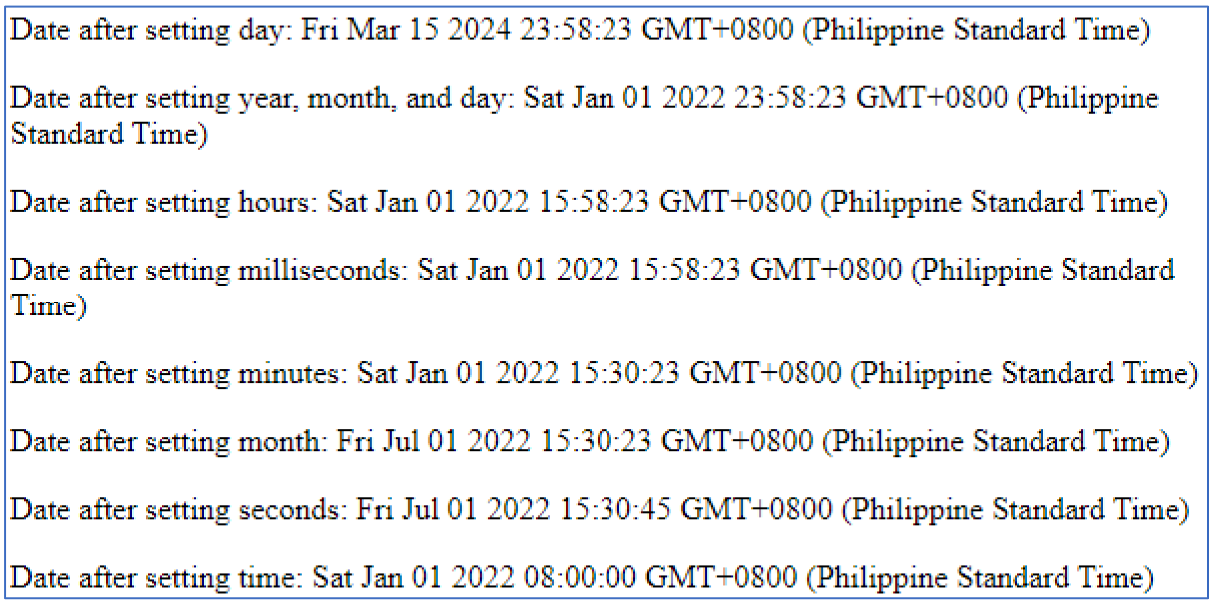
Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Date Setting Methods Demo</title>
</head>
<body>

<div id="demo">
  <!-- Results will be displayed here -->
</div>
```

```
<script>
let date = new Date();
console.log("Original date:", date);

// Set the day as a number (1-31)
date.setDate(15);
document.getElementById("demo").innerHTML += "<p>Date after setting day: " +
date + "</p>";

// Set the year (optionally month and day)
date.setFullYear(2022, 0, 1);
document.getElementById("demo").innerHTML += "<p>Date after setting year,
month, and day: " + date + "</p>";

// Set the hour (0-23)
date.setHours(15);
document.getElementById("demo").innerHTML += "<p>Date after setting hours: " +
date + "</p>";

// Set the milliseconds (0-999)
date.setMilliseconds(500);
document.getElementById("demo").innerHTML += "<p>Date after setting
milliseconds: " + date + "</p>";

// Set the minutes (0-59)
date.setMinutes(30);
document.getElementById("demo").innerHTML += "<p>Date after setting minutes: "
+ date + "</p>";

// Set the month (0-11)
date.setMonth(6);
document.getElementById("demo").innerHTML += "<p>Date after setting month: " +
date + "</p>";

// Set the seconds (0-59)
date.setSeconds(45);
document.getElementById("demo").innerHTML += "<p>Date after setting seconds: "
+ date + "</p>";

// Set the time (milliseconds since January 1, 1970)
date.setTime(1640995200000);
```

```
document.getElementById("demo").innerHTML += "<p>Date after setting time: " +
date + "</p>";
</script>
</body>
</html>
```

Output:



```
Date after setting day: Fri Mar 15 2024 23:58:23 GMT+0800 (Philippine Standard Time)

Date after setting year, month, and day: Sat Jan 01 2022 23:58:23 GMT+0800 (Philippine
Standard Time)

Date after setting hours: Sat Jan 01 2022 15:58:23 GMT+0800 (Philippine Standard Time)

Date after setting milliseconds: Sat Jan 01 2022 15:58:23 GMT+0800 (Philippine Standard
Time)

Date after setting minutes: Sat Jan 01 2022 15:30:23 GMT+0800 (Philippine Standard Time)

Date after setting month: Fri Jul 01 2022 15:30:23 GMT+0800 (Philippine Standard Time)

Date after setting seconds: Fri Jul 01 2022 15:30:45 GMT+0800 (Philippine Standard Time)

