2nd CAA

Alejandro Pérez

November 24th, 2022

Table of Contents

Question 1

CSS allows you to assign a maximum or minimum size to an element. Give an example of the use of each of the two options (giving a maximum size or a minimum size to an element), briefly describing its operation.

Answer 1 *

CSS is capable of sizing elements (text, media, and so on) in a smart way. Rather than having a fixed size set (which is also possible), it allows setting a minimum and/or maximum size for a given element. Thus, it is possible to set `min-height`, `max-height`, `min-width` and `max-width` according to our needs. In the case of `min-` sizes, we are telling *CSS* the **minimum** size that the box where the element is displayed will have. Similarly, when using `max-` sizes we are limiting the maximum size of the box where the content will be displayed.

Let us consider an example for each of these properties:

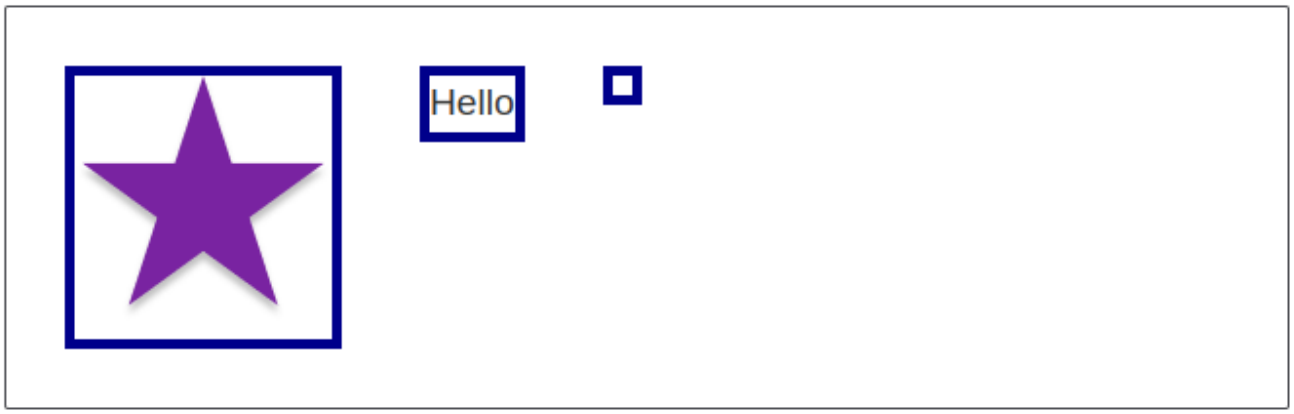

Figure 1: `min-` sizes example

In this case, we are setting that the boxes will **at least** have a width and height of `20px`. Both the `star` image and the `Hello` text need more than `20px` so *CSS* will provide the needed space for the boxes. The box with no content in it displays only the box of `20x20px` since no extra space is needed to display that element.

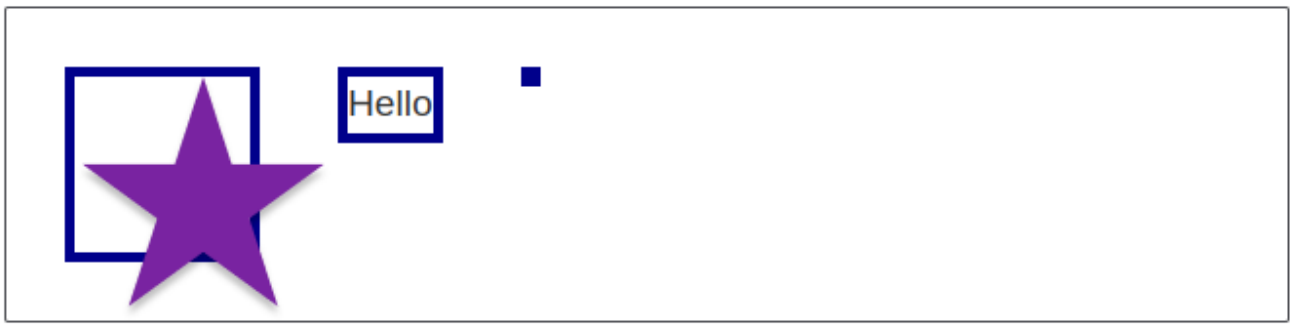

Figure 2: `max-` sizes example

For this second example, we set a **maximum** width and height of `100px`. This means that boxes can adjust their size according to the content displayed in them, but never surpass the maximum limit. The `Hello` text and the empty boxes both display with the necessary space, since they both need less than `100px` to display their content, but the `star` image is larger than the said maximum value. In this case, the content of the box will overflow. To prevent issues like this, it may be a good idea to set the property `overflow: auto;` in case our content doesn't fit a box with limited sizes.

