Programmazione Web a.a. 2017/2018
Javascript

- PhD Ing.Antonino Raucea
- antonino.raucea@dieei.unict.it

Introduction

JavaScript is the programming language of HTML and the Web

- JavaScript is a client-side script, meaning the browser processes the code instead of the web server.
- Client-side scripts are commonly used when we want to validate data before sending it to the web server, adjusting the interface in response to user feedback, and for implementing other advanced features.
- This being said, this is less of an issue now, and JavaScript can reduce the number of communications to a server, reducing transmission time and improving performance.

JavaScript and Java are completely different languages, both in concept and design.

- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.
- ECMA-262 is the official name of the standard. ECMAScript is the official name of the language.

Why Study JavaScript?

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

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

JavaScript Versions

Year	Name	Description
1997	ECMAScript 1	First Edition.
1998	ECMAScript 2	Editorial changes only.
1999	ECMAScript 3	Added Regular Expressions. Added try/catch.
	ECMAScript 4	Was never released.
2009	ECMAScript 5	Added "strict mode". Added JSON support.
2011	ECMAScript 5.1	Editorial changes.
2015	ECMAScript 6	Added classes and modules.
2016	ECMAScript 7	Added exponential operator (**). Added Array.prototype.includes.

Browser Support

- ECMAScript 3 is fully supported in all browsers.
- ECMAScript 5 is fully supported in all modern browsers*.
- ECMAScript 6 is partially supported in all modern browsers.
- ECMAScript 7 is poorly supported in all browsers.

> ECMAScript 6 is also called ECMAScript 2015.
>
> ECMAScript 7 is also called ECMAScript 2016.

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is getElementById().

This example uses the method to "find" an HTML element (with id="demo") and changes the element content (innerHTML) to "Hello JavaScript":

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

JavaScript Can Change HTML Attributes

Get the handle of an HTML object by getElementById()

```
<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">
Turn on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

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

JavaScript Can Change HTML Attributes

Get the handle of an HTML object by getElementById()

```
<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">
Turn on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

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

JavaScript Can Change HTML Styles (CSS)

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

- Change font size

```
document.getElementById("demo").style.fontSize = "35px";
or
document.getElementById('demo').style.fontSize = '35px';
```

- Hide an item

```
document.getElementById("demo").style.display = "none";
or
document.getElementById('demo').style.display = 'none';
```

- Show an item

```
document.getElementById("demo").style.display = "block";
or
document.getElementById('demo').style.display = 'block';
```

JavaScript Where To: The <script> Tag

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

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

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

JavaScript Functions and Events

- A JavaScript function is a block of JavaScript code, that can be executed when "called" for.
- For example, a function can be called when an event occurs, like when the user clicks a button.
- You can place any number of scripts in an HTML document.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

JavaScript in <head>

```
<!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>
```

JavaScript in <body>

```
<!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>
```

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

External JavaScript

- External file: myScript.js

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

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the file extension .js.

To use an external script, put the name of the script file in the ***src (source)*** attribute of a **<script>** tag:

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

Placing scripts in external files has some advantages:

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

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

External References

External scripts can be referenced with a full URL or with a path relative to the current web page.

This example uses a full URL to link to a script:

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

JavaScript Output

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:

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

<h1>My First Web Page</h1>
<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():

```
<!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 fully loaded, will delete all existing HTML:

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

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

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

</body>
</html>
```

JavaScript Syntax

- A computer program is a list of "instructions" to be "executed" by the computer.
- In a programming language, these program instructions are called statements.
- JavaScript is a programming language.
- JavaScript statements are separated by semicolons.

In HTML, JavaScript programs are executed by the web browser.

JavaScript Syntax

- JavaScript Statements
  - JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.
- JavaScript Values
  - The JavaScript syntax defines two types of values: Fixed values and variable values.
  - Fixed values are called **literals**. Variable values are called **variables**.
- JavaScript Literals
  - The most important rules for writing fixed values are:
  - **Numbers** are written with or without decimals: **10.50** **1000**
  - Strings are text, written within double or single quotes: **"John Doe"**

JavaScript Variables

In a programming language, variables are used to store data values.

