Created by Dr. Zoltán Subecz, GAMF educational materials were used.
John von Neumann University, Kecskemét, GAMF Faculty of Engineering and Computer Science, Informatics Department
subecz.zoltan@gamf.uni-neumann.hu

Only informative text = we don't study it, only additional information

Web-programming-1 Lecture

In this document, I have written many comments to the code of the exercises. You can find the solutions without explanations in the file **Solutions.zip**.

I have written the explanations after the // signs for simplicity.
These must be removed from the HTML and CSS files for proper operation.
A valid HTML comment would be: <!-- Comment -->

You don't have to memorize the code: there is no Exam in the Lecture, you have to use it only in the Homework.

1-2 - HTML, CSS

In the lecture we study **client-side languages**: **HTML, CSS, JavaScript**. The browser understands these. In the Seminar, we will also learn the **server-side language PHP**. The browser does not understand PHP, you need a server that interprets and processes it.

Description of websites

- HTML (Hypertext Markup Language): description of the logical structure and content
- CSS (Cascading Style Sheets): description of the format (sizes, colors, positioning, etc.)

HTML - HyperText Markup Language

Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It defines the content and structure of web content. It is often assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript, a programming language.

Simple webpage

exercise-01.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>First page</title>
  </head>
  <body>
    <h1>Our first page</h1>
    <p>Simple text</p>
  </body>
</html>
```

- The <!DOCTYPE html> declaration specifies that this is an HTML5 document
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the HTML page
- The <title> element specifies the title of the HTML page (which appears in the browser's address bar or on the page's tab)
- The <body> element defines the body of the document and is a container for all visible content, such as headings, paragraphs, images, links, tables, lists, etc.
- The <h1> element defines a large heading
  Headings are defined by tags from <h1> to <h6>, where H1 is the highest (or most important) level and H6 is the lowest.
- The <p> element defines a paragraph

Some other basic tags

 : line break

<hr> : horizontal rule or line
<!-- ... --> : comment

Character formatting

... : bold text
... : italic style
<big>...</big> : bigger text
<small>...</small> : smaller text
... : emphasized text
... : text with strong importance
_{...} : subscript text. e.g. a_2
^{...} : superscript text e.g. a^2

Hyperlinks

GAMF

Anchor links: a link bound to a portion of a document, which is often called a fragment.

Fragments are marked with anchors:
Izsáki út 10.

Hyperlink to the anchor
Post address

Tables

Create the next table:

1 row, 1 cell	1 row, 2 cell
2 row, 1 cell	2 row, 2 cell

exercise-02.html

```
<!DOCTYPE html>
<html >
  <head>
    <meta charset="utf-8">
    <title>Táblázat</title>
  </head>
  <body>
    <table border="1">
        <tr>
          <td>1 row, 1 cell</td>
          <td>1 row, 2 cell</td>
        </tr>
        <tr>
          <td>2 row, 1 cell</td>
          <td>2 row, 2 cell</td>
        </tr>
    </table>
  </body>
</html>
```

Lists

Create the following unordered and ordered lists:

- list element
- list element

1. list element
2. list element

exercise-03.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Lists</title>
  </head>
  <body>
    <ul>
       <li>list element</li>
       <li>list element</li>
    </ul>
    <ol>
       <li>list element</li>
       <li>list element</li>
    </ol>
  </body>
</html>
```

Images

```
<img src="image.jpg" alt="alternative text">
```

with sizing:

```
<img src="image.jpg" width="300px" alt="alternative text">
```

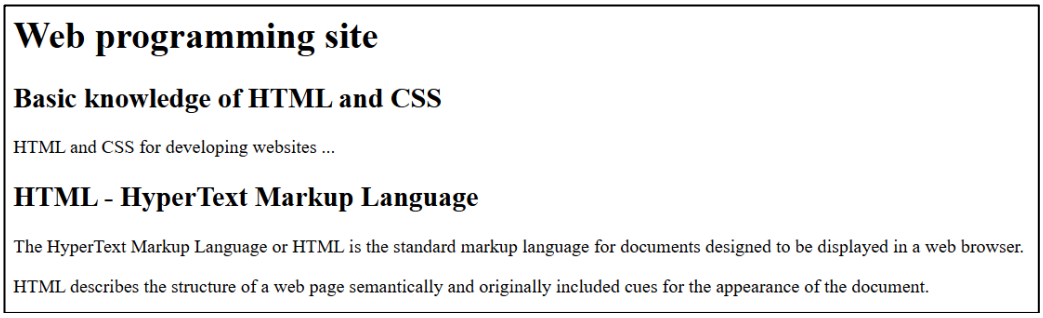
CSS - Cascading Style Sheet

Cascading Style Sheets (CSS) is a style sheet language used for specifying the presentation and styling of a document written in a markup language such as HTML or XML.

- Visual layout and presentation of content
- Styles can be used to separate the structure and appearance of the page
- External style sheets speed up the implementation of a uniform appearance

Exercise: HTML page to which we will apply CSS formatting

Create the following HTML page:

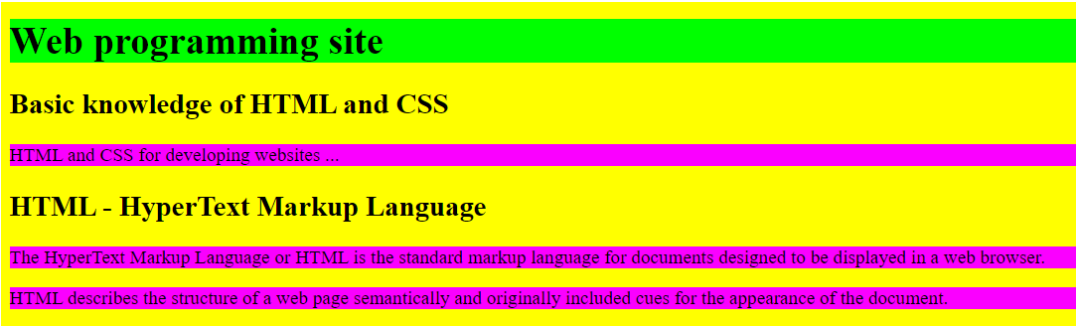


exercise-04a.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Lecture 1</title>
</head>
<body>
   <h1>Web programming site</h1>
   <h2>Basic knowledge of HTML and CSS</h2>
   <p>HTML and CSS for developing websites ...</p>
   <h2>HTML - HyperText Markup Language</h2>
   <p>The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser.</p>
   <p>HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.</p>
</body>
</html>
```

Applying CSS to the page

The content should be the same as in the previous exercise, just change the style:



exercise-04b.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Lecture 1</title>
   <style>
      body {background-color: yellow}
      h1 {background-color: #00ff00}
      h2 {background-color: transparent}
      p {background-color: rgb(250,0,255)}
```

```
    </style>
</head>
<body>
    <h1>Web programming site</h1>
    <h2>Basic knowledge of HTML and CSS</h2>
    <p>HTML and CSS for developing websites ...</p>
    <h2>HTML - HyperText Markup Language</h2>
    <p>The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser.</p>
    <p>HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.</p>
</body>
</html>
```

background-image

The content should be the same as in the previous exercise, just change the style:

exercise-04c.html

```
……….
    <style>
        body {background-color: yellow}
        body {
            background-image: url('internet.png');
            background-repeat: no-repeat;
            background-position: top center;
        }
    </style>
</head>
<body>
……….
</body>
</html>
```

Borders

The content should be the same as in the previous exercise, just change the style:

Web programming site

Basic knowledge of HTML and CSS

HTML and CSS for developing websites ...

HTML - HyperText Markup Language

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser.

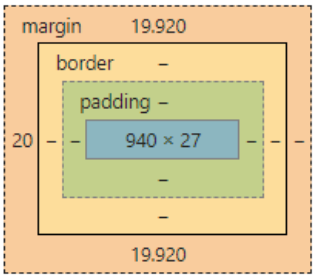
HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

exercise-04d.html

```
……..
   <style>
      h1 {border-style: dotted}
      h2 {border-style: double}
      p {border-style: solid}
   </style>
</head>
<body>
……..
</body>
</html>
```

Difference between margin and padding: Box Model

https://www.w3schools.com/css/css_boxmodel.asp
Right click on page / Inspect / Select an element



- **Padding** - area around the content inside the border.
- **Margin** - area outside the border.

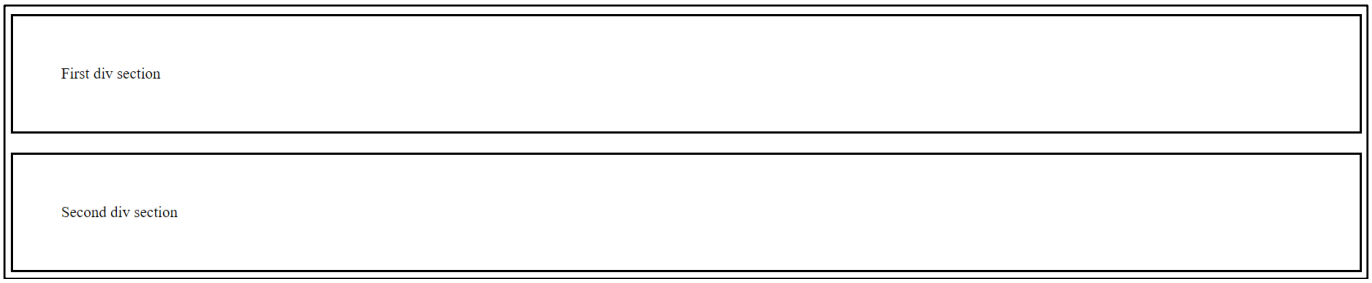
If we enter 4 data for the margin, they are: top, right, bottom, left

```
p { margin: 2cm 5px 2em 5% }
```

if 2 data is given: top and bottom, right and left p { margin: 5px 10px }
if 1 data is given: applies to all 4 directions p { margin: 10px }

Exercise

Create the following page with 2 blocks



exercise-05.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Lecture 2</title>
        <style>
                div {
                        padding: 50px;
                        border: solid;
                        margin: 20px;
                }
        </style>
</head>
<body>
        <div>First div section</div>
        <div>Second div section</div>
</body>
</html>
```

Difference Between Class and ID: in HTML-ben and in CSS

ID: Tags with **unique identifiers:**

in HTML:

```
<p id="cr">Copyright...</p>
```

In CSS

```
#cr {
  text-align: center;
  color: white;
  font-weight: bold
}
```

CLASS: Tags with the same features can be classified into classes

in HTML

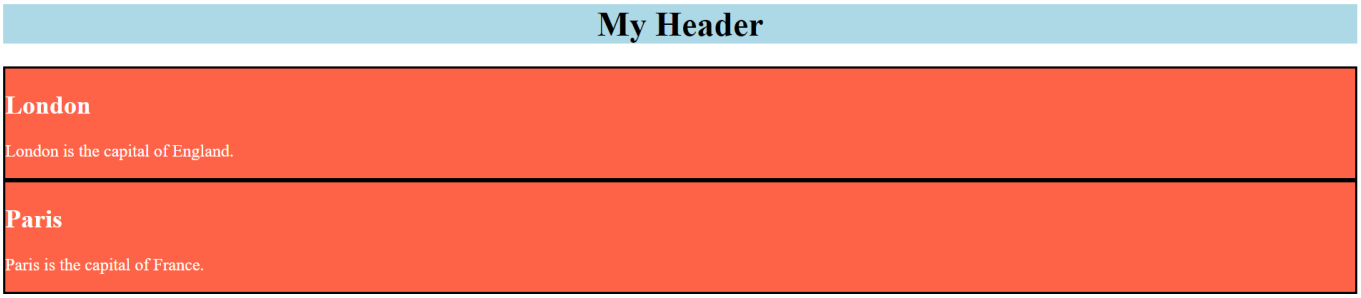
```
<p class="fruit">apple</p>
<p class="fruit">banana</p>
```

in CSS:

```
p.fruit { text-align: right }
```

Exercise

Create the next page:



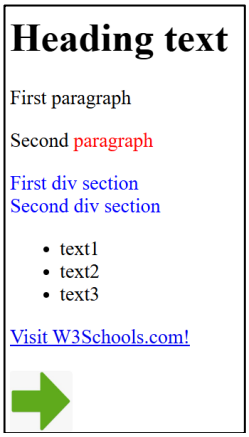
Use ID for the Header and CLASS for the cities.

exercise-06.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>Lecture 1</title>
      <style>
         #myHeader {
            background-color: lightblue;
            color: black;
            text-align: center;
         }
         .city {
            background-color: tomato;
            color: white;
            border: 2px solid black;
         }
      </style>
   </head>
   <body>
      <h1 id="myHeader">My Header</h1>
      <div class="city">
         <h2>London</h2>
         <p>London is the capital of England.</p>
      </div>
      <div class="city">
         <h2>Paris</h2>
         <p>Paris is the capital of France.</p>
      </div>
   </body>
</html>
```

Exercise –Span: inline container: it is easily styled by CSS

Create the next page:

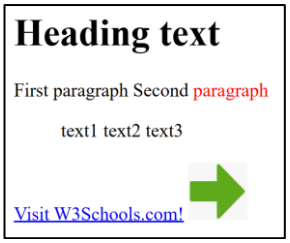


exercise-07.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Lecture 2</title>
   <style>
      span {color: red}
      div {color: blue}
   </style>
</head>
<body>
   <h1>Heading text</h1>
   <p>First paragraph</p>
   <p>Second <span>paragraph</span></p>
   <div>First div section</div>
   <div>Second div section</div>
   <ul>
      <li>text1</li>
      <li>text2</li>
      <li>text3</li>
      </ul>
   <a href="https://www.w3schools.com/">Visit W3Schools.com!</a>
   <p></p>
   <img src="arrow.jpg" width="50">
</body>
</html>
```

Placing list items in a row, hiding items

Create the following page. Do not change anything inside the <body> tag in the previous task.



feladat-08.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Lecture 2</title>
        <style>
                p,li {display: inline}
                span {color: red}
                div {display: none}
        </style>
</head>
<body>
……
</body>
</html>
```

Float

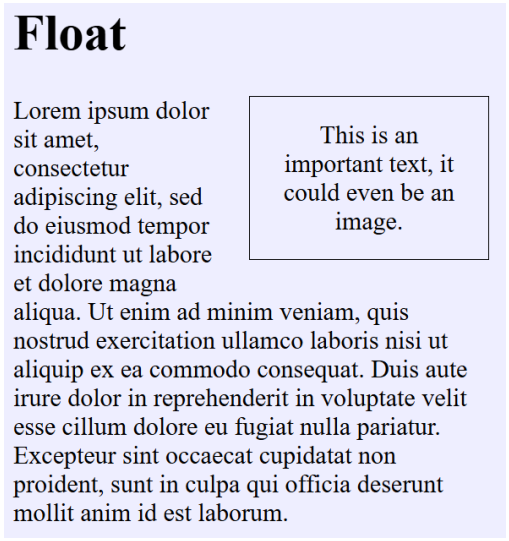
First
Second
Fourth Third
FifthSixth

exercise-09.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Float</title>
    <style type="text/css">
       .right { float:right; }
       .left { float:left; }
    </style>
  </head>
  <body>
    <div>First</div>
    <div>Second</div>
    <div class="right">Third</div>
    <div>Fourth</div>
    <div class="left">Fifth</div>
    <div>Sixth</div>
  </body>
</html>
```

The First and Second DIV boxes are placed below each other because they are block-level elements.
We put the Third one on the right and wrap it around with the next element from the left (Fourth)
We put the Fifth one on the left and wrap it around with the next element from the right (Sixth)

Exercise - Float



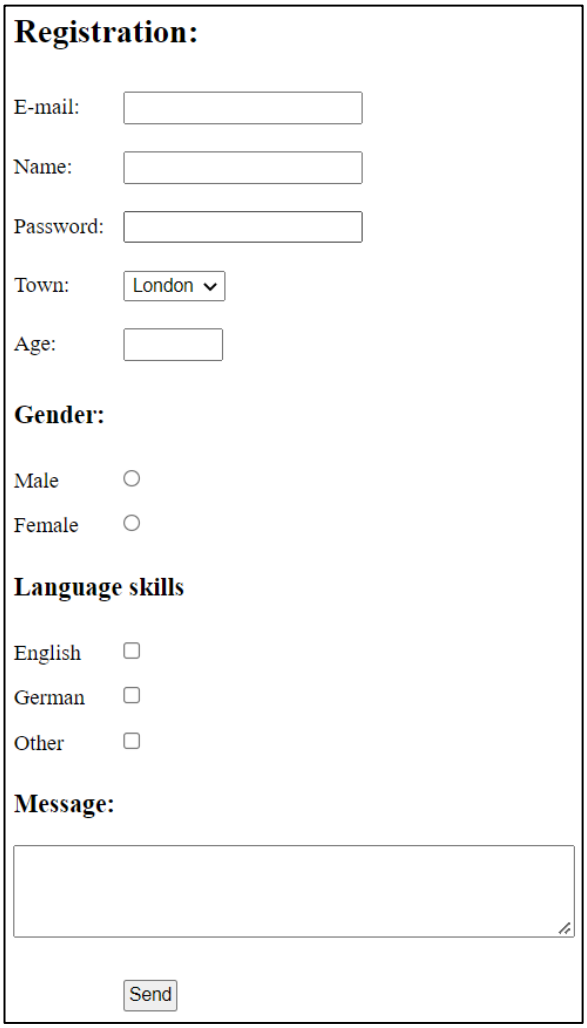
exrcise-10.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
 <head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Float</title>
  <STYLE TYPE="text/css">
     div.floatright {
        float:right;
        width:120px;
        margin:0 0 15px 20px;
        padding:15px;
        border:1px solid black;
        text-align:center;
     }
  </STYLE>
 </head>
 <body style="background-color: #eeeeff;">
  <h1>Float</h1>
  <div style="width:300px">
   <div class="floatright">
    This is an important text, it could even be an image.
   </div>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
  </div>
 </body>
</html>
```

We nested the two DIV boxes, so the width set for the outer DIV also defines the inner one.

Forms

Create the next form:



exercise-11.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Form</title>
   <meta charset="utf-8">
   <link type="text/css" rel="stylesheet" href="style-11.css">
</head>
<body>
      <h2>Registration:</h2>
      <form action="process.php" method="post">
            <label>E-mail:</label><input type="email" name="email"><br>
            <label>Name:</label><input type="text" name="name"><br>
            <label>Password:</label><input type="password" name="passw"><br>
            <label>Town:</label>
            <select name="city">
               <option value="1">London</option>
               <option value="2">Paris</option>
               <option value="3">Berlin</option>
```

```
            <option value="4">Rome</option>
        </select><br>
        <label>Age:</label><input type="number" name="age" min="10" max="120"><br>
            <h3>Gender:</h3>
            <label>Male</label><input type="radio" name="gender"  value="1"><br/>
            <label>Female</label><input type="radio" name="gender"  value="2">
            <h3>Language skills</h3>
            <label>English</label><input type="checkbox" name="english">
            <br>
            <label>German</label><input type="checkbox" name="german">
            <br>
            <label>Other</label><input type="checkbox" name="other">
            <p></p>
            <h3>Message:</h3>
            <textarea name="message" rows="4" cols="50"></textarea>
            <p></p>
        <label></label><input type="submit" value="Send">
    </form>
</body>
</html>
```

style-11.css

```
label {
   width: 60px;
   display: inline-block;
}
input, select {
   margin: 10px 0;
   padding: 2px;
}
```

Horizontal navigation

Create the next Horizontal navigation

The buttons are blue, and when you hover your mouse over them, they turn red.
Clicking on the link will open the page in a new browser tab.

exercise-12.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
 <head>
   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
   <title>Horizontal navigation</title>
   <style type="text/css">
    ul {
      float: left;
      width: 100%;
      padding: 0; margin: 0;
```

```
      list-style-type: none;
    }
    li {
      display: inline;
    }
    a {
      float: left;
      padding: 3px 5px;
      margin: 0px 1px;
      text-decoration: none;
      color: white;
      background-color: #3399ee;
    }
    a:hover {
      background-color: #cc6600;
    }
  </style>
 </head>
 <body style="background-color: #eeeeff;">
// list with hyperlinks:
  <ul>
// target="_blank": open the page in a new browser tab
    <li><a href="https://www.bbc.com" target="_blank">BBC</a></li>
    <li><a href="https://www.cnn.com" target="_blank">CNN</a></li>
    <li><a href="https://www.bloomberg.com" target="_blank">Bloomberg</a></li>
  </ul>
 </body>
</html>
```

Creating the page structure with DIVs

Create the next page with DIVs:

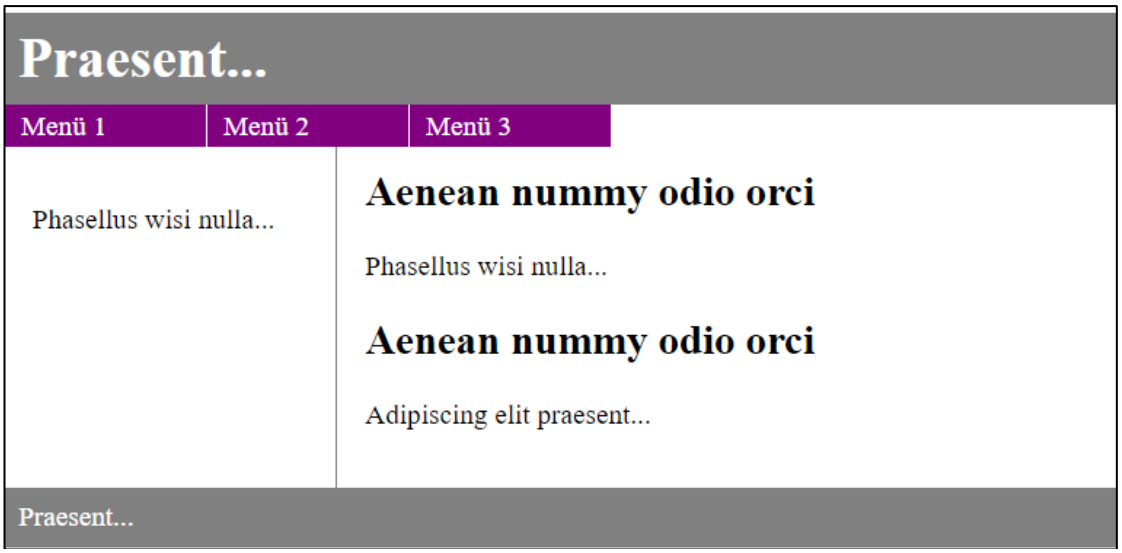
exercise-13.html

```
<!DOCTYPE html>
<html>
```

```
<head>
   <meta charset="utf-8">
   <title>Creating the page structure with DIVs</title>
   <style>
      #intro {color: red; font-weight: bold;}
      div.container {
         width: 100%;
         margin: 0px;
         border: 1px solid gray;
         line-height: 150%;
      }
      div.header, div.footer {
         padding: 0.5em;
         color: white;
         background-color: gray;
      }
      h1.header {
         padding: 0;
         margin: 0;
      }
      div.left {
         float: left;
         width: 160px;
         margin: 0;
         padding: 1em;
      }
      div.content {
         margin-left: 190px;
         border-left: 1px solid gray;
         padding: 1em;
      }
   </style>
</head>
<body>
         <p id="intro">Introduction</p>
   <div class="container">
      <div class="header">
         <h1 class="header">Praesent...</h1>
      </div>
      <div class="left">
         <p>Phasellus wisi nulla...</p>
      </div>
      <div class="content">
         <h2>Aenean nummy odio orci</h2>
         <p>Phasellus wisi nulla...</p>
         <p>Adipiscing elit praesent...</p>
      </div>
      <div class="footer">Praesent...</div>
   </div>
</body>
</html>
```

Creating the page structure with new HTML5 tags: header, footer, nav, aside, section, article + horizontal navigation

Create your next page with new HTML5 elements: header, footer, nav, aside, section, article + horizontal navigation



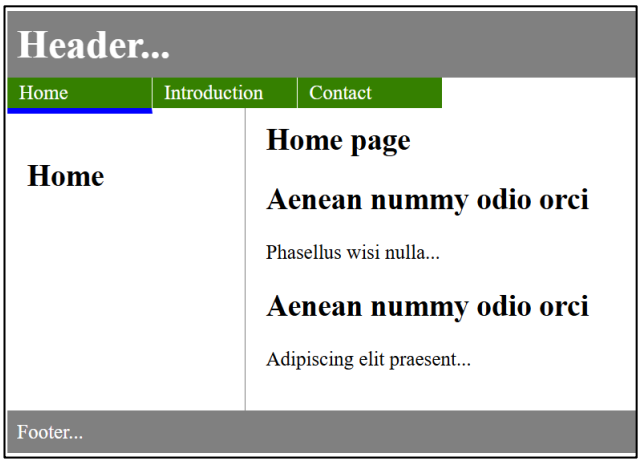
exercise-14.html

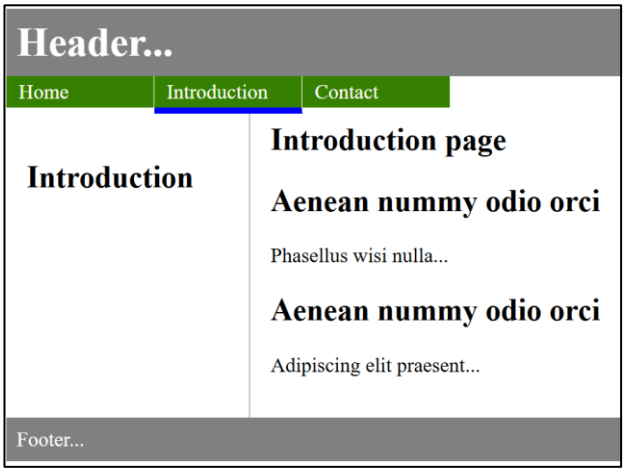
```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Lecture 2</title>
   <style>
      header, footer {
         padding: 0.5em;
         color: white;
         background-color: gray;
      }
      h1.header {
         padding: 0;
         margin: 0;
      }
      aside {
         float: left;
         width: 160px;
         margin: 0;
         padding: 1em;
      }
      section {
         margin-left: 190px;
         border-left: 1px solid gray;
         padding: 1em;
      }
      ul {
         float:left;
         width:100%;
         padding:0;
```

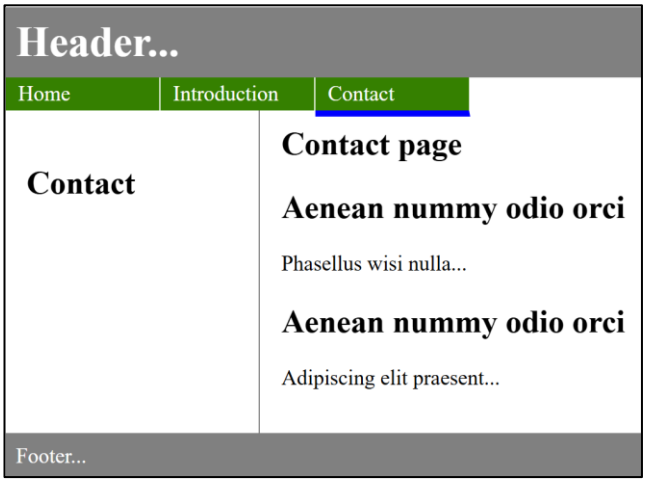
```
            margin:0;
            list-style-type:none;
        }
        a {
            float:left;
            width:6em;
            text-decoration:none;
            color:white;
            background-color:purple;
            padding:0.2em 0.6em;
            border-right:1px solid white;
        }
        a:hover { background-color:#ff3300}
        li { display:inline }
    </style>
</head>
<body>
    <header>
        <h1 class="header">Praesent...</h1>
    </header>
    <nav>
        <ul>
            <li><a href="#">Menü 1</a></li>
            <li><a href="#">Menü 2</a></li>
            <li><a href="#">Menü 3</a></li>
        </ul>
    </nav>
    <aside>
            <p>Phasellus wisi nulla...</p>
    </aside>
    <section>
        <article>
            <h2>Aenean nummy odio orci</h2>
            <p>Phasellus wisi nulla...</p>
        </article>
        <article>
            <h2>Aenean nummy odio orci</h2>
            <p>Adipiscing elit praesent...</p>
        </article>
    </section>
    <footer>Praesent...</footer>
</body>
</html>
```

Uniform pages with Horizontal navigation

- 3 pages: Home, Introduction, Contact
- Uniform pages: when clicking on the menus: the header, footer, menu bar are the same, only the content part is different
- In the menu, we mark the active menu item with a thick blue line.