Date after setting time: Sat Jan 01 2022 08:00:00 GMT+0800 (Philippine Standard Time)
```

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_module4_lab.js).

Lab Tasks:

1. **String Manipulation:**
   Task 1: Create a JavaScript function that takes a string input and returns the reverse of the string.
   Task 2: Write a script that replaces all occurrences of a specific word in a given string with another word.
2. **Numeric Operations:**
   Task 1: Develop a JavaScript function that calculates the factorial of a given positive integer.
   Task 2: Implement a script to find the sum of all prime numbers within a specified range.
3. **Array Handling:**
   Task 1: Write a function to merge two arrays into a single sorted array.
   Task 2: Create a script that finds the largest and smallest elements in an array.

4. **Date Management:**
   Task 1: Develop a JavaScript function to calculate the difference (in days) between two given dates.
   Task 2: Write a script to format a date object into different date formats (e.g., ISO, Short, Long).

Summary

- In the exploration of JavaScript fundamentals encompassing strings, numbers, arrays, and date formats, learners embarked on a comprehensive journey delving into the intricacies of these essential data types and their manipulation techniques. From understanding the structure and behavior of JavaScript strings to mastering advanced topics such as template strings and string searching methods, students navigated through a curated curriculum designed to equip them with practical skills for string manipulation. Similarly, the exploration of JavaScript numbers and arrays delved into arithmetic operations, handling large integers with BigInt, and employing various array methods for sorting, searching, and iteration. Additionally, learners delved into the intricacies of working with JavaScript dates, understanding date formats, and leveraging date methods for effective date manipulation. Through a structured approach encompassing lessons, exercises, and hands-on labs, students gained proficiency in utilizing these fundamental JavaScript concepts, empowering them to confidently tackle real-world programming challenges in web development projects.

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/

MODULE 5
JAVASCRIPT CONDITIONS

Introduction

Conditional statements are used to perform different actions based on different conditions. Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this. In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

Learning Outcomes

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

1. Understand how if-else statements are used for conditional execution.
2. Explore how switch statements evaluate expressions to match specific cases.
3. Apply if-else and switch statements in practical JavaScript scenarios.

Lesson 1. JS If Else

A. The If Statement

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax:

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

Note that if is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

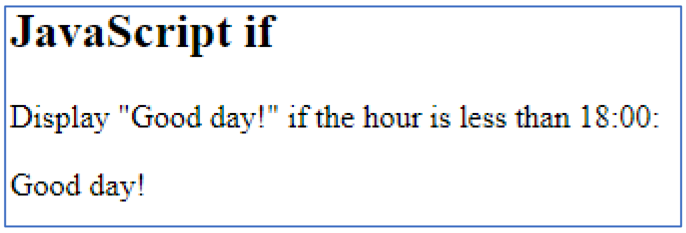
Example:

Make a "Good day" greeting if the hour is less than 18:00:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript if</h2>
<p>Display "Good day!" if the hour is less than 18:00:</p>
<p id="demo">Good Evening!</p>

<script>
if (new Date().getHours() < 18) {
  document.getElementById("demo").innerHTML = "Good day!";
}
</script>
</body>
</html>
```

The result of greeting will be:

Output:



```
JavaScript if

Display "Good day!" if the hour is less than 18:00:

Good day!
```

Use the else statement to specify a block of code to be executed if the condition is false:

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

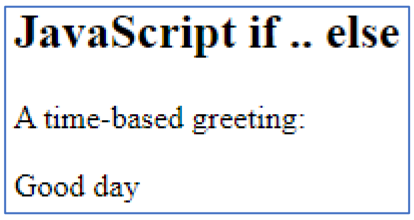
Example:

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

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

<h2>JavaScript if .. else</h2>

<p>A time-based greeting:</p>

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

<script>
```

```
const hour = new Date().getHours();
let greeting;

if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}

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

</body>
</html>
```

The result of greeting will be:



JavaScript if .. else

A time-based greeting:

Good day

Use the else if statement to specify a new condition if the first condition is false.

Syntax:

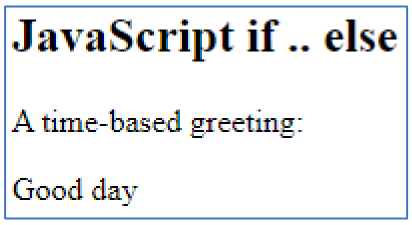
```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

Example:

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript if .. else</h2>
<p>A time-based greeting:</p>
<p id="demo"></p>

<script>
const time = new Date().getHours();
let greeting;
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
document.getElementById("demo").innerHTML = greeting;
</script>

</body>
</html>
```

Output:



JavaScript if .. else

A time-based greeting:

Good day

Lesson 2. JS Switch

The switch statement is used to perform different actions based on different conditions.

A. The JavaScript Switch Statement

Use the switch statement to select one of many code blocks to be executed.

Syntax:

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Example:

The getDay() method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

This example uses the weekday number to calculate the weekday name:

B. The break Keyword

When JavaScript reaches a break keyword, it breaks out of the switch block.

This will stop the execution inside the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

Note: If you omit the break statement, the next case will be executed even if the evaluation does not match the case.

C. The default Keyword

The default keyword specifies the code to run if there is no case match:

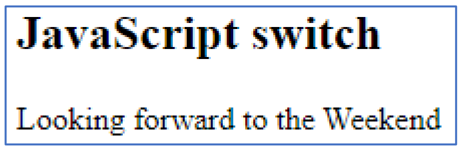
Example:

The getDay() method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript switch</h2>
<p id="demo"></p>
      <script>
      let text;
      switch (new Date().getDay()) {
        case 6:
          text = "Today is Saturday";
          break;
        case 0:
          text = "Today is Sunday";
          break;
        default:
          text = "Looking forward to the Weekend";
      }
      document.getElementById("demo").innerHTML = text;
      </script>
</body>
</html>
```

Output:

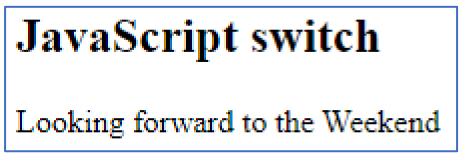


```
JavaScript switch

Looking forward to the Weekend
```

The default case does not have to be the last case in a switch block:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript switch</h2>
<p id="demo"></p>
        <script>
        let text;
        switch (new Date().getDay()) {
          default:
            text = "Looking forward to the Weekend";
            break;
          case 6:
            text = "Today is Saturday";
            break;
          case 0:
            text = "Today is Sunday";
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



```
JavaScript switch

Looking forward to the Weekend
```

Note: If default is not the last case in the switch block, remember to end the default case with a break.

