# **Sass Essentials**

# Responsive design principles

Responsive design makes websites usable on all kinds of devices. There are 4 principles:

## Fluid layouts

Use **%** or **vh** or **vw** units instead of px for elements that should adapt to viewport (usually layout). Use **max-width** instead of width.

## Responsive units

use **rem** unit instead of px for most lengths.

## Flexible images

Always use **%** for image dimensions, together with the **max-width** property.

## Media queries

To change CSS styles on certain viewport widths (breakpoints). Media queries alone are not enough. We really need to start creating a fluid layout right from the beginning.

**NOTE | Remember that responsive web design does not works if this meta tag is not included in the HTML file:**

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

# Advanced CSS properties

## background-position

To position a background image so that a specific part of the image will always be visible.

Background-position: top | bottom | x% y%

## clip-path

To clip a part of an element so as to give it a shape other than regular rectangular shapes. Can be used with functions like polygon, circle, ellipse, etc.

* polygon: requires the coordinates of multiple points in reference to the starting point of the element.

clip-path: polygon(0 0, 100% 0, 100% 70%, 0 100%);

* circle: this function requires the radius and then the position of its center point.

clip-path: circle(50% at 50% 50%)

## animation

To perform an animation defined by the @keyframes directive. This is a shorthand property for (in order):

* animation-name
* animation-duration
* animation-timing-function
* animation-iteration-count
* animation-delay: usually used with **animation-fill-mode** set to backwards to make the element appear on the page only when the animation is executed, and not beforehand.
* the ‘animation’ shorthand property can accept ‘infinite’ as the third input to perform the animation in an infinite loop.

## background-clip

Used with webkit prefix usually combined with other properties to give text elements a linear gradient color. It should be set to ‘text’ in this case. Refer to advanced CSS practices.

## outline and outline-offset

outline: 1.5rem solid $color-primary

outline-offset: 2rem;

## transition

The transition property can accept multiple properties at the same time. For each in the transition command, we can set a duration time, then a timing function and then a delay time.

Transition: transform .2s, width .4s <timing-fn> .2s

**NOTE | the transition property does not apply on the background-image property, including any linear or radial gradient.**

### Transition timing functions

Used to create custom timing curve for transitions to execute. Consider using the presets presented by these two websites:

* <https://cubic-bezier.com/>
* [Easing Functions Cheat Sheet (easings.net)](https://easings.net/)

## Transform

The transform property can accept multiple transforming functionalities.

transform: skewY(-7deg) skewX(15deg);

**NOTE | the transform function can do multiple transform functionalities of different types at the same time.**

### transform-origin

Used to specify the geometric origin where the transform property should be executed in reference to it.

transform-origin: right;

## backface-visibility

Usually used to hide the rear side of an element when attempting to rotate it.

backface-visibility: hidden;

## perspective

Used to give perspective to an element’s dimensions.

perspective: 150rem;

## background-blend-mode

Used to blend a background image with a linear gradient on it. It has different modes, such as lighten, darken, screen, multiply, overlay, etc.

background-blend-mode: screen;

## Gradients

We can use multiple types of gradients, including linear and radial.

## box-decoration-break

Used to tell CSS to consider each line of a text element a separate element and then style them consistently. (-webkit-)

box-decoration-break: clone;

## shape-outside

Used to form the shape of an element. usually used to make a text wrap around the shape. Note that this shape will not be visible. Refer to advanced CSS practices. Similar to clip-path, this property receives a function like polygon, circle, etc. Note that for this property to work on an element, the element should be floated. (-webkit-)

shape-outside: circle(50% at 50% 50%);

## filter

Used on images to apply multiple filters. Accepts certain functions to implement filters, such as blur, brightness, etc.

filter: blur(3px) brightness(80%);

## object-fit

Usually set to ‘cover’ and used with height and width of a video element manually set to 100% in order to fix the coverage of the video playback over the whole background. It can also be set to ‘fill’ but it would then make the video lose its original aspect ratio.

object-fit: cover;

**NOTE | this is very similar to background-size set to cover for background images. Object-fit can also be used on images, especially when images are arranged in a CSS grid. In this case each image should be places inside a parent HTML element (usually figure) and the object-fit property of the image should be set to cover. This will only work if both width and height of the image element is manually set (usually 100%).**

## Text layout

### column-count, column-gap, and column-rule

Used usually together to arrange a text element in columns.

