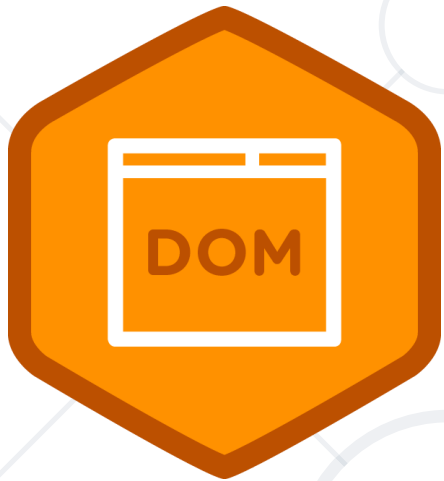


# DOM and Events



**SoftUni Team**  
Technical Trainers



**SoftUni**



**Software University**

<https://softuni.bg>

[sli.do](https://sli.do)

**#QA-FrontEnd**

# Table of Contents

1. Browser API
2. Document Object Model
3. HTML Elements
4. Targeting HTML Elements
5. Using the DOM API
6. DOM Manipulation
7. The DOM Event
8. Event Handling





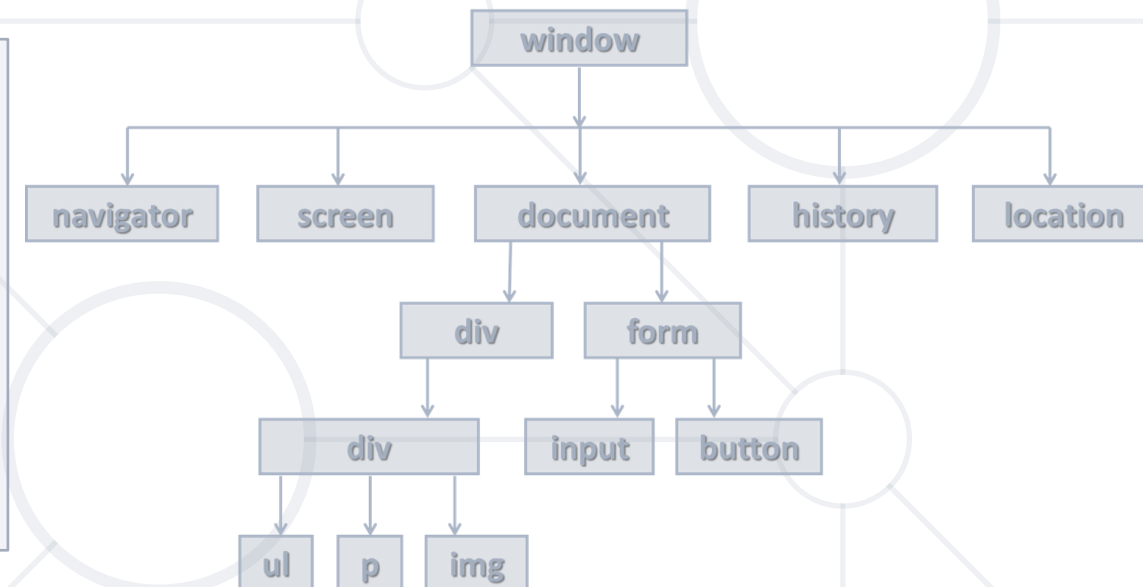
**Browser API**

# Browser Object Model (BOM)

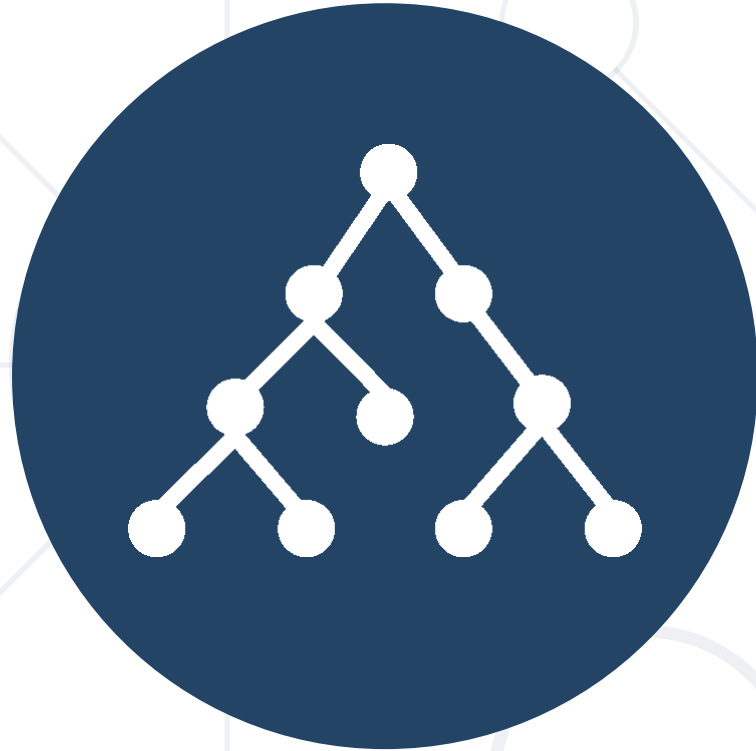
- Browsers expose some objects like **window**, **screen**, **navigator**, **history**, **location**, **document**, ...



```
console.dir(window);  
console.dir(navigator);  
console.dir(screen);  
console.dir(location);  
console.dir(history);  
console.dir(document);
```



