Web Application Development
(LTAT.05.004)

CSS
CASCADING STYLE SHEETS

MOHAMAD GHARIB
UNIVERSITY OF TARTU
HTML - Recap

Index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>HTML</title>
  </head>
  <body>
    <h1>This is a heading (h1)</h1>
    This is Paragraph 1
    Nested tags
  </body>
</html>
```
HTML5 - Recap

HTML4

- <div id="header">
- <div id="nav">
- <div id="sidebar">
  - <div id="content">
  - <div id="post">
  - <div id="post">
- <div id="footer">

HTML5

- <header>
- <nav>
- <main>
  - <article>
  - <article>
  - <article>
- <footer>
CSS

Cascading Style Sheets
Before we start ...

What do you need to know before starting?
- HTML and Basic knowledge of Document Object Model (DOM).

What we will cover:
- What CSS is,
- CSS Syntax,
- How we can use CSS to style our webpages.

What you are expected to learn:
- Core aspects of CSS,
- How you can use them to style your webpages.

How this lecture contributes to the course

HTML
The structure of the webpage

CSS
The styling of the webpage

JavaScript
Makes a webpage interactive
CSS (Cascading Style Sheets) is a style sheet language that is used for describing the presentation of a document written in a markup language like HTML.

CSS describes **how** content is to be displayed, not **what** is being displayed.

CSS rules control the look (Style) of web pages, they can change text size and color, control text formatting, control elements location on the page, etc.

CSS can be **embedded** into HTML document, HTML element or placed into separate `.css` file.
CSS short history

CSS was proposed by Håkon Wium Lie in 1994. CSS has various levels.

CSS level 1 (1996)
CSS level 2 (1998)
CSS level 3 (1999 - ): specification structured in modules, which can evolve independently. Some modules are already reached level 4.

Why did CSS have to evolve? Because the web and HTML have evolved.
CSS are rules.

Each rule consists of a selector and a declaration.

Each declaration is made up of a property and a value.

```css
selector {
  property: value;
}
```

```css
h1 {
  font-size: 24px
}
```
Why we needed CSS?
Before CSS ...

<table width=105 cellpadding=1 cellspacing=0 border=0 bgcolor="336699">
<tr valign=top><td width=105 align=center>
<font face=arial,helvetica size=-1 color=white><b>News Break</b></font></td></tr>
<tr valign=top><td>
<table cellpadding=5 cellspacing=0 border=0>
<tr valign=top bgcolor=white>
<td><a href="/ajc/living/movies/0902/24barber.html"><img src="/ajc/home/images/barbershop0925.jpg" width=100 height=81 border=0></a></td>
</tr>
</table>
<font size=1 face=arial,helvetica color=black>Ice Cube</font>

Source: http://www.holovaty.com/writing/125/
Why do we need to use CSS?

Consider you want to create a website with `<h1>` tags that have red color, you have to code each `<h1>`, as follows:

```
<h1><font color="red"> This is a red H1 tag </font></h1>
```

It is not feasible to type that for each `<h1>` tag.

Using CSS, all you need to do is type a regular `<h1>` tag, and add a style rules for `<h1>` in the Style Sheet, as follows:

```
h2 {color: red}
```

If we want to change the color of `<h1>` tags to another color, we just need to do a minor change in the CSS sheet.

The HTML `<font>` tag was used in HTML4, but no longer supported in HTML5
To understand how CSS works, we need to understand how a browser displays HTML documents. A browser processes an HTML document in a number of stages:

1. The browser loads the HTML document.
To understand how **CSS** works, we need to understand how a **browser** displays **HTML** documents. A browser processes an **HTML** document in a number of **stages**:

2. The browser **parses** the **HTML** document.
To understand how CSS works, we need to understand how a browser displays HTML documents. A browser processes an HTML document in a number of stages:

3. The browser creates the DOM (Document Object Model).
To understand how *CSS* works, we need to understand how a *browser* displays *HTML* documents. A browser processes an *HTML* document in a number of *stages*:

4. The browser *fetches* most of the *resources* that are linked to by the *HTML* document (e.g., embedded images, videos, and linked *CSS files*).
To understand how CSS works, we need to understand how a browser displays HTML documents. A browser processes an HTML document in a number of stages:

5. The browser creates CSS Object Model (CSSOM) - parses the fetched CSS, and sorts the different rules by their selector types (e.g., element, class, ID)
To understand how **CSS** works, we need to understand how a **browser** displays **HTML** documents. A browser processes an **HTML** document in a number of **stages**:

6. The browser **creates** the **Render Tree** - contains only visible nodes - based on the **DOM** and **CSSOM**.
To understand how CSS works, we need to understand how a browser displays HTML documents. A browser processes an HTML document in a number of stages:

7. The browser calculates the positions and dimensions of each node on screen.
To understand how CSS works, we need to understand how a browser displays HTML documents. A browser processes an HTML document in a number of stages:

1. Load HTML
2. Parse HTML
3. Create DOM tree
4. Create Render tree
5. Load CSS
6. Parse CSS and creates CSSOM
7. Render Tree is generated that contain only visible nodes.
8. It calculates the positions and dimensions of each node on screen.
9. It converts each node in the render tree to actual pixels on screen.

The browser converts each node in the render tree to actual pixels on screen.
DOM, CSSOM, and Render Tree – in brief

Document Object Model (DOM)
CSS Object Model (CSSOM)
Document Object Model (DOM)

```html
<!DOCTYPE html>
<html>
  <head>
    <meta ...
    <title>...
  </head>
  <body>
    <h1>...</h1>
    <p>...</p>
    <ul>
      <li>...</li>
      <li>...</li>
      <li>...</li>
      <li>...</li>
    </ul>
    <p>...</p>
  </body>
</html>
```
<!DOCTYPE html>
<html>
<head>
    <meta ... />
    <title>...</title>
</head>
<body>
    <h1>...</h1>
    <p>...</p>
    <ul>
        <li>...</li>
        <li>...</li>
        <li>...</li>
    </ul>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
<meta ...
<title>...
</title>
</head>
<body>
<h1>...</h1>
<p>...</p>
<ul>
<li>...</li>
<li>...</li>
<li>...</li>
<li>...</li>
</ul>
</body>
</html>
DOM, CSSOM, and Render Tree

```html
<!DOCTYPE html>
<html>
<head>
    <meta ...
    <title>...</title>
</head>
<body>
    <h1>...</h1>
    <p>...</p>
    <ul>
        <li>...</li>
        <li>...</li>
        <li>...</li>
    </ul>
    <p>...</p>
</body>
</html>
```
Selectors

Selectors are patterns used to select the element(s) you want to style.
Universal selector

The CSS universal selector (*) matches elements of any type.

HTML

```html
<p>This is a paragraph.</p>
<div>This is a division.</div>
```

CSS

```css
*{
  color: green
}
```

Browser

This is a paragraph.
This is a division.

Comments in CSS can be enclosed within /* comment */
The CSS type selector matches elements by node name. In other words, it selects all elements of the given type within a document.

**HTML**

```html
<p> This is a paragraph. </p>
```

**CSS**

```css
p {
    Color: red
}
```

**Browser**

This is a paragraph.
The CSS **class selector** matches elements based on the contents of their **class attribute**.
Multiple classes selector

The CSS **multiple class selector** matches all elements that have all listed classes.

**HTML**

```html
<p class="p1 p2">This is a paragraph.</p>
```

**CSS**

```
.p1.p2 {Color: green}
```

**Browser**

This is a paragraph.
Grouping styles and selectors

Each *rule* can include *multiple styles* using semicolons to separate them:

```css
h1{
    Color: blue;
    font-style: italic;
}
```

*Multiple selectors* that have the same *styles* can be grouped using commas to separate them:

```css
h1, h2, h3{
    Color: blue;
}
```
The CSS ID selector matches an element based on the value of its id attribute.

Only one id attribute is allowed per one HTML element and it’s value has to be unique on that page.

**HTML**