&\_\_text {

    column-count: 2;

    column-gap: 4rem;

    column-rule: 1px solid $color-grey-light-2;

  }

### hyphens

Used to enable hyphenation for a text element. To make this work properly, set the ‘language’ attribute of the html tag correctly.

 &\_\_text {

    hyphens: auto;

  }

## background-repeat

used to determine whether a background image is allowed to repeat itself throughout the whole are of the background or not. To prevent it from repeating:

background-repeat: no-repeat;

## fill: currentColor

a keyword that can be used for color properties, such as fill, and makes that specific property inherit the color property of the current element or its parent element.

## mask-image and mask-size

allows to see through the element on which this property is used. For instance, mask-image can be used on a ‘before’ pseudo-element with a background color, setting the URL address of an svg file as the mask. This should usually be used along with the mask-size property set to cover. (-webkit-)

&\_\_item::before {

    content: "";

    display: inline-block;

    height: 1.2rem;

    width: 1.2rem;

    background-color: var(--color-primary);

    -webkit-mask-image: url("../img/chevron-forward-outline.svg");

    -webkit-mask-size: cover;

  }

# Advanced CSS selectors

## Combining pseudo-elements and pseudo-classes

Consider this code example:

.btn::after {}

.btn:hover::after {}

The second line selects the ‘after’ pseudo-element when it is hovered.

## The not pseudo-class selector

To select all the elements with the same name except one of them, here as an example, the last child.

.row {

  &:not(:last-child) {}

}

## Root pseudo-class selector

Used to define CSS custom properties, also known as CSS variables.

:root {

  --color-primary: #eb2f64;

}

## Target pseudo-class selector

The element that an anchor element refers to through its href property is known as the target. This target element can be selected using the ‘target’ pseudo element.

&:target {}

So we can apply specific styles as soon as an element is targeted through a click on an anchor element.

## Attribute selector

For instance, to select an img element with its ‘alt’ attribute:

[alt=”logo”] {}

The ‘class’ is also another attribute of any HTML element. when attempting to select multiple elements with similar, but not the same, class names:

[class^=”col-”] {} // class name starts with

[class\*=”col-”] {} // class name contains

[class$=”col-”] {} // class name ends with

## Direct child selector

Typically, ‘>’ is used to select a direct child of an element. now to select all the children elements of a parent element:

.section-features {

  & > \* {}

}

This selector is usually used when we want to select more then one direct child without being concerned about the element type. Otherwise, if there was only one child, we could simply use the regular child selector in CSS.

## Direct and general sibling selector

Direct sibling selector: in this example, the label (sibling of the input) should be placed right after the input element in HTML.

.input + .label {}

General sibling selector: in this case, the label (sibling of the input) is not placed right after the input element, and there are other elements in between.

.input ~ .label {}

## Pseudo-class and pseudo-element selectors related to input elements

* To select a textbox input when it is focused, use the ‘focus’ pseudo-class

&\_\_input:focus {}

* To select the placeholder of a textbox input, use the ‘placeholder’ pseudo-element:

&\_\_input::placeholder {}

**NOTE | this selector may need to be implemented with a –webkit-prefix.**

&::-webkit-input-placeholder {}

* To select the invalid state of a textbox input element of email type, we use the ‘invalid’ pseudo-class along with the ‘focus’ pseudo-class:

&\_\_input:focus:invalid {}

* To implement an effect in which the input’s label is only shown as soon as the user types an input inside a textbox, use the ‘placeholder-shown’ pseudo-class. Note that the placeholder is a part of the input itself, while we want to implement an effect on the label of the input. The label is the adjacent sibling element of the input element. So use the direct (or adjacent) sibling selector in reference to the input element. For this selector to work, the label element should be placed after the input element in HTML. Note that placeholder-shown selector works only when the placeholder is shown, meaning that the user has not typed anything in the input.

&\_\_input:placeholder-shown + &\_\_label {}

* We cannot style radio buttons in CSS. However, we can use the ‘checked’ pseudo-class of the radio-button to style a custom-designed radio button. Remember that the custom-designed radio button is a span element inserted inside the label element of the radio button. We would have to hide the original radio button, but we know that since the input tag and the label tags are connected, once the label is clicked, the original radio button would become selected, although hidden.

.radio-input:checked + .radio-label .radio-btn {}

## Selecting marked text elements

Using the ‘selection’ pseudo-element, we can style any marked text on our page.

::selection {}

## last-of-type selector

similar to last-child, but it only selects the element of the same type which appears last in HTML.