- The **global object** in the browser is **window**



**Document Object Model (DOM)**

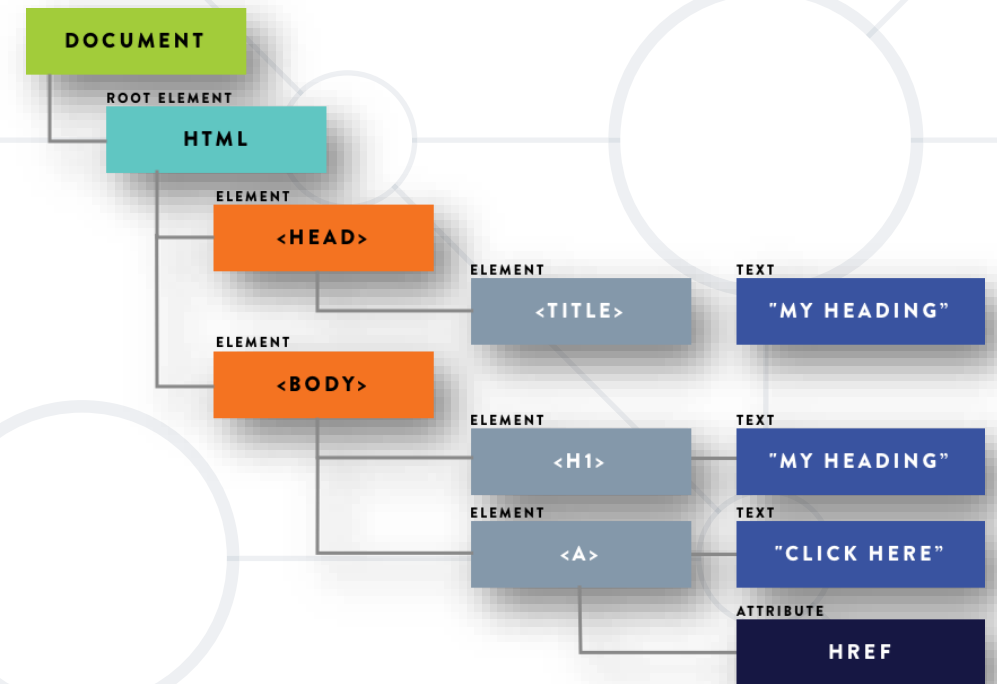
# Document Object Model

- The **DOM** represents the document as **nodes** and **objects**
  - That way, the programming languages **can connect** to the page
- The **HTML DOM** is an **Object Model** for **HTML**
  - It defines
    - HTML elements as **objects**
    - **Properties**
    - **Methods**
    - **Events**



- The browser **parses** HTML and creates a **DOM Tree**

```
<html>
  <head>
    <title>My Heading</title>
  </head>
  <body>
    <h1>My Heading</h1>
    <a href="/about">Click Here</a>
  </body>
</html>
```



- The elements are **nested** in each other and create a **hierarchy**
  - Like the hierarchy of a **street address** – Country, City, Street, etc.



