SCA Workshop : guide and summary.

Antoine Kahn
University of British Columbia - Université Paris Cité - Sciences Po Paris
name.family@sciencespo.fr

Abstract

This document is a simple and beginner-friendly introduction to web development, focusing on HTML and CSS. It's aimed at people without a technical background.

1 Introduction

One day, maybe, while browsing your favorite website, you accidentally clicked on "inspect" or "view page source." On your screen, you saw a dark page with some words written in color, and for a second, you thought you were a powerful hacker in the Matrix.

1.1 Code or no code ?

What you saw is the code behind the website. When creating a website, you generally have two options. You can use something like WordPress or Wix, where building your website is almost as easy as creating a simple Word document. No coding is required on your part, and you can simply choose where to place things without any IT experience. A huge portion of the internet is built using these platforms, and many web projects can be completed this way. But, as you might guess, big websites like Amazon don't run on WordPress. There are actual people coding those websites "directly."

1.2 HTML, CSS and beyond

So, what exactly is this "code" we're talking about? Well, the most basic "languages" are HTML and CSS. In fact, almost all the technologies used today are either built on HTML and CSS, or are closely dependent on them. That's what we're going to teach you today.

Back in the day, you were pretty limited if you only used HTML and CSS. You might remember needing Java to play your favorite game on jeux.fr. Nowadays, you can create impressive things using only HTML and CSS. However, most companies use other languages, which are actual programming languages, to accomplish things that can't be done with just HTML and CSS (for example, JavaScript). And (as per industry standards), they often use what we call "frameworks," which are essentially tools to create coherent and efficient web applications.

But first things first: why are HTML and CSS not considered programming languages? Why, if you tell an IT person that you "program" in HTML, will they laugh ?

1.3 Programming languages

Without getting into technical details, it's because HTML and CSS are more like description languages than programming languages. If you've coded before, even just a little, you won't find variables, loops, conditions, red error messages, or even the concept of "executing your code." None of that exists in HTML and CSS. Think of these languages as a "coded" version of a word processing

software: you add elements, modify them, mark something as a heading, italicize text, and so on. There can't be any real "errors," just a webpage that doesn't look the way you want. On one hand, this avoids certain frustrations—like spending three hours stuck because you forgot a closing bracket in your code. On the other hand, it greatly limits the possibilities: for example, with pure HTML/CSS, you can't really create what we call back-end systems, like databases with users, signups, logins, etc. You also won't be able to create many animations or certain stylish layouts, which is part of front-end development. The technologies used to do these two things are constantly changing, but you'll often hear about Angular, Vue, SQL, and others.

1.4 Why ?

That said, why learn HTML and CSS if WordPress can do just as well (to be honest, even better) than you, at least until you reach a certain skill level? First, because learning how the technologies we use every day work (in general terms) is necessary to avoid being dependent on the tools. Second, because HTML and CSS are really the foundation: you could start learning more advanced web development technologies, but you'll quickly find yourself lost, not truly understanding how your page works. It's generally a good starting point because in a relatively short amount of time, you can create something in HTML and CSS that you can be proud of—whereas it might take you a week to display three lines of text in other languages. However, it has the disadvantage of being far less versatile than other technologies, as HTML/CSS is very rarely used outside the world of web development.

With that said, let's get started!

2 Work Environment

To code in HTML/CSS, you'll need two things.

2.1 Web browser

First, a web browser—any will do. Most of the time, the display will be the same across different browsers as long as you're using the latest version. You can use Chrome, Firefox, etc.

2.2 Text editor

Then, you'll need to choose a text editor: this is where you'll write your code, which your browser will display. There's a bit of a debate over which text editor is the best. I think Sublime Text is a good starting point because it doesn't have too many unnecessary features and is quite customizable. More advanced text editors include PHP Storm or VS Code. Note that, technically, you could code in any simple notepad (Notepad on Windows, TextEdit on Mac, etc.). A text editor is just there to "help" you—you'll soon understand why.

3 First HTML Page

3.1 Create index.html

To start, create a regular folder somewhere on your computer, for example on your desktop, and name it whatever you want, like "ubc_learning_exchange_coding." These two elements don't really matter. Then, open your text editor and create a file named index.html. The name here does (slightly) matter: if you ever host your website, this is the file users will "land" on first. We'll explain all of that later. Save your file in the newly created folder, and open it with your web browser (e.g open with > google chrome). Wow, a blank page should open, amazing !

3.2 Here's my website !

There's something quite important to understand here. You're in your web browser, so you might think that the file you just opened is a website. But take a closer look at the URL (its address): it doesn't look like the typical URL of a website with "http" and "www." Instead, it just shows the location of your file. In reality, your "website" is, for now, just a file. If you want to share it with a friend, you'll need to send them the file (via email, for example), and they'll have to open it the same way you just did.

Your file is only on your computer, and it's not even hosted locally, so no one else has access to it. It's like a PDF file that you keep saved in your downloads. Therefore, it's not a site that anyone can access simply by knowing its address. The point here is: avoid sending the file path to share your work—it won't work.

4 Structure

Now, edit your index.html file by copying and pasting the following:

HTML structure

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-
    width,initial-scale=1">
        <title></title>
</head>
<body>

</body>
</html>
```

Save your file and refresh the page: nothing changes! But there's no error—actually, the code you just entered doesn't display anything. In fact, it's the structure of your page, which is necessary to have. It's like writing an essay: you can write "I)a)" and "I)b)" on your paper, which won't earn you any points by itself, but if you don't do it, no one will understand your plan. The structure of the code is what allows your web browser to understand the different parts of your page. Think of your webpage as a human body: it has different parts (a head, etc.). Creating a website is like creating a human: you define what's in the head and the body.

Here, you start with <!DOCTYPE html>: this is like telling the web browser, "OK, this is an HTML page" (just in case it didn't already understand that from the fact the file is named index.html). This was necessary in the past, and while it might not be as crucial today, you should still include it for compatibility reasons.

Now, here's where things get tricky. Notice two things:

First, you see words between < and >: these are called tags. Most of the time, a tag is like a door—it needs to be closed. The usual syntax is: <tag> content of the tag </tag>. A tag represents an element of your site: it could be a form, a heading, or an image. Adding a tag is like adding a body part or an idea to your human-site.

Secondly, notice that the different tags are not aligned; they are indented (<meta> is to the right of <head>, for example). Unlike in Python (if you're familiar with it), indentation is not essential in HTML. In other words, your site will work just fine if you align everything to the left with no indentation. However, aside from aesthetic considerations (yes, code can be beautiful), you'll get lost without indentation. Let's say, for your human-site, you want to add an arm. So you add an <arm> tag (which, of course, doesn't really exist). But now, let's say you want to add a bracelet to the wrist.

The HTML logic goes like this: you add a <wrist> tag inside the <arm> tag, then a <bracelet> tag inside the <wrist> tag. Without indentation, it looks like this:

Arm & bracelet - no indentation

```
<arm>
<wrist>
<bracelet>
</bracelet>
</wrist>
</arm>
```

Now, if your site has 1,782 lines of code, poorly indented code will give you nightmares (trust me). The following code is much clearer, and it's super important that you understand the logic: indentation makes your code clearer by showing which elements are dependent on (or "included within") which other elements.

Arm & bracelet - indentation

```
<arm>
    <wrist>
        <bracelet>
        </bracelet>
    </wrist>
</arm>
```

Now, back to our main topic:

Back to basics

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-
    width,initial-scale=1">
        <title></title>
</head>
<body>

</body>
</html>
```

You have told your browser, with the first line, that we're going to communicate in HTML. Now, you use the <html> tag: this means, "OK, here's the beginning of the human (the website) that I'm going to build." My human has a head first (<head>). You might wonder why the <head> tag isn't indented, given that the head is "inside" the human. That's a good sign—it means you understand the logic. It's a matter of style: often, we do it this way because the content within the <head> tags is usually quite long, but feel free to do it however you want; it's not a big deal.

4.1 <head>

The <head> tag of a website contains, essentially, the metadata of the site, meaning all the data that isn't directly displayed on the site. To start with, you'll need to indicate:

- The encoding (here <meta charset="utf-8">), which allows for the recognition of French accents. For example, if you want to write in Mandarin, use a different charset whose code you can easily find on Wikipedia.

- <meta name="viewport" content="width=device-width, initial-scale=1"> is not very important; I believe it's a matter of compatibility.

- <title></title> is the title of your page, which will appear in the tab of your web browser. You can modify it by, for example, writing your name, saving it, and then checking the result in your browser.

Notice that some of those tags do not follow the normal structure <tag> content of the tag </tag>, and that's OK. It's because they don't really have "content".

5 <body>

Now that we've taken care of the head of our site-person, it's time to focus on its body: this is where the fun begins.

5.1 Texts

There are plenty of ways to write text in HTML, but depending on its role in your page, you'll need to place it in a different tag.

5.1.1 Titles

First, we have the headings. They go from <h1>, <h2>, ... up to <h6>. <h1> is a heading, <h2> is a subheading, <h3> is a sub-subheading, and so on. Try adding this to your index.html file :

Titles

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-
    width,initial-scale=1">
        <title></title>
</head>
<body>

    <h1> Emily Cooper </h1>
    <h2> Social Advisor at Savoir </h2>

</body>
</html>
```

Save it, refresh, and there you go! The beginning of Emily Cooper's resume. You can go on and write, for instance, <h3> Under the supervision of Sylvie </h3>, which is like an information "less important" than the fact that Emily is a Social Advisor at Savoir.

Now, for my first confession: in reality, the order you give to your elements (for example, using an <h2> instead of an <h3>) doesn't really matter because, as we will see later with CSS, you can easily make an <h3> element larger than an <h2> element, and so on. BUT, don't forget about it too quickly: it's good practice to ensure that your tag choices roughly reflect the hierarchical order of your elements. First, because otherwise your code will be a mess. Second, because a site with consistent tags is better indexed, meaning it appears higher in Google search results (look up SEO if you want to learn more about the subtle art of optimizing your site's ranking). Finally, consistent tags help make your site more accessible, for example, to visually impaired individuals who rely on clear HTML code to navigate your site.

5.1.2 Paragraphs

For paragraphs (e.g long texts), just use the   tag

5.1.3 Lists

Lists in HTML have a somewhat unusual format:

Lists

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-
    width,initial-scale=1">
        <title></title>
</head>
<body>

    <h1> Emily Cooper </h1>
    <h2> Social Advisor at Savoir </h2>

    <h2> Boyfriends </h2>
                <ul>
                        <li>Doug</li>
                        <li>Mathieu Cadault</li>
                        <li>Thomas</li>
                        <li>Timothée</li>
                        <li>Alfie</li>
                        <li>Gabriel</li>
                        <li>Marcello</li>
                </ul>

</body>
</html>
```

The <ul> tag indicates that you are starting a list, and the <li> tags differentiate the various items. Of course, the tag indicates the end of a list item, and the tag signifies that you are finishing your list.

5.2 Comments

Now, let's discuss a slightly different element that can go in the body or anywhere else in your code: comments. The concept is very simple: it's a piece of text, written in human language, that the computer should not take into account, and it serves only one purpose: to make your code more readable. The concept is the same as comments in a Google Doc, for example: it's a little note to remind you how something works or something to add. Here's the structure, along with some examples :

Comments structure

```
<!-- This is a comment : the computer will not care about it.-->
```

Comments in action

```
<!DOCTYPE html>
<html>
<head>
        <!-- TO DO : add stuff in the head -->
</head>
<body>


    <!-- That's a way to add pictures to your code -->
    <img src="pic/emily.png" alt="Cute picture of me."/>


</body>
</html>
```

5.3 Pictures

Do you really think our favorite Instagram addict was going to create a website without a photo of herself? Here's how to add a photo of Emily.

Pictures

```
<!DOCTYPE html>
<html>
<head>
        <!-- From now on, we'll stop copying the code every time. -->
</head>
<body>

    <img src="pic/emily.png" alt="Cute picture of me."/>

</body>
</html>
```

You may have noticed that the <img> tag is one of the exceptions: it doesn't have a start and end tag. Its structure is always the same: the src parameter indicates the location of the file. Here, for example, in my folder where my index.html is located, I have a subfolder called pic that contains a file named emily.png. To make this work for you, you need to replace this path with the location of the photo you want to include on your site. You can also use the link to a photo found on the internet, but that can cause problems; generally, we try to save the image locally. Speaking of which, for posterity, it's a good idea to try to keep your images as lightweight as possible, so you don't end up with large bills if you host your site one day (so you can ignore what I just said for now). The alt attribute indicates the text to display when hovering over the image with the cursor or if the image file is not found. Again, it's not mandatory, but it's good practice to always add an alt tag for accessibility and SEO reasons.

5.4 Links

Emily has no less than 23 billion followers on Instagram! To grow her community, she decides to add a link to her profile on her site.

Links

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

</head>
<body>

    <a href="https://www.instagram.com/emilyinparis/" target="_blank">
        Click here to check my Instagram profile !!!
    </a>

</body>
</html>
```

Here's the translation:

I recommend really trying this one out to understand how it works. The href option indicates the target of the link. Here, it's an external link (Emily's account), but you can also link to another file located in your working directory. For example, if in addition to your index.html file, you have a file named academics.html, which contains Emily's scientific publications (because yes, she has many strings to her bow), just write:

Intern links

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

</head>
<body>

    <a href="academics.html" target="_blank">
        Click here to check my publications  !!!
    </a>

</body>
</html>
```

The target="_blank" option simply indicates that when clicked, it will open in a new tab.

5.5 Forms

We're almost done. Emily has several million secret admirers and companies looking to recruit her. She also has visa issues because she can't get an appointment at the Paris prefecture. So, she needs a way for people to contact her!

Intern links

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

</head>
<body>

    <form action="contact.php" method="POST">

                <label for="nom"> What's your name ? </label>
                <input required type="text" name="nom"
    placeholder="Name">

                <label for="mail"> What's ur e-mail ? </label>
                <input required type="text" name="mail"
    placeholder="E-mail">

                <label for="wanna_be"> Who are u ? </label>
                <select required name="wanna_be">
                        <option>My next boyfriend.</option>
                        <option>A marketing agency.</option>
                        <option>The french administration.</option>
                </select>

                <input type="submit" name="submit">


        </form>

</body>
</html>
```

Don't panic, but it's a bit of a mess. A form logically begins with the <form> tag and ends with the </form> tag. The action parameter is the limit of this introduction: to process the information from this form (for example, to send an email or save the data somewhere), you'll need something more than just HTML and CSS. Maybe we'll cover that in another workshop; maybe not. The same goes for method="POST". Basically, your form looks nice, but you won't be able to do anything with it for now, sorry. The rest speaks relatively for itself, and I encourage you to test the code to understand for yourself the usefulness of the different tags. Also, try right-clicking > inspect to discover this mystical feature of your browser that allows you to understand your code in detail, and often its errors. Note that there are many different types of inputs (checkboxes, colors, dates, etc.). You can easily find the list online, for example, on the Mozilla website.

6 CSS

You can be proud of yourself because you just built your first website! But throughout this little journey, you've probably thought that while these stories about tags are amusing, the result is... ugly. And you're right: unless you're a genius mathematician[1], you have to be quite bold to present a website made solely with HTML, in other words, an ugly site.

This is where CSS comes in. With your HTML skills, you can create a human (a website) that is absolutely beautiful, like a statue displayed in the Louvre. But your human will be completely naked. CSS is its clothing. Without HTML, CSS is useless to you: you have no one to dress up (understand:

[1] https://michel.talagrand.net/

adding color, choosing pretty fonts, etc.). Without CSS, HTML condemns you to ugliness and nudity. In both cases, you can't leave your house (understand: publish your website).

6.1 CSS + HTML

So create a file named style.css, just like you created the index.html file, in the same folder. In this case, the name of the style.css file doesn't matter at all. You then need to link you HTML page to you CSS page. In other words, you indicate that the properties (making a button red, changing the font of a title) that you will write in your style.css file should apply to the elements in your index.html file.

CSS link

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

    <!-- Add this to your head -->
    <link rel="stylesheet" type="text/css" href="style.css">
    <!-- -->

</head>
<body>

</body>
</html>
```

Now, how do we "code" in CSS ? That's even easier than in HTML, as it always follows the same format :

CSS format

```
things_you_want_to_modify
{
    property:value;
}
```

For instance, this will turn all your h1 titles pink :

Pink Titles

```
h1
{
    color:pink;
}
```

6.2 Class & Id

With this, you can modify all the h1 elements, all the paragraphs, all the lists, etc. But how do you modify just one h1, assigning it properties that other h1 elements shouldn't have? For example, you want to ensure that one image is a certain size while another image is a different size. To do this, you need to return to the HTML and add an id to an element.

Lists

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-
    width,initial-scale=1">
        <title></title>
</head>
<body>

    <h1> Emily Cooper </h1>
    <!-- Here -->
    <h2 id="position"> Social Advisor at Savoir </h2>

    <h2 id="boyfriends"> Boyfriends </h2>
                <ul>
                        <li>Doug</li>
                        <li>Mathieu Cadault</li>
                        <li>Thomas</li>
                        <li>Timothée</li>
                        <li>Alfie</li>
                        <li>Gabriel</li>
                        <li>Marcello</li>
                </ul>

</body>
</html>
```

And then, in your style.css file :

Id CSS

```
#position
{
    color:pink;
}
#boyfriends
{
    color:blue;
}
```

Note: IDs are unique, meaning you cannot use the same ID for two different elements (in practice, this usually causes problems, but in this case, it's really not good practice, especially if you later add JavaScript to your site). If you want to add properties to multiple elements, but not to all elements (for example, if you want to style an h2 and an h3 in pink, but not the other h2 and h3 elements), you can use classes. The principle is exactly the same as for IDs, except that instead of using id, you use class, and instead of #id_name, you use .class_name in your CSS.

6.3 Usefull things

You can do plenty of things just with CSS. There are therefore many (many) properties available. If you're looking to underline text in green, search for that on Google and you'll quickly find the answer. Here's a basic list that allows you to do some nice things.

background-color	Use this to color your body, or any box-like element
font-family	The font of a text. Use google-font if you want cool fonts, just copy-past the link they'll provide you in your head. Be careful, not GDPR-friendly.
font-size	Use em instead of px to set the size of a text to make it more responsive.
text-decoration	Often used to underline stuff, or to remove the dirty aspect of link using the value none;
width	Usefull for pictures.
margin	Use either margin-top, margin-right, margin-left, margin-bottom or margin : value_top value_right value_btm, value_left. Use % instead of px for responsiveness.
padding	Similar to margin, but states the distance between an element and the things inside it; not the distance between the element and the stuff outside it. Exemples are great to understand it.

6.4 How do I center a div ?

We come to what I believe is one of the trickiest parts of CSS (besides animations, but you won't be using those every day): alignment. Specifying exactly where you want your elements to be is quite complicated, or rather, it is governed by a rather unique logic. Moreover, there is an imperative to consider, which I've touched on here and there, but which constitutes a nightmare for many people: responsiveness, meaning that your site looks good on all screen formats. Resize your window to realize that your site, which looks so good on a large screen, is completely unreadable on mobile, etc.

We won't go into detail here about how and why we can align things. In fact, you need to think of a website as a collection of containers; it's a set of small bricks nested within each other, squares that you want to place in certain spots and not others, relatively to each other. To do this, it used to be a huge hassle. For some time now, something called Flexbox has been invented, which is still a hassle but less so. To create a container with multiple elements, use the div tag :

Div

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-
    width,initial-scale=1">
        <title></title>
</head>
<body>
    <div id=container>
        <h1> Emily Cooper </h1>
        <h2 id="position"> Social Advisor at Savoir </h2>
        <img src="pic/emily.png" alt="Cute picture of me."/>
    </div>
</body>
</html>
```

By doing this, you're not doing anything. You're creating a sort of big box (currently invisible) that contains three elements: an h1 title, an h2 title, and an image. By default, these three elements are stacked one after the other, but if I do this:

FlexBox 101

```
#container
{
    display:flex;
}
```

Bam, the three elements are aligned! Then, there are many properties that I don't know by heart that allow for more precision, such as managing the spacing between the three elements, the vertical alignment relative to which of the three elements, etc., etc. In the inspect tool of Chrome (and a bit in Firefox), you have a very useful tool that allows you to test all the Flexbox properties (for example: justify-content: space-around;). It's a skill to learn, but it's really necessary to write clean code on this side to avoid having to redo everything at the end for the mobile version of your project.

7 Conclusion

We're reaching the end of this very brief introduction. With this, you can already create some pretty cool sites. This was obviously not exhaustive; you'll find books of several hundred pages that are. The idea is to learn how to search for the functionalities you want to implement and to always try to produce clean code. When you're just starting out, and if you want to progress, try not to use ChatGPT at all. If you want to publish your site, obviously pay attention to the legal aspects. You can use your own server (complicated and risky, but cheap) or pay for hosting (a bit more expensive). To go further:

7.1 More HTML/CSS

- w3.org

- openclassrooms.com

- youtube

7.2 Advanced Web Dev

- https://legacy.reactjs.org/community/courses.html

- https://www.w3schools.com/php/php_intro.asp (people usually hate PHP tho)

- https://www.w3schools.com/sql/sql_intro.asp

7.3 General CS

- https://pll.harvard.edu/course/cs50-introduction-computer-science

- tryhackme.com

- https://linux.org/

References

[1] https://emilyinparis.fandom.com/wiki/Emily_Cooper