.paragraph:last-of-type {}

# Advanced CSS directives

## @keyframes <animation name> {}

Used to implement animations by keyframes. Keyframes directive should not be written in any selector. It is an independent code block.

@keyframes moveInLeft {

  0% {

<css properties>

  }

  80% {

<css properties>

  }

  100% {

<css properties>

  }

}

To use the define animation we use related CSS properties:

.heading-primary-main {

  animation-name: moveInLeft;

  animation-duration: 1s;

  animation-timing-function: ease-out;

  animation-iteration-count: 3;

  animation-delay: 3s;

  animation-fill-mode: backwards;

}

## @include <mixin name> {}

Used to insert a block of properties of a mixin in the code block of a selector.

## @extend %<extend name> {}

Used to include a selector for a certain block of properties.

## @content

Used to introduce a placeholder for CSS properties in a mixin.

@mixin respond-phone {

  @media (max-width: 600px) {

    @content;

  }

}

When included in a selector, any properties used will be inserted in it:

html {

   @include respond-phone {

    font-size: 50%;

  }

}

## @if $<variable-name> == <value> {}

Used to check for a certain condition. Usable in mixins. For example:

@mixin respond($breakpoint) {

  @if $breakpoint == phone {

    @media (max-width: 37.5em) {

      @content;

    }

  }

}

# Advanced CSS relative units

## %

* Percentages are measured relative to their parent’s font-size, if used to specify font-size.
* Percentages are measured relative to their parent’s width, if used to specify lengths

## em

* Em is measured relative to their parent’s font-size if used to specify font-size
* Em is measured relative to the current font-size, if used to specify lengths

## rem

* Rem is always measured relative to the document’s root font-size

## vh and vw

* Vh and vw are percentage measurements of the viewport’s height and width

# Advanced CSS practices

## Converting px to rem and performing a more effective global reset

Using the inheritance and considering the principles of code maintenance we can do this:

\*,

\*::after,

\*::before {

  margin: 0;

  padding: 0;

  box-sizing: inherit;

}

body {

box-sizing: border-box;

}

## Changing the stacking contexts

The z-index property is usually used, but it may be necessary to consider the effect of these CSS properties on the stacking context:

* Opacity > values other than 1
* Transform
* Filter

Note that the z-index property only works on an elment if the element’s position property is defined. ‘relative’ can be used if no special positioning value is needed.

## Doing calculations in CSS using the CSS calc() function

Allows us to do mathematical calculations and mix units. Note that #{} is needed around Sass variables in case we want to use them in the CSS calc function.

.col-1-of-2 {

    width: calc((100% - #{$gutter-horizontal}) / 2);

}

## Applying gradient color to text

1. We should first apply a linear gradient to the text element’s background
2. Make the text element an inline-block element so that the background color will only occupy the width of the text
3. Set the text color property to **transparent**
4. Using a **webkit** prefix, we should set the background-clip property to text.

## Making text wrap around a circle-shaped element

A series of demands should be met:

* Define a height and width for the element that should be shaped
* Make the element float
* Create the form of the shape with shape-outside property (-webkit-)
* Make the shape actually appear on the page with clip-path. Refer to advanced CSS properties.
* clip property to text.

## Making a video element play in the background

1. Make the video cover the whole area of the section in the background by setting its height and width to 100%. However, the video will preserve its aspect ratio.
2. To fix the 100% covering problem we set the video’s object-fit property to cover.

&\_\_video {

    height: 100%;

    width: 100%;

    object-fit: cover;

  }

## Inserting multiple colors in a linear-gradient function

Note that ‘transparent’ is accepted by CSS as a valid color.

linear-gradient(

      105deg,

      rgba($color-white, 0.9) 0%,

      rgba($color-white, 0.9) 50%,

      transparent 50%

    ), url("../img/nat-10.jpg");

## Hiding a popup completely off the page

.popup {

opacity: 0;

visibility: visible;

}

## BEM class naming system

BEM stand for Block-Level-Element

.block {}

.block\_\_element {}

.block\_\_element—modifier {}

Example: the header on a webpage can be considered a block in the BEM system. It is a standalone component that is meaningful on its own. All the elements inside it should be named with the ‘header’ first. For example, the logo container inside the header should be named ‘header\_\_logo-box’. Also the logo inside this logo container should be names ‘header\_\_logo’ because with this logo with this specific size is only usable in the header. The text container in the header should also be called ‘header\_\_text-box’, but the heading text inside it may not be called relative to the header. This heading may be named independently because it should be usable throughout the whole webpage. So this heading text my be called ‘heading-primary’, and it can be considered a standalone and meaningful element on its own.

