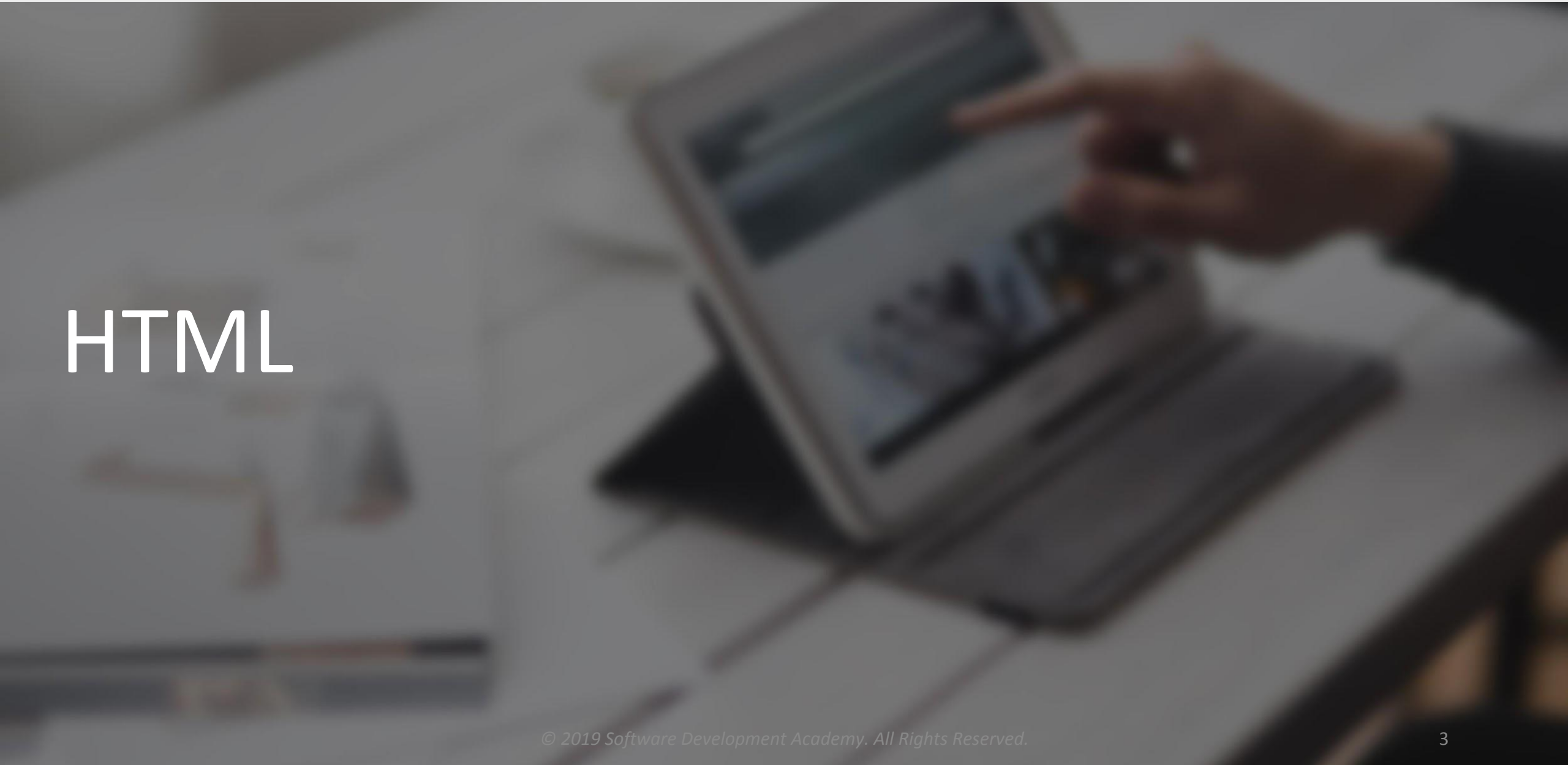

HTML / CSS / JS

Agenda

1. Html basics
2. Html5
3. Html Forms
4. CSS basics
5. CSS layout
6. CSS pseudo-classes
7. CSS3
8. JS
9. ES6 JS
10. TypeScript



HTML

What is HTML?

- It means HyperText Markup Language.
- In other words - the node-based text message that is commonly sent over using HTTP.
- HTML is a language that tells the browsers (like Firefox, Chrome, IE etc.) what exactly to show to the user.
- HTML is basic building blocks for every website!

Tags and attributes

- HTML consists of tags written in a fairly simple format:

```
<tag>content…</tag>
```

- Some tags do not have content inside of them, so they are self-closing:

```
<tag />
```

- Some tags also have attributes to pass the data to the element:

```
<tag attribute-name="attribute-value">content…</tag>
```

Basic HTML example

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Page Title</title>
  </head>
  <body>
    <h1>This is a Heading</h1>
    <p>This is a paragraph</p>
    <ul>
      <li>And I am a list item!</li>
      <li>And I am another!</li>
    </ul>
  </body>
</html>
```

- Note the format of the text - every node is either closed by a **/>** closure, or closed by similar node with a **</** prefix.
- The red items here are called **tags**. They are predefined keys that browser understands as building blocks!
- The 4 main blocks here would be **<!doctype>**, **<html>**, **<head>** and **<body>**. We shall learn more about them in a short while!

Doctype

- Doctype is an optional first element of the HTML document.
- It defines the version of the HTML used further.
- Modern HTML5 document type declaration:

```
<!DOCTYPE html>
```

<html>

- It is the main containing tag of the HTML code.
- Every other node goes **inside** the **<html>** node!
- Modern browsers can often parse the code even without it. It is however advised to use it for readability of the code - as later on, it can get a bit messy. 😊

<head>

- Contains mostly the **meta** information of the document - like the title, tags or links for resources.
- **<title>**…**</title>** tag contains the title of the document displayed in the browser.
- **<link rel="" type="" href="" />** tag defines the link to some external resource - often CSS. We shall cover more on this topic once we reach CSS. 😊

<body>

- Contains visual information of the page - the content that user sees!
- Blocks within body include headings, paragraphs, lists, tables and forms.

Practice time!

1. Create a new file with a **.html** extension. You can choose any name - although common practice is naming it index.html.
2. Open the file using any text editor. Even Notepad will do, but it will be easier if you use something more developer-friendly - like Notepad++ or Sublime Text.
3. Write up your first HTML code! Create a simple HTML structure using **<html>**, **<head>**, **<title>**, and **<body>** tags.
4. Within the **<body>** tag, add the following tag: **<h1>Hello World!</h1>**.
5. Open your file using any web browser - and see what happens! 😊

Headings and paragraphs

- **<h1>, <h2>, …, <h6>** tags contain headings for the content - and vary in size, h1 being the biggest and h6 being smallest.
- **<p>** is a paragraph tag - for common text values.
- **<pre>** means a pre-formatted text - this tag will respect the line endings within the given text.
- **
** tag adds a break on the page.

Text formatting

- **<b>** - Bold text.
- **** - Important text.
- **<i>** - Italic text.
- **** - Emphasized text.
- **<mark>** - Marked text.
- **<small>** - Small text.
- **<sub>** - Subscript text.
- **<sup>** - Superscript text.

Practice time!

1. Add a little text to your HTML document. For instance, you can take this sample news article - https://waylink-english.co.uk/simply-news/teenager-round-world-voyage.
2. Split the text into headings and paragraphs using **<h..>** and **<p>** tags. If you use the example text above, you can try adding the name of the article as a h1 and the first line of each paragraph as <h3>. Don't worry too much about how it looks for now. 😊
3. Add some formatting to the text!

Images

- **<img>** tag is a self-closing tag that allows us to add images to our web page. It can hold several attributes:
  - src - a relative link to the image that should be displayed,
  - alt - a description of the image,
  - width,
  - height.