- **DOM Methods**

- **Actions** you can perform on HTML elements
- HTML DOM method is an action you can do, e.g., add or delete an HTML element

- **DOM Properties**

- Values of HTML elements that you can **set** or **change**
- HTML DOM property is a value that you can **get** or **set**, e.g., changing the content of an HTML element

- JavaScript can **interact** with web pages via the **DOM API**
  - Check the **contents** and **structure** of elements on the page
  - Modify element **style** and **properties**
  - Read **user input** and react to **events**
  - **Create** and **remove** elements
- Most actions are performed when an **event** occurs
  - Events are "**fired**" when something of interest happens

- Code can be **executed in the page** in different ways
  - Directly in the **developer console**, during **debugging**
  - As a page **event handler**, e.g., user **clicks** on a button

```
<button onclick="console.log('Hello, DOM!')">Click Me</button> event
```

- Via **inline** script, using **<script>** tags

```
<script>  
  function sum(a, b) {  
    let result = a + b;  
    return result;  
  }  
</script>
```

- By **importing** from external file
  - Most **flexible method**



# HTML Elements

- The DOM Tree is comprised of **HTML elements**
- Elements are **JS objects** with **properties** and **methods**
  - They can be **accessed** and **modified** like regular objects
- To change the contents of the page
  - **Select** an element to obtain a **reference**
  - **Modify** its **properties**

- **Attributes** are **defined** by **HTML**
  - Attributes **initialize** DOM properties
  - Property **values** can **change** via the DOM API
- The HTML attribute and the DOM property are **technically not** the same thing
- Since the outcome is the same, in practice you will almost never encounter a difference!

- The **HTML DOM** allows JavaScript to change the content of **HTML elements**
  - **innerHTML**
  - **textContent**
  - **value**
  - **style**
  - and more

- To access raw HTML

```
element.innerHTML = "<p>Welcome to the DOM</p>";
```

```
<html>
  <head></head>
  ▼ <body>
    <div id="main">This is JavaScript!</div>
  </body>
</html>
```



```
<html>
  <head></head>
  ▼ <body>
    ▼ <div id="main">
      <p>Welcome to the DOM</p>
    </div>
  </body>
</html>
```

- This will be **parsed** – beware of **XSS attacks**!
- Changing **textContent** or **innerHTML** removes all child nodes



- The contents of HTML elements are stored in text nodes
  - To access the contents of an element

```
let text = element.textContent; //This is JavaScript!  
element.textContent = "Welcome to the DOM";
```

```
<html>  
  <head></head>  
  <body>  
    <div id="main">This is JavaScript!</div>  
  </body>  
</html>
```



```
<html>  
  <head></head>  
  <body>  
    <div id="main">Welcome to the DOM</div>  
  </body>  
</html>
```

- If the element has children, returns all text **concatenated**

# Accessing Element Values

- The **values** of input elements are **string properties** on them

```
<html>
  <head></head>
  <body>
    <div id="main">
      <p>Welcome to the DOM</p>
      <input id="num1" type="text">
    </div>
  </body>
</html>
```

```
type: "text"
useMap: ""
validationMessage: ""
▶ validity: ValidityState
value: "56"
valueAsNumber: NaN
▶ webkitEntries: Array[0]
webkitdirectory: false
width: 0
```

```
let num = Number(element.value);
element.value = 56;
```



**Targeting Elements**

# Targeting Elements

- There are a few ways to **find** a certain **HTML element** in the **DOM**
  - By ID → **getElementById()**
  - By class name → **getElementsByClassName()**
  - By tag name → **getElementsByTagName()**
  - By CSS selector → **querySelector()**, **querySelectorAll()**
- These methods return a **reference** to the element, which can be **manipulated** with JavaScript



# Targeting by Tag and Class Names

- The **tag name** specifies the **type** of element – **div**, **p**, **ul**, etc.

```
const elements = document.getElementsByTagName('p');  
// Select all paragraphs on the page
```

