LAB 3: INTRODUCTION TO JavaScript

What is JavaScript?

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

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

How to use Javascript in your document

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

Example:

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript in Body</h2>

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

<script>

document.getElementById("demo").innerHTML = "My First JavaScript";

</script>

</body>

</html>
```

A JavaScript `function` is a block of JavaScript code, that can be executed when "called" for. For example, a function can be called when an **event** occurs, like when the user clicks a button.

JavaScript in <head> or <body>?

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

Example:

```
<!DOCTYPE html>
<html>

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

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

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

Example:

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

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

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

</body>
</html>
```

External JavaScript

Scripts can also be placed in external files: the name of the file for example is myScript.js and contains this code:

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

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

```
<!DOCTYPE html>

<html>

<body>

<h2>External JavaScript</h2>

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

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

<p>(myFunction is stored in an external file called "myScript.js")</p>

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

</body>

</html>
```

JavaScript can "display" data in different ways:

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

Using innerHTML

To access an HTML element, JavaScript can use the `document.getElementById(id)` method. The `id` attribute defines the HTML element. The `innerHTML` property defines the HTML content:

Example:

<!DOCTYPE html>

<html><body>

<h2>My First Web Page</h2>

<p>My First Paragraph.</p>

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

<script>

document.getElementById("demo").innerHTML = 5 + 6;

</script>

</body>

</html>

Using document.write()

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

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>
</body>
</html>
```

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

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```

Using window.alert()

You can use an alert box to display data:

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
window.alert(5 + 6);
</script>
</body>
</html>
```

Using console.log()

For debugging purposes, you can use the `console.log()` method to display data.

Example

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

<h2>Activate debugging with F12</h2>

<p>Select "Console" in the debugger menu. Then click Run again.</p>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by a computer. In a programming language, these programming instructions are called **statements**. A **JavaScript program** is a list of programming **statements**. JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.

Example:

```
var a, b, c;      // Declare 3 variables
a = 5;            // Assign the value 5 to a
b = 6;            // Assign the value 6 to b
c = a + b;        // Assign the sum of a and b to c
```

multiple statements on one line are allowed: `a = 5; b = 6; c = a + b;`

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable. The following lines are equivalent:

```
var person = "Hege";
var person="Hege";
```

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

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

Example

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}. The purpose of code blocks is to define statements to be executed together. One place you will find statements grouped together in blocks, is in JavaScript functions:

Example

```
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello Dolly!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
```

Single Line Comments

Single line comments start with `//`.

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

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

Example

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";

// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Multi-line Comments

Multi-line comments start with `/*` and end with `*/`.

Any text between `/*` and `*/` will be ignored by JavaScript.

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

Example

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding `//` in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

Example

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

JavaScript Variables

JavaScript variables are containers for storing data values.

In this example, `x`, `y`, and `z`, are variables:

Example

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

From the example above, you can expect:

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

Much Like Algebra

In this example, `price1`, `price2`, and `total`, are variables:

Example

```
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
```

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

The Assignment Operator

In JavaScript, the equal sign (`=`) is an "assignment" operator, not an "equal to" operator. This is different from algebra. The following does not make sense in algebra: `x = x + 5`

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

The "equal to" operator is written like `==` in JavaScript.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

Example

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

Declaring JavaScript Variables

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

You declare a JavaScript variable with the `var` keyword:

`var carName;` or `var carName = "Volvo";`

In the example below, we create a variable called `carName` and assign the value "Volvo" to it. Then we "output" the value inside an HTML paragraph with id="demo":

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Variables</h2>
<p>Create a variable, assign a value to it, and display it:</p>
<p id="demo"></p>
<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
</body>
</html>
```

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with `var` and separate the variables by **comma**:

```
var person = "John Doe", carName = "Volvo", price = 200;
```

JavaScript Operators

The **assignment** operator (`=`) assigns a value to a variable.

Assignment

```
var x = 10;
```

The **addition** operator (`+`) adds numbers:

Adding

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

The **multiplication** operator (`*`) multiplies numbers.

Multiplying

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

Arithmetic operators are used to perform arithmetic on numbers:

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

++	Increment
--	Decrement

JavaScript String Operators

The `+` operator can also be used to add (concatenate) strings.

