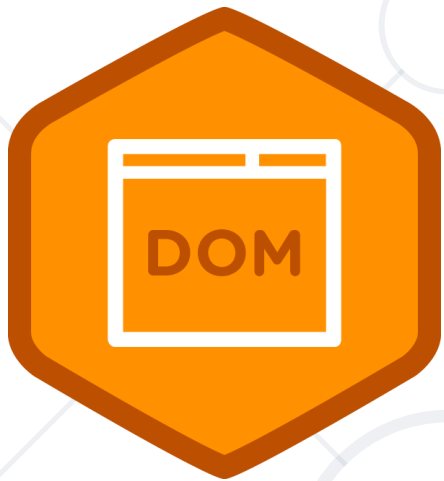


# DOM Introduction

## Document Object Model



**SoftUni Team**  
**Technical Trainers**



**SoftUni**



**Software University**

<https://softuni.bg>



sli.do

#js-advanced

# Table of Contents

1. **Browser API**
2. **Document Object Model**
3. **HTML** Elements
4. **Targeting** Elements
5. Using the **DOM API**





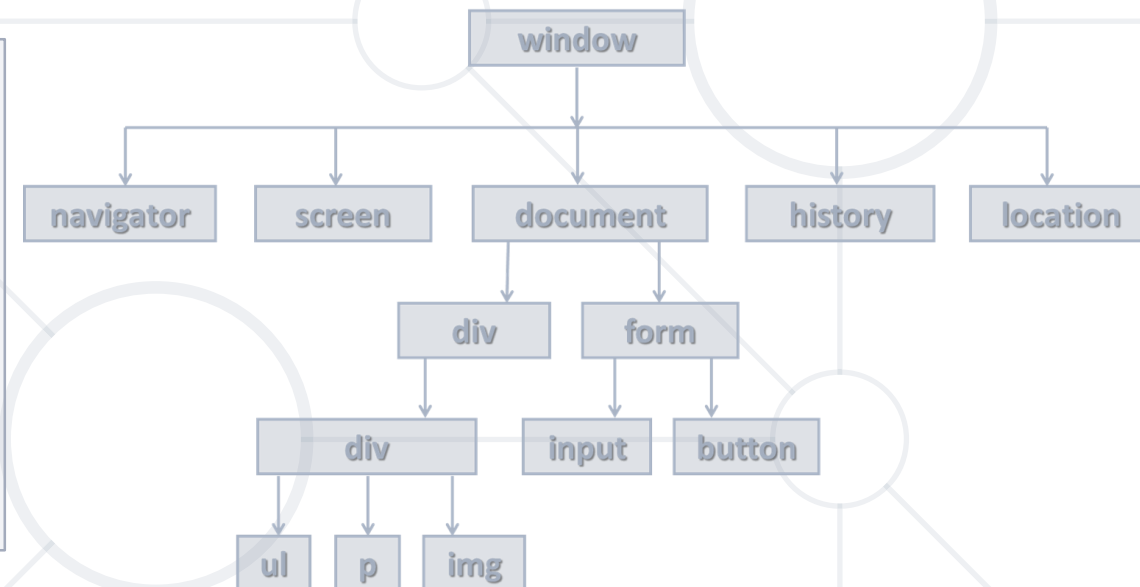
**Browser API**

# 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);
```



- Most of this **API** will be examined in the **next course**

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