```html
<p id="pId">This is a paragraph.</p>
```

**CSS**

```css
#pId {
  Color: blue;
}
```

**Browser**

This is a paragraph.
**Attribute selector**

The CSS **attribute selector** matches elements based on the presence or **value of a given attribute**.

**HTML**

```html
<p title="t1">This is a t1 paragraph.</p>
<p title="t2">This is a t2 paragraph.</p>
```

**CSS**

```
#p[title] {font-weight: bold}
#p[title = "t2"] {font-weight: lighter}
```

**Browser**

This is a t1 paragraph.

This is a t2 paragraph.
Adjacent sibling combinator

The adjacent sibling combinator (+) separates two selectors and matches the second element only if it immediately follows the first element, and both are children of the same parent element.

**HTML**

```html
<div>This is a division.</div>
<p>This is paragraph 1.</p>
<p>This is paragraph 2.</p>
```

**CSS**

```css
div + p {color: green;}
```

**Browser**

This is a division.
This is paragraph 1.
This is paragraph 2.
General sibling combinator

The general sibling combinator (~) separates two selectors and matches the second element only if it follows the first element (though not necessarily immediately), and both are children of the same parent element.

HTML

```html
<div>This is a division.</div>
<p>This is paragraph 1.</p>
<p>This is paragraph 2.</p>
```

CSS

```
div ~ p {color: green;}
```

Browser

This is a division.
This is paragraph 1.
This is paragraph 2.
Child combinator

The child combinator (>) is placed between two CSS selectors. It matches only those elements matched by the second selector that are the immediate children of elements matched by the first.

**HTML**

```html
<div>
  <span>Span inside div.
    <span>Span inside Span. </span>
  </span>
</div>

<span>Span outside div. </span>
```

**CSS**

```css
span {color: blue;}
div > span {color: green;}
```

**Browser**

- Span inside div. Span inside span.
- Span outside div.
The **descendant combinator**, represented by a single space ( ) character, combines **two selectors** such that elements matched by the **second selector** are selected if they have an **ancestor** element matching the **first selector**.

**HTML**

```html
<div>
  <span>Span inside div.
    <span>Span inside Span. </span>
  </span>
</div>

</span> Span outside div. </span>

**CSS**

```
span {color: blue;}
div span {color: green;}
```

**Browser**

```
Span inside div. Span inside span.
Span outside div.
```
Pseudo-classes allow applying a style to an element not only in relation to the content of the DOM tree, but also in relation to external factors like the history of the navigator (e.g., :visited), or the position of the mouse (e.g., :hover).

```css
selector: pseudo-class {
  property: value;
}

button:hover {
  background-color: green;
}

a:active {
  color: red;
}
```
A CSS pseudo-element is a keyword added to a selector that lets you style a specific part of the selected element(s).

```
selector: pseudo-element {
    property: value;
}
```

```
p::first-line {
    color: red;
}
```

```
p::first-letter {
    font-size: xx-large;
}
```
Where do you put the styles?
Where do you put the styles?

Style information can be located:

- Externally to the pages, in a separate file.
- Within each page.
- Inline with individual tags.
External styles

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

Style information can be located:
- Externally to the pages, in a separate file.
- Within each page.
- Inline with individual tags.
Internal styles

<!DOCTYPE html>
<html>
<head>
<style>
body { background-color: black; }
</style>
</head>
<body>
...
</body>
</html>

Style information can be located:

• Externally to the pages, in a separate file.
• **Within each page.**
• Inline with individual tags.
Inline styles

Style information can be located:

- Externally to the pages, in a separate file.
- Within each page.
- Inline with individual tags.

<h2 style="color: red;">This is a paragraph.</h2>
Cascade
Why the term “cascading”? 

In CSS, multiple styles can be applied to a particular element.

The browser will interpret these styles in a top-down (Cascading) fashion.

All matching rules for a particular selector will be applied, except where they conflict with each other. In this case, the last rule to be declared is applied.