```
<img src="img.jpg" alt="Picture" width="500" height="600">
```

Hyperlinks

- **<a>** is a container tag that allows us to link and access different web pages - either on our own website, or any external resource.
- It generally holds 2 attributes:
  - href - relative link to the page where we want to direct the user on click,
  - target - optional attribute, use “blank” for opening a link in a new tab.

Practice time!

1. Add an image to your webpage using **<img>** tag. Try adjusting its height and width using attributes!
2. Add a link to some other page - either a new html file, or perhaps some external resource!

Lists

- **<ul>** tag defines an **Unordered List. ** tags inside that container will be rendered as list items.
- **<ol>** tag defines an **Ordered List**. It uses **<li>** tags as well, but their display will change! This time, it will be numbered.
- Both <ul> and <ol> tags can be nested within each other!

Practice time!

1. Create an unordered list, for example create a list of your TODOs for the day or your favourite websites!
2. Create an ordered list, for example take a classic poem http://www.wordsforlife.org.uk/songs/house-jack-built and do an ordered list of the paragraphs like so:

```
1. This is the house that Jack built.
2. This is the malt,
   That lay in the house that Jack built.
3. This is the rat,
   That ate the malt,
   That lay in the house that Jack built.
```

HTML Tables

- Data tables are defined by **<table>** tag.
- Table has a **<thead>** and **<tbody>** container tabs to separate the data and the headers.
- **<thead>** contains **<tr>** and **<th>** tags.
- **<tbody>** contains **<tr>** and **<td>** tags.
- **<tr>** is a table row containing multiple cells.
- **<th>** is a table header.
- **<td>** is a table cell.

Basic Table example

```
<!DOCTYPE html>
<html>
<body>
  <table>
    <thead>
      <tr>
        <th>First name</th>
        <th>Last name</th>
        <th>Age</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>John</td>
        <td>Wick</td>
        <td>33</td>
      </tr>
      <tr>
        <td>Eve</td>
        <td>Jackson</td>
        <td>27</td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

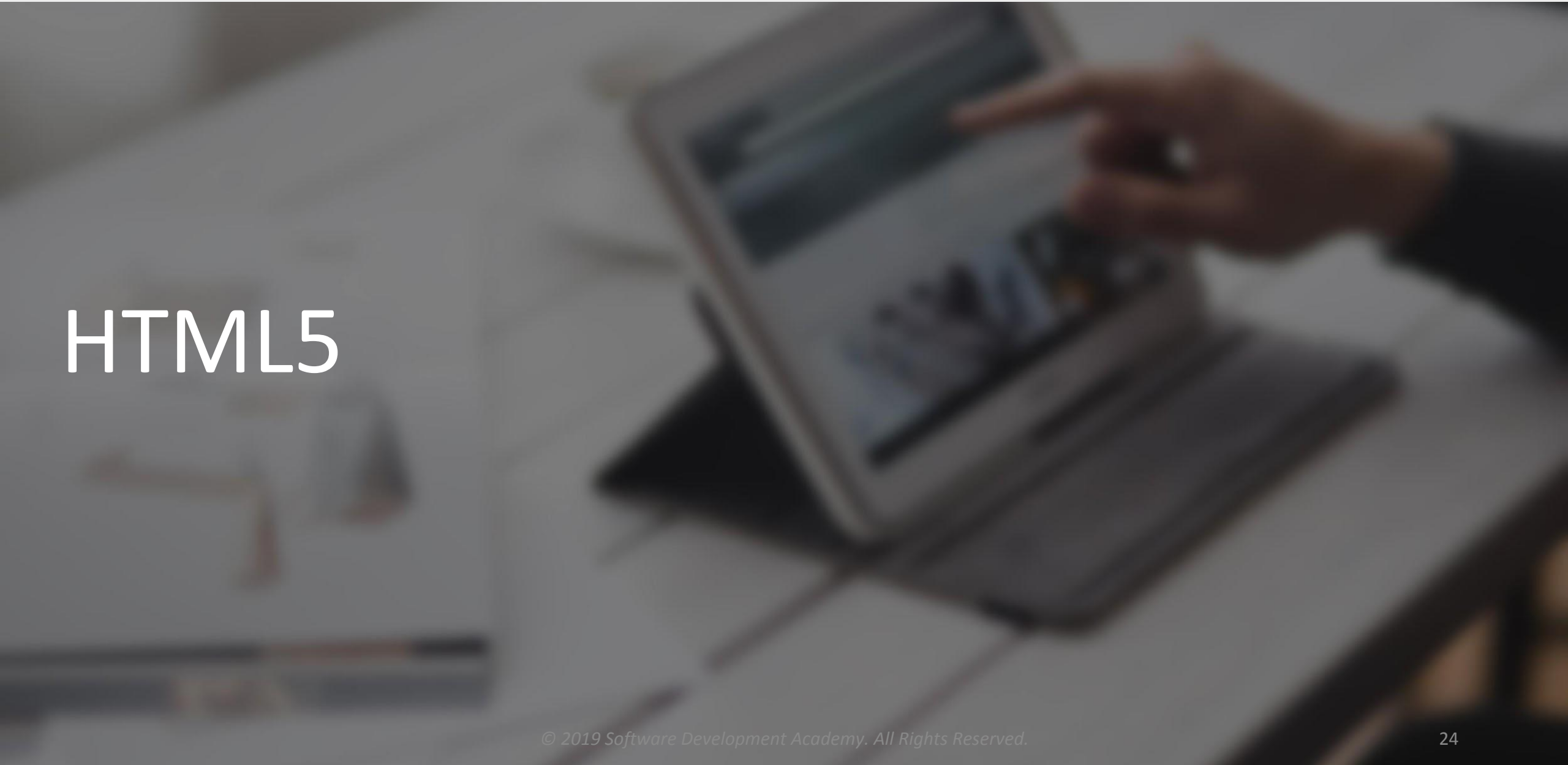
- How many columns does this table have?
- How many data rows does this table have?

Practice time!

1. Create a data table based on a well-known riddle - https://udel.edu/~os/riddle.html. Suggested data columns are: **Nationality**, **House Color**, **Beverage**, **Cigar Brand**, **Pet**.
2. Try to solve the riddle! Of course, this is not mandatory. 😊

Container tags

- HTML generally has two types of displayed data - blocks and inline elements. Blocks take up 100% of the page width, while inline elements take up only the space that is required for them.
- ** ** tag is the most commonly used block-type tag.
- **** tag is the most commonly used inline tag.



HTML5

HTML versions

- Since 2011, current version of HTML is 5.0. However, most of the tags that we reviewed previously, were present ever since version 4!
- HTML5 mostly brought semantic elements - tags that can specify exactly what type of content you will write in the text.

HTML5 - important tags

- **<article>** - defines an article in a document.
- **<header>** - defines a header for a document or section.
- **<footer>** - defines a footer for document or section.
- **<section>** - a section in document.
- **<nav>** - navigation links.
- **<canvas>** - drawable (usually via JS) graphics.
- **<svg>** - rendering svg elements.
- **<audio>**
- **<video>**

Practice time!

