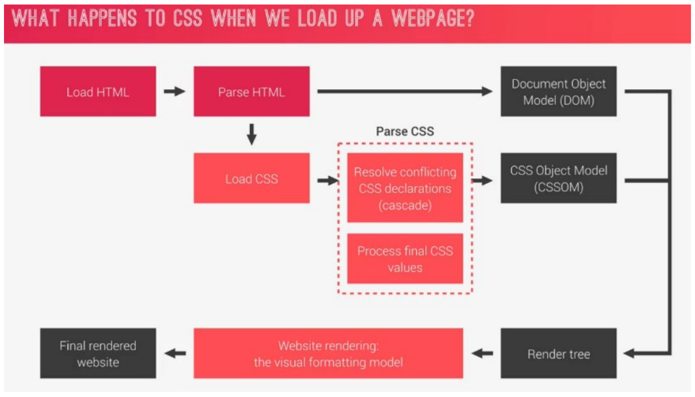
Advanced css and sass Theory and tricks

1. The best way to perform a basic reset using the universal selector
2. How to set project-wide font definitions
3. How to clip parts of elements using clip-path
4. The easiest way to center anything with the transform, top and left properties;
5. Alt description for images serve for SEO(Search engine optimization!!!!!!!!)
6. H1 is also important for SEO optimization. We can split the header with span and trick the SEO to pick more words for indexing the page.(☺)
7. How to create CSS animation using @keyframes and the animation property
8. What pseudo-elements and pseudo-classes are;
9. How and why to use the ::after pseudo-element;
10. How to create a creative hover animation effect using the transition property
11. **How CSS works?**
    Three Pillars of writing Good HTML and CSS: Responsive Design, Maintainable and scalable code, Web performance
12. Responsive design: -Fluid layouts, -Media queries, -Responsive design, -Correct units, -Desktop-first vs. Mobile-first.
13. Maintainable and scalable code: -Clean, -Easy-to-understand, -Growth, -Reusable, -How to organize files, -How to name classes, -How to structure HTML.
14. Web performance: -Less HTTP requests, -Less code, -Compress code, -Use a CSS preprocessor, -Less images, -Compress images.
15. How css works behing the scenes - overview: The browser loads the HTML. -It parses the HTML and build a DOM(Document Object Model) tree. -It also loads the css liked from the HTML and parse it(Resolve conflicting CSS declarations-cascade, Process final CSS values->for different screens and percentages). After the CSS file is parses it is build into CSS Object Model(CSSOM)(similar to DOM). After all the parsing is done the browser engine takes the processed filed and makes a render tree for Website rendering: the visual formatting model(box-model, floats, positioning). Finally the website is rendered to the screen.


WHAT HAPPENS TO CSS WHEN WE LOAD UP A WEBPAGE?
Load HTML
Parse HTML
Document Object Model (DOM)
Parse CSS
Load CSS
Resolve conflicting CSS declarations (cascade)
Process final CSS values
CSS Object Model (CSSOM)
Final rendered website
Website rendering: the visual formatting model
Render tree

16. How CSS is parsed: The cascade and specificity. Terminology: a css rule is build with: selectors to select the html element; Declaration block that contains declarations, which are formed by a property and the declared value.
17. Cascade is the process of combining different stylesheets and resolving conflicts between different CSS rules and declarations, when more than one rule applies to a certain element. Sources of css: author, user(changing in browser), and Browser(the default declaration ex. A Link)
18. How the cascade conflict is resolved? Importance > Specificity > Source Order
19. Importance: 1.User !important declarations; 2.Author !important declarations; 3.Author declarations; 4.User declarations; 5.Default browser declarations;
20. Specificity: 1.Inline styles; 2.IDs; 3.Classes, pseudo-classes, attribute; 4.Elements, pseudo-elements
21. Source order: The last declaration in the code will override all other declarations and will be applied.
22. CASCADE and SPECIFICITY rules:
    a. Css declaration marked with !important have the highest priority
    b. But, only use !important as a last resource. It's better to use correct specificityes – more maintainable code!
    c. Inline styles will always have priority over styles in external stylesheets;
    d. A selector what contains 1 ID is more specific than one with 1000 classes
    e. A selector that contains 1 class im more specific than one with 1000 elements
    f. The universal selector * has no specificity value (0,0,0,0)
    g. Rely more on specificity than on the order of selectors
    h. But rely on order when using 3rd-party stylesheets – always put your author stylesheet last(our file)
23. Even if we use hover or other pseudo-elements it may be a problem if the specificity is higher on other css rule.
24. How css is processed: Value processing: 1.Declared value(author declarations) -> 2.Cascaded value(after the cascade) -> 3.Specified value(defaulting, if there is not cascaded value -> 4.Computed value(converting relative values to absolute) -> 5.Used value(final calculation, based on layout) -> 6.Actual value(browser and device restrications)
25. The size for text is default 16px set by the browser. If there is not value specified for some declarations the value for them is 0(ex. Padding, margin…)
26. How units are converted from relative to absolute PX
27. CSS Value Processing rules:
    a. Each property has an initial value, used if nothing is declared (and if there is no inheritance