D. Common Code Blocks

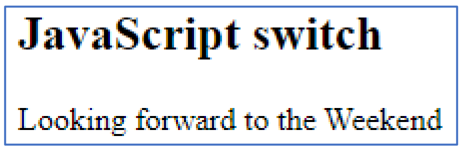
Sometimes you will want different switch cases to use the same code.

In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript switch</h2>
<p id="demo"></p>
        <script>
        let text;
        switch (new Date().getDay()) {
          case 4:
          case 5:
```

```
        text = "Soon it is Weekend";
        break;
      case 0:
      case 6:
        text = "It is Weekend";
        break;
      default:
        text = "Looking forward to the Weekend";
    }
    document.getElementById("demo").innerHTML = text;
    </script>
</body>
</html>
```

Output:



```
JavaScript switch

Looking forward to the Weekend
```

E. Switching Details

If multiple cases matches a case value, the first case is selected.

If no matching cases are found, the program continues to the default label.

If no default label is found, the program continues to the statement(s) after the switch.

F. Strict Comparison

Switch cases use strict comparison (===).

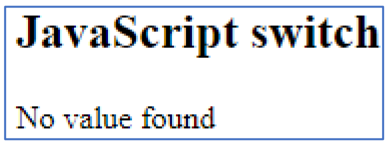
The values must be of the same type to match.

A strict comparison can only be true if the operands are of the same type.

In this example there will be no match for x:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript switch</h2>
<p id="demo"></p>
    <script>
    let x = "0";

    switch (x) {
      case 0:
        text = "Off";
        break;
      case 1:
        text = "On";
        break;
      default:
        text = "No value found";
    }
    document.getElementById("demo").innerHTML = text;
    </script>
</body>
</html>
```

Output:



```
JavaScript switch

No value found
```

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_module5_lab.js).
- Create a new JavaScript file (e.g., script.js) linked to the HTML file.

Lab Tasks:

Task 1: Basic Conditional Statements:

1. Task students with writing a JavaScript program that prompts the user to enter their age and then determines and displays whether they are eligible to vote (age 18 or older) using if-else statements.

Task 2: Multiple Conditions with if-else:

1. Create a JavaScript program that will determine the grade of a student based on their exam score.
2. Create a JavaScript program that prompts the user to enter their exam score and then assigns and displays the corresponding grade (A, B, C, D, or F) using nested if-else statements.

Task 3: Utilizing the Switch Statement:

1. Task students with refactoring their previous program (from Task 3) to use a switch statement instead of nested if-else statements.

Summary

- This module delves into JavaScript conditions, primarily focusing on if-else statements and the switch statement.
- **JS If Else:** This section explores the fundamental if-else statement in JavaScript. It covers how to structure conditional statements using if-else, allowing for the execution of different code blocks based on the evaluation of a specified condition. Examples and syntax explanations are provided to illustrate its usage and versatility in programming.
- **JS Switch:** The module also introduces the switch statement in JavaScript, offering an alternative approach to handling multiple conditions. It elaborates on the syntax and functionality of the switch statement, demonstrating how it can streamline code organization and improve readability in scenarios involving numerous conditional branches.
- **Values Integration:** Throughout the discussion on JavaScript conditions, the text emphasizes the importance of attentiveness and intelligence in understanding and effectively implementing conditional logic. Attentiveness aids in grasping the nuances of conditional statements, while intelligence enables programmers to make informed decisions regarding code structure and flow based on varying conditions.

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

MODULE 6
JAVASCRIPT LOOPS, BREAK CONTINUE AND ITERABLES

Introduction

Loops can execute a block of code a number of times.
JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- for/of - loops through the values of an iterable object
- while - loops through a block of code while a specified condition is true
- do/while - also loops through a block of code while a specified condition is true

Learning Outcomes

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

1. Understand the syntax and structure of JavaScript for loops.
2. Understand the differences between for-in and for-of loops.
3. Understand scenarios where break statements are beneficial.

Lesson 1. For Loop

Loops are handy, if you want to run the same code over and over again, each time with a different value. Often this is the case when working with arrays:

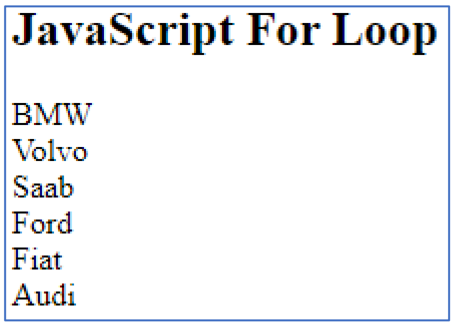
Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For Loop</h2>
<p id="demo"></p>
<script>
const cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];
let text = "";
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Output:



```
JavaScript For Loop

BMW
Volvo
Saab
Ford
Fiat
Audi
```

A. The For Loop

The for statement creates a loop with 3 optional expressions:

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

Expression 1 is executed (one time) before the execution of the code block.

