Table of Contents

1. The DOM API
2. Targeting & Selecting Elements
3. Creating & Manipulating Elements
4. Handling Events

Have a Question?

Software University

sli.do

#js-front-end

DOM API

Document Object Model

JavaScript in the Browser

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

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

event

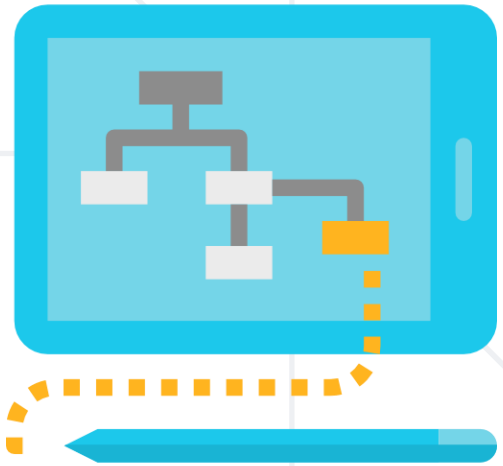
  - Via **inline** script, using **`<script>`** tags

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

  - By **importing** from external file – most **flexible method**

Document Object Model

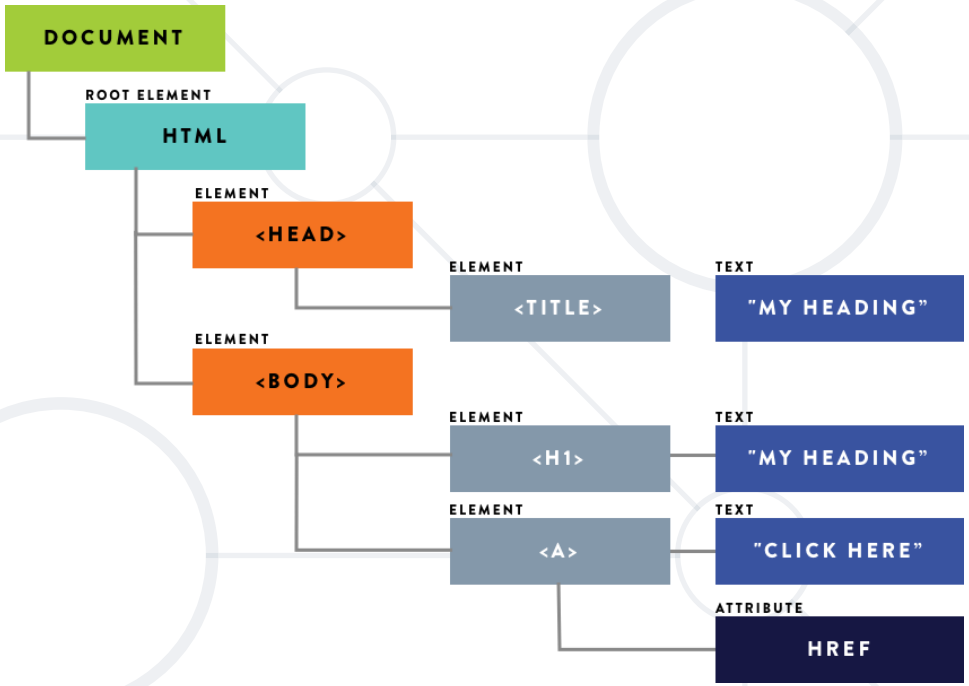
- The **DOM** represents the document as **nodes** and **objects**
  - That way, the programming languages **can connect** to the page
- The **HTML DOM** is an **Object Model** for **HTML**. It defines:
  - HTML elements as **objects**
  - **Properties**
  - **Methods**
  - **Events**



From HTML to DOM Tree

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

```html
<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

- **DOM Methods** - actions you can perform on HTML elements
- **DOM Properties** - values of HTML elements that you can **set** or **change**

Example: DOM Methods

- HTML DOM **method** is an action you can do (like **add** or **delete** an HTML element)

```
<!doctype html>
…<html> == $0
  ▼<head>
     <title>Intro to DOM</title>
   </head>
  ▼<body>
     <h1>Introduction to DOM</h1>
   ▼<ul>
       <li>DOM Methods example</li>
       <li>DOM Properties example</li>
     </ul>
   </body>
 </html>
```

```
>
let h1Element = document.getElementsByTagName('h1')[0];
console.log(h1Element);
<h1>Introduction to DOM</h1>
```

Example: DOM Methods

- HTML DOM **property** is a value that you can **get** or **set** (changing the content of an HTML element)

```
<!doctype html>
…<html> == $0
  ▼<head>
     <title>Intro to DOM</title>
   </head>
  ▼<body>
     <h1>Introduction to DOM</h1>
   ▼<ul>
      <li>DOM Methods example</li>
      <li>DOM Properties example</li>
     </ul>
   </body>
</html>
```

```
let secondLi = document.getElementsByTagName('li')[1];
```

```
secondLi.innerHTML += " - DONE"
```

Introduction to DOM

- DOM Methods example
- DOM Properties example - DONE

Using the DOM API

- 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
- All of this **and more** will be examined in upcoming lessons

Elements and Properties

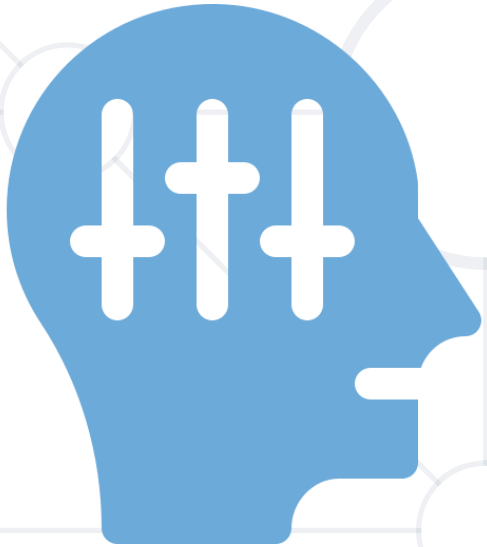
- The DOM Tree is comprised of **HTML elements**
- Elements are **JS objects** with **properties** and **methods**
  - They can be **accessed** and **modified** like regular objects
- To change the contents of the page:
  - **Select** an element to obtain a **reference**
  - **Modify** its **properties**

Attributes and 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!

DOM Manipulations

- The **HTML DOM** allows JavaScript to change the content of **HTML elements**
  - `innerHTML`
  - `textContent`
  - `value`
  - `style`
  - And many others to be discussed in upcoming lessons



Accessing Element HTML

- 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

Accessing Element Text

- 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;
```