- **Class names** are used for **styling** and easier **selection**

```
const elements = document.getElementsByClassName('list');  
// Select all elements having a class named 'list'
```

- Both methods return a live **HTMLCollection**
  - **Even if** only **one** element is selected! This is a **common mistake!**

- **CSS selectors** are strings that follow CSS syntax for matching
- They allow very fast and powerful element matching
  - **"#main"**
    - Returns the element with ID "main"
  - **"#content div"**
    - Selects all **<div>**s inside **#content**
  - **".note, .alert"**
    - All elements with class "note" or "alert"
  - **"input[name='login']"**
    - **<input>** with name "login"

# NodeList vs. HTMLCollection

- Both interfaces are **collections** of **DOM nodes**
- **NodeList** can contain **any** node type, including **text** and **whitespace**
- **HTMLCollection** contains only **Element nodes**
- Both have **iteration** methods, **HTMLCollection** has an extra **namedItem** method
- **HTMLCollection** is **live**, while **NodeList** can be either **live** or **static**



- **NodeList** and **HTMLCollection** are **NOT** arrays but can be **indexed** and **iterated**

```
const elements = document.querySelectorAll('p');  
const first = elements[0];  
// Select the first paragraph on the page  
  
for (let p of elements) { /* ... */ }  
// Iterate over all entries
```

- Both can be **explicitly converted** to an array

```
const elementArray = Array.from(elements);  
const elementArr2 = [...elements]; // Spread syntax
```



- Every DOM Element has a **parent**
  - Parents can be accessed by property **parentElement** or **parentNode**

```
▼ <div>  
  <p>This is a paragraph.</p>  
  <p>This is another paragraph.</p>  
</div>
```

Accessing the  
first child

```
let firstP = document.getElementsByTagName('p')[0];  
console.log(firstP.parentElement);
```

Accessing the  
child's parent

```
▶ <div>...</div>
```

- When some element contains other elements, that means it is **parent** of those elements
  - Those elements are **children** to the **parent**
    - They can be accessed by property **children**

```
▼ <div>  
  <p>This is a paragraph.</p>  
  <p>This is another paragraph.</p>  
</div>
```

```
▼ HTMLCollection(2) [p, p]  
  ► 0: p  
  ► 1: p  
  length: 2
```

```
let pElements = document.getElementsByTagName('div')[0].children;
```

Returns live  
HTMLCollection



# Using the DOM API

Common Techniques and Scenarios

- Page scripts can be **loaded** from an external file

- Use the **src** attribute of the **script element**

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

- **Functions** from script files are in the **global scope**
  - Can be referenced and **executed** from **events** and **inline** scripts
  - **Multiple** script files in a page can see **each other**
- Pay attention to **load order**!

- Content can be **hidden** or **revealed** by changing its **display** style
  - This is a **common technique** to display content dynamically

- To **hide** an element

```
const element = document.getElementById('main');  
element.style.display = 'none';
```

- To **reveal** an element, set **display** to anything that isn't **'none'** (including **empty string**)

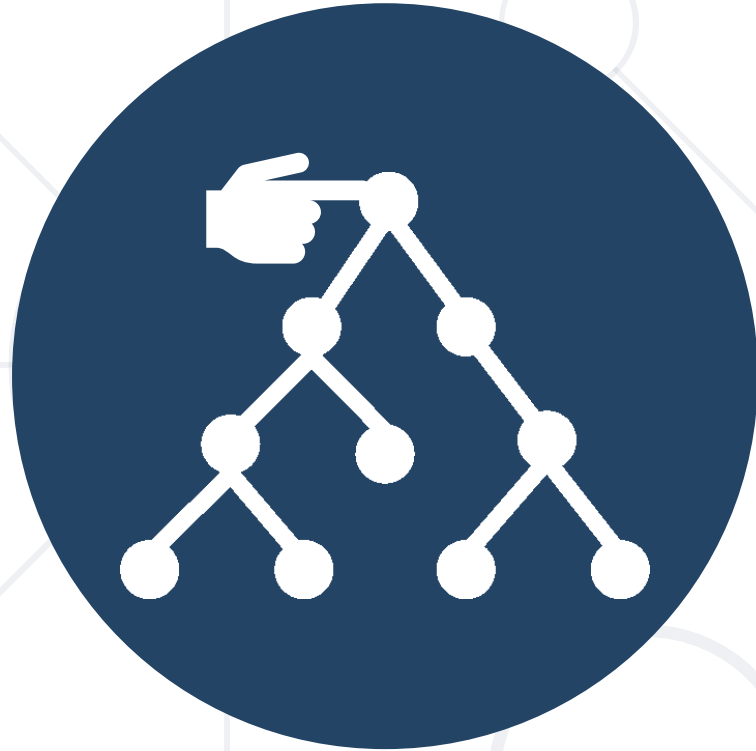
```
element.style.display = ''; // Can be 'inline', 'block', etc.
```