1. Use some of the new tags on your HTML page. Try splitting the content into readable sections.

HTML forms

<form>

- Web pages often contain forms.
- HTML defines multiple tags for form elements.
- Whole form is wrapped in **<form>** tag.
- **<form>** accepts attributes like **method** and **action -** method specifies the HTTP Request Method (GET / POST), and action specifies the URL of the application that accepts the data.

<input>

- Input fields are crucial parts of web forms.
- Input **type** is defined as an attribute, and can take a lot of values.
- Examples - *text, checkbox, button, color, date, email, password, number, radio, submit*.
- Inputs can also have a **required** attribute, that would mean that you cannot submit a form without filling the input first.

<label>

- Labels are a short description for each input - that is why each of them has a **for** attribute. Its value should be equal to the **id** attribute of the input it is mapped to!
- Clicking on a mapped label would also trigger a click on the relevant input.

Basic form example

```
<form action="/action_page.php">
  <div class="container">
    <h1>Register</h1>
    <p>Please fill in this form to create an account.</p>
    <hr>

    <label for="email"><b>Email</b></label>
    <input type="text" placeholder="Enter email" id="email" required>

    <label for="pwd"><b>Password</b></label>
    <input type="password" placeholder="Enter password" id="pwd" required>

    <label for="pwd-repeat"><b>Repeat password</b></label>
    <input type="password" placeholder="Repeat password" id="pwd-repeat"
required>

    <button type="submit" class="register-button">Register</button>
  </div>
</form>
```

<select>

- **<select>** tag renders a dropdown field with options mapped by **<option>** tag. Note that actual value sent to the server is put in “value” attribute of each option:

```
<select>
    <option value="mercedes">Mercedes</option>
    <option value="bmw">BMW</option>
    <option value="porsche">Porsche</option>
</select>
```

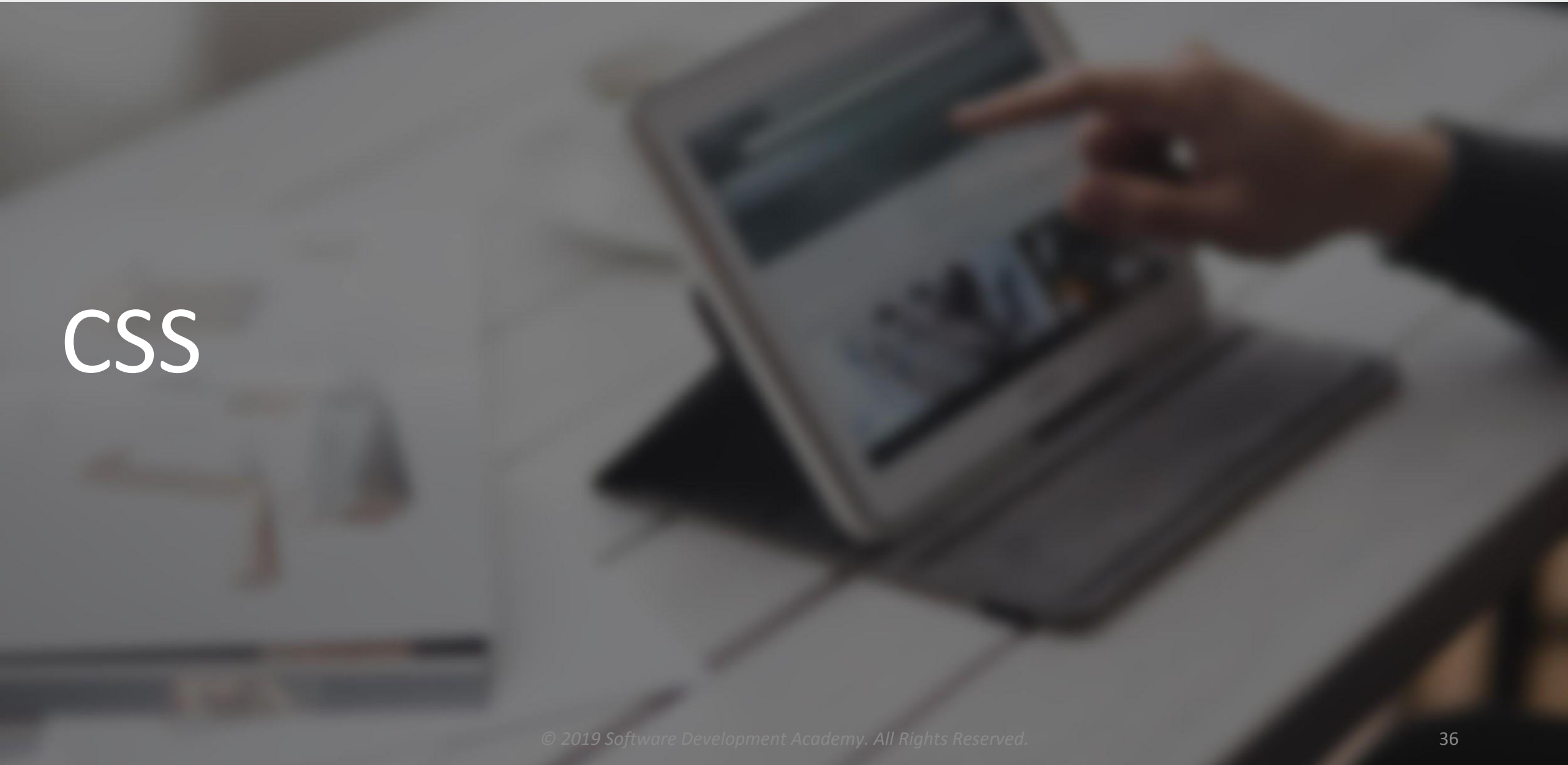
<textarea>

- **<textarea>** tag renders a large text input field with possible line breaks and a changeable input size with predefined **rows** and **cols** attributes:

```
<textarea rows="4" cols="50">
This is a sample text designed to show how the text
would all fit in the large textbox.
</textarea>
```

Practice time!

1. Create a new file called register.html.
2. Add a simple user registration form using **<form>** and **<input>** tags.
3. Try using a variety of inputs - for instance, try using **<input type="file">** for a file selector!



CSS

What is CSS?

- CSS means Cascading Style Sheets.
- It is a language used to define the styles of the HTML document.
- That means that each separate HTML tag can be shown in its own way. We can change color, size, alignment, margins and almost everything. 😊

How to use CSS?

- CSS can be applied in three possible ways:
  - referring to an external **.css** document using **<link>** tag in HTML,
  - written in the **<style>** tag in the **<head>** section of the document,
  - written in the **style** attribute of any HTML tag.
- Best practice is to use the external .css file to store all styles for you.

Referring to external CSS

- A referred file can be present locally or even be an online resource.

```
<head>
  …
  <link rel="stylesheet" type="text/css" href="style.css">
  …
</head>
```

CSS syntax

- CSS follows a straightforward pattern - you define an element to be styled, next step is to add some style attributes to it:

```
h1 {
    color: white;
    text-align: center;
}
```

CSS selectors

- Elements can be selected in 3 possible ways:
  - by **tag name** (*h1, p, body* …),
  - by **class attribute**,
  - by **id attribute**.
- Styles connected to the **id attribute** have the highest priority, followed by **class attribute** and ending with **tag name** being the lowest priority. It is advised to use **class attribute** for styling.