* *Screenshots taken from MDN's interactive editor*

Question 2

Question 2.1

Research the existing accessibility legislation in your country of residence. In case there is no legislation in your country of residence, investigate the USA legislation. What are the main rules that regulate it?

Answer 2.1 In Spain, there is some regulation of accessibility of websites, though there appears to be no legislation according to the W3C's Web Accessibility Laws & Policies. That site is to be updated in **November 2022** and the current data appears to be from **2018**. Rather, this legislation is defined in the *BOE* (Boletín Oficial del Estado), in this Royal Decree-Law. It generally dictates a set of rules to ensure accessibility for certain entities. The decree aims to guarantee some accessibility so that people with disabilities or the elderly can access websites and smartphone applications without discrimination. This regulation is subject to the *Official Journal of the European*

Union (available here), the decree states, which releases further regulations that should be adopted by sites and applications according to the new needs that arise.

Question 2.2

Do a little planning for the accessibility of the website you are going to create in Part 2 of this PEC. Take as a reference the section Implement accessibility in your project, which you will find in Stage 6 of Unit 5. You will have to answer questions such as:

- What alternative texts should the images have?
- If a user navigates through the links via the keyboard (using the tab key), will they be able to clearly see which link has the focus?

Answer 2.2 Generally, in order to avoid accessibility inconsistencies it is important to meet all the good practices we have learned so far.

Here is a list of things to consider in order to ensure accessibility is met:

- Avoid using words like `here` or `from this link` as link text, prefer to have the link text contain information related to the actual link. This is more useful for screen readers to understand.

```
<a href="archlinux-2022.11.01-x86_64.iso">Download Arch Linux for x86_64</a>
```

- Make navigation links' boxes large for users with touchscreens.
- Hover and selection link colors and borders should be clear for users relying only on their keyboard.
- Use proper block and inline tags to let screen readers understand the type of content. E.g. the `<time>` can help screen readers announce dates and times more reliably, not to mention it also helps with *SEO* (Search Engine Optimization).
- Use skip link navigation and utilize the `screen-reader-text` class to clarify content that might be confusing for blind or otherwise disabled people.

Question 3

Explain, in your own words, what the concepts of cascade, specificity and inheritance are in the CSS language. Next, calculate the specificity of the selectors in the following list (do not show only the final value, indicate how to reach it step by step), and order them from highest to lowest specificity:

- `.header #author`
- `#author #nationality`
- `#author + span::first-letter`
- `p`
- `footer > .featured`
- `h1 + img[src*="UOC"]`

Answer 3

Cascade: property of *CSS* which consists of applying selector rules as in a cascade, i.e. from top to bottom. If you happen to repeat rules within the same cascade layer, the last one is the only one that will be taken into account, and the first one will be ignored.

Specificity: In cases where two or more rules target the same content, a conflict occurs. Specificity in *CSS* helps resolve these conflicts by measuring which of the conflicting selector has priority over the rest. The one with higher specificity will be the one to affect the target content.

Inheritance: Some *HTML* elements can have children elements. Inheritance consists on the ability of children element to inherit some of the properties of such element. For example, you can set the global font size of color of a page under the `html` selector, and elements in your page will base their font color or size depending on that parent's value. Not all attributes can be inherited, and attributes can be manually set to inherit or not inherit depending on our needs.

Order from lowest to highest specificity:

No.	Selector	Identifier	Classes	Elements	Total Specificity	Explanation
1	`p`	0	0	1	0-0-1	Single element
2	`footer > .featured`	0	1	1	0-1-1	One element and one standard class
3	`h1 + img[src*="UOC"]`	0	1	2	0-1-2	Two elements with one having a pseudo-class
4	`#author + span::first-letter`	1	0	2	1-0-2	One ID and an element which has a pseudo-element
5	`.header #author`	1	1	0	1-1-0	One ID and one class
6	`#author #nationality`	2	0	0	2-0-0	Two classes

Ordering by specificity is quite easy. We have seen that identifiers, classes and elements all play a role in calculating specificity. To order selectors based on specificity, we must take into account the following priority order:

`identifier > class > element`

Knowing this, if we look for example at the selectors `h1 + img[src*="UOC"]` (0-1-2) and `#author + span::first-letter` (1-0-2), it is clear that the latter has higher specificity, since an ID will always be prioritized over a class or an element.