JavaScript uses the var keyword to declare variables.

An equal sign is used to assign values to variables.

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

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

Javascript as a programming language

- Operators: + - * /
- Assignment operator: =
- Expressions: An expression is a combination of values, variables, and operators, which computes to a value
- Concatenate string operator: "John" + " " + "Doe"
- Comments:
  - // comment in line;
  - /* comment */

JavaScript Identifiers

Identifiers are names.

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

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

In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).

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

Numbers are not allowed as the first character.
This way JavaScript can easily distinguish identifiers from numbers.

JavaScript is Case Sensitive

All JavaScript identifiers are case sensitive.

The variables **lastName** and **lastname**, are two different variables.

```
var lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

JavaScript does not interpret VAR or Var as the keyword var.

JavaScript and Camel Case

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

- Hyphens*: first-name, last-name, master-card, inter-city.
- Underscore: first_name, last_name, master_card, inter_city.
- Upper Camel Case (Pascal Case): FirstName, LastName, MasterCard, InterCity.
- Lower Camel Case:firstName, lastName, masterCard, interCity.

*Hyphens are not allowed in JavaScript. It is reserved for subtractions.

JavaScript White Space

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

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

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

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

JavaScript Keywords

JavaScript statements often start with a keyword to identify the Ja

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

JavaScript Operator

There are new operator

Value	Operator	Description	Example
17	new	Create	new Date()
15	typeof	Type	typeof x
12	>>>	Shift right (unsigned)	x >>> 2
10	==	Equal	x == y
10	===	Strict equal	x === y

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 Types are Dynamic.

```
var x;               // Now x is undefined
var x = 5;           // Now x is a Number
var x = "John";      // Now x is a String
```

Javascript Data Type Cast: Number to String

- var x = "Volvo" + 16; => “Volvo16”
- var x = 16 + 4 + "Volvo"; => “20Volvo”
- var x = "Volvo" + 16 + 4; => “Volvo164”

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):

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

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

JavaScript Arrays

JavaScript objects are written with curly braces.

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

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

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

The typeof Operator

- 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:

```
typeof ""                    // Returns "string"
typeof "John"                // Returns "string"
typeof "John Doe"            // Returns "string"
typeof 0                     // Returns "number"
typeof 314                   // Returns "number"
typeof 3.14                  // Returns "number"
typeof typeof “Hello”        // Returns ???? (Try it)
```

Undefined

- In JavaScript, a variable without a value, has the value undefined. The typeof is also undefined.
- Any variable can be emptied, by setting the value to undefined. The type will also be undefined.

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

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

Empty and Null Values

Empty

- An empty value has nothing to do with undefined.
- An empty string has both a legal value and a type.

```
var car = "";                    // The value is "", the typeof is "string"
```

Null

- In JavaScript null is "nothing". It is supposed to be something that doesn't exist.
- Unfortunately, in JavaScript, the data type of null is an object.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null;              // Now value is null, but type is still an object
```

•Difference Between Undefined and Null

- Undefined and null are equal in value but different in type:

```
typeof undefined              // undefined
typeof null                   // object

null === undefined            // false
null == undefined             // true
```

Primitive Data

A primitive data value is a single simple data value with no additional properties and methods.

The typeof operator can return one of these primitive types:

- string
- number
- boolean
- undefined

```
typeof "John"                // Returns "string"
typeof 3.14                  // Returns "number"
typeof true                  // Returns "boolean"
typeof false                 // Returns "boolean"
typeof x                     // Returns "undefined" (if x has no value)
```

Complex Data

The typeof operator can return one of two complex types:

- function
- object

The typeof operator returns object for both objects, arrays, and null.

The typeof operator does not return object for functions.

```
typeof {name:'John', age:34} // Returns "object"
typeof [1,2,3,4]             // Returns "object" (not "array", see note below)
typeof null                  // Returns "object"
typeof function myFunc(){}   // Returns "function"
```

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).

```
function myFunction(p1, p2) {
    return p1 * p2;
}
```

JavaScript Objects

- You have already learned that JavaScript variables are containers for data values.
- Objects are variables too. But objects can contain many values.

```
var car = {type:"Fiat", model:"500", color:"white"};
```

Object Properties

- The name:values pairs (in JavaScript objects) are called properties.

