WEB PROGRAMMING 0601632 2

BY DR BUNDIT THANASOPON

JAVASCRIPT REVISION

SRC: HTTPS://WWW.W3SCHOOLS.COM/JS/DEFAULT.ASP

EXAMPLES OF WHAT JAVASCRIPT CAN DO

- JavaScript Can Change HTML Content
  - `document.getElementById("demo").innerHTML = "Hello JavaScript";`
- JavaScript Can Change HTML Attribute Values
  - `document.getElementById(”myImage").src = "hello.jpg";`
- JavaScript Can Change HTML Styles (CSS)
  - `document.getElementById("demo").style.fontSize = "35px";`
- JavaScript Can Hide HTML Elements
  - `document.getElementById("demo").style.display = "none";`

THE <SCRIPT> TAG

- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

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

- External JavaScript

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

EXTERNAL JAVASCRIPT ADVANTAGES

- Placing scripts in external files has some advantages:
  - It separates HTML and code
  - It makes HTML and JavaScript easier to read and maintain
  - Cached JavaScript files can speed up page loads

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

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

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().

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

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

JAVASCRIPT KEYWORDS

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 SYNTAX

- JavaScript Identifiers
  - Identifiers are names.
  - In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).
  - In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($).
- JavaScript is Case Sensitive
  - All JavaScript identifiers are **case sensitive**.
  - The variables lastName and lastname, are two different variables
- JavaScript Comments
  - Code after double slashes // or between /* and */ is treated as a **comment**.

JAVASCRIPT OPERATORS

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES6)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

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
&&	Logical and
||	Logical or
!	Logical not

JAVASCRIPT DATA TYPES

- JavaScript Types are Dynamic - this means that the same variable can be used to hold different data types:

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

- 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 (true or false)
  - undefined

JAVASCRIPT DATA TYPES

- 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 return “ function” for functions.

```
typeof {name:'John', age:34}
// Returns "object"
typeof [1,2,3,4]
// Returns "object"
typeof null
// Returns "object"
typeof function myFunc(){}
// Returns "function"
```

The **typeof** operator returns **"object"** for arrays because in JavaScript arrays are objects.

JAVASCRIPT FUNCTONS

- 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;   // The function returns the product of p1 and p2
}
```

- The () Operator Invokes the Function
  - toCelsius refers to the function object, and toCelsius() refers to the function result.
  - Accessing a function without () will return the function definition instead of the function result:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

JAVASCRIPT OBJECTS

- What is an object? In real life, a car is an **object**. A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

- All cars have the same **properties**, but the property **values** differ from car to car.
- All cars have the same **methods**, but the methods are performed **at different times**.

JAVASCRIPT OBJECTS

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

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

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.

JAVASCRIPT EVENTS

```
<button onclick="document.getElementById('demo').innerHTML =
Date()">The time is?</button>

<button onclick="this.innerHTML = Date()">The time is?</button>

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

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

EXAMPLES

```
<button>Click me</button>
<p>No handler here.</p>
<script>
    let button = document.querySelector("button");
    button.addEventListener("click", () => {
console.log("Button clicked."); });
</script>
```

```
<button>Act-once button</button>
<script>
      let button = document.querySelector("button");
      function once() { console.log("Done.");
        button.removeEventListener("click", once); }
      button.addEventListener("click", once);
</script>
```

PROPAGATION

- For most event types, handlers registered on nodes with children will also receive events that happen in the children.
- If a button inside a paragraph is clicked, event handlers on the paragraph will also see the click event.

- At any point, an event handler can call the **stopPropagation** method on the event object to prevent handlers further up from receiving the event

```
<p>A paragraph with a <button>button</button>.</p>
<script>
  let para = document.querySelector("p");
  let button = document.querySelector("button");
  para.addEventListener("mousedown", () => {
      console.log("Handler for paragraph.");
  });
  button.addEventListener("mousedown", event => {
      console.log("Handler for button.");
      if (event.button == 2) event.stopPropagation();
  });
</script>
```

DEFAULT ACTIONS

- Many events have a default action associated with them.
  - If you click a link, you will be taken to the link's target.
  - If you press the down arrow, the browser will scroll the page down.
  - If you right-click, you'll get a context menu.
- You can use **preventDefault** to prevent the default action from happening.

```
<a href="https://developer.mozilla.org/">MDN</a>
<script>
  let link = document.querySelector("a");
  link.addEventListener("click", event => {
    console.log("Nope."); event.preventDefault();
  });
</script>
```

LET'S DO SOME EXERCISES

- Complete exercise 1 - 3

JAVASCRIPT ARRAYS

- An array is a special variable, which can hold more than one value at a time.