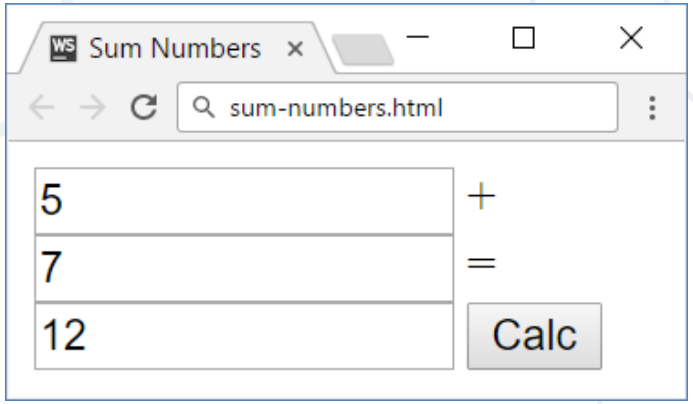
Problem: Sum Numbers

- Write a JS function to sum two numbers (fill the missing code)

```
<input type="text" id="num1" /> +
<input type="text" id="num2" /> =
<input type="text" id="sum" readonly="readonly" />
<input type="button" value="Calc" onclick="calc()" />

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

calc.js

```
function calc() {
  // TODO
}
```



Solution: Sum Numbers

```
function calc() {
  let num1 = document.getElementById('num1').value;
  let num2 = document.getElementById('num2').value;

  let sum = Number(num1) + Number(num2);

  document.getElementById('sum').value = sum;
}
```

Check your solution here: https://judge.softuni.org/Contests/Compete/Index/3794#0

Control Content via Visibility

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

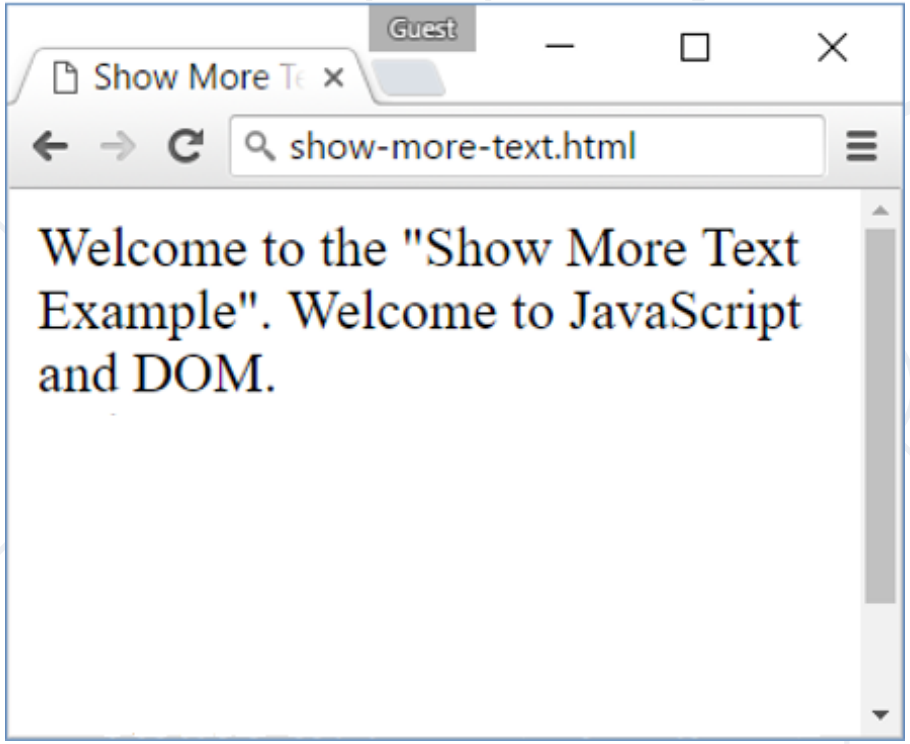
```
const element = document.getElementById('main');
element.style.display = 'none';
```

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

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

Problem: Show More Text

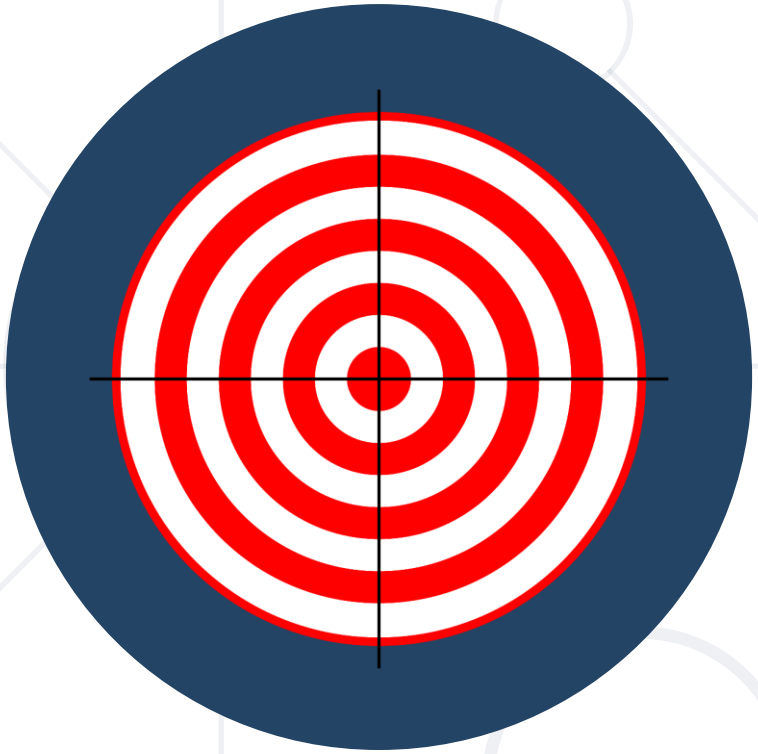
- A HTML page holds a short text + link "***Read more …***"
  - Clicking on the link shows more text and hides the link



Problem: Show More Text – HTML

