3rd CAA

Web Standards & Languages

Alejandro Pérez Bueno

December 6th, 2022

Table of Contents

Question 1 Positioning

Indicate which are the four main types of positioning in *CSS* and describe, in your own words, their main characteristics.

Answer 1

The four main types of positioning and their main characteristics are the following:

- **Static Positioning**: this is the standard way in which elements are laid out. It simply positions the elements following the flow of the document. Elements go right where we expect them to go if we did not set this property.
- **Relative Positioning**: this positioning type allows us to move the position of an element relative to its position in the normal flow. For example, if we set the property `top: 30px`, the element will be `30` pixels below its "normal" position.
- **Absolute Positioning**: used to position element relative to other (parent) positioned elements. It is quite neat in cases where we might want to position some content on a box (`<div>`). When no parent element is postioned, the base `html` element will be used to position the element.
- **Fixed Positioning**: allows for positioning elements relative to the viewport of the page, so it is in some sense detached from the other content on the screen. This is useful to keep some parts of the webpage (navigation bar, "move to top" button, etc) visible at all times.

There is also a fifth positioning type, **Sticky Positioning**, which is a mix of **relative** and **fixed** positioning, allowing for convent to be positioned on some part of the document until a threshold is reached, which makes the element display in a fixed position on the viewport.

Question 2 Page Layout

We need to develop a structure with a parent element (`.row`) that will act as a row and twelve child elements (`.col`) that act as columns. Characteristics:

- Four columns per row.
- Even columns must be twice as wide as the odd ones.
- The space between columns has to be `25px`.
- The space between rows has to be `15px`.
- The rows have to have a minimum height of `150px` and an automatic maximum (this height will vary depending on its content).

Write the necessary *CSS* rules for the parent and the child elements.

Answer 2

Here is the *CSS* code to implement the table structure, inside *HTML* code for reference:

```
<!DOCTYPE html>
<html>
<body style="margin: 0;">
<style>
.col
{
    border: 1px solid blue;
    background-color: lightblue;
    border-radius: 5px;
```

```
    text-align: center;
}
.row
{
    display: grid;
    grid-template-columns: 1fr 2fr 1fr 2fr;
    grid-auto-rows: minmax(150px, auto);
    row-gap: 15px;
    column-gap: 25px;

}
</style>
<div class="row">
    <div class="col">col1</div>
    <div class="col">col2</div>
    <div class="col">col3</div>
    <div class="col">col4</div>
    <div class="col">col5</div>
    <div class="col">col6</div>
    <div class="col">col7</div>
    <div class="col">col8</div>
    <div class="col">col9</div>
    <div class="col">col10</div>
    <div class="col">col11</div>
    <div class="col">col12</div>
    <div class="col">col13</div>
</div>
</body>
</html>
```

Question 3 Responsive Design

Question 3.1

Explain, in your own words, the advantages that we can get by developing a site using the principles of responsive design, compared to one developed with a fixed width.

Answer 3.1 Modern devices come in all shapes and sizes. From ultra-wide monitors to the smallest smartwatches, all tech that can browse the web should be able to enjoy a coherent experience. Responsive design is not only crucial to make content easily enjoyable on every type of screen, but also allows for making the whole experience much more coherent. A small device visiting a site may need a more vertical interface with some parts of the site squeezed inside a hamburger menu. Similarly, devices with wider screens should have things laid out differently. Namely, paragraphs need not stretch the whole width, but should rather split into different columns. Lastly, responsive design helps adapt the web interface depending on the size of the browser window, changing how content is displayed and resizing elements when necessary.

Fixed width sites are harder to view on certain displays, as content will often overflow into vertical and horizontal scrollbars, as well as content will bunch up in a rather meaningless way, making navigation on the site a real hustle.