Solution

There's not much new in it. What's new is marked in red.

home.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Creating the page structure with new HTML5 tags</title>
   <link type="text/css" rel="stylesheet" href="style.css">
</head>
<body>
   <header>
      <h1 class="header">Header...</h1>
   </header>
   <nav>
      <ul>
         <li><a href="home.html" class="active">Home</a></li>
         <li><a href="introduction.html">Introduction</a></li>
         <li><a href="contact.html">Contact</a></li>
      </ul>
   </nav>
```

```
    <aside>
        <h1>Home</h1>
    </aside>
    <section>
      <h1>Home page</h1>
      <article>
        <h2>Aenean nummy odio orci</h2>
        <p>Phasellus wisi nulla...</p>
      </article>
      <article>
        <h2>Aenean nummy odio orci</h2>
        <p>Adipiscing elit praesent...</p>
      </article>
    </section>
    <footer>Footer...</footer>
</body>
</html>
```

introduction.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Creating the page structure with new HTML5 tags</title>
    <link type="text/css" rel="stylesheet" href="style.css">
</head>
<body>
    <header>
      <h1 class="header">Header...</h1>
    </header>
    <nav>
      <ul>
        <li><a href="home.html">Home</a></li>
        <li><a href="introduction.html" class="active">Introduction</a></li>
        <li><a href="contact.html">Contact</a></li>
      </ul>
    </nav>
    <aside>
      <h1>Introduction</h1>
    </aside>
    <section>
      <h1>Introduction page</h1>
      <article>
        <h2>Aenean nummy odio orci</h2>
        <p>Phasellus wisi nulla...</p>
      </article>
      <article>
        <h2>Aenean nummy odio orci</h2>
        <p>Adipiscing elit praesent...</p>
      </article>
    </section>
    <footer>Footer...</footer>
</body>
```

```
</html>
```

contact.html

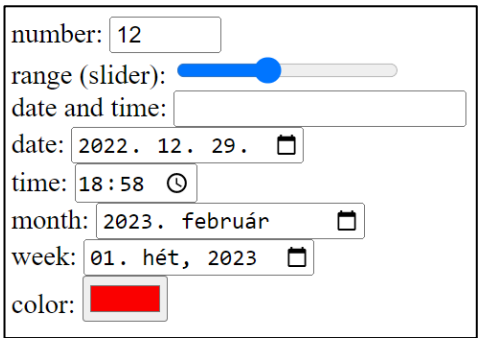
```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Creating the page structure with new HTML5 tags</title>
   <link type="text/css" rel="stylesheet" href="style.css">
</head>
<body>
   <header>
      <h1 class="header">Header...</h1>
   </header>
   <nav>
      <ul>
         <li><a href="home.html">Home</a></li>
         <li><a href="introduction.html">Introduction</a></li>
         <li><a href="contact.html" class="active">Contact</a></li>
      </ul>
   </nav>
   <aside>
      <h1>Contact</h1>
   </aside>
   <section>
      <h1>Contact page</h1>
      <article>
         <h2>Aenean nummy odio orci</h2>
         <p>Phasellus wisi nulla...</p>
      </article>
      <article>
         <h2>Aenean nummy odio orci</h2>
         <p>Adipiscing elit praesent...</p>
      </article>
   </section>
   <footer>Footer...</footer>
</body>
</html>
```

style.css

```
header, footer {
   padding: 0.5em;
   color: white;
   background-color: gray;
}
h1.header {
   padding: 0;
   margin: 0;
}
aside {
   float: left;
   width: 160px;
   margin: 0;
```

```
    padding: 1em;
}
section {
    margin-left: 190px;
    border-left: 1px solid gray;
    padding: 1em;
}
ul {
    float:left;
    width:100%;
    padding:0;
    margin:0;
    list-style-type:none;
}
a {
    float:left;
    width:6em;
    text-decoration:none;
    color:white;
    background-color:rgb(53, 128, 0);
    padding:0.2em 0.6em;
    border-right:1px solid white;
}
a.active{
    border-bottom:5px solid blue;
}
a:hover { background-color:#ff3300}
li {
    display:inline;
}
```

New HTML5 input tags



It is not in **Solutions.zip**.

……

```
<body>
        number: <input type="number" min="10" max="20" step="2"><br>
        range (slider): <input type="range" min="0" max="10" step="2" value="6"><br>
        date and time: <input type="datetime"><br>
```

```
        date: <input type="date"><br>
        time: <input type="time"><br>
        month: <input type="month"><br>
        week: <input type="week"><br>
        color: <input type="color">
</body>
……
```

Addition

A given section can have multiple style definitions applied to it. Which one will be valid? Which one will override the other?

Rule 4 is the strongest, rule 1 is the weakest:

1. **Browser default settings**
2. **External stylesheet**

```
<!DOCTYPE html>
<html>
        <head>
        <link rel="stylesheet" href="mystyle.css">
        </head>
<body>
……
```

3. **Definition (Internal) in the HTML header**

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
    h2 { background-color: yellow }
</style>
        <head>
<body>
……
```

4. **Inline style:**

```
<h1 style="color: #000099">This is a heading</h1>
```

Media Types - Apply a Different Stylesheet for Each Device

all, print, projection, screen

```
<link rel="stylesheet" type="text/css"  href="screen.css" media="screen" />
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
```

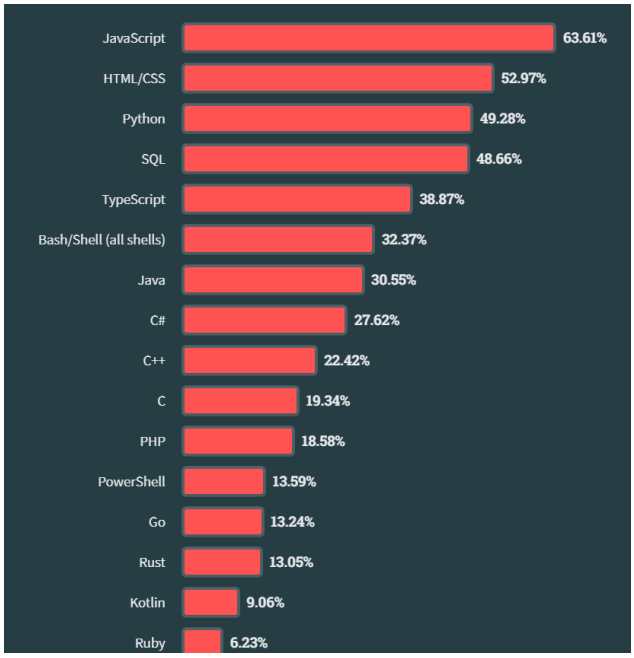
3-4 – JavaScript - Why do we study so much JavaScript? => one of the most popular programming languages

Introduction

Developer: Brendan Eich + Netscape, First standard version: 1997,
Currently used by 98% of websites, NodeJS also uses it

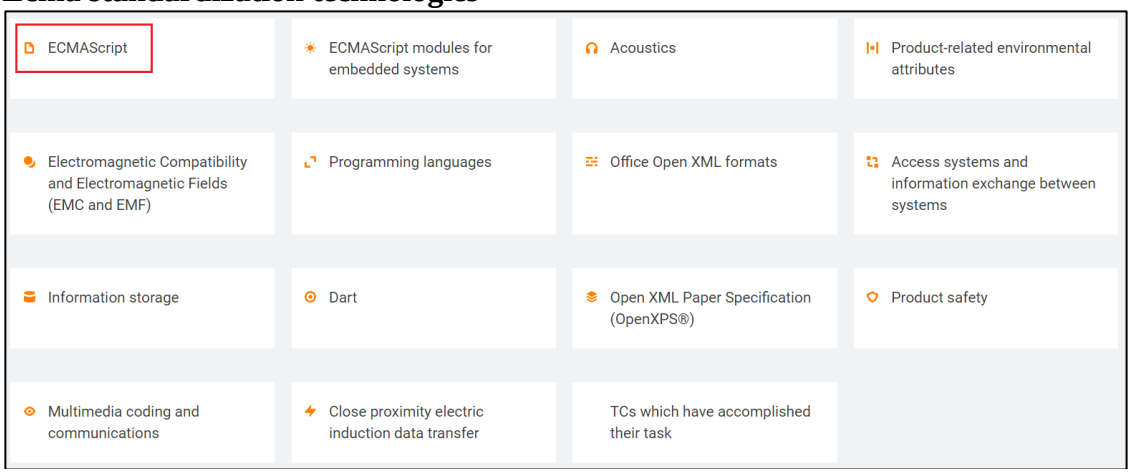
JavaScript is one of the most popular programming languages

https://survey.stackoverflow.co/2023/
How many of those surveyed used it:



JavaScript is a high-level, often just-in-time compiled language that conforms to the **ECMAScript** standard.

ECMA: Ecma International https://www.ecma-international.org/
Ecma International is an industry association dedicated to the standardization of information and communication systems
Ecma standardization technologies



ECMAScript

http://www.ecma-international.org/publications/standards/Stnindex.htm
https://en.wikipedia.org/wiki/ECMAScript

ECMAScript is a JavaScript standard intended to ensure the interoperability of web pages across different browsers.

ECMAScript is commonly used for client-side scripting on the World Wide Web, and it is increasingly being used for writing server-side applications and services using Node.js and other runtime environments.

Major implementations
JavaScript, SpiderMonkey, V8, ActionScript, JScript, QtScript, InScript, Google Apps Script

JScript is Microsoft's legacy dialect of the ECMAScript standard that is used in Microsoft's Internet Explorer 11 and older.

The most popular implementation of ECMAScript is **JavaScript**.

JavaScript was invented by Brendan Eich in 1995, and became an **ECMA standard** in 1997.

ECMAScript Editions

Edition	Date published	Name
1	June 1997	
2	June 1998	
3	December 1999	
4	*Abandoned* (last draft 30 June 2003)	
5	December 2009	
5.1	June 2011	
6	June 2015[10]	ECMAScript 2015 (ES2015)
7	June 2016[11]	ECMAScript 2016 (ES2016)
8	June 2017[12]	ECMAScript 2017 (ES2017)
9	June 2018[13]	ECMAScript 2018 (ES2018)
10	June 2019[14]	ECMAScript 2019 (ES2019)
11	June 2020[15]	ECMAScript 2020 (ES2020)
12	June 2021[16]	ECMAScript 2021 (ES2021)
13	June 2022[17]	ECMAScript 2022 (ES2022)

Hello World! exercise

exercise-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>JavaScript basics</title>
   </head>
   <body>
      <script>
         document.write("Hello World!")
      </script>
   </body>
</html>
```

Printed: Hello World!

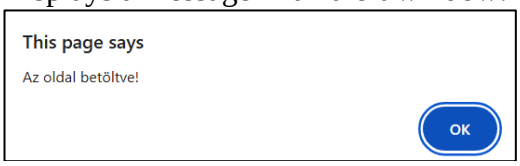
Simple event handling - onload Event

exercise-02.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>JavaScript basics</title>
   <script>
      function message() {
         alert("The page is loaded!");
      }
   </script>
</head>
<body onload="message()">
   Teszt oldal.
</body>
</html>
```

onload Event: executes a script once a web page has completely loaded all content.
https://www.w3schools.com/jsref/event_onload.asp

Displays a message in an alert window:



Then after clicking OK it will say: Test page.

If JavaScript is disabled in the browser

If the browser cannot execute the contents of the script element for some reason (e.g. JavaScript is disabled), the contents of the noscript element will be displayed:

```
<script>
document.write("JS is supported.")
</script>
<noscript>
JavaScript is not supported.
</noscript>
```

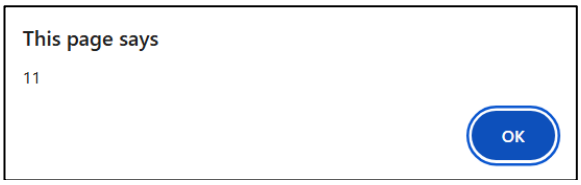
Variables, If condition, Date

exercise-03.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>JavaScript basics</title>
```

```
</head>
<body style="background-color: #eeeeff;">
   <script type="text/javascript">
      var d = new Date()
      var time = d.getHours();
      alert(time);
      if (time < 12)
         document.write("<strong>Good morning!</strong>");
      else
         document.write("<strong>Good afternoon!</strong>");
      var message = "<br />Good " + ((time < 12) ? "morning" : "afternoon") + "!";
      document.write(message);
   </script>
</body>
</html>
```

in Alert window:



Then prints:
Good morning!
Good morning!

Arrays, For loop

exercise-04.html

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8">
	<title>JavaScript basics</title>
</head>
<body style="background-color: #eeeeff;">
	<script type="text/javascript">
		var mycars = new Array()
		var x
		mycars[0] = "Saab"
		mycars[1] = "Volvo"
		mycars[2] = "BMW"
		for (x in mycars)
			document.write(mycars[x] + "<br />")
	</script>
</body>
</html>
```

Kiírja:
Saab
Volvo
BMW

Check it:

Right click on page / View page source
/ Inspect

JavaScript is cached by the browser, so if we change the JavaScript code and refresh the browser, we often do not see the effect of the modification.

Temporarily disable caching in Chrome (while the Inspector is open):
right click on page / Indpect / Network / Disable cache (while DevTools is open)

JavaScript Scope, Block, Function(Local) , Global, var let, const

JavaScript Scope is the area where a variable (or function) exists and is accessible.
Scope determines the usability and visibility of variables. There are 3 types of scope in JavaScript: Block, Function(Local), Global

block: {…}

Var
Variables declared with the var keyword can NOT have block scope.
Var declarations have **global** or **function (local)** scope. The scope is **global** if a var variable is declared outside a function. In this case, the variables can be used throughout the script A var has **function (local)** scope if it is declared inside a function. In this case, it can only be accessed within that function.

local scope:

exercise1:

```
function newFunction() {
    var hello = "hello world";
}
newFunction();
document.write(hello);   // error: it doesn't print anything because hello is not defined.
```

global scope:

exercise2:

```
var hello = "hello world-global";
function newFunction() {
    hello = "hello world";
}
newFunction();
document.write(hello);          // prints: hello world
```

exercise3:

```
var hello = "hello world-global";
function newFunction() {
   var hello = "hello world";
}
newFunction();
document.write(hello);          // prints: hello world-global
```

let
A let scope-ja: block. block: {}
A variable declared with let can only be used within the given block.

exercise1:

```
let greeting = "say Hi";
let times = 4;
if (times > 3) {
    let hello = "say Hello instead";
    document.write(hello);  // "say Hello instead"
}
document.write(hello)       // It doesn't print anything because hello is not defined here.
```

exercise2:

```
let greeting = "say Hi";
if (true) {
    let greeting = "say Hello instead";
    document.write(greeting);        // "say Hello instead"
}
document.write(greeting);            // "say Hi"
```

Const

This is also in block scope, like let scope.
Its value cannot be changed:

```
const greeting = "say Hi";
greeting = "say Hello instead";      // error: Assignment to constant variable
document.write(greeting);            // nem ír ki semmit az előző utasítás hibája miatt
```

Named functions, Anonymous functions and arrow function

Named function

```
function sum1(a,b) {                    // name: sum1
    return a+b;
}
document.write(sum1(5, 3)+" ");         // => 8
```

Anonymous function

```
function() {              // it has no name
    // Function Body
}
```

Defining an anonymous function that prints a message to the console. The function is then stored in the **greet variable**. We can call the function by invoking **greet().**

```
const greet = function () {
    console.log("Hello");
};
greet();
```

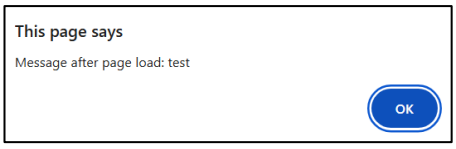
```
var sum2 = function (a,b) {             // sum2: variable
    return a+b;
}
document.write(sum2(5, 3)+" ");         // => 8
```

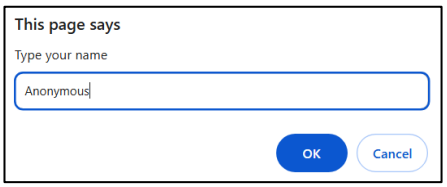
arrow function

```
var sum3 = (a, b) => a+b;
document.write(sum3(5, 3));             // => 8
```

Arrow functions allow you to specify functions more concisely.

Alert box, Prompt box, onclick event





Hello Anonymous!

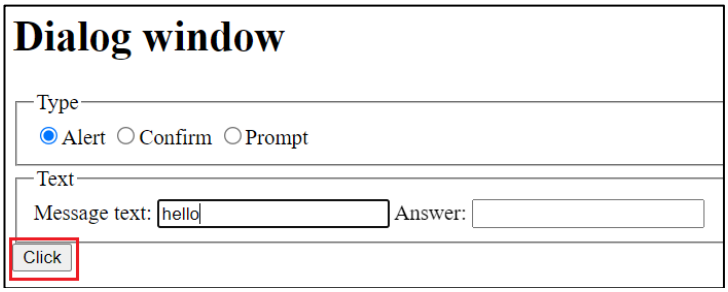
exercise-05.html

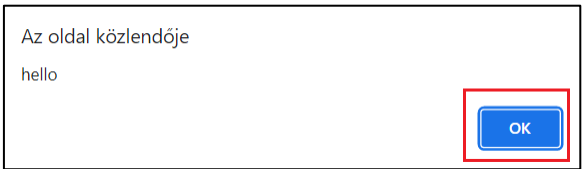
```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>JavaScript basics</title>
   <script type="text/javascript">
      a = "test";
      function message() {
         alert("Message after page load: " + a);
      }
      function display_prompt() {
         var name = prompt("Type your name","Anonymous")
         if (name != null && name != "")
            document.write("Hello " + name + "!")
      }
   </script>
</head>
<body onload="message()">
   <input type="button" onclick="display_prompt()" value="Button">
</body>
</html>
```

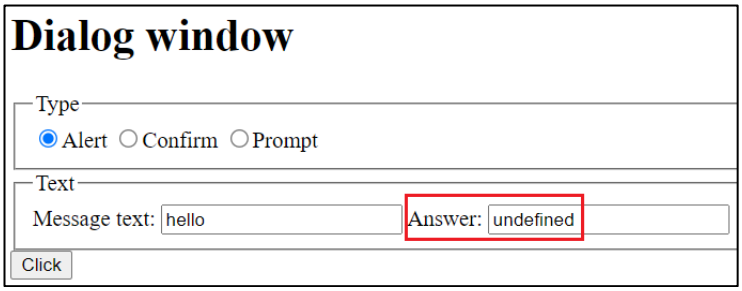
Alert box, Confirm box, Prompt box, switch-case

The Answer text box is read only.

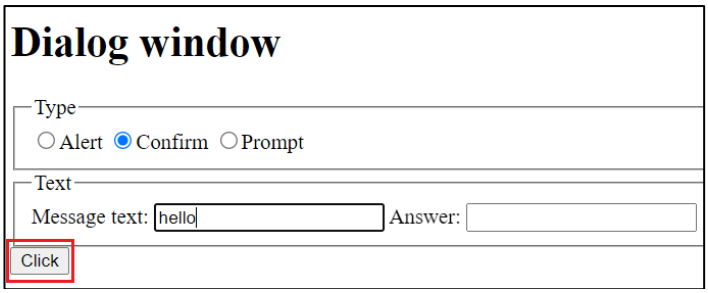
with Alert box

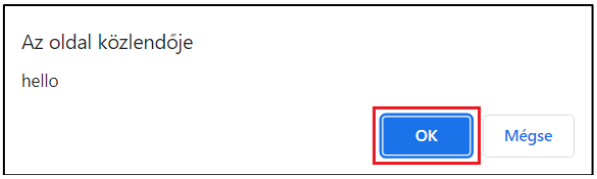


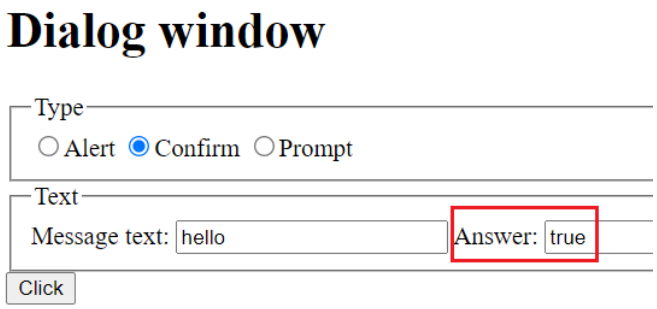




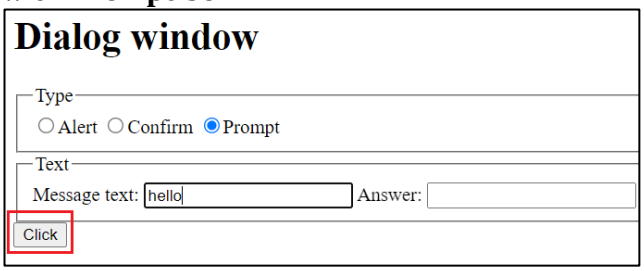
with Confirm box

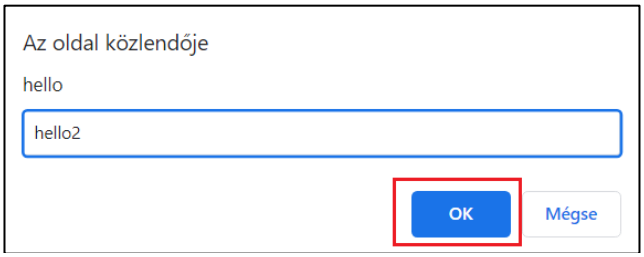


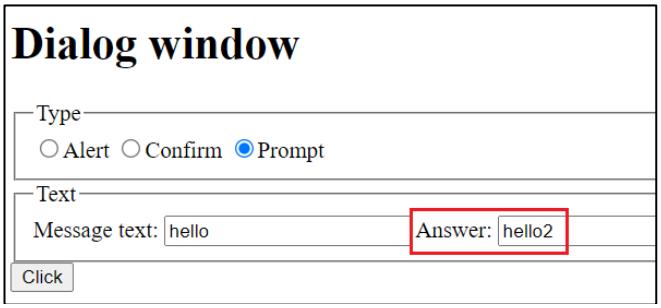




with Prompt box







exercise-06.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
```

```
    <title>JavaScript basics</title>
    <script type="text/javascript">
        function message() {
            switch (true) {
            case document.form1.type1[0].checked:
                document.form1.answer.value = alert(document.form1.text1.value)
                break
            case document.form1.type1[1].checked:
                document.form1.answer.value = confirm(document.form1.text1.value)
                break
            case document.form1.type1[2].checked:
                document.form1.answer.value = prompt(document.form1.text1.value)
                break
            }
        }
    </script>
</head>
<body>
    <h1>Dialog window</h1>
    <form name="form1">
        <fieldset>
            <legend>Type</legend>
            <label><input type="radio" name="type1">Alert</label>
            <label><input checked="checked" type="radio" name="type1">Confirm</label>
            <label><input type="radio" name="type1">Prompt</label>
        </fieldset>
        <fieldset>
            <legend>Text</legend>
            <label>Message text: <input type="text" name="text1"></label>
            <label>Answer: <input type="text" name="answer" readonly></label>
        </fieldset>
        <input onclick="message();" type="button" name="button" value="Click">
    </form>
</body>
</html>
```

getElementById

The **getElementById()** method returns the element with the given ID.

Create the following page:
Open/close the “Work place” box by clicking on the checkbox:

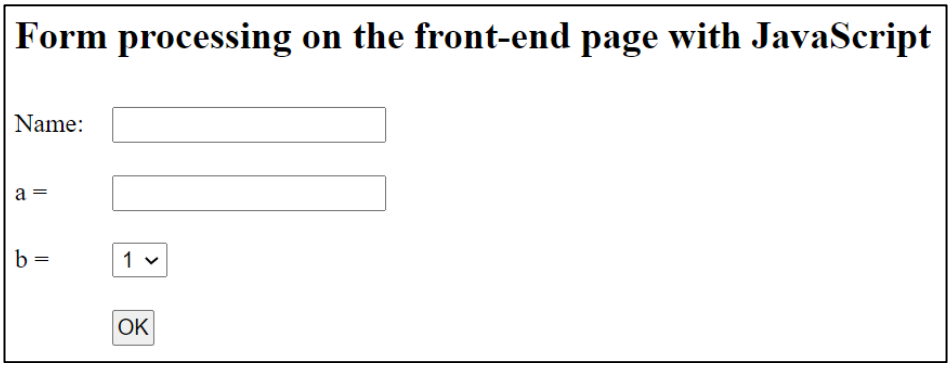
exercise-07.html

```
<!DOCTYPE html>
<html>
```

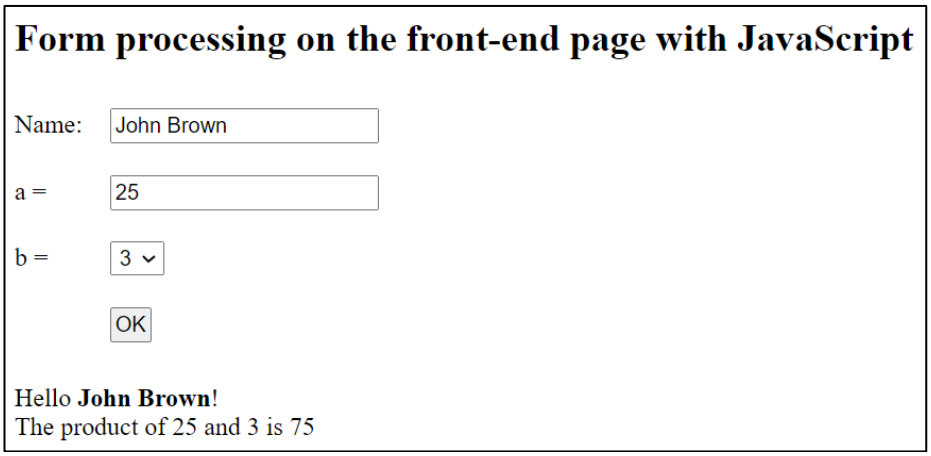
```
<head>
        <meta charset="utf-8">
        <title>JavaScript basic</title>
</head>
<body>
   <h1>Dialog window</h1>
   <input type="checkbox" onclick="document.getElementById('info').style.visibility = this.checked ?
'visible' : 'hidden'"> I work <br>
   <div id="info" style="visibility:hidden">
      Work place: <input name="workplace" type="text">
   </div>
</body>
</html>
```

Form processing with JavaScript

Complete the following task:
- The user must enter a name in the first text box.
- You can only enter a number in the second text box
- You can choose from numbers 1-5 in the drop-down list.



After filling the form and clicking the button:



exercise-07c.html

```
<!DOCTYPE html>
<html>
```

```
<head>
  <meta charset="utf-8">
  <title>form processing</title>
  <style type="text/css">
        label {
                width: 60px;
                display: inline-block;
        }
        input, select {
                 margin: 10px 0;
                 padding: 2px;
        }
  </style>
  <script type="text/javascript">
        function send() {
                var res = document.getElementById('result');
                res.innerHTML = "Hello <strong>" + document.getElementById('name').value +
"</strong>!<br>";
                var number1 = document.getElementById('num1').value;
                var number2 = document.getElementById('num2').value;
                res.innerHTML += ("The product of " + number1 + " and " + number2 + " is " +
number1*number2);
        }
  </script>
</head>
<body>
        <h2>Form processing on the front-end page with JavaScript</h2>
        <label>Name:</label><input type="text" id="name"><br>
        <label>a =</label><input type="text" id="num1"><br>
        <label>b =</label><select id="num2">
                <option value="1">1</option>
                <option value="2">2</option>
                <option value="3">3</option>
                <option value="4">4</option>
                <option value="5">5</option>
        </select><br>
        <label></label><input type="submit" onclick="send()" value="OK">
        <p id="result"></p>
</body>
</html>
```

If we entered restrictive expressions into the INPUT elements, e.g.:
<input type="text" id="num1" pattern="[1-9][0-9]*" required>
they would have no effect here, because the content is not checked when the button is clicked, the onclick="send()" method is executed immediately.

Clicking a button jumps to a given item

exercise-07d.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
```

```
    <title>JavaScript basics</title>
    <script type="text/javascript">
        function send() {
            var elem = document.getElementById("id1");
            elem.innerHTML="Hello";
            elem.scrollIntoView();
        }
    </script>
</head>
<body>
    <input type="submit" onclick="send()" value="Write"><br>
    <script type="text/javascript">
        for(var i= 0; i<=100; i++)
            document.writeln("<p>Text</p>");
    </script>
    <H1 id="id1"></H1>
    <script type="text/javascript">
        for(var i= 0; i<=100; i++)
            document.writeln("<p>Text</p>");
    </script>
</body>
</html>
```

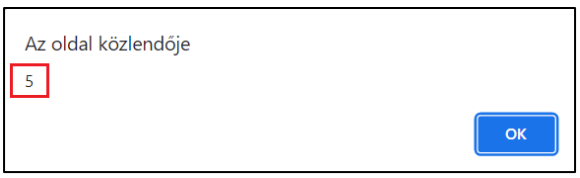
Counts button clicks

exercise-07e.html

```
<html>
<head>
    <title>Button Clicker</title>
</head>
<body>
    <script type="text/javascript">
    var clicks = 0;
    function hello() {
        clicks += 1;
        document.getElementById("clicks").innerHTML = clicks;
    };
    </script>
    <p>Clicks: <a id="clicks">0</a></p>
    <button type="button" onclick="hello()">Click me</button>
</body>
</html>
```

Exercise – 10 clickable buttons, continue

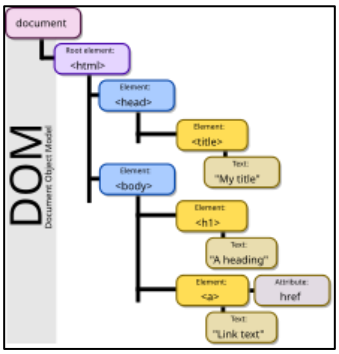
Numbers divisible by 3 should be left out:



exercise-08.html

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>JavaScript basics</title>
</head>
<body>
   <h1>Dynamic buttons</h1>
   <script type="text/javascript">
      for (i = 1; i <= 10; i++) {
         if (i%3 == 0)
            continue;
         document.write("<input type=button value=\"" + i + "\" onclick=\"alert(" + i + ")\"> ")
      }
   </script>
</body>
</html>
```

Document Object Model (DOM)



The Document Object Model (DOM) is a cross-platform and language-independent interface that treats an HTML or XML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects.

We can modify/access the page with JavaScript if it is run after the DOM of the HTML document is created!

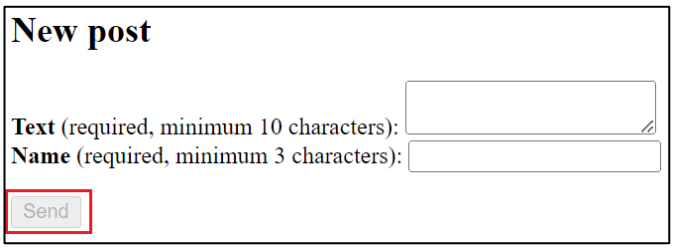
=> … onload="init()" … It runs when the page loads

Accessing elements of the page e.g.: getElementById(…), getElementsByClassName(…)

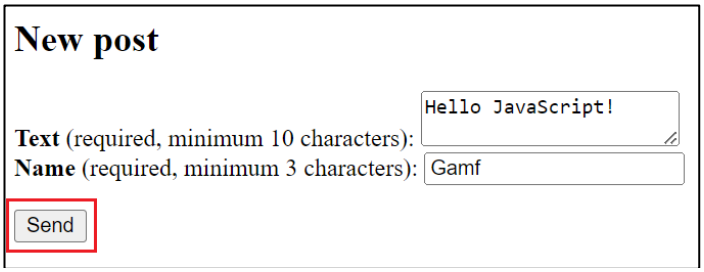
Form validation on client side, addEventListener

The correct completion of the form is first checked on the client side. Only then is it sent to the server side if everything is correct. It is also necessary to check the received data on the server side (server-side validation).

Make the Send button inactive after the page has loaded:



Check each time you type a character to see if the conditions are met. If so, activate the Send button:



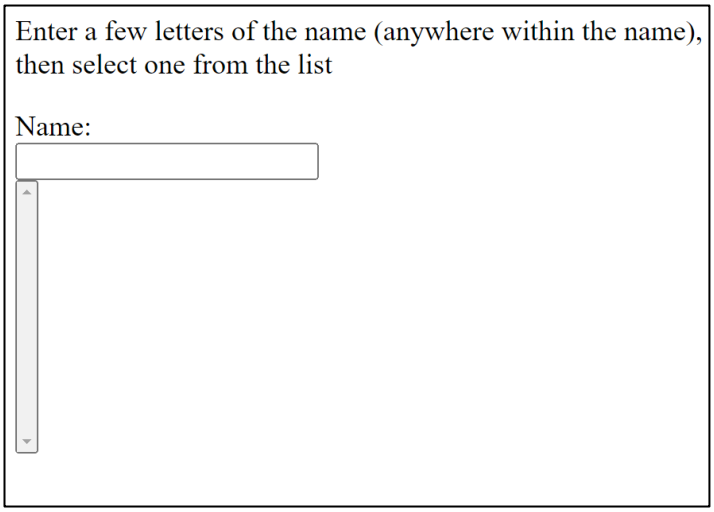
The **addEventListener()** method assigns an event handler to an element.

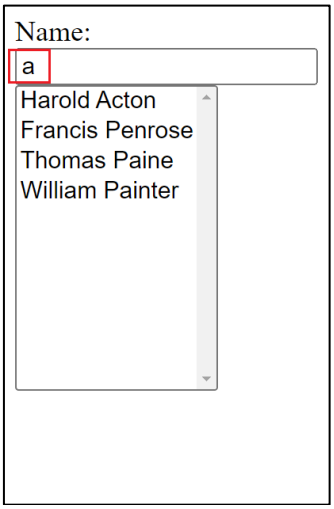
exercise-09.html

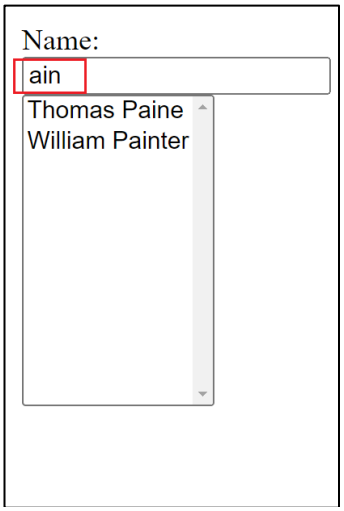
```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>JavaScript basic</title>
   <script type="text/javascript">
      function formCheck() {
         document.getElementById('postsend').disabled = (document.getElementById('text1').value.length
< 10) || (document.getElementById('name1').value.length < 3)
      }

      function init() {
         document.getElementById('postsend').disabled = true;
         document.getElementById('text1').addEventListener('keydown', formCheck);
         document.getElementById('name1').addEventListener('keydown', formCheck);
      }
   </script>
</head>
<body onload="init()">
   <h2>New post</h2>
   <label for="text1"><strong>Text</strong> (required, minimum 10 characters):</label>
   <textarea id="text1" name="text1"></textarea><br>
   <label for="name1"><strong>Name</strong> (required, minimum 3 characters):</label>
   <input id="name1" name="name1" type="text" size="20"><br>
   <p></p>
   <button name="postsend" id="postsend" type="submit">Send</button>
</body>
</html>
```

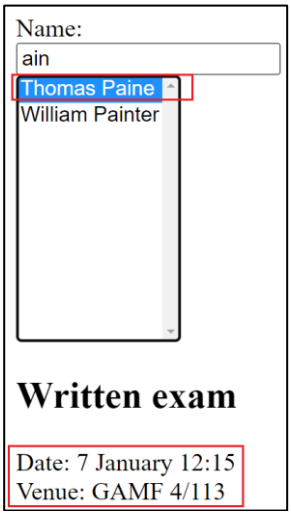
Exercise - nested Array, indexOf - The next task is similar: Table Filter/Search => Only informative text

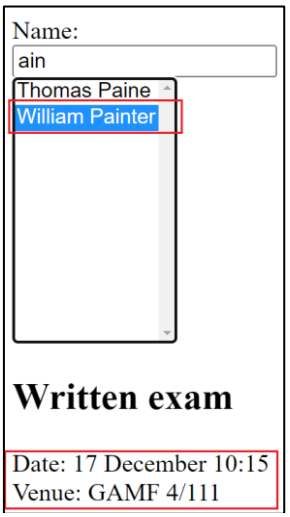
In the program, enter an array of names and the data associated with the names. Enter a few characters of a name (anywhere in the name). Print the names associated with it in a drop-down list. From the printed names, the user chooses one and print the data of the chosen one in a paragraph:











exercise-11.html

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8">
	<title>JavaScript basic</title>
	<script type="text/javascript">
		var examinees = new Array(
				new Array("Harold Acton","15 December","8:45","GAMF 4/8"),
				new Array("Henry Green","5 January","13:30","GAMF 4/109"),
				new Array("Stephen Fry","10 January","11:45","GAMF 4/108"),
				new Array("Royston Ellis","12 January","10:45","GAMF 4/110"),
				new Array("Francis Penrose","3 January","12:30","GAMF 4/113"),
				new Array("Thomas Paine","7 January","12:15","GAMF 4/113"),
				new Array("William Painter","17 December","10:15","GAMF 4/111")
			)

		function typing(){
			// Delete all data
			var writtenexam = document.getElementById('writtenexam');
			writtenexam.style.display = 'none';
			// Objects for typing the name and displaying the list of names (name1 and list1)
			var name1 = document.getElementById('name1');
```

```
                var list1 = document.getElementById('list1');
                // We build the list in the str variable
                var str = '';
                // If the name field is not empty, a list is generated
                if (name1.value.length>0)
                        for (i=0; i<examinees.length; i++) {
                                var exname = examinees[i][0];
                                // If the current name contains the sample text (name1.value), then a
good name has been found
                                if (exname.indexOf(name1.value) != -1)
                                        str += '<option value="o' + i + '">' + exname + '<'+
'/option>\n';
                        }
                // Copy the str to list1.innerHTML:
                list1.innerHTML = str;
        }

        function choose(){
                // The chosen name (in the name1 variable)
                var name1 = list1.options[list1.selectedIndex].text;
                // Find the name in the array
                for (i = 0; i < examinees.length; i++)
                        if (examinees[i][0] == name1)
                                break;
                // The data must be filled in, based on the data of the array
                var idate = document.getElementById('idate');
                idate.innerHTML = examinees[i][1] + " " + examinees[i][2];
                var ivenue = document.getElementById('ivenue');
                ivenue.innerHTML = examinees[i][3];
                // Display the data: makes the DIV block visible
                var writtenexam = document.getElementById('writtenexam');
                writtenexam.style.display = 'block';
        }
    </script>
</head>
<body>
    <p>Enter a few letters of the name (anywhere within the name), <br />then select one from the
list</p>
    Name:<br /><input type="text" id="name1" onKeyUp="typing();"></input>
    <br />
    <select size="10" id="list1" onChange="choose();">
    </select>
    <div id="writtenexam" style="display: none">
        <h2>Written exam</h2>
        <p>
            Date: <span id="idate"></span><br>
            Venue: <span id="ivenue"></span>
        </p>
    </div>
</body>
</html>
```

Add new elements to the DOM, createElement, appendChild

Az oldal kezdetben tartalmazzon egy bekezdést. Az oldal betöltődése után adjunk hozzá még egy bekezdést és egy listát 2 elemmel.

JavaScript nélkül:
Bekezdés1

JavaScript-el:

Bekezdés1

Bekezdés2

- elem1
- elem2

exercise-14.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>exercise</title>
  <script type="text/javascript">
    function add() {
        this.paragraph2 = document.createElement("p");
        this.paragraph2.innerHTML="Paragraph2";
        document.body.appendChild(this.paragraph2);
        this.list1=document.createElement("ul");
        document.body.appendChild(this.list1);
        this.elem=document.createElement("li");
        this.elem.innerHTML="elem1";
        this.list1.appendChild(this.elem);
        this.elem=document.createElement("li");
        this.elem.innerHTML="elem2";
        this.list1.appendChild(this.elem);
    }
// onload event: is activated when the page is fully loaded and the DOM has been built:
    window.onload = function() {
        add();
    };
  </script>
</head>
<body>
  <p>
    Paragraph1
  </p>
</body>
</html>
```

JavaScript is asynchronous

exercise-15.html

```
<!DOCTYPE html>
<html>
<head>
   <title>JavaScript</title>
   <meta charset="utf-8">
   <script type="text/javascript">
      function print2() {
         document.writeln("Hello2<br>");
      }
      function print3() {
         // prints after a 1 second delay:
         setTimeout(function(){
            document.writeln("Hello3<br>");
         }, 1000);
      }
      window.onload = function() {
         document.writeln("Hello1<br>");
         print2();
         print3();
         document.writeln("Hello4<br>");
      }
   </script>
</head>
<body>
</body>
</html>
```

Written in this order:
Hello1
Hello2
Hello4
Hello3

Processing a Table with JavaScript

Sorting a Table

https://www.w3schools.com/howto/howto_js_sort_table.asp

We dont't deal with CSS here.

exercise-16.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Sort a HTML Table Alphabetically</title>
   <style>
      table {
         border-spacing: 0;
         width: 50%;
         border: 1px solid #ddd;
      }
      th, td {
         text-align: left;
         padding: 16px;
      }
      tr:nth-child(even) {
         background-color: #f2f2f2
      }
   </style>
</head>
<body>
   <p>Click the button to sort the table alphabetically, by name:</p>
   <p><button onclick="sortTable()">Sort</button></p>
   <table id="myTable">
      <tr>
         <th>Name</th>
         <th>Country</th>
      </tr>
      <tr>
         <td>Berglunds snabbkop</td>
         <td>Sweden</td>
```

```
    </tr>
    <tr>
      <td>North/South</td>
      <td>UK</td>
    </tr>
    <tr>
      <td>Alfreds Futterkiste</td>
      <td>Germany</td>
    </tr>
    <tr>
      <td>Koniglich Essen</td>
      <td>Germany</td>
    </tr>
    <tr>
      <td>Magazzini Alimentari Riuniti</td>
      <td>Italy</td>
    </tr>
    <tr>
      <td>Paris specialites</td>
      <td>France</td>
    </tr>
    <tr>
      <td>Island Trading</td>
      <td>UK</td>
    </tr>
    <tr>
      <td>Laughing Bacchus Winecellars</td>
      <td>Canada</td>
    </tr>
  </table>
  <script>
    function sortTable() {
      var table, rows, switching, i, x, y, shouldSwitch;
      table = document.getElementById("myTable");
      switching = true;
// It loops through the rows until the rows are sorted.
      while (switching) {
        switching = false;
        rows = table.rows;
// It cheks the rows one by one. If row i is greater than row i+1, it swaps them:
        for (i = 1; i < (rows.length - 1); i++) {
          shouldSwitch = false;
          x = rows[i].getElementsByTagName("TD")[0];
          y = rows[i + 1].getElementsByTagName("TD")[0];
// If the i.th element is greater than the next one, then it must be swapped: shouldSwitch = true;
          if (x.innerHTML.toLowerCase() > y.innerHTML.toLowerCase()) {
            shouldSwitch = true;
// Exits the for loop:
            break;
          }
        }
// If it needs to be swapped:
        if (shouldSwitch) {
```

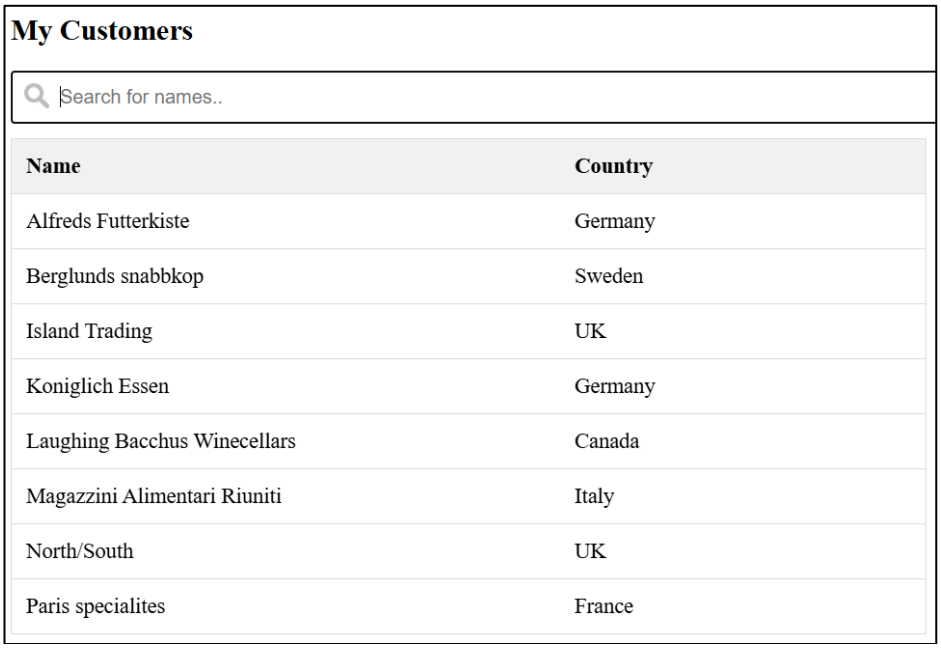
```
// Inserts the i+1st element before the ith element:
                rows[i].parentNode.insertBefore(rows[i + 1], rows[i]);
                switching = true;
              }
            }
        }
    </script>
</body>
</html>
```

insertBefore(newNode, referenceNode)

inserts a node before a reference node as a child of a specified parent node.
If the given node already exists in the document, insertBefore() moves it from its current position to the new position.
https://developer.mozilla.org/en-US/docs/Web/API/Node/insertBefore

Table Filter/Search

https://www.w3schools.com/howto/howto_js_filter_table.asp



We dont't deal with CSS here.

exercise-17.html

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
        * {
            box-sizing: border-box;
        }
        #myInput {
            background-image: url('searchicon.png');
            background-position: 10px 10px;
```

```
        background-repeat: no-repeat;
        width: 100%;
        font-size: 16px;
        padding: 12px 20px 12px 40px;
        border: 1px solid #ddd;
        margin-bottom: 12px;
      }
      #myTable {
        border-collapse: collapse;
        width: 50%;
        border: 1px solid #ddd;
        font-size: 18px;
      }
      #myTable th, #myTable td {
        text-align: left;
        padding: 12px;
      }
      #myTable tr {
        border-bottom: 1px solid #ddd;
      }
      #myTable tr.header, #myTable tr:hover {
        background-color: #f1f1f1;
      }
    </style>
</head>
<body>
    <h2>My Customers</h2>
    <input type="text" id="myInput" onkeyup="myFunction()" placeholder="Search for names.."
title="Type in a name">
    <table id="myTable">
      <tr class="header">
        <th style="width:60%;">Name</th>
        <th style="width:40%;">Country</th>
      </tr>
      <tr>
        <td>Alfreds Futterkiste</td>
        <td>Germany</td>
      </tr>
      <tr>
        <td>Berglunds snabbkop</td>
        <td>Sweden</td>
      </tr>
      <tr>
        <td>Island Trading</td>
        <td>UK</td>
      </tr>
      <tr>
        <td>Koniglich Essen</td>
        <td>Germany</td>
      </tr>
      <tr>
        <td>Laughing Bacchus Winecellars</td>
        <td>Canada</td>
```

```
      </tr>
      <tr>
        <td>Magazzini Alimentari Riuniti</td>
        <td>Italy</td>
      </tr>
      <tr>
        <td>North/South</td>
        <td>UK</td>
      </tr>
      <tr>
        <td>Paris specialites</td>
        <td>France</td>
      </tr>
    </table>
    <script>
      function myFunction() {
        var input, filter, table, tr, td, i, txtValue;
        input = document.getElementById("myInput");
        filter = input.value.toUpperCase();
        table = document.getElementById("myTable");
// collects the rows of the table into a list:
        tr = table.getElementsByTagName("tr");
// goes through all lines in a cycle:
        for (i = 0; i < tr.length; i++) {
// looks for <td>, and not <th>
          td = tr[i].getElementsByTagName("td")[0];
// it it <td>?:
          if (td) {
            //txtValue = td.innerText;
            txtValue = td.textContent;
// if the specified text is found in the row, it displays the row, otherwise it hides it:
            if (txtValue.toUpperCase().indexOf(filter) > -1)
              tr[i].style.display = "";
            else
              tr[i].style.display = "none";
          }
        }
      }
    </script>
</body>
</html>
```

Solution without FOR loop with FILTER function – Only informative text

A,

```
      function myFunction() {
        var input, filter, table, trs, tds, filtered;
        input = document.getElementById("myInput");
        filter = input.value.toUpperCase();
        table = document.getElementById("myTable");
        trs = table.getElementsByTagName("tr");
        tds = [].filter.call(trs, el => el.getElementsByTagName('td').length > 0 );
```

```
        filtered = [].filter.call(tds, el =>
el.getElementsByTagName('td')[0].textContent.toUpperCase().indexOf(filter) >= 0 );
        Array.from(trs).map(tr=>{
            if(!filtered.includes(tr))
                tr.style.display = "none";
        });
    }
```

B,

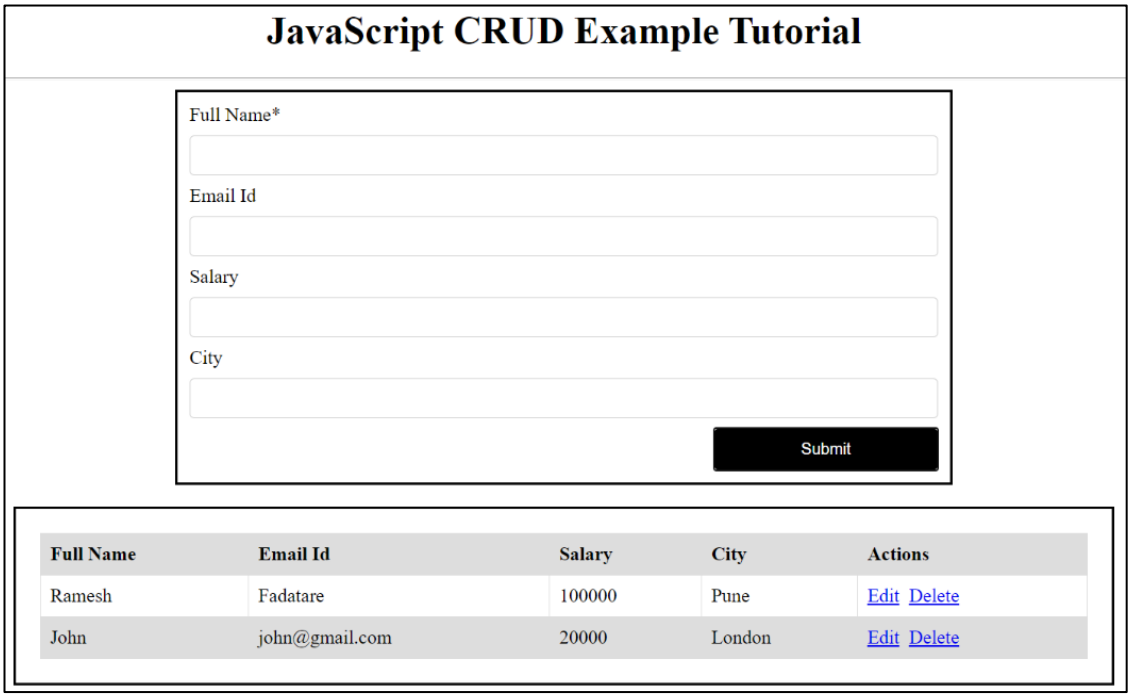
```
    function Filt(tr) {
        var input, filter;
        input = document.getElementById("myInput");
        filter = input.value.toUpperCase();
        if(tr.getElementsByTagName('td').length > 0)
            if(tr.getElementsByTagName('td')[0].textContent.toUpperCase().indexOf(filter) >= 0)
                return tr;
    }
    function myFunction3() {
        var table, tr, td, i, txtValue;
        table = document.getElementById("myTable");
        trs = table.getElementsByTagName("tr");
        let filtered = Object.values(trs).filter(el=>Filt(el));
        Array.from(trs).map(tr=>{
            if(!filtered.includes(tr))
                tr.style.display = "none";
        });
    }
```

CRUD application – with an Array

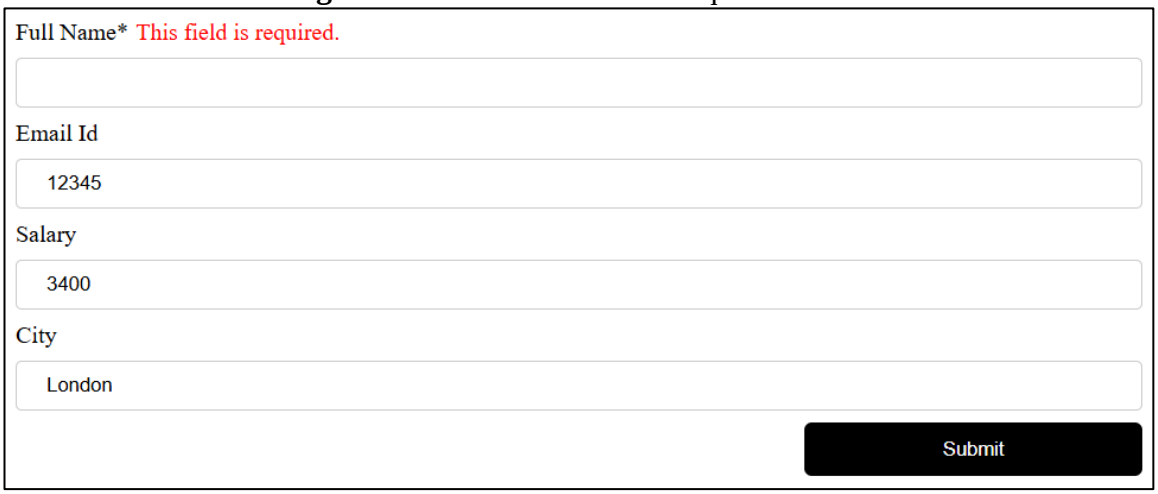
https://www.freecodecamp.org/news/learn-crud-operations-in-javascript-by-building-todo-app/
https://www.javaguides.net/2020/11/javascript-crud-example-tutorial.html

Create the following CRUD application. **CRUD**: Create, Read, Update, Delete

exercise-18.html, style.css, script.js



Validation error message: The Full Name field is required.