CSS selectors

```
<html>
<head>
<style>
  h1 {
    color: blue;
  }
  .center {
    text-align: center;
    color: red;
  }
  #first {
    color: green;
  }
</style>
</head>
<body>
  <h1 class="center" id="first">Heading!</h1>
  <p class="center">Paragraph.</p>
</body>
</html>
```

- What color does the heading have?
- What color does the paragraph have?

Practice time!

1. Create a new file called **style.css**.
2. Add a link in your HTML files to **style.css** using **<link>** tab.
3. Add some colors to your document! Use the **tag**, **class** and **id** selectors to change the **color** of your elements.
4. If that's too easy, try also changing **text-align**, **background-color** and **font-family** of your HTML elements. 😊

Descendant selectors

- As the name hints, CSS can be scoped using a combination of selectors.
- For example, a selector **#my-element div** will target all ** ** tags that are children for **#my-element** ancestor.

Descendant selectors

```
<ul>
  <li>Item 1</li>
  <li>
    <ol>
      <li>Sub-item 2A</li>
      <li>Sub-item 2B</li>
    </ol>
  </li>
</ul>
```

- What CSS selector would target all **<li>** elements under the **<ul>** tag?

Child selectors

- The **>** symbol allows us to target elements that are direct children of the parent element.

```
#my-element>div {
…
}
```

Next element selectors

- The **+** symbol allows us to target the element that is immediately after the parent element. Note that it might not be the best practice to use it often! 😊

```
label+p {
…
}
```

Combined selector

- The , symbol allows us to target multiple elements at once.

```
selector1, selector2 {
…
}
```

HTML attribute selector

- Elements can also be targeted by their custom HTML attributes - any attribute will do.

```
[name=abc] {
…
}
```

CSS properties

- We have already seen some properties (like color or font-family) in the example before, so let's take a look at what other properties we could change!
- **Color** and **background-color** accept literal values (eg. *green, red, magenta, …*) or HEX RGB values (eg. *#000, #0FF9BB, …).*
- **Width** and **height** accept values like *10px, 100%* or a bit more complex (eg. *2em, 10rem, …).*

Font styling

- Attribute **font-family** accepts font values like *"Times New Roman"* or *"Calibri"* and sets the font style of the text elements.
- **Font-size** accepts values like *22px* and sets text size.
- **Font-weight** accepts *bold, normal, lighter* and *bolder* values or numbers from 100 to 900.
- **Text-decoration** accepts values like *underline* and *strikethrough* and sets the decorations of the text.

CSS borders

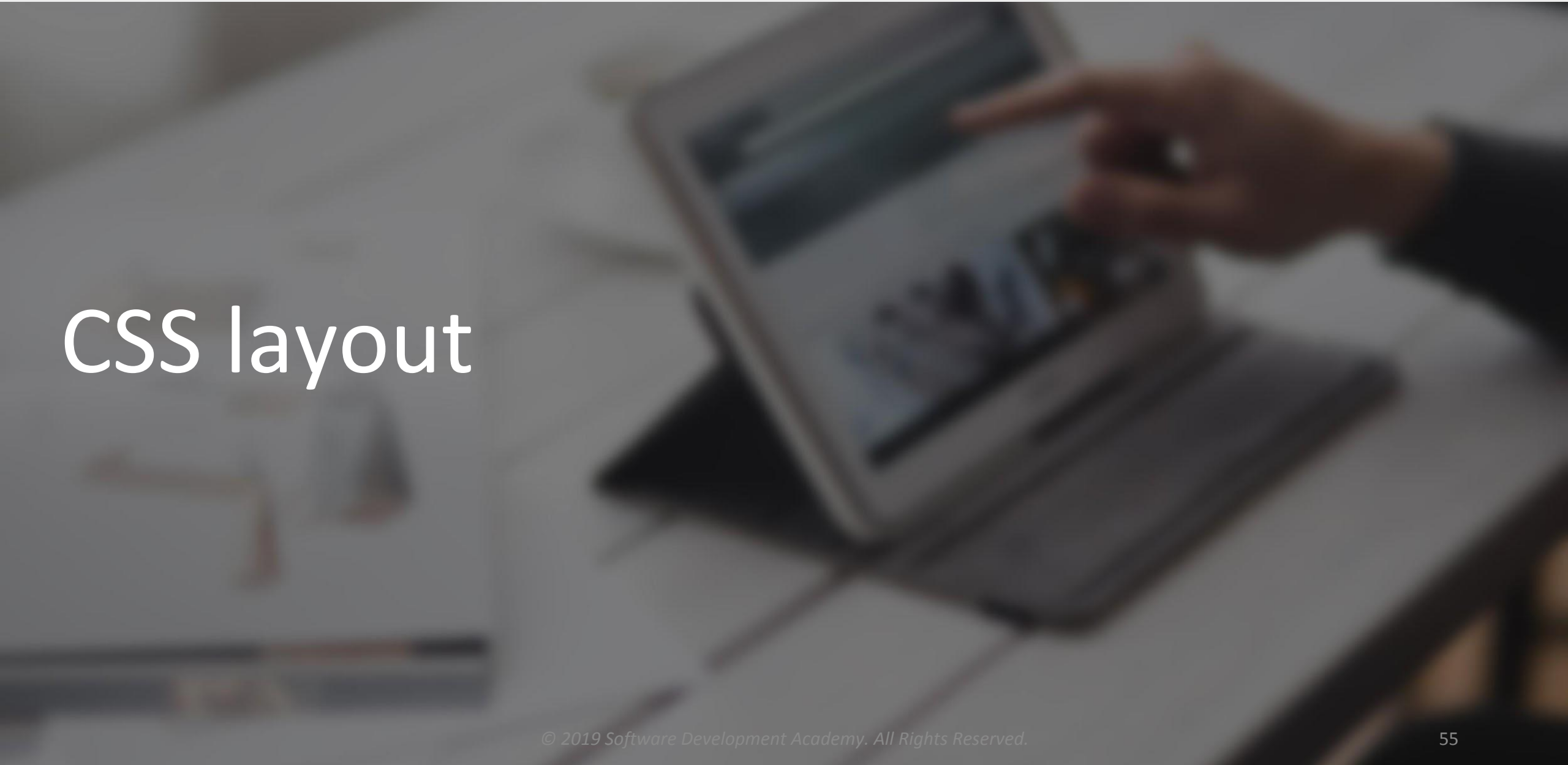
- **Border** attribute accepts several parameters:
  - thickness of the border (eg. *1px*),
  - style of the border (eg. *solid*),
  - color of the border (eg. *#000*).
- Borders for different sides can be adjusted. For example, you can use **border-right** or **border-top** property to style just one side of the element!

Element alignment

- **Text-align** property accepts *left, right, center* and *justify* values and aligns the text horizontally within the container. Note that it does not necessarily mean that the text will be in the center of the page! 😊
- **Padding** property is tricky - it accepts 4 pixel values clockwise - padding from top, right, bottom and left (eg. *1px 10px 2px 10px)*. Padding for the element means adjusting its own inner padding.
- **Margin** property works similarly to **padding -** but affects the outer padding.

Practice time!

1. Let’s add some styling to your HTML document! Change the background color of the whole page, then choose a new font for the **<h1>** elements, and add an underline for the **<h2>** elements.
2. Add some positioning, use **text-align** to position the sections of text on your page.
3. Add a border for the paragraphs on the page and make them easy to notice.
4. Use **margin** and **padding** to increase the distance between the elements on your page.



CSS layout

Positioning elements

- CSS can be a bit tricky with where exactly the elements of the page are placed. You might have noticed it already. 😊

Positioning elements

- All elements by default have a property named **position**. Its default value is *static*, meaning the elements are just rendered straightforward on the nearest available space.
- Obviously, we can change that. **Position** attribute accepts several other values: *absolute*, *fixed*, *sticky* and *relative*.
- We can also change their position dependent on their normal placement using **top, left, right** and **bottom** properties.

Positioning elements

- An element with **position: relative;** is positioned relative to its normal position.
- An element with **position: fixed;** is positioned relative to the viewport, what means it always stays in the same place even if the page is scrolled.
- An element with **position: absolute;** is positioned relative to the nearest positioned ancestor.
- An element with **position: sticky;** is positioned based on the user's scroll position.

Practice time!

1. Let’s try some of the positioning tags. Add a **<header>** element to your website, make it visible and make it *fixed* throughout all the page so it is visible even when we scroll down!
2. Let’s experiment a bit. We want a small (20x20) icon in the top right corner, but below the header. Let’s try an *absolute* position for it.
3. Make a small container element, and position some text in the bottom right corner of it. Do not forget that the parent element must be at least *relative* to the *absolute* element!

Displaying elements

- All elements by default have a **display** property but it varies on the type of the element.
- Text elements (eg. **span**) usually have **display: inline**, and block elements (eg. **div**) have **display: block**.
- *Inline* means that the element will take up only the space it needs for its content.
- *Block* means that the element will take up all the width that is available to it. It starts on the new line.

Displaying elements

- There are some other **display** property values for displaying elements - like *inline-block, table* or lately *flex* and *grid*.
- Elements can also be positioned within the display using the old-fashioned **float** property that accepts values like *left* or *right* and makes the element stick to the left or right side of the container element overlapping the non-floating elements.

Practice time!

1. Let’s try playing with the **display** property. Create a class named .container, render it as a *block* element, and use it as a parent for different content sections in the HTML file. Add a padding and margins for it as well!
2. Let’s add a navigation section to our website. Use the *inline-block* **display** value to render it as a list of buttons!

CSS pseudo-classes

What are pseudo-classes?

- Sometimes you want to add an effect to an element - a small interaction with the user or a visual effect.
- Pseudo-classes allow us to do just that - in a limited way.

Pseudo-class syntax

- Pseudo-classes have a fairly straightforward syntax:

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

Possible pseudo-classes

- **:hover** defines a style that should be applied to the element once it is hovered by cursor.
- **:first-child** defines a style for the first element in a list of elements.
- **:nth-child(2)** does that for any of the child elements. 😊
- **:link** is a style for unvisited links.
- **:visited** is a style for visited links.
- **:active** and **:focus** are styles for elements that are currently focused on by the user.

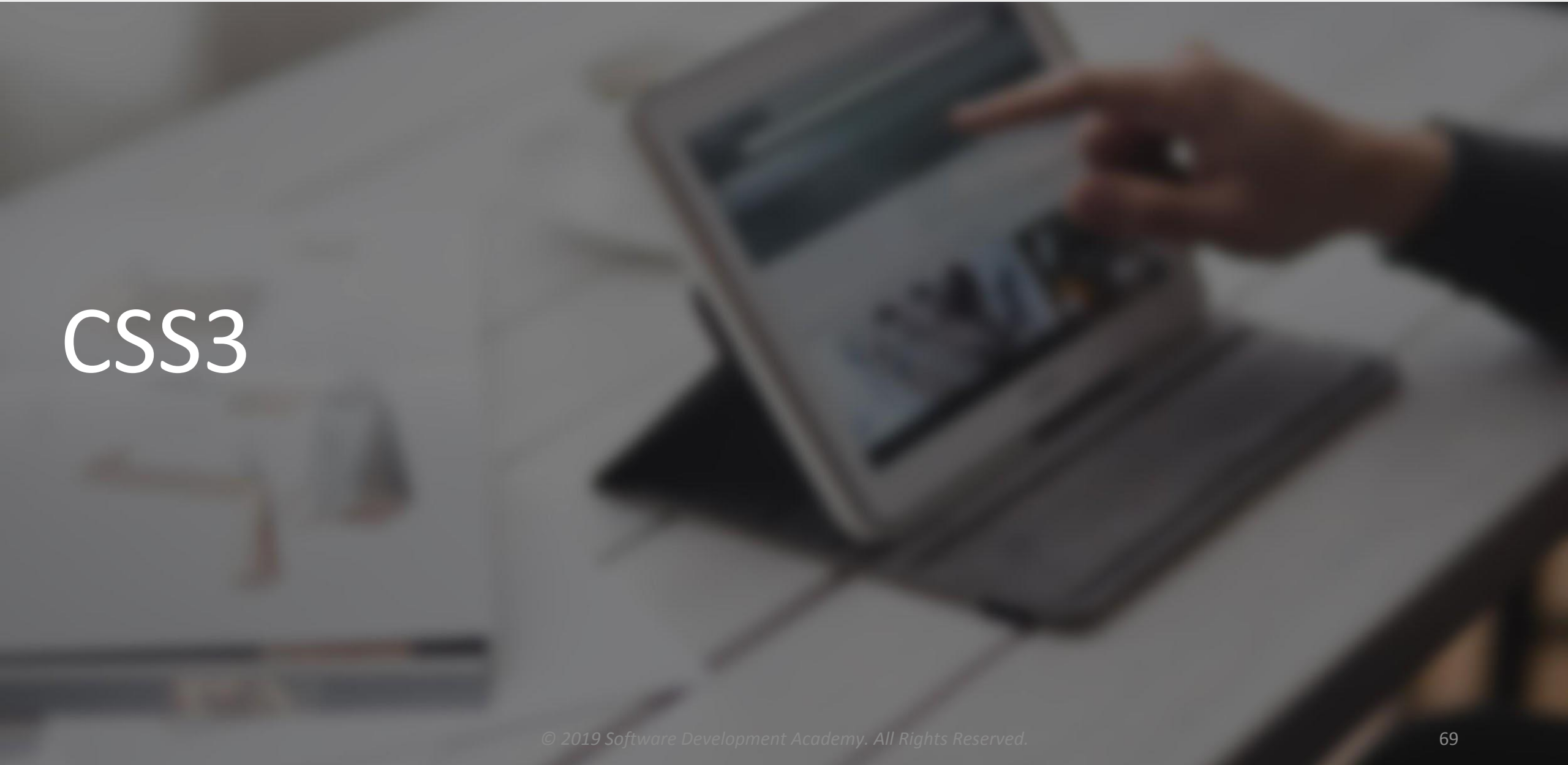
Usage example

```
<html>
<head>
<style>
  p {
    display: none;
    background-color: yellow;
    padding: 20px;
  }

  div:hover p {
    display: block;
  }
</style>
</head>
<body>
  <div>Hover over me to show the p element!
    <p>Tada! Here I am!</p>
  </div>
</body>
</html>
```

- The **<p>** tag is not displayed by default.
- However, once we hover over the ** ** element, it suddenly appears!

Practice time!

1. Add a small hover effect to the text present in your HTML. For example, add a background color and an underline!
2. Create a navigation menu effect. In the header navigation, once the user hovers over one of the menu items, show advanced menu options right below!



CSS3

What is it?

- Just along with HTML5, a new standardized version of CSS was defined in 2011 in order to introduce new features to the Web.
- CSS3 brought new properties, values and selectors.
- It is currently still in development, but most of its features are already working with most browsers.

CSS3 new properties

- **Border-radius** - allows for rounded borders.
- **Box-shadow** - draws a shadow around the element.
- **Text-shadow** - draws a shadow around the text.
- **Opacity** - defines the opacity.
- **Transform** - allows to rotate the element.
- **Font-face** - allows to import custom fonts.

CSS3 layout options

- Layout properties were changed in CSS3. Instead of using **float: left**, we can now use **display: flex** or **display: grid** property.
- **Flex** content also comes with a variety of properties:
  - **justify-content:** *flex-start | flex-end | center | space-between | space-around | space-evenly | start | end,*
  - **flex-direction:** *row| column | row-reverse | column-reverse,*
  - **flex-wrap:** no*wrap | wrap | wrap-reverse,*
  - **order:** int number.

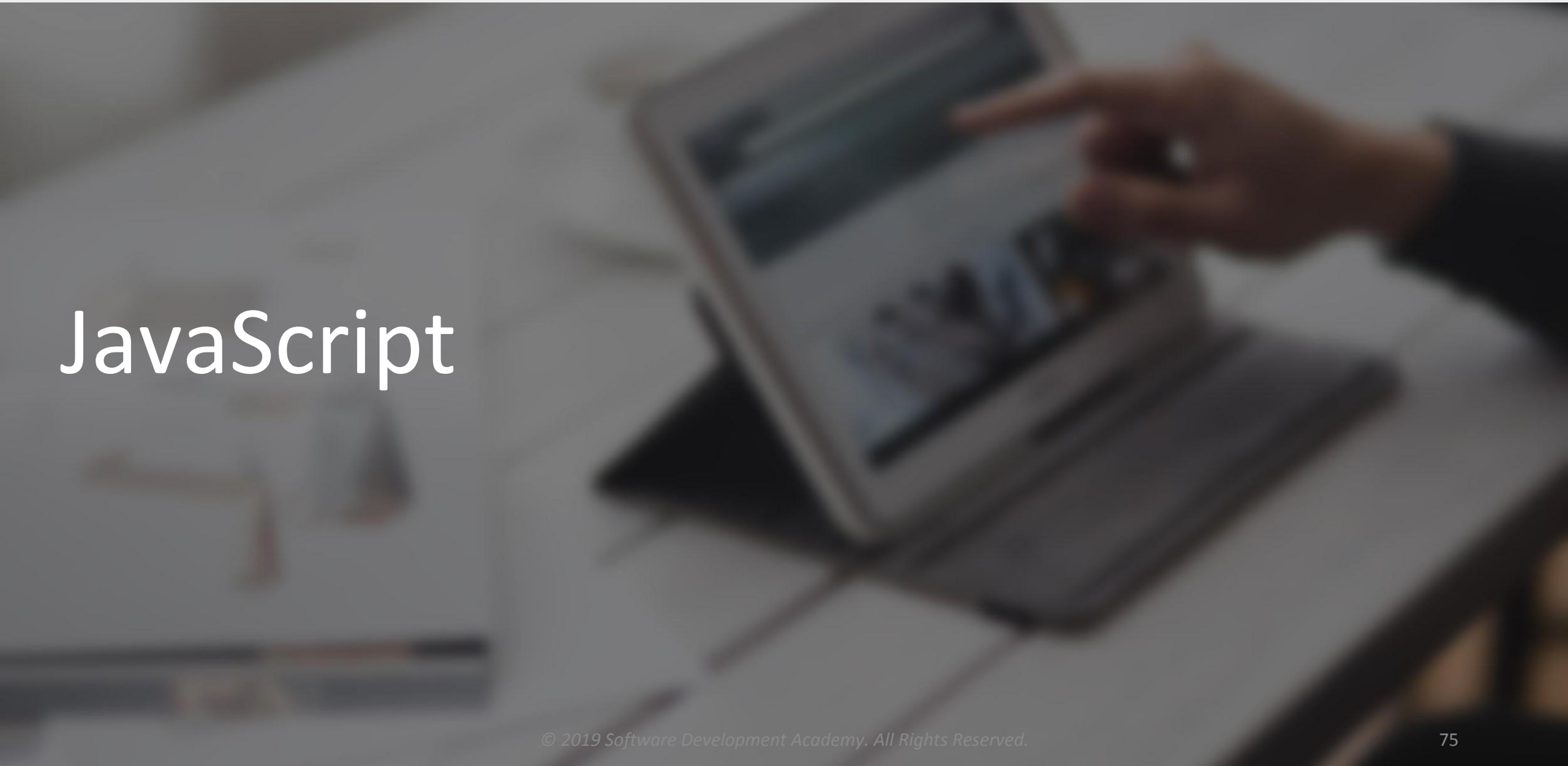
CSS3 @media tag

- **@media** rule is used to apply different styles for different screen sizes or even devices.

```
@media screen and (min-width: 480px) {
    body {
        background-color: lightgreen;
    }
}
```

Practice time!

1. This concludes the CSS course. It is time for us to create something unique using our new knowledge. Try creating a very simple news page, a personal blog or perhaps a data page for your business - the choice is yours! 😊
2. Use https://www.w3schools.com/css/css_examples.asp as a cheat sheet if you need any quick hints on how to implement something.



JavaScript

What is it?

- JavaScript is a programming language designed to add interactivity to web pages.
- Lately, it is also one of the leading languages for developing full-scale web applications.
- JavaScript is a language that can be compiled by browsers on-the-go.

Adding a script to the page

- To execute JS code, you can simply put it in **<script>** tag in your HTML.
- Just like with CSS, it is considered cleaner to create separate .js files to contain the code.
- To do that just add an attribute to script tag like so:

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

Practice time!

1. Create a new file with a .js extension - like script.js or index.js.
2. Inside, write the following code:

```
alert('Hello world!');
```

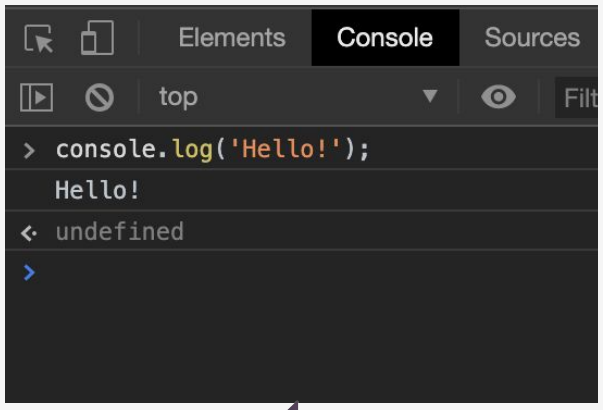
3. Include the .js file in your HTML file using the **<script>** tag.
4. Open your page in browser - a popup message will greet you!

Alerts and logs

- There are two commands of simple outputs of your JS code - **alert** and **console.log**.
- The first one creates a dialog window, as you might have seen in the previous example.
- The second one prints the output into *Console* tab in your browser!