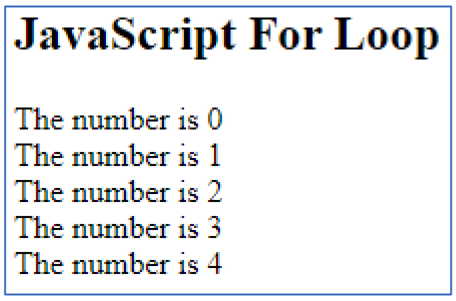
Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For Loop</h2>
<p id="demo"></p>
        <script>
        let text = "";
        for (let i = 0; i < 5; i++) {
          text += "The number is " + i + "<br>";
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



```
JavaScript For Loop

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
```

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

Expression 1

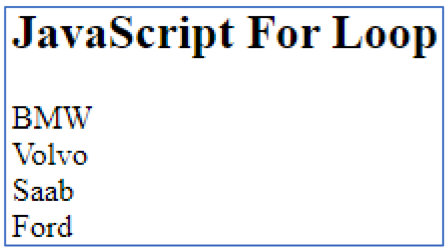
Normally you will use expression 1 to initialize the variable used in the loop (let i = 0).

This is not always the case. JavaScript doesn't care. Expression 1 is optional.

You can initiate many values in expression 1 (separated by comma):

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For Loop</h2>
<p id="demo"></p>
        <script>
        const cars = ["BMW", "Volvo", "Saab", "Ford"];
        let i, len, text;
        for (i = 0, len = cars.length, text = ""; i < len; i++) {
          text += cars[i] + "<br>";
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:

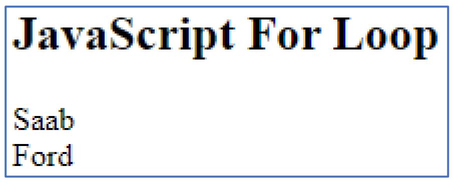


```
JavaScript For Loop

BMW
Volvo
Saab
Ford
```

And you can omit expression 1 (like when your values are set before the loop starts):

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For Loop</h2>
<p id="demo"></p>
        <script>
        const cars = ["BMW", "Volvo", "Saab", "Ford"];
        let i = 2;
        let len = cars.length;
        let text = "";

        for (; i < len; i++) {
          text += cars[i] + "<br>";
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



```
JavaScript For Loop

Saab
Ford
```

Expression 2

Often expression 2 is used to evaluate the condition of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 2 is also optional.

If expression 2 returns true, the loop will start over again. If it returns false, the loop will end.

Note: If you omit expression 2, you must provide a break inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Often expression 3 increments the value of the initial variable.

Expression 3

This is not always the case. JavaScript doesn't care. Expression 3 is optional.

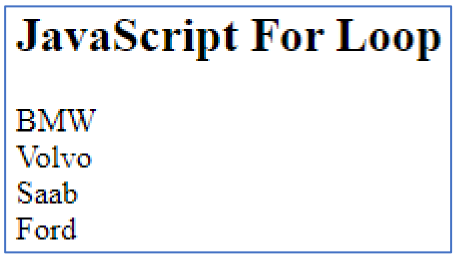
Expression 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Expression 3 can also be omitted (like when you increment your values inside the loop):

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

```
<h2>JavaScript For Loop</h2>
<p id="demo"></p>
        <script>
        const cars = ["BMW", "Volvo", "Saab", "Ford"];
        let i = 0;
        let len = cars.length;
        let text = "";

        for (; i < len; ) {
          text += cars[i] + "<br>";
          i++;
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



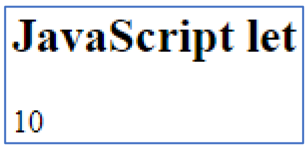
```
JavaScript For Loop

BMW
Volvo
Saab
Ford
```

B. Loop Scope

Using var in a loop:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript let</h2>
<p id="demo"></p>
<script>
        var i = 5;
        for (var i = 0; i < 10; i++) {
          // some statements
        }
        document.getElementById("demo").innerHTML = i;
</script>
</body>
</html>
```

Output:

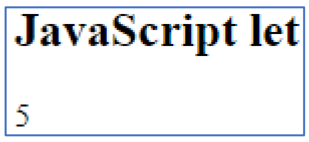


```
JavaScript let

10
```

Using let in a loop:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript let</h2>
<p id="demo"></p>
        <script>
        let i = 5;
        for (let i = 0; i < 10; i++) {
          // some statements
        }
        document.getElementById("demo").innerHTML = i;
        </script>
</body>
</html>
```

Output



```
JavaScript let

5
```

In the first example, using var, the variable declared in the loop redeclares the variable outside the loop.

In the second example, using let, the variable declared in the loop does not redeclare the variable outside the loop.

When let is used to declare the i variable in a loop, the i variable will only be visible within the loop.

Lesson 2. JavaScript For In

1. **The For In Loop**

The JavaScript for in statement loops through the properties of an Object:

Syntax:

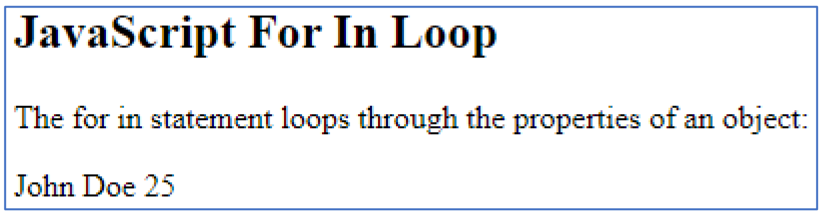
```
for (key in object) {
  // code block to be executed
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For In Loop</h2>
<p>The for in statement loops through the properties of an object:</p>
<p id="demo"></p>
        <script>
        const person = {fname:"John", lname:"Doe", age:25};
```