Question 3.2

Write the necessary style rules so that they are applied correctly in the following cases:

- On screens with at least `660` pixels of width, all document headings must have a font size of `110%` with respect to the value specified in their parent, be centered and have `10%` padding to the left and right.
- On screens with a width of at least `769` pixels and a maximum of `1023` pixels, class `2` headers must have a font size `130%` of the value specified in their parent, and class `3` through `6` headers must have a font size `120%` of the value specified in their parent.

- On screens with at least `1024` pixels of width, all document headings must be aligned to their default value and have no left and right padding.

Answer 3.2

- Case 1 Code

```
@media screen and (min-width: 660px)
{
    h1, h2, h3, h4, h5, h6
    {
        font-size: 110%;
        text-align: center;
        padding: auto 10% auto 10%;
    }
}
```

- Case 2 Code

```
@media screen and (min-width: 769px) and (max-width: 1023px)
{
    h2
    {
        font-size: 130%;
    }
    h3, h4, h5, h6
    {
        font-size: 120%;
    }
}
```

- Case 3 Code

```
@media screen and (min-width: 1024px)
{
    h1, h2, h3, h4, h5, h6
    {
        text-align: initial;
        padding: auto 0 auto 0;
    }
}
```

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?

Here are the differnent *HTML* entities I have used:

HTML entity	*UTF-8* character
`>`	>
`&`	&
`&eur;`	€
`&reg;`	®

In general it was only necessary to use the `>` enitity in order to escape the `>` character, to prevent *HTML* from treating it as the closing part of an element. Other characters I used entities for are generally some special symbols,

in case the corresponding *UTF-8* unicode characters cannot be displayed on the editor some people may be using. They are also very handy if your keyboard has no way of typing a certain character

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:

- Fonts and body properties

```
body
{
    width: 100%;
    margin: 0 auto 0 auto;
    font-family: 'Mulish';
}

h1, .result-link, .sidebar a, .price, .pager-n, .signup-section a
{
    font-family: 'Open Sans';
    font-stretch: condensed;
}
```

We set all elements to use `100%` of the available width (parent container), and set the font family as defined in the document of this exercise. All text will be rendered using `Mulish` fonts, except for the main title, result text, sidebar text, price text, pager and sigunp button text, which will rather use `Open Sans Condensed` fonts.

- Header

```
header
{
    background-color: #F3702A;
    position: relative;
    height: 100px;
}

.header-button
{
    position: absolute;
    top: 50%;
    right: 5%;
    transform: translateY(-50%);
}
```

The top header has an orange background, color, an image defined in *HTML* and a set of buttons from the class `header-button`. I right-align these buttons and center them vertically using absolute positioning with respect to the header.

- Navigation

```
nav
{
    position: relative;
    height: 40px;
}

nav, .category
{
    background-color: #EAEBE9;
}

nav ul
```

```
{
    height: 100%;
    padding-left: 10px;
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
}

nav li
{
    padding-left: 20px;
    display: inline-block;
}

nav a
{
    display: block;
    line-height: 40px;
    font-weight: bold;
}

nav a, .sidebar a
{
    font-size: 22px;
}

nav a:hover, nav a:active, nav a:focus,
.sidebar a, .sidebar a:hover::after, .sidebar a:focus::after, .sidebar a:active::after,
.orange,
.result-link,
.footer-phone
{
    color: #F3702A;
}
```

We set the background color of the navigation area, which is made of a list of elements with links to navigate the pages (though they all link to `#`). We display the `li` elements as `inline-block` to remove the bullet points and have the elements display horizontally. Then, every `a` element is displayed as a `block` to make sure that the whole containing block of the link is activable, rather than just the link text being activable. Again, we center the nav bar vertically inside its container using absolute positioning.