Explanation of the HTML and CSS entities used

This last part corresponds to the code part of this practical (2. Exercise).

HTML entities: why have you chosen to use those specific tags?

The only *HTML* entity tag I have used is `>`, to distinguish the "greater than" sign `>`. I used it on the secondary page `ethe.html`.

The CSS used must be explained. Why have you chosen those selectors and properties?

For *CSS* code I used various forms of selectors and attributes inside. Here are the most notable ones:

- Initial values

```
*,
*::after,
*::before
{
    margin: 0;
    padding: 0;
    box-sizing: inherit;
}

html
{
    font-size: 10px;
    max-width: 1200px;
    width: 80vw;
    font-family: Arial;
```

```
    margin-left: auto;
    margin-right: auto;
}
```

This part set all base margin and paddings to zero by default, as well as forcing the box sizing to be inherited. This allowed to easily fine-tune the structure of the pages instead of using the browser's default settings which were often not ideal. We also change the base font size from `16px`, to `10px`, because it makes it much easier to convert `rem` values to `px` and vice versa (multiples of `10`).

The `html` selector formats the main display of the page. Specifically, we set the maximum width of our pages to `1200px`. We set the width of the page to `80%` of the view (`80vw`), and the left and right margins to `auto` to center the content.

- Screen Reader Settings

```
.screen-reader-text
{
    height: 0;
    width: 0;
    margin: -1px;
    overflow: hidden;
    position: absolute;
}
```

I added this accessibility selector following the convention. A user relying on a screen reader will have the opportunity to click on this invisible link and skip the header navigation bar, effectively focusing on the content of the page. This class can also be used to display extra content only meant for disabled users, in order to clarify content.

- Header

```
header
{
    position: relative;
    background-image: url(../img/header.jpg);
    background-position: left 25%;
    background-size: cover;
    border-radius: 1.5rem;
    height: 10rem;
}

h1
{
    width: 90%;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    font-size: 3.5rem;
    color: white;
    text-align: center;
}
```

The header includes the `h1` tag, which is centered with absolute positioning with respect to the `header` block, thanks to it having relative positioning. We set the proper background color and sizes, and center the text by first aligning it on the middle. This makes the top left corner of `h1` appear on the middle, so in order to center it properly we can reduce the `x` and `y` positions by `50%`.

- Navigation

```
nav
{
```

```
    height: 6.5rem;
    background-color: #e9791a;
    border-radius: 1.5rem;
    position: relative;
    text-align: center;
}

nav ul
{
    width: 100%;
    list-style-type: none;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}

nav li
{
    display: inline-block;
    width: 6rem;
    text-align: center;
}

nav li:not(:last-child)
{
    margin-right: 0.6rem;
}

nav a
{
    display: inline-block;
    line-height: 3.5rem;
    width: 100%;
    height: 100%;
    font-weight: bold;
    color: white;
    border: 0.1rem solid #4433ff;
    background-color: #4433ff;
    border-radius: 0.5rem;
    text-decoration: none;
}

nav a:focus,
nav a:hover,
nav a:active
{
    outline: none;
    color: #e9791a;
    background-color: white;
}
```

Using the same absolute-relative positioning as with the header, we can arrange our `ul` from the `nav` tag vertically. Then, we set the `li` and its inner `a` tags to display as `inline-block`, in order to lay them out horizontally and have the links' "click area" spread out to match the height of the whole box. Additionally, we set the proper coloring for text and backgrounds depending on the state of the link. We set right padding for every element of the list except the last one, since we do not need the extra gap after the last button in our navigation bar.