```
Welcome to the "Show More Text
Example".

<a href="#" id="more" onclick=
"showText()">Read more …</a>

<span id="text" style=
"display:none">Welcome to …</span>

<script>
  function showText() {
    // TODO
  }
</script>
```

- See the DOM tree here: http://software.hixie.ch/utilities/js/live-dom-viewer/?saved=4275



Targeting & Selecting Elements

Id, Class, Tag and Query Selectors

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 ID - Example

- The **ID attribute** must be **unique** on the page

```
const element = document.getElementById('main');
console.log(element);
```

```
<html>
  ▶ <head> ··· </head>
  ▼ <body>
    ▼ <div id="main">
      ▼ <article class="list">
          <p>First</p>
          <p>Second</p>
          <p>Third</p>
        </article>
      </div>
    </body>
</html>
```

```
▼ div#main
    accessKey: ""
    accessKeyLabel: ""
    align: ""
    assignedSlot: null
  ▶ attributes: NamedNodeMap [ id="main" ]
```

Targeting by Tag and Class Names – Example

- 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**

Query Selectors - Example

- Select the **first matching** element

```
const mainDiv = document.querySelector('#main');
// Select the element with ID 'main'

const element = document.querySelector('p');
// Select the first paragraph on the page
```

- Select **all** matching elements
  - Returns a **static NodeList**

```
const elements = document.querySelectorAll('article.list');
// Select all <article> elements having a class named 'list'
```

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**

Iterating Element Collections

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

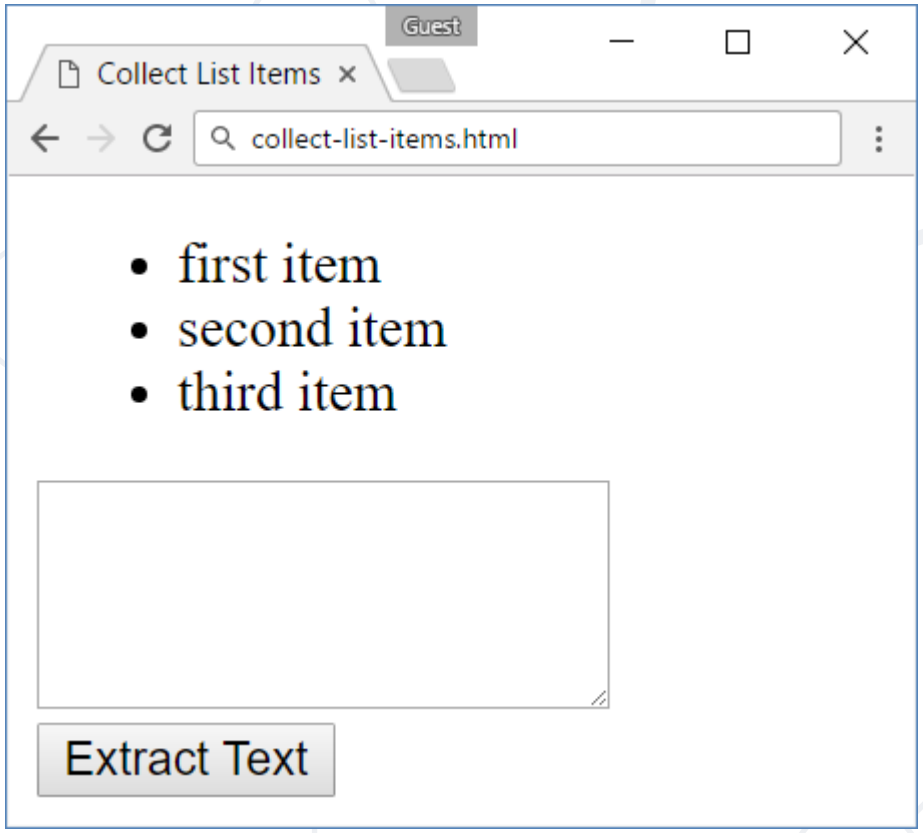
```
const elements = document.querySelectorAll('p');
const first = elements[0];
// Select the first paragraph on the page