Example

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

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

Example

```
var txt1 = "What a very ";
txt1 += "nice day";
```

The result of txt1 will be:

```
What a very nice day
```

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

Example

```
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;
```

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

```
10
55
Hello5
```

JavaScript Comparison Operators

Operator	**Description**

==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript Logical Operators

Operator	**Description**
&&	logical and
||	logical or
!	logical not

JavaScript Data Types

JavaScript variables can hold many **data types**: numbers, strings, objects and more:

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

JavaScript Arrays

JavaScript arrays are written with square brackets. Array items are separated by commas. The following code declares (creates) an array called `cars`, containing three items (car names):

Example:

<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Arrays</h2>
<p>Array indexes are zero-based, which means the first item is [0].</p>

<p id="demo"></p>
<script>
var cars = ["Saab","Volvo","BMW"];

document.getElementById("demo").innerHTML = cars[1];
</script>

</body>
</html>

The following example also creates an Array, and assigns values to it:

Example

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Access the Elements of an Array

You access an array element by referring to the **index number**.

This statement accesses the value of the first element in `cars`:

```
var name = cars[0];
```

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Changing an Array Element

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

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

Example

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

Access the Full Array

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

Example

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

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

Examples

```
var x = cars.length;   // The length property returns the number of
elements
var y = cars.sort();   // The sort() method sorts arrays
```

Array methods are covered in the next chapters.

The length Property

The `length` property of an array returns the length of an array (the number of array elements).

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;   // the length of fruits is 4
```

Accessing the First Array Element

Example

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
```

Accessing the Last Array Element

Example

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var last = fruits[fruits.length - 1];
```

Adding Array Elements

The easiest way to add a new element to an array is using the `push()` method:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");    // adds a new element (Lemon) to fruits
```

New element can also be added to an array using the `length` property:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";    // adds a new element (Lemon) to
fruits
```

n JavaScript, **arrays** always use **numbered indexes**.

Example

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length;      // person.length will return 3
var y = person[0];          // person[0] will return "John"
```

JavaScript Objects

JavaScript objects are written with curly braces `{}`.

Object properties are written as name:value pairs, separated by commas.

Example

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Objects</h2>

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

<script>

var person = {

  firstName: "John",

  lastName : "Doe",

  age : 50,

  eyeColor : "blue"

};

document.getElementById("demo").innerHTML =

person.firstName + " is " + person.age + " years old.";

</script>

</body>

</html>
```

The typeof Operator

You can use the JavaScript `typeof` operator to find the type of a JavaScript variable.

The `typeof` operator returns the type of a variable or an expression:

Example

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

JavaScript Functions

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

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function which performs a calculation, and
returns the result:</p>
<p id="demo"></p>
<script>
function myFunction(p1, p2) {
  return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
</body>
</html>
```

The code to be executed, by the function, is placed inside curly brackets: **{}**

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

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function Return

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

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

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

Example

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

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function which performs a calculation and returns the result:</p>
<p id="demo"></p>
<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;
function myFunction(a, b) {
  return a * b;
}
</script>
</body>
</html>
```

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Functions</h2>
<p>Outside myFunction() carName is undefined.</p>
<p id="demo1"></p>
<p id="demo2"></p>
<script>
myFunction();
function myFunction() {
  var carName = "Volvo";
  document.getElementById("demo1").innerHTML =
  typeof carName + " " + carName;
}
document.getElementById("demo2").innerHTML =
typeof carName;
</script>
</body>
</html>
```

JavaScript Events

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

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected. HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

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

With double quotes:

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

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

Example

```
<!DOCTYPE html>
<html>
<body>
<button onclick="this.innerHTML=Date()">The time is?</button>
</body>
</html>
```

Or

```
<!DOCTYPE html>

<html>

<body>

<p>Click the button to display the date.</p>

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

<script>

function displayDate() {

  document.getElementById("demo").innerHTML = Date();

}

</script>

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

</body>

</html>
```

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed

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

What can JavaScript Do?

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

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

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

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

String Methods and Properties

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Length

The `length` property returns the length of a string:

Example

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

Finding a String in a String

The `indexOf()` method returns the index of (the position of) the `first` occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

The `lastIndexOf()` method returns the index of the **last** occurrence of a specified text in a string:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

Both `indexOf()`, and `lastIndexOf()` return -1 if the text is not found.

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("John");
```

Searching for a String in a String

The `search()` method searches a string for a specified value and returns the position of the match:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

The slice() Method

`slice()` extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

The result of res will be:

```
Banana
```

The substring() Method

`substring()` is similar to `slice()`.

The difference is that `substring()` cannot accept negative indexes.

Example

```
var str = "Apple, Banana, Kiwi";
var res = str.substring(7, 13);
```

Replacing String Content

The `replace()` method replaces a specified value with another value in a string:

Example

```
str = "Please visit Microsoft!";
var n = str.replace("Microsoft", "W3Schools");
```

Converting to Upper and Lower Case

A string is converted to upper case with `toUpperCase()`:

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper
```

A string is converted to lower case with `toLowerCase()`:

Example

```
var text1 = "Hello World!";       // String
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower
```

The concat() Method

`concat()` joins two or more strings:

Example

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2);
```

String.trim()

The `trim()` method removes whitespace from both sides of a string:

Example

```
var str = "       Hello World!        ";
alert(str.trim());
```

The toString() Method

The `toString()` method returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

Example

```
var x = 123;
x.toString();            // returns 123 from variable x
(123).toString();        // returns 123 from literal 123
(100 + 23).toString();   // returns 123 from expression 100 + 23
```

The toExponential() Method

`toExponential()` returns a string, with a number rounded and written using exponential notation.

A parameter defines the number of characters behind the decimal point:

Example

```
var x = 9.656;
x.toExponential(2);     // returns 9.66e+0
x.toExponential(4);     // returns 9.6560e+0
x.toExponential(6);     // returns 9.656000e+0
```

Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert variables to numbers:

- The `Number()` method
- The `parseInt()` method
- The `parseFloat()` method

`parseInt()` parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

Example

```
parseInt("10");         // returns 10
parseInt("10.33");      // returns 10
parseInt("10 20 30");   // returns 10
parseInt("10 years");   // returns 10
parseInt("years 10");   // returns NaN
```

The parseFloat() Method

`parseFloat()` parses a string and returns a number. Spaces are allowed. Only the first number is returned:

Example

```
parseFloat("10");        // returns 10
parseFloat("10.33");     // returns 10.33
parseFloat("10 20 30");  // returns 10
parseFloat("10 years");  // returns 10
parseFloat("years 10");  // returns NaN
```

JavaScript if else and else if

Conditional Statements

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

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
}
```

The else Statement

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
}
```

The else if Statement

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":

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

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.

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:

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

The result of day will be:

```
Wednesday
```

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

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

The For Loop

The `for` loop has the following syntax:

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

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

Statement 2 defines the condition for executing the code block.

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

Example

```
for (i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

The For/In Loop

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

Example

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
  text += person[x];
}
```

The For/Of Loop

The JavaScript `for/of` statement loops through the values of an iterable objects `for/of` lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

The `for/of` loop has the following 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.

Looping over an Array

Example

```
var cars = ['BMW', 'Volvo', 'Mini'];
var x;

for (x of cars) {
  document.write(x + "<br >");
}
```

Looping over a String

Example

```
var txt = 'JavaScript';
var x;

for (x of txt) {
  document.write(x + "<br >");
}
```

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:

Example

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

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.

The `break` statement breaks the loop and continues executing the code after the loop (if any):

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}
```

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:

Example

```
for (i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

JavaScript Form Validation

HTML form validation can be done by JavaScript. If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>
<form name="myForm" action="/action_page.php" onsubmit="return
validateForm()" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Automatic HTML Form Validation

HTML form validation can be performed automatically by the browser:If a form field (fname) is empty, the `required` attribute prevents this form from being submitted:

```
<!DOCTYPE html>

<html>

<body>

<form action="/action_page.php" method="post">

  <input type="text" name="fname" required>

  <input type="submit" value="Submit">

</form>

<p>If you click submit, without filling out the text field,

your browser will display an error message. </p>

</body></html>
```

EXO1: Write an HTML document. By using JavaScript functions, change the content of any tag <P>, its style, and finally hide the content of the <p> element.

EXO2: In mathematics, the Fibonacci numbers are the numbers in the following integer sequence, called the Fibonacci sequence, and characterized by the fact that every number after the first two is the sum of the two preceding ones. Depending on the chosen starting point of the sequence (0 or 1) the sequence would look like this: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …or this: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Write a function to return an n^{th} element in Fibonacci sequence. Then the result will be printed in a web page which, when loaded, displays an input pop-up which requests the rank of the function then displays in the page:

"At hh hours mm, the n th term of the Fibonnacci sequence is ... ",

hh and mm being the time at which the script is executed.

The text should appear:
- in a brown area with a yellow dotted border
- with an orange font, the text being centered in the area
- the n is displayed in italics, the result in large and bold;

EXO3: Arrays

On the same page, create an array of integers of size= 10, initialized randomly by values <100.

- Display it, with the elements separated by "-"
- Sort it,
- Display the result.

ExO4:

On the same page, insert an image.
- Make that when the mouse passes over the image, it changes, then when the mouse leave the image, the 1st is displayed.

- In the following: add a button to your page, like this one:

```
<input type = "button"  id = "idbouton" name = "nameofbuttom" value =
"Button text" onclick = "function()" />
```

-Create a function that displays a pop-up giving the following information about an image: the source, name and size (height × width) when clicking on it. Test it on your previous image.

-Using the previous image, create a function that randomly generates a number between 0 and 5 allowing you to choose between
6 images then which modifies the src and id attributes of the above image according to the image chosen at random.