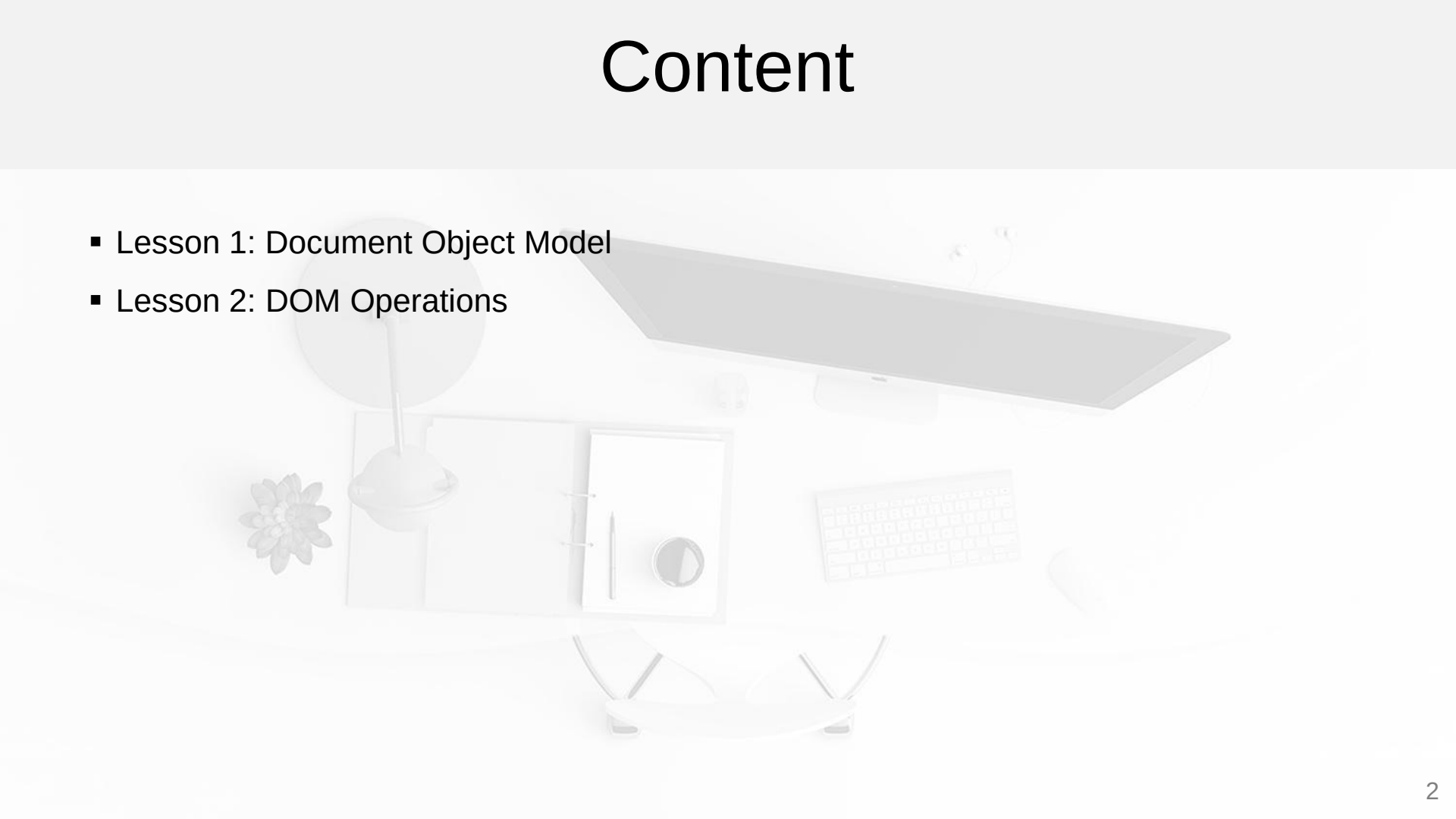
Web Programming

Lecturer: Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

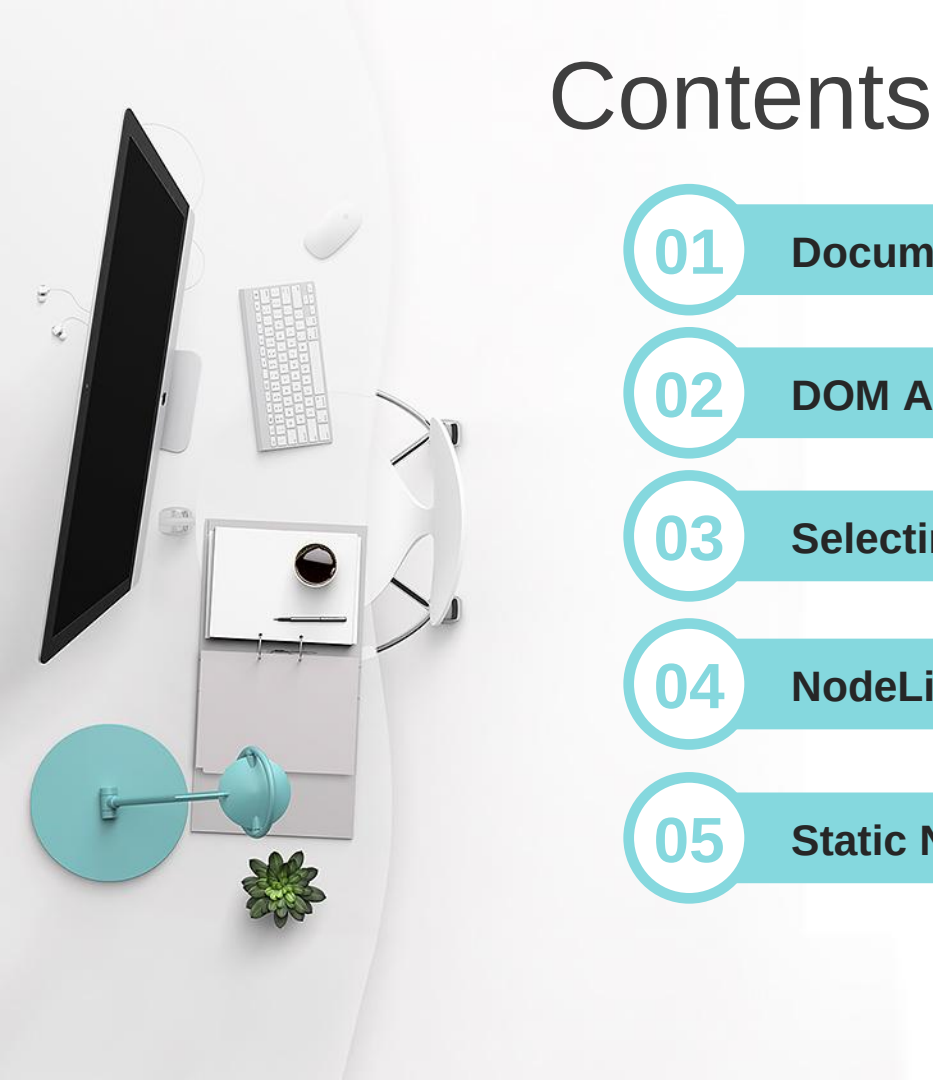
Content

- Lesson 1: Document Object Model
- Lesson 2: DOM Operations



Document Object Model

The True power of dynamic web pages



Contents