for (let p of elements) { /* … */ }
// Iterate over all entries
```

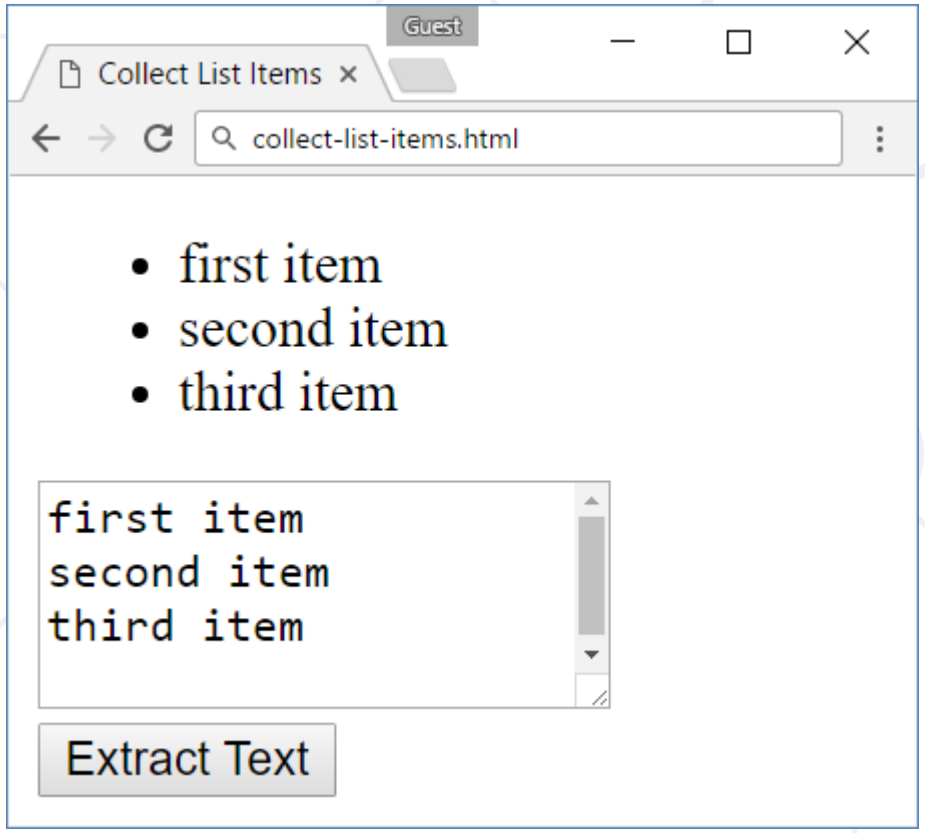
- Both can be **explicitly converted** to an array

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

Problem: Collect List Items

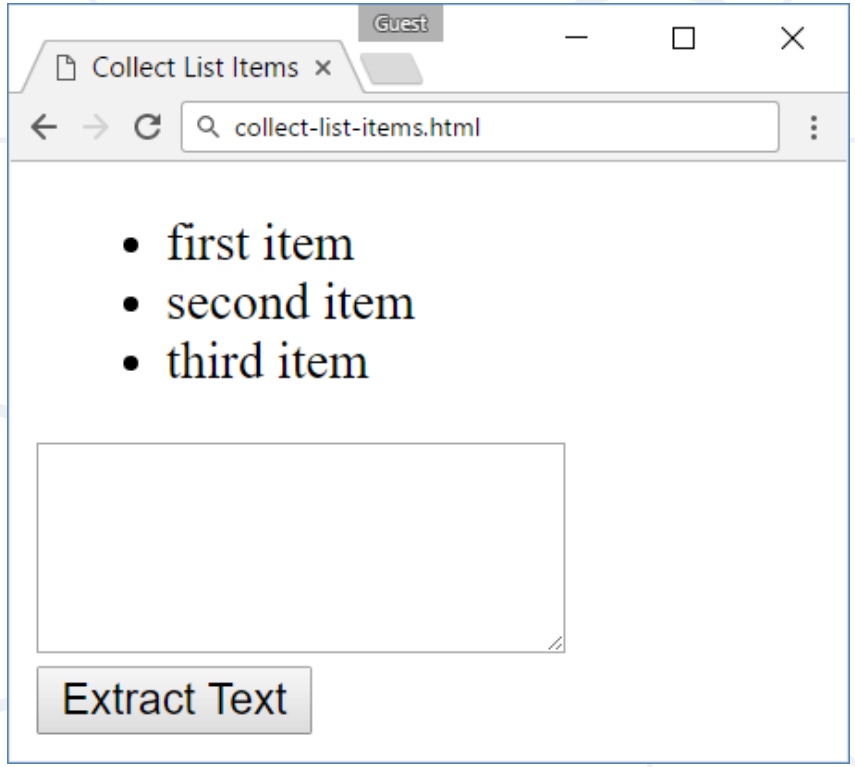
- Collect the **list items** from given HTML list and append their **text** to given **text area**





Problem: Collect List Items – HTML

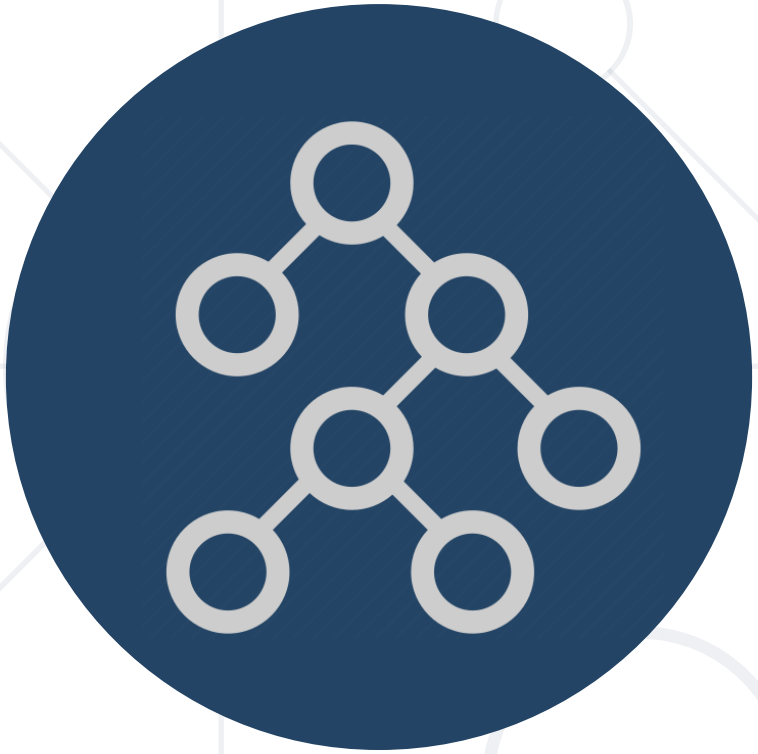
```html
<ul id="items">
  <li>first item</li>
  <li>second item</li>
  <li>third item</li>
</ul>