## Sass files architecture: the 7-1 pattern

Basically, it includes 7 folders of partial Sass files and 1 main Sass file. Partial file names start with an underscore. Here are the 7 folders:

* Base: this folder should contain
* \_animations.scss
* \_base.scss: this file will contain really low-level code such as resets and styles for the html and body tags.
* \_typography.scss
* \_utilities.scss
* Components: this folder will contain one file for each single component.
* Layout: components are held together by the layout of the page. So any declarations related to the layout should be stored here. For instance, the ‘\_header.scss’ file may be stored here, instead of the components folder. (This is probably subjective. It may also be true to consider it a component.)
* Pages: specific styles for specific pages of the website should be stored here. This folder will certainly include a ‘\_home.scss’ file.
* Themes: used in the case of designing a web app with different themes.
* Abstracts: This will contain any code that will not output any CSS.
* \_functions.scss
* \_mixins.scss
* \_variables.scss
* Vendors: where third-party CSS, like the CSS files of bootstrap, can be placed.

## Responsive image in HTML

### Art direction and density switching

This strategy contains both density switching and art direction. This would involve using a picture element in which a source element and an img element is inserted.

#### Density switching

serving a high/low-resolution version of the same image for high/low-resolution screens. Use ‘srcset’ attribute instead of ‘src’ for the img element. **Density descriptors** used after each source path inform the browser whether the specified source leads to a high/low-resolution image without the browser having to download both images.

<img

srcset="img/logo-green-1x.png 1x, img/logo-green-2x.png 2x"

/>

#### Art direction

We tell the browser to use a specific image on a specific screen width. Use the picture HTML element. In this example, we are telling the browser to choose between the two images specified in the source tag if max-width is 37.5em. If not, the browser will choose between the sources specified in the img tag.

<picture class="footer\_\_logo">

     <source

       srcset="

         img/logo-green-small-1x.png 1x,

         img/logo-green-small-2x.png 2x

            "

       media="(max-width: 37.5em)"

       />

       <img

         srcset="img/logo-green-1x.png 1x, img/logo-green-2x.png 2x"

         alt="Full logo"

        />

</picture>

### Density and resolution switching

Unlike art direction, where we force the browser to use a certain image according to a media query, in this approach we allow the browser to choose the best image for the current viewport and pixel density situation. After each source path in the srcset attribute, specify a **width descriptor**, which tell the browser the width of each image without the browser having to actually download the image. Also use the ‘sizes’ attribute to inform the browser about the approximate width of the image displayed at different media query screen widths. We should divide the displayed image width by the media query screen width (e.g. 171px/900px). At the end of the sizes attribute, we define the image’s default size. So the browser would know which image to display for a certain screen width and screen resolution.

<img

srcset="img/nat-1.jpg 300w, img/nat-1-large.jpg 1000w"

sizes="(max-width: 900px) 20vw, (max-width: 600px) 30vw, 300px"

src="img/nat-1-large.jpg"

/>

NOTE | remember to also include a src attribute defining a fallback for displaying the image in case the user is using an old browser that does not support these responsive image strategies.

## Responsive image in CSS

Use media queries to target screen resolution in addition to screen width. Note that 192dpi is the resolution of Apple high-resolution retina screen and it is usually used as a reference.

@media (min-resolution: 192dpi) and (min-width: 37.5em), (min-width: 125em) {

      url("../img/hero.jpg");

  }

**NOTE | while we always try to load high-resolution images for high-resolution screens, we must always keep an eye for small high-resolution screens, where we don’t need to provide very huge image qualities. That is why we used the ‘and’ to include an additional condition for serving a high-resolution image. So as a conclusion we can say that we only need to provide high-quality images in 2 cases:**

1. **High-resolution screens (2x) with a rather large size (minimum 600px usually).**
2. **Low-resolution screens (1x) with a very large size (minimum 2000px usually).**

## Browser support

To check (feature query) if the browser supports a feature and then let a series of CSS declarations apply.

@supports (<-webkit-property name>: <value>) or (<property name>: <value>) {}

## Media queries

### Order of handling media queries

1. Base
2. Typography
3. General layout and grid
4. Page layout
5. Components

### Limit media queries to screens

Note that it is a good practice to include the ‘screen’ condition in all media queries. This will make the queries only apply if the page is being rendered into a screen, and not being printed on a paper.