b. Browsers specify a root font-size for each page (usually 16px)
c. Percentages and relative values are always converted to pixels
d. Percentages are measured relative to their parent's font-size, if used to specify font-size;
e. Percentages are measured relative to their parent's witdth, if used to specify lengths
f. "Em" are measured relative to their parent font-size, if used to specify font-size;
g. "Em" are measured relative to the current font-size, if used to specify lengths
h. "rem" are always measured relative to the document's root font-size
i. "vh" and "vw" are simply percentage measurements of the viewport's height and width.

28. Inheritance in CSS:
    a. Inheritance passes the values for some specific properties from parents to children – more maiontainable code
    b. Properties related to text are inherited: font-family, font-size, color
    c. Properties like margin, padding, border are not inherited
    d. The computed value of a property is what gets inherited, NOT the declared value
    e. Inheritance of a property only works if no one declares a value for that property
    f. The inherit keyword forces inheritance on a certain property
    g. The initial keyword resets a property to its initial value
29. How CSS renders a website: The visual formatting model => Algorithm that calculates boxes and determines the layout of these boxes, for each element in the render tree, in order to determine the final layout of the page.
    a. Dimensions of boxes: the box model
    b. Box type: inline, block and inline-block
    c. Positioning scheme: floats and positioning
    d. Stacking contexts
    e. Other elements in the render tree
    f. Viewport size, dimensions of images….
30. **The box Model**: Content(text, images…); Padding(transparent area around the content, inside of the box); Border(goes around the padding and the content); Margin(space between boxes); Fill area(area that gets filles with background color or background image-doesn't contain the margin).
31. Total width = right border + right padding + specified width + left padding + left border

Total height = top border + top padding + specified height + bottom padding + bottom border

32. To solve this issue we use the box model with box-sizing: border-box =>total width = specified width and the total height = specified height
33. **Box types**: 1.inline(content is distributed in lines; Occupies only content's space; No line-breaks; No heights and widths; Paddings and margins only horizontal-left and right)->display: inline; 2.block-level(Elements formatted visually as blocks; 100% of parent's width; Vertically, one after another; Box-model applies as showed->display:block(display:flex, display:list-item, display:table); 3.inline-block(a mix of block and inline; Occupies only content's space; No line-breaks, Box model applies as showed) -> display:inline-block.
34. **Position schemes**: 1.normal flow(default positioning scheme; NOT floated; NOT absolutely positioned; Elements laid out according to their source order)->default position: relative; 2.absolute positioning(Element is removed from the normal flow; No impact on surrounding content or elements; We use top bottom, left and right to offset the element from tis relatively positioned container)-> position: absolute, position:fixed) and 3.floats(Element is removed from the normal flow; Text and inline elements will wrap around the floated element; The container will not adjust its height to the element(need to use clearfix)-> float: left, float:right;
35. **Stacking context**-> z-index, but there are other declarations that can influence the stacking context: opacity, transform, a filter, other properties might create a different stacking context
36. CSS architecture, components and BEM. Maintainable and scalable code: Clean, Modular, reusable, ready for growth.
37. Architect mindset: Think(Think about the layout of your webpage or web app before writing code); Build(Build your layout in HTML and CSS with a consisten structure for naming classes); Architect(Create a logical architecture for your CSS with files and folders)
38. Thinking about the layout: Component-driven design -> Modular building blocks that make up interfaces; -Held together by the layout of the page; -Re-usable across a project, and between different projects; Independent, allowing us to use them anywhere on the page;
39. Building with meaningful class names: BEM => Block Element Modifier; BLOCK: standalone component that is meaningful on its own; ELEMNT: part of a block that has no standalone meaning; MODIFIER: a different version of a block or an element. In code: .block{}(recipe) ; .block__element{}(info,title); .block__element—modifier{}(round)
40. Architecting with files and folders: the 7-1 pattern: 7 different folders for partial Sass files, and 1 main Sass file to import all other files into a compiled CSS

stylesheet. The 7 Folders: base/; components/; layout/; pages/; themes/; abstaracts/; vendors/.

