DLA PIPER

DLA Piper

Jour 1

Simple page

DLA PIPER

Introduction

Welcome!
Today, we will show you an example of a very simple web page. The given wireframe only contains HTML (HyperText Markup Language), the key language in web development. We'll let you improve upon this simple page thanks to two other language, CSS (Cascading StyleSheet), which allows to customize you page style (colors, formatting …), and JS (JavaScript), which allows a website to become dynamic ("responsive"). For any question, please first visit this site, very well made. If, after your research, you still have doubts, we're at your disposal.

Crafting your request like this: *'ill understood topic* W3Schools' will save you significant amounts of time.

Given sources will show commentaries or FIXMEs to guide you.

Step 1 - Simple HTLM

For this step, we ask you to complete the existing page by adding **inside a text tag** the following message:

```
Hello World! The date is:
```

Still inside this tag, add an empty **span**, with the `clock` id:

```
<span id="clock"></span>
```

Of course, we'll add the code printing the current date in following steps.
Right beneath this text, add two **button tags**:

- The firs tone will show 'Turn on the red light!'
- The second one will show 'Paint it black!'

In the same manner, we will link the buttons to their actions later.

Step 2 - HTML - Form and Table

+ Form

Most websites will, at some point, need to ask some information to the user. This is called a **form**, allowing the user to enter text, images, documents… directly into the webpage. This information will, *in fine*, be gathered by a **server** to bo stored and saved.
Today, we will simply store this information **locally**, or temporarily.
Add a **form containing three forms**:

- A text field to enter a name. This field is called `nameInput`
- A numeric field to enter an age. This field is called `ageInput`
- A numeric field to enter a participation amount. This field is called `participationInput`

Remember to associate each field with a **label**, as well as a **submit button** at the end of the form.

+ Table

Now that the user can enter some information, we can print them! We will see how to print them dynamically later. For now, add an **HTML table**:

- The first line is the **table header**, composed of three columns (one per field, in the same order)
- The following lines will be added dynamically later.

Step 3 - Javascript and simple CSS

+ CSS

CSS is a very simple but flexible language, allowing to format a web page easily. For instance, the website we have here is nowhere near responsive: it doesn't adapt to screen size, nor does it adapt to different devices (iPhone, tablet, pc …).
To fix this, the world wide web gives some simple and efficient **CSS directives**:

- Add a tag defining the **viewport** of your webpage.
- In the given CSS file, add a selector for the whole **body** of your page.
- Add those directives in the previous selector:

```
display: flex;
flex-direction: column;
align-content: center;
align-items: center;
```

We won't spend too much time explaining each of them; just be aware we are using flexbox, and that it allows to layout a webpage really easily and in a flexible manner. Usually, they are explicit enough so that you understand what they do. But feel free to browse the given link for more explanation. Mastering this technology can become really time-consumming though.

+ JavaScript

For this step, it is primordial that you **correctly identify your HTML tags**.

The `id` attribute is your friend.

Now that we have the static wireframe of our page, let's make it dynamic. We'll start bu adding actions to our buttons. For this tutorial purposes, we give you the code for the 'Paint it Black!' button.

```
document.getElementById('toBlackButton').onclick = function dateToBlack() {
    document.getElementById('clock').classList.add("black");
    document.getElementById('clock').classList.remove("red");
};
```

JavaScript is a dynamic and flexible (sometimes too flexible…) allowing to manipulate you page elements **in real time**; HTML being a simple markup language, interpreted only once on page load, it doesn't allow dynamic actions by itself. It is just an ingredient list. JavaScript is the 'recipe' part, how you chain actions, what are their effects, etc…

`document` reprensents our page. We call the `getElementById` function, which does exactly as advertised: it gets the element with the id passed as parameter.
`on-click` is an **event**: in most cases, it represents some user action. Generally, an event is an action that comes 'from the outside', from another part of the program, from a different source … We assign (fill) this event with a function, called `dateToBlack`, which also gets another element in our page, in order to manipulate its `classes`. Understand **CSS class** here, a way to identify our elements to style them more easily from CSS. Refer to W3Schools and the given source code to understand the link between **CSS classes** and **HTML Attributes**.

Traditionally, we wouldn't put JavaScript directly inside the HTML page, but in its own files instead, referenced via specific tags. Today, for clarity reasons, we'll make everything fit in one HTML and CSS file.

Following the same pattern than the given function, bind a function to the `on-click` envent of our other button, which will color the date in red.

Don't forget to remove the 'black' class!

Step 4 - JavaScript form

Our form doesn't do anything yet. Check the code in step4: the `validateForm` function checks that the user didn't leave any field blank before submitting the form. If everything went fine, it returns a list of what the user typed in, or an empty list otherwise.
Then, check the `addToTable` function. This one is called when the user clicks on the form's submit button, checks the form's content thanks to the previous function, then fills the table with the newly formed data.
Following the given example, fix the end of the function so that the two other fields (age and participation) get their value added to a new line in the table.

Step 5 - CSS Framework

This step is just what you could see when using a **CSS framework**, a (huge) list of prebuilt CSS selectors allowing you to style your elements just by adding a class to them.
The one we use here is called MDL, and implements Google's Material Design. Take time to see how the elements have been modified compared to the previous step: some simple modifications to the `class` attribute and your page suddenly is material!
Other CSS frameworks exist, especially Bootstrap or Foundation.