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

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Object Methods

- Methods are actions that can be performed on objects.
- Methods are stored in properties as function definitions.

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Object example

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

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

HTML 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.

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

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

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

```
Math.PI;                    // returns 3.141592653589793
Math.round(4.7);            // returns 5
Math.round(4.4);            // returns 4
Math.pow(8, 2);             // returns 64
Math.sqrt(64);              // returns 8
Math.abs(-4.7);             // returns 4.7
Math.ceil(4.4);             // returns 5
Math.floor(4.7);            // returns 4
Math.min(0, 150, 30, 20, -8, -200);  // returns -200
Math.max(0, 150, 30, 20, -8, -200);  // returns 150
Math.random();              // returns a random number
```

Math.random()

- Math.random() returns a random number between 0 (inclusive), and 1 (exclusive):

```
Math.random();   => [0,1[
```

- Random Integers

```
Math.floor(Math.random() * 10); => [0,9]
Math.floor(Math.random() * 10) + 1; => [1,10]
```

JavaScript JSON

- JSON is a format for storing and transporting data.
- JSON is often used when data is sent from a server to a web page.

- JSON stands for JavaScript Object Notation
- JSON is lightweight data interchange format
- JSON is language independent *
- JSON is "self-describing" and easy to understand

JavaScript JSON example

```
{
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]
}
```

The JSON Format Evaluates to JavaScript Objects
The JSON format is syntactically identical to the code for creating JavaScript objects.

Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

Converting a JSON Text to a JavaScript Object

A common use of JSON is to read data from a web server, and display the data in a web page.

For simplicity, this can be demonstrated using a string as input.

First, create a JavaScript string containing JSON syntax:

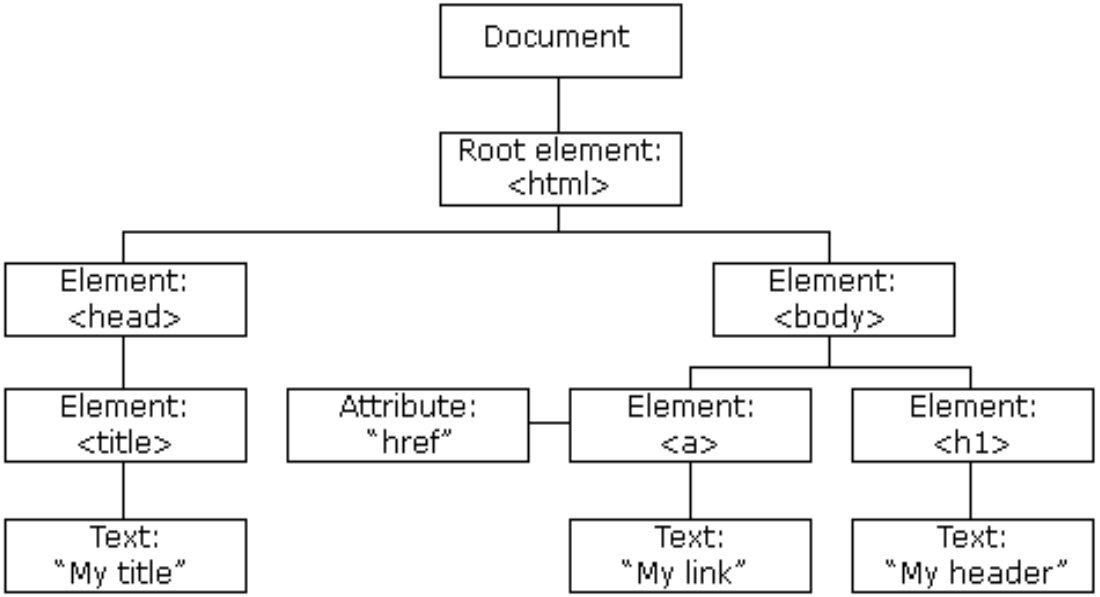
```
var text = '{ "employees" : [' +
     '{ "firstName":"John" , "lastName":"Doe" },' +
     '{ "firstName":"Anna" , "lastName":"Smith" },' +
     '{ "firstName":"Peter" , "lastName":"Jones" } ]}';

var obj = JSON.parse(text);
```

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a Document Object Model of the page.