@if $breakpoint == tab-port {

    @media only screen and (max-width: 56.25em) {

      @content;

    }

  }

We need this condition anywhere that a condition is inserted into the @media directive.

To identify if the device on which the webpage is being rendered is a touch device or not, we can check the hovering ability of the device in the media query:

@media only screen and (max-width: 56.25em), only screen and (hover: none) {}

### Writing complex conditions for media queries

In order to include make a media query apply only if all conditions in a series of conditions are met, we can use the ‘and’ keyword.

@media (min-resolution: 192dpi) and (min-width: 600px) {}

In order to make a media query apply only if one of a series of conditions is met, we can separate conditions with commas.

@media (min-resolution: 192dpi) and (min-width: 600px), (min-width: 2000px) {}

### Writing media queries with mixins

Create a mixin which conditionally inserts a media query based on an if condition:

@mixin respond($breakpoint) {

  @if $breakpoint == phone {

    @media (max-width: 37.5em) {

      @content;

    }

  }

  @if $breakpoint == tab-port {

    @media (max-width: 56.25em) {

      @content;

    }

  }

}

Create a mixin for each media query using this syntax:

@mixin respond-phone {

  @media (max-width: 600px) {

    @content;

  }

}

Then we include this media query in any element that should react to it:

html {

   @include respond-phone {

    font-size: 50%;

  }

}

## Writing HTML entities in CSS

To insert HTML entities in our webpage using CSS we should use different syntaxes. A list on CSS tricks website provides all the codes you need in HTML and CSS to insert HTML entities.

# Advanced Flexbox tips

Here are a list of CSS properties that can be used on a flex container and flex items:

## Flex container

### Flex-direction

This determines the direction of the main axis of the flexbox. Values:

* Row
* Row-reverse
* Column
* Column-reverse

### Flex-wrap

This determines whether or not the flexbox items are allowed to wrap into a new line if there is not enough space in the first line of the flex container. Elements in the new line will behave according to their individual properties that defines their grow, shrink, and base. Values:

* Wrap
* Wrap-reverse

### Justify-content\*

This determines how flex items will be aligned along the main axis of the flex container. Values:

* Flex-start
* Flex-end
* Center
* Space-between
* Space-around
* Space-evenly

### Align-items\*

This determines how flex items will be justified along the cross axis of the flex container. This is mostly used along with the justify-content property. Values:

* Stretch
* Flex-start
* Flex-end
* Center
* Baseline

### Align-content

This only applies when there is more than 1 row of flex items in a flex container. In this case, align-items control how the rows are aligned along the cross axis of the flex container if there is empty space. Values:

* Stretch
* Flex-start
* Flex-end
* Center
* Space-between
* Space-around

## Flex items

### Align-self

Very similar to align-items, but for one flex item. Usually used along with a declaration of align-items, in order to align one single item differently. Values:

* Auto
* Stretch
* Flex-start
* Flex-end
* Center
* Baseline

### Order

Defines the order in which one flex item should appear inside the flex container. This property accepts an integer number.

### Flex grow, shrink, basis

These three properties together allows flexbox to determine the width of a flex item.

#### Flex-grow

Defines how much an item can grow if there is empty space available. Accepts 0 or 1.

#### Flex-shrink

Defines how much an item can shrink if there is a shortage of space. Accepts 0 or 1.

#### Flex-basis

Defines the element’s base width. Accepts either ‘auto’ or a length number, or even a percentage.

#### Flex

A shorthand property for the three properties mentioned above.

# Advanced CSS grid tips

## Using min-content, max-content, and minmax()

grid-template-columns: min-content 1fr 1fr 1fr

Using min-content makes the width/height of the column/row decrease as much as it is needed to embrace the longest unit of content (word if all texts) in different rows of a column, causing line breaks for cells with more content.

grid-template-columns: max-content 1fr 1fr 1fr

Using max-content makes the width/height of the column/row increase as much as it is needed to embrace the maximum content of all rows of one column with no line breaks.

Grid-template-rows: repeat(2,

minmax(150px, min-content)

);

The minmax function defines the minimum and maximum height/width of a row/column. Min-content and max-content keywords can be used in this function.

## Naming grid lines

This makes it a lot easier to arrange grid items especially in regard to changes applied in media queries for responsive design.