HTML

```html
<h2>This is h2</h2>
```

CSS

```css
h2 {font-style: italic;}
h2 {color: darkblue;}
```  

Browser

```
This is h2
```
CSS Specificity

Selectors might have different specificities.
When two or more CSS rules that point to the same element conflict, the selector with the highest specificity value will apply its style to that HTML element.
Selectors might have different specificities.

When two or more CSS rules that point to the same element conflict, the selector with the highest specificity value will apply its style to that HTML element.

HTML

```html
<h1 class="Myh1">My Heading</h1>
```

CSS

```css
.Myh1 { color: blue; }

h1 { color: green; }
```

Browser

My Heading
Selectors might have different specificities.
When two or more CSS rules that point to the same element conflict, the selector with the highest specificity value will apply its style to that HTML element.

```
<html>
<head>
    <style>
        .MyH1 { color: blue; }
        h1 { color: green !important; }
    </style>
</head>
<body>
    <h1 class="MyH1">My Heading</h1>
</body>
</html>
```
The Box Model
The **CSS box model** is a box that wraps around every HTML element. It consists of:

- **Content**, where text and images appear.
- **Padding** clears an area around the content.
- **Border** goes around the padding and content.
- **Margin** clears an area outside the border.

**Note:** the **padding** and **margin** boxes are transparent.

[Visit W3Schools](https://www.w3schools.com/css/css_boxmodel.asp)
The Box Model

The **CSS box model** is a box that wraps around every HTML element. It consists of:

- **Content**, where text and images appear.
- **Padding** clears an area around the content.
- **Border** goes around the padding and content.
- **Margin** clears an area outside the border.

**Note:** the padding and margin boxes are transparent.

[https://www.w3schools.com/css/css_boxmodel.asp](https://www.w3schools.com/css/css_boxmodel.asp)

```css
div{
  width: 300px;
  padding: 50px;
  border: 15px;
  margin: 20px;
}
```
FlexBox & Grid Modules
The FlexBox (Flexible Box) Module

The **Flexbox** Layout module aims at providing a more efficient way to lay out, align and distribute space among **items** in a **container**.

**Flexbox** is a whole module that includes a whole set of properties. Some related to the **container** others to the **items**.

To start using the Flexbox model, you need to first define a flex container.

The **FlexBox** in the diagram represents a **flex container** (the blue area) with three flex items.

Simple and well described tutorial: [https://css-tricks.com/snippets/css/a-guide-to-flexbox/](https://css-tricks.com/snippets/css/a-guide-to-flexbox/)
The FlexBox (Flexible Box) Module

**Container’s properties**
- display
- flex-direction
- flex-wrap
- justify-content
- align-items

**Items’ properties**
- flex-grow
- flex-shrink
- align-self
The FlexBox (Flexible Box) Module

Container’s properties
- display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items’ properties
- flex-grow
- flex-shrink
- align-self
The FlexBox (Flexible Box) Module

**Container’s properties**
- display
- flex-direction
- flex-wrap
- justify-content
- align-items

**Items’ properties**
- flex-grow
- flex-shrink
- align-self
The FlexBox (Flexible Box) Module

**Container’s properties**
- display
- flex-direction
- flex-wrap
- justify-content
- align-items

**Items’ properties**
- flex-grow
- flex-shrink
- align-self
The FlexBox (Flexible Box) Module

**Container’s properties**
- display
- flex-direction
- flex-wrap
- justify-content
- align-items

**Items’ properties**
- flex-grow
- flex-shrink
- align-self
The FlexBox (Flexible Box) Module

Container’s properties
- display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items’ properties
- flex-grow
- flex-shrink
- align-self
The FlexBox (Flexible Box) Module

Container’s properties
- display
- flex-direction
- flex-wrap
- justify-content
- align-items

Items’ properties
- flex-grow
- flex-shrink
- align-self
The FlexBox (Flexible Box) Module

Container’s properties
  display
  flex-direction
  flex-wrap
  justify-content
  align-items

Items’ properties
  flex-grow
  flex-shrink
  align-self
A nice game to master FlexBox: https://flexboxfroggy.com/
The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