The HTML DOM model is constructed as a tree of Objects:



The HTML DOM (Document Object Model)

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

The DOM Programming Interface

HTML DOM methods are actions you can perform (on HTML Elements).

HTML DOM properties are values (of HTML Elements) that you can set or change.

- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as objects.
- The programming interface is the properties and methods of each object.
- A property is a value that you can get or set (like changing the content of an HTML element).
- A method is an action you can do (like add or deleting an HTML element).

Example

- **getElementById** is a method, while **innerHTML** is a property

```
<html>
<body>

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

- The most common way to access an HTML element is to use the id of the element.
- The easiest way to get the content of an element is by using the innerHTML property.
- The innerHTML property is useful for getting or replacing the content of HTML elements.

The innerHTML property can be used to get or change any HTML element, including <html> and <body>.

JavaScript HTML DOM Document

The HTML DOM document object is the owner of all other objects in your web page.

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

DOM Methods

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements

Method	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.setAttribute(*attribute, value*)	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

DOM Methods

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*element*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

Document Objects

- document.anchors
- document.baseURI
- document.body
- document.cookie
- document.doctype
- document.documentElement
- document.documentMode
- document.documentURI
- document.domain
- document.domConfig
- document.embeds
- document.forms
- document.head
- document.images
- document.implementation
- document.inputEncoding
- document.lastModified
- document.links
- document.readyState
- document.referrer
- document.scripts
- document.strictErrorChecking
- document.title
- document.URL

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are a couple of ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with id="intro":

```
var myElement = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in myElement).

If the element is not found, myElement will contain null

Finding HTML Elements by Tag Name

This example finds all <p> elements:

```
var x = document.getElementsByTagName("p");
```

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use getElementsByClassName().

This example returns a list of all elements with class="intro".

```
var x = document.getElementsByClassName("intro");
```

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

This example returns a list of all <p> elements with class="intro".

```
var x = document.querySelectorAll("p.intro");
```

Finding HTML Elements by HTML Object Collections

This example finds the form element with id="frm1", in the forms collection, and displays all element values:

```
var x = document.forms["frm1"];
var text = "";
var i;
for (i = 0; i < x.length; i++) {
    text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

The following HTML objects (and object collections) are also accessible:

document.anchors
document.body
document.documentElement
document.embeds
document.forms
document.head
document.images
document.links
document.scripts
document.title

JavaScript HTML DOM - Changing HTML

JavaScript can create dynamic HTML content:

Date: Sun Dec 17 2017 22:46:48 GMT+0100 (ora solare Europa occidentale)

In JavaScript, document.write() can be used to write directly to the HTML output stream:

Changing HTML Content:

The easiest way to modify the content of an HTML element is by using the innerHTML property.

JavaScript HTML DOM - Changing CSS

The HTML DOM allows JavaScript to change the style of HTML elements.

Changing HTML Style

To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property = new style
```

Create the Animation Using JavaScript

```
#container {
    width: 400px;
    height: 400px;
    position: relative;
    background: yellow;
}
#animate {
    width: 50px;
    height: 50px;
    position: absolute;
    background: red;
}
```

```
<div id ="container">
    <div id ="animate">My animation will go here</div>
</div>


  function myMove() {
      var elem = document.getElementById("animate");
      var pos = 0;
      var id = setInterval(frame, 5);
      function frame() {
          if (pos == 350) {
              clearInterval(id);
          } else {
              pos++;
              elem.style.top = pos + 'px';
              elem.style.left = pos + 'px';
          }
      }
  }
```

JavaScript HTML DOM Events

- HTML DOM allows JavaScript to react to HTML events
- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

  onclick=JavaScript

  - When a user clicks the mouse
  - When a web page has loaded
  - When an image has been loaded
  - When the mouse moves over an element
  - When an input field is changed
  - When an HTML form is submitted
  - When a user strokes a key

Example

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

<p id=”p1” onclick="this.innerHTML = 'Ooops!'">Click on this text!</p>
<p id=”p2” onclick="changeText(this)">Click on this text!</p>

<script>
function changeText(id) {
    id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

The onload and onunload Events

The onload and onunload events are triggered when the user enters or leaves the page.