```
let b = 8;  
console.log(this.b); // undefined
```

```
var a = 5;  
console.log(this.a); // 5
```

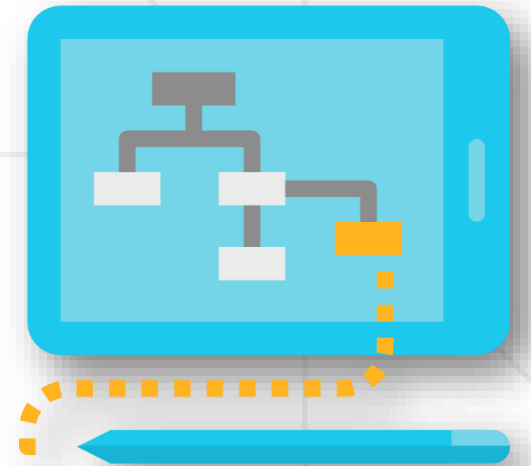
```
function foo() {  
  console.log("Simple function call");  
  console.log(this === window); // true  
}  
foo();
```





# Document Object Model (DOM)

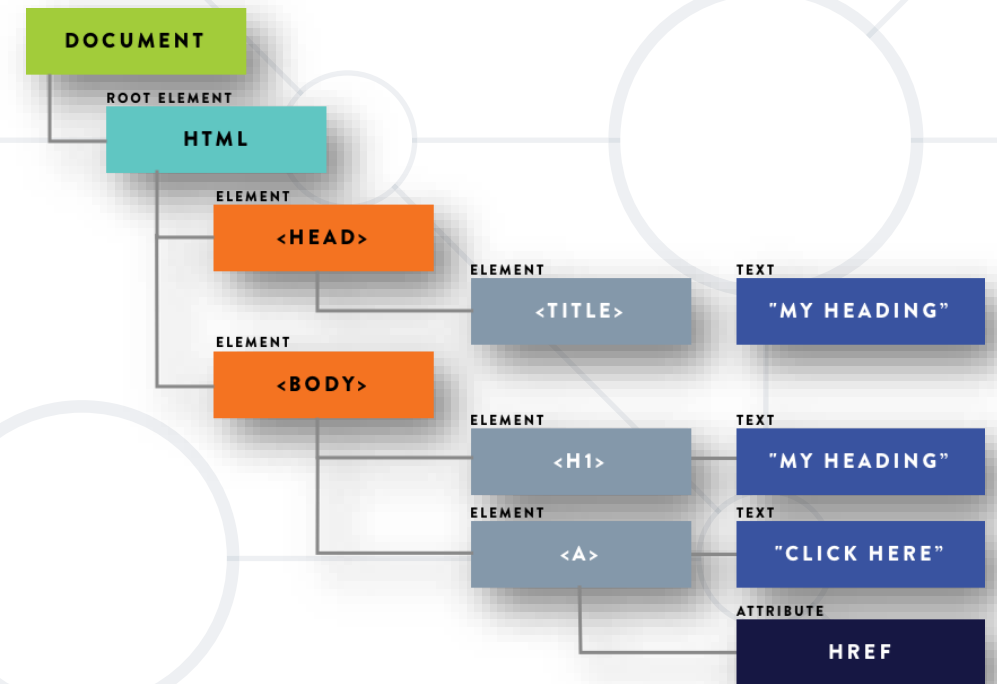
- The **DOM** represents the document as **nodes** and **objects**
  - That way, the programming languages **can connect** to the page
- The **HTML DOM** is an **Object Model** for **HTML**. It defines:
  - HTML elements as **objects**
  - **Properties**
  - **Methods**
  - **Events**





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

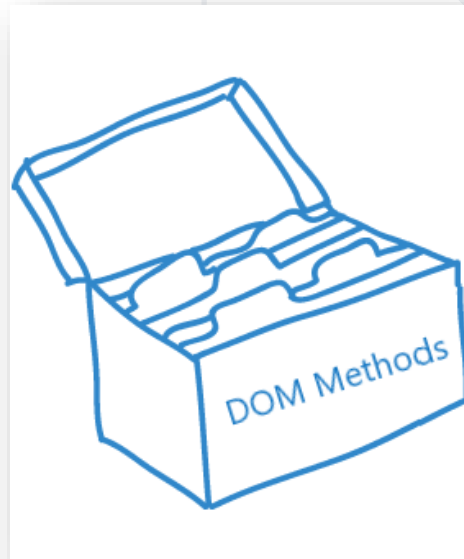
```
<html>
  <head>
    <title>My Heading</title>
  </head>
  <body>
    <h1>My Heading</h1>
    <a href="/about">Click Here</a>
  </body>
</html>
```



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

# DOM Methods

- **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>
```

- 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

- 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

# 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**





# 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 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



- 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: Edit Element

- Create function **edit()** that takes **three** parameters:
  - A **reference** to an HTML element
  - Two strings – **match** and **replacer**
- Replace all occurrences of **match** inside the **text content** of the given element with **replacer**

```
▼ <body>  
  <h1>Hello, %insert name here%!</h1>  
</body>
```

```
'%insert name here%',  
'Document Object Model'
```



```
▼ <body>  
  <h1>Hello, Document Object Model </h1>  
</body>
```