```
          let txt = "";
          for (let x in person) {
            txt += person[x] + " ";
          }
          document.getElementById("demo").innerHTML = txt;
          </script>
</body>
</html>
```

Output:



JavaScript For In Loop

The for in statement loops through the properties of an object:

John Doe 25

Example Explained:

- The for in loop iterates over a person object
- Each iteration returns a key (x)
- The key is used to access the value of the key
- The value of the key is person[x]

2. **For In Over Arrays**

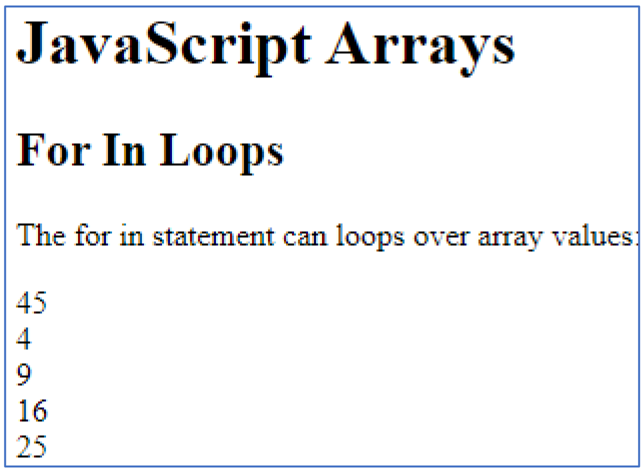
The JavaScript for in statement can also loop over the properties of an Array:

Syntax:

```
for (variable in array) {
  code
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>For In Loops</h2>
<p>The for in statement can loops over array values:</p>
<p id="demo"></p>
          <script>
          const numbers = [45, 4, 9, 16, 25];
          let txt = "";
          for (let x in numbers) {
            txt += numbers[x] + "<br>";
          }
          document.getElementById("demo").innerHTML = txt;
          </script>
</body>
</html>
```



```
JavaScript Arrays

For In Loops

The for in statement can loops over array values:

45
4
9
16
25
```

Do not use for in over an Array if the index order is important.

The index order is implementation-dependent, and array values may not be accessed in the order you expect.

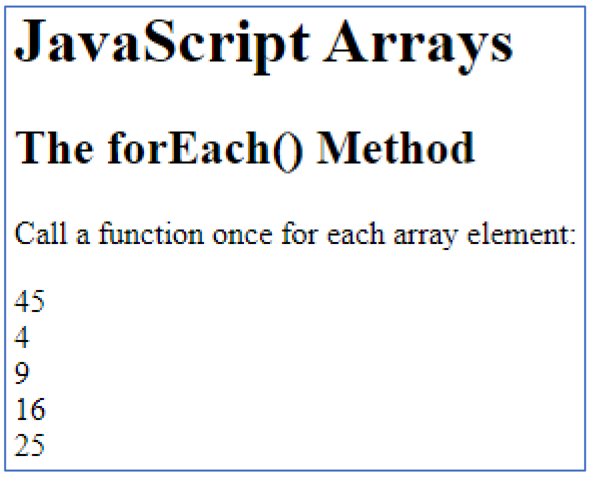
It is better to use a for loop, a for of loop, or Array.forEach() when the order is important.

3. **Array.forEach**()

   The forEach() method calls a function (a callback function) once for each array element.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>The forEach() Method</h2>
<p>Call a function once for each array element:</p>
<p id="demo"></p>
<script>
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = txt;

function myFunction(value, index, array) {
  txt += value + "<br>";
}
</script>
</body>
</html>
```

Output:



```
JavaScript Arrays

The forEach() Method

Call a function once for each array element:

45
4
9
16
25
```

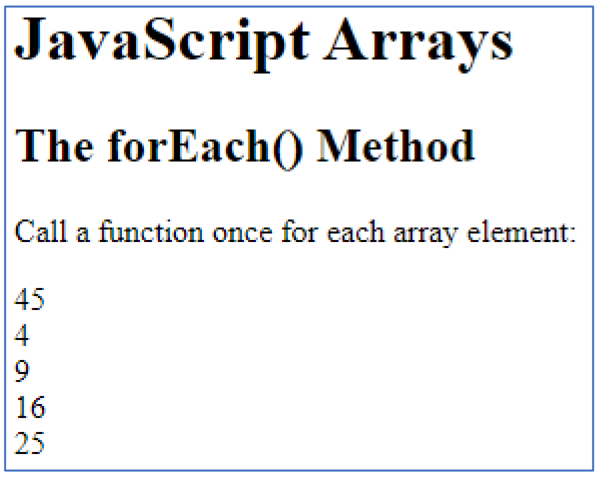
Note that the function takes 3 arguments:

- The item value
- The item index
- The array itself

The example above uses only the value parameter. It can be rewritten to:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
<h2>The forEach() Method</h2>
<p>Call a function once for each array element:</p>
<p id="demo"></p>
<script>
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = txt;

function myFunction(value) {
  txt += value + "<br>";
}
</script>
</body>
</html>
```

Output:



```
JavaScript Arrays

The forEach() Method

Call a function once for each array element:

45
4
9
16
25
```

Lesson 3. JavaScript Loop Of

The JavaScript for of statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

Syntax:

```
for (variable of iterable) {
  // code block to be executed
}
```

variable - For every iteration the value of the next property is assigned to the variable. Variable can be declared with const, let, or var.

iterable - An object that has iterable properties.

Browser Support:

For/of was added to JavaScript in 2015 (ES6)

Safari 7 was the first browser to support for of:

Chrome 38	Edge 12	Firefox 51	Safari 7	Opera 25
Oct 2014	Jul 2015	Oct 2016	Oct 2013	Oct 2014

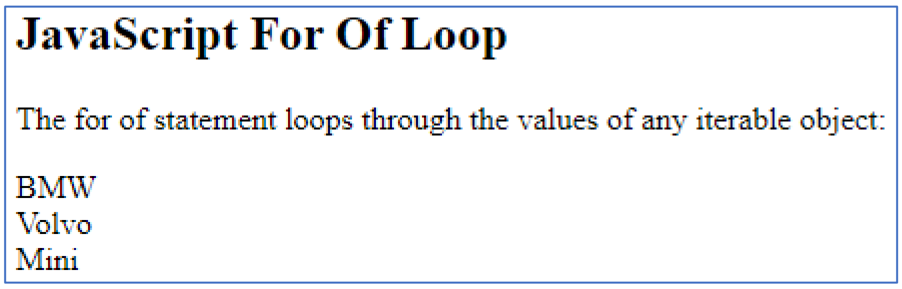
For/of is not supported in Internet Explorer.

1. **Looping over an Array**

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For Of Loop</h2>
<p>The for of statement loops through the values of any iterable object:</p>
<p id="demo"></p>
```

```
<script>
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x + "<br>";
}

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

</body>
</html>
```

Output:



JavaScript For Of Loop

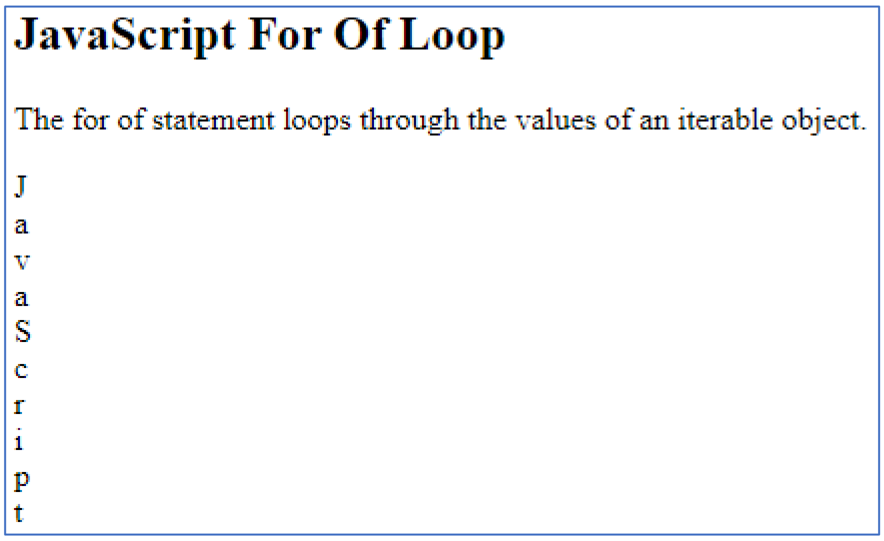
The for of statement loops through the values of any iterable object:

```
BMW
Volvo
Mini
```

2. **Looping over a String**

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript For Of Loop</h2>
<p>The for of statement loops through the values of an iterable object.</p>
<p id="demo"></p>

<script>
let language = "JavaScript";

let text = "";
for (let x of language) {
  text += x + "<br>";
}

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

</body>
</html>
```

Output:



```
JavaScript For Of Loop

The for of statement loops through the values of an iterable object.

J
a
v
a
S
c
r
i
p
t
```

Lesson 4. JavaScript Loop While

Loops can execute a block of code as long as a specified condition is true.

1. **The While Loop**

   The while loop loops through a block of code as long as a specified condition is true.

   Syntax:

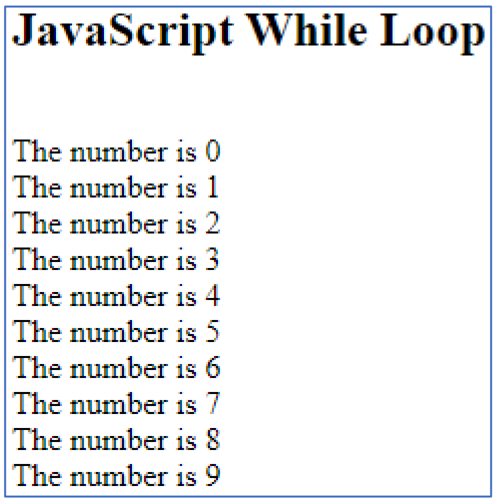
   ```
   while (condition) {
     // code block to be executed
   }
   ```

   Example:

   In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

   ```
   <!DOCTYPE html>
   <html>
   <body>
   <h2>JavaScript While Loop</h2>
   <p id="demo"></p>
   <script>
   let text = "";
   let i = 0;
   while (i < 10) {
     text += "<br>The number is " + i;
     i++;
   }
   document.getElementById("demo").innerHTML = text;
   </script>
   </body>
   </html>
   ```

Output:



```
JavaScript While Loop


The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

Note: If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

2. **The Do While Loop**

The do while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

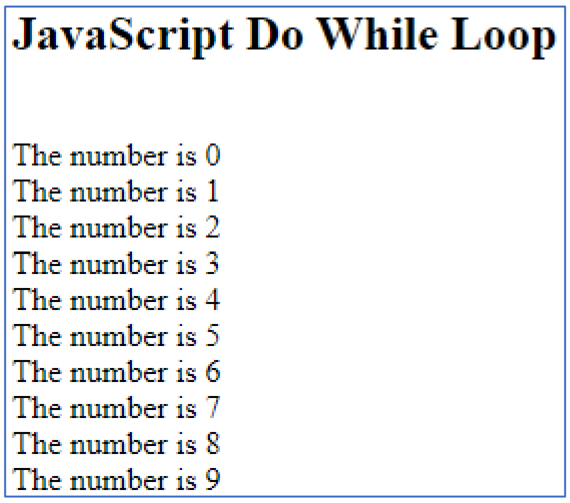
```
do {
  // code block to be executed
}
while (condition);
```