<textarea id="result">
</textarea>

<br>

<button onclick="extractText()">
Extract Text</button>
```



Solution: Collect List Items

```
function extractText() {
  let itemNodes =
    document.querySelectorAll("ul#items li");
  let textarea =
    document.querySelector("#result");
  for (let node of itemNodes) {
    textarea.value += node.textContent + "\n";
  }
}
```

Check your solution here: https://judge.softuni.org/Contests/Compete/Index/3794#2



DOM Manipulation

Parents & Children. CRUD Operations.

Parents and Child Elements

- 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>
```

Parents and Child Elements

- When some element contains other elements, that means he is **parent** of those elements
- They 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>
```

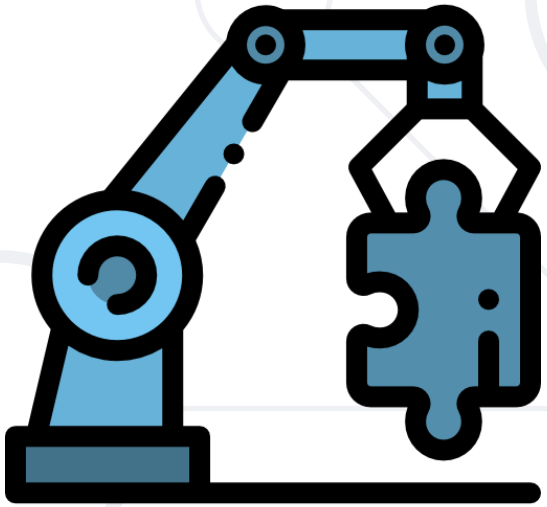
```
▼HTMLCollection(2) [p, p]
  ▶0: p
  ▶1: p
   length: 2
```

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

Returns live HTMLCollection

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`**
- 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 DOM Elements

- 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

Manipulating Node Hierarchy

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

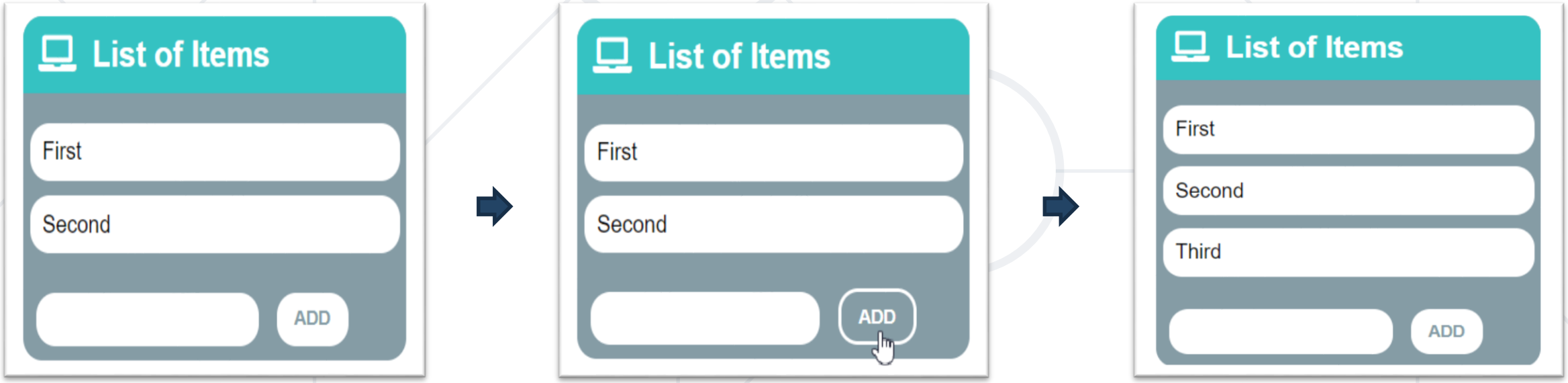
```
let p = document.createElement("p");
let li = document.createElement("li");
li.appendChild(p);
```

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

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

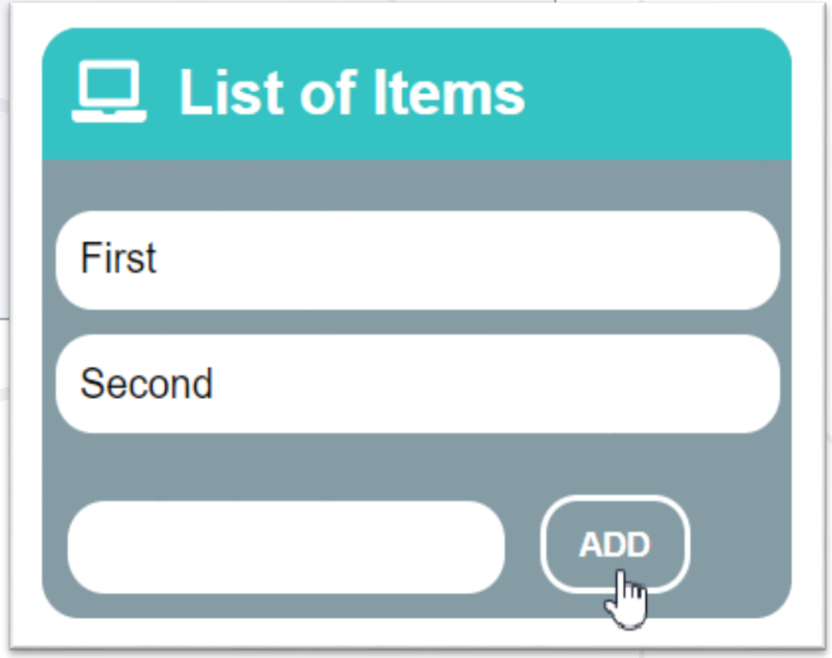
Problem: List of Items

- Create a HTML page holding a **list of items** + **text box** + **button** for adding more items to the list
  - Write a function to **append** the specified text to the list



Problem: List of Items – HTML

