Web Application Development
(LTAT.05.004)

JAVASCRIPT - III

MOHAMAD GHARIB
UNIVERSITY OF TARTU
JavaScript - III
Before we start ...

What do you need to know before starting?
• HTML, CSS and Document Object Model.

What we will cover:
• What JavaScript is,
• JavaScript data types, variables, loops, etc.
• How we can use JavaScript to dynamically create, update our page content, etc.

What can JavaScript do?
JavaScript can dynamically modify/manipulate an HTML page, validate user input (Form validation), react to events and user input (special effects), basic math calculations, etc.

How this lecture contributes to the course

- HTML
  The structure of the webpage
- CSS
  The styling of the webpage
- JavaScript
  Makes a webpage interactive
Document Object Model (DOM)
The Document Object Model (DOM) is a programming interface for HTML and XML documents.

The DOM represents the page in a structured way that allows programs to select and modify the document structure, style, and content.

The DOM represents the document as nodes and objects.
<!DOCTYPE html>
<html>
    <head>
        <meta ... />
        <title>...</title>
    </head>
    <body>
        <h1>...</h1>
        <p>...<span>...</span>...</p>
        <ul>
            <li>...</li>
            <li>...</li>
            <li>...<span id=x>...</span>...</li>
        </ul>
        <p>...</p>
    </body>
</html>
<!DOCTYPE html>
<html>
<head>
<meta ... />
<title>...</title>
</head>
<body>
<h1>...</h1>
<p>...<span>...</span>...</p>
<ul>
<li>...</li>
<li>...</li>
<li>...<span id= x>...</span>...</li>
</ul>
<p>...</p>
</body>
</html>
What can JavaScript do?

- Changes all HTML elements and attributes in the page;
- Changes all the CSS styles in the page;
- Creates new HTML elements and attributes;
- Removes existing HTML elements and attributes;
- Creates new events in the HTML page.
- Reacts to existing events in the HTML page;
What can JavaScript do?

- **Changes** all HTML elements and attributes in the page;
- **Changes** all the CSS styles in the page;
- **Creates** new HTML elements and attributes;
- **Removes** existing HTML elements and attributes;
- **Creates** new events in the HTML page;
- **Reacts** to existing events in the HTML page.

To do that, we need to learn the following:

- DOM nodes **inter-relationships**;
- DOM data types;
- DOM methods and properties;
- Locating DOM elements;
- Accessing, modifying and removing DOM elements/CSS properties;
- Creating, appending, and removing DOM elements;
- DOM traversing;
- DOM events and event listeners.
The nodes in the DOM 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 any number of **children**.
- **Siblings** are nodes with the same parent.
DOM Data Types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>represents the root document object itself.</td>
</tr>
<tr>
<td>Node</td>
<td>every object within the document is a node (e.g., an element node, a text node or attribute node).</td>
</tr>
<tr>
<td>Element</td>
<td>represents, in simple terms, an HTML element.</td>
</tr>
<tr>
<td>Attribute</td>
<td>represents, in simple terms, an attribute of an HTML element.</td>
</tr>
<tr>
<td>NodeList/HTMLColl</td>
<td>represents an array of elements, which can be accessed by index. Ex. returned by the method document.getElementsByTagName().</td>
</tr>
<tr>
<td>NamedNodeMap</td>
<td>represents a collection of objects (e.g., Attribute). a namedNodeMap is like an array, but the items are not ordered and accessed by name or index.</td>
</tr>
</tbody>
</table>
DOM properties and Methods

DOM methods are actions you can perform (on an elements). E.g., `getElementById()`

DOM properties are values (of an elements) that you can set, get or change. E.g., `innerHTML`

```javascript
let element = document.getElementById("demo");

element.innerHTML = "Hello World!";
```
Locating DOM Elements

There are several ways to locate DOM elements:

- Finding HTML elements by **id**.
- Finding HTML elements by **tag name**.
- Finding HTML elements by **class name**.
- Finding HTML elements by **CSS selectors**.
Locating DOM Elements

There are several ways to locate DOM elements:

• Finding HTML elements by id.
• Finding HTML elements by tag name.
• Finding HTML elements by class name.
• Finding HTML elements by CSS selectors.

```html
<p> This is a Paragraph <span id="span1"> This is a Span within a Paragraph </span></p>

let idEle = document.getElementById("span1");
// retrieve the element with the specified id
console.log(idEle);

<span id="span1"> This is a Span within a Paragraph </span>
```
Locating DOM Elements