- Main Heading

```
.result-header
{
    margin-top: 10px;
    margin-bottom: 10px;
    padding-left: 15px;
    font-weight: normal;
}

.result-header span
{
    padding-left: 10px;
    font-size: 22px;
}
```

This corresponds to the `h1` styling. It essentially sets the font sizes to match those from the sample images. The

`span` is merely used to decrease the font size and add an extra gap before the text `(326 Products)`

- Page container and width

```
.page
{
    display: grid;
    grid-template-columns: 1fr 4fr;
}

#main, h1, .category, table, .footer-contact
{
    min-width: 800px;
    max-width: 1170px;
    margin: 20px auto 20px auto;
}
```

These second rule is simple, just defines the max width of some elements, leaving other elements to occupy `100%` of the viewport width, as well as some margins on the left and on the right. The `page` class defines a `div` containing the search results on the right and the sidebar on the left. We use the ratio `1fr 4fr` to ensure that most space is used by the pen search results.

- Result Card

```
.pen-result, .big-result
{
    height: auto;
    overflow: hidden;
    border-radius: 10px;
    border: 1px solid #736D75;
    display: grid;
    grid-template-columns: 0.8fr 2fr 1fr;
}

.pen-result:not(:last-child)
{
    margin-bottom: 30px;
}

.big-result
{
    grid-template-columns: 1.1fr 2fr 1.1fr;
    border: none;
}
```

The search results from `result-list.html` look like cards with a border and a small gap between them. Elements in the class `pen-result` are set to look this way. They have a grey border with rounded corners, and the content inside them is arranged on a grid with three columns: one for the image of the pen, another for its name and description, and another one for the pricing of said pen.

On the second page, the displayed card is almost the same, but has no border and the columns do not have the same proportions. Changing the `grid-template-columns` and setting the `border` to `none` easily solves this.

- Result Card Image

```
.result-image
{
    margin: 15px;
    position: relative;
}
```

```
.result-image img
{
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}
```

The image container on the left of the card will have some margins to ensure that the rounded corners do not overlap with the image corners, and the image itself will be centered inside that `result-image` class (`div`) with absolute positioning.

- Result Card Body

```
.result-body
{
    margin: 10px 5px 10px 5px;
}

.result-link
{
    font-weight: bold;
    font-size: 23px;
    display: block;
}

.result-features, .result-body p:not(:first-child)
{
    margin: 10px auto 10px auto;
}

.result-body p:first-child
{
    font-weight: bold;
    font-size: 20px;
}

.result-features
{
    list-style-type: none;
}

.result-features li::before
{
    content: "\00B7";
    padding-right: 5px;
}
```

The `result-body` class includes a link to the pen itself (identified with class `result-link`), with larger orange font. Every result (even the card in the page `single-product.html`) has a list of features (identified with class `result-features`). We changed the bullet icon to one matching the image we were given. Lastly, the first `p` inside the `result-body` will have larger font and `bold` for font weight, corresponding to the `Description` text on the second page before the rest of the body.

- Result Card Price

```
.result-price
{
    padding-left: 20px;
```

```
    padding-right: 20px;
    position: relative;
    color: white;
    background-color: #736D75;
}

.pen-result .result-cart
{
    position: absolute;
    right: 5%;
    bottom: 5%;
}

.pen-result .price
{
    position: absolute;
    top: 5%;
    right: 5%;
}

.big-result .result-price
{
    display: flex;
    justify-content: center;
    flex-direction: column;
    gap: 20px;
}

.big-result .result-action
{
    display: grid;
    text-align: right;
    grid-template-columns: 3fr 1fr;
    line-height: 60px;
    font-weight: bold;
    font-size: 20px;
}

.price
{
    margin: 5px;
    font-size: 60px;
    font-weight: bold;
}

.big-result .price::after
{
    content: "Unit";
    padding-left: 10px;
    color: white;
    font-size: 16px;
    font-family: Mulish;
    font-weight: normal;
}
```

This section corresponds to the third and last column of the grid in our result cards (remember that this also applies to the single card on the second page). The `big-result` refers to the *bigger* card found in `single-product.html`,

and `pen-result` refers to the other cards from `product-list.html`.