```
<h1>List of Items</h1>
<ul id="items"><li>First</li><li>Second</li></ul>
<input type="text" id="newItemText" />
<input type="button" value="Add" onclick="addItem()">
<script>
function addItem() {
  // TODO: Add new item to the list
}
</script>
```



Solution: List of Items

```
function addItem() {
  let text = document.getElementById('newItemText').value;
  let li = document.createElement("li");
  li.appendChild(document.createTextNode(text));
  document.getElementById("items").appendChild(li);
      //clearing the input:
  document.getElementById('newItemText').value = '';
}
```

Check your solution here: https://judge.softuni.org/Contests/Compete/Index/3794#3

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>
```

Problem: Delete from Table

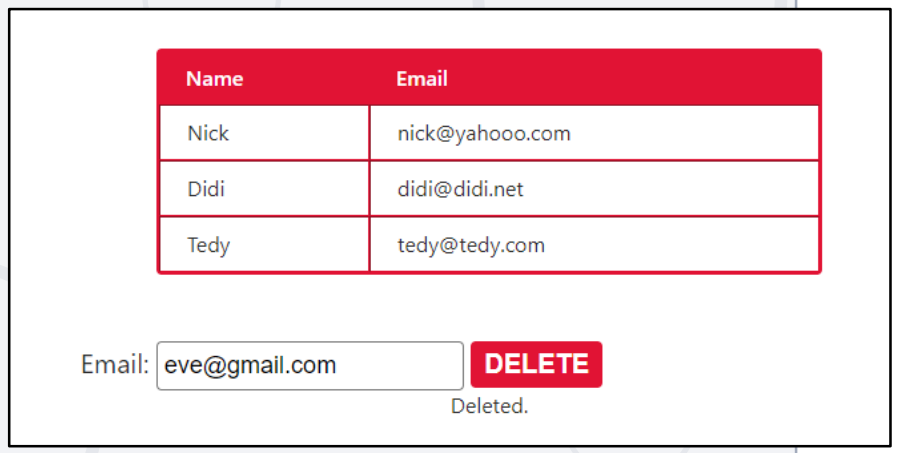
```
<table border="1" id="customers">
 <tr><th>Name</th><th>Email</th></tr>
 <tr><td>Eve</td><td>eve@gmail.com</td></tr>
 <tr><td>Nick</td><td>nick@yahooo.com</td></tr>
 <tr><td>Didi</td><td>didi@didi.net</td></tr>
 <tr><td>Tedy</td><td>tedy@tedy.com</td></tr>
</table>
Email: <input type="text" name="email" />
<button onclick="deleteByEmail()">Delete</button>
<div id="result" />
```

Name	Email
Eve	eve@gmail.com
Nick	nick@yahooo.com
Didi	didi@didi.net
Tedy	tedy@tedy.com

Email: DELETE

Solution: Delete from Table

```
function deleteByEmail() {
  let email = document.getElementsByName("email")[0].value;
  let secondColumn = document.querySelectorAll(
    "#customers tr td:nth-child(2)");
  for (let td of secondColumn)
    if (td.textContent == email) {
      let row = td.parentNode;
      row.parentNode.removeChild(row);
      document.getElementById('result').
        textContent = "Deleted.";
      return;
    }
  document.getElementById('result').textContent = "Not found.";
}
```



Check your solution here: https://judge.softuni.org/Contests/Compete/Index/3794#4

DOM Events

Event Types, Handling Events, Delegation

Event Types in DOM API

- **Mouse** events

```
click
mouseover
mousmouse
outedown
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 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
}
```

Event Listener

- addEventListener();

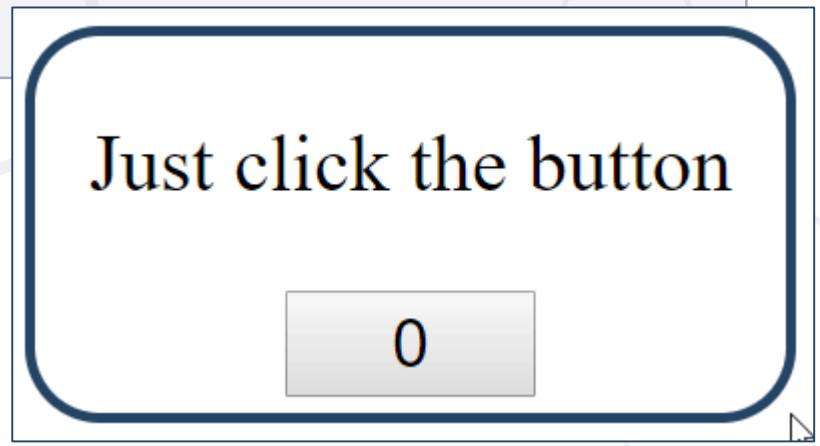
```
htmlRef.addEventListener('click', handler , false);
```

- removeEventListener();