41. How to use BEM method in practice
42. What is Sass and how does it work?
43. Sass is a CSS preprocessor, and extension of CSS that adds power and elegance to the basic language. Sass Source code ---Sass compiler---→Compiled css code.
44. Main Sass features:
    a. **Variables**: for reusable values such as colors, font-sizes, spacing etc.
    b. **Nesting**: to nest selectors inside of one another, allowing us to write less code
    c. **Operators**: for mathematical operation right inside of CSS
    d. **Partials and imports**: to write CSS in different files and importing them all into one single file;
    e. **Mixings**: to write reusable pieces of CSS code
    f. **Functions**: similar to mixins, with the difference that they produce a value that can than be used
    g. **Extends**: to make different selectors inherit declarations that are common to all of them
    h. **Control directives**: for writing complex code using conditionals and loops
45. We need to use clearfix when dealing with floats if the parent element hight collapses . //the after clears the float. We use it on the parent element
46. **Node.js** Allows developers to write and run JavaScript applications on the server. Developers started using node.js to also write tools to help them with local web development.
47. **NPM** is a simple command line interface that allows developers to install and manage packages on their local computers. There are all kinds of open source tools, libraries and frameworks needed for modern development. Modern web development could simply not exist without a package manager.
48. Using npm on the project: 1.npm init; npm install node-sass –save-dev(specify that we use the tool for developing-this is based on every library); if we move the project we can use "npm install" for intalling all the dependencies; npm uninstall jquery –save for uninstall the library
49. For compiling the scss file we need to define a script: "compile:sass": "node-sass sass/main.scss css/style.css". The browser doesn't know that we use sass because it will use css.
50. Live-server installed globally : npm install live-server -g
51. Basic principles for responsive design: 1.**Fluid grids and layouts**(To allow content to easily adapt to the current viewport width used to browse the website.

Uses % rather than px for all layout-related lengths; 2. **Flexible/Responsive Images**(Images behave differently than text content, and so we need to ensure that they also adapt nicely to the current viewport); 3.**Media Queries**(To change styles on certain viewport widths(breakpoints), allowing us to create different version of our website for different widths).

52. Layout types: Float layouts, Flexbox, CSS Grid.
53. Thing to learn: -thinking about components; -How and why to use utility classes; -How to use the background-clip property; -how to transform multiple properties simultaneously; -How to use the outline-offset property together with outline; -how to style elements that are NOT hovered while others are;
54. Using the Emmet for generating the HTML faster: ".text"-> creating a div with the class text; "section.section-about" creates a section with the class name section-about;
55. Can use outline instead of border and we can set an outline-offset to that
56. Feature section: How to include and use an icon font; Another way of creating the "skewed section" design; How and when to use the direct child selector;
57. Tours section: How to build an amazing, rotating card; -How to use perspective in CSS; 0How to use the backface-visibility property; -Using background blend modes; -How and when to use box-decoration-break.
58. //if the text is split on 2 lines we can interpret each of them as two separate elements and we can apply the decorations on each of them -> box-decoration-break: clone;
59. Stories section: How to make text flow around shapes with shape-outside and float; -how to apply a filter to images; -hot to create a background video covering an entire section; -how to use the <video> HTML element; -how and when to use the object-fit property(this is to cover the parent with the html element – in my case with a background video)
60. Booking section: -How to implement "solid-color gradients"; -How the generatlr and adjacent sibling celectors work and why we need them; -How to use the::input-placeholder pseudo-element; -how and when to use the :focus, :invalid, placeholder-shown and :checked pseudo-classes; -Techniques to build custom radio buttons;
61. Navigation part: What the "checkbox hack" is and how it works; -How to create custom animation timing functions using cubic Bezier curves; -How to animate "solid-color gradients"; -How and why to use transform-origin; -In generatl: create an amazingly creative effect;
62. Building a popup with only CSS: -How to build a nice popup with only CSS ; -How to use the :target pseudo-class; -how to create boxes with equal height using

display: table-cell; -how to create CSS text columns; -How to automatically hyphenate words using hypens

63. **Responsive design**
64. Mobile-first VS Desktop-first and breakpoints.
65. Desktop fist: Start writing CSS for the desktop: large screen; - Then, media queries shrink design to smaller screens.
66. Mobile first: Start writing CSS for mobile devices: small screen; Then, media queries expand design to a large desktop screen; -Forces us to reduce websites and apps to the absolute essentials.
67. Media queries don't add any importance or specificity to selectors, so code order matters – media queries at the end.
68. Is Mobile-first right for you? **PROS**: -100% optimized for the mobile experience; -Reduces websites and apps to the absolute essentials; -Results in smaller, faster and more efficient products; -Prioritizes content over aesthetic design, which may be desirable. **CONS**: -The desktop version might feel overly empty and simplistic; -More difficult and counterintuitive to develop; -Less creative freedom, makin ti more difficult to create distinctive products; -Clients are used to see a desktop version of the site as a prototype; -Do your users even use the mobile internet? What's the purpose of your website?
69. No matter what you do, always keeps both desktop and mobile in mind.
70. Selecting the breakpoints: The options: Bad(devices screens-like iphone screens), Good(group different devices-mobiles, tablets, laptops), Perfect(content into your design, design for some screen and when the content breaks define a new break point).
71. How to use a powerful Sass mixing to write all our media queries; -How to use the @content and @if Sass directives; -Taking advantage of Chrome DevTools for responsive design.
72. **Responsive Images**: The goal of responsive images is to serve the right image to the right screen size and device, in order to avoid downloading unnecessary large images on smaller screens.
73. 3 Uses cases: Resolution switching(decrease image resolution on smaller screen), Density switching-special case of resolution switching but the screen size doesn't matter(double the resolution on the photo), Art direction(other image, remodeled)
74. Responsive images in HTML – how to use the srcset attribute on the <img> and <source> elements, together with density descriptions; -how and why to use the <picture> element for art direction; -how to write media queries in HTML.-How to allow the browser to decide the best image to download, using the srcset attribute, width descriptors, and the sizes attribute of the <img> element.