We store the data in an array.
There may be identical records in the array. If we want to avoid this, we need to add a unique ID to each element.

exercise-18.html

```
<!DOCTYPE html>
<html>
<head>
   <title>
      JavaScript CRUD Example Tutorial
   </title>
   <link rel="stylesheet" href="style.css">
</head>
<body>
   <h1><center>JavaScript CRUD Example Tutorial</center></h1>
   <hr>
   <div class="employee-form">
// preventDefault(): disables the default event handler assigned to the button click on the form.
```

```
//        prevents page reloading
// When the button is clicked, it runs the JS onFormSubmit() function:
      <form onsubmit="event.preventDefault();onFormSubmit();">
         <div>
// Displays the error message This field is required in case of validation error
//        By default the following is hidden: <label class="validation-error hide"…
//        check: script.js: validate() method
            <label>Full Name*</label><label class="validation-error hide"
id="fullNameValidationError">This field is required.</label>
            <input type="text" name="fullName" id="fullName">
         </div>
         <div>
            <label>Email Id</label>
            <input type="text" name="email" id="email">
         </div>
         <div>
            <label>Salary</label>
            <input type="text" name="salary" id="salary">
         </div>
         <div>
            <label>City</label>
            <input type="text" name="city" id="city">
         </div>
         <div  class="form-action-buttons">
            <input type="submit" value="Submit">
         </div>
      </form>
   </div>
   <br/>
// Creates the table structure:
   <div class = "employees-table">
         <table class="list" id="employeeList">
            <thead>
               <tr>
                  <th>Full Name</th>
                  <th>Email Id</th>
                  <th>Salary</th>
                  <th>City</th>
                  <th>Actions</th>
               </tr>
            </thead>
            <tbody>

            </tbody>
         </table>
   </div>
   <script src="script.js"></script>
</body>
</html>
```

script.js

```
var selectedIndex = null;
```

```
var array1 = new Array();      // We store the data in this array
// If the user fills out the form and clicks the Submit button:
function onFormSubmit() {
// validate(): checks that the form has been filled out correctly
   if (validate()) {
// Reads the data from the input fields and returns it in the formData list.
      var formData = readFormData();
// If selectedIndex == null: insert the new record after clicking the button
//         otherwise modify the selected record.
//         selectedIndex is null by default, will get a value on Update
      if (selectedIndex==null)
         insertNewRecord(formData);
      else
         updateRecord(formData);
      resetForm();
   }
}

// Reads the data from the input fields and returns it in a list
function readFormData() {
   var formData = {};
   formData["fullName"] = document.getElementById("fullName").value;
   formData["email"] = document.getElementById("email").value;
   formData["salary"] = document.getElementById("salary").value;
   formData["city"] = document.getElementById("city").value;
   return formData;
}

// Inserts the new row at the end of the array and puts two links at the end of the row: Edit, Delete
//         data: form data in a list
function insertNewRecord(data) {
// Put the new record at the end of the array:
   array1[array1.length]=
{"fullName":data.fullName,"email":data.email,"salary":data.salary,"city":data.city};
// Print the array:
   printArray();
}

// Print the array into the table below:
function printArray(){
   var table = document.getElementById("employeeList").getElementsByTagName('tbody')[0];
// Empty the existing table:
   table.innerHTML="";
   var newRow;
   for (i = 0; i < array1.length; i++) {
// Inserts a new row into the table:
      newRow = table.insertRow(table.length);
// Inserts an empty cell into the new row:
      cell1 = newRow.insertCell(0);
// Inserts the fullName data from the array into the cell:
      cell1.innerHTML = array1[i].fullName;
      cell2 = newRow.insertCell(1);
      cell2.innerHTML = array1[i].email;
```

```
        cell3 = newRow.insertCell(2);
        cell3.innerHTML = array1[i].salary;
        cell4 = newRow.insertCell(3);
        cell4.innerHTML = array1[i].city;
        cell4 = newRow.insertCell(4);
// adds two links to the end of the row: Edit, Delete
//         Pass the index of the selected row as a parameter to the onEdit and onDelete functions:
        cell4.innerHTML = '<a onClick="onEdit('+i+')">Edit</a>' + '<a
onClick="onDelete('+i+')">Delete</a>';
    }
}

// Clears input fields. e.g. call it after submitting data.
function resetForm() {
    document.getElementById("fullName").value = "";
    document.getElementById("email").value = "";
    document.getElementById("salary").value = "";
    document.getElementById("city").value = "";
    selectedIndex=null;
}

// The selected record is loaded into the top form for editing:
function onEdit(index) {
    document.getElementById("fullName").value = array1[index].fullName;
    document.getElementById("email").value = array1[index].email;
    document.getElementById("salary").value = array1[index].salary;
    document.getElementById("city").value = array1[index].city;
    selectedIndex=index;
}

// Modifies the selected record with the retrieved form data:
//         Called on Update
function updateRecord(formData) {
    array1[selectedIndex].fullName=formData.fullName;
    array1[selectedIndex].email=formData.email;
    array1[selectedIndex].salary=formData.salary;
    array1[selectedIndex].city=formData.city;
    printArray();
}

function onDelete(index) {
    if (confirm('Are you sure to delete this record ?')) {
        array.splice(index, 1); // Deleting the entry with the specified index
        resetForm();
        printArray();
    }
}

// validation: The Full Name field is required.
function validate() {
    isValid = true;
// If the Full Name field is empty (validation error):
    if (document.getElementById("fullName").value == "") {
```

```
        isValid = false;
// Displays the hidden error message:
        document.getElementById("fullNameValidationError").classList.remove("hide");
    } else {
// if there is no validation error (Full Name field is filled in)
        isValid = true;
// If the validation error message is not hidden, hide it:
        if (!document.getElementById("fullNameValidationError").classList.contains("hide"))
            document.getElementById("fullNameValidationError").classList.add("hide");
    }
    return isValid;
}
```

We dont't deal with CSS here.

style.css

```
.employee-form {
    border-style: solid;
    /* margin-bottom: 10px; */
    /* margin-left: 10px; */
    padding: 10px;
    /* width: 50%; */
    margin: auto;
    width: 50%;
    /* border: 3px solid green; */
    /* padding: 10px; */
}
.employees-table {
    border-style: solid;
    /* margin-bottom: 10px; */
    /* margin-left: 10px; */
    padding: 20px;
    /* width: 50%; */
    margin: auto;
    width: 70%;
    /* border: 3px solid green; */
    /* padding: 10px; */
}
body > table{
    width: 80%;
}
table{
    border-collapse: collapse;
}
table.list{
    width:100%;
}
td, th {
    border: 1px solid #dddddd;
    text-align: left;
    padding: 8px;
}
tr:nth-child(even),table.list thead>tr {
```

```
    background-color: #dddddd;
}
input[type=text], input[type=number] {
    width: 100%;
    padding: 8px 20px;
    margin: 8px 0;
    display: inline-block;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}
input[type=submit] {
    width: 30%;
    background-color: black;
    color: white;
    padding: 10px 18px;
    /* margin: 0px 0; */
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
form div.form-action-buttons{
    text-align: right;
}
a{
    cursor: pointer;
    text-decoration: underline;
    color: #0000ee;
    margin-right: 4px;
}
label.validation-error{
    color:   red;
    margin-left: 5px;
}
.hide{
    display:none;
}
```

CRUD application – without Array - With operations on the table – Only informative text

The HTML and CSS files are the same as in the previous case.

script.js

```
var selectedRow = null
// If the user fills out the form and clicks the Submit button:
function onFormSubmit() {
// validate(): checks the correct completion of the form
    if (validate()) {
// Reads the data from the input fields and returns it in the formData list:
        var formData = readFormData();
// If selectedRow == null: insert the new record after clicking the button
//         otherwise modify the selected record.
```

```
//          selectedRow is null by default, and will get a value on Update
        if (selectedRow == null)
// // Inserts the new row at the end of the table and adds two links at the end of the row: Edit, Delete
            insertNewRecord(formData);
        else
            updateRecord(formData);
// Clears the form data:
        resetForm();
    }
}
// Reads the data from input fields and returns it in a list.
function readFormData() {
    var formData = {};
    formData["fullName"] = document.getElementById("fullName").value;
    formData["email"] = document.getElementById("email").value;
    formData["salary"] = document.getElementById("salary").value;
    formData["city"] = document.getElementById("city").value;
    return formData;
}
// Inserts the new row at the end of the table and adds two links at the end of the row: Edit, Delete
//          data: form data in a list
function insertNewRecord(data) {
    var table = document.getElementById("employeeList").getElementsByTagName('tbody')[0];
// Inserts a new row at the end of the table:
    var newRow = table.insertRow(table.length);
// Inserts an empty cell into the new row:
    cell1 = newRow.insertCell(0);
// Inserts the fullName data from the form into the cell
    cell1.innerHTML = data.fullName;
    cell2 = newRow.insertCell(1);
    cell2.innerHTML = data.email;
    cell3 = newRow.insertCell(2);
    cell3.innerHTML = data.salary;
    cell4 = newRow.insertCell(3);
    cell4.innerHTML = data.city;
    cell4 = newRow.insertCell(4);
// /adds two links to the end of the line: Edit, Delete
//          this: indicates the link that the user clicked on
    cell4.innerHTML = `<a onClick="onEdit(this)">Edit</a>
                <a onClick="onDelete(this)">Delete</a>`;
}
// Clears input fields. e.g. call it after submitting data
function resetForm() {
    document.getElementById("fullName").value = "";
    document.getElementById("email").value = "";
    document.getElementById("salary").value = "";
    document.getElementById("city").value = "";
    selectedRow = null;
}
// Reloads the selected row data into the input fields.
//          It helps Update to re-load the original values
//          td: represents the link that was clicked. Check: insertNewRecord
function onEdit(td) {
```

```
// td.parentElement.parentElement: the selected row, because cell4 is the parent of link(td)
//         and newRow is the parent of cell4. Check: insertNewRecord
    selectedRow = td.parentElement.parentElement;
// The form is filled with the data of the selected row
    document.getElementById("fullName").value = selectedRow.cells[0].innerHTML;
    document.getElementById("email").value = selectedRow.cells[1].innerHTML;
    document.getElementById("salary").value = selectedRow.cells[2].innerHTML;
    document.getElementById("city").value = selectedRow.cells[3].innerHTML;
}
// Update the selected record with the given form data as paramether:
//         Called on Update
function updateRecord(formData) {
    selectedRow.cells[0].innerHTML = formData.fullName;
    selectedRow.cells[1].innerHTML = formData.email;
    selectedRow.cells[2].innerHTML = formData.salary;
    selectedRow.cells[3].innerHTML = formData.city;
}
function onDelete(td) {
// After confirmation, deletes the selected row:
    if (confirm('Are you sure to delete this record ?')) {
// as at onEdit: td.parentElement.parentElement: the selected row
        row = td.parentElement.parentElement;
// Deletes the row with the given index from the table.
        document.getElementById("employeeList").deleteRow(row.rowIndex);
        resetForm();
    }
}
// Validation: The Full Name field is required.
function validate() {
    isValid = true;
// If the Full Name field is empty (validation error):
    if (document.getElementById("fullName").value == "") {
        isValid = false;
// Displays the hidden error message:
        document.getElementById("fullNameValidationError").classList.remove("hide");
    } else {
// if there is no validation error (the Full Name field is filled in)
        isValid = true;
// If the validation error message is not hidden, hide it:
        if (!document.getElementById("fullNameValidationError").classList.contains("hide"))
            document.getElementById("fullNameValidationError").classList.add("hide");
    }
    return isValid;
}
```

5 - Other new features of HTML5 and ChartJS

Web Storage – better than cookies

Disadvantages and Limitations of Cookies

- They go with every HTML message, slowing down communication
- Cookies go unencrypted in messages
- The size of the cookie is limited (4 kB)

Two new mechanisms in HTML5

- Session storage
- Local storage

Their use is justified in different situations
Modern browsers all support

Session Storage

The user performs a simple process in a browser window (there can be multiple processes in different windows)
sessionStorage variable:

- stores data in a session
- once the user closes the window, its contents are lost

The next example counts the number of times the user has refreshed the page in the current session. Closing and reopening the browser tab restarts the page count from 1.

01-sessionstorage/index.html

```
<!DOCTYPE HTML>
<html>
<body>
	<script type="text/javascript">
		if( sessionStorage.hits )
		  sessionStorage.hits = Number(sessionStorage.hits) +1;
		else
		  sessionStorage.hits = 1;
		document.write("Total Hits :" + sessionStorage.hits );
	</script>
	<p>Refresh the page to increase number of hits.</p>
	<p>Close the window and open it again and check the result.</p>
</body>
</html>
```

Local Storage

localStorage variable:

- the stored data is shared by multiple open windows/browser tabs
- its value remains even after the window is closed
- no size limit (we can even store user mailboxes or documents)

Sensitive data should not be stored for long periods

- delete a variable: localStorage.remove('hits')
- delete all variables: localStorage.clear()

Like the previous exercise, but closing and reopening the browser tab will continue the calculation from where it left off (the previous state is preserved)

02-localstorage/index.html

```
<!DOCTYPE HTML>
<html>
<body>
	<script type="text/javascript">
		if( localStorage.hits )
			localStorage.hits = Number(localStorage.hits) +1;
		else
			localStorage.hits = 1;
		document.write("Total Hits :" + localStorage.hits );
	</script>
	<p>Refresh the page to increase number of hits.</p>
	<p>Close the window and open it again and check the result.</p>
</body>
</html>
```

Web Workers – Running JavaScript code in the background - Parallel programming with JavaScript

JavaScript is designed to run single-threaded
Execution that takes longer (e.g. long cycles) will block the browser

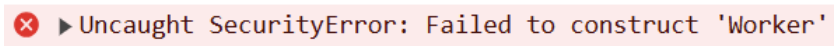
03-web_workers/index-without-web-worker.html

```
<!DOCTYPE HTML>
<html>
<head>
<title>Big for loop</title>
 <script>
    function bigLoop(){
      for (var i = 0; i <= 10000000000; i += 1)
        var j = i;
      alert("Completed " + j + "iterations" );
    }
    function sayHello(){
      alert("Hello sir...." );
    }
 </script>
</head>
<body>
  <input type="button" onclick="bigLoop();" value="Big Loop">
  <input type="button" onclick="sayHello();" value="Say Hello">
</body>
</html>
```

A web server is also required.

Due to security concerns, access to local files from browsers is restricted. Web browsers (and JavaScript) can only access local files **with user permission**.

right click on page / Inspect / Console



You have already learned server-side programming in Seminar. Both solutions are good:
A, with PHP Development Server:
in the folder where the html file is, at the command line:
\xampp\php\php -S localhost:80
http://localhost/index-with-web-worker.html
B, with XAMPP

03-web_workers / bigLoop.js

```
for (var i = 0; i <= 1000000000; i += 1)
  var j = i;
postMessage(j);        // post a message back to the HTML page.
```

Since workers run in a separate thread from the one that created them, communication needs to happen via postMessage. The **postMessage()** method of the Worker interface sends a message to the worker's inner scope. This accepts a single parameter, which is the data to send to the worker.

03-web_workers / index-with-web-worker.html

```
<!DOCTYPE HTML>
<html>
	<head>
		<meta charset="utf-8">
		<title>Web Worker</title>
		<script>
			function bigLoop(){
				if (typeof(Worker) !== "undefined") {
					var worker = new Worker('bigLoop.js');
					worker.onmessage = function (event) {
						alert("Completed " + event.data + "iterations" );
					};
				} else {
					alert("Sorry, your browser does not support Web Workers..." );
				}
			}
			function sayHello(){
				alert("Hello...." );
			}
		</script>
	</head>
	<body>
		<input type="button" onclick="bigLoop();" value="Big Loop">
		<input type="button" onclick="sayHello();" value="Say Hello">
	</body>
</html>
```

The Web Worker can be stopped from the script that started it using the worker.terminate() method.

Server Sent Events - SSE – It improves the limitation of AJAX: always the client initiates there

We will learn Ajax later.

The limitation of AJAX: always the client initiates
In the case of multi-player events (chat, games), the server cannot initiate the sending
Solutions with traditional technology:

- **polling technique** (JS timer and AJAX): retrieval per time unit; displaying "newness"
  Disadvantage: a lot of unnecessary server load; opening-closing a new connection every time
- **long polling**: when the request arrives: the server only sends a response if there is new data, otherwise it waits to send back

Real solutions: WebSocket and SSE

WebSocket:

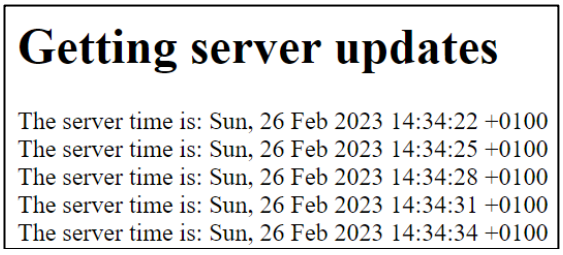
- does not use HTTP protocol

SSE:

- uses HTTP protocol
- completely handled by the browser
- easy to program on the server side
- does not generate unnecessary load during creation-destruction

You have already learned server-side programming in Seminar. Both solutions are good:
A, with PHP Development Server:
in the folder where the html file is, at the command line:
\xampp\php\php -S localhost:80
http://localhost/index-with-web-worker.html
B, with XAMPP

Exercise-1



……

demo_sse.php

```
<?php
    header('Content-Type: text/event-stream');
    header('Cache-Control: no-cache');
    $time = date('r');
    echo "data: The server time is: {$time}\n\n";
    flush();
```

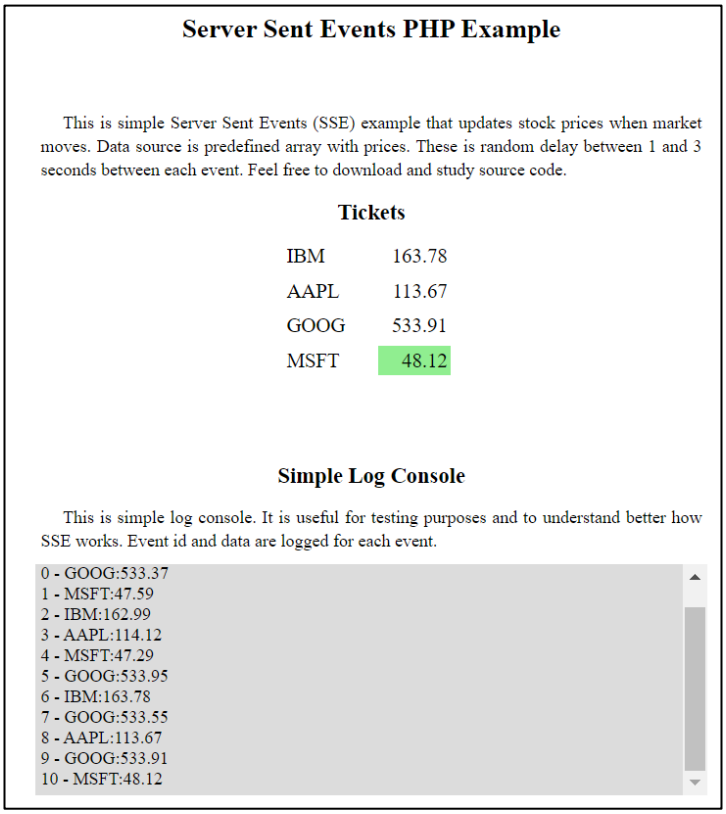
```
?>
```

index.html

```
<!DOCTYPE html>
<html>
<body>
	<h1>Getting server updates</h1>
	<div id="result"></div>
	<script>
		if(typeof(EventSource) !== "undefined") {
			var source = new EventSource("demo_sse.php");
			source.onmessage = function(event) {
				document.getElementById("result").innerHTML += event.data + "<br>";
			};
		} else
			document.getElementById("result").innerHTML = "Sorry, your browser does not support server-sent events...";
	</script>
</body>
</html>
```

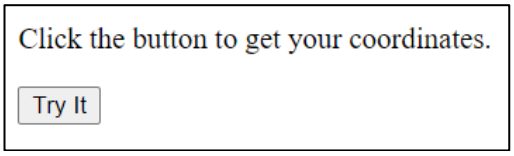
Exercise-2 – We just test it, we don't analyze the code

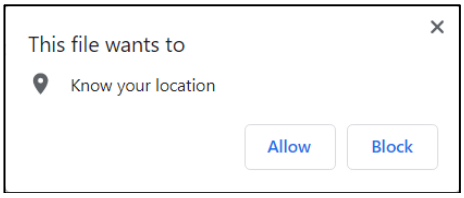
The page updates the stock price when it changes:

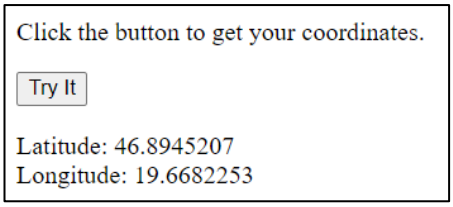


Geolocation API

- Querying the user's geographic location
- Personal data, can only be retrieved with the user's consent



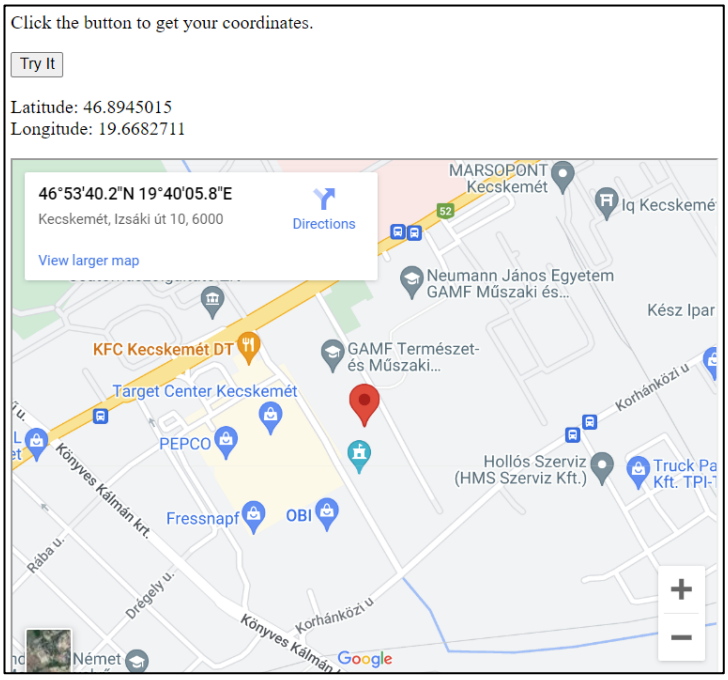




05-geolocation / index.html

```
<!DOCTYPE html>
<html>
<body onload="myFunction()">
	<p>Click the button to get your coordinates.</p>
	<button onclick="getLocation()">Try It</button>
	<p id="demo"></p>
	<script>
		var x = document.getElementById("demo");
		function getLocation()  {
			if (navigator.geolocation)
				navigator.geolocation.getCurrentPosition(showPosition);
			else
				x.innerHTML = "Geolocation is not supported by this browser.";
		}
		function showPosition(position)  {
			x.innerHTML = "Latitude: " + position.coords.latitude + "<br>Longitude: " +
position.coords.longitude;
		}
	</script>
</body>
</html>
```

Positions inserted into Google maps – Only informative text – we try it, but we don't analyze the code



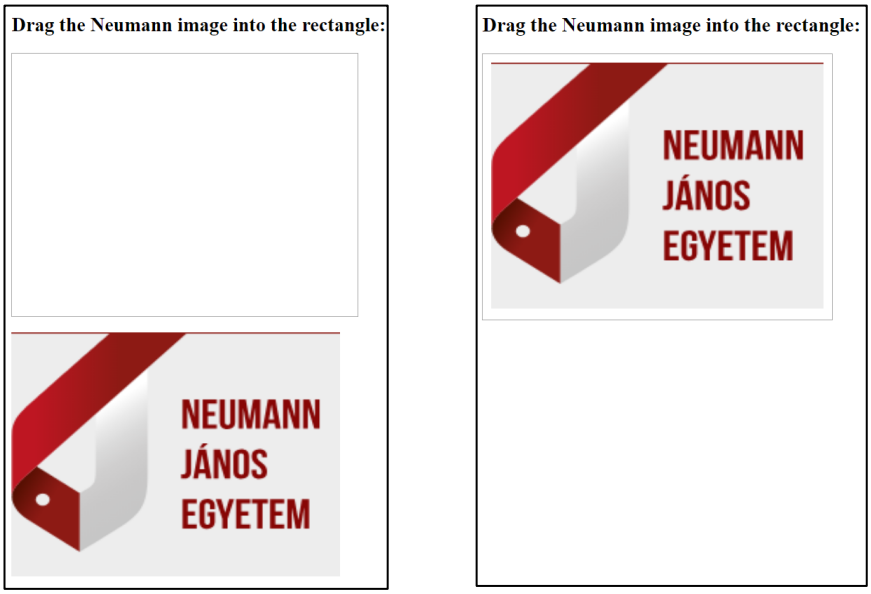
05-geolocation / index2.html

```
<!DOCTYPE html>
<html>
<body onload="myFunction()">
	<p>Click the button to get your coordinates.</p>
	<button onclick="getLocation()">Try It</button>
	<p id="demo"></p>
	<script>
		var x = document.getElementById("demo");
		function getLocation()  {
			if (navigator.geolocation)
				navigator.geolocation.getCurrentPosition(showPosition);
			else
				x.innerHTML = "Geolocation is not supported by this browser.";
		}
		function showPosition(position)  {
			x.innerHTML = "Latitude: " + position.coords.latitude + "<br>Longitude: " +
position.coords.longitude;
			var newContent = '<iframe src = "https://maps.google.com/maps?q=' +
position.coords.latitude + ',' + position.coords.longitude + '&hl=es;z=14&amp;output=embed"
width="600" height="450"></iframe>';
			var contentHolder = document.getElementById('content-holder');
			contentHolder.innerHTML = newContent;
		}
	</script>
	<p id="content-holder">Google maps: Waiting for GPS coordinates ...</p>
</body>
</html>
```

Additional methods:
watchPosition() – returns the position continuously (e.g. for a program running on moving vehicles)
clearWatch() – stops watchPosition()

Drag and drop API

Exercise-1



- img
  - draggable="true": make the element draggable
  - ondragstart="drag(event)": Call the drag(event) function when dragging
- div
  - ondragover: allows other elements to be dropped here
  - ondrop: specifies what happens when dropped

index.html

```
<!DOCTYPE HTML>
<html>
<head>
	<style>
		#div1 {width:400px;height:300px;padding:10px;border:1px solid #aaaaaa;}
	</style>
	<script>
// A drop engedélyezéséhez megakadályozzuk az elem alapértelmezett kezelését.
//	Enélkül nem működik a drop:
//	https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/drop_event
		function allowDrop(ev) {
			ev.preventDefault();
		}
		function drag(ev) {
// The dataTransfer.setData() method sets the data type and value of the transferred data.
//	data type of ev.target.id: "text"
//	ev.target.id: ID of the dragged item
			ev.dataTransfer.setData("text", ev.target.id);
		}
		function drop(ev) {
			ev.preventDefault();
// dataTransfer.getData(): gets the transferred data
			var data = ev.dataTransfer.getData("text");
// appending the transferred element to the target element in the DOM:
```

```
                ev.target.appendChild(document.getElementById(data));
            }
        </script>
</head>
<body>
        <h2>Drag the Neumann image into the rectangle:</h2>
// ondragover: we allow other elements to be dropped here
//        By default, data/elements cannot be dropped into other elements.
// ondrop: we specify what happens when dropped
        <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
        <br>
// draggable="true": Make the element draggable
// ondragstart="drag(event)": when dragging, we call the drag(event) function
        <img id="drag1" src="nje.png" draggable="true" ondragstart="drag(event)" width="400"
height="300">
</body>
</html>
```

Exercise-2 - Back and forth – Only informative text – we try it, but we don't analyze the code

code: back-and-forth.html

Canvas – Graphics with JavaScript

Creating two-dimensional graphics and animations with JavaScript

```
<canvas id="mycanvas" width="100" height="100"></canvas>
```

Checking browser support, extracting drawing environment

```
var canvas =  document.getElementById("mycanvas");
if (canvas.getContext){
  var ctx = canvas.getContext('2d');
} else { ... // unsupported canvas }
```

The <canvas> element is just a container for the graphics. We need to use JavaScript to actually draw the graphics.

Draw the Canvas frame:

01-canvas.html

```
<!DOCTYPE HTML>
<html>
  <head>
```

```
    <style>
      #mycanvas{border:1px solid red;}
    </style>
  </head>
  <body>
    <canvas id="mycanvas" width="100" height="100"></canvas>
  </body>
</html>
```

Draw rectangles and squares

02-rectangles.html

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      #test {width: 100px; height:100px; margin: 0px auto;}
    </style>
    <script type="text/javascript">
      function drawShape(){
        var canvas = document.getElementById('mycanvas'); // Get the canvas element using the DOM
        if (canvas.getContext){ // Make sure we don't execute when canvas isn't supported
          var ctx = canvas.getContext('2d'); // use getContext to use the canvas for drawing
// The fillRect() method draws a "filled" rectangle. The default color of the fill is black.
          ctx.fillRect(25,25,100,100); // Draw shapes
// The clearRect() method clears the specified pixels within a given rectangle.
          ctx.clearRect(45,45,60,60);
// The strokeRect() method draws a rectangle (no fill). The default color of the stroke is black.
          ctx.strokeRect(50,50,50,50);
        }
        else
          alert('Your browser doesn\'t support Canvas.');
      }
    </script>
  </head>
  <body id="test" onload="drawShape();">
    <canvas id="mycanvas"></canvas>
  </body>
</html>
```

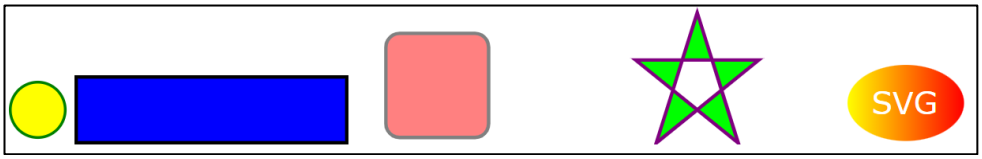
További Canvas példák – Csak olvasmány

A **Megoldások.zip** fájlban talál további példákat.

SVG - Graphics with HTML – without JavaScript

SVG: Scalable Vector Graphics. SVG defines vector-based graphics in XML format.

Let's draw some shapes with SVG:



svg.html

```
<!DOCTYPE html>
<html>
<body>
	<svg width="100" height="100">
	  <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />
	</svg>
	<svg width="400" height="100">
	  <rect width="400" height="100" style="fill:rgb(0,0,255);stroke-width:10;stroke:rgb(0,0,0)" />
	</svg>
	<svg width="400" height="180">
	  <rect x="50" y="20" rx="20" ry="20" width="150" height="150"
style="fill:red;stroke:black;stroke-width:5;opacity:0.5" />
	</svg>
	<svg width="300" height="200">
	  <polygon points="100,10 40,198 190,78 10,78 160,198" style="fill:lime;stroke:purple;stroke-
width:5;fill-rule:evenodd;" />
	</svg>
	<svg height="130" width="500">
		<defs>
		 <linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%">
			<stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
			<stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
		 </linearGradient>
		</defs>
		<ellipse cx="100" cy="70" rx="85" ry="55" fill="url(#grad1)" />
		<text fill="#ffffff" font-size="45" font-family="Verdana" x="50" y="86">
		 SVG
		</text>
		Sorry, your browser does not support inline SVG.
	</svg>
</body>
</html>
```

SVG vs Canvas

- **SVG**: language for describing 2D graphics in XML
  SVG is XML-based, all its elements are available in the SVG DOM; JavaScript event handlers can be assigned to the elements
  In SVG, the drawn shape is an object, the appearance of which is immediately modified by the browser if its attributes are changed
  Supports event handlers
- **Canvas**: draws 2D graphics using JS
  Canvas: display per pixel; the browser “does not know” the elements; for modification, the objects (and what they cover) must be redrawn
  Does not support event handlers

SVG	Canvas
SVG uses geometric shapes to render graphics	Canvas uses pixels
Vector based (composed of shapes)	Raster based (composed of pixel)
SVG has better scalability. So it can be printed with high quality at any resolution.	Canvas has poor scalability. Hence it is not suitable for printing on higher resolution.
SVG gives better performance with smaller number of objects or larger surface.	Canvas gives better performance with smaller surface or larger number of objects.
SVG can be modified through script and CSS.	Canvas can be modified through script only.
Multiple graphical elements, which become the part of the page's DOM tree.	Single element similar to <img> in behavior. Canvas diagram can be saved to PNG or JPG format.

ChartJS – Drawing graphs with Canvas and JavaScript

https://www.chartjs.org/
https://www.chartjs.org/docs/latest/samples/information.html
Many types of graphs can be drawn.

Készítsük el a következő 2 adatsorból álló vonaldiagramot:

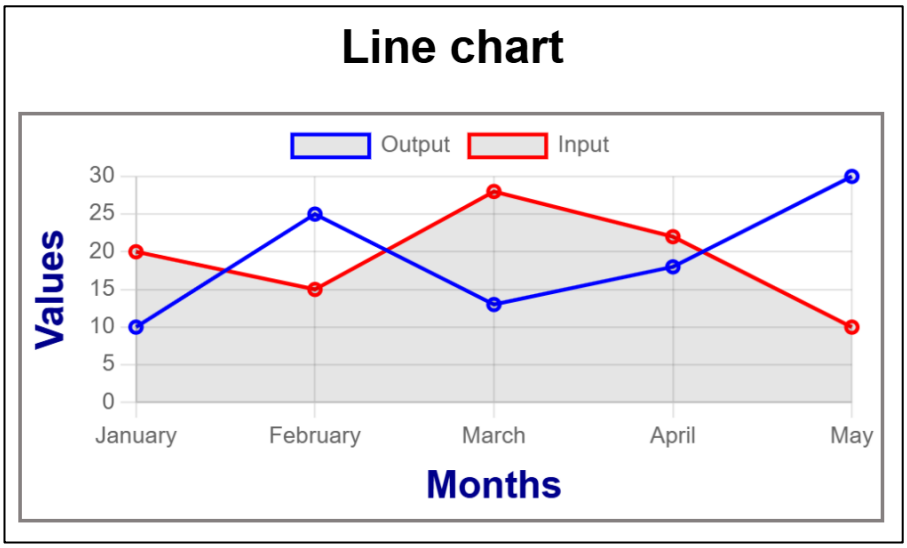


chart1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Chart.js Line Chart</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            margin: 20px;
            text-align: center;
        }
        h1 {
            color: green;
        }
        canvas {
            border: 2px solid #858080;
        }
    </style>
</head>
<body>
    <h2>Line chart</h2>
    <canvas id="myLineChart" width="380" height="180"></canvas>
    <script>
        // data for showing the line chart
        let labels = ['January', 'February', 'March', 'April', 'May'];
        let dataset1Data = [10, 25, 13, 18, 30];
        let dataset2Data = [20, 15, 28, 22, 10];

        // Creating line chart
        let ctx = document.getElementById('myLineChart').getContext('2d');
        let myLineChart = new Chart(ctx, {
            type: 'line',
            data: {
                labels: labels,
                datasets: [
                    {
                        label: 'Output',
                        data: dataset1Data,
                        borderColor: 'blue',
                        borderWidth: 2,
                        fill: false,
                    },
                    {
                        label: 'Input',
                        data: dataset2Data,
                        borderColor: 'red',
                        borderWidth: 2,
                        fill: true,
                    },
                ]
            },
            options: {
                responsive: true,
                scales: {
                    x: {
```

```
                title: {
                    display: true,
                    text: 'Months',
                    font: {
                        padding: 4,
                        size: 20,
                        weight: 'bold',
                        family: 'Arial'
                    },
                    color: 'darkblue'
                }
            },
            y: {
                title: {
                    display: true,
                    text: 'Values',
                    font: {
                        size: 20,
                        weight: 'bold',
                        family: 'Arial'
                    },
                    color: 'darkblue'
                },
                beginAtZero: true,
                scaleLabel: {
                    display: true,
                    labelString: 'Values',
                }
            }
        }
    }
  });
</script>
</body>
</html>
```

Chart2 – Only informative text

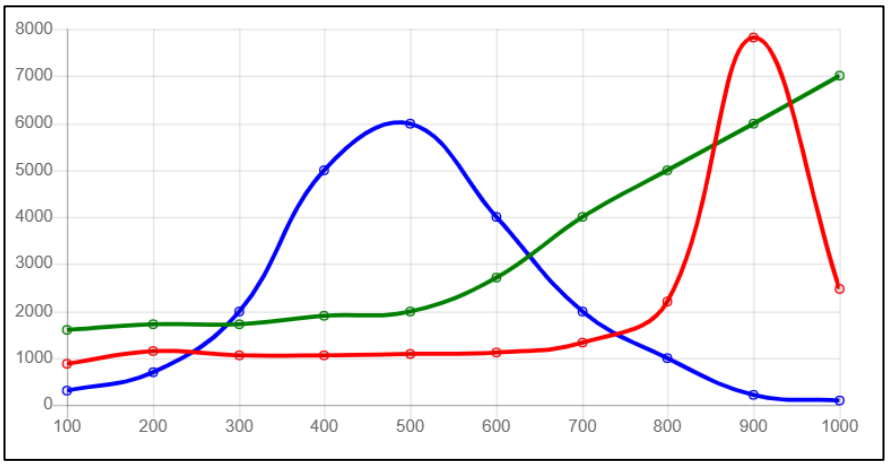


chart2.html

```
<!DOCTYPE html>
<html>
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script>
<body>
<canvas id="myChart" style="width:100%;max-width:600px"></canvas>

<script>
const xValues = [100,200,300,400,500,600,700,800,900,1000];

new Chart("myChart", {
  type: "line",
  data: {
    labels: xValues,
    datasets: [{
      data: [860,1140,1060,1060,1070,1110,1330,2210,7830,2478],
      borderColor: "red",
      fill: false
    }, {
      data: [1600,1700,1700,1900,2000,2700,4000,5000,6000,7000],
      borderColor: "green",
      fill: false
    }, {
      data: [300,700,2000,5000,6000,4000,2000,1000,200,100],
      borderColor: "blue",
      fill: false
    }]
  },
  options: {
    legend: {display: false}
  }
});
</script>
```

6 – AJAX - Asynchronous JavaScript And XML

AJAX: Asynchronous JavaScript And XML.

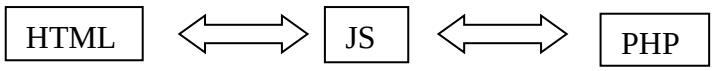
https://www.w3schools.com/js/js_ajax_intro.asp
https://en.wikipedia.org/wiki/Ajax_(programming)

AJAX is a developer's dream, because you can:

- Read data from a web server - after the page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background

We don't reference the PHP file in the HTML file: we will only do so in the Javascript file!

- There is no direct connection between HTML and PHP at all
  JS communicates with both HTML and PHP separately
- JS acts as a controller (after the page loads)
  JavaScript is playing an increasingly important role these days.



JSON introduction – Only informative text

We will see that when making an AJAX call, the data is transferred between the parties in JSON format {…}. We do not need to know the JSON format because the protocol solves it in the background, so the following is Only informative text:

JSON:
JSON (JavaScript Object Notation) is a lightweight data-interchange format. **It is easy for humans to read and write. It is easy for machines to parse and generate.** It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely **language independent** but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.
JSON is built on two structures:
A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.
https://www.json.org/json-en.html

A JSON example:

```
{
  "first_name": "John",
  "last_name": "Smith",
  "is_alive": true,
  "age": 27,
  "address": {
```

```
    "street_address": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postal_code": "10021-3100"
  },
  "phone_numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Thomas",
    "Trevor"
  ],
  "spouse": null
}
```


The AJAX exercise - CRUD

The following page contains a web service (API) with the following properties:

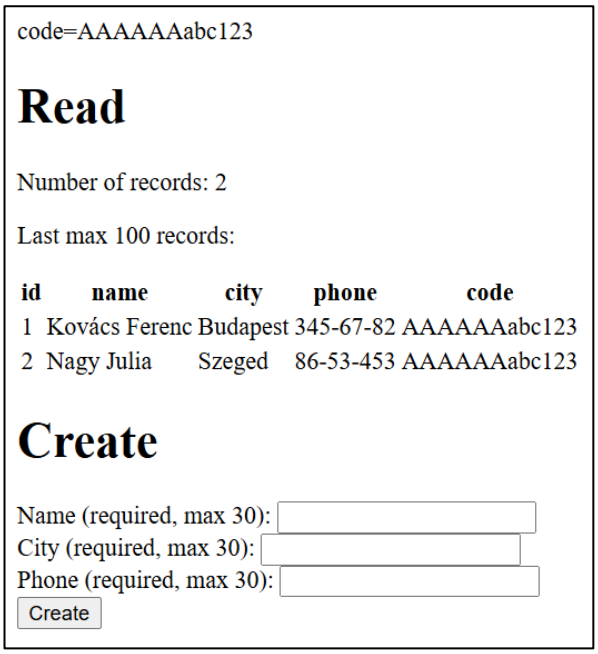
http://gamf.nhely.hu/ajax1/

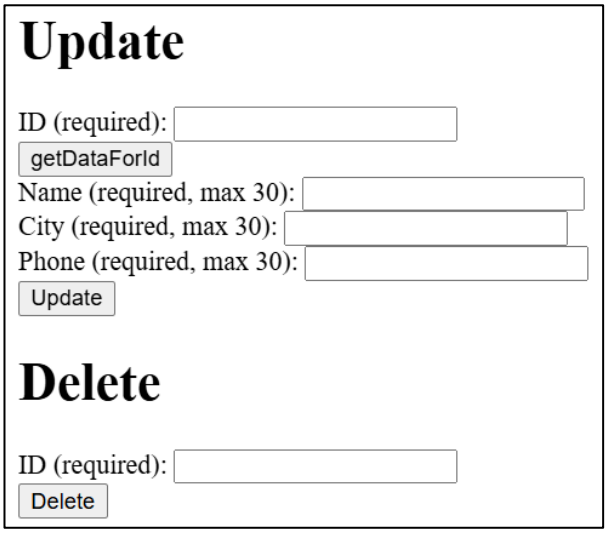
- The API reads/writes from the database in the background.
- It implements CRUD operations: Create, Read, Update, Delete
- It accepts POST requests with parameters
- All data in the table is of type String (VARCHAR).
- For each request, the code parameter must be sent, which identifies the user
  The code parameter should be in the form: **AAAAAAefg456**
  where AAAAAA is the student's Neptun code, efg456: a few-character code that the student invents and writes after the Neptun code, so only he/she knows the code. For the following requests, it only allows access to the rows belonging to the student's code
- For op=read and code parameters:
  Queries the data of the database table.
  Returns the following data in an array in JSON format:
    "rowCount" is the number of records
    "maxNum" is the maximum number of records passed, which is 100
    "list" => array() is the records in a list in key-value pairs:
      keys: id, name, city, phone, code
- op= create and name, city, phone, code parameters:
  Inserts a record formed from the first 100 characters of the name, city, phone, code parameters into the table.
  Returns how many records the statement affected (0 or 1)
- op= update with id, name, city, phone, code:
  Modifies the record with the given id and code with the new values provided
  Returns how many records the statement affected (0 or 1)
- op= delete with id, code:
  Deletes the record with the given id and code

Returns how many records the statement affected (0 or 1)

Create an AJAX application using the web-service API and display the data on the website.
The following files need to be created: **ajax.html, ajax.js**
In the create and update functions, the input fields cannot be empty and can contain a maximum of 30 characters. Check these in the JS file (validation).
In the Update section, there should be an input field where the user enters the ID, and a getDataForId button, which will first read the data into the input fields, where we can modify them.
Provide feedback on the success of the Create, Update, Delete operations on the website.

The appearance of the page:





Testing the web-service API with cURL – 5 minutes – Before development, we check if the web service works

On the command line:
You need to sort it into a single line first!

A, READ
curl -X POST "http://gamf.nhely.hu/ajax1/" -H "Content-Type: application/x-www-form-urlencoded" -d "code=AAAAAAabc123&op=read"

```
curl -X POST "http://gamf.nhely.hu/ajax1/" -H "Content-Type: application/x-www-form-urlencoded" -d "code=AAAAAAabc123&op=read"
```

Response:
{"list":[{"id":1,"name":"Kov\u00e1cs Ferenc","city":"Budapest","phone":"345-67-82","code":"AAAAAAabc123"},{"id":2,"name":"Nagy Julia","city":"Szeged","phone":"86-53-453","code":"AAAAAAabc123"}],"rowCount":2,"maxNum":100}

B, CREATE
curl -X POST "http://gamf.nhely.hu/ajax1/" -H "Content-Type: application/x-www-form-urlencoded" -d "code=AAAAAAabc123&op=create&name=Nagy Ilona&city=Debrecen&phone=70/345-12-46"

Response:

1

If we run Read again, we can see that it has added the new record.

id	name	city	phone	code
1	Kovács Ferenc	Budapest	345-67-82	AAAAAAabc123
2	Nagy Julia	Szeged	86-53-453	AAAAAAabc123
33	Nagy Ilona	Debrecen	70/345-12-46	AAAAAAabc123

Now the new record was id=33.

C, UPDATE
Change the City field of record ID=33 to Miskolc:

curl -X POST "http://gamf.nhely.hu/ajax1/" -H "Content-Type: application/x-www-form-urlencoded" -d "code=AAAAAAabc123&op=update&id=33&name=Nagy Ilona&city=Miskolc&phone=70/345-12-46"

Response:
1

If we run Read again, we can see that the record is updated.

D, DELETE

Delete the record with ID=33:

curl -X POST "http://gamf.nhely.hu/ajax1/" -H "Content-Type: application/x-www-form-urlencoded" -d "code=AAAAAAabc123&**op=delete**&id=33"

Response:
1

If we run Read again, we can see that the record is deleted.

Solution

The HTML file for the following 2 solutions is the same:

ajax.html (there is nothing new in it)

```
<!DOCTYPE HTML>
<html>
 <head>
 <meta charset="utf-8">
 <script type="text/javascript" src = "ajax.js"></script>
 <title>Ajax</title>
 </head>
 <body>
  <div id = 'code'></div>
  <div id = 'readDiv'></div>
  <div id = 'createDiv'>
   <h1>Create</h1>
   Name (required, max 30): <input id="name1" type="text"></input><br>
```

```
    City (required, max 30): <input id="city1" type="text"></input><br>
    Phone (required, max 30): <input id="phone1" type="text"></input><br>
    <button type="submit" onclick="create();">Create</button><br>
    <div id = 'createResult'></div>
   </div>
   <div id = 'updateDiv'>
    <h1>Update</h1>
    ID (required): <input id="idUpd" type="text"></input><br>
    <button type="submit" onclick="getDataForId();">getDataForId</button><br>
    Name (required, max 30): <input id="name2" type="text"></input><br>
    City (required, max 30): <input id="city2" type="text"></input><br>
    Phone (required, max 30): <input id="phone2" type="text"></input><br>
    <button type="submit" onclick="update();">Update</button><br>
    <div id = 'updateResult'></div>
   </div>
   <div id = 'deleteDiv'>
    <h1>Delete</h1>
    ID (required): <input id="idDel" type="text"></input><br>
    <button type="submit" onclick="deleteF();">Delete</button><br>
    <div id = 'deleteResult'></div>
   </div>
  </body>
</html>
```

We study two solutions:

- with the newer Fetch API (from 2015)
- with the older XMLHttpRequest (XHR) JavaScript class (from 1999) – Only informative text

Solution – 1 – with the newer Fetch API

async, await, fetch

async: async makes a function return a **Promise**

The keyword **async before a function** makes the function return a **promise**

https://www.w3schools.com/js/js_async.asp

Promise: https://www.w3schools.com/js/js_promise.asp

A **Promise** is an Object that links **Producing code** and **Consuming code**

"I **Promise** a Result!"

"**Producing code**" is code that can take some time

"**Consuming code**" is code that must wait for the result

You must use a Promise method to handle promises.

fetch: elhoz

https://www.w3schools.com/jsref/api_fetch.asp

The fetch() method starts the process of fetching a resource from a server.

The fetch() method returns a **Promise** that resolves to a Response object.

await: makes a function wait for a Promise

can be used with asynchronous operation (JavaScript is asynchronous)

The **await** keyword can only be used inside an **async** function.

https://www.w3schools.com/js/js_async.asp

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await

The **await** keyword makes the function pause the execution and wait for a resolved promise before it continues.

When sending using the GET method, we pass the parameters in the URL
When sending using the POST method, we send this in the body:

ajax.js

```
code="AAAAAAabc123";
url="http://gamf.nhely.hu/ajax1/";
async function read() {
  document.getElementById("code").innerHTML="code="+code;
  let response = await fetch(url, {
      method: 'post',
      cache: 'no-cache',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
      body: "code="+code+"&op=read"
  });
  let data = await response.text();
  data = JSON.parse(data);
  let list = data.list;
  // !!! We first create it as a String and only add it to the DOM at the end: divRead.innerHTML=str;
  str="<H1>Read</H1>";
  str+="<p>Number of records: "+data.rowCount+"</p>";
  str+="<p>Last max "+data.maxNum+" records:</p>";
  str+="<table><tr><th>id</th><th>name</th><th>city</th><th>phone</th><th>code</th></tr>";
  for(let i=0; i<list.length; i++)
    str +=
"<tr><td>"+list[i].id+"</td><td>"+list[i].name+"</td><td>"+list[i].city+"</td><td>"+list[i].phone+"</td
><td>"+list[i].code+"</td></tr>";
  str +="</table>";
  document.getElementById("readDiv").innerHTML=str;
}

async function create(){
  // name: reserved word
  nameStr = document.getElementById("name1").value;
  city = document.getElementById("city1").value;
  phone = document.getElementById("phone1").value;
  if(nameStr.length>0 && nameStr.length<=30 && city.length>0 && city.length<=30 &&
phone.length>0 && phone.length<=30 && code.length<=30){
    let response = await fetch(url, {
      method: 'post',
      cache: 'no-cache',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
      body: "code="+code+"&op=create&name="+nameStr+"&city="+city+"&phone="+phone
    });
    let data = await response.text();
    if(data>0)
      str="Create successful!";
    else
    str="Create NOT successful!";
```

```
    document.getElementById("createResult").innerHTML=str;
    document.getElementById("name1").value="";
    document.getElementById("city1").value="";
    document.getElementById("phone1").value="";
    read();
  }
  else
    document.getElementById("createResult").innerHTML="Validation error!!";
}

async function getDataForId() {
  let response = await fetch(url, {
      method: 'post',
      cache: 'no-cache',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
// Using the already used op=read request, we can extract the data associated with the given ID:
      body: "code="+code+"&op=read"
  });
  let data = await response.text();
  data = JSON.parse(data);
  let list = data.list;
  for(let i=0; i<list.length; i++)
// select the record whose ID matches the given ID,
//         and enter the data into the input fields:
    if(list[i].id==document.getElementById("idUpd").value){
      document.getElementById("name2").value=list[i].name;
      document.getElementById("city2").value=list[i].city;
      document.getElementById("phone2").value=list[i].phone;
    }
}

async function update(){
  // name: reserved word
  id = document.getElementById("idUpd").value;
  nameStr = document.getElementById("name2").value;
  city = document.getElementById("city2").value;
  phone = document.getElementById("phone2").value;
  if(id.length>0 && id.length<=30 && nameStr.length>0 && nameStr.length<=30 && city.length>0 &&
city.length<=30 && phone.length>0 && phone.length<=30 && code.length<=30){
    let response = await fetch(url, {
      method: 'post',
      cache: 'no-cache',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
      body:
"code="+code+"&op=update&id="+id+"&name="+nameStr+"&city="+city+"&phone="+phone
    });
    let data = await response.text();
    if(data>0)
      str="Update successful!";
```

```
    else
    str="Update NOT successful!";
    document.getElementById("updateResult").innerHTML=str;
// at the end we empty the input fields:
    document.getElementById("idUpd").value="";
    document.getElementById("name2").value="";
    document.getElementById("city2").value="";
    document.getElementById("phone2").value="";
    read();
  }
  else
    document.getElementById("updateResult").innerHTML="Validation error!!";
}

//delete: resetved word
async function deleteF(){
  id = document.getElementById("idDel").value;
  if(id.length>0 && id.length<=30){
    let response = await fetch(url, {
      method: 'post',
      cache: 'no-cache',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
      body: "code="+code+"&op=delete&id="+id
    });
    let data = await response.text();
    if(data>0)
      str="Delete successful!";
    else
    str="Delete NOT successful!";
    document.getElementById("deleteResult").innerHTML=str;
    document.getElementById("idDel").value="";
    read();
  }
  else
    document.getElementById("deleteResult").innerHTML="Validation error!!";
}

window.onload = function() {
    read();
};
```

Solution – 2 - with the older XMLHttpRequest (XHR) JavaScript class – Only informative text

ajax.js

```
code="AAAAAAabc123";
url="http://gamf.nhely.hu/ajax1/";
var xmlHttp=new XMLHttpRequest();
function read() {
  document.getElementById("code").innerHTML="code="+code;
  xmlHttp.open("POST",url,true);
```

```
  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
  var params = "code="+code+"&op=read";
  xmlHttp.onreadystatechange = () => {
    if(xmlHttp.readyState == 4 && xmlHttp.status == 200) {
      let data = xmlHttp.responseText;
      data = JSON.parse(data);
      let list = data.list;
      str="<H1>Read</H1>";
      str+="<p>Number of records: "+data.rowCount+"</p>";
      str+="<p>Last max "+data.maxNum+" records:</p>";
      str+="<table><tr><th>id</th><th>name</th><th>city</th><th>phone</th><th>code</th></tr>";
      for(let i=0; i<list.length; i++)
        str +=
"<tr><td>"+list[i].id+"</td><td>"+list[i].name+"</td><td>"+list[i].city+"</td><td>"+list[i].phone+"</td
><td>"+list[i].code+"</td></tr>";
      str +="</table>";
      document.getElementById("readDiv").innerHTML=str;
    }
  };
  xmlHttp.send(params);
}

function create(){
  // name: reserved word
  nameStr = document.getElementById("name1").value;
  city = document.getElementById("city1").value;
  phone = document.getElementById("phone1").value;
  if(nameStr.length>0 && nameStr.length<=30 && city.length>0 && city.length<=30 &&
phone.length>0 && phone.length<=30 && code.length<=30){
    xmlHttp.open("POST",url,true);
    xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    var params = "code="+code+"&op=create&name="+nameStr+"&city="+city+"&phone="+phone;
    xmlHttp.onreadystatechange = () => {
      if(xmlHttp.readyState == 4 && xmlHttp.status == 200) {
        let data = xmlHttp.responseText;
        if(data>0)
          str="Create successful!";
        else
        str="Create NOT successful!";
        document.getElementById("createResult").innerHTML=str;
        document.getElementById("name1").value="";
        document.getElementById("city1").value="";
        document.getElementById("phone1").value="";
        read();
      }
    };
    xmlHttp.send(params);
  }
  else
    document.getElementById("createResult").innerHTML="Validation error!!";
}

function getDataForId() {
```

```
  xmlHttp.open("POST",url,true);
  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
  var params = "code="+code+"&op=read";
  xmlHttp.onreadystatechange = () => {
    if(xmlHttp.readyState == 4 && xmlHttp.status == 200) {
      let data = xmlHttp.responseText;
      data = JSON.parse(data);
      let list = data.list;
      for(let i=0; i<list.length; i++)
        if(list[i].id==document.getElementById("idUpd").value){
          document.getElementById("name2").value=list[i].name;
          document.getElementById("city2").value=list[i].city;
          document.getElementById("phone2").value=list[i].phone;
        }
    }
  };
  xmlHttp.send(params);
}

function update(){
  // name: reserved word
  id = document.getElementById("idUpd").value;
  nameStr = document.getElementById("name2").value;
  city = document.getElementById("city2").value;
  phone = document.getElementById("phone2").value;
  if(id.length>0 && id.length<=30 && nameStr.length>0 && nameStr.length<=30 && city.length>0 &&
city.length<=30 && phone.length>0 && phone.length<=30 && code.length<=30){
    xmlHttp.open("POST",url,true);
    xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    var params =
"code="+code+"&op=update&id="+id+"&name="+nameStr+"&city="+city+"&phone="+phone;
    xmlHttp.onreadystatechange = () => {
      if(xmlHttp.readyState == 4 && xmlHttp.status == 200) {
        let data = xmlHttp.responseText;
        if(data>0)
          str="Update successful!";
        else
        str="Update NOT successful!";
        document.getElementById("updateResult").innerHTML=str;
        document.getElementById("idUpd").value="";
        document.getElementById("name2").value="";
        document.getElementById("city2").value="";
        document.getElementById("phone2").value="";
        read();
      }
    };
    xmlHttp.send(params);
  }
  else
    document.getElementById("updateResult").innerHTML="Validation error!!";
}

//delete: resetved word
```

```
function deleteF(){
  id = document.getElementById("idDel").value;
  if(id.length>0 && id.length<=30){
    xmlHttp.open("POST",url,true);
    xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    var params = "code="+code+"&op=delete&id="+id;
    xmlHttp.onreadystatechange = () => {
      if(xmlHttp.readyState == 4 && xmlHttp.status == 200) {
        let data = xmlHttp.responseText;
        if(data>0)
          str="Delete successful!";
        else
        str="Delete NOT successful!";
        document.getElementById("deleteResult").innerHTML=str;
        document.getElementById("idDel").value="";
        read();
      }
    };
    xmlHttp.send(params);
  }
  else
    document.getElementById("deleteResult").innerHTML="Validation error!!";
}
window.onload = function() {
    read();
};
```

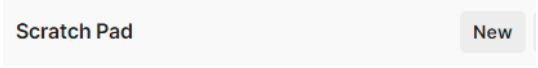
Testing an Web Service API with Postman – 10 minutes – Similar to cURL but with a graphical application

https://www.postman.com/
https://www.postman.com/downloads/
Portable (no need to install):
https://portapps.io/app/postman-portable/
Online: regisztrálni kell, de ingyenes:
https://www.postman.com/downloads/

Download the Portable version.
Run the exe file => Extract it to the desired folder.
Run the application.
No need to register an account.

https://learning.postman.com/docs/getting-started/introduction/
https://learning.postman.com/docs/getting-started/sending-the-first-request/
https://learning.postman.com/docs/sending-requests/requests/

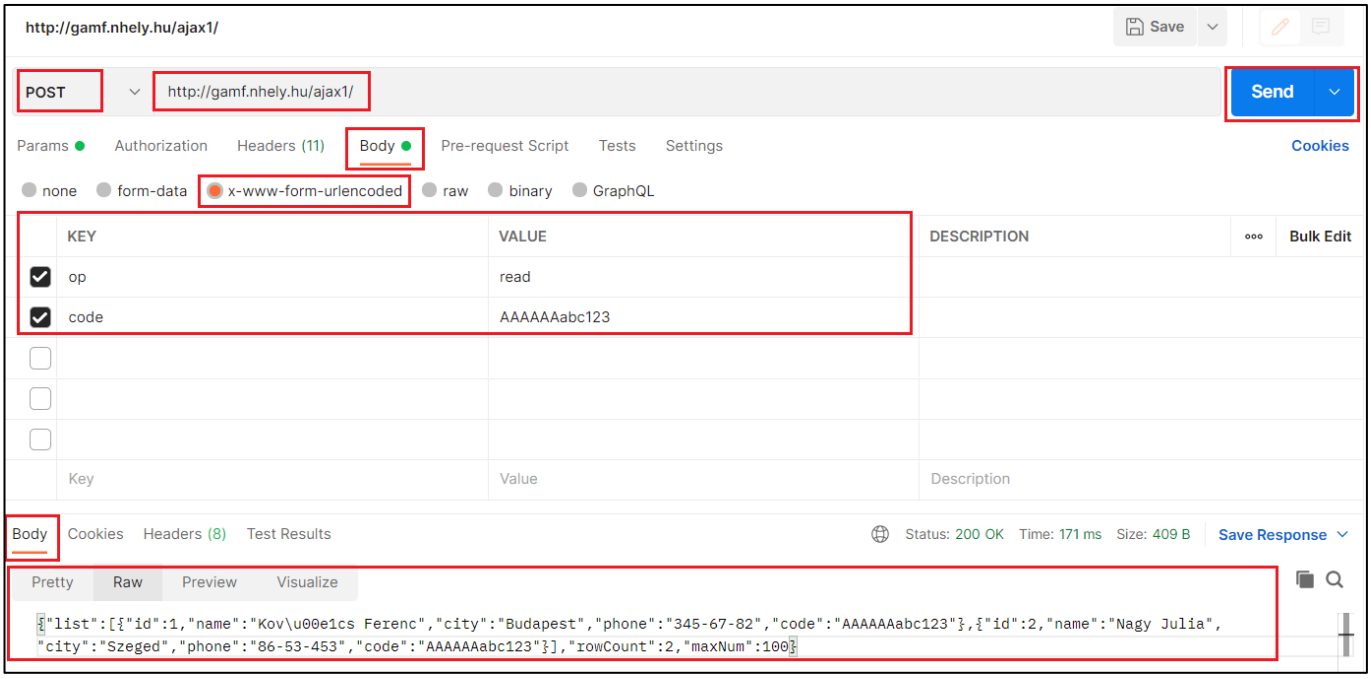
Create a request



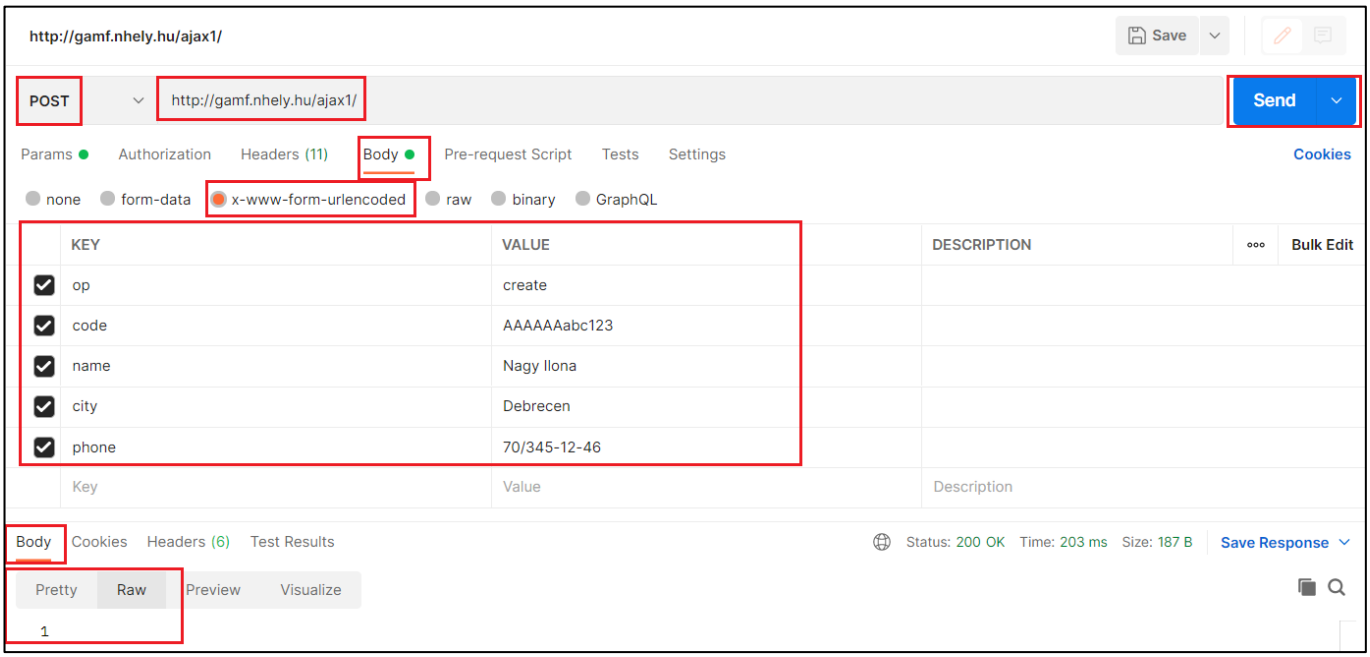
Scratch Pad / New
=> HTTP Request

http://gamf.nhely.hu/ajax1/

A, Read



B, Create



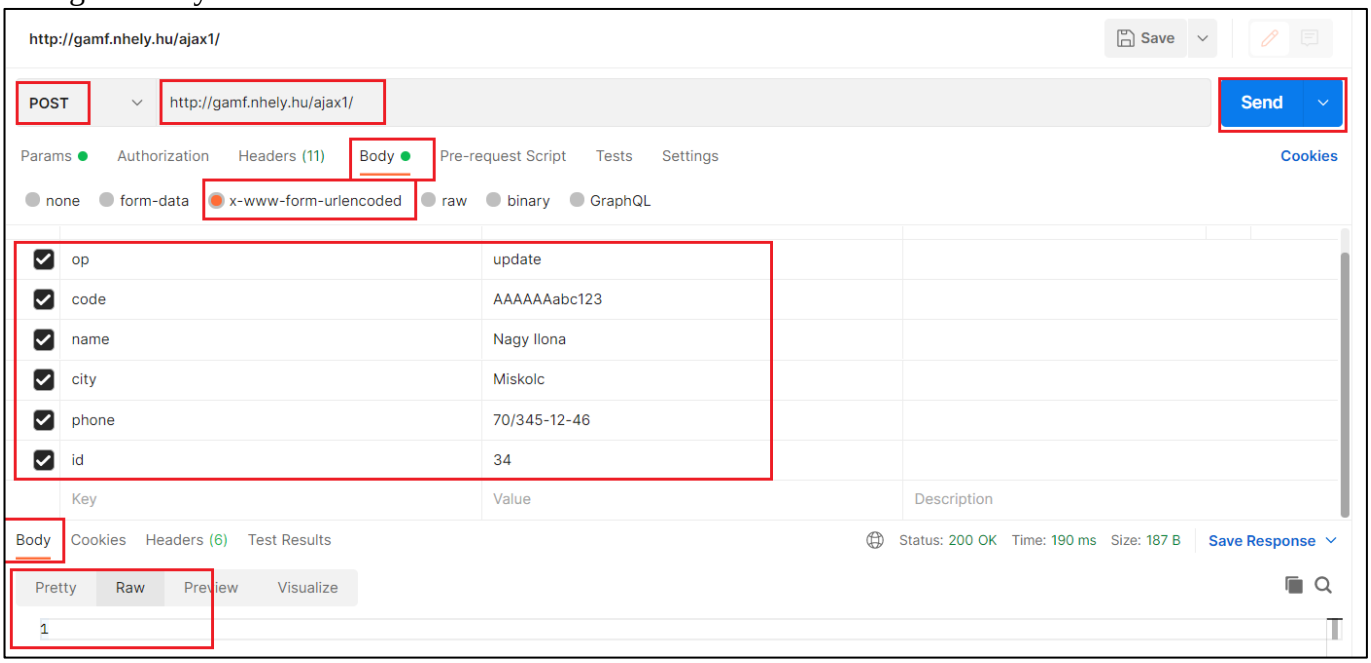
If we run Read again, we see that it has added the new record. This is also visible in our AJAX application.

id	name	city	phone	code
1	Kovács Ferenc	Budapest	345-67-82	AAAAAAabc123
2	Nagy Julia	Szeged	86-53-453	AAAAAAabc123
34	Nagy Ilona	Debrecen	70/345-12-46	AAAAAAabc123

Now the new record was id=34.

C, Update

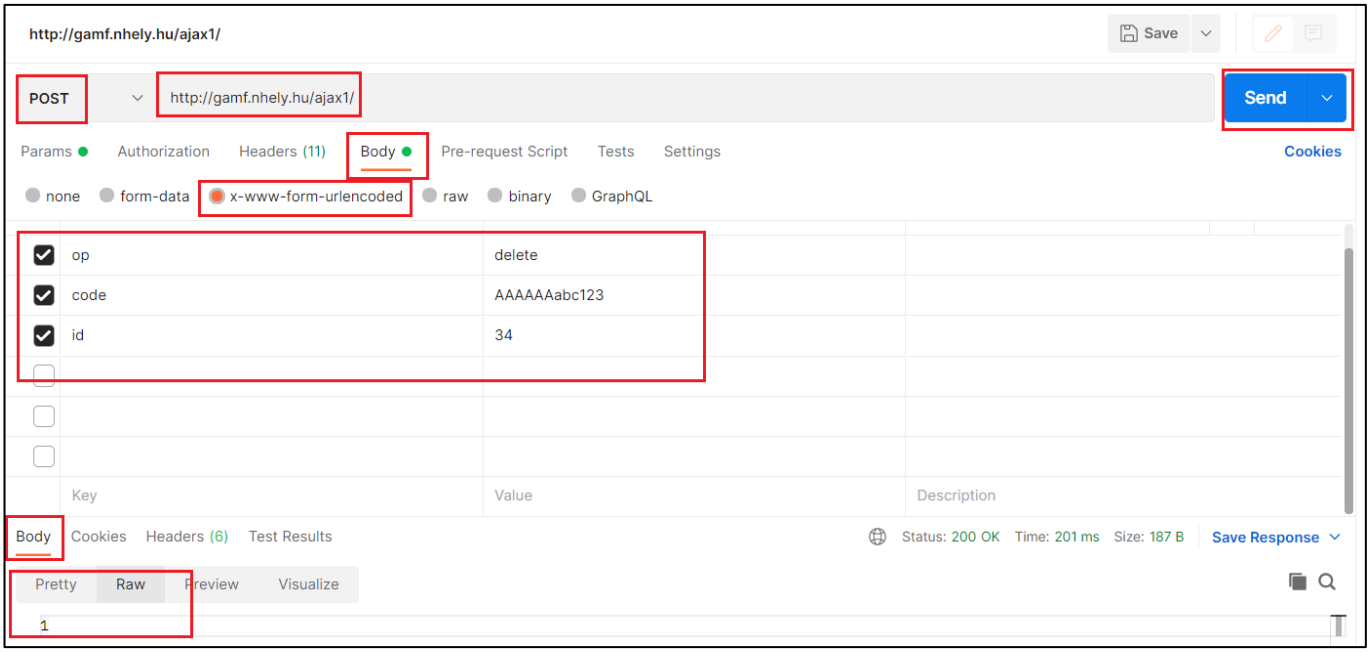
Change the City field of record ID=34 to Miskolc:



If we run Read again, we will see the change. This is also visible in our AJAX application.

D, Delete

Delete the record with ID=34:



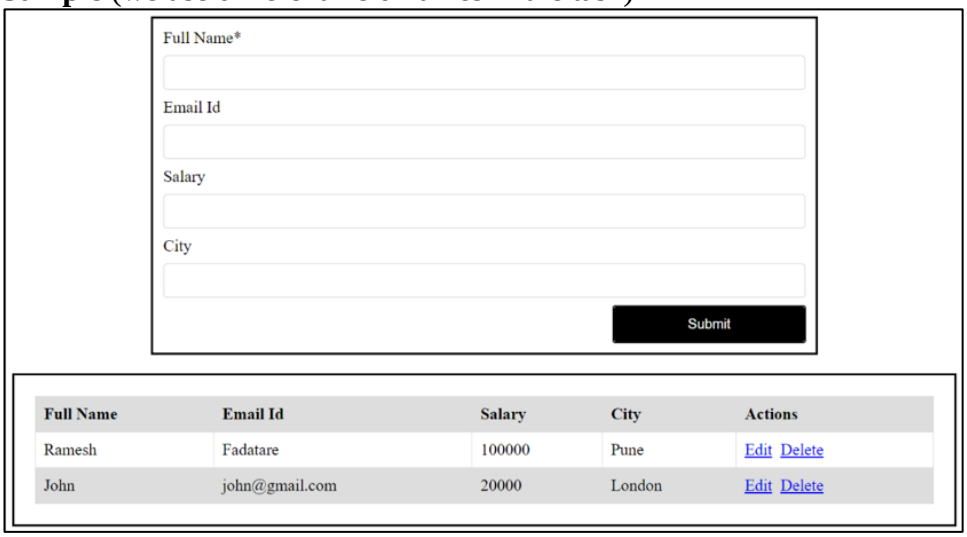
If we run Read again, we will see the change. This is also visible in our AJAX application.

Practicing Homework – Rearrange the Appearance

In the task, we implemented the 4 CRUD functions in separate parts. Reorganize the layout to the following form:

- Have a form for **Create** and **Update** operations
- Add **Edit** and **Delete** links after the records

Sample (we use different field names in the task)



Online test1 – Works fine when copied to HTTP host

e.g. in nethely.hu host

Online test2 – Doesn't work when copied to HTTPS host => requires a proxy

Copying to HTTPS storage and loading the ajax.html file:
- does not load data from the database (Read)
- right click on the page / Inspect / Console

The page at 'https://*****/ajax.html' was loaded over HTTPS, but requested an insecure resource 'http://gamf.nhely.hu/ajax1/'. This request has been blocked; the content must be served over HTTPS.

proxy:
AjaxApi.php

```
<?php
   $fields = [];
   foreach(["op", "code", "name", "city", "phone", "id"] as $f)
      if(isset($_POST[$f]))
         $fields[$f] = $_POST[$f];
   $ch = curl_init();
   curl_setopt($ch, CURLOPT_URL, "http://gamf.nhely.hu/ajax1/");
   curl_setopt($ch, CURLOPT_HEADER, 0);
   curl_setopt($ch, CURLOPT_POST, true);
   curl_setopt($ch, CURLOPT_USERAGENT, "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36");
   curl_setopt($ch, CURLOPT_POSTFIELDS, $fields);
   curl_exec($ch);
```

használata: a ajax.js fájlban ehelyett:

url="http://gamf.nhely.hu/ajax1/";

ehhez hasonlót kell használni:

url="https://example.com/AjaxApi.php";

7-React basics - JavaScript library-framework

Some call React a library, some call it a framework.

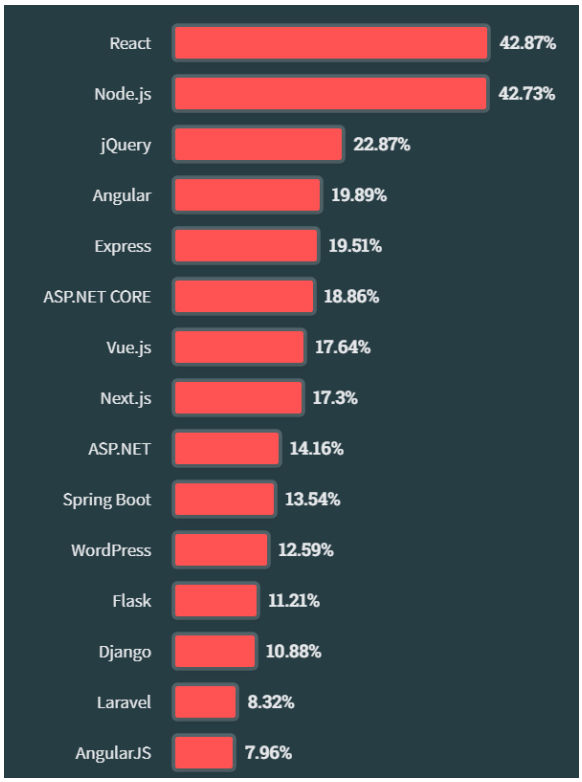
Introduction

Some React descriptions:
https://reactjs.org/tutorial/tutorial.html
https://www.w3schools.com/REACT/default.asp
https://react-tutorial.app/
https://www.javatpoint.com/reactjs-tutorial
https://www.tutorialspoint.com/reactjs/index.htm
https://www.newline.co/fullstack-react/
https://ibaslogic.com/react-tutorial-for-beginners/
https://www.freecodecamp.org/news/react-tutorial-build-a-project/

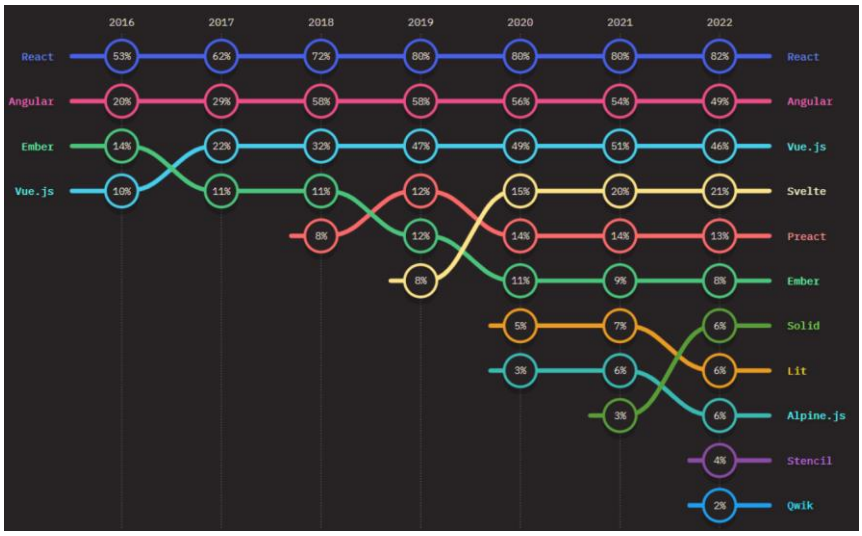
- React is a JavaScript library for building user interfaces.
- React is used to build single-page applications.
- React allows us to create reusable UI components.

Why React? - React vs Angular vs Vue

https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe-prof
How many of those surveyed used:

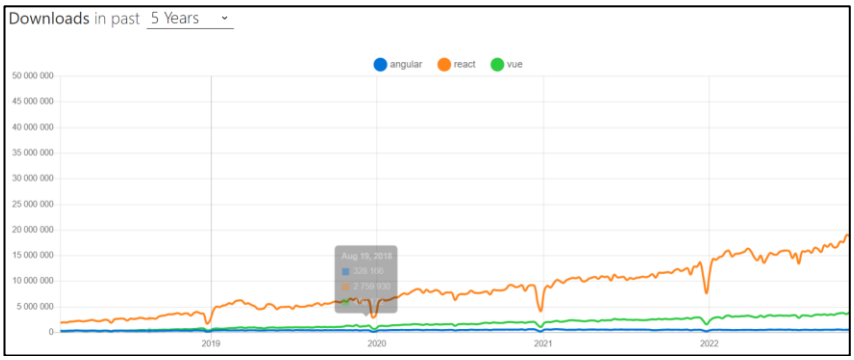


https://www.lambdatest.com/blog/best-javascript-frameworks/

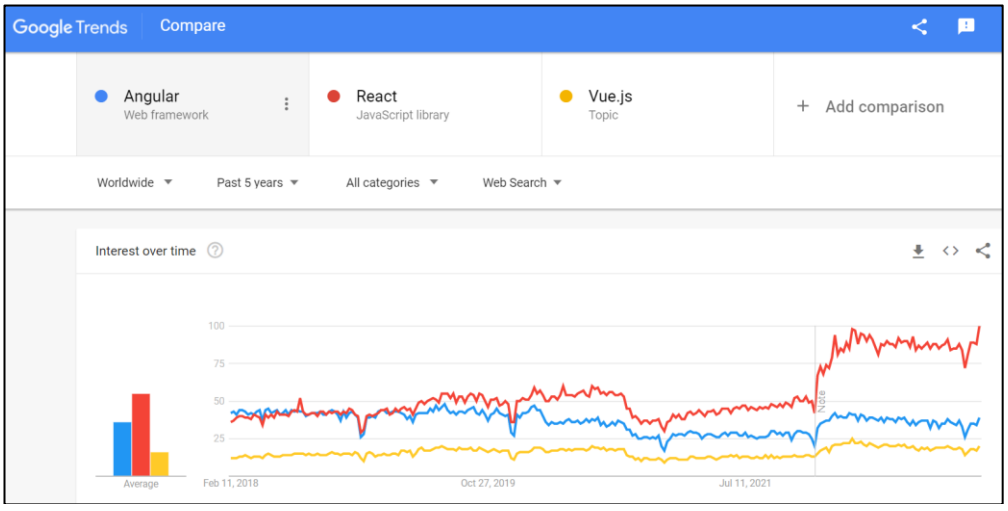


NPM trends

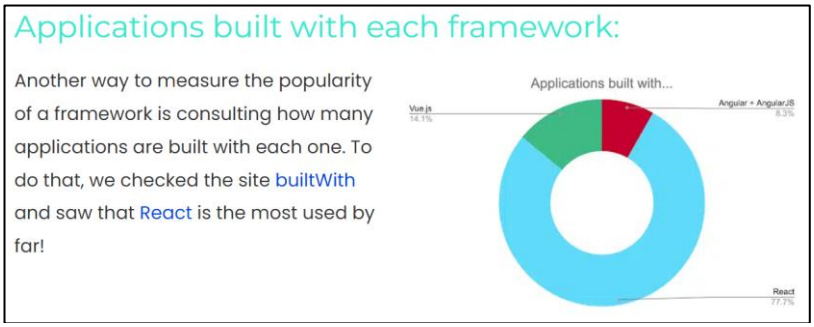
https://npmtrends.com/angular-vs-react-vs-vue

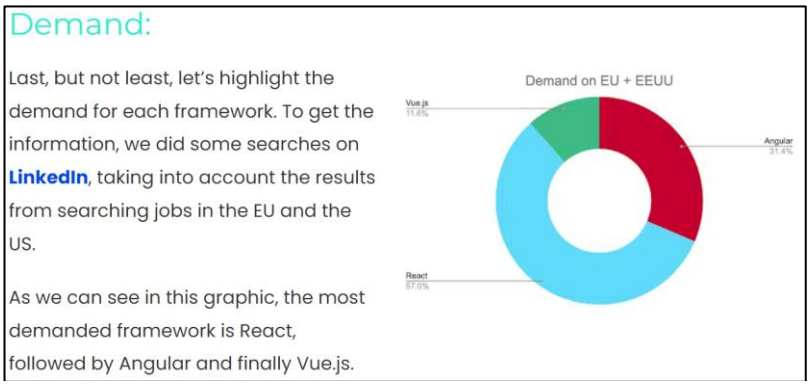


Google trends
https://trends.google.com/trends/explore?q=%2Fg%2F11c6w0ddw9,%2Fm%2F012l1vxv,%2Fg%2F11c0vmgx5d



https://www.zartis.com/angular-vs-react-vs-vuejs/





React editions

19.0.0 (December, 2024)
18.0.0 (March 2022)
17.0.0 (October 2020)
16.0 (September 2017)
15.0.0 (April 2016)
14.0 (October 2015)
13.0 (March 2015)
12.0 (October 2014)

https://www.lambdatest.com/blog/best-javascript-frameworks/

Prominent Websites Built with React:
Airbnb, Asana, BBC, Cloudflare, Codecademy, Dropbox, Facebook, GitHub, Imgur, Instagram, Medium, Netflix, OkCupid, Paypal, Periscope, Pinterest, Product Hunt, Reddit, Salesforce, Scribd, Shopify, Slack, Snapchat, Squarespace, Tesla, The New York Times, Typeform, Twitter, Uber, Udemy, WhatsApp, Zendesk.

Prominent Websites Built with Angular:
Google, Allegro, Blender, Clickup, Clockify, Delta, Deutsche Bank, DoubleClick, Freelancer, Forbes, Guardian, IBM, Instapage, iStock, JetBlue, Lego, Mailerlite, Microsoft Office, Mixer, Udacity, Upwork, Vevo, Walmart, Weather, WikiWand, Xbox, YouTube.

Prominent Websites Built with Vue.js:
9gag, Adobe, Apple Swift UI, Behance, Bilibili, BMW, Chess, Font Awesome, GitLab, Hack The Box, Laravel, Laracasts, Louis Vuitton, Namecheap, Netlify, Netguru, Nintendo, Pluralsight, Shien, Stack Overflow, Trivago, Trustpilot, Upwork, Wizzair, Zoom.

Babel/standalone – with CDN links- JavaScript compiler: for practice: compiles at runtime, therefore slower => we will use a faster method later

The browser understands JavaScript, but does not understand React => We need a compiler:
Babel/standalone

https://babeljs.io/
https://babeljs.io/docs/en/

Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.

https://babeljs.io/docs/en/babel-standalone

@babel/standalone provides a standalone build of Babel for use in browsers and other non-Node.js environments.

Babel, JSX and React

Babel can convert JSX syntax!

JSX: JSX stands for JavaScript XML. JSX allows us to write HTML in React.

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.
JSX converts HTML tags into react elements.
You are not required to use JSX, but JSX makes it easier to write React applications.

Rendering HTML code into a DIV with CSS.

exercise-03.html

```
<!DOCTYPE html>
<html>
 <head>
  <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>     <style>
            table, th, td {
            border: 1px solid black;
            border-collapse: collapse;
            }
      </style>
 </head>
 <body>
      <div id="root"></div>
      <script type="text/babel">
            const myelement = (
              <table>
                  <tr>
                    <th>Name</th>
                  </tr>
                  <tr>
                    <td>John</td>
                  </tr>
                  <tr>
                    <td>Elsa</td>
                  </tr>
```

```
            </table>
        );
        ReactDOM.render(myelement, document.getElementById('root'));
    </script>
  </body>
</html>
```

ReactDOM: ReactDom is a core react package that provides methods to interact with the Document Object Model or DOM.

createRoot: lets you create a root to display React components inside a browser DOM node.

render: renders a piece of JSX (“React node”) into a browser DOM node.
render: ad, nyújt

Some smaller tasks: variables, fragment, className

exercise-04.html

In the previous example, the 3 lines in the <head> section will be the same in each example, so I've labeled them ……. from now on:

```
<!DOCTYPE html>
<html>
  <head>
……..
  </head>
  <body>
        <div id="root1"></div>
        <div id="root3"></div>
        <div id="root4"></div>
        <div id="root5"></div>
        <div id="root6"></div>
        <script type="text/babel">
            const myElement1 = <h1>React is {5 + 5} times better with JSX</h1>;
            const root1 = ReactDOM.createRoot(document.getElementById('root1'));
            root1.render(myElement1);

            const myElement3 = (
// Multiple elements must be placed in ONE container element, e.g. a DIV,
//      which we use to connect them to the DOM.
              <div>
                  <p>I am a paragraph.</p>
                  <p>I am a paragraph too.</p>
              </div>
            );
            const root3 = ReactDOM.createRoot(document.getElementById('root3'));
            root3.render(myElement3);

            const myElement4 = (
// Alternatively, you can use a fragment to organize multiple elements into a unit.
              <>
```

```
                <p>I am a paragraph.</p>
                <p>I am a paragraph too.</p>
            </>
        );
        const root4 = ReactDOM.createRoot(document.getElementById('root4'));
        root4.render(myElement4);

        // All items must be closed:
        const myElement5 = <input type="text" />;          ehelyett: <input type="text">
        const root5 = ReactDOM.createRoot(document.getElementById('root5'));
        root5.render(myElement5);

// In HTML we write it like this: <h1 class="myclass">, but since class is a reserved word in JavaScript,
//       we write it like this:   <h1 className="myclass">
        const myElement6 = <h1 className="myclass">Hello World</h1>;
        const root6 = ReactDOM.createRoot(document.getElementById('root6'));
        root6.render(myElement6);
    </script>
  </body>
</html>
```

React is 10 times better with JSX, a = 20

- Apples
- Bananas
- Cherries

I am a paragraph.

I am a paragraph too.

Hello World

Components, Props

https://www.w3schools.com/REACT/react_components.asp

Components are like **JavaScript functions** that **return HTML elements**. Components are **independent and reusable bits of code**.

Two types of Components

- Function component
- Class component

https://www.w3schools.com/REACT/react_props.asp
Props: function's arguments: They contain key-value pairs e.g. color="red"

exercise-05.html

```
<!DOCTYPE html>
<html>
  <head>
```

```
……..
  </head>
  <body>
        <div id="root1"></div>
        <div id="root2"></div>
        <div id="root3"></div>
        <div id="root4"></div>
        <script type="text/babel">
// Function Component, returns HTML:
                function Car() {
                  return <h2>Hi, I am a Car!</h2>;
                }
                const root1 = ReactDOM.createRoot(document.getElementById('root1'));
                root1.render(<Car />);

                function Car2(props) {
                  return <h2>I am a {props.color} Car2!</h2>;
                }
                const root2 = ReactDOM.createRoot(document.getElementById('root2'));
                root2.render(<Car2 color="red"/>);

                function Car3() {
                  return <h2>I am a Car3!</h2>;
                }
                function Garage() {
                  return (
                        <>
                          <h1>Who lives in my Garage?</h1>
                          <Car3 />
                        </>
                  );
                }
                const root3 = ReactDOM.createRoot(document.getElementById('root3'));
                root3.render(<Garage />);

// Class component
//A class component must include the extends React.Component statement. This statement creates an
// inheritance to React.Component, and gives your component access to React.Component's functions.
                class Car4 extends React.Component {
                  render() {
                        return <h2>Hi, I am a Car4!</h2>;
                  }
                }
                const root4 = ReactDOM.createRoot(document.getElementById('root4'));
                root4.render(<Car4 />);
        </script>

  </body>
</html>
```

Hi, I am a Car!
I am a red Car2!
Who lives in my Garage?

I am a Car3!
Hi, I am a Car4!

Multi-level Prop – with JSON

feladat-06.html

```
<!DOCTYPE html>
<html>
  <head>
……..
  </head>
  <body>
    <div id="root3"></div>
    <script type="text/babel">
       function Car3(props) {
         return <h2>I am a { props.brand.model }!</h2>;
       }
       function Garage3() {
         const carInfo = { name: "Ford", model: "Mustang" };          // JSON
         return (
           <>
             <h1>Who lives in my garage?</h1>
             <Car3 brand={ carInfo } />
           </>
         );
       }
       const root3 = ReactDOM.createRoot(document.getElementById('root3'));
       root3.render(<Garage3 />);
    </script>
  </body>
</html>
```

Prints:
Who lives in my garage?
I am a Mustang!

Do not use getElementById and getElementsByClassName selectors inside the component

You can have such an ID or Class outside the component on the webpage.

Storing componenst in JS files – CORS error => Web server is needed

The main feature of React is the reuse of code. It's recommended to store the components in separated JS files. The first character of the filename must be an uppercase letter.

A, Using this method without web-server gives CORS error.

Comp1.js

```
class Class1 extends React.Component {
   render(){
```

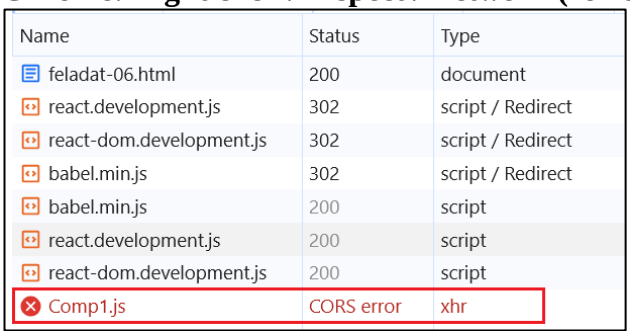
```
      return ( <div>Hello From React </div> );
   }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Class1 />);
```

exercise-07.html

```
<!DOCTYPE html>
<html>
  <head>
……..
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel" src="Comp1.js"></script>
  </body>
</html>
```

Open **exercise-07.html** file in browser => Empty page

Chrome: Right click / Inspect / Network (refresh the page):



Name	Status	Type
feladat-06.html	200	document
react.development.js	302	script / Redirect
react-dom.development.js	302	script / Redirect
babel.min.js	302	script / Redirect
babel.min.js	200	script
react.development.js	200	script
react-dom.development.js	200	script
Comp1.js	CORS error	xhr

The browser has not allowed the code to run: for security reasons, it is not possible to run any code read from other pages. Reason for blocking:
CORS: Cross-origin resource sharing
https://en.wikipedia.org/wiki/Cross-origin_resource_sharing
Cross-origin resource sharing (CORS) is a mechanism to safely bypass the same-origin policy, that is, it allows a web page to access restricted resources from a server on a domain different than the domain that served the web page. A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos. Certain "cross-domain" requests, notably Ajax requests, are forbidden by default by the same-origin security policy.

B, Copying the files to web-server: there is no error.

- On local computer using XAMPP
  Try it!
  - XAMPP Control Panel: Start Apache web-server
  - copy files to c:\xampp\htdocs\exercise folder:
  - http://localhost/exercise/exercise-06.html
    Prints: Hello From React
- Using Internet host pl. nethely.hu. You have to use it for Homework

C, Developement without Babel/standalone method, we'll use it later, solves this problem as well.

Events

https://www.w3schools.com/REACT/react_events.asp

Similar to JavaScript events
We have to use camelCase format onClick
Writing React event handlers: onClick={shoot}

We have 2 buttons on page:

exercise-09.html

```
<!DOCTYPE html>
<html>
  <head>
……….……
  </head>
  <body>
    <div id="root1"></div>
    <div id="root2"></div>
    <script type="text/babel">
// component:
      function Football1() {
// arrow function:
        const shoot = () => {
          alert("Great Shot!");
        }
        return <button onClick={shoot}>Take the shot!</button>;
      }
      const root1 = ReactDOM.createRoot(document.getElementById('root1'));
      root1.render(<Football1 />);

// Like the previous component, only here the shoot function also has an input parameter:
      function Football2() {
        const shoot = (a) => {
           alert(a);
        }
        return <button onClick={() => shoot("Goal!")}>Take the shot!</button>;
      }
      const root2 = ReactDOM.createRoot(document.getElementById('root2'));
      root2.render(<Football2 />);
    </script>
  </body>
</html>
```

useState Hook – preserving the value and state of variables

We can store and modify the states of variables in components without using a class.
We can set an initial value.
We provide a setter method to set the current value of the variable: count=>setCount
This allows us to preserve the values of variables between function calls. Otherwise, the variables would disappear when the function ends.

e.g. number of clicks: When a button is clicked, the value of the variable is increased by 1.
Hook: variable+initial value+Setter

exercise-11.html

```
<!DOCTYPE html>
<html>
  <head>
……..
  </head>
  <body>
    <div id="root1"></div>
    <div id="root2"></div>
    <script type="text/babel">
      const initialValue = React.useState;
      function Example() {
// defines a new state variable called count. initialValue: 0
        const [count, setCount] = initialValue(0);
        return (
        <div>
          <p>You clicked {count} times</p>
// Increases the count value by 1 when the button is clicked:
          <button onClick={() => setCount(count + 1)}>Click me</button>
        </div>
        );
      }
      const root1 = ReactDOM.createRoot(document.getElementById('root1'));
      root1.render(<Example />);

      function FavoriteColor() {
        const [color, setColor] = initialValue("red");
        return (
        <>
          <h1>My favorite color is {color}!</h1>
          <button type="button" onClick={() => setColor("blue")}>Blue</button>
        </>
        )
      }
      const root2 = ReactDOM.createRoot(document.getElementById('root2'));
      root2.render(<FavoriteColor />);
    </script>
  </body>
</html>
```

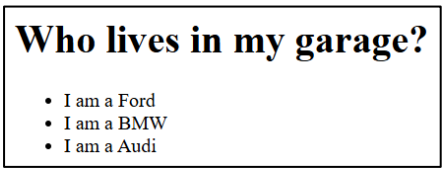
If the component state changes, the component is re-rendered => provides dynamic content in response to user interaction

e.g. here if the count or color states change, then the component is re-rendered, so the new value appears on the page.
By default, when your component's state or props change, your component will re-render.
https://legacy.reactjs.org/docs/react-component.html
React automatically re-renders components whenever there is a change in their state or props and provides dynamic content in

accordance with user interactions.
https://www.geeksforgeeks.org/re-rendering-components-in-reactjs/

Lists, loop, map, Keys

https://www.w3schools.com/REACT/react_lists.asp



We can render lists using loops. To do this, we use the map() Javascript array method.
Each element of the list will be a separate component.

exercise-12a.html

```
<!DOCTYPE html>
<html>
  <head>
……..
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      function Car(props) {
        return <li>I am a { props.brand }</li>;
      }

      function Garage1() {
// cars list
        const cars = ['Ford', 'BMW', 'Audi'];
        return (
        <>
          <h1>Who lives in my garage?</h1>
          <ul>
// For each element (car) in the cars list, call the Car component with the parameter brand={car}
            {cars.map((car) => <Car brand={car} />)}
          </ul>
        </>
        );
      }
      const root = ReactDOM.createRoot(document.getElementById('root'));
      root.render(<Garage1 />);
    </script>
  </body>
</html>
```

The website appears fine, but we get a warning: right click / Inspect /Console

Warning: Each child in a list should have a unique "key" prop.
Solution => Keys

Keys

Used to identify items in a list.

Helps keep track of items. If an item is updated or deleted, only that item will be re-rendered and not the entire list.
https://www.geeksforgeeks.org/reactjs-keys/

> React JS keys are an important concept for handling dynamic content, especially when rendering lists of elements. Keys help React identify which items have changed, been added, or removed, allowing for efficient updates to the user interface.

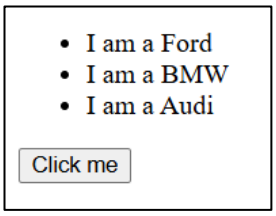
We can't access keys within a component (nor do we want to).

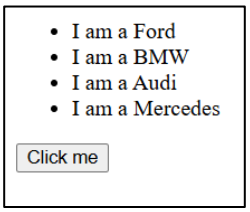
Rewrite the previous example so that we use keys.

exercise-12b.html

```
<!DOCTYPE html>
<html>
  <head>
……….……
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      function Car(props) {
        return <li>I am a { props.brand }</li>;
      }
      function Garage() {
        return (
          <>
            <h1>Who lives in my garage?</h1>
            <ul>
              {cars.map((car) => <Car key={car.id} brand={car.brand} />)}
            </ul>
          </>
        );
      }
      const cars = [
        {id: 1, brand: 'Ford'},
        {id: 2, brand: 'BMW'},
        {id: 3, brand: 'Audi'}
      ];
      const root = ReactDOM.createRoot(document.getElementById('root'));
      root.render(<Garage />);
    </script>
  </body>
</html>
```

After the list is displayed, modify the list: click a button to add a new item.
To make the component re-render after the list is modified: use a useState state variable that is modified after the list is modified.





exercise-12c.html

```
<!DOCTYPE html>
<html>
  <head>
……..
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      const initialValue = React.useState;
      function Car(props) {
        return <li>I am a { props.brand }</li>;
      }
      function Garage() {
        const [count, setCount] = initialValue(0);
        return (
          <>
            <h1>Who lives in my garage?</h1>
            <ul>
              {cars.map((car) => <Car key={car.id} brand={car.brand} />)}
            </ul>
            <button onClick={() =>  {cars.push({id: 5, brand: 'Mercedes'});setCount(count + 1);}}>Click
me</button>
          </>
        );
      }
      const cars = [
        {id: 1, brand: 'Ford'},
        {id: 2, brand: 'BMW'},
        {id: 3, brand: 'Audi'}
      ];
      const root = ReactDOM.createRoot(document.getElementById('root'));
      root.render(<Garage />);
    </script>
  </body>
</html>
```

8-9 – React – Development without Babel/standalone – Install React on local computer - vite + React - we can create a faster application this way: no need to compile at runtime

We don’t install the apps in class, but we run and study from **Solutions.zip**.

Install Node.js:

Webpage and download: https://nodejs.org
Download the LTS (Long Term Support) version https://nodejs.org/en/download/
Download Windows Installer, and install it

vite + React

https://learn.microsoft.com/en-us/windows/dev-environment/javascript/react-on-windows

```
npm create vite@latest my-react-app -- --template react
  cd my-react-app
  npm install
```

Sample application created with installation

Run

```
  npm run dev
```

http://localhost:5173/

The files

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

/src/main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

StrictMode

https://react.dev/reference/react/StrictMode
<StrictMode> lets you find common bugs in your components early during development.

App.jsx

```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)
  return (
    <>
      <div>
        <a href="https://vite.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.jsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}
export default App
```

Build and Run on Remote Server

Register with a free hosting provider e.g. nethely.hu
Description: **Internet-Hosting-Provider-PHP-Hungarian Nethely.hu-in English.docx**

npm run build

Copy the contents of the **dist** folder to the remote server using FTP.
Enter the URL, set on the hosting in your browser:
It works!

01- Router – Creating Single-page Application (SPA)

https://developer.mozilla.org/en-US/docs/Glossary/SPA
https://en.wikipedia.org/wiki/Single-page_application
https://www.bloomreach.com/en/blog/what-is-a-single-page-application

A single-page application (SPA) is a web app that is presented to the user through a single HTML page.

https://www.w3schools.com/REACT/react_router.asp

Install the react-router-dom modult: In Command prompt:
npm i -D react-router-dom@latest

npm create vite@latest router -- --template react
 cd router
 npm install

The files:

/src:
main.jsx
App2.jsx
Layout.js
Home.js
Blogs.js
Contact.js
NoPage.js

/src/main.jsx
```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
// import './index.css'
import App2 from './App2.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App2 />
  </StrictMode>,
)
```

/scr/App2.jsx
```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./Layout";
import Home from "./Home";
import Blogs from "./Blogs";
import Contact from "./Contact";
import NoPage from "./NoPage";
export default function App2() {
  return (
    <BrowserRouter>
```

```
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
```

Layout.jsx

```
import { Outlet, Link } from "react-router-dom";
const Layout = () => {
  return (
    <>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/blogs">Blogs</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Outlet />
    </>
  )
};
export default Layout;
```

Home.jsx

```
const Home = () => {
    return <h1>Home page</h1>;
  };
export default Home;
```

Blogs.jsx

```
const Blogs = () => {
    return <h1>Blog Articles</h1>;
  };
export default Blogs;
```

Contact.jsx

```
const Contact = () => {
    return <h1>Contact Me</h1>;
  };
```

export default Contact;

Start on local computer

npm run dev

http://localhost:5173/

Build and Run on a Remote Server – If using a Router, you need a .htaccess file!

npm run build

Copy the contents of the **dist** folder to the remote server using FTP.
Enter the URL, set on the hosting site in your browser:
It seems to work fine!

Problem
We go to a page, e.g. Contact
/contact
In the browser: Reload => gives a 404 error
Cause: the server is trying to load the /contact path, but it does not exist on the server

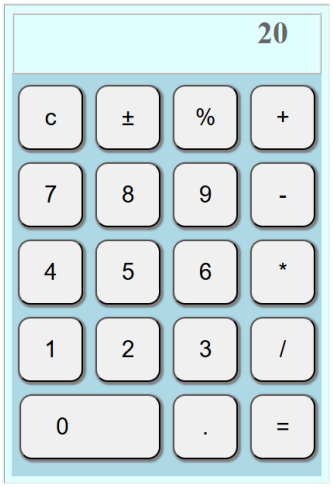
Solution
Send all calls to the index.html file:

.htaccess

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /
  RewriteRule ^index\.html$ - [L]
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteCond %{REQUEST_FILENAME} !-l
  RewriteRule . /index.html [L]
</IfModule>
```

02- Calculator app

https://medium.com/@gusya59/building-a-simple-calculator-with-react-react-hooks-and-grid-f6724baed5e



After each calculation, you must press the c button to start a new operation!

npm create vite@latest calculator -- --template react
 cd calculator
 npm install

index.html

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <link rel="icon" type="image/svg+xml" href="/vite.svg" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Vite + React</title>
 </head>
 <body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
 </body>
</html>
```

main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'
createRoot(document.getElementById('root')).render(
 <StrictMode>
  <App />
 </StrictMode>,
)
```

Create a **components** folder in the src folder.

src/App.jsx

```
import Calculator from "./components/Calculator";
function App() {
 return <Calculator />;
}
export default App;
```

Create a common component for the calculator buttons called CompButton. This is how we call the button in Calculator.jsx.

Input parameter: keyValue={"+"} This is where we specify the function of the button.

The onClick event: onClick={handleOperation}

```
<CompButton keyValue={"+"} onClick={handleOperation} />
<CompButton keyValue={7} onClick={handleOperation} />
```

handleOperation => Find in components/CompButton.jsx file

We include the className attributes for CSS formatting.

src/components/CompButton.jsx

```
import "./CompButton.css";
//props: keyValue, onClick and className
//        className is only present at two components, e.g.:
//                <CompButton className="key-zero" keyValue={0} onClick={handleOperation} />
function CompButton(props) {
  return (
// Set the className property
// Set the button click event handler:
//        In Calculator.jsx, call it like this: onClick={handleOperation}
//        Call the handleOperation event handler function with the props.keyValue parameter
    <button className={`${props.className}`} onClick={() => props.onClick(props.keyValue)}>
// Set the text that appears on the button:
      {props.keyValue}
    </button>
  );
}
export default CompButton;
```

Examples of clicking buttons

We use 3 state variables:

```
  const [prevValue, setPrevValue] = useState(null);
  const [nextValue, setNextValue] = useState("0");
  const [op, setOp] = useState(null);
  useEffect(() => {}, [op, nextValue, prevValue]);
```

The current value of the nextValue state variable is written in the input field:

```
        <div className="result">{nextValue} </div>
```

A, If we click the c (delete) button:

```
        <CompButton keyValue={"c"} onClick={handleOperation} />
        => CompButton.jsx: onClick={() => props.onClick(props.keyValue)}
        => Calls the handleOperation function with the parameter "c"
        in the handleOperation function: if the parameter = "c"
        if (value === "c") {
            clearData();
        }
        =>   sets the values of the NextValue and PrevValue state variables to 0:
          const clearData = () => {
            setNextValue("0");
```

```
    setPrevValue(0);
  };
```

B, If we click on one of the digit buttons:

pl. <CompButton keyValue={9} onClick={handleOperation} />

We call the handleOperation function with parameter 9

```
          if (Number.isInteger(value)) {
            handleNum(parseInt(value, 10));
      =>   const handleNum = (number) => {
// Stores the NextValue in a String, which is composed of the pressed digit characters
// If the state variable nextValue has the value "0", then the value of nextValue is the character
//         that was pressed just now,
// otherwise, adds the character that was pressed just now to the end of the nextValue String:
            setNextValue(nextValue === "0" ? String(number) : nextValue + number);
          };
```

C, If any of the operation symbols are pressed

pl. <CompButton keyValue={"+"} onClick={handleOperation} />

We call the handleOperation function with the "+" parameter

```
if (value in CalculatorOperations) {  // if the pressed character is an operation symbol
    if (op === null) {  // If the op state variable is null
      setOp(value);     // Sets the value of the op state variable to the given operation signal
// The value of nextValue (the number entered so far) will be the value of PrevValue
      setPrevValue(nextValue);
      setNextValue("");         // Sets NextValue to an empty String
    }
// If there is already an operation in op, it will be replaced by the new one
// e.g. if we press two operation signs in a row, the last one will be the current one
    if (op) {
      setOp(value);
    }
// If all three state variables already have values (we pressed the = character),
//         then we execute the operation
    if (prevValue && op && nextValue) {
      performOperation();
    }
  }
```

Perform operation: => performOperation();:

```
  const performOperation = () => {
// calls the CalculatorOperations[op] operation, puts the result in the temp variable
    let temp = CalculatorOperations[op](
      parseFloat(prevValue),
      parseFloat(nextValue)
    );
    setOp(null);
// The value of NextValue will be the value of temp: this is what appears in the input field!
    setNextValue(String(temp));
    setPrevValue(null);
  };
```

src/components/Calculator.jsx

```
import React, { useState, useEffect } from "react";
import CompButton from "./CompButton";
import "./Calculator.css";

function Calculator() {
// We use 3 state variables:
//        prevValue: first number, nextValue: second number, op: operation
//        we perform the given operation on the first and second numbers. e.g. 45 + 32
// The prevValue is stored as a number. This is e.g. the first entered digit
  const [prevValue, setPrevValue] = useState(null);
// Stores the NextValue in a String, which is concatenated from the pressed digit characters.
//        It must be stored in a String for the concatenation.
  const [nextValue, setNextValue] = useState("0");
  const [op, setOp] = useState(null);
  useEffect(() => {}, [op, nextValue, prevValue]);



  const CalculatorOperations = {
    "/": (firstValue, secondValue) => firstValue / secondValue,
    "*": (firstValue, secondValue) => firstValue * secondValue,
    "+": (firstValue, secondValue) => firstValue + secondValue,
    "-": (firstValue, secondValue) => firstValue - secondValue,
    "=": (firstValue, secondValue) => secondValue,
  };

  const performOperation = () => {
    let temp = CalculatorOperations[op](
      parseFloat(prevValue),
      parseFloat(nextValue)
    );
    setOp(null);
    setNextValue(String(temp));
    setPrevValue(null);
  };

  const handleNum = (number) => {
    setNextValue(nextValue === "0" ? String(number) : nextValue + number);
  };

  const insertDot = () => {
    if (!/\./.test(nextValue)) {
      setNextValue(nextValue + ".");
    }
  };
  const percentage = () => {
    setNextValue(parseFloat(nextValue) / 100);
    if (prevValue && nextValue === "") {
      setPrevValue(parseFloat(prevValue) / 100);
    }
  };
  const changeSign = () => {
    setNextValue(parseFloat(nextValue) * -1);
```

```
};
const clearData = () => {
  setNextValue("0");
  setPrevValue(0);
};

const handleOperation = (value) => {
  if (Number.isInteger(value)) {
    handleNum(parseInt(value, 10));
  } else if (value in CalculatorOperations) {
    if (op === null) {
      setOp(value);
      setPrevValue(nextValue);
      setNextValue("");
    }
    if (op) {
      setOp(value);
    }
    if (prevValue && op && nextValue) {
      performOperation();
    }
  } else if (value === "c") {
    clearData();
  } else if (value === "\xB1") {
    changeSign();
  } else if (value === ".") {
    insertDot();
  } else if (value === "%") {
    percentage();
  }
};

return (
  <div className="calculator">
    <div className="calculator-input">
      <div className="result">{nextValue} </div>
    </div>
    <div className="calculator-keypad">
      <div className="keys-function">
        <CompButton keyValue={"c"} onClick={handleOperation} />
        <CompButton keyValue={"\xB1"} onClick={handleOperation} />
        <CompButton keyValue={"%"} onClick={handleOperation} />
      </div>
      <div className="keys-operators">
        <CompButton keyValue={"+"} onClick={handleOperation} />
        <CompButton keyValue={"-"} onClick={handleOperation} />
        <CompButton keyValue={"*"} onClick={handleOperation} />
        <CompButton keyValue={"/"} onClick={handleOperation} />
        <CompButton keyValue={"="} onClick={handleOperation} />
      </div>
      <div className="keys-numbers">
        <CompButton keyValue={9} onClick={handleOperation} />
        <CompButton keyValue={8} onClick={handleOperation} />
```

```
        <CompButton keyValue={7} onClick={handleOperation} />
        <CompButton keyValue={6} onClick={handleOperation} />
        <CompButton keyValue={5} onClick={handleOperation} />
        <CompButton keyValue={4} onClick={handleOperation} />
        <CompButton keyValue={3} onClick={handleOperation} />
        <CompButton keyValue={2} onClick={handleOperation} />
        <CompButton keyValue={1} onClick={handleOperation} />
        <CompButton className="key-dot" keyValue={"."} onClick={handleOperation} />
        <CompButton className="key-zero" keyValue={0} onClick={handleOperation} />       </div>
     </div>
    </div>
  );
}
```
export default Calculator;

The application is already working, but add some style.
We don't deal with CSS here.

src/components/CompButton.css
```
button {
    width: 4.2rem;
    height: 4.2rem;
    margin: 0.4rem;
    text-align: center;
    font-size: 150%;
    border-radius: 0.8rem;
    box-shadow: 0.1rem 0.1rem 0.1rem 0.1rem grey;
  }
```

src/components/Calculator.css
```
.calculator {
    width: 20rem;
    height: 30rem;
    background-color: lightcyan;
    display: grid;
    padding: 1%;
    position: absolute;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
    margin: auto;
    border: ridge;
  }

  .keys-numbers {
    grid-area: keys-numbers;
    direction: rtl;
  }

  .keys-numbers .key-zero {
    width: 9.1rem;
    text-align: left;
```

```
    padding-left: 15%;
  }
  .keys-operators {
    grid-area: keys-operators;
  }
  .keys-function {
    grid-area: keys-function;
  }
  .calculator-keypad {
    padding-top: 0.3rem;
    background-color: lightblue;
    display: grid;
    grid-template-columns: repeat(4, 5rem);
    grid-template-rows: repeat(5, 5rem);
    grid-template-areas: "keys-function keys-function keys-function keys-operators" "keys-numbers keys-numbers keys-numbers keys-operators" "keys-numbers keys-numbers keys-numbers keys-operators" "keys-numbers keys-numbers keys-numbers keys-operators" "keys-numbers keys-numbers keys-numbers keys-operators";
  }
  .calculator-input {
    border-color: black;
    border: ridge;
    text-align: right;
    padding-right: 10%;
    font-weight: bold;
    font-size: xx-large;
  }

  .result {
    min-height: 3rem;
    color: #666666;
  }
```

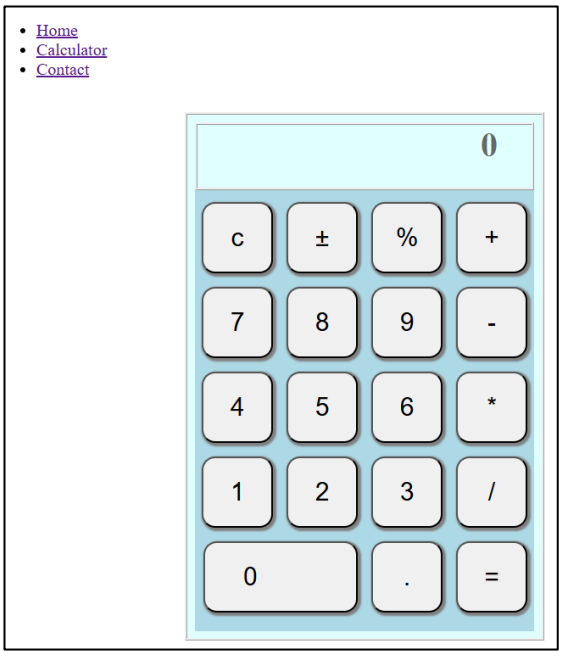
Run on a remote server

npm run build

Copy the contents of the **dist** folder to the remote server using FTP.

Enter the URL, set on the host in a browser:
It works!

Integrating the Calculator application into the Router application



Complete the Router application:
Copy the **src/components** folder to the **src** folder.

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

src/main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'
createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

src/App.jsx

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./Layout";
import Home from "./Home";
import Calculator from "./components/Calculator";
import Contact from "./Contact";
```

```
import NoPage from "./NoPage";
export default function App2() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="calculator" element={<Calculator />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
```

src/Layout.jsx

```
import { Outlet, Link } from "react-router-dom";
const Layout = () => {
  return (
    <>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/calculator">Calculator</Link>
          </li>
          <li>
            <Link to="/contact">Contact</Link>
          </li>
        </ul>
      </nav>

      <Outlet />
    </>
  )
};
export default Layout;
```

The rest has not been modified.

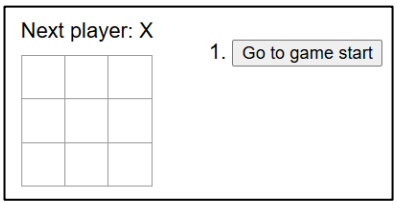
Build and copy to the Internet server - .htaccess file is needed because of the router!

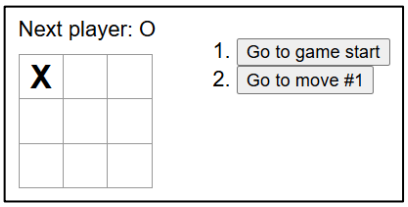
npm run build

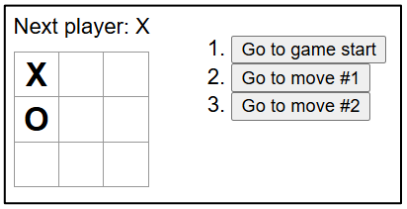
Copy the contents of the **dist** folder to the remote server via FTP.
Enter the URL address, set on the hosting in a browser:
It works fine!

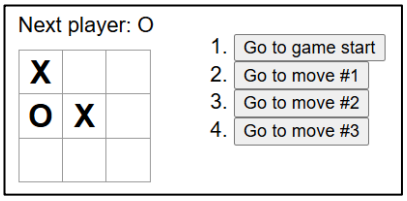
03- Tic-Tac-Toe game app

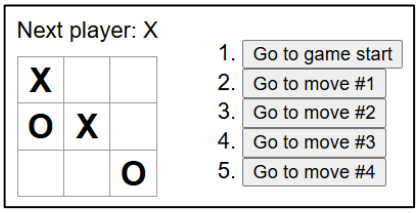
https://react.dev/learn/tutorial-tic-tac-toe

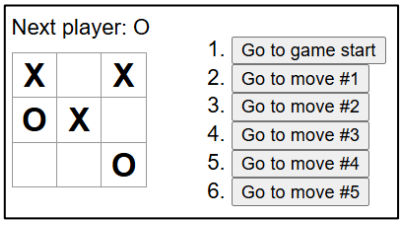


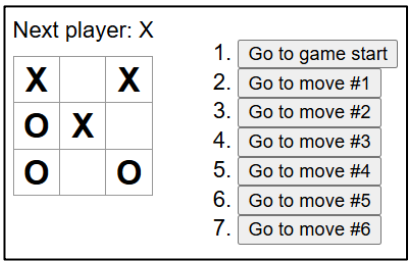


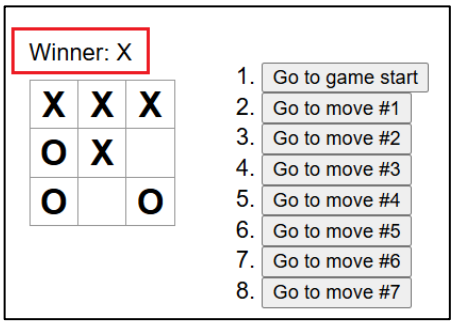












Next to the board, after each step, it places buttons that the user can click to replay the given state. To do this, we need to store all partial states: this is stored in the history state array.

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

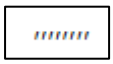
src/main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
```

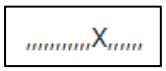
```
    <App />
  </StrictMode>,
)
```

Since we used: export default in the Game() function, this is what will be displayed on the page.
export default function Game()

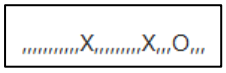
Use the following state variables in your application:
const [history, setHistory] = useState([Array(9).fill(null)]);
Contains nested arrays
It stores the current state for each step. It stores each state because we see that after each step it places buttons next to the board, which we can click to replay the given state.
At startup, it contains a 9-element array with null elements.
If we print the value of the history array at the end of the handlePlay(nextSquares) function, we see that the array is populated like this:



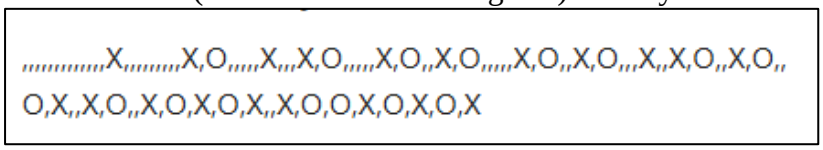
1 array of 9 elements containing NULL values



After the first array, a second array of 9 elements: the first X is placed
…….
After the first two arrays, a third array of 9 elements: the second X is also placed:



…….
After 9 moves (final result of another game): 9 arrays of 9 elements:



If we print it like this: alert(history[0]); or alert(history[1]);
then we see that in each case a 9-element array is printed (if we have already reached the given step).

const [currentMove, setCurrentMove] = useState(0);
Az utolsó lépés sorszáma: 1..9

src/App.jsx

```
import { useState } from 'react';

// The square component, which is a button element. It looks like a square on the template due to the CSS
// It has two input parameters:
//        value: this will be the text written in the button
//        onSquareClick: the event handler for clicking the button
function Square({ value, onSquareClick }) {
  return (<button className="square" onClick={onSquareClick}>{value}</button>);
}

// Draws the playing field
//        xIsNext: true or false: X is next?
//        squares: array: 0..8, this contains the steps so far on the 9-element board: X or O
//                not in chronological order, but in the order of their location on the board
//        param3: with this parameter we will pass an event handler function (handlePlay)
function Board({ xIsNext, squares, param3 }) {
```

```
// if we click on the i. element of the table (array) with elements 0..8
  function handleClick(i) {
/ if there is a winner or we click on a place that has already been clicked: exit:
    if (calculateWinner(squares) || squares[i]) {
      return;
    }
// slice: returns a portion of the array. Since it has no parameters, it returns a copy of the array
//         https://www.w3schools.com/jsref/jsref_slice_array.asp
//         nextSquares: after adding the new step this will be the new position
    const nextSquares = squares.slice();
// If X is next, it places an X in the clicked location, otherwise it places an O.
    if (xIsNext) {
      nextSquares[i] = 'X';
    } else {
      nextSquares[i] = 'O';
    }
//         param3: with this parameter we will pass an event handler function (handlePlay)
    param3(nextSquares);
  }
// calculateWinner: determines whether there is already a winner based on the current standings
//         winner: the winner's sign (X or O), if there is no winner then null
  const winner = calculateWinner(squares);
// Put the current status into the status variable:
//         'Winner: ' …    OR 'Next player: ' ….
//         Write this above the board:    <div className="status">{status}</div>
  let status;
  if (winner) {
    status = 'Winner: ' + winner;
  } else {
    status = 'Next player: ' + (xIsNext ? 'X' : 'O');
  }

// Draw the board based on the current state of the squares array
  return (
    <>
      <div className="status">{status}</div>
      <div className="board-row">
        <Square value={squares[0]} onSquareClick={() => handleClick(0)} />
        <Square value={squares[1]} onSquareClick={() => handleClick(1)} />
        <Square value={squares[2]} onSquareClick={() => handleClick(2)} />
      </div>
      <div className="board-row">
        <Square value={squares[3]} onSquareClick={() => handleClick(3)} />
        <Square value={squares[4]} onSquareClick={() => handleClick(4)} />
        <Square value={squares[5]} onSquareClick={() => handleClick(5)} />
      </div>
      <div className="board-row">
        <Square value={squares[6]} onSquareClick={() => handleClick(6)} />
        <Square value={squares[7]} onSquareClick={() => handleClick(7)} />
        <Square value={squares[8]} onSquareClick={() => handleClick(8)} />
      </div>
    </>
  );
```

```
}

// it starts with this at starting:
export default function Game() {
// the two state variables:
//        history: 0..8 array: this contains the steps in chronological order, not in order of their location
//        on the table
  const [history, setHistory] = useState([Array(9).fill(null)]);
  const [currentMove, setCurrentMove] = useState(0);
// X is next?
  const xIsNext = currentMove % 2 === 0;
  const currentSquares = history[currentMove];

// nextSquares: new position after the move
//        array: 0..8, this contains the previous moves on the 9-element board: X or O
//                not in chronological order, but in the order of their location on the board
  function handlePlay(nextSquares) {
// …: spread operator:
//        The spread operator can be used to add elements to an array. It returns a new array after adding
//        some elements.
//        https://www.geeksforgeeks.org/add-elements-to-a-javascript-array/
//                const arr = [ 10, 20, 30, 40 ];
//                const newArr = [ ...arr, 50, 60 ];
//                console.log("Updated Array: ", newArr);
//                Updated Array:  [ 10, 20, 30, 40, 50, 60 ]
// Adds the new 9-element array to history
    const nextHistory = [...history.slice(0, currentMove + 1), nextSquares];
    setHistory(nextHistory);
// sets the step we are at:
    setCurrentMove(nextHistory.length - 1);
  }

// click the button: jumps to the given position
  function jumpTo(nextMove) {
    setCurrentMove(nextMove);
  }
// .map: goes through the elements of the history array (it contains small arrays of 9 elements).
//        at each step, the given small array is denoted by squares,
//        the given index in the large array is denoted by move: at the first step move=0
//        https://www.geeksforgeeks.org/javascript-array-map-method/
//        moves: creates a list: <li> elements, puts a button in each
//        we saw that by clicking on the button we can print a previous state
//        we then put the list elements in a numbered list: <ol>
//                <ol>{moves}</ol>
//        that's why it puts numbers in front of it
  const moves = history.map((squares, move) => {
    let description;
    if (move > 0) {
      description = 'Go to move #' + move;
    } else {
      description = 'Go to game start';
    }
    return (
```

1. Go to game start
2. Go to move #1
3. Go to move #2
4. Go to move #3
5. Go to move #4
6. Go to move #5
7. Go to move #6

```
      <li key={move}>
        <button onClick={() => jumpTo(move)}>{description}</button>
      </li>
    );
  });

  return (
    <div className="game">
      <div className="game-board">
        <Board xIsNext={xIsNext} squares={currentSquares} param3 ={handlePlay} />
      </div>
      <div className="game-info">
        <ol>{moves}</ol>
      </div>
    </div>
  );
}

// if there is a winner, then determine its sign (X or O), otherwise null
function calculateWinner(squares) {
// Checks if there are 3 identical elements in rows, columns, diagonals
// these are rows, columns, diagonals:
  const lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ];
// Checks if there are 3 identical elements in rows, columns, diagonals
  for (let i = 0; i < lines.length; i++) {
    const [a, b, c] = lines[i];
    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
      return squares[a];
    }
  }
  return null;
}
```

Run on a remote server

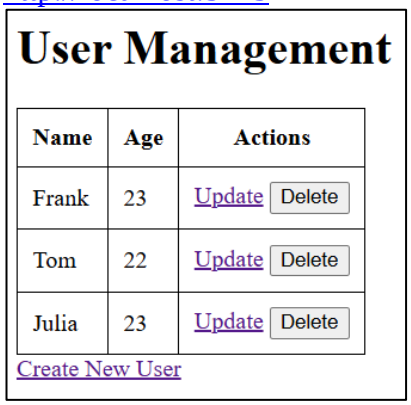
npm run build

Copy the contents of the **dist** folder to the remote server using FTP.
Enter the URL set on the hosting in a browser:
It works!

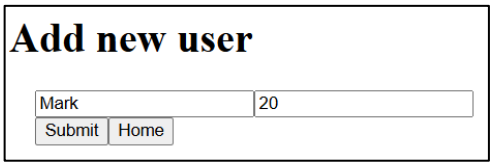
04-Crud app – With routes, parameters between routes, Redirect, data is stored in an array

Based on:
https://www.geeksforgeeks.org/how-to-do-crud-operations-in-reactjs/

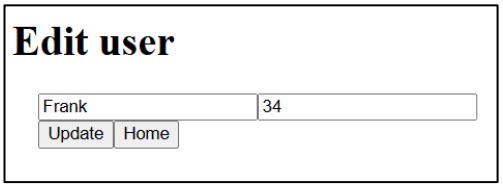
http://localhost:5173



http://localhost:5173/create



http://localhost:5173/edit




```
npm create vite@latest crud -- --template react
  cd crud
  npm install
npm i react-router-dom
```

main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

index.css

```
table,th,td {
    border: 1px solid black;
    border-collapse: collapse;
    padding: 10px;
}
```

App.jsx

```
import React from "react";
import {
    BrowserRouter as Router,
```

```
    Route,
    Routes,
} from "react-router-dom";
import "./App.css";
import Create from "./components/Create";
import Edit from "./components/Edit";
import Home from "./components/Home";

function App() {
    return (
        <div>
            <Router>
                <Routes>
                    <Route path="/" element={<Home />} />
                    <Route path="/create" element={<Create />}/>
                    <Route path="/edit" element={<Edit />}/>
                </Routes>
            </Router>
        </div>
    );
}
export default App;
```

The array may contain identical elements. If you want to avoid this, add also an id field.

components/array.jsx

```
const array = [
    {
        Name: "Frank",
        Age: "23",
    },
    {
        Name: "Tom",
        Age: "22",
    },
    {
        Name: "Julia",
        Age: "23",
    },
];
export default array;
```

components/Home.jsx

```
import React from "react";
import array from "./array";
import { Link, useNavigate } from "react-router-dom";

function Home() {
    let history = useNavigate();

    function deleted(index) {
        array.splice(index, 1); // Deleting the entry with the specified index
        history("/"); // Redirecting to the same page to re-render
```

```
    }

    return (
        <div style={{ margin: "20px" }}>
            <h1>User Management</h1>
            <table >
                <tbody>
                    <tr>
                        <th>Name</th>
                        <th>Age</th>
                        <th>Actions</th>
                    </tr>
                    {array.map((item, ind) => {
                        return (
                            <tr key={ind}>
                                <td>{item.Name}</td>
                                <td>{item.Age}</td>
                                <td>
                                    <Link to={`/edit`} state={{ index: ind, name: item.Name, age:
item.Age}}>Update</Link>   
                                    <button onClick={() => deleted(ind)}>Delete</button>
                                </td>
                            </tr>
                        );
                    })}
                </tbody>
            </table>
            <Link to="/create">Create New User</Link>
        </div>
    );
}
export default Home;
```

components/Create.jsx

```
import React, { useState } from "react";
import array from "./array";
import { Link, useNavigate } from "react-router-dom";

function Create() {
    // Making usestate for setting and fetching a value in jsx
    const [name, setname] = useState("");
    const [age, setage] = useState("");

    // Using useNavigation for redirecting to pages
    let history = useNavigate();

    // Function for creating a post/entry
    const handelSubmit = (e) => {
        e.preventDefault(); // Prevent reload

        // Fetching a value from usestate and pushing to javascript object
        let a = name, b = age;
        if (name == "" || age == "") {
```

```
            alert("invalid input");
            return;
        }
        array.push({ Name: a, Age: b });

        // Redirecting to home page after creation done
        history("/");
    };

    return (
        <>
            <h1>Add new user</h1>
            <form style={{ margin: "20px" }}>
                {/* Fetching a value from input textfirld in a setname using usestate*/}
                <input onChange={(e) => setname(e.target.value)} type="text" placeholder="Enter Name"
required />
                {/* Fetching a value from input textfirld in a setage using usestate*/}
                <input onChange={(e) => setage(e.target.value)} type="number" placeholder="Age" required
/><br />
                {/* handing a onclick event in button for firing a function */}
                <button onClick={(e) => handelSubmit(e)} type="submit">Submit</button>
                {/* Redirecting back to home page */}
                <Link to="/"><button variant="info">Home</button>
                </Link>
            </form>
        </>
    );
}
export default Create;
```

components/Edit.jsx

```
import React, { useEffect, useState } from "react";
import array from "./array";
import { Link, useNavigate, useLocation} from "react-router-dom";

function Edit() {
    // Here usestate has been used in order to set and get values from the jsx
    const [name, setname] = useState("");
    const [age, setage] = useState("");
    const [index, setindex] = useState("");

    // Used for navigation with logic in javascript
    let history = useNavigate();

    // Function for handling the edit and pushing changes of editing/updating
    const handelSubmit = (e) => {
        // Preventing from reload
        e.preventDefault();
        if (name == "" || age == "") {
            alert("invalid input");
            return;
        }
```

```
        // Getting an index of an array
        let a = array[index];

        // Putting the value from the input textfield and replacing it from existing for updation
        a.Name = name;
        a.Age = age;

        // Redirecting to main page
        history("/");
    };

    const location = useLocation();
    // Useeffect take care that page will be rendered only once
    useEffect(() => {
        setindex(location.state.index);
        setname(location.state.name);
        setage(location.state.age);
    }, []);

    return (
        <>
            <h1>Edit user</h1>
            <form style={{ margin: "20px" }}>
                {/* setting a name from the input textfiled */}
                <input value={name} onChange={(e) => setname(e.target.value)} type="text"
placeholder="Enter Name"/>
                {/* setting a age from the input textfiled */}
                <input value={age} onChange={(e) => setage(e.target.value)} type="number"
placeholder="Age"/><br />
                {/* Hadinling an onclick event running an edit logic */}
                <button onClick={(e) => handelSubmit(e)} type="submit" >Update</button>
                {/* Redirecting to main page after editing */}
                <Link to="/"><button>Home</button></Link>
            </form>
        </>
    );
}
export default Edit;
```

npm run dev

http://localhost:5173/

REACT sample applications on the web

https://www.codewithfaraz.com/article/196/25-react-projects-for-beginners-with-source-code
https://www.geeksforgeeks.org/reactjs-projects/
https://legacy.reactjs.org/community/examples.html
https://sean-smith.surge.sh/assets/portfolio/25-react-projects/index.html
https://www.freecodecamp.org/news/react-projects-for-beginners-easy-ideas-with-code/
https://codegnan.com/react-js-projects/

https://medium.com/design-bootcamp/top-5-react-projects-for-beginners-509c29c91336
https://www.zegocloud.com/blog/react-projects
https://devaradise.com/react-example-projects/
https://hackr.io/blog/react-projects

https://github.com/ianshulx/React-projects-for-beginners

https://www.freecodecamp.org/news/master-react-by-building-25-projects/
https://contactmentor.com/best-react-projects-for-beginners-easy/

10 – Object oriented JavaScript

We have already met object-oriented JavaScript, since **JavaScript is an object-oriented (OOP) programming language**. **All objects in JavaScript** (except primitives).
Even in the document.write(…) instruction used in the first exercises, the **document is an object** whose method is write(…).
So the **novelty** will not be object orientation, but e.g. class creation, constructor, inheritance, class, …

Javascipt is an object-oriented programming language from the beginning, but the class declaration (class) only appeared in ECMAScript 6 (2015). Before that, inheritance was implemented using prototypes and not classes.

OOP Javascript - using prototypes – Only informative text

Prototype, prototype-chain

Each object has a prototype data member that points to another object. This other object is called the prototype of the object in question. The prototype of the object also has a prototype, and so on. We call this prototype chain. At the end of the chain is an object pointing to the null object.

Inheritance based on prototypes - example

First, we create a FooBase class, which will have a variable, and this variable will have getter and sette methods, and a toString method. This will be the parent class.

```
// Class constructor:
function FooBase(value) {
	this.value = value;
}
FooBase.prototype.setValue = function(value){
	this.value = value;
	return this;
}
FooBase.prototype.getValue = function(){
	return this.value;
}
FooBase.prototype.toString = function(){
	return this.getValue();
}
```

Now we create a FooChild class that inherits the properties of our FooBase class. We derive from the FooBase class:

```
FooChild.prototype = new FooBase();
// Define the FooChild class:
function FooChild(){
// We call the ancestor's constructor:
FooBase.apply( this, arguments );
}
// Override the toString() method:
FooChild.prototype.toString = function(){
return 'FooChild value értéke: ' + this.getValue();
```

```
}
myFooBase = new FooBase(0);
baseToString = myFooBase.toString();
document.write(baseToString);        // => 0
myFooChild = new FooChild(0);
childToString = myFooChild.toString();
document.write(childToString);       // => 0
```

1-2 exercises

Solve tasks first by using prototypes and then by using the language elements introduced in ECMAScript 6 to define classes.

1st exercise

Create an **Image** class for handling image objects, which:

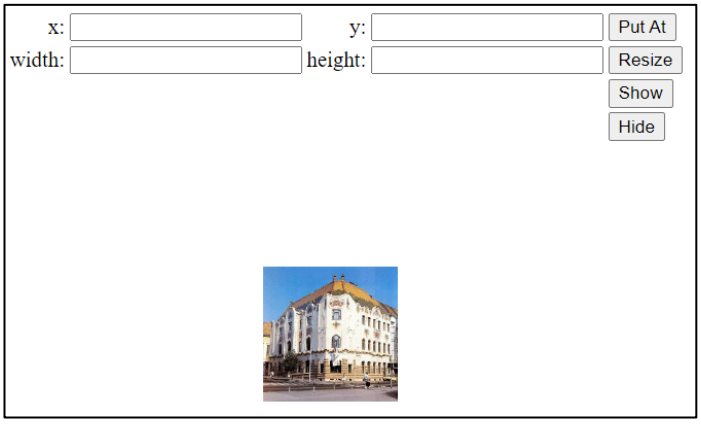
a) Its constructor receives the following parameters:
- **src**: the url of the source file of the image
- **x, y**: the position of the upper left corner of the image within the browser window,
- **width, height**: dimensions of the image.

b) Its constructor initializes the **img** attribute of the object based on the received parameters (to create it, use: **document.createElement("img");** to place it in the **DOM**, use: **document.body.appendChild(this.image)** and set the appropriate style properties.

c) Its methods are as follows:
- **show**: displays the image in the browser,
- **hide**: hides the image,
- **putAt(x,y):** moves the upper left corner of the image to the position received as a parameter,
- **resize(width,height)**: resizes the image to the size given as a parameter.

d) Test the developed class: create an html file for it.

The task includes the file **example.jpg**, which you can find in the solutions.



2nd exercise

As in the previous task, there is also a subtitle under the image, which must be moved together with the image. We use the previous **Image** class and derive the **SubtitledImage** class from it.

A, Soluion – with prototypes – Only informative text

Can be found in **Solutions.zip**

B, Solution – With classes

B/1. solution – without subtitle
We create 2 files: **obj.js, obj.html**

obj.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Objects in JavaScript</title>
   <script type="text/javascript" src="obj.js">
   </script>
</head>
<body>
   <script type="text/javascript">
// Calling with the new operator returns an object
//         places the given image in the given location with the given size
// Create the Image object constructor in obj.js
//         200, 200: upper left corner x,y coordinate
//         100, 100: width, height
      var myimage = new Image("example.jpg", 200, 200, 100, 100);
   </script>
// puts elements in a table
   <table>
      <tr>
// text boxes for entering data
      <td align="right">x:</td><td><input type="text" id="x" name="x"></td>
      <td align="right">y:</td><td><input type="text" id="y" name="y"></td>
// To press the Put At button, call the putAt method of the myimage object.
// Places the image in the location specified in the two text boxes
      <td><button onclick="myimage.putAt(x.value,y.value)">Put At</button></td>
      </tr>
      <tr>
      <td align="right">width:</td><td><input type="text" id="width" name="width"></td>
      <td align="right">height:</td><td><input type="text" id="height" name="height"></td>
// Resize the image to the value specified in the two text boxes
      <td><button onclick="myimage.resize(width.value,height.value)">Resize</button></td>
      </tr>
      <tr>
      <td colspan="4"></td>
// show the image
      <td><button onclick="myimage.show()">Show</button></td>
      </tr>
      <tr>
      <td colspan="4"></td>
// Hides the image
      <td><button onclick="myimage.hide()">Hide</button></td>
      </tr>
   </table>
</body>
```

```
</html>
```

obj.js

```
class Image {
// Constructors can be defined with the optional constructor attribute.
// We do not need to specify it if the name of the creating function is the same as that of the class.
// This will be the so-called default constructor
// Creates and places the image object on the page when instantiated:
    constructor(src, x, y, width, height) {
// this.img = … We define a (global) variable (property) img for the class
        this.img = document.createElement("img");
        this.img.src = src;
        this.img.style.position = "absolute";
        this.img.style.left = x+"px";
        this.img.style.top = y+"px";
        this.img.width = width;
        this.img.height = height;
        document.body.appendChild(this.img);
    }
    show() {
        this.img.style.visibility = "visible";
    }
    hide() {
        this.img.style.visibility = "hidden";
    }
    putAt(x, y) {
        this.img.style.left = x+"px";
        this.img.style.top = y+"px";
    }
    resize(width, height) {
        this.img.width = width;
        this.img.height = height;
    }
}
```

B/2. solution – With subtitle

objsub.html - is the same as at A/2 solution

```
<!DOCTYPE html>
<html>
<head>
    <title>Objects in JavaScript</title>
    <script type="text/javascript" src="objsub.js">
    </script>
</head>
<body>
    <script type="text/javascript">
        var myimage = new SubtitledImage("example.jpg", 200, 200, 100, 100, "With subtitle");
    </script>
    <table>
        <tr>
        <td align="right">x:</td><td><input type="text" id="x" name="x"></td>
```

```
      <td align="right">y:</td><td><input type="text" id="y" name="y"></td>
      <td><button onclick="myimage.putAt(x.value,y.value)">Put At</button></td>
      </tr>
      <tr>
      <td align="right">width:</td><td><input type="text" id="width" name="width"></td>
      <td align="right">height:</td><td><input type="text" id="height" name="height"></td>
      <td><button onclick="myimage.resize(width.value,height.value)">Resize</button></td>
      </tr>
      <tr>
      <td colspan="4"></td>
      <td><button onclick="myimage.show()">Show</button></td>
      </tr>
      <tr>
      <td colspan="4"></td>
      <td><button onclick="myimage.hide()">Hide</button></td>
      </tr>
   </table>
</body>
</html>
```

objsub.js

```
// this class is the same as in obj.js
class Image {
   constructor(src, x, y, width, height) {
      this.img = document.createElement("img");
      this.img.src = src;
      this.img.style.position = "absolute";
      this.img.style.left = x+"px";
      this.img.style.top = y+"px";
      this.img.width = width;
      this.img.height = height;
      document.body.appendChild(this.img);
   }

   show() {
      this.img.style.visibility = "visible";
   }

   hide() {
      this.img.style.visibility = "hidden";
   }

   putAt(x, y) {
      this.img.style.left = x+"px";
      this.img.style.top = y+"px";
   }

   resize(width, height) {
      this.img.width = width;
      this.img.height = height;
   }
```

```
}

class SubtitledImage extends Image {
   constructor(src, x, y, width, height, subtitle) {
// calls the default constructor of the parent class (Image):
// Creates and places the image object on the page on instantiation
      super(src, x, y, width, height);
      this.subtitle = document.createElement("div");
      this.subtitle.innerHTML = subtitle;
      this.subtitle.style.position = "absolute";
      this.subtitle.style.left = x+"px";
      this.subtitle.style.top = (y+height)+"px";
      document.body.appendChild(this.subtitle);
   }

   show() {
// Calls the show() method of the parent class (Image).
      super.show();
      this.subtitle.style.visibility = "visible";
   }

   hide() {
      super.hide();
      this.subtitle.style.visibility = "hidden";
   }

   putAt(x, y) {
      super.putAt(x, y);
      this.subtitle.style.left = x+"px";
      this.subtitle.style.top = (parseInt(y)+parseInt(this.img.height))+"px";
   }

   resize(width, height) {
      super.resize(width, height);
      this.subtitle.style.top = (parseInt(this.img.style.top)+parseInt(height))+"px";
   }
}
```

3rd exercise

Write a JavaScript source file called exercise3.js that defines the following classes:

1. Define the **UniversityEmployee** class, whose member variables are **name**, **address** and **salary**; the **salaryModify** method adds the value received as a parameter to the value of the salary attribute.
2. Define the **Teacher** class, which is an extension of the **UniversityEmployee** class, its additional data members are **department** and the initially empty **courses** array, its methods are the **course** that takes the subject received as a parameter, and **numberCourses**, which returns the number of subjects.

The result of executing the specified test.html can be seen in the image:

Initial table

Name	Address	Salary	Department	Courses
John Smith	Budapest	450000	---	---
Kate Brown	Kecskemét	500000	Informatics	0:
Tom Green	Szeged	475000	Mechanics	0:

Table after modification

Name	Address	Salary	Department	Courses
John Smith	Budapest	500000	---	---
Kate Brown	Kecskemét	500000	Informatics	2: Programming 1,Programming 2
Tom Green	Szeged	500000	Mechanics	0:

Solution

exercise3.js

```
class UniversityEmployee {
    constructor(name, address, salary) {
        this.name = name;
        this.address = address;
        this.salary = salary;
    }

    salaryModify(addition) {
        this.salary += addition;
    }
}

class Teacher extends UniversityEmployee {
    constructor(name, address, salary, department) {
        super(name, address, salary);
        this.department = department;
        this.courses = [];
    }

    course(course) {
        this.courses.push(course);
    }
    numberCourses() {
        return this.courses.length;
    }
}
```

test.html

```
<!DOCTYPE html>
<html>
<head>
    <title>3rd exercise</title>
    <meta charset="utf-8">
    <script type="text/javascript" src="exercise3.js">
    </script>
    <script type="text/javascript">
```

```
      var objects = [];
      objects.push(new UniversityEmployee('John Smith', 'Budapest', 450000));
      objects.push(new Teacher('Kate Brown', 'Kecskemét', 500000, 'Informatics'));
      objects.push(new Teacher('Tom Green', 'Szeged', 475000,'Mechanics'));
      console.log(objects);
   </script>
</head>
<body>
   <table border="1">
      <caption>Initial table</caption>
      <tr><th>Name</th><th>Address</th><th>Salary</th><th>Department</th><th>Courses</th></tr>
      <script type="text/javascript">

document.write("<tr><td>"+objects[0].name+"</td><td>"+objects[0].address+"</td><td>"+objects[0].sa
lary+"</td><td>---</td><td>---</td></tr>");

document.write("<tr><td>"+objects[1].name+"</td><td>"+objects[1].address+"</td><td>"+objects[1].sa
lary+"</td><td>"+objects[1].department+"</td><td>"+objects[1].numberCourses()+":
"+objects[1].courses+"</td></tr>");

document.write("<tr><td>"+objects[2].name+"</td><td>"+objects[2].address+"</td><td>"+objects[2].sa
lary+"</td><td>"+objects[2].department+"</td><td>"+objects[2].numberCourses()+":
"+objects[2].courses+"</td></tr>");
      </script>
   </table>
   <script type="text/javascript">
      objects[0].salaryModify(50000);
      objects[2].salaryModify(25000);
      objects[1].course("Programming 1");objects[1].course("Programming
2");objects[1].tantargy("Visual programming");
      objects[2].course("Web-programming I");objects[2].course("Web-programming II");
   </script>
   <br>
   <table border="1">
      <caption>Table after modification</caption>
      <tr><th>Name</th><th>Address</th><th>Salary</th><th>Department</th><th>Courses</th></tr>
      <script type="text/javascript">

document.write("<tr><td>"+objects[0].name+"</td><td>"+objects[0].address+"</td><td>"+objects[0].sa
lary+"</td><td>---</td><td>---</td></tr>");

document.write("<tr><td>"+objects[1].name+"</td><td>"+objects[1].address+"</td><td>"+objects[1].sa
lary+"</td><td>"+objects[1].department+"</td><td>"+objects[1].numberCourses()+":
"+objects[1].courses+"</td></tr>");

document.write("<tr><td>"+objects[2].name+"</td><td>"+objects[2].address+"</td><td>"+objects[2].sa
lary+"</td><td>"+objects[2].department+"</td><td>"+objects[2].numberCourses()+":
"+objects[2].courses+"</td></tr>");
      </script>
   </table>
</body>
</html>
```

11. PHP basics

Under development…