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

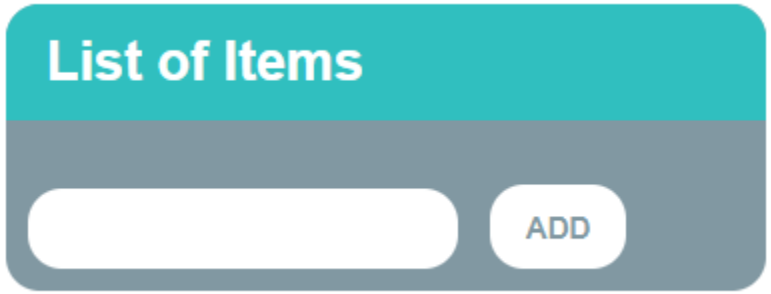
Attaching Click Handler

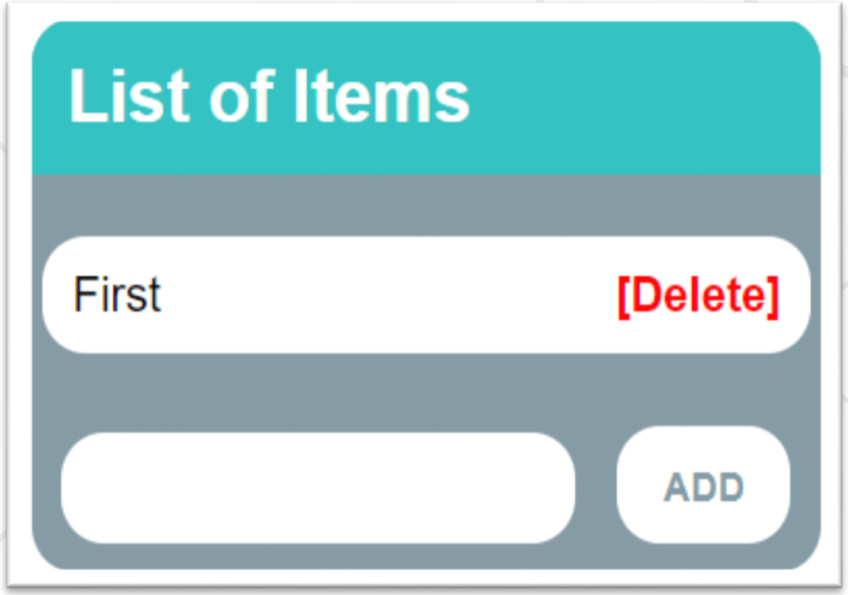
```
const button = document.getElementsByTagName('button')[0];

button.addEventListener('click', clickMe);

function clickMe(e) {
  const target = e.currentTarget;
  const targetText = target.textContent;
  target.textContent = Number(targetText) + 1;
}
```



Problem: Add / Delete Items

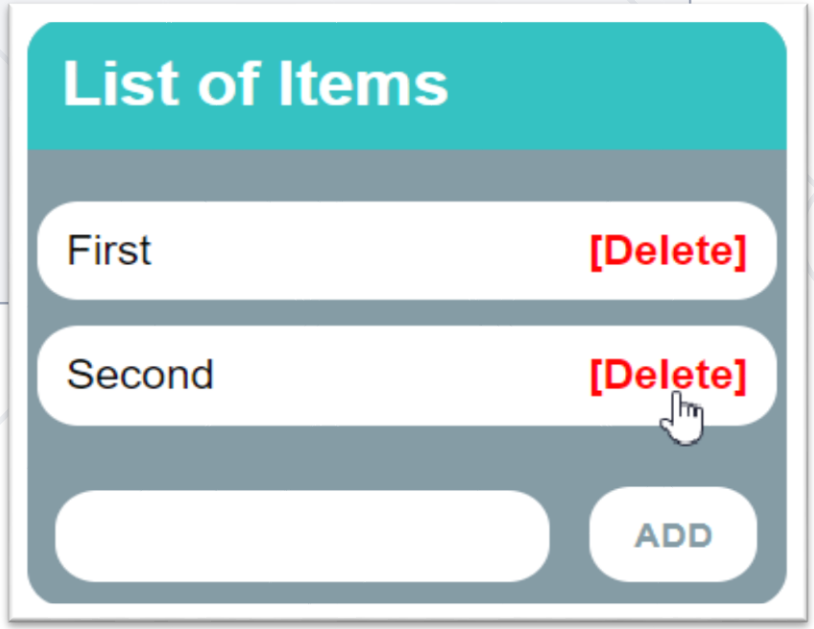
- Extend the previous problem
  - Implement `[Delete]` action as link after each list item





Problem: Add / Delete Items – HTML

```
<h1>List of Items</h1>
<ul id="items"></ul>
<input type="text" id="newText" />
<input type="button" value="Add" onclick="solve()">
<script>
function solve() {
    // TODO...
}
</script>
```



Solution: Add / Delete Items

```
function solve() {
  let newElement = document.getElementById("newText").value;
  let list = document.getElementById("items");

  if (newElement.length === 0) return;

  let listItem = document.createElement("li");
  listItem.textContent = newElement;

  let remove = document.createElement("a");
  let linkText = document.createTextNode("[Delete]");
  // Continued on the next slide ...
```

Solution: Add / Delete Items

```
  remove.appendChild(linkText);
  remove.href = "#";
  remove.addEventListener("click", deleteItem);

  listItem.appendChild(remove);
  list.appendChild(listItem);

  function deleteItem() {
    listItem.remove();
  }
}
```

Check your solution here: https://judge.softuni.org/Contests/Compete/Index/3794#5

Events Handler Execution Context

- In event handlers, **this** refers to the event **source element**

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

- Pay attention when using **object methods** as event listeners!
  - **this** may not behave as you expect with objects