- Author and subtitle

```
.author
{
    font-size: 1rem;
    margin-top: 2rem;
    margin-bottom: 2rem;
    text-align: right;
    margin-right: 0;
}

.author span
{
    color: #e9791a;
    font-weight: bold;
}

h2 ~ p
{
    text-align: justify;
    margin-top: 1rem;
    margin-bottom: 2rem;
    margin-left: 9rem;
    margin-right: 11rem;
    font-size: 1.1rem;
}
```

Moving on below the navigation bar, we have the author which is set to be aligned to the right. This is a special class I call `author`. In order to color the name of the author, I used a span element. Then, we style the paragraph below in a different way from other paragraphs. We select this one paragraph by looking for siblings of `h2`. These documents only have one level 2 header, so this particular selector does the job. This special paragraph is narrower with justified text in italics.

- Borders around certain sections

```
section.border
{
    margin-top: 1rem;
    border: 0.1rem solid black;
    border-radius: 1.5rem;
    overflow: auto;
}
```

The document is split into various sections inside a main `article` tag. The website we have to replicate has some of the content with a rounded border around it. I created a `border` class to add this specific border to the right `section` tags.

- Block Quotes

```
blockquote
{
    position: relative;
    margin-top: 1.5rem;
    margin-bottom: 1.5rem;
    padding-top: 1rem;
    padding-left: 3rem;
    padding-right: 16rem;
}

blockquote::before
```

```
{
    position: absolute;
    content: open-quote;
    top: 0;
    left: 0;
    transform: translate(-65%, -25%);
    font-size: 4.5rem;
    color: #11b2e6;
}
```

Block-level quotes have a specific styling. First of all, they are indented (defined in other *CSS* selector), and their text is narrower. Then, to add the quotation mark at that place I had to use a similar absolute-relative position, to ensure that the quotation mark appeared on the top left of the quoted text.

- Description Lists

```
dl
{
    margin-left: 7rem;
}

dd
{
    padding-left: 2.5rem;
    text-indent: -2rem;
    padding-right: 16rem;
    padding-top: 0.6rem;
    padding-bottom: 1rem;
    line-height: 1.5rem;
}

dt
{
    font-weight: bold;
}
```

Styling description lists was mostly trivial. After setting some padding and changing the line height to replicate the model website, the main challenge here was padding the first line of our `dd` elements more on the left than every other line. I ended up setting negative indentations to tackle this issue.

- Lists

```
.list,
.list_nest1
{
    margin: 1rem 1rem 1rem 4rem;
}

.list_nest1
{
    list-style-type: square;
}

li.atom
{
    list-style-type: none;
}

li.atom::before
```

```
{
    content: "\269B";
    font-weight: bold;
    padding-right: 0.2rem;
}
```

To distinguish normal `li` elements from `nav li` elements, I created specific classes for each type of list I needed. I used the `list` and `list_nest1` classes to display normal lists. I set both classes to leave a margin `4rem` on the left, which creates the nested indentations I was looking for. I also created the `atom` class which uses a special character for its bullet points (an atom icon).

- Figures

```
figure img
{
    margin: 0 auto;
    display: block;
    width: 45%;
}

figcaption
{
    text-align: center;
    margin-bottom: 1rem;
}
```

Our figure was overflowing other content and the caption was not centered. To fix this, we simply set the `text-align=center` for the caption, and set automatic horizontal margins for the image and scale it down to about `50%`.

- Underline Variations

```
span.underline
{
    text-decoration: underline;
    text-decoration-style: dotted;
}

h3 span.underline
{
    text-underline-offset: 0.3rem;
}
```

Some words in the text had a dotted underline. In the `HTML` code I used `span` tags along with an `underline` class to identify them. The second selector above is for a very specific case. Looking at the guide image of the page `index.html`, I noticed that the dotted underline was slightly below the solid underline from the `h3` tag. To match the guide image, I set proper offset to the underline.

- Footer

```
footer
{
    display: flex;
    height: 7rem;
    font-size: 1rem;
    overflow: hidden;
    color: white;
    background-color: #4433ff;
    border-radius: 1.5rem;
    align-items: center;
    justify-content: center;
```

```
    flex-direction: column;
}

footer p
{
    text-indent: 0;
    margin: 0;
    margin-top: 0.5rem;
    margin-bottom: 0.5rem;
    width: 85%;
    font-size: 0.9rem;
    text-align: center;
}

footer a
{
    color: white;
}
```

For the footer I decided to use `flexbox`. It allowed for lots of simple and 'flexible' ways to arrange the two paragraphs to fit in the footer background. The rest was rather straightforward to tweak.