# Solution: Edit Element

```
function edit(ref, match, replacer) {  
  const content = ref.textContent;  
  const matcher = new RegExp(match, 'g');  
  const edited = content.replace(matcher, replacer);  
  ref.textContent = edited;  
}
```

Check your solution here: <https://judge.softuni.bg/Contests/2760#0>



# 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 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**

- **CSS selectors** are strings that follow CSS syntax for matching
- They allow very fast and powerful element matching, e.g.:
  - **"#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"

- 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
```

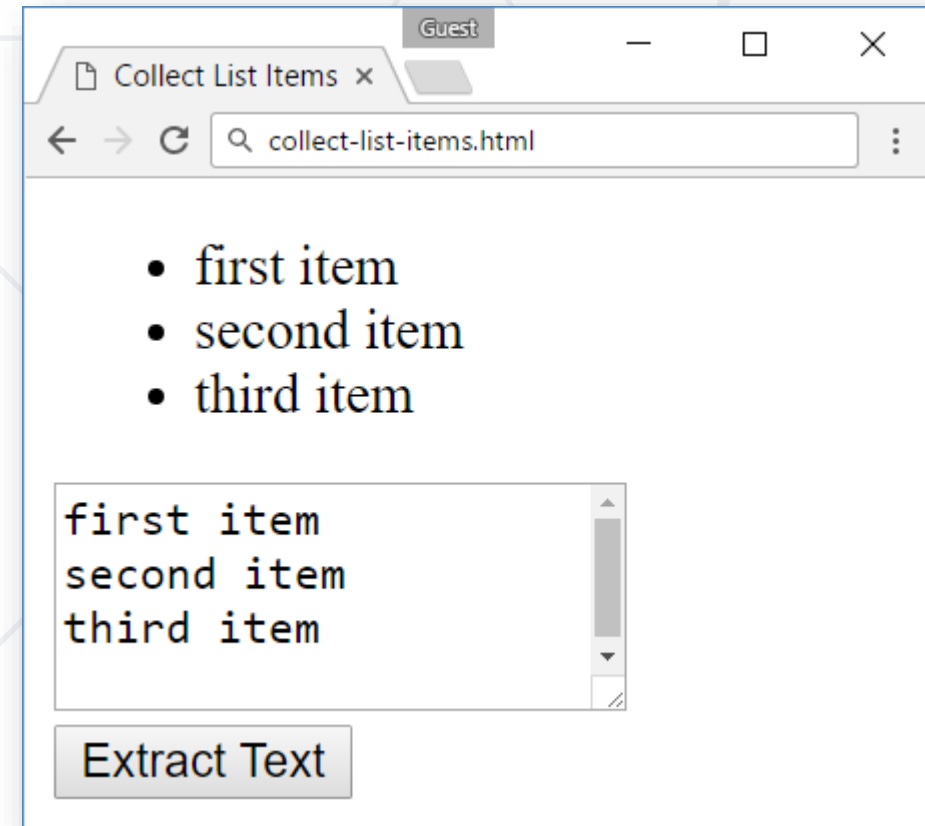
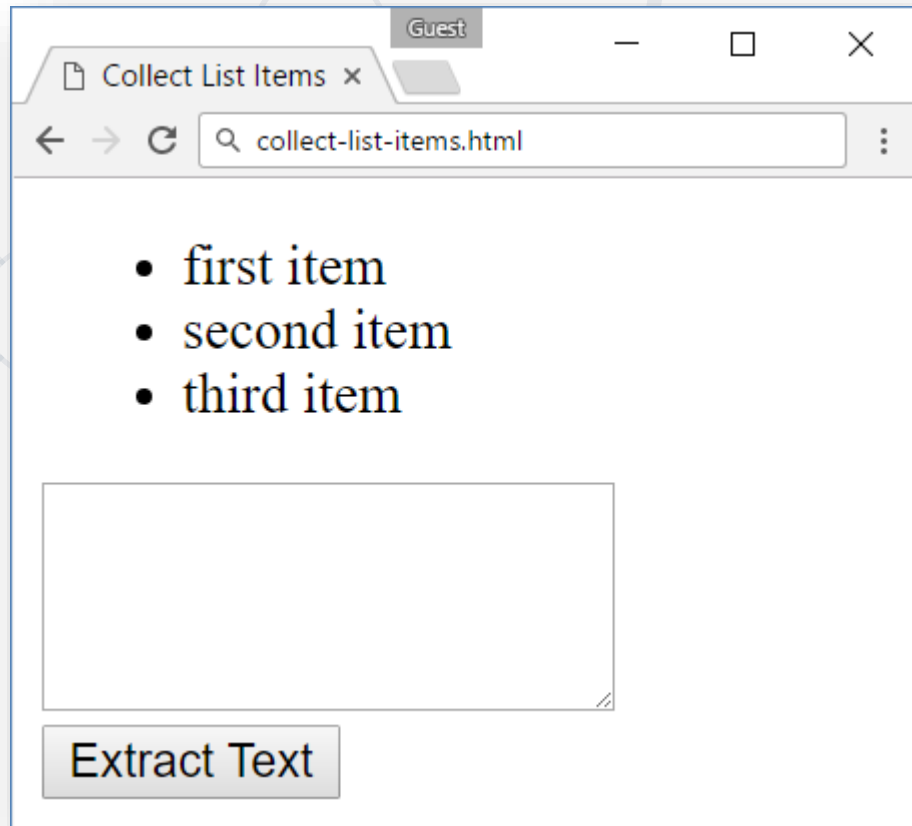
- Both can be **explicitly converted** to an array

```
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**





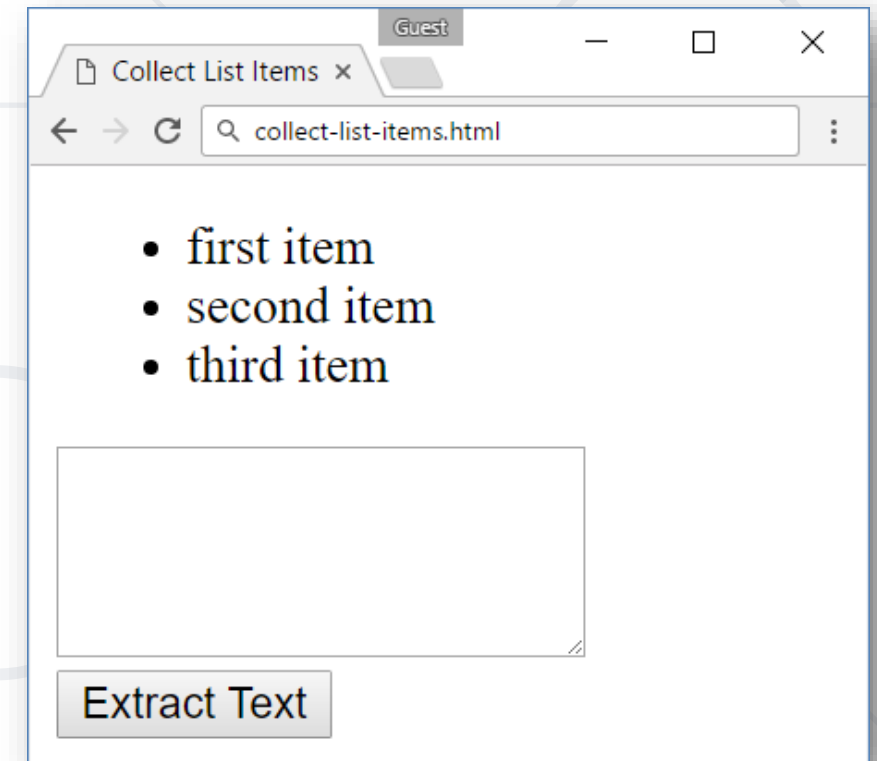
# Problem: Collect List Items – HTML

```
<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.bg/Contests/2760#1>

# 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>
```

- 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



# 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**!

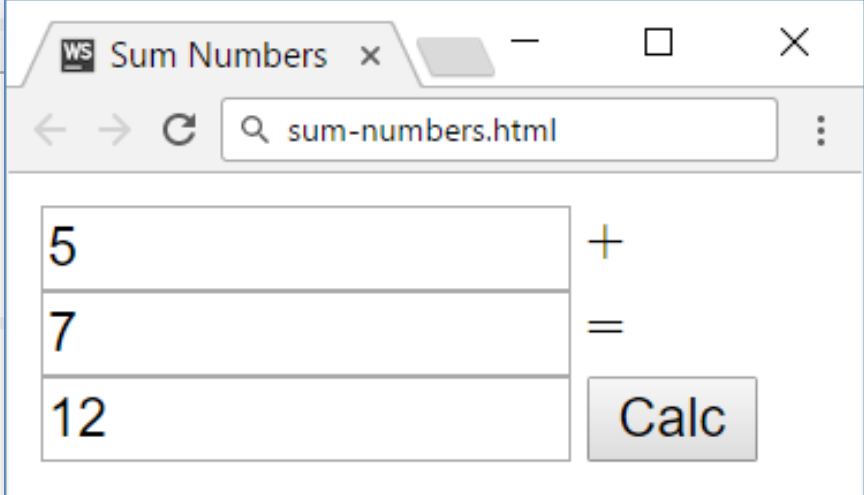
# Problem: Sum Numbers

- 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  
}
```



5	+
7	=
12	<button>Calc</button>

# 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.bg/Contests/2760#2>



- 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>

# Solution: Show More Text

```
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() {
    document.getElementById('text')
      .style.display = 'inline';
    document.getElementById('more')
      .style.display = 'none';
  }
</script>
```



- 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
```

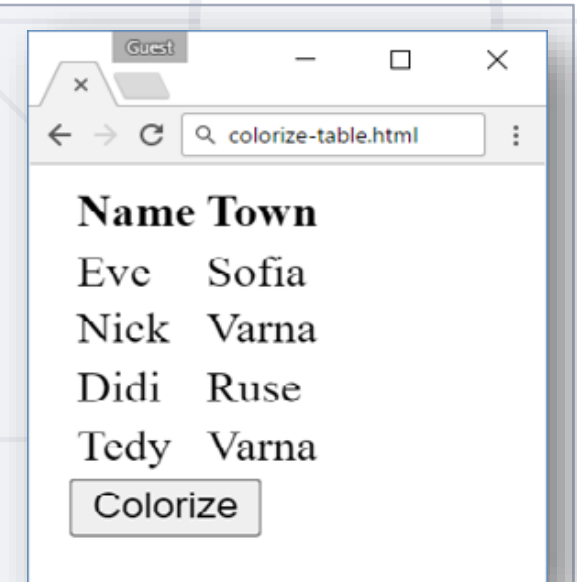
```
const thirdLi = list.getElementsByTagName('li')[2];  
// Third <li> inside the selected <ul>
```

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

# Problem: Colorize Table Rows

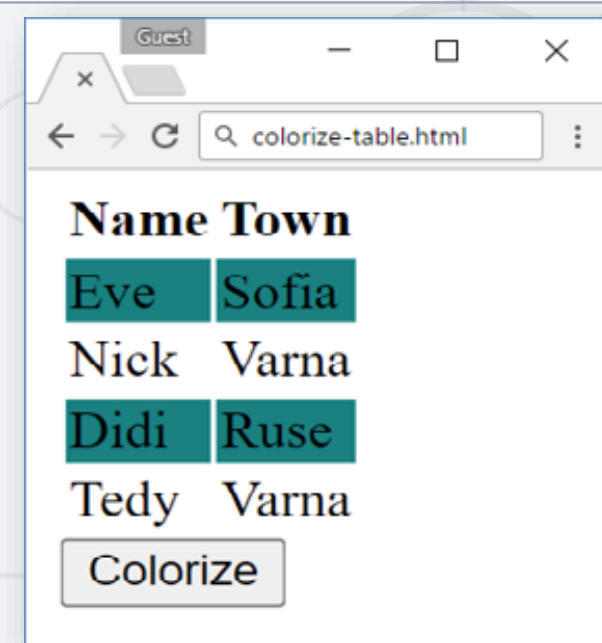
- A HTML page holds a **table** with **rows**
  - On button click, colorize in color "**teal**" all even rows

```
<table border="1">  
  <tr><th>Name</th><th>Town</th></tr>  
  <tr><td>Eve</td><td>Sofia</td></tr>  
  <tr><td>Nick</td><td>Varna</td></tr>  
  <tr><td>Didi</td><td>Ruse</td></tr>  
  <tr><td>Tedy</td><td>Varna</td></tr>  
</table>  
<button onclick="colorizeRows()">Colorize</button>
```