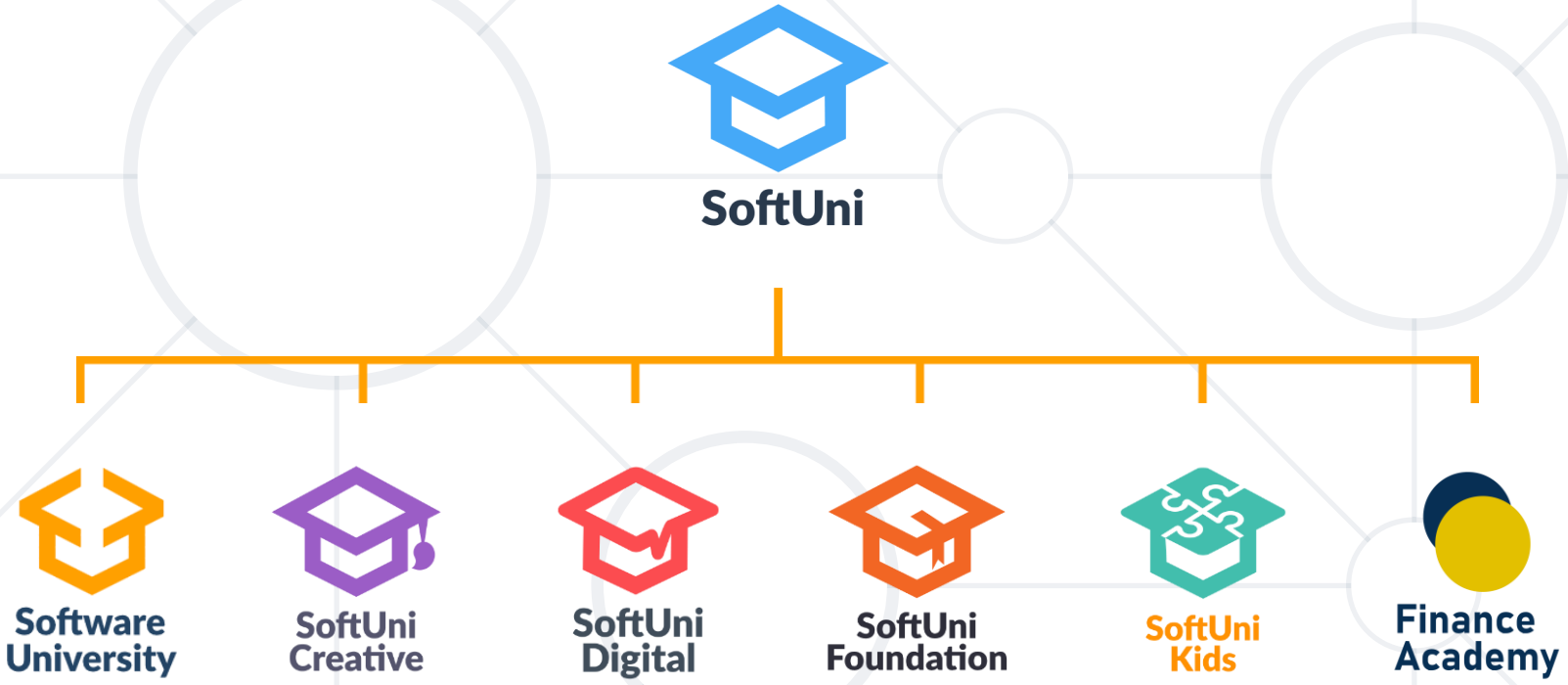
Summary

- DOM – programming API for HTML documents
- Selecting DOM Elements
  - By Id, By Class Name, By Tag Name
  - Query Selectors
- The DOM Tree can be manipulated
  - Creating, Updating, Deleting Children/Parent Elem ents
- DOM Events
  - Select Type & Handler Function

License

- 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

Questions?



© SoftUni – https://about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

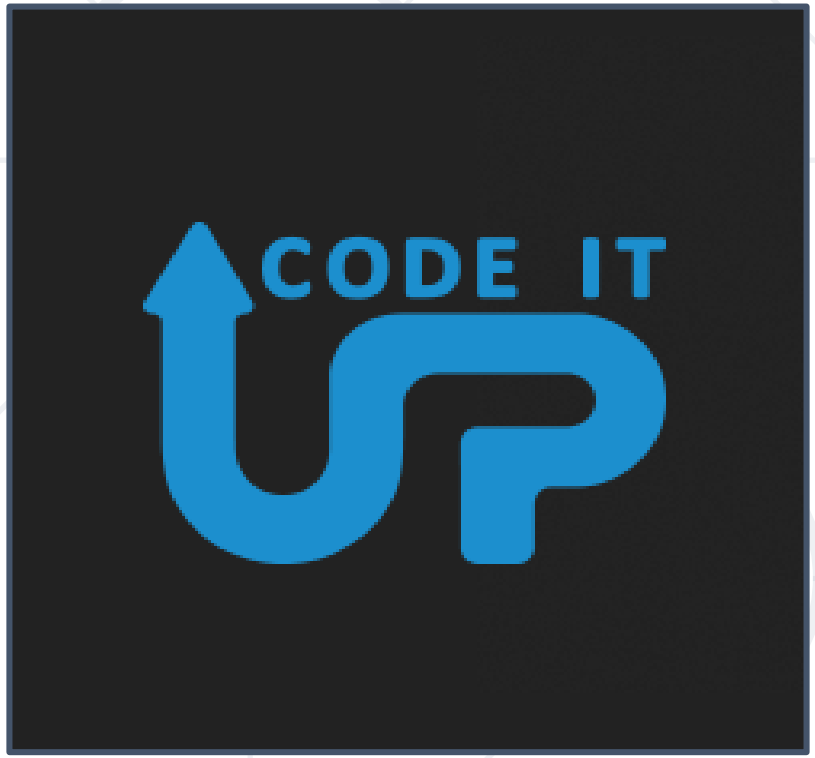
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg