Lab 1

In this lab, we're going to cover some more HTML and CSS, and have some hands-on with them and with JS.

IDE

Web Development can be done in basically any text editor (even Microsoft Word, but please don't) since it doesn't need any special tools/compilers. Popular choices include code editors such VS Code (https://code.visualstudio.com/), Notepad++ (https://notepad-plus-plus.org/), or Sublime Text (https://www.sublimetext.com/), CLI editors such as VIM (https://www.vim.org/) or nano, or even full-fledged IDEs such as WebStorm (https://www.jetbrains.com/webstorm/). An editor that supports syntax highlighting and basic Git is more than enough for us, and we will go with **VS Code**.

You also need a web browser such as Google Chrome, Firefox, Edge, or Safari. It's always a good idea to make sure your website works on as many browsers as possible, but we're going to stick with **Chrome** for now since it's the most popular.

VS Code supports HTML/CSS/JS syntax highlighting and hover hints out of the box. We are also going to use it with the Live Server (https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer) extension to not worry about the hosting.

Lecture Review

HTML

First, let's take a look at the generic `HTMLElement` (https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement). We can see the Document Object Model (DOM (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction)) representation of an HTML element, which is the API we use in JavaScript to read/modify the element. We can only get these objects through native functions (such as `document.getElementById`). It consists of **properties** (which can be either writable or read-only), **methods**, and **events** emitted by this element that we can listen to. The MDN page also contains a link to the **standard** specification and a **browser compatibility** matrix.

> *Side note: The DOM is actually designed to be language-independent, meaning languages other than JavaScript can also implement it, and even theoretically replace JS if the browser supports it.*

MDN also has a page that lists all HTML elements (https://developer.mozilla.org/en-US/docs/Web/HTML/Element). Let's go through some of the important ones:

- `html` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/html): the root (top-level element) of an HTML document. Everything else should be inside of it.
- `head` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/head): contains metadata about the document, such as its title (appears on the tab head), the scripts and style sheets it needs, and more.
- `body` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/body): the actual content the users see and can interact with.

Consider this as boilerplate (like `class Main { public static void main(String args[]) {} }` in Java 😁) you need to wrap around your document.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <!-- Some content -->
  </body>
</html>
```

There is also the preamble `<!DOCTYPE HTML>` (https://developer.mozilla.org/en-US/docs/Glossary/Doctype) that basically tells the browser we are abiding by the HTML standard (thank you, Internet Explorer 🙄). *Read more here (https://hsivonen.fi/doctype/) if interested*.

Going further, we have:

- `h1` - `h6` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements): section **h**eadings. Think of them as the table of contents.
- `header` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/header) and `footer` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/footer): as their names suggest.
- `nav` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/nav): **nav**igation links. Usually used within the `header`.
- `main` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/main): the important content of the page (compare to `header` / `footer` / `aside`).
- `p` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p): a **p**aragraph of text.
- `div` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div): content **div**ision. Doesn't represent anything. Used only to group content and apply styling.
- `ol` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ol)/ `ul` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ul) and `li` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/li): **o**rdered/**u**nordered **l**ist and **l**ist **i**tem. `li` must be inside `ul` or `ol`.
- `a` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/a): **a**nchor element, creates a link.
- `img` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img): an **im**a**g**e.

- `button` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button): try to guess this one on your own :)
- *and a lot more…*

Important note on semantics: different elements *mean* different things. Don't use an `h3` tag if all you want is just bigger text (use the CSS `font-size` instead), and don't use a `p` tag for captioning a media element. This helps your website be more *accessible*.

But where do we write our CSS styles and JavaScript logic? Undoubtedly, in:

- `style` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/style): contains the CSS styles. Must be in the `head`.
- `script` (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script): contains JavaScript code to run.

Keep in mind that the elements are processed in order, top to bottom. This will become important later.

CSS

Cascading Style Sheets (https://developer.mozilla.org/en-US/docs/Web/CSS) is the language used to add styles to our elements, describing how they should be rendered.

Common properties:

- `width` (https://developer.mozilla.org/en-US/docs/Web/CSS/width)/ `height` (https://developer.mozilla.org/en-US/docs/Web/CSS/height)
- `color` (https://developer.mozilla.org/en-US/docs/Web/CSS/color): text color
- `background-color` (https://developer.mozilla.org/en-US/docs/Web/CSS/background-color): you guess this one on your own
- `border` (https://developer.mozilla.org/en-US/docs/Web/CSS/border): shorthand property for the width, style, and color.
- `margin` (https://developer.mozilla.org/en-US/docs/Web/CSS/margin)/ `padding` (https://developer.mozilla.org/en-US/docs/Web/CSS/padding): both control spacing around the content, but `padding` adds the space inside the container box while `margin` adds it outside it.
- `position` (https://developer.mozilla.org/en-US/docs/Web/CSS/position): controls the element's position strategy in a document
- `display` (https://developer.mozilla.org/en-US/docs/Web/CSS/display): defines whether the element is treated as a block or inline element, as well as the layout strategy used for its children (grid, flex, …)

Colors can use the pre-defined strings (`black` , `red` , `antiquewhite` , …), hex values (`#123456` , `#abc`), or RGB(A) values (`rgb(0, 0, 0)` , `rgba(255, 128, 73, 0.4)`).

For defining lengths, we have absolute and relative units. The most common absolute unit is `px` (pixel) (https://developer.mozilla.org/en-US/docs/Glossary/CSS_pixel). Relative units include percentages (https://developer.mozilla.org/en-US/docs/Web/CSS/percentage), `em` (font size of the parent), `vw` / `vh` (viewport width/height, think of it as `%` of screen). It's better for responsiveness (mobile-friendliness) in general to use relative units wherever possible.

Shorthand properties are properties that combine multiple other properties in one line. For example, setting `border: 2px dashed black` is equivalent to writing

```
border-width: 2px;
border-style: dashed;
border-color: black;
```

They may also not map the values 1:1 to properties, such as with `padding`. Setting `padding: 1em` will set it in all 4 directions (top, bottom, left, right), while setting `padding: 5px 20px` would use the first value for top/bottom and the second for left/right, and so on.

CSS also contains some built-in functions, such as `rgb()` (https://developer.mozilla.org/en-US/docs/Web/CSS/color_value/rgb), `url()` (https://developer.mozilla.org/en-US/docs/Web/CSS/url), and `calc()` (https://developer.mozilla.org/en-US/docs/Web/CSS/calc) (used to perform calculations using various units).

JS

If you already coded in Java/C++ before, then you're already familiar with the syntax and all you need is a JavaScript CheatSheet (https://htmlcheatsheet.com/js/) and you're good to go.

We also need to get familiar with the "standard library" provided by the browsers. We have already seen the `window` object as the global container of everything. We came across the `document` object that allows us to interact with the DOM, so let's dive deeper into that. One commonly used function is `document.getElementById()`, which returns the DOM object corresponding the HTML element given the special attribute `id`. It returns an `Element` (https://developer.mozilla.org/en-US/docs/Web/API/Element) object that contains all information about the element including its attributes, children, location in document, width/height, and more, as well as methods allowing to interact with the object, the most common of which is the `.addEventListener()` that allows us to register handlers for various events that occur on this element (such as clicking).

We can also display data using `console.log` (for debugging) or `alert` (for the user), and we can `prompt` the user for input.

DevTools

To debug our apps, we can use the Developer Tools integrated into the browser. Typically, you can launch them using the shortcut `CTRL+SHIFT+I` or by right clicking anywhere and selecting the option "*Inspect Element*". Your mileage may vary depending on the chosen browser.

They include so many useful tools, but we are currently interesting in only 2 of them:

Elements

This tab shows the HTML and CSS of your page. It is interactive, so you can click on an element and it will highlight it on the page and show you the associated styles. The styles are grouped by the selector and you can modify or (de)activate each property (or even add new ones) to see its effects right away. It will also show when a property is overriden in another selector.

Console

In the console tab, you can view any data you log to the console from your application (using `console.log` or any of the other `console.` methods) and you can run arbitrary JavaScript code there, having access to all the global variables in your code. You also have access to the special variable `$0` denoting the currently selected element in the **Elements** tab (`$` is a valid identifier character in JS).

Task

Now finally the fun part! For the labs, we are going to work on individual projects that we will build on every lab, which will be a personal website that can act as a portfolio for each of you.

But first, let's practice some HTML and JS by writing a website to convert temperatures from degrees Fahrenheit (why does it even still exist anyway?) to degrees Celsius:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Lab 1</title>
    </head>
    <body>
        <label for="fahrenheit">Enter the amount in Fahrenheit:</label>
        <input id="fahrenheit" type="number" placeholder="Example: 32" />
        <button id="convert">Convert</button>
        Result: <p id="result">awaiting input...</p>

        <script>
            const inputEl = document.getElementById('fahrenheit');
            const convertButton = document.getElementById('convert');
            const resultEl = document.getElementById('result');
            convertButton.addEventListener('click', function() {
                const fahrenheit = parseInt(inputEl.value);
                // TODO: validate that the input value is a valid number
                const celsius = (5/9) * (fahrenheit - 32);
                resultEl.innerHTML = `${fahrenheit}&deg;F is ${celsius}&deg;C`;
            });
        </script>
    </body>
</html>
```

Now let’s get to the actual lab project. We would like to build a personal portfolio website where you will be able to document the projects you worked and anything else you want people to see about you. Feel free to be creative and add multiple pages. You can check this (https://collegeinfogeek.com/personal-website-examples/), this (https://dribbble.com/tags/simple_portfolio), or this (https://www.freelancer.com/community/articles/must-visit-personal-portfolio-websites) for some examples and inspiration, or simply search on Google. For simpler examples, you can check GitHub Profile READMEs, such as this one (https://github.com/illright/illright/blob/main/README.md), or this full list of examples (https://github.com/abhisheknaiidu/awesome-github-profile-readme).

Here is something to get you started:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ivan's website</title>
        <style>
            body {
                padding: 0 200px;
            }
            a {
                margin: 20px;
            }
        </style>
    </head>
    <body>
        <header>
            <h1>Ivan Ivanovich Ivanov</h1>
            <nav>
                <a href="/about.html">About Me</a>
                <a href="/photos.html">Photos</a>
                <a href="/portfolio.html">Portfolio</a>
                <a href="/contact.html">Contact Me</a>
            </nav>
        </header>
        <main>
            <!-- Add some content here -->
        </main>
    </body>
</html>
```

`href` = **h**ypertext **ref**erence
`px` = **p**i**x**els (note that it's better to use `em` and `rem` where possible)
Relative vs. Absolute URLs (https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL#absolute_urls_vs_relative_urls)

Homework

Add some content to your website. For example, add a section to the page that contains social links (email, GitHub, Telegram, …), but have them be icon links (use the logo as the image).

The more beautiful your website is, the higher your grade can be, so try to use CSS to make it as aesthetically pleasing as you can.

The exact requirements and criteria can be found on Moodle.