The result cards from the first page are generally smaller in size, so this section is easier to organize with just absolute positioning. We start by setting the background color to gray, and then position two elements on this area. We position the price with the `price` class selector at the top right, and then the cart icon at the bottom right.

The larger single result card is laid out completely different, as it has more items inside it. First, the whole area is made of flex items in vertical layout. This includes the price (`price`), the text underneath it, the horizontal separator (`hl`), and the bottom actions (`result-action`). There is not much to say about the price, text and separator, so let's focus on `result-action`. This section contains the heart and cart icons and the text beside them. It is all organized in a small `2x2` grid, but slightly aligned to the right.

- Sidebar

```
.sidebar
{
    margin-right: 20px;
}

.sidebar li
{
    position: relative;
    list-style-type: none;
    border-bottom: 1px dotted black;
}

.sidebar li:first-child
{
    border-top: 1px dotted black;
}

.sidebar a::after
{
    position: absolute;
    right: 5%;
    content: "\271A";
    font-size: inherit;
    font-weight: inherit;
}

.sidebar a
{
    display: block;
    line-height: 40px;
    padding-left: 5px;
    font-weight: bold;
}
```

The elements on the left side are elements of a list in *HTML*, and they generally do not have too many bells and whistles. The hardest was to add the `+` icon after every element of the list. I ended up using the `::after` pseudo-element to append that icon after every `li` element in the sidebar. The styles and colors of the links are defined later in the *CSS* code.

- Pager

```
.pager
{
    list-style-type: none;
    margin-top: 50px;
    margin-bottom: 50px;
```

```
    text-align: center;
    display: flex;
    justify-content: center;
    align-items: center;
}

.pager a
{
    line-height: 60px;
    display: block;
}

.pager-n
{
    width: 60px;
    font-size: 18px;
    font-weight: bold;
}
```

The pager section contains a set of buttons laid out in a list similar to the navigation bar. It uses a flex layout to better adapt the *UI* and to allow for easier centering of the numbers on the pager. The left and right buttons are images from sprites, defined later in the code.

- Specs Table

```
table
{
    border-collapse: collapse;
    table-layout: fixed;
    border: 2px solid #EAEBE9;
    text-align: center;
}

thead
{
    border: 2px solid #EAEBE9;
    background-color: #EAEBE9;
}

th
{
    text-align: left;
    padding: 5px;
}

td
{
    vertical-align: top;
    padding: 5px 5px 5px 10px;
    width: 25%;
}

td:not(:last-child)
{
    border-right: 1px dashed black;
}

tr:not(:last-child)
```

```
{
    border-bottom: 1px dashed black;
}
```

The table has several styling rules. Firstly, there are no gaps between cells, so we set the `border-collapse` accordingly. Then we set the border color and the background color for the head of the table. Every cell (`td`) has some padding, and the text is aligned to the left. Also, every cell will take up `25%` of the table width, so that in the end we have `4` columns and `2 rows`.

- Sign Up Section with Static background image

```
.signup-section
{
    height: 250px;
    position: relative;
    color: white;
    font-weight: bold;
}

.signup-section::before
{
    content: "";
    width: 100%;
    background-image: url(../img/Black-Twist-Pen-on-Notebook-pexels-mohammad-danish.jpg);
    background-attachment: fixed;
    background-size: 100%;
    background-position: bottom;
    position: absolute;
    filter: contrast(50%) brightness(60%);
    height: 250px;
}

.signup-section p
{
    font-size: 30px;
    position: relative;
    text-align: center;
    top: 45%;
    left: 50%;
    transform: translate(-50%, -45%);
}

.signup-section a
{
    color: white;
    font-size: 18px;
    line-height: 35px;
    padding-left: 10px;
    padding-right: 10px;
    position: absolute;
    top: 70%;
    left: 50%;
    transform: translate(-50%, -70%);
    border-radius: 5px;
    background-color: #F3702A;
}

.signup-section a:hover, .signup-section a:active
```

```
{
    background-color: #EAEBE9;
    color: black;
}

.signup-section a:focus
{
    color: white;
    background-color: black;
}
```

The section uses a static background image. This image was placed in the `::before` pseudo-element, because otherwise the filtering options applied to the background image would also apply to the other children in the section (text, signup button). This background image is set to static thanks to `background-attachment: fixed;`. Then the text and signup button are both placed on top with absolute positioning.