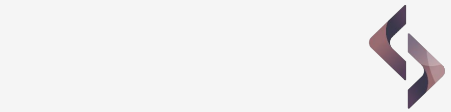
JS and Browser Console

- Most of the modern browsers come with a very useful JS console that allows to debug the code showing the logs and even allows us to execute some code ourselves! It looks like this in Chrome:



Variables in JS

- JS features a **dynamic** typing. That means that all variables can be declared without specifying their type!
- Variables are declared using keyword **var**.
- For example:
  - `var text1 = “abc”;`
  - `var text2 = ‘abc’;`
  - `var number1 = 123;`
  - `var number2 = 5.01234;`
  - `var boolean1 = true;`



Data comparison in JS

- Due to the dynamic types, it is often hard to compare the values of your variables.
- There are two operators for comparison:
  - **==** compares *values,*
  - **===** compares *values and types*.

Common operators

- Most operators work the same way as they would in Java:
  - `console.log(1 + 2); // prints 3`
  - `console.log(1 % 2); // prints 1`
  - `console.log("abc" + "def"); // prints "abcdef"`
  - `console.log(true && false); // prints false`

Practice time!

1. Create a new variable named **myName** and assign your name as a string value to it.
2. Add a console command to write **“Hello” + myName** so that the page would greet you by name!

For/while loops and conditionals

```
var text = "";
var i;
for (i = 0; i < 5; i++) {
  text += " The number is " + i + " ";
}

while (i <= 10) {
  if (i % 2 === 0) {
    text += " The number is " + i;
  }
  i++;
}

alert(text);
```

- What numbers would be shown in the alert?

Arrays

- Arrays are declared using **[]** notation eg. **var a = [1, '4', 5];**.
- Arrays can hold any types of variables at the same time.
- To refer to an element of the array above, use **a[2]**.
- To get the length of the array, use **a.length**.
- To remove the last element of the array, use **a.pop()**.
- To add an element to the array, use **a.push()**.

Objects

- JSON Objects are declared using **{}** notation - like **var a = { prop: ‘value’ };**.
- Objects can hold any keys with any values at the same time.
- To refer to an prop of the object above, use **a.prop** or **a[‘prop’]**.
- To get the keys of the object, use **Object.keys(a);**.

Functions

- Functions are declared as follows:

```
function myFunction(p1, p2) {
    return p1 * p2;
}
```

- To call the function above, we would write:

```
myFunction(3, 4);
```

Practice time!

1. Try to solve the classic Fibonacci sequence using **arrays** and **for** loops.
2. For the reference - it goes like this: 0, 1, 1, 2, 3, 5, 8 …
3. Try to fetch the 10^{th} number in the sequence using JS and output it in the console.

Accessing HTML

- Initial idea of JS was to manipulate the document - so a bunch of keywords is reserved for that.
- Keyword **document** allows us to get HTML elements and manipulate them however we want, for example:

```
var a = document.getElementById("someId");
var b = document.getElementsByClassName("someClass");
```

Accessing HTML

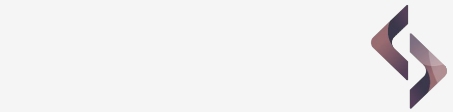
```
<html>
<body>


<h2>My first web page</h2>
<p>My first paragraph</p>

<p id="demo">My first paragraph</p>

<script>
  // Let's find an element!
  var element = document.getElementById("demo");
  // Let's set the element's text first.
  element.innerHTML = "<h2>Hello</h2>";
  // And then mess with it!
  element.style = "color: red; transform: rotate(180deg);";
</script>

</body>
</html>
```

- Note that **innerHTML** and **style** are not the only properties that we can change!
- Try using **console.log** to figure out what other properties we could affect.

Binding events

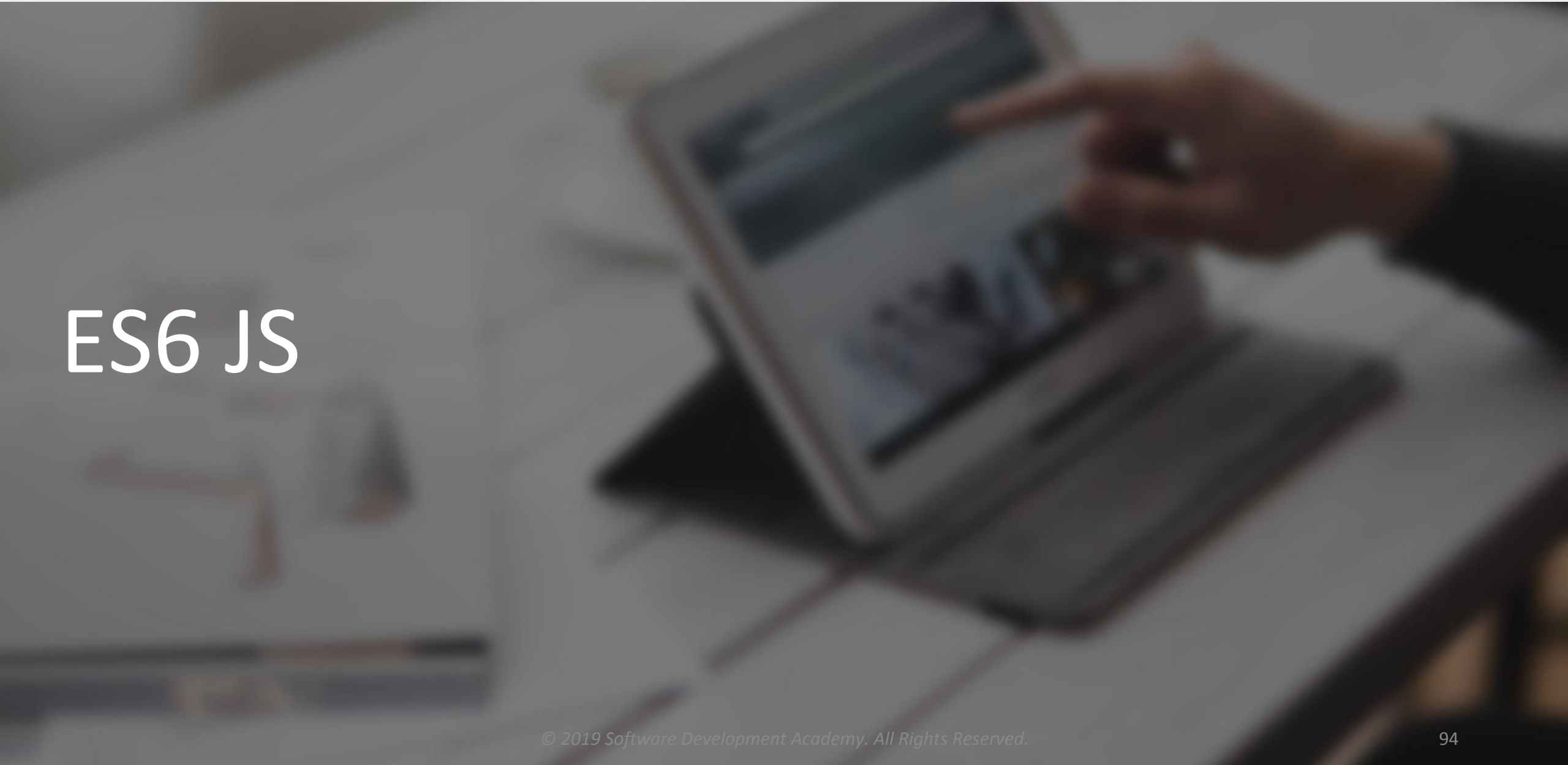
- We can also bind user interaction events to the HTML code.
- For example, we can add an **onclick** listener for every HTML element!

```
<p onclick=”clickMe()” id=”demo”>Click!</p>
<script>
    function clickMe() {
        alert(‘hello!’);
    }
</script>
```



Practice time!