- Sometimes we need to target an element based on its **relation** to other **similar elements**
  - e.g., **row** or **column** in a table, **list item**, etc.
- Can be done either by **index** or with a **CSS selector**

```
const list = document.getElementsByTagName('ul')[0];  
// First <ul> on the page
```

```
const thirdLi = list.getElementsByTagName('li')[2];  
// Third <li> inside the selected <ul>
```

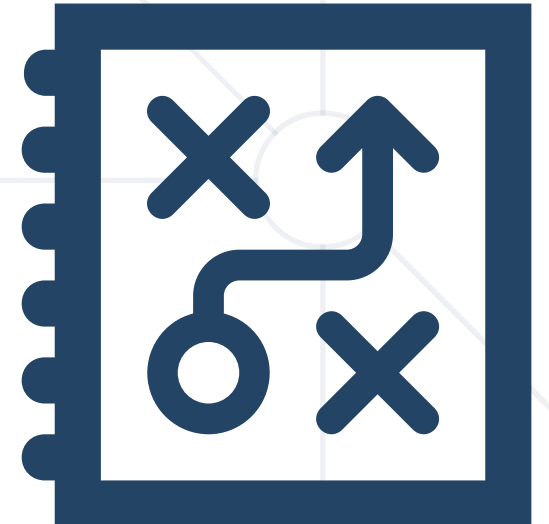
```
const thirdLi = document.querySelector('ul li:nth-child(3)');  
// Third <li> inside the first <ul> on the page
```



# DOM Manipulation

# DOM Manipulations

- We can **create**, **append** and **remove** HTML elements dynamically
  - **appendChild()**
  - **removeChild()**
  - **replaceChild()**





# Creating New DOM Elements

- HTML elements are created with **document.createElement**
  - This is called a **Factory Pattern**
- Variables holding HTML elements are **live**
  - If you **modify** the contents of the variable, the DOM is **updated**
  - If you **insert** it somewhere in the DOM, the original is **moved**
- Text added to **textContent** will be **escaped**
- Text added to **innerHTML** will be **parsed** and turned into actual HTML elements → beware of **XSS attacks!**

- Creating a new DOM element

```
let p = document.createElement("p");  
let li = document.createElement("li");
```

Tag name

- Create a copy / cloning DOM element

```
let li = document.getElementById("my-list");  
let newLi = li.cloneNode(true);
```

- Elements are created **in memory** – they don't exist on the page
- To become visible, they must be **appended** to the DOM tree

- **appendChild**

- Adds a new child, as the **last child**

```
let p = document.createElement("p");  
let li = document.createElement("li");  
li.appendChild(p);
```

- **prepend**

- Adds a new child, as the **first child**

```
let ul = document.getElementById("my-list");  
let li = document.createElement("li");  
ul.prepend(li);
```

# Deleting DOM Elements

```
<ul id="items">  
  <li class="red">Red</li>  
  <li class="blue">Blue</li>  
</ul>
```

```
▼ <body>  
  ▼ <ul id="items">  
    <li class="red">Red</li>  
    <li class="blue">Blue</li>  
  </ul>  
</body>
```

```
let redElements =  
  document.querySelectorAll("#items li.red");  
redElements.forEach(li => {  
  li.parentNode.removeChild(li);  
});
```

```
▼ <body>  
  ▼ <ul id="items">  
    <li class="blue">Blue</li>  
  </ul>  
</body>
```