Simple and well described tutorial:  https://css-tricks.com/snippets/css/complete-guide-grid/
The Grid Module

**Grid Container’s properties**
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
- align-content

**Grid Items’ properties**
- grid-column
- justify-self
- grid-row
- align-self
- grid-area
The Grid Module

Grid Container’s properties
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
- align-content

Grid Items’ properties
- grid-column
- justify-self
- grid-row
- align-self
- grid-area
The Grid Module

Grid Container’s properties
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
- align-content

Grid Items’ properties
- grid-column
- justify-self
- grid-row
- align-self
- grid-area
The Grid Module

Grid Container’s properties
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
- align-content

Grid Items’ properties
- grid-column: justify-self
- grid-row: align-self
- grid-area
The Grid Module

**Grid Container’s properties**
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
- align-content

**Grid Items’ properties**
- grid-column
- justify-self
- grid-row
- align-self
- grid-area
The Grid Module

Grid Container’s properties
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
  - align-content

Grid Items’ properties
- grid-column
  - justify-self
- grid-row
  - align-self
- grid-area
The Grid Module

Grid Container’s properties

display
grid-template-columns
grid-template-rows
gap
justify-content
align-content

Grid Items’ properties

grid-column
grid-row
grid-area
The Grid Module

Grid Container’s properties
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
- align-content

Grid Items’ properties
- grid-column
- justify-self
- grid-row
- align-self
- grid-area
The Grid Module

Grid Container’s properties

display
grid-template-columns
grid-template-rows
gap
justify-content
align-content

Grid Items’ properties

grid-column
grid-row
grid-area
justfy-self
align-self
The Grid Module

Grid Container’s properties
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
- align-content

Grid Items’ properties
- grid-column
- justify-self
- grid-row
- align-self
- grid-area
The Grid Module

Grid Container’s properties
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
- align-content

Grid Items’ properties
- grid-column
- justify-self
- grid-row
- align-self
- grid-area
The Grid Module

Grid Container’s properties
- display
- grid-template-columns
- grid-template-rows
- gap
- justify-content
- align-content

Grid Items’ properties
- grid-column
- justify-self
- grid-row
- align-self
- grid-area
A nice game to master Grid Module: [https://cssgridgarden.com/](https://cssgridgarden.com/)
Positioning

The position property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).
Static positioning is the default that every element gets — it just means "put the element into its normal position in the document layout flow".

Statically positioned elements are not affected by the top, bottom, left, and right properties.
Relative positioning is putting the element relatively to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position.

Other content will not be adjusted to fit into any gap left by the element.

**HTML**

```html
<p class="relative"> This is a relative paragraph </p>
```

**CSS**

```
.relative { position: relative;
            left: 30px;
            top: 30px;
        }
```

**Browser**

This is a static paragraph

This is a relative paragraph
An absolutely positioned element no longer exists in the normal document layout flow. Instead, it sits on its own layer separate from everything else.

An absolutely positioned element is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport).

If an absolute positioned element does not has a positioned ancestors, it uses the document body.

```
<absolute>
  This is an absolute paragraph
</absolute>
```

CSS
```
.absolute{
  position: absolute;
  left: 0px;
  top: 40px;
}
```
Whereas **absolute positioning** fixes an element in place relative to the `<html>` element or its nearest positioned ancestor, **fixed positioning** fixes an element in place **relative** to the browser viewport itself.

The **top**, **right**, **bottom**, and **left** properties are used to position the element.

**HTML**

```html
<p class="fixed"> This is a fixed paragraph </p>
```

**CSS**

```css
.fixed { position: fixed;
  bottom: 0px;
  right: 40px;
}
```
Sticky

An element with position: sticky; is positioned based on the user's scroll position.

A sticky element toggles between relative and fixed, depending on the scroll position.