1. Let’s use JS functions to add some interactivity to our page.
2. Let’s create a more advanced navigation menu. When a user clicks on the link in the menu, an expanded section is shown right underneath it.
3. Let’s create a small gallery - when a user clicks on an image in the list of images, that image becomes bigger.
4. For additional points, try creating an overlaying popup with an image on image click! 😊



ES6 JS

What is ES6 JS?

- Just like with CSS3 and HTML5, JS also encountered some changes that greatly simplify our lives.
- Changes introduced around 2015 are called ES6 (as in EcmaScript 6), or ES2015.

Variable declaration

- **Var** keyword had a few issues. Scoping, loose typing, issues with overriding, and even some security concerns.
- Instead, it is advised to now use **let** and **const**.
- **Let** defines a variable that can be overridden later on within the same scope.
- **Const** defines a variable that will not be changed later.

Arrow functions

- Functions can now be declared using a much shorter syntax.

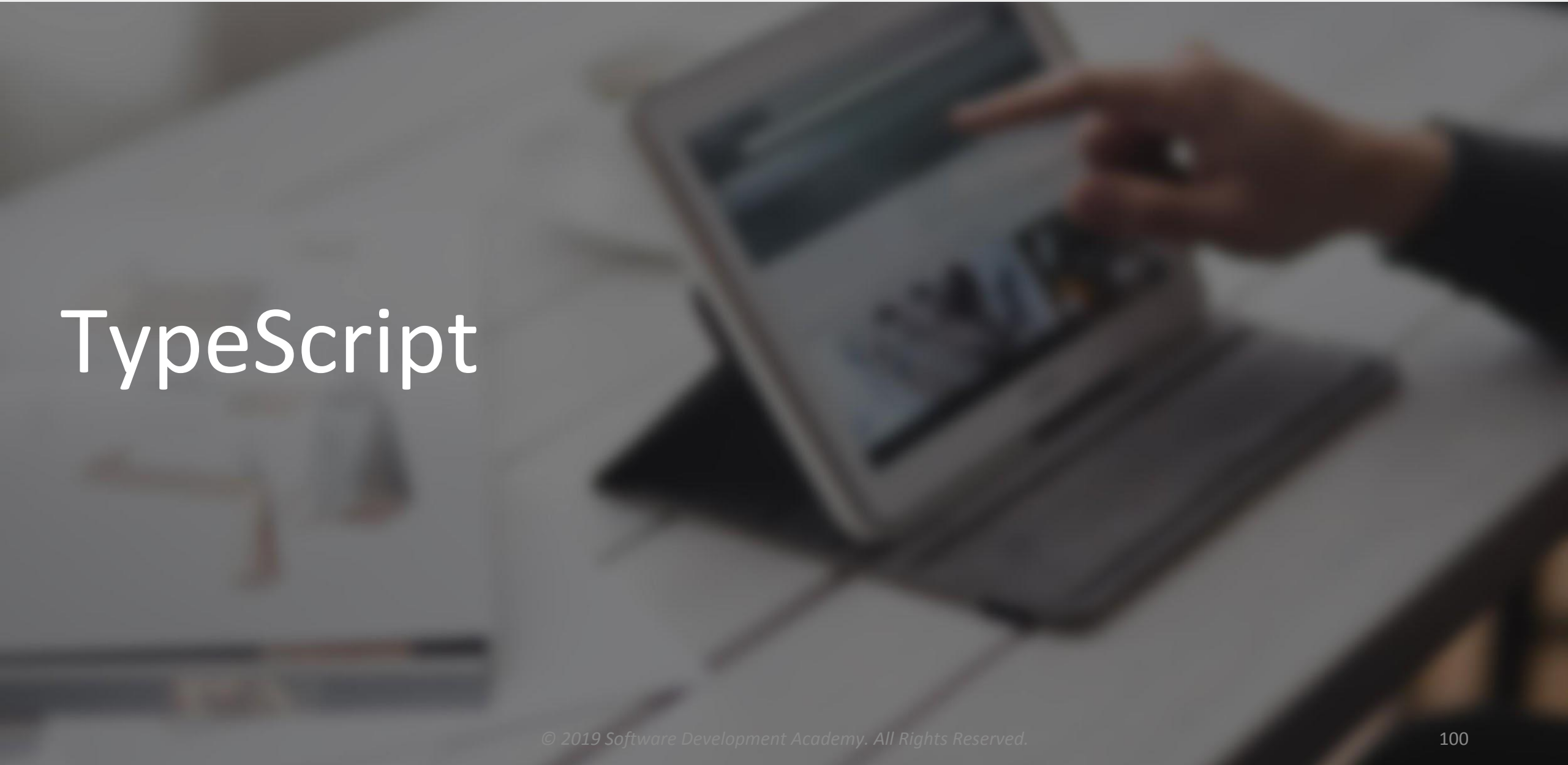
```
var multiplyES5 = function(x, y) { // ES5
    return x * y;
};

const multiplyES6 = (x, y) ⇒ { return x * y }; // ES6
```

Array/Object operators

```
const numbers = [1, 2, 3];
const oddNumber = numbers.find((x) ⇒ x % 2 == 1);
console.log(oddNumber); // 1
const oddNumberIndex = numbers.findIndex((x) ⇒ x % 2 == 1);
console.log(oddNumberIndex); // 0

let fruits = ['Apple', 'Orange', 'Banana'];
let newFruitArray = [...fruits]; // copy of array

let arr1 = ['A', 'B', 'C'];
let arr2 = ['X', 'Y', 'Z'];
let result = [...arr1, ...arr2]; // sum of arrays

const obj = { hello: '1', world: '2' };
const { hello, world } = obj; // destructed object
```

Practice time!

1. Refine your previous code to adhere to ES6 standards! Pay attention to scoped variables and function usage. 😊



TypeScript

What is TypeScript?

- Even more additions to JavaScript!
- This time - a language that *transpiles* to JavaScript using external tools, and is required mostly for development purposes.
- It brings **static types** to JavaScript in order to standardize the codebase and make the code less bugged, although it adds up a little maintenance time. 😊

Main points

- All Javascript **is valid** TypeScript.
- TypeScript files have a .ts extension.
- By default, browsers do not read TypeScript - so it has to be transpiled into JavaScript.
- To compile TypeScript files into JavaScript files, an external compiler is used.
- TS is used in a variety of JS frameworks - including Angular.

Examples of TS

```
function greeter(person: string) {
  return “Hello, “ + person;
}

let user = “Jane User”;
```

- Note the **:string** type parameter in the argument of the function.
- It means that if a value of a different type is given to the function, TS will throw an error!
- We can also define the structure of our classes in TS. The idea is the same - validation of fields!

```
document.body.textContent = greeter(user);


interface Person {
  firstName: string;
  lastName: string;
}
```

Practice time!

1. Install NPM on your local machine. It is a package manager for JS tools and can be downloaded from https://nodejs.org/en/.
2. Once installed, you need a tool named Tsc. Open your Command Prompt or Terminal and type:

   npm install -g typescript

3. Create a file named **index.ts** . Inside it, create a simple interface for a Student consisting of:
   1. Name
   2. Program
   3. Marks
   4. Active
4. Compile the index.ts into JS using the following command:

   tsc index.ts