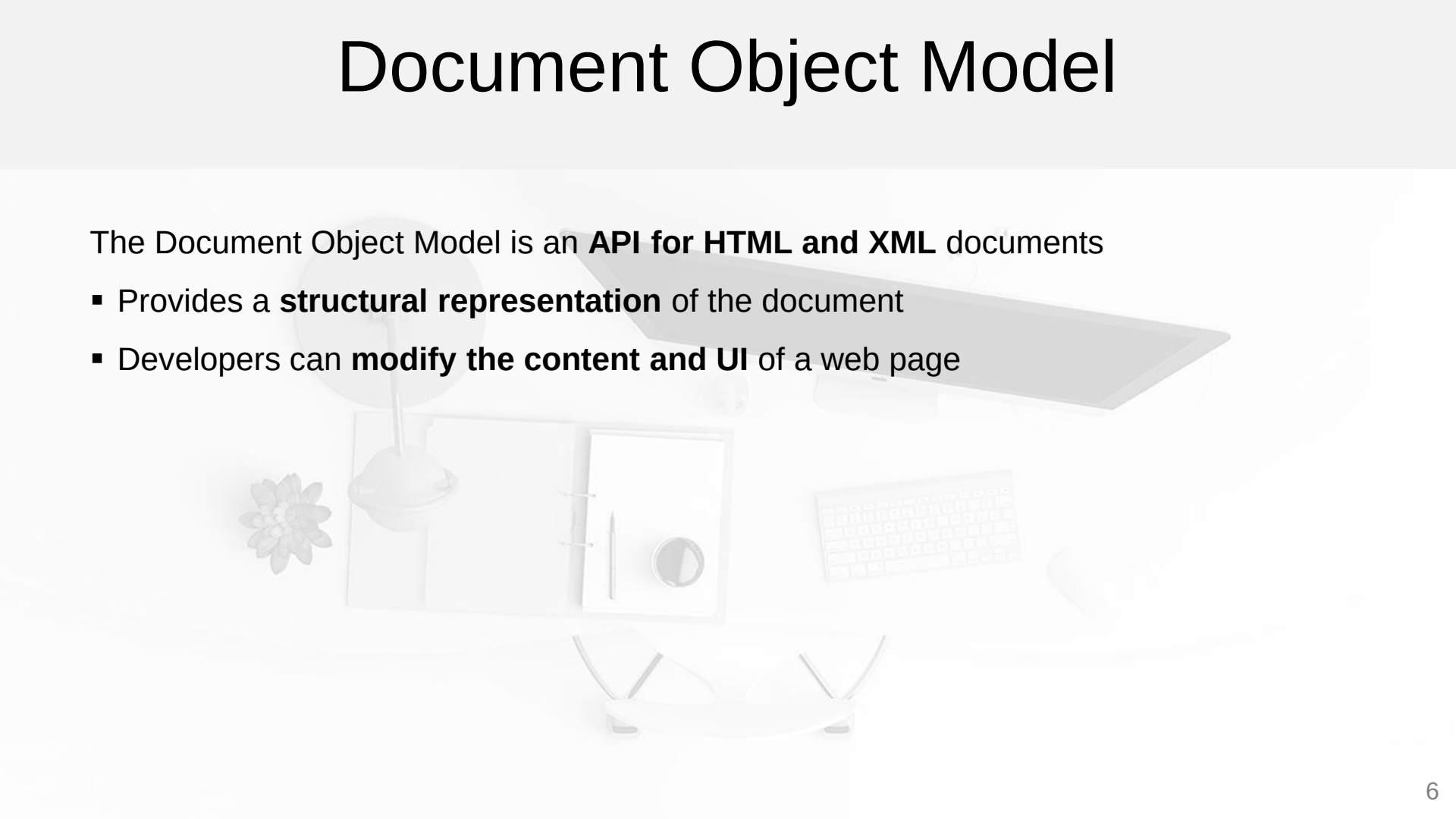
1. Document Object Model (DOM)

The way to access HTML with JavaScript

Document Object Model

The Document Object Model is an **API for HTML and XML** documents

- Provides a **structural representation** of the document
- Developers can **modify the content and UI** of a web page