- Contact Information

```
.footer-contact
{
    width: 60%;
    align-items: center;
    display: grid;
    grid-template-columns: 1fr 1fr 0 2fr;
    gap: 10px;
}

.footer-social, .footer-info, .footer-contact img
{
    padding-top: 20px;
    padding-bottom: 20px;
}

.footer-info
{
    padding-left: 20px;
}

.footer-title
{
    font-weight: bold;
}

.footer-phone
{
    font-size: 23px;
}

.footer-social
{
    text-align: center;
}

.footer-social li
{
    display: inline-block;
}
```

```
.footer-social li:not(:first-child)
{
    padding-left: 15px;
}

.vl
{
    height: 100%;
    border-left: 1px solid #EAEBE9;
}
```

This section of the page is another grid, which has for columns an image, some text, a vertical separator and a list of social network links with icons.

- Footer

```
.footer-policy
{
    background-color: #272F2E;
    height: 60px;
    display: flex;
    justify-content: center;
}

.footer-policy li
{
    text-align: center;
    height: 100%;
    display: inline-block;
}

.footer-policy li:not(:first-child) a:before
{
    content: "\2219";
    padding-right: 30px;
    color: white;
}

.footer-policy a
{
    line-height: 60px;
    display: block;
    color: white;
}

.footer-policy li:not(:last-child) a
{
    padding-right: 30px;
}
```

The footer has a dark background color, covers the whole width, and has some flex elements with links to various page 'policies'. These links work similarly to those from the navigation bar and the pager.

- Sprites

```
.header-button-login, .header-button-about, .header-button-cart,
.pager-prev, .pager-next, .result-cart, .result-heart,
.footer-facebook, .footer-linkedin, .footer-twitter, .footer-instagram
{
```

```
    display: inline-block;
    background-image: url(../img/sprite.png);
    height: 60px;
    width: 60px;
    background-size: 660px;
    background-repeat: no-repeat;
}

a:hover, a:focus, a:active
{
    outline: none;
    background-position-y: -60px;
}

.footer-facebook
{
    background-position: 0 0;
}

.footer-linkedin
{
    background-position: -60px 0;
}

.footer-twitter
{
    background-position: -120px 0;
}

.footer-instagram
{
    background-position: -180px 0;
}

.header-button-login
{
    background-position: -240px 0;
}

.header-button-about
{
    background-position: -300px 0;
}

.header-button-cart
{
    background-position: -360px 0;
}

.result-cart
{
    background-position: -420px 0;
}

.result-heart
{
```

```
    background-position: -480px 0;
}

.pager-prev
{
    background-position: -540px 0;
}

.pager-next
{
    background-position: -600px 0;
}
```

All icons are stored in a single image file, rather than having one image per icon. This helps reduce the bandwidth needed to load a page since a spritesheet uses less space than a lot of smaller images. Thus, for every element that has one of these icons will use this spritesheet as the background image. Every button displays as a `60x60` square, but our spritesheet is way larger than that. We observe that there are `11` sprites per row and that the top row corresponds to normal icons whereas the second one corresponds to sprites in the hover state.

Knowing all this, we set the width of our spritesheet to `(11 sprites) x (60px per sprite) = 660px`, and also set all sprites to display with a height and width of `60px`. Then all we have to do to display a sprite is set the proper `background-position` for each and every icon, using multiples of `60`.

Lastly, to update images in the hover state, all we have to do is use the same `background-position` for the `x` and change the `background-position-y` to `-60px` (the second row).