The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onload and onunload events can be used to deal with cookies.

```
<body onload="checkCookies()">
```

The onchange Event

The onchange event is often used in combination with validation of input fields.

Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

```
<input type="text" id="fname" onchange="upperCase()">
```

The onmouseover and onmouseout Events

The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element:

```
<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver(obj) {
    obj.innerHTML = "Thank You"
}

function mOut(obj) {
    obj.innerHTML = "Mouse Over Me"
}
</script>
```

The onmousedown, onmouseup and onclick Events

The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

```
<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-color:#D94A38;width:90px;height:20px;padding:40px;">
Click Me</div>

<script>
function mDown(obj) {
    obj.style.backgroundColor = "#1ec5e5";
    obj.innerHTML = "Release Me";
}

function mUp(obj) {
    obj.style.backgroundColor="#D94A38";
    obj.innerHTML="Thank You";
}
</script>
```

JavaScript HTML DOM EventListener

- The addEventListener() method, Add an event listener that fires when a user clicks a button:

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

The addEventListener() method attaches an event handler to the specified element.

The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The addEventListener() method makes it easier to control how the event reacts to bubbling.

When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the removeEventListener() method.

JavaScript HTML DOM Navigation

With the HTML DOM, you can navigate the node tree using node relationships.

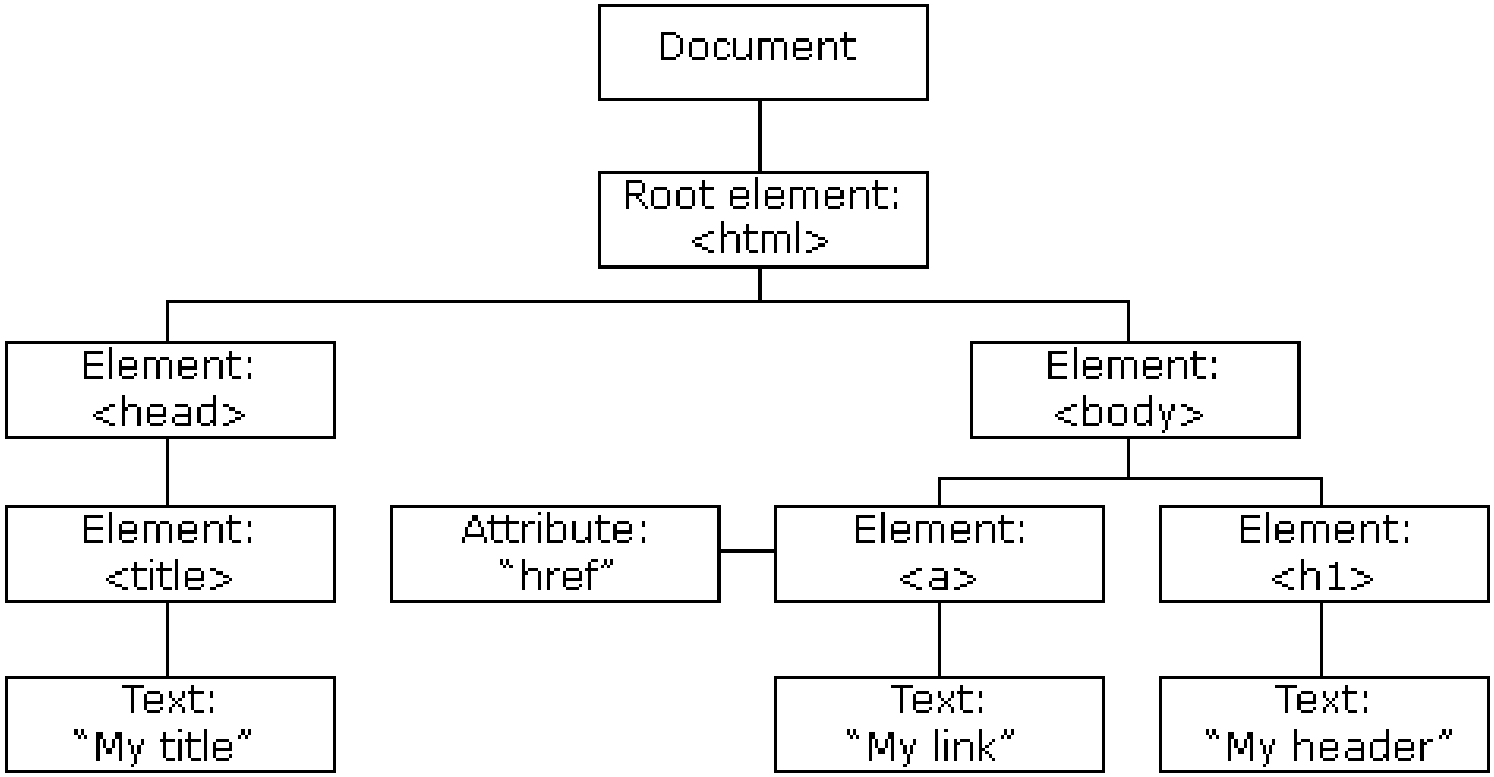
According to the W3C HTML DOM standard, everything in an HTML document is a node:

- The entire document is a document node
- Every HTML element is an element node
- The text inside HTML elements are text nodes
- Every HTML attribute is an attribute node (deprecated)
- All comments are comment nodes

With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.

New nodes can be created, and all nodes can be modified or deleted.

JavaScript HTML DOM Navigation



Node Relationships

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

Node Relationships

From the HTML above you can read:

<html> is the root node
<html> has no parents
<html> is the parent of <head> and <body>
<head> is the first child of <html>
<body> is the last child of <html>
and:

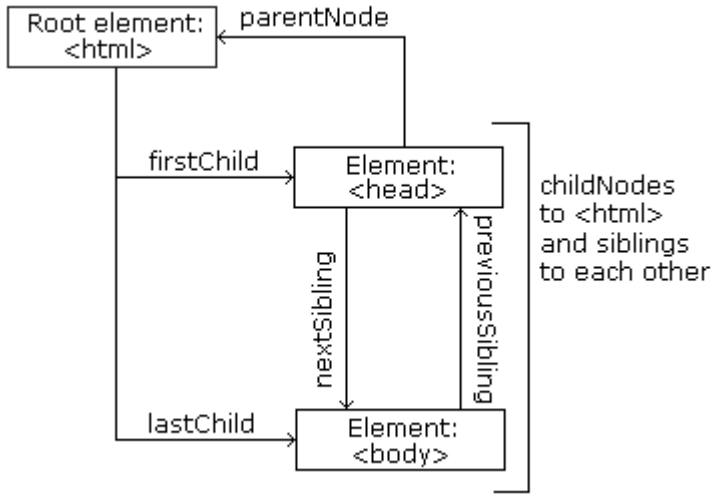
<head> has one child: <title>
<title> has one child (a text node): "DOM Tutorial"
<body> has two children: <h1> and <p>
<h1> has one child: "DOM Lesson one"
<p> has one child: "Hello world!"
<h1> and <p> are siblings

```
<html>

  <head>
      <title>DOM Tutorial</title>
  </head>

  <body>
      <h1>DOM Lesson one</h1>
      <p>Hello world!</p>
  </body>

</html>
```



Navigating Between Nodes

You can use the following node properties to navigate between nodes with JavaScript:

parentNode
childNodes[nodenumber]
firstChild
lastChild
nextSibling
previousSibling

Fine