.container {

grid-template-colmuns: [line-name] 100px [line-name line-name-2] 200px; }

naming the grid lines created by the repeat function:

.container {

grid-template-columns: repeat(3, [line-start] 1fr [line-end]) 200px [grid-end];

}

Each line name will be followed by a number to prevent conflicting line names; line-start 1, line-end 1, line-start2, line-end 2, line-start 3, and so on.

## Setting the grid-auto-flow

note that we can use the grid-auto-flow property to determine whether the remaining grid items should appear as implicit rows or columns.

.container {

grid-auto-flow: column dense; }

the ‘dense’ keyword can be used when arranging grid items manually, creating empty grid areas unwantedly.

## Styling implicit grid rows or columns

grid-auto-rows: <value><unit> //works if we have implicit rows

grid-auto-columns: <value><unit> // works if we have implicit columns

## Aligning grid items and tracks

To align/justify grid items inside grid cells use:

align-items: <keyword>

justify-items: <keyword>

To align/justify grid tracks inside the grid container use:

align-content: <keyword>

justify-content: <keyword>

To align/justify only 1 grid item in its grid cell, use:

align-self: <keyword>

justify-self: <keyword>

**NOTE | this will overwrite the align-items and justify-items properties set on the grid container.**

## Auto-fit and auto-fill

grid-template-columns: repeat(auto-fill, 100px);

auto-fill will make the grid create as much columns as possible in the grid container. Even the empty columns will preserve their 100px of width.

grid-template-columns: repeat(auto-fit, 100px);

auto-fit will make the grid create as much columns as possible in the grid container, but the empty columns will collapse and have no width. However, they are still there.

Using the auto-fit keyword along with the minmax function, the grid will adapt itself to different screen sizes automatically.

.container {

Width: 90%;

Grid-template-rows: repeat(2, minmax(150px, min-content));

Grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));

}

## Building a photo gallery with grid

Images preserve their aspect ratio and therefore, they will not appear exactly between the grid lines. However, we can make an image display only the part of it between grid lines, so within a grid cell, and the rest of it would be cropped and invisible:

1. Place the image element inside another parent container, usually a figure element.

<figure class="gallery\_\_item gallery\_\_item--1">

        <img src="img/gal-1.jpeg" alt="Gallery image 1" class="gallery\_\_img" />

</figure>

1. Use object-fit property on the image element and set it to cover. For this property to work, we should define the image’s height and width, and we should also set the image element’s display property to block.

&\_\_img {

    width: 100%;

    height: 100%;

    object-fit: cover;

    display: block;

  }

# Sass

## Installing Sass

To check if node is already installed in the project:

node –v

1. Create package.json file:

npm init

1. Install Sass compiler: after this step the node\_modules folder is added to the project.

npm install node-sass --save-dev

Note that we usually don’t share the node\_modules folder. Anyone who gets access to the project file, should install all necessary packages using:

npm install

1. Write scripts in the package.json file and create the Sass folder and main.scss file. The script is supposed to watch the Sass files for any change and then trigger the compile.

“scripts”: {

"devserver": "live-server",

“watch:sass”: “node-sass sass/main.scss css/style.css -w”

"start": "npm-run-all --parallel devserver watch:sass",

}

First, remember to install npm-run-all package. We need it to tell the command line perform several steps one after another, or simultaneously.

npm install npm-run-all --save-dev

Inside the windows command prompt or visual studio code terminal we use:

npm run start

## Setting up Sass build process

First, for the auto-prefixing stage, we need to install the autoprefixer and the postcss-cli packages:

npm install autoprefixer --save-dev

npm install postcss-cli --save-dev

Then we add these scripts to the package.json file:

“scripts”: {

"compile:sass": "node-sass sass/main.scss css/style.comp.css",

"prefix:css": "postcss --use autoprefixer -b 'last 10 versions' css/style.comp.css -o css/style.prefix.css",

"compress:css": "node-sass css/style.prefix.css css/style.css --output-style compressed",

"build:css": "npm-run-all compile:sass prefix:css compress:css"

}

We now use in the command line:

npm run build:css

## Sass features

### Variables

To define Sass variables:

$color-primary: #f9ed69;

To use this variable:

nav {

background-color: $color-primary;

}

### Nesting

This is used to select an element inside another element without having to rewrite the parent’s selector over and over.

.nav {

li {

}

}

With nesting we can select pseudo-classes and pseudo-elements. Wherever we put ‘&’ the parent’s name is replaced.

.nav {

&::after {} // & = .nav

li {

&:first-child {} // & = li

}

}

### Operators

### Partials and imports

Partials should be named with an underscore at the beginning of their names. To import partials into the main.scss:

@import “base/base”

Note that we don’t need to use the underscore nor the file format in the import command. Sass will automatically understand it.

### Mixins

To write any reusable piece of CSS code. This is how we define a mixin:

@mixin style-link-text($color) {

text-decoration: none;

text-transform: uppercase;

color: $color;

}

To use the mixin inside any selector:

.btn-main:link, .btn-hot:link {

@include style-link-text($color-text-light);

}

### Functions

Similar to mixins, with the difference that they produce a value that can be used later. This is how we define a function:

@function divide($a, $b) {

@return $a/$b;

}

Then to use the function in any selector:

nav {

margin: divide(60, 2) \* 1px;

}

### Extends

To make different selectors inherit declarations that are common to all of them. It basically adds up selectors for one block of declarations. This is how we define an extend and note that we can also use mixins where we define an extend:

%btn-placeholder {

width: $width-button;

@include style-link-text($color-text-light);

}

To use this extend in any selector:

.btn-main {

&:link {

@extend %btn-placeholder;

}

}

### Control directives

Content may be added later.

## Sass built-in functions

### Color functions

* darken(<color>, <percentage value>)
* lighten(<color>, <percentage value>)

## Advanced Sass workflow

There are a couple of steps to follow.

### Always visually design the page first

This helps to exactly determine how each row and column on your webpage will behave as the amount of content, screen width and height change. This will, in turn, help you exactly determine the CSS grid properties in a way that the whole page layout would be responsive right from the beginning. Remember to at least create a brief sketch of what should be graphically implemented on the webpage.

### Implement CSS and Sass file structure

#### One final CSS file

This would be a single CSS file usually called ‘style.css’ to which the HTML file will be linked through a stylesheet link tag. This file will be placed inside a ‘css’ folder located in the root directory of the project, and will contain the compiled Sass code when the webpage is ready for production.

#### 7-1 Sass structure

Create a Sass file structure according to the 7-1 principle, where there is one ‘main.scss’ file that is only concerned with importing all the code in other 7 folders that contain Sass partial files. This main SCSS file will finally be compiled into the ‘style.css’ file, which is connected to our HTML file via a stylesheet link element.

### Implement build process

This is a process through which the Sass code will pass and be compiled, prefixed, compressed, and finally built stage by stage. Each stage would be defined by a script in the package json file, where a collective script, activated by the ‘npm-run-all’ package, will run all these stages one after another automatically.

### Global reset & font size

A global reset should be implemented for all elements, for all after and before pseudo-elements and for the body tag.

\*,

\*::after,

\*::before {

  margin: 0;

  padding: 0;

  box-sizing: inherit;

}

html {

  font-size: 62.5%;

}

body {

  box-sizing: border-box;

}

### Implement a responsive CSS grid

This usually involves using the grid-template-rows property first, in order to define the exact behavior of row heights in relation to the amount of content, screen width and height.

Then we try to determine the behavior of column widths. It is a usual practice to implement a grid with 8 or 12 or 16 predefined columns in order to be able to limit the width of some components, and also make some other components full-bleed.

# Advanced HTML techniques

## Semantic elements for images and their captions

HTML ‘figure’ element can include an ‘img’ element along with a ‘figcaption’ element for the image’s caption.

<figure>

<img />

<figcaption></figcaption>

</figure>

## Including a video element

HTML ‘video’ element is used with certain attributes that tells the element how to behave on the webpage. The ‘video’ element should include two ‘source’ elements defining two file formats for browser support. After the source elements a bare text element can be included that will be shown to the user in case their browsers don’t support any of the file formats. Note that we usually put the video element inside another div element.

<div class="bg-video">

<video class="bg-video\_\_content" autoplay muted loop>

<source src="img/video.mp4" type="video/mp4" />

<source src="img/video.webm" type="video/webm" />

Your browser does not support this content!

</video>

</div>

## Using radio buttons

HTML radio button input requires an ‘id’ attribute, just like all other types of inputs, and a ‘name’ attribute on all the radio button options. The name attribute will cause all options become deactivated once one option is selected.