Example:

The example below uses a do while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Do While Loop</h2>
<p id="demo"></p>
<script>
let text = ""
let i = 0;
do {
  text += "<br>The number is " + i;
  i++; }
while (i < 10);
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Output:



```
JavaScript Do While Loop

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

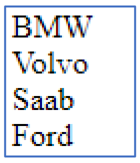
3. **Comparing For and While**

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

The loop in this example uses a for loop to collect the car names from the cars array:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
        <script>
        const cars = ["BMW", "Volvo", "Saab", "Ford"];
        let i = 0;
        let text = "";
        for (;cars[i];) {
          text += cars[i] + "<br>";
          i++;
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:

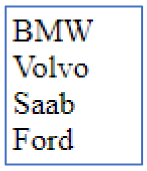


```
BMW
Volvo
Saab
Ford
```

The loop in this example uses a while loop to collect the car names from the cars array:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
        <script>
        const cars = ["BMW", "Volvo", "Saab", "Ford"];
        let i = 0;
        let text = "";
        while (cars[i]) {
          text += cars[i] + "<br>";
          i++;
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



```
BMW
Volvo
Saab
Ford
```

Lesson 5. JavaScript Break and Continue

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

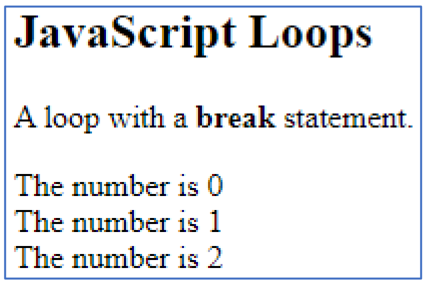
1. **The Break Statement**

   You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.

   The break statement can also be used to jump out of a loop:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Loops</h2>
<p>A loop with a <b>break</b> statement.</p>
<p id="demo"></p>

<script>
let text = "";
for (let i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Output:



```
JavaScript Loops

A loop with a break statement.

The number is 0
The number is 1
The number is 2
```

In the example above, the break statement ends the loop ("breaks" the loop) when the loop counter (i) is 3.

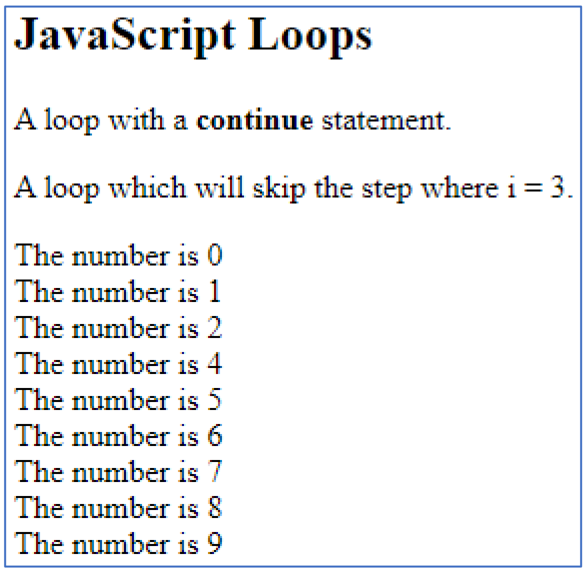
2. **The Continue Statement**

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Loops</h2>
<p>A loop with a <b>continue</b> statement.</p>

<p>A loop which will skip the step where i = 3.</p>

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

<script>
let text = "";
for (let i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Output:



```
JavaScript Loops

A loop with a continue statement.

A loop which will skip the step where i = 3.

The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

3. **JavaScript Labels**

   To label JavaScript statements you precede the statements with a label name and a colon:

```
label:
statements
```

The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.

Syntax:

```
break labelname;
continue labelname;
```

The continue statement (with or without a label reference) can only be used to skip one loop iteration.

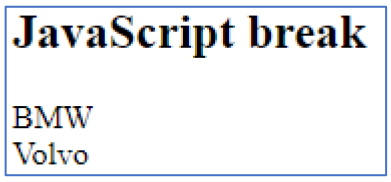
The break statement, without a label reference, can only be used to jump out of a loop or a switch.

With a label reference, the break statement can be used to jump out of any code block:

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript break</h2>
<p id="demo"></p>
<script>
const cars = ["BMW", "Volvo", "Saab", "Ford"];
let text = "";

list: {
  text += cars[0] + "<br>";
  text += cars[1] + "<br>";
  break list;
  text += cars[2] + "<br>";
  text += cars[3] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Output:



```
JavaScript break

BMW
Volvo
```

Lesson 6. JavaScript Iterables

Iterables are iterable objects (like Arrays).

Iterables can be accessed with simple and efficient code.

Iterables can be iterated over with for..of loops

1. **The For Of Loop**

   The JavaScript for..of statement loops through the elements of an iterable object.

   Syntax:

   ```
   for (variable of iterable) {
     // code block to be executed
   }
   ```

2. **Iterating**

   Iterating is easy to understand.

   It simply means looping over a sequence of elements.