It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position: fixed).

```
<p class="static">This is a sticky paragraph</p>
```

Browser

This is a static paragraph
This is a absolute paragraph
This is a relative paragraph
This is a sticky paragraph
This is a fixed paragraph

CSS

```css
.static{
  position: sticky;
  top:0;
}
```
The `z-index` Property

`z-index` values affect where positioned elements sit on the z-axis;

- Positive values move them higher up the stack.
- Negative values move them lower down the stack.
- By default, all elements have a `z-index` of auto, which is 0.

**HTML**

```html
<p class="relative">This is a relative paragraph</p>
<p class="sticky">This is a sticky paragraph</p>
```

**CSS**

```css
.relative { position: relative; 
    left: 30px; 
    top: 30px; 
    z-index: 1; 
    background-color: #5721ad; }
.sticky { 
    position: sticky; 
    top: 0px; 
    background-color: green; }
```
CSS Values and Units
Absolute length units

The **absolute length units** are fixed and a length expressed in any of these will appear as exactly that size.

**Absolute length** units (except px) are **not recommended for use** on screen, because screen sizes vary so much.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cm</td>
<td>Centimeters</td>
</tr>
<tr>
<td>mm</td>
<td>Millimeters</td>
</tr>
<tr>
<td>Q</td>
<td>Quarter-millimeters</td>
</tr>
<tr>
<td>in</td>
<td>Inches</td>
</tr>
<tr>
<td>px</td>
<td>Pixels</td>
</tr>
</tbody>
</table>

Pixels (px) are relative to the viewing device.
Relative length units specify a length relative to another length property.

Relative length units scale better between different rendering mediums.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Relative to</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Relative to the parent element</td>
</tr>
<tr>
<td>em</td>
<td>Relative to the font-size of the element</td>
</tr>
<tr>
<td></td>
<td>(2em means 2 times the current font size).</td>
</tr>
<tr>
<td>rem</td>
<td>Relative to font-size of the root element.</td>
</tr>
<tr>
<td>vw</td>
<td>Relative to 1% of the width of the viewport</td>
</tr>
<tr>
<td>vh</td>
<td>Relative to 1% of the height of the viewport</td>
</tr>
<tr>
<td>vmin</td>
<td>Relative to 1% of viewport's smaller dimension</td>
</tr>
<tr>
<td>vmax</td>
<td>Relative to 1% of viewport's larger dimension</td>
</tr>
</tbody>
</table>
**@rules (at-rules)**

An **at-rule** is a **CSS statement** that instructs **CSS** how to behave.

They begin with an at sign, '@', followed by an **identifier** and includes everything up to the next semicolon, ';' or the next CSS block, whichever comes first.

@rule syntax

```
@identifier (rule);
```
@rules (at-rules)

The @import CSS at-rule is used to import style rules from other style sheets.

These rules must precede all other types of rules, except @charset rules;

@rule syntax

@identifier (rule);

@import

@import url("fineprint.css") print;
@import url('landscape.css') screen
and (orientation: landscape);
The **media** CSS **at-rule** can be used to apply part of a style sheet based on the result of one or more media queries.

With it, you specify a **media query** and a block of CSS to **apply** to the document **if and only if** the media query **matches** the device on which the content is being used.

```css
@media (min-width: 800px)
.header {
  flex-direction: column;
  align-items: center;
}
```

```css
@media (min-width: 1000px)
.header {
  flex-direction: row;
  justify-content: space-between;
}
```
Pros and Cons of using CSS

Pros

• Greater control of the appearance of web pages.
• Easier management of style changes large websites.

Cons

• Different browsers may interpret CSS Styles in different ways.
• Some browsers may not show some styles at all.
CSS

Try it yourself …
You can open the **Developer Tools** by right clicking, then, inspect.

In the **Element tab**, click on any element you want, you can see the applied styles in the **Styles tab**.

The developer tool enables you to make style changes and see them immediately.
Extra reading/exercises

- W3Schools is a free educational website for learning to code online. With their "Try it Yourself" editor, you can edit the CSS code and view the result immediately: https://www.w3schools.com/css/
Thank You for your attention

Mohamad Gharib
mohamad.gharib@ut.ee

unitartu
tartuuniversity