There are several ways to locate DOM elements:

- Finding HTML elements by **id**.
- Finding HTML elements by **tag name**.
- Finding HTML elements by **class name**.
- Finding HTML elements by **CSS selectors**.

```javascript
let tagEles =
document.getElementsByTagName("p");
// retrieve all elements with the specified
tagName (e.g., <p>) as an array
console.log(tagEles);
console.log(tagEles[0]);

HTMLCollection(n)
<p> ... </p>
```

<p> This is a Paragraph <span id="span1">
This is a Span within a Paragraph </span></p>
There are several ways to locate DOM elements:

- Finding HTML elements by **id**.
- Finding HTML elements by **tag name**.
- Finding HTML elements by **class name**.
- Finding HTML elements by **CSS selectors**.

```javascript
let tagEles = document.getElementsByTagName("class1");
// retrieve all elements with the specified class name (<class1>) as an HTMLcollection/NodeList
console.log(tagEles);
console.log(tagEles[0]);
```

```html
<li class="class1">This is a list item 1 </li>
<li class="class1">... </li>
```
There are several ways to locate DOM elements:

- Finding HTML elements by `id`.
- Finding HTML elements by `tag name`.
- Finding HTML elements by `class name`.
- Finding HTML elements by `CSS selectors`.

```javascript
let cssEle = document.querySelector(".class1");
console.log(cssEle);
// retrieve the first element with the specified css selector
// .querySelectorAll retrieves all elements, which returns NodeList(n)

<li class="class1">This is a list item 1 </li>

<li class="class1"> ... </li>
```
There are several ways to locate DOM elements:

- Finding HTML elements by id.
- Finding HTML elements by tag name.
- Finding HTML elements by class name.
- Finding HTML elements by CSS selectors.

```javascript
let cssEleName = document.querySelector('p[name]');
console.log(cssEleName);
// retrieve the first element with the specified attribute
// .querySelectorAll retrieves all elements
```


```html
<p class="class1" name=para2"> This is a second Paragraph</p>
```
There are several ways to locate DOM elements:

- Finding HTML elements by id.
- Finding HTML elements by tag name.
- Finding HTML elements by class name.
- Finding HTML elements by CSS selectors.

```html
<p class="class1" name="para2"> This is a second Paragraph</p>

let cssEleNameVal = document.querySelector('p[name="para2"]');
console.log(cssEleNameVal);
// retrieve the first element with the specified attribute and specified value
// .querySelectorAll retrieves all elements

<p class="class1" name="para2"> This is a second Paragraph</p>
```
Accessing Element’s Attributes

DOM elements are located to access and modify their “attributes” values.

Accessing elements and their attributes can be done in several ways:

- `textContent`
- `innerHTML`
- `id`
- `className`
- `getAttribute("AttName")`

```javascript
<p id="p2" class="class2">...
  <span id="span1">... </span>
</p>

let idEleP = document.getElementById("p2");
console.log(idEleP.textContent);
// This is a Paragraph  This is a Span within a Paragraph
console.log(idEleP.innerHTML);
// ...<span id="span1"> ... </span>
console.log(idEleP.id); // p2
console.log(idEleP.className); // class2
console.log(idEleP.getAttribute("class"));
//class2 - getAttribute will return the value of the specified attribute
```
Accessing Element’s Attributes

After locating an element, we can access its attributes in several ways:

- `.attributes`
- `.hasAttributes()`
- `.hasAttribute(Att)`

```html
<p id="p2" class="class2">...</p>
<span id="span1">....</span></p>
```

```javascript
let idEleP = document.getElementById("p2");

const attributes = idEleP.attributes;
// returns NamedNodeMap{} of all attributes
console.log(attributes);

// .hasAttributes() - no parameters are required
console.log(idEleP.hasAttributes() ? 'have attributes' : 'does not have attributes');

console.log(idEleP.getAttribute("class") ? 'has a class attribute' : 'does not has a class attribute');
```
Modifying Element's Attributes

Modifying element’s attributes can be done in two ways:

1. change an attribute directly
2. change an attribute through the setAttribute method

```html
<p id="p2" class="class2">...</p>
<span id="span1"> ... </span>
```

```javascript
let idEleP = document.getElementById("p2");

console.log(idEleP.textContent);
// This is a Paragraph  This is a Span within a Paragraph

// Change an attribute directly
console.log(idEleP.className="class3");
//class3

//change an attribute through the setAttribute method
idEleP.setAttribute("class", "class4")
console.log(idEleP.className); //class4
```
The `Element.classList` is a read-only property that returns a live `DOMTokenList` collection of the class attributes of the element.

```html
<p id="p2" class="class2">...</p>
<span id="span1">... </span>
</p>

```javascript
let idEleP = document.getElementById("p2");

// return true or false depending on whether the element has the class
console.log(idEleP.classList.contains("show"));

// return the class on the specified index
console.log(idEleP.classList.item("0"));

// toggle the existence of the class
idEleP.classList.toggle("show");
```
Set/remove Element’s CSS Property

We can **set CSS properties** for an element individually, or by groups, using

```javascript
let idEleP = document.getElementById("p2");

// set a CSS property
idEleP.style.color = "red";

// set property using .setProperty()
idEleP.style.setProperty("font-size", "20px");

// set CSS properties
idEleP.style.cssText = "font-whight:bold; color:red;"; // input as a text

// remove a property
idEleP.style.removeProperty("color");
```

We can **remove** element’s CSS property, using

```javascript
ele.style.removeProperty("Att")
```
Creating and Appending Elements

To add a new element to the DOM, we must create the element (element node) first, and then append it to an existing element.

```
.createElement("ele")
.createAttribute("Att")
.createTextNode("Txt")
.createComment("Txt")
```

```
<p id="p2" class="class2">...
<span id="span1">... </span></p>
```

```
let newElement = document.createElement("p");
// creates an element of the specified type
let newAttr = document.createAttribute("name");
// creates a new attribute node, and returns it
let newText = document.createTextNode("New P element");
// creates a text Node
let newComment = document.createComment("Just a new P element");
// creates a new comment node, and returns it.
```
Creating and **Appending** Elements

To add a **new element** to the DOM, we must **create** the element (element node) first, and then **append** it to an existing element.

```javascript
let newElement = document.createElement("p");
newElement.appendChild(newText);
// Append textNode to an element
newElement.appendChild(newComment);
// Append a comment to an element
document.body.appendChild(newElement);
// Append the element to the body of the document
otherEle.appendChild(newElement);
// Append the element to the another element
newElement.removeChild()
// to remove an element
```

We can **remove** an element using:

```javascript
.remove(ele)
```
Creating and **Appending** Elements

Beside `.appendChild`, there are other methods that allow us to control where we can append/add an element:

- **Append(ele(s) | ..)**
- **Prepend()**
- **Before()**
- **After()**

```javascript
let lil = document.getElementById("li1");
// locate an li in the HTML document
let createNewLi = document.createElement("li");
// creates a new li element

lil.append(createNewLi);
// inserts a set of Node objects or string objects after the last child of the Ele.

lil.prepend(createNewLi);
// inserts a set of Node objects or string objects before the first child of the Ele.

lil.before(createNewLi);
// inserts a set of Node in the children list of this Ele's parent, just before this Ele.

lil.after(createNewLi);
// inserts a set of Node in the children list of this Ele's parent, just before this Ele.
```
Creating and **Appending Elements**

`.append()` and `.prepend()` adds the **new elements** as **child elements** to the target element.

`.after()` and `.before()` adds the **new elements** as **sibling elements** to the target element.
### .appendChild() vs .append()

<table>
<thead>
<tr>
<th>.append()</th>
<th>.appendChild()</th>
</tr>
</thead>
<tbody>
<tr>
<td>• allows you to add <strong>multiple items</strong>;</td>
<td>• allows <strong>only a single item</strong>;</td>
</tr>
<tr>
<td>• accepts <strong>Node objects and strings</strong>;</td>
<td>• accepts only <strong>Node objects</strong>;</td>
</tr>
<tr>
<td>• does not have a <strong>return value</strong>;</td>
<td>• <strong>returns</strong> the appended Node object;</td>
</tr>
</tbody>
</table>
Dealing with Children

There are several properties to deal with children:

- `.children`
- `.childNode`
- `.firstChild`
- `.lastChild`
- `.firstElementChild`
- `.lastElementChild`

```javascript
// newElement is the parent element
console.log(newElement.children);
// returns only element nodes as HTMLCollection even it is one element
console.log(newElement.childNodes);
// returns a NodeList object that contains a collection (list) of the element's child nodes.
console.log(newElement.firstChild);
// returns the first element's child nodes
console.log(newElement.lastChild);
// returns the last element's child nodes
console.log(newElement.firstElementChild);
// returns the first elements' child elements
console.log(newElement.lastElementChild);
// returns the last element's child elements
```
DOM Traversing

Node properties to navigate among nodes:

• `parentNode`
• `childNodes[nodenum]`
• `firstChild`
• `lastChild`
• `nextSibling`
• `previousSibling`
DOM Traversing

Node properties to navigate among nodes:

- `parentNode`
- `childNodes[nodenum]`
- `firstChild`
- `lastChild`
- `nextSibling`
- `previousSibling`

```javascript
let li1 = document.getElementById("li1");
// li1 is the parent element
console.log(li1);
console.log(li1.nextSibling);
console.log(li1.previousSibling);
console.log(li1.nextElementSibling);
console.log(li1.previousElementSibling);
console.log(li1.parentElement);
```
DOM Events

Events are actions or occurrences that happen in the system of concern.

The system will fire a signal of when an event occurs, and also provide a mechanism by which some kind of action can be automatically taken when the event occurs.
DOM Events

Events are actions or occurrences that happen in the system of concern.

The system will fire a signal of when an event occurs, and also provide a mechanism by which some kind of action can be automatically taken when the event occurs.

Events examples:
Clicking the mouse over a certain element or hovering the cursor over a certain element.
The user pressing a key on the keyboard.
The user resizing or closing the browser window.
A web page finished loading.
A form being submitted.
An error occurring.
…
Adding an Event

We can register different event handlers on elements in an HTML document.

Events are normally used in combination with functions, and the function will be executed when the event occurs (e.g., a user clicks a button, mouse leave, mouse over).

```html
<button type="button" id="btn"> Click Me! </button>
```

```javascript
let evBtn = document.getElementById("btn");
evBtn.onclick = function() {
  console.log("button is clicked");
document.getElementById('btn').textContent = "button is clicked";
};
```

> button is clicked
// the textContent of the button change to "button is clicked"

A list of HTML DOM events: https://www.w3schools.com/jsref/dom_obj_event.asp
Adding an Event Listener

The `addEventListener()` method attaches an event handler to the specified element. You can add many event handlers to the same element. When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, and make it easier to control how the event reacts to bubbling.

```html
<button type="button" id="btn2">Click Me!</button>

function randomColorfun(ev) {
  if (ev.target.id === 'btn2') {
    let randomColor = Math.floor(Math.random() * 16777215).toString(16);
    document.body.style.backgroundColor = randomColor;
  }
  btn2.addEventListener('click', randomColorfun);
}
```

// changes the background color of the document
Removing an Event Listener

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

```html
<button type="button" id="btn3"> Click Me! </button>
```

```javascript
btn3.addEventListener('click', function(ev) {
    if (ev.target.id === 'btn3') {
        btn2.removeEventListener('click', randomColorFun);
    }
}, false);
//false - Remove the handler from bubbling.
```

// changes the background color of the document
Bubbling and Capturing

When an event is **fired** on an **element** that has parent elements, browsers run two different phases — the **capturing phase** and the **bubbling phase**.

**In the capturing phase:**
The browser checks to see if the element's outermost ancestor (\texttt{<html>}) has an onclick event handler registered on it in the **capturing phase**, and runs it if so.

Then, it moves on to the next element inside <html> and does the same thing, then the next one, and so on until it reaches the element that was actually clicked on.

**In the bubbling phase,**
The browser checks to see if the element that was actually clicked on has an onclick event handler registered on it in the **bubbling phase**, and runs it if so.

Then, it moves on to the next immediate ancestor element and does the same thing, then the next one, and so on until it reaches the <html> element.
The standard event object has a function stopPropagation(), which when invoked on a handler event, the event does not bubble any further up the chain, so no more handlers will be invoked.

```javascript
let btnPro = document.getElementById('btnPro');
let divBtnPro = document.getElementById('divBtnPro');

btnPro.addEventListener('click', function(e) {
    console.log('The button was clicked!');
e.stopPropagation();
});
divBtnPro.addEventListener('click', function() {
    console.log('The div was clicked!');
});
```
Preventing default behavior

Several HTML elements are implemented with some default behaviours, which might not be desirable in some cases. For example submitting a form content even if it was not valid.

```html
<form id="myForm" action="/lec6.html">
  <hr/>
  <label> Name: </label>
  <input type="text" id="nameForm"/>
  <br>
  <button id="myButton" type="submit">
    Submit form
  </button>
  <hr/>
</form>

let name = document.getElementById('nameForm');

document.getElementById("myForm").addEventListener("submit", function(e) {
  if (name.value === '') {
    console.log("prevent submitting");
    e.preventDefault();
  }
});
```
Browser Object Model (BOM)
The **window** object is supported by all browsers. It represents the **browser's window**.

All global JavaScript objects, functions, and variables automatically become members of the window object.

```javascript
window.innerHeight
window.innerWidth
// return the inner height/ width of the browser window (in pixels)

// The window.screen object can be written without the window prefix.
screen.width
screen.height
// returns the width/ height of the visitor's screen in pixels.

screen.availWidth
screen.availHeight
// return width/Height of the user's screen excluding the interface features like the Windows taskbar.
```

- `screen.pixelDepth`
- `screen.colorDepth`

// It returns the color resolution in bits per pixel. It is an alias to the colorDepth property: 1 (1 bit per pixel), 4, 24, 32, or 48.
The window object is supported by all browsers. It represents the browser's window.

**Notes on resize:**

You cannot resize a window that was not created by `window.open`.

You must open the created window with **resizable feature**.

You cannot resize a window when there are more than one tab in it.

```javascript
myWin = window.open("https://courses.cs.ut.ee/2022/WAD/fall", "myWindowName", "resizable");
//open window

myWin.resizeTo(500, 500);
//resize window to 500x500

myWin.resizeBy(-100, -100);
//make window smaller relatively by 100px

myWin.moveTo(50, 50);
//move window to specified x,y location
```
The Window Object – Location

The **location object** contains information about the current URL.

The **location object** is a property of the **window object**.

- `window.location.href` // `http://....e6-js-3.html`
- `window.location.hostname` // `127.0.0.1`
- `window.location.pathname` // `/lecture6-js-3.html`
- `window.location.protocol` // `http:`
- `window.location.port` // `5500`
The **history object** contains the URLs visited by the user (in the browser window).

The **history object** is a property of the **window object**.

```javascript
history.length // The history.length returns the number of URLs in the history stack.

// To navigate to a URL in the history, you use the back(), forward(), and go() methods.
history.back();
history.forward();
history.go();
// positive numbers -> forward
// Negative numbers -> backwards
```
Popup Boxes

JavaScript has three kinds of popup boxes:

- Alert box
- Confirm box
- Prompt box
Popup Boxes

Alert box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

```javascript
window.alert("Hello");
```
Popup Boxes

Confirm box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

```javascript
let confirmMsg = window.confirm("Do You agree?");
if (confirmMsg === true) {
  console.log("OK pressed");
} else {
  console.log("Cancel pressed");
}
```
Popup Boxes

A prompt box

is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

let promptMsg = prompt("Please enter your age");
Timing Events

The **window object** allows execution of code at specified **time intervals**.

These **time intervals** are called **timing events**.
Timing Events

The **window object** allows execution of code at specified **time intervals**.

These **time intervals** are called **timing events**.

```javascript
setTimeout(function() {
    console.log("set Timeout Msg");
}, 3000);
```

**setTimeout(function, milliseconds)**

Executes a function, after waiting a specified number of milliseconds.
Timing Events

The **window object** allows execution of code at specified **time intervals**.

These **time intervals** are called **timing events**.

```
function msg() {
  console.log(`Message to console`);
}
setInterval(msg, 1000);
```
Timing Events

The **window object** allows execution of code at specified **time intervals**.

These **time intervals** are called **timing events**.

The **clearInterval()** method stops the **executions** of the function specified in the **setInterval()** method.

The **clearInterval()** method uses the variable returned from **setInterval()**.

```javascript
function msg() {
  console.log(`Message to console`);
}

setInterval(msg, 1000);

let counter = setInterval(msg, 1000);
clearInterval(counter);
```
Extra reading/exercises

- **W3Schools** is a free educational website for learning to code online. With their "Try it Yourself" editor, you can edit the HTML code and view the result immediately: https://www.w3schools.com/js/

- **Edabit** (https://edabit.com/challenges) is a coding platform with over 10,000 interactive coding challenges ranging from Very Easy to Expert.

- **LeetCode** (https://edabit.com/challenges) is also a coding platform with interactive coding challenges, but its challenges are more complex than Edabit.
Thank You
for your attention

Mohamad Gharib
mohamad.gharib@ut.ee

unitartu
tartuuniversity