   Here are some easy examples:

   - Iterating over a String
   - Iterating over an Array

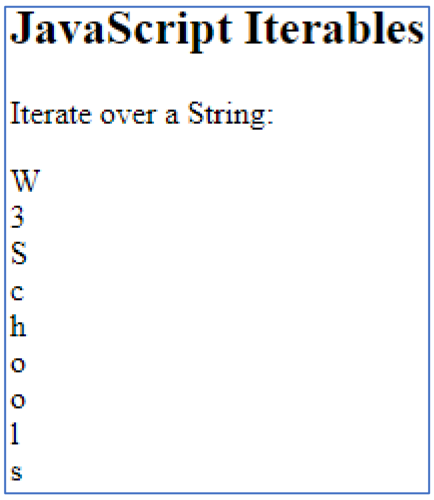
3. **Iterating Over a String**

   You can use a for..of loop to iterate over the elements of a string:

   ```
   <!DOCTYPE html>
   <html>
   <body>
   <h2>JavaScript Iterables</h2>
   <p>Iterate over a String:</p>
   <p id="demo"></p>
         <script>
         // Create a String
         const name = "W3Schools";

         // List all Elements
   ```

```
        let text = ""
        for (const x of name) {
          text += x + "<br>";
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



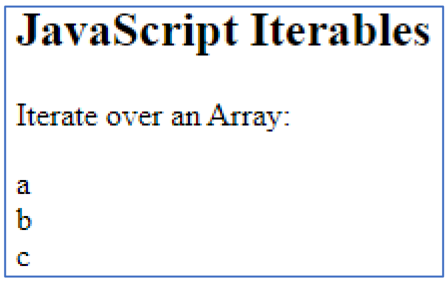
```
JavaScript Iterables

Iterate over a String:

W
3
S
c
h
o
o
l
s
```

4. **Iterating Over an Array**

   You can use a for..of loop to iterate over the elements of an Array:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Iterables</h2>
<p>Iterate over an Array:</p>
<p id="demo"></p>
        <script>
        // Create aa Array
        const letters = ["a","b","c"];
        // List all Elements
        let text = "";
        for (const x of letters) {
          text += x + "<br>";
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



JavaScript Iterables

Iterate over an Array:

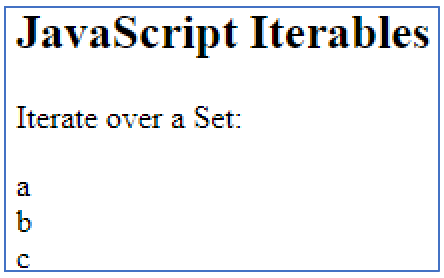
a
b
c

5. **Iterating Over a Set**

   You can use a for..of loop to iterate over the elements of a Set:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Iterables</h2>
<p>Iterate over a Set:</p>
<p id="demo"></p>
        <script>
        // Create a Set
        const letters = new Set(["a","b","c"]);
        // List all Elements
        let text = "";
        for (const x of letters) {
          text += x + "<br>";
        }
        document.getElementById("demo").innerHTML = text;
        </script>
</body>
</html>
```

Output:



JavaScript Iterables

Iterate over a Set:

a
b
c

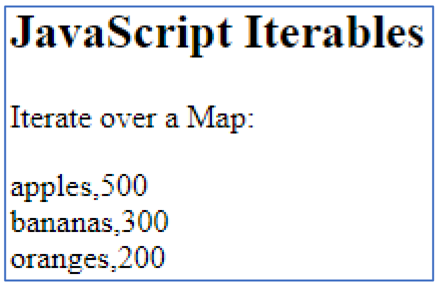
6. **Iterating Over a Map**

   You can use a for..of loop to iterate over the elements of a Map:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Iterables</h2>
<p>Iterate over a Map:</p>
<p id="demo"></p>
```

```
        <script>
        // Create a Map
        const fruits = new Map([
          ["apples", 500],
          ["bananas", 300],
          ["oranges", 200]
        ]);

        // List all entries
        let text = "";
        for (const x of fruits) {
          text += x + "<br>";
        }

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

Output:



```
JavaScript Iterables

Iterate over a Map:

apples,500
bananas,300
oranges,200
```

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_module6_lab.html).
- Create a new JavaScript file (e.g., script.js) linked to the HTML file.

Lab Tasks:

Task 1:

- Implement a for loop to iterate from 1 to 10 and print the square of each number.

Task 2

- Use a for/in loop to iterate over the properties of a predefined object and display key-value pairs.

Task 3

- Utilize a for/of loop to iterate over an array of strings and capitalize the first letter of each string.

Task 4

- Create a while loop that repeatedly prompts the user to enter a number until they input a negative number.

Task 5:

- Implement a do/while loop to simulate a simple guessing game where the user has to guess a predefined number.

Summary

- JavaScript offers a range of loop structures to iterate through code blocks or collections of data. The traditional for loop executes a block of code a specified number of times, while the for/in loop iterates over the properties of an object. The for/of loop, on the other hand, is preferred for iterating over the values of iterable objects like arrays and strings, providing a concise syntax for such operations. Meanwhile, the while loop repeats a block of code while a condition holds true, and the do/while loop ensures the block is executed at least once before evaluating the condition.
- Within these loops, JavaScript provides two control flow statements: break and continue. The break statement allows for the premature termination of a loop or switch statement, transferring control to the next statement outside the loop. Conversely, the continue statement skips the current iteration and proceeds to the next iteration of the loop. These statements offer fine-grained control over loop execution and can be particularly useful for handling specific conditions or streamlining code logic. Additionally, JavaScript supports iterables, enabling loops like for/of to iterate over the values of objects that implement the @@iterator method, such as arrays, strings, maps, and sets. This iterable support enhances the flexibility and efficiency of loop operations in JavaScript, facilitating the manipulation of diverse data structures with ease.

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

- **END OF MODULE FOR PRELIMINARY TERM PERIOD** –

SUBMISSION OF THE ASSESSMENT AND EXAMINATION FOR MIDTERM PERIOD WILL BE ANNOUNCED VIA iLearnU or GC

MAKE SURE TO CHECK THE DATES AND NOT FORGET TO TAKE IT AS SCHEDULED