# **The DOM Event**

## Event Object and Types

# Event Object

- Calls its **associated function**
- Passes a **single argument** to the function – a **reference** to the event object
- Contains **properties** that describe the event
  - Which **element** triggered the event
  - Screen **coordinates** where it occurred
  - What is the **type** of the event
  - And more



# Event Types in DOM API

## ■ Mouse events

click  
mouseover  
mouseout  
mousedown  
mouseup

## ■ Touch events

touchstart  
touchend  
touchmove  
touchcancel

## ■ DOM / UI events

load  
unload  
resize  
dragstart / drop

## ■ Keyboard events

keydown  
KeyPress  
keyup

## ■ Focus events

focus (got focus)  
blur (lost focus)

## ■ Form events

input  
change  
submit  
reset



**Event Handling**



# Event Handler

- Event registration is done by providing a **callback function**
- Three ways to register for an event:
  - With **HTML Attributes**
  - Using **DOM element properties**
  - Using **DOM event handler** – preferred method



```
function handler(event){  
    // this --> object, html reference  
    // event --> object, event configuration  
}
```

- **addEventListener();**

```
htmlRef.addEventListener( 'click' , handler);
```

- **removeEventListener();**

```
htmlRef.removeEventListener( 'click' , handler);
```



- In event handlers, **this** refers to the event **source element**
  - **target** is the element that triggered the event
  - **currentTarget** is the element that the event listener is attached to

```
element.addEventListener("click", function(e) {  
    console.log(this === e.currentTarget); // true  
});
```

- Pay attention when using **object methods** as event listeners!

```
const myObject = {  
  value: 42,  
  handleClick: function () { console.log(this) },  
};  
  
myObject.handleClick(); // { value: 42, handleClick: f}  
const myButton = document.getElementsByTagName("button")[0];  
myButton.addEventListener("click", myObject.handleClick);  
// User clicks the button - this == myButton
```

# Attaching Hover Handler

```
const button = document.getElementsByTagName("button")[0];

button.addEventListener("mouseover", function (e) {
    const buttonElementStyles = e.currentTarget.style;
    buttonElementStyles.backgroundColor = "red";
});

button.addEventListener("mouseout", function (e) {
    const buttonElementStyles = e.currentTarget.style;
    buttonElementStyles.backgroundColor = "blue";
});
```

# Attaching Input Handler

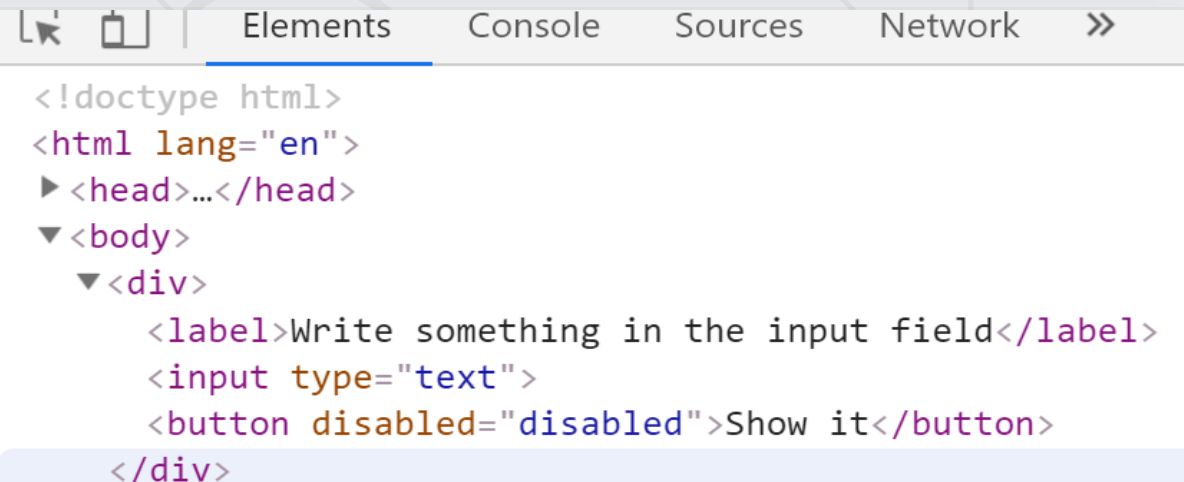
```
const inputField = document.getElementsByTagName('input')[0];
const button = document.getElementsByTagName('button')[0];

inputField.addEventListener('input', function () {
    button.setAttribute('disabled', 'false')
});
```

