Lab instructions: HTML + CSS

In this course we will work with HTML, CSS and JavaScript. In this first lab we will start looking into HTML and CSS from the beginning. With HTML and CSS it is possible to make almost all the layout (and some interaction) that is necessary for an interface to a standard application. In a later lab we will also cover the JavaScript necessary for the project. Note that these labs are starting from the very beginning and will teach you the basics necessary for your own exploring of the languages.

This lab instruction is to some extent also an instruction for HTML and CSS, and the examples that are provided are not always given as finished program files. In the case we refer to a file in the tutorial, it will be indicated by the file name in small lettering to the left:

index.html

These files are provided to you for exploration. Don't just look at the file, but try to change things in the file and see what happens. In the beginning there will not be very much that can be changed around, but towards the end you have large possibilities. If you want to learn the most, then try to guess at what the result will be after a certain change in the files, **before** you reload the file to view the results. This gives you a more thorough learning. If the expectations were wrong, then try to understand why you made a wrong guess.

A very good source for understanding the different building blocks used in this tutorial can be found on the w3schools website:

- http://www.w3schools.com/html/default.asp
- http://www.w3schools.com/css/default.asp

(there is also a similar page for JavaScript:

- http://www.w3schools.com/js/default.asp)

Here you can find almost everything you need to know about HTML5 and its companions CSS and JavaScript. However, in this tutorial, the intention is to provide a little bit smoother way into the area, starting from a few very concrete small examples.

HTML

HTML is a so called *Markup Language*, which means that it is not run like an executable file, but rather interpreted as a structure in which the content should be placed. HTML is in this way similar to XML, but HTML has a more defined set of markup *tags*. A tag is a string enclosed within "<" and ">", for example "`<h1>`" (which defines a heading on the first level). Most tags should have a closing tag as well, for example `</h1>`. The text between an opening and a closing tag is called an *HTML element*.

HTML tags and *HTML elements* are two important terms that you should remember throughout the course. It will be used when we start working with JavaScript as well.

The HTML code is most often interpreted in a web browser, and unfortunately different browsers will interpret some tags slightly differently. These differences are fairly well documented, but for you, the most important consequence is that you should use the same browser within the project.

The most important advantage of using HTML/CSS files is that they are encoded as plain text. This mean that it is a human readable code, and can for example be transferred and used without having to be compiled or packed into archives. It is (if it is

well structured, of course) easy to share between members of a project, and is also suitable for some trial and error, since changes are directly visible when the page is reloaded.

The simplest file in HTML is an empty file, but then there are some stuff that should be placed within a file in order to make it a well-defined HTML file. If you create a new and "empty" HTML file in WebStorm, the content of this smallest file will be roughly as follows:

index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>

</body>
</html>
```

Create a new document in WebStorm so that you have a file that looks like the one above. Save it in a new folder (this will be our sandbox folder for this lab). The main file in a HTML project should have a name such as main.html or index.html (which indicates that it is the one that is the entrance page to the project).

NOTE: You should already now remember that HTML is not case sensitive as far as the tags are concerned.

Thus, tags such as `<hEaD>…</HEAD>` works just as fine as `<head>…</head>` but it is better if you decide for some convention already from the beginning. Above all, make sure that your code is readable (also by using indentation as in the example above). HTML and CSS are difficult to debug anyway. Don't make your work more difficult than needed.

If you open this document in the web browser, you might be a bit disappointed. All you can see is a blank web page. But that is only because we haven't written any text in it yet. All that is there are some tags that in themselves need some content.

Some explanations:

- `<!DOCTYPE html>` is a meta command, which simply states that this content should be read as HTML instead of plain text. You don't have to understand this at the moment, just make sure that it is available in all HTML-files you create.
- `<html lang="en">…</html>` means that the html content of the document is between the tags and that it is written in English.
- `<head> … </head>` is the head of the document. This is where most of the administrative parts of the document goes. Normally the content here is not displayed directly (but see more about this below). Here is also where you include other files, e.g., CSS-files, and JavaScript files. Try to write something between the `<title>…</title>`-tags and reload the file in the browser. Did you see any difference?

- `<body>…</body>` is the content part of the document. Here is where the content that will be displayed on the page goes. This is also where we will put most of the project material. And now we are ready to do some HTML marking.

There are also some other tags in the "empty" file, but we will not go into them in more detail. It will be clearer later, how the different items contribute to the HTML code.

Plain text

As you can see text in HTML is written just as plain text. There is no formatting of the text in the source file. Try to start by putting the cursor into the body part of your file and write some text.[1] Two or three short paragraphs will be enough to start with. (Don't use any tags to begin with). As you may see when you load the file the two paragraphs are joined with a space between them. This is because within HTML text without tags is treated as if they were written just after each other. Any whitespace will be treated as a space. Also, which is worth noting several spaces after each other will also be treated as *one single space*. If you want to have more space between words or letters you have to use non-breaking spaces or special spacers in the text. To get paragraphs, you have to enclose the text within `<p>…</p>`-tags.

```
<p>Some text</p> <p>Some more text</p>
```

As you can see, this makes two paragraphs, even if they are written on one single line. You can also not that there is some padding between the paragraphs (they are not written directly after each other). Your indentation and layout of the text in a HTML file is only dependent on the tags you use and how the styles of these are defined.

Now it is possible to start creating more interesting HTML files. Extend the file by adding a Heading on level 1 before the two paragraphs. Reload the page and see if you were expecting the right thing.

```
<h1>This is the first Chapter</h1>
```

The heading might look a bit big to start with, but we will change this later in the lab. Now we proceed to make subheadings. Add a subheading on level 2 to the file.

```
<h2>This is the first subchapter</h2>
```

As you may have realised already, HTML does not help you with the numbering of paragraphs, in the way that Word can do. We can help this as well, but we will get back to that later in the course (it is not a major issue for the project).

Add some more text to the document, and some more headings (there are several more levels than h1 and h2).

> Always add a closing tag to the standard tags. In some cases the closing tag is optional (e.g., `</p>`), but it is not considered good style to leave it out.

Italics and Boldface

HTML was initially intended to support semantic markup of documents. That is, for example, the reason that we have paragraph and body text. Unfortunately this was destroyed by the addition of the tags `<i>…</i>` and `<b>…</b>`. Both these (and a few others) affect the looks of the text by using *italics* and *bold face*. The semantic meaning is to provide markers for *emphasis* and *strong emphasis*. Therefore the use of these direct

[1] If you are tired of writing text samples, why not use the famous "Lorem Ipsum"-text? On the page `http://en.lipsum.com/` you will find a Lorem Ipsum generator that will give you sample text to use. There you can also find some interesting facts about the well-known text.

tags are discouraged and you should instead use the semantic markers `<em>…</em>` (for emphasis) and `<strong>…</strong>` (for strong emphasis). Try to emphasise some words in your text. Here it also worth noting that tags can be nested as long as they are closed in the reversed order.

NOTE: it is not a good style to use `<em><strong>…</em></strong>` instead of the proper `<em><strong>…</strong></em>`. You could get some unexpected results if you close the tags in the wrong order.

By now you should have a file that contains both headings of different sizes as well as some text in different paragraphs.

> You should be careful to always finish an inner tag before finishing the outer, i.e., use
> `<tag1>`
> `  <tag2>…`
> `  </tag2>`
> `</tag1>`
> rather than
> `<tag1>`
> `  <tag2>…`
> `</tag1>`
> `  </tag2>`
> or else you might get some both unexpected and undefined results.

Lists

The last type of tags we will look at initially are lists. We can make both bulleted and numbered lists in HTML directly.

A bulleted list is made by using the tag `<ul>…</ul>` (unordered list) and a numbered list uses the tag `<ol>…</ol>` (ordered list). Within the list you use the tag `<li>…</li>` for each list item in both list types. A typical unordered list would therefore look like:

Lists.html

```
<ul>
  <li>This is the first item.</li>
  <li>This is the second item.</li>
  <li>...and so on...</li>
</ul>
```

Add one in your file, and see what you get. Then just change the `<ul>` to `<ol>` (and the closing tags) and reload the file.

Links

The word Hyper in HTML actually means that there should be possibilities to get from one part of the text to another in a non-linear fashion. This is done by using hyperlinks. A hyperlink is an *HTML element* that responds to clicking by loading a new HTML page, moving to another anchor in the text, or in case of JavaScript activates a script. The essential structure of the link is (essentially) the same in all three cases. In the following example we link to the Google search page in the first link and to a local html file in the second. The `<a …>` tag initiates a link and takes one argument (in HTML called *attribute*), the path to which the hyperlink refers (href = hyper reference). The element enclosed by the tag, can be any element, such as a string of text, an image or an icon, as long as it is visible on the web page and can be clicked on. Even movies and animated canvases can of course be clickable if needed.

```
<a href="http://www.google.com/">The google search engine</a>

<a href="nextPage.html">Next page of this site.</a>
```

All tags can have on or more attributes, and the most common case is in the form of "`attribute= "String"`. If we use an attribute that is not defined, nothing happens (just as if we try to use a tag that is undefined). There are many clever attributes on the HTML

elements. Try for example inserting the attribute "`title="Here I am"`" into a `<p>`-tag. When you hover the mouse over the clause a tooltip with the text will be shown.

Images and other media

What is a web page without an image? An image is easy to add. The image is actually also one of the tags that does not require a closing tag (see the box above on closing tags). An image is easily added by using the `<img…>` tag.

```
<img src="mypic.jpg" alt="A picture of me">
```

In this tag there is first a declaration of the image location (in this case it has to be in the same folder as the html file), and then a string of text that will be used in case it is not possible to see the picture. This is the case, for example when a person with a visual impairment accesses the home page. Then the alternative text is used by the screen reader (which cannot interpret the image itself). You should always include an *alt string* to images movies, videos, etc. This is of course a requirement for your project too.

If you include images and other larger items on your page, you must have them on your own server. To include images through the URL on other servers is not a way to make yourself popular. What happens is that when someone uses your page, the image will be loaded from the other server, using that server's bandwidth. If you get heavy traffic, that may become expensive for the other web site owner

Divs

There is a special HTML element that is called a div. This element has no other purpose than to keep other elements (visually) together. One example of when this is interesting is when you have a long list of item in an ordered or unordered list, and want them to be uninterrupted by other elements, such as images or similar. Then you would put a div around the list, and then you can position the list anywhere you want on the page.

You could informally say that a div is a miniature web page, that can be placed anywhere on a bigger web page or div (yes, they can be nested). For now, we can just acknowledge their existence, and we will use them a bit more in the next part of the lab.

More HTML stuff?

As you may have guessed the HTML produced now is only the tip of the iceberg. There is much more to add in terms of tags and html elements. However, that is left as a follow up of the lab for you to do on your own. In the mean time we will take a look at how we can use cascading style sheets to change the looks of the page that you have now. Before you start looking at the CSS, add some more tags and elements, and try to make your page a little bit more elaborate.

CSS

Cascading style sheets (CSS) are a very powerful construct in the design of web pages (and other types of interactive applications as well). Through the use of CSS we can completely alter the layout, structure and looks of the web page, as well as some surficial aspects of the interaction. This is something we are going to explore in this part of the lab. But first we need to understand a few important concepts of the combination of HTML and CSS, namely classes and id:s.

Classes and ID:s

Every HTML element can be assigned two types of identifier, a class identifier and an id. Both of these are used to address the element in the CSS and later in the JavaScript programs. The identifiers are given as *attributes* to the tag surrounding the element. The attribute is either "`class="string"`" or "`id="string"`".

A `class` attribute is used to address all elements that have the same class attribute set. The elements themselves can be of different kinds (have different tags) but will be following the same definitions. The `id` tag is used to address a specific element, and in this case the value of the attribute should also be unique.

Elements can also be addressed in CSS and JavaScript, as we shall see here soon. Addressing elements is the most common use of CSS. Adding a class or id attribute even if you do not use them will not cause any problem. It will just be text being ignored by the browser.

Styles

The basic component of a style sheet is the concept of a *style*. A style is essentially a property that can be set on an HTML element, which defines how this element behaves on the page. For example, it is possible to set the colour of an element. This can be done in several ways:

- inline (not really recommended)
- in the head of the html file (internal CSS)
- in a separate style sheet (external CSS)

Of these the last one is recommended for most projects you will work in. The use of separate style sheets will allow you to use a very flexible approach to the interface design. However, we will also see how an inline and head definition looks. Inline styles look like normal attributes, and gives the style and its value as a colon-separated pair.

```
<h1 style="color:blue">This is a blue heading!</h1>
```

If we have several attributes on a single element, this quickly becomes very awkward to type and read. Also if we have to use styles in several places, it quickly becomes more handy to collect all the style definitions in the head of the HTML file. The pure HTML code becomes more easy to read, and most importantly, any change only needs to be done in the internal style sheet, in order for it to be applied to all objects of a certain kind.

InternalStylSheet.html

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {background-color:lightgrey}
        h1   {color:blue}
        p    {color:green}
    </style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

<h2>This is a subheading.</h2>
```

```
<h1>This is another heading</h1>
<p>This is another paragraph.</p>

</body>
</html>
```

However, if the web site consists of several pages, and needs to have a consistent design and layout, also the internal style sheet becomes tedious to maintain. Whenever we change a style in some way, this needs to be updated in every HTML file. Thus, the best way to use style sheets is to place them as external. There are several advantages to this approach:

- If we want to change the looks of an interface, we need just to change external style sheet (this can even be done with a JavaScript program).
- The same style sheet can be used in many pages, and we only need to change the style in one place.
- It is easy to get a good overview over the styles (although it might also be somewhat overwhelming in complex systems).

Instead of defining the style in the head we replace the definitions with a single import line (where "one.css" is the name of the file containing the style sheet):

```
<link rel="stylesheet" href="one.css">
```

in the head of the HTML file. If we create a new CSS file, we are given an empty file in WebStorm. So we need to add the styled from the internal style sheet.

```
body {background-color:lightgrey}
h1   {color:red}
p    {color:green}
```

As can be seen a style consists of a keyword (the HTML tag) and the style definition within curly brackets. When we want to have more definitions on the same tag, they are separated with semicolons within the curly brackets.

```
body {background-color:lightgrey}
h1   {
      color:red;
      font-family:verdana;
      }
p    {color:green}
```

In this style sheet we only use the standard tags, but we can also set styles on classes and id:s. This is done essentially in the same way. To differ between keywords we add a "." before classes and a "#" before id:s in the style sheet.

two.css

```
body {
    background-color:lightgrey
}
h1   {
    color:blue
}
p    {
    color:green;
    font-family:cursive;
}
.error {
```

```
    color:red;
}
#not1 {
    font-family: fantasy;
}
```

Using the tags and the class and id attributes together we can select almost any kind of combination of HTML elements and apply a certain attribute on them.

But what happens if we set the same element to different values at the same time in the style sheet? There are two rules that guide this. First they are assigned according to the kind of attribute used: ID is the strongest, class is the second strongest and the element tag is the weakest assignment. Thus an ID definition overrides a class definition which overrides a element tag definition.

Second, if two different rules apply to the same class or the same ID (for example in different style sheet files) the last rule is the strongest. So if a certain id is first given the type `color:blue` and later given a type `color:red`, the colour of the font will be red, rather than blue. This is not recommended as a standard way of working since it may cause very strange errors if it is not well documented.

> If you happen to change a style for some specific element, and regardless how you change it it will not appear in the desired way, there is very large possibility that there is an overriding style definition somewhere.
> This is a very common error to make, and likewise very difficult to find. But starting the debugging in the style sheet(s) is never wrong in such a case.

In the example above, it is possible to see that the id "not1" will apply the font-family "fantasy" to a paragraph, even though the "p" element has been assigned the font-family cursive. Also the class "error" will apply to any element that has class "error" in its attribute. In the file MixedStyles.html we use this stylesheet, in order to produce some strange styles. Go through the file and see if you understand the way the different style attributes override each other. In other words, try to understand why the different HTML elements are given their colours, font types and sizes. As shown, it can get pretty messy, if we are not careful.

CSS sizing and positioning

CSS can not only be used to change type and colour. It is also very good for layouts. In old times, the "best" way to position elements over the page was to use *tables*. However, it was not in any way a good way, and it provided a far too inflexible way to work with layouts. And of course, tables were originally not meant to be used for layout. With the advent of CSS, the whole layout process changed. Any HTML element can be given a size and a position on the page using CSS styling.

In the file `FloatingTitle.html` we can see that a heading can be floating over the text, using a position property set to "fixed" (see stylesheet "`float.css`"). However, this renders the text slightly difficult to read, which we would like to avoid. One way of avoiding this is to use div:s. An example of how div:s can be used is displayed in the file "`float.html`".[2] Here we use a div containing an image and the caption for it. The rest of the text floats around the image box. We will look a bit more into this example next.

The position property is described on the page:
http://www.w3schools.com/css/css_positioning.asp
together with other positioning attributes. Try to change the different positioning means and see how this changes the way in which the page is displayed.

[2] This example is adapted from w3Schools web site.

Using div:s and CSS together

Single elements are difficult to position in a useful manner. However, if we group objects in larger lumps, we get a much more resourceful way of using CSS for positioning. We can think of the div:s as puzzle pieces which each contain a certain bit of information, and have to be placed at a certain place in order to make sense. Regardless of the content of the div, the placement has to be easy and flexible.

When we are using div:s for placement, it turns out that we can place them wherever we want, almost. There are some constraints, but we will not go into details about them this time. For an example, have a look at the code in `FloatingDivs.html` and `float2.css`. Here we have started on a file that will display the Heading fixed at the top, and the text floating, scrollable underneath it. However, we haven't come all the way, since the text still floats a bit underneath the Heading text.

FloatingDivs.html + float2.css

Try to see if you can adjust the stylesheet so that they do not overlap. Also note that in the stylesheet we can make any div scrollable, in case the content does not fit within the size of the div. This means that we can place several longer articles on a web page, each with a small physical size, but scrollable to incorporate a larger content.

One final task for this file. Where in the HTML file is the definition of the header? What happens if you move that div to some other place? Try some different placements and see how this affects the layout? What kind of conclusions can we draw from this?

What now?

After this lab you should hopefully have some ideas about how:

- HTML works
- CSS works
- HTML + CSS work together

Now you will have to start exploring the ideas within the lab on your own. Draw some different layouts on a piece of paper and then see whether you can implement them using HTML and CSS. Use the resource collections on the W3schools web site and expand your knowledge about both HTML and CSS with this knowledge, and the information you can find in the pdf resources uploaded to the course website.

On a later lecture we will give you some more examples on how you can use CSS to master your HTML layouts, and together with this tutorial we hope that you will be up and running on the path of interface design.