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

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

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

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

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

JAVASCRIPT ARRAY ITERATION METHODS - FOREACH

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

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

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

- Note that the function takes 3 arguments:
  - The item value
  - The item index
  - The array itself

JAVASCRIPT ARRAY ITERATION METHODS - MAP

- The map() method creates a new array by performing a function on each array element.
  - The map() method **does not** execute the function for array elements without values.
  - The map() method **does not** change the original array.
- This example multiplies each array value by 2:

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

JAVASCRIPT ARRAY ITERATION METHODS - FILTER

- The filter() method creates a new array with array elements that passes a test.
- This example creates a new array from elements with a value larger than 18:

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

JAVASCRIPT ARRAY ITERATION METHODS - REDUCE

- The reduce() method runs a function on each array element to produce (reduce it to) a single value.
- The reduce() method works from left-to-right and does not reduce the original array.
- This example finds the sum of all numbers in an array:

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction);

function myFunction(total, value, index, array) {
  return total + value;
}
```

Note that the function takes 4 arguments:

- The total (the initial value / previously returned value)
- The item value
- The item index
- The array itself

JAVASCRIPT ARRAY ITERATION METHODS - INDEXOF

- The indexOf() method searches an array for an element value and returns its position.
- **Note:** The first item has position 0, the second item has position 1, and so on.

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");
```

- Array.lastIndexOf() is the same as Array.indexOf(), but searches from the end of the array.

SPLICE

- The splice() method adds/removes items to/from an array, and returns the removed item(s).

array.splice(*index*, *howmany*, *item1*, ..., *itemX*)

Parameter	Description
index	Required. An integer that specifies at what position to add/remove items, Use negative values to specify the position from the end of the array
howmany	Optional. The number of items to be removed. If set to 0, no items will be removed
item 1, ..., *itemX*	Optional. The new item(s) to be added to the array

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 1, "Lemon", "Kiwi");
```

LET'S DO SOME EXERCISES

- Complete exercise 4 - 5

WINDOW LOCALSTORAGE PROPERTY

- The localStorage and sessionStorage properties allow to save key/value pairs in a web browser.
  - The localStorage object stores data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.
  - The localStorage property is read-only.

```
localStorage.setItem("key", "value");

var lastname = localStorage.getItem("key");

localStorage.removeItem("key");
```

JAVASCRIPT COOKIES

- Cookies are data, stored in small text files, on your computer.
  - When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user - **stateless**
- Cookies were invented to solve the problem "how to remember information about the user":
  - When a user visits a web page, his name can be stored in a cookie.
  - Next time the user visits the page, the cookie "remembers" his name.

- When a browser requests a web page from a server, cookies belonging to the page is added to the request. This way the server gets the necessary data to "remember" information about users.

CREATE A COOKIE WITH JAVASCRIPT

- JavaScript can create, read, and delete cookies with the ”document.cookie” property.
  - Create

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";
```

  - Read

```
var x = document.cookie;
```

  - Delete

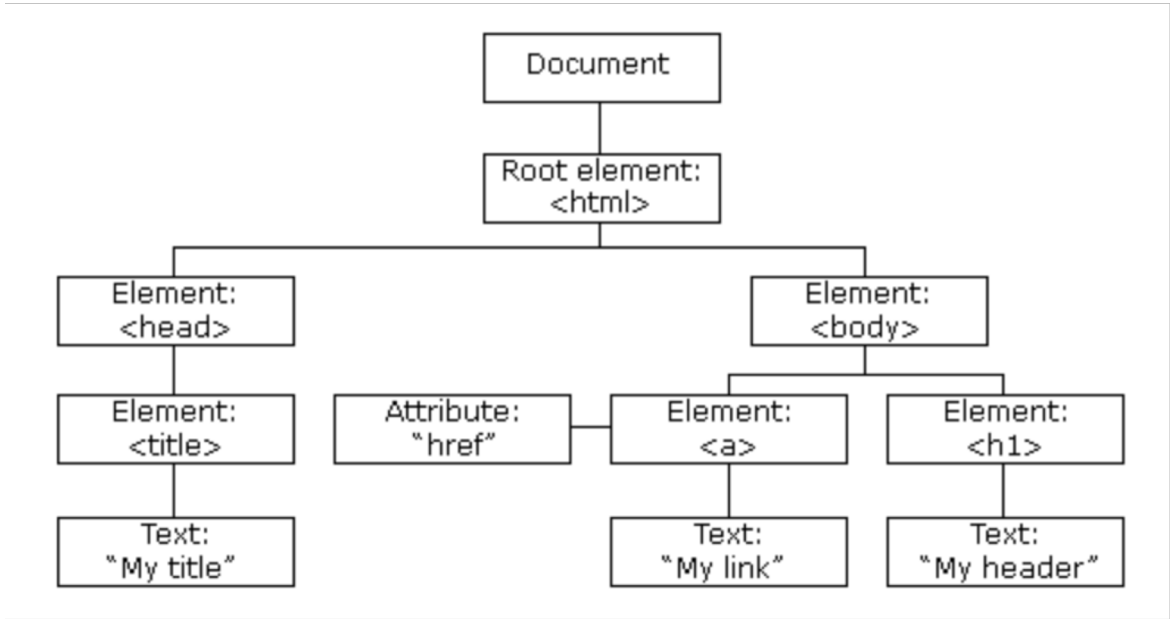
```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
```

TRY this js library: https://github.com/js-cookie/js-cookie

THE HTML DOM (DOCUMENT OBJECT MODEL)

THE HTML DOM

- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.
- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects**:



WHAT CAN JAVASCRIPT DO WITH THE HTML DOM?

- 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

THE DOM PROGRAMMING INTERFACE

- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as **objects**.
  - 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).

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

- In the example above, getElementById is a **method**, while innerHTML is a **property**.

THE HTML DOM DOCUMENT OBJECT

- 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.
- **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
document.querySelector(*selectors*)	Find an element by css selector
document.querySelectorAll(*selectors*)	Find elements by css selector

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

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

JAVASCRIPT HTML DOM EVENTS

- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- Examples of HTML events:
  - 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

```
<h1 onclick="changeText(this)">
Click on this text!</h1>
```

JAVASCRIPT HTML DOM EVENTLISTENER

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

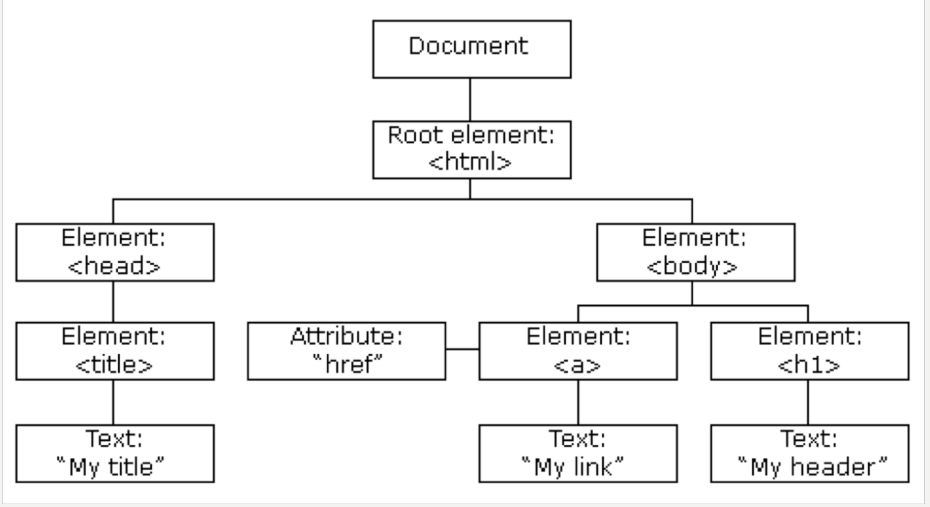
- The addEventListener() method attaches an event handler to the specified element.
- The addEventListener() method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object.

- The removeEventListener() method removes event handlers that have been attached with the addEventListener() method:

```
element.removeEventListener("mousemove", myFunction);
```

JAVASCRIPT HTML DOM NAVIGATION

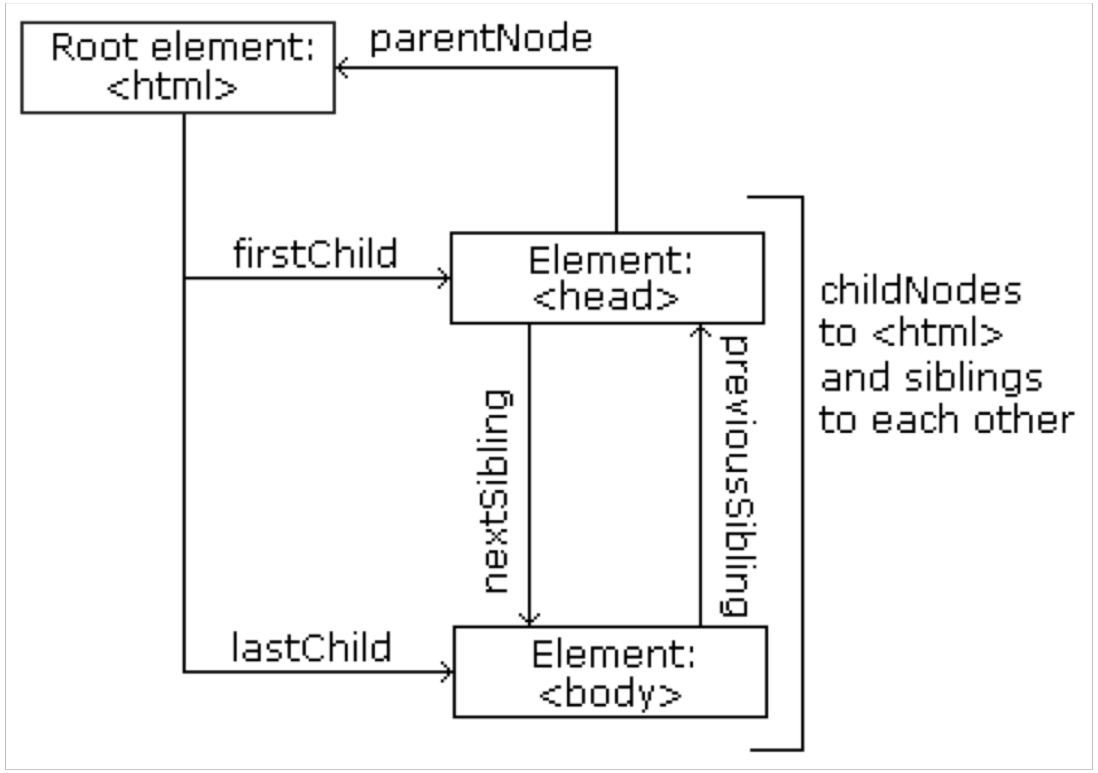
- According to the W3C HTML DOM standard, everything in an HTML document is a node:
  - The entire document is a document node
  - Every HTML element is an element node
  - The text inside HTML elements are text nodes
  - All comments are comment nodes



- 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.

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

```
<html>

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

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

</html>
```

<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>

<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

NAVIGATING BETWEEN NODES

- You can use the following node properties to navigate between nodes with JavaScript:
  - parentNode
  - childNodes[*nodenumber*]
  - firstChild
  - lastChild
  - nextSibling
  - previousSibling

- The nodeName property specifies the name of a node.
- The nodeValue property specifies the value of a node.
- The nodeType property is read only. It returns the type of a node.

LET'S DO SOME EXERCISES

- Complete exercise 6-8