Write something in the input field

Show it

div 304 x 71.2



```
<!doctype html>
<html lang="en">
  <head>...</head>
  <body>
    <div>
      <label>Write something in the input field</label>
      <input type="text">
      <button disabled="disabled">Show it</button>
    </div>
```

```
const button = document.getElementById('myButton');
function handleClick() {
    alert('Button clicked!');
}

button.addEventListener('click', handleClick);

// Add a timeout to remove the event listener after 5 seconds
setTimeout(function() {
    button.removeEventListener('click', handleClick);
    alert('Event listener removed!');
}, 5000);
```

# Multiple Listeners

- The **addEventListener()** method also allows you to add many listeners to the same element, without overwriting existing ones

```
element.addEventListener("click", myFirstFunction);  
element.addEventListener("click", mySecondFunction);  
element.addEventListener("mouseover", myThirdFunction);  
element.addEventListener("mouseout", myFourthFunction);
```

*Note that you don't use the "on" prefix for the event use "click" instead of "onclick"*





```
const input = document.getElementsByTagName('input')[0];

// First event listener
input.addEventListener('focus', function () {
    console.log('Input focused (First listener)');
});

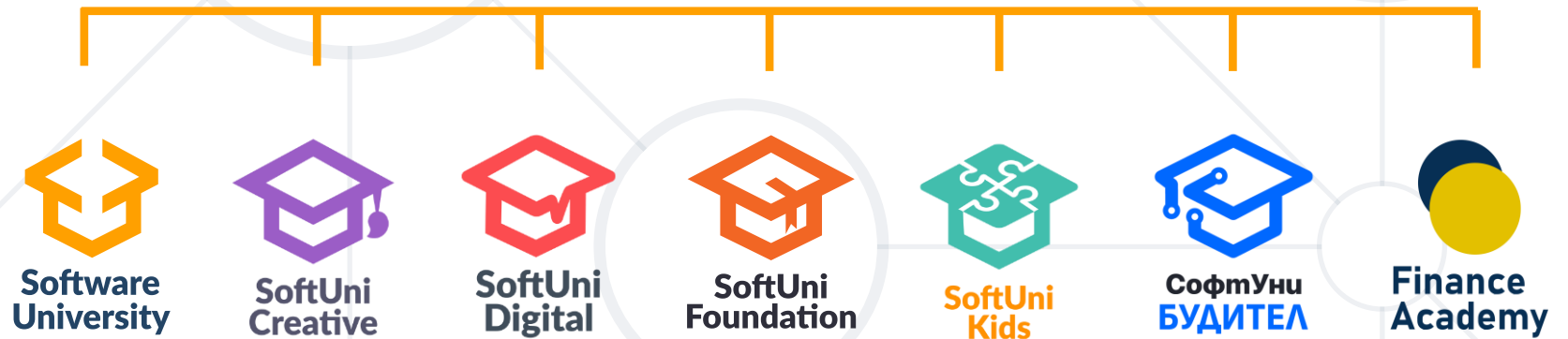
// Second event listener
input.addEventListener('focus', function () {
    console.log('Input focused (Second listener)');
});

// Input focused (First listener)
// Input focused (Second listener)
```

- **BOM** == Browser API
- **DOM** == programming API for HTML documents
  - Selecting DOM Elements by **id**, **class** or **query selectors**
  - DOM **Properties** & HTML **Attributes**
  - **Manipulating** the DOM tree
  - User **interaction** triggers events



# Questions?



# Diamond Partners



THE CROWN IS YOURS



- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [about.softuni.bg](http://about.softuni.bg)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