# Solution: Colorize Table Rows

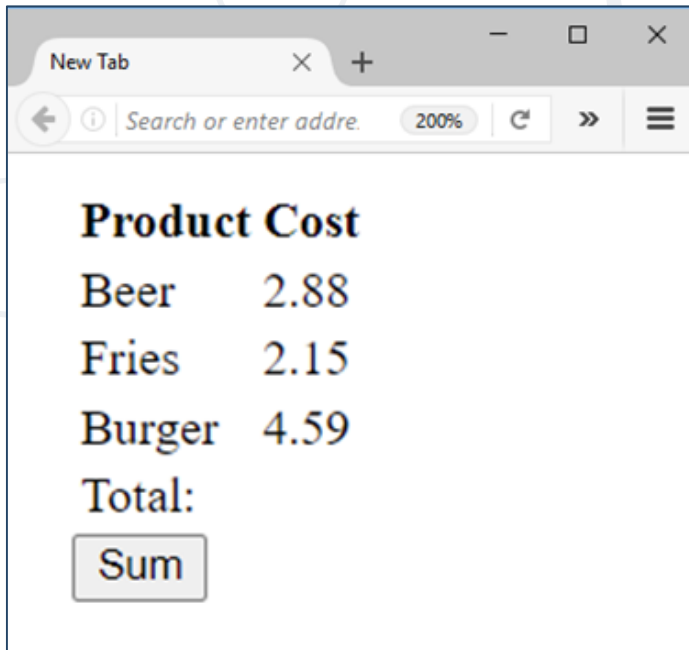
```
function colorizeRows() {  
  let rows = document.  
    querySelectorAll("table tr");  
  let index = 0;  
  for (let row of rows) {  
    index++;  
    if (index % 2 == 0)  
      row.style.background = "teal";  
  }  
}
```



Check your solution here: <https://judge.softuni.bg/Contests/2760#4>

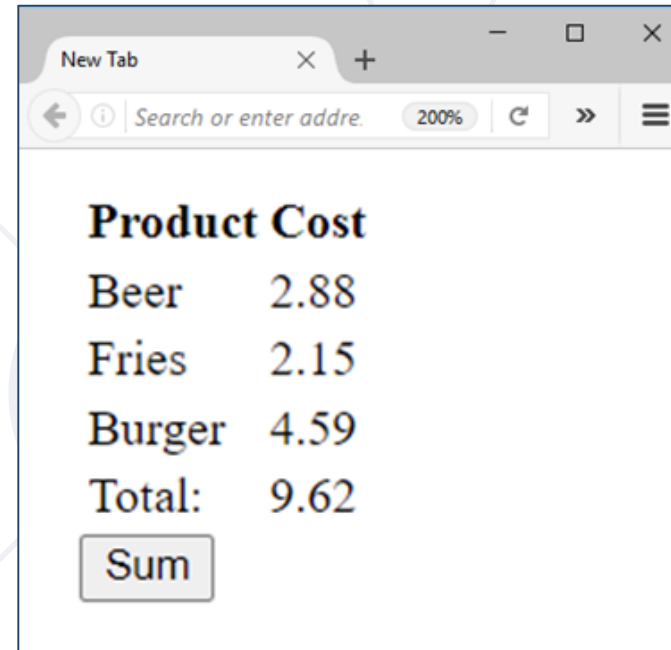
# Problem: Sum Table

- Find the **first table** and sum all values in the **last column**
- Display the result inside element with ID **"sum"**



Product Cost	
Beer	2.88
Fries	2.15
Burger	4.59
Total:	

Sum



Product Cost	
Beer	2.88
Fries	2.15
Burger	4.59
Total:	9.62

Sum



# Problem: Sum Table

- Sample HTML

```
<table>
  <tbody>
    <tr><th>Product</th><th>Cost</th></tr>
    <tr><td>Beer</td>    <td>2.88</td></tr>
    <tr><td>Fries</td>    <td>2.15</td></tr>
    <tr><td>Burger</td>    <td>4.59</td></tr>
    <tr><td>Total:</td>    <td id="sum"></td></tr>
  </tbody>
</table>
<button onclick="sum()">Sum</button>
```