<div class="form\_\_group">

<div class="form\_\_radio-group">

  <input

  type="radio"

  class="form\_\_radio-input"

  id="small"

  name="size"

/>

<label for="small" class="form\_\_radio-label">Small tour group</label>

</div>

<div class="form\_\_radio-group">

<input

type="radio"

  class="form\_\_radio-input"

  id="large"

  name="size"

/>

<label for="large" class="form\_\_radio-label"

       >Large tour group</label>

</div>

</div>

## Checkbox/Radio hack

The main idea is to implement a checkbox, but usually hidden, and have a label connected to that checkbox. Then when we click on the label, the checkbox would become checked. At this point we would be able to use the ‘checked’ pseudo-class on the checkbox input element to apply certain styles to another element.

## HTML form

A form element on the webpage usually starts with a ‘form’ element with an ‘action’ attribute specifying where the form input data should be sent to. In front-end design it would usually be just #.

### Inputs and labels

In order to establish a connection between input elements and their labels, we usually insert an ‘id’ attribute on the input tag, to which the ‘for’ attribute on the label tag will refer. Remember that pair of input and label elements are usually placed inside a form group div.

<input id=”<id>”>

<label for=”<id>”></label>

This makes it so that when we click on the label, the input element will become focused if it is a textbox, or it will become checked if it is a checkbox or a radio button.

**NOTE | The placeholder on input elements, specially textboxes, do not inherit the font family and color properties automatically. We would have to explicitly determine them for the input element and set them to ‘inherit’.**

### Form input element attributes

#### Type

This attribute defines the type of input element.

* Text: Setting the type attribute of an input element to text will create a textbox.
* Email: Setting the type attribute of an input element to email will create a textbox that expects a valid email address.
* Radio: Setting the type attribute of an input element to radio will create a radio button.

**NOTE | in order to make radio buttons connected to each other, so that only one of them can be selected at a time, we would have to insert a ‘name’ attribute on all radio input tags, of course with the same name.**

#### Required

This will force the user to input a value for the textbox.

#### Name

This attribute is usually used on radio input types and it makes it so that only one radio button can be selected at a time.

# Emmet notation

The Emmet notation helps us to write short syntaxes instead of having to write all characters of a regular HTML code.

## Element

The name of the element should be typed in a regular text format. If the element that we want to create is a div, we don’t need to write <div>, and we can just write its class name.

<element-name>

## Classes

To write classes we start the class name with a dot character right after the element’s name. if the element is a div, we can start with a dot at the beginning.

<element-name>.<class-name1>.<class-name2>

## Defining child elements

To write a child element for a parent element we start the child’s element name after a > sign.

Parent-element.class-name>child-element.class-name

## Attributes

Attributes can be written in brackets. Remember that each attribute name and value should be defined in a separate pair of brackets.

<element>.<class-name>[<attribute>=”<value>”][<attribute>=”<value>”]

## Element content

In order to define an element’s text content we put write the content in { }.

## Automatic numbering

To make Emmet write numbers automatically for multiple elements, we use the $ sign. This can be used in classes, in element contents, in attribute names and values, and nearly anywhere.

<element-name>.<class-name-$>

**NOTE | in case the automatic numbering is used to determine the ‘src’ attribute of an element, make sure the file names are in accordance with what you want to implement here.**

## Multiplier

To make Emmet repeat an element n times, we multiply the element by n.

(<element>.<class-name>)\*n

**NOTE | this is usually used with automatic numbering where the final HTML code will include multiple elements with automatically numbered properties.**

# Live server

We can install the live server package globally, since it will be used in all our projects.

npm install live-server –g

to run the live server on a front-end project, we should run ‘live-server’ command in the terminal. Remember that the terminal’s current directory should be the projects root directory where the ‘index.html’ file is located.

**NOTE | we can define a flag for the ‘live-server’ script in order to make it run our webpage in a specific browser. For instance, in case we want it to be opened in Firefox:**

“scripts”: {

“devserver”: “live-server –browser=firefox”,

}

# Windows command prompt

Note that these command may not work in the Visual Studio Code’s command terminal.

To see the content of the current folder we use

dir

To move to a specific folder from the current folder

cd <folder name>

to move back one folder from the current folder

cd ..

to create a new folder

mkdir <foldername>

to create a new file

type nul > <filename.extension>

to copy a file from the current folder to one folder up

copy <filename.extension> ..

to move a file from the current folder to one folder up

move <filename.extension> ..

to remove a file

del <filename.extension>

to remove a folder

rmdir <foldername>