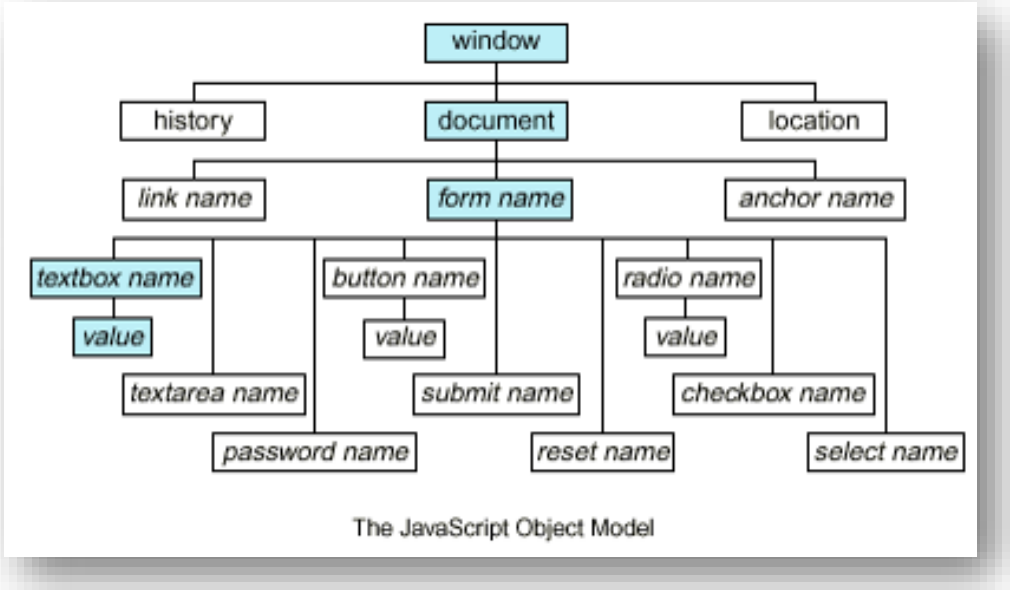
Document Object Model

- The DOM **consists** of **objects to manipulate** a web page
  - All the **properties**, **methods** and **events** are **organized into objects**
  - Those objects are accessible through **programming languages** and **scripts**

- How to use the DOM of an HTML page?