# Solution: Sum Table

```
function sum() {  
  let table = document.querySelectorAll("table tr");  
  let total = 0;  
  for (let i = 1; i < table.length; i++) {  
    let cols = table[i].children;  
    let cost = cols[cols.length - 1].textContent;  
    total += Number(cost);  
  }  
  document.getElementById("sum").textContent = total;  
}
```

# Problem: Extract Parenthesis

- Extract all **parenthesized text** from a **target** paragraph
  - Your function will receive an element ID to parse
  - Return the result as string, joined by "; "

```
...<!DOCTYPE html> == $0
<html lang="en">
  <head>...</head>
  <body>
    <p id="content">
      "
      The Rose Valley (Bulgaria) is located just south of the Balkan Mountains
      (Kazanlak).The most common oil-bearing rose found in the valley is the pink-
      petaled Damask rose (Rosa damascena Mill).
    "
  </p>
</body>
</html>
```

```
>>let text = extract("content")|
```



Bulgaria;  
Kazanlak;  
Rosa damascena Mill;

# Problem: Extract Parenthesis

## ■ Sample HTML

```
<p id="content">
```

```
  The Rose Valley (Bulgaria) is located just south of the  
  Balkan Mountains(Kazanlak).The most common oil-bearing rose  
  found in the valley is the pink-petaled Damask rose (Rosa  
damascena Mill).
```

```
</p>
```

```
<p id="holder">
```

```
  Lorem ipsum dolor sit amet, (consectetur adipiscing elit),  
  sed do eiusmod (tempor) incididunt ut labore (et dolore  
magna) aliqua.
```

```
</p>
```

# Solution: Extract Parenthesis

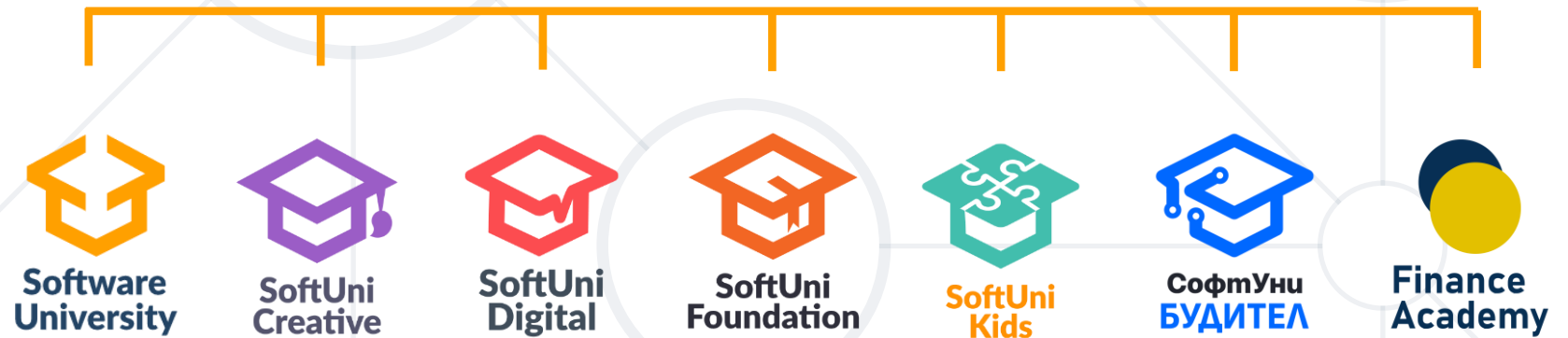
```
function extract(elementId) {  
  let para = document.getElementById(elementId).textContent;  
  let pattern = /\(([^\)]+)\)/g;  
  let result = [];  
  
  let match = pattern.exec(para);  
  while(match) {  
    result.push(match[1]);  
    match = pattern.exec(para);  
  }  
  
  return result.join('; ');  
}
```

Check your solution here: <https://judge.softuni.bg/Contests/2760#6>

- BOM – Browser API
- DOM
  - **DOM** is a programming API for HTML XML documents
  - Selecting DOM elements
    - By **Id**
    - By **Class** Name
    - **Query** Selectors
  - DOM **Properties** & HTML **Attributes**



# Questions?



# SoftUni Diamond Partners





- 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>



- 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)