  Write JavaScript to interact with the DOM

  JavaScript uses the DOM API (native implementation for each browser)



The JavaScript Object Model

2. DOM API

Using the DOM with JavaScript

The DOM API

- The DOM API consists of **objects and methods** to interact with the HTML page
  - Can **add** or **remove** HTML **elements**
  - Can **apply styles** dynamically
  - Can **add** and **remove** HTML **attributes**

- DOM introduces objects that represent HTML elements and their properties
  - **document.documentElement** is <**html**>
  - **document.body** is the **body** of the page

DOM Objects

- Each of the HTML elements has corresponding DOM object type
  - **HTMLLIElement** represents <**li**>
  - **HTMLAudioElement** represents <**audio**>

- Each of these objects has the appropriate properties
  - **HTMLAnchorElement** has **href** property
  - **HTMLImageElement** has **src** property
  - **HTMLInputElement** has **value** property

DOM Objects

- The **document** object is a special object

  It represents the **entry point for the DOM API**

- DOM elements **have properties** that **match attributes of HTML elements**

  id, className, style, onclick, etc.

DOM Objects

Common Properties

Property	Description
className	Sets or returns the value of the class attribute of an element
classList	Returns the CSS classnames of an element. It is read-only, but you can use the methods listed below, to add, toggle or remove CSS classes from the list.
innerHTML	Sets or returns the content of an element, without element
outerHTML	Sets or returns the content of an element (including the start tag and the end tag)
innerText/textContent	Sets or returns the text content of a node and its descendants
tagName	Returns the tag name of an element (in UPPERCASE)
scrollTop	Sets or returns the number of pixels an element’s content is scrolled vertically

```
<!DOCTYPE html>
▼<html lang="bg" style>
  ▶<head>…</head>
  ▼<body>
    ▼<div id="Wrapper">
      ▶<header id="MainHeader">…</header>
      ▶<nav class="kendo-style-black">…</nav>
      ▼<div id="MainContainer">
        ▼<div id="ImportantMessages">
          ▶<div class="importantMessageWarning">…</div>
          </div>
        ▶<section id="MainContent">…</section>
        </div>
      ▶<footer id="MainFooter">…</footer>
      </div>
    </body>
  </html>
```

3. Selecting DOM Elements

Selecting HTML Elements

HTML elements can be found and cached into variables using the DOM API

- **Select single element**

```
var header = document.getElementById('header');
var nav = document.querySelector('#main-nav');
```

- **Select a collection of elements**

```
var inputs = document.getElementsByTagName('li');
var radiosGroup = document.getElementsByName('genders[]');
var header = document.querySelectorAll('#main-nav li');
```

- **Using predefined collections of elements**

```
var links = document.links;
var forms = document.forms;
```

Using getElementsBy…() Methods

DOM API contains methods for selecting elements based on some characteristic

- **getElementById**(id)

  Returns a **single element** or **null**

  var header = document.getElementById(‘header’);

- **getElementsByClassName**(className)

  Returns a **collection of elements**

  var posts = document.getElementsByClassName(‘post-item’);

Using getElementsBy…() Methods

DOM API contains methods for selecting elements based on some characteristic

- **getElementsByTagName**(tagName)

  Returns a **collection of elements**

  var sidebars = document.getElementsByTagName('sidebar');

- **getElementsByName**(name)

  Returns a **collection of elements**

  var gendersGroup = document.getElementsByName('genders[]');

Using querySelector() Methods

- The DOM API has methods that use **CSS-like selectors** to find and select HTML elements
  - **querySelector**(selector) returns the **left most element** that matches the selector
  - **querySelectorAll**(selector) returns **a collection of all elements** that match the selector

- Both methods take as a string parameter

  The CSS selector of the element

```
//the element with id="header"
var header = document.querySelector('#header');
//li elements contained in element with id=main-nav
var navItems = document.querySelectorAll('#main-nav li');
```

Selecting Nested Elements

The DOM API can be used to select elements that are inside other elements

Select all divs that are inside an element with id “wrapper”

```
var wrapper = document.getElementById('wrapper');
var divsInWrapper = wrapper.getElementsByTagName('div');
```

Selecting an element from within a form

Syntax:

- document.forms[“**formName**”][“**fieldName**”]
- document.forms.**formName**[.elements.**fieldName**[.HTMLAttribute]]

4. NodeLists

The result of querying DOM elements

NodeLists

A NodeList is a **collection returned** by the DOM selector methods:

- getElementsByTagName(tagName)
- getElementsByName(name)
- getElementsByClassName(className)
- querySelectorAll(selector)

```
var divs = document.getElementsByTagName('div');
var queryDivs = document.querySelectorAll('div');
for(var i=0; i < divs.length; i++) {
    //do stuff with divs[i]…
}
```

NodeLists

NodeList looks **like an array**, but is not

- It's an object with properties similar to array

  Has length and indexer

- You should use **for-of loop** to traverse

  Traversing an array with **for-in loop works unexpected**:

```
for (var i in divs) {
    console.log('divs[' + i + '] = ' + divs[i]);
}
//divs[0] = ...
//divs[1] = ...
//divs[length] = ...
//divs[item] = ...
```

5. Static NodeList and Live NodeList

What is that? What is the difference?

Static NodeList and Live NodeList

- There are two kinds of NodeLists
  - getElementsBy…() return a **LiveNodeList**
  - querySelectorAll() returns a **StaticNodeList**

- **Live lists keep track** of the selected elements
  - Even when changes are made to the DOM
  - While static list contains the elements as they were at the execution of the method

- Keep in mind that a **LiveNodeList** is **slower** than a regular array

  Need to cache its length for better performance

Live NodeLists

Example

```
const parent = document.getElementById('parent');
let child_nodes = parent.childNodes;
console.log(child_nodes.length); // let's assume "2"
parent.appendChild(document.createElement('div'));
console.log(child_nodes.length); // outputs "3"
```

DOM Operations

Creating dynamic pages

Contents

1. DOM Elements

What is the DOM built from?

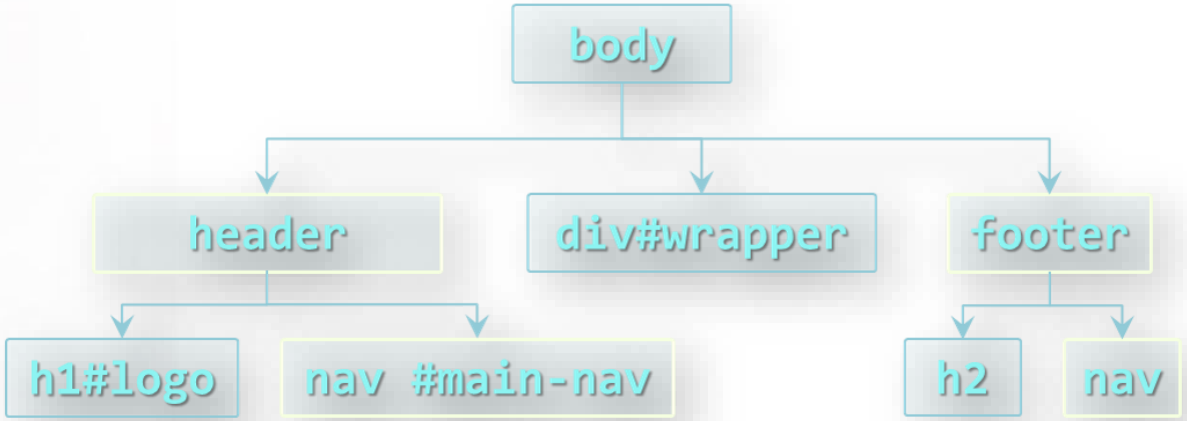
DOM Elements

- A DOM element is a JavaScript object that represents an element from the HTML
  - **Selected** using any of the DOM selectors
  - **Created** dynamically from code

- DOM elements can be changed

  These changes are **immediately applied** to the DOM and the HTML page

```
//changes the content of the div
selectedDiv.innerHTML = "changed";
//changes the background of the div to "#456"
selectedDiv.style.background = "#456";
var div = document.createElement("div");
```



2. Traversing the DOM

Traversing the DOM

- DOM elements have properties about their position in the DOM:
  - Their parent
  - Their children
  - Their Siblings

    Elements immediately before and after the element

- These properties can be used to traverse through the DOM

Traversing the DOM

- **element.parentNode**
  - Returns the **direct parent** of the element
  - The parent of document is null

- **element.children**

  Returns a nodeList of the child nodes (not text (whitespaces) and comment nodes)

Traversing the DOM

Example

Traverse a <UL> with <LI>s:

```
function iterateList(listId){
    var trainersList = document.getElementsById(listId);
    var parent = trainersList.parentNode;
    log("parent of trainers-list: " + parent.nodeName +
        " with id: " + parent.id);
    var children = trainersList.childNodes;
    log("elements in trainers-list: " + children.length);
    log("element in trainers-list");
    for (var i = 0, len = children.length; i < len; i+=1) {
        var subItem = children[i];
        log(subItem.nodeName + " content: " + subItem.innerText);
    }
}
```

Using the Named Elements in DOM Objects

- DOM elements have some properties for special elements around them:
  - **First** and **last** child nodes
  - The element **before/after** the current node

- The named elements are:
  - firstChild and lastChild
  - nextElementSibling
  - previousElementSibling

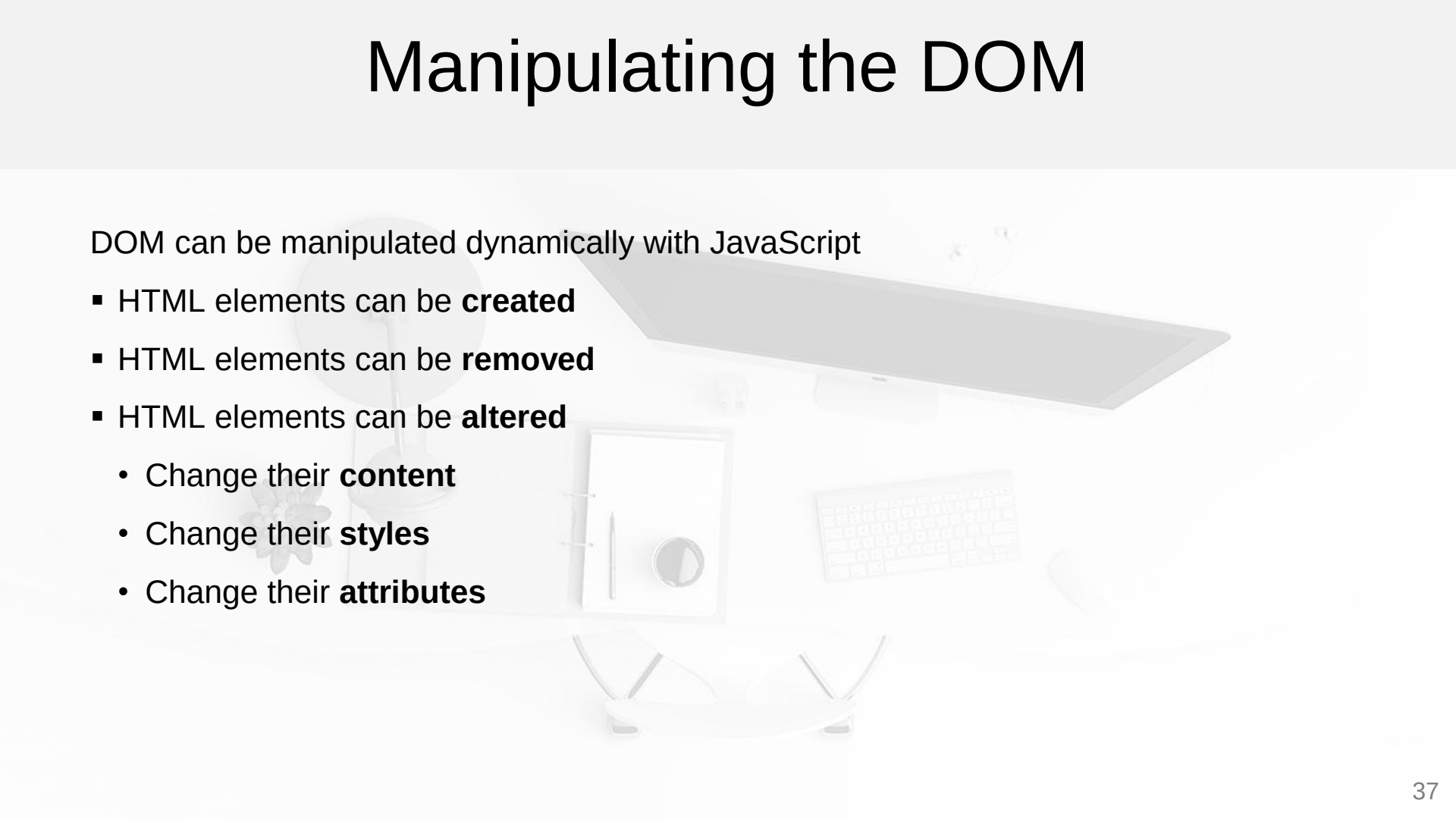
3. Manipulating the DOM

Making a web page dynamic

Manipulating the DOM

DOM can be manipulated dynamically with JavaScript

- HTML elements can be **created**
- HTML elements can be **removed**
- HTML elements can be **altered**
  - Change their **content**
  - Change their **styles**
  - Change their **attributes**



Creating HTML Elements

The document object has a method for creation of HTML elements

- **document.createElement**(elementName)
- Returns an object with the corresponding HTML element type

```
var liElement = document.createElement("li");
console.log(liElement instanceof HTMLLIElement); //true
console.log(liElement instanceof HTMLElement); //true
console.log(liElement instanceof HTMLDivElement); //false
```

Creating HTML Elements

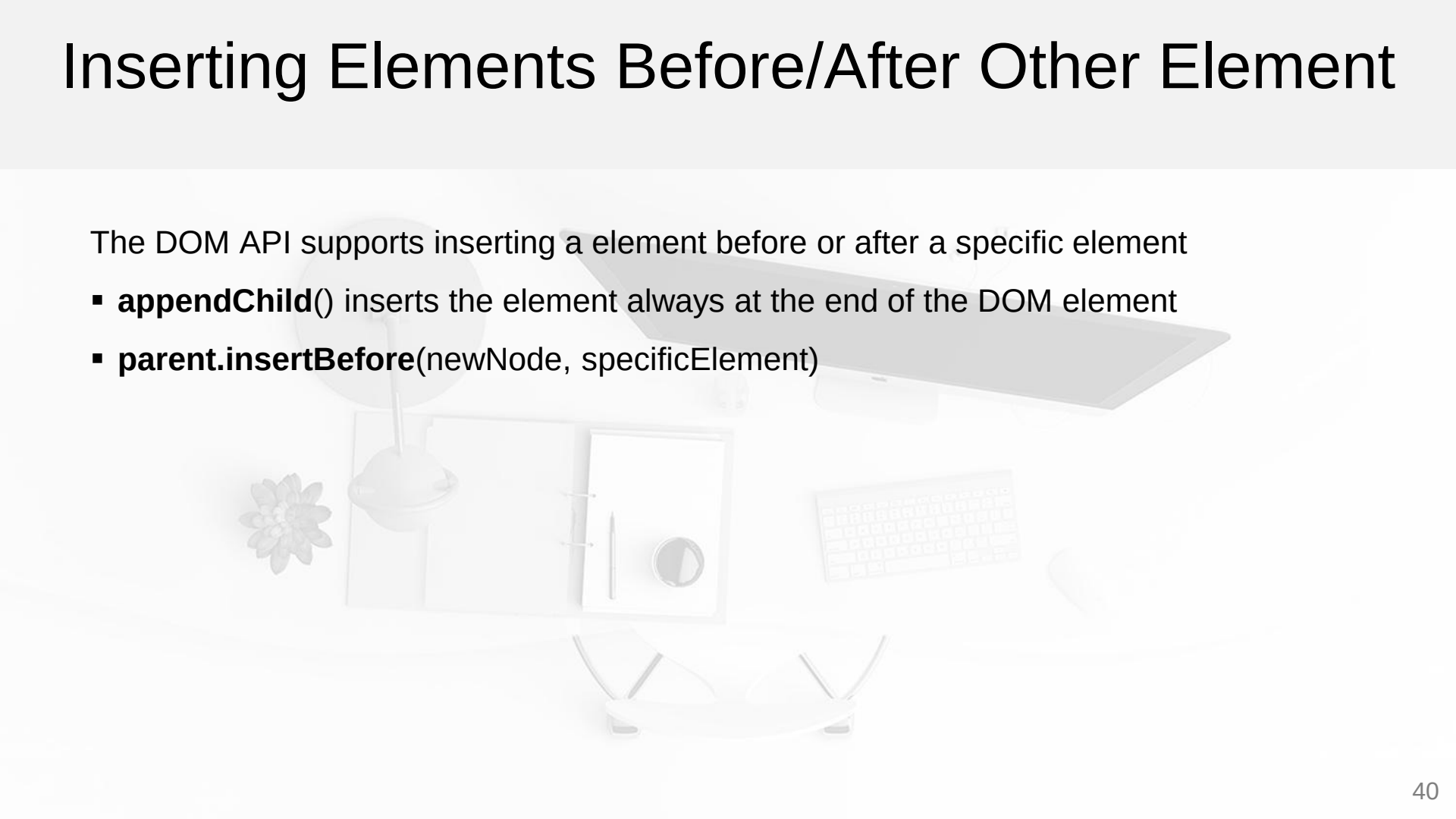
- After an HTML element is created it can be treated as if it was selected from the DOM
- When HTML elements are created dynamically, they are just JavaScript objects
  - They are still not in the DOM (the web page)
  - New HTML elements **must be appended** to DOM

```
var studentsList = document.createElement("ul");
var studentLi = document.createElement("li");
studentsList.appendChild(studentLi);
document.body.appendChild(studentsList);
```

Inserting Elements Before/After Other Element

The DOM API supports inserting a element before or after a specific element

- **appendChild**() inserts the element always at the end of the DOM element
- **parent.insertBefore**(newNode, specificElement)



Removing Elements

Elements can be removed from the DOM

- Using **element.removeChild**(elToRemove)

  Pass the element-to-remove to their parent

- Using **element.remove()**

  Remove the specified element

```
var trainers = document.getElementsByTagName("ul")[0];
var trainer = trainers.getElementsByTagName("li")[0];
trainers.removeChild(trainer);
```

Altering the Elements

- DOM elements can be removed and/or changed

  Both the node's children and the node itself

- With the DOM API each DOM element node can be altered
  - Change its properties
  - Change its appearance

Altering the Elements

Keep in mind that each HTML element is unique in the DOM

If JavaScript changes its appearance or its position, it is still the same element object

```
<div id="f">
    <p id="the-p">text</p>
</div>
<div id="s"></div>
…
var second = document.getElementById("s");
var theP = document.getElementById("the-p");
second.appendChild(theP);
…
//the DOM is:
<div id="f"></div>
<div id="s">
    <p id="the-p">text</p>
</div>
```

Altering the Attributes

- Using **element.removeAttribute**(*attributeName*) to remove the specified attribute from the element

- Using **element.setAttribute(***attributeName*, *attributeValue***)** to add the specified attribute to the element, and gives it the specified value

```
// Add the class attribute
document.getElementsByTagName("H1")[0].setAttribute("class", "democlass");

// Remove the class attribute
document.getElementsByTagName("H1")[0].removeAttribute("class");
```

Altering the Style

The style of each HTML element can be altered using JavaScript

Meaning changing the style attribute

The inline styles, not CSS

```
var div = document.getElementById("content");
div.style.display = "block";
div.style.width = "123px";
```

Do not forget the unit

HTML DOM Style Object

Property	Description	CSS
alignContent	Sets or returns the alignment between the lines inside a flexible container when the items do not use all available space	3
alignItems	Sets or returns the alignment for items inside a flexible container	3
alignSelf	Sets or returns the alignment for selected items inside a flexible container	3
animation	A shorthand property for all the animation properties below, except the animationPlayState property	3
animationDelay	Sets or returns when the animation will start	3
animationDirection	Sets or returns whether or not the animation should play in reverse on alternate cycles	3
animationDuration	Sets or returns how many seconds or milliseconds an animation takes to complete one cycle	3
animationFillMode	Sets or returns what values are applied by the animation outside the time it is executing	3

Source: https://www.w3schools.com/jsref/dom_obj_style.asp

DOM Objects

Common Methods

Method	Description
appendChild()	Adds (appends) a new child node to an element
getAttribute(“attrName”)	Returns the value of an element's attribute
hasAttribute(“attrName”)	Returns true if an element has a given attribute
remove()	Removes an element from the DOM
removeAttribute(“attrName”)	Removes an attribute from an element
setAttribute(“attrName”)	Sets or changes an attribute's value

Link: https://www.w3schools.com/JSREF/dom_obj_all.asp

4. DOM Optimizations

Everybody likes it fast, right?

Appending DOM Elements

- The DOM API provides a method for appending DOM elements to an element

  The **appendChild**() method

- **parentNode.appendChild**(node) appends the DOM element node to the DOM element parentNode

  If parentNode is appended to the DOM, the node is also appended

Optimizing the Appending of Elements

- Appending elements to the DOM is a **very slow operation**
  - When an element is appended to the DOM, the DOM is **rendered anew**
  - All newly created elements must be appended together

- Here comes the **DocumentFragment** element
  - It is a **minimal DOM element**, with no parent
  - It is used to **store ready-to-append** elements and append them at once to the DOM

Optimizing the Appending of Elements

Using **DocumentFragment**

- Append the elements to a DocumentFragment
- Appending DocumentFragment to the DOM appends only its child elements

```
var dFrag = document.createDocumentFragment();
dFrag.appendChild(div);
//appending more elements
document.body.appendChild(dFrag);
```

Faster Creation of Elements

- **Creating a DOM element** is a slow operation
  - Create the element
  - Set its **content**
  - Set its **style**
  - Set its **attributes**

- This is an issue when creating many elements that have a common structure

  Only one or two things are different for all

Faster Creation of Elements

- Creating a **dynamic list** of elements
  - All LI elements have the **same classes**, **styles**, **attributes**
  - Only the **innerHTML** is different
- **DomElement.cloneNode**(true) can be used

  Creates a full copy (deep copy) of the element

```
var clonedNode = someElement.cloneNode(true)
```


Summary


Q&A