75. Responsive images in CSS: -how to implement responsive images in CSS; -How to use resolution media queries to target high-resolution screens with 2x; How to combine multiple conditions in media queries.
76. Browser support: Always check caniuse.com before using a Modern css property in production; -use graceful degradation with @supports. -How to use @supports feature queries; -Implement graceful degradation on selected properties; -How to use backdrop-filter(sepia, brightness, blur… applied on the background elements).
77. Steps for building the version for deploying: compilation, concatenation, prefixing, compressing.
78. Install npm concat "npm intall concat –save-dev for concatenating all the css files. npm intall autoprefixer --save-dev, npm install postcss-cli, npm install postcss-cli –save-dev (npm i postcss@8.1.0 is working). Now in package json I have the developing files and the deployment files build separately.
79. **Flexbox**: Flexbox is a new module in CSS 3 that makes it easy to align elements to one another, in different directions ans orders; -The main idea behind flexbox is to give the container the ability to expand and to shrink elements to best use all the available space; -Flexbox replaces float layouts, using less, and more readable and logical code; Flexbox completely changes the way that we build one-dimensional layouts; A true revolution in CSS.
80. Flex box terms: flex container, flex items, main axis, cross axis.
81. Flexbox-header section: Why to use SVG icons vs. font icons; -How to find, generate and use SVG sprites in HTML; -how to change the color of an SVG icon in CSS; -How to use more advanced flexbox alignment techniques, including justify-content, align-items, align-self and flex.
82. Navigation section: How to use scaleY and multiple transition properties with different setting, to create a creative hove effect; -how and why to use the currentColor CSS variable; -How to use some more advanced flexbox alignment techniques, including flex-direction, justify-content and align-items.
83. Hotel-overview part: -How to create an infinite animation; How to use margin; auto with flexbox, and why it's so powerful; -continue to use flexbox properties for easy positioning and alignment;
84. this is a trick to not use all the space when hover or something, and the margin is growing like it was set like flex:1 => margin-right: auto;
85. Hotel description part: Continue to use flexbox, including flex-wrap to build a multi-column list; - How and why to use CSS masks with mask-image and mask-size.
86. The media queries must be declared from top to bottom when using the max-width, so that the correct overwriting rules to apply
87. **CSS Grid** (a whole new mindset):

a. CSS grid Layout is a brand new module that brings a two-dimensional grid system to CSS for the first time;
b. CSS Grid replaces float lyouts, using less, and more readable and logical CSS and HTML
c. CSS Grid works perfectly together with Flexbox, which is best to handle one-dimensional components and layouts;
d. CSS Grid completely changes the way that we envision and build two-dimensional layouts(we can get rid of bootstrap)

88. CSS grid terminology: display: grid(display:grid-inline). We have 2 axis : row axis and column axis and we can't change that. The grid is build by a grid container and it contains multiple grid items. Grid lines and gutter(between the grid columns). Grid track(row) and Grid track(column) . The area between 2 horizontal and vertical lines is called the Grid area. If the area is between the adjiacent 2 column and 2 row lines that the area is called Grid cell.
89. **Emmet** command: .container>.item.item--$*6, .feature{feature $}*6, (figure.gallery__item.gallery__item--$>img.gallery__img[src="img/gal-$.jpeg"][alt="Gallery image $"])*13
90. A fractional unit represents a unit that occupy all the available space
91. Grid basics: creating a grid, "fr" Unit, positioning grid Items, spanning grid items, naming grid lines(this is how it should be done by experienced devs), implicit grids vs. explicit grids, aligning grids, aligning tracks.
92. The explicit grid is the grid we define, and the cell we don't define is called the implicit grid.
93. You can style the grid even if you don't know exactly how many items will be with the implicit grid properties: //can set the height for the rows for the implicit grid
 grid-auto-rows:80px;
//automatically added as rows
 grid-auto-flow: column;
grid-auto-columns: .5fr;
94. Using min-content, max-content and the minmax() function.
95. we can use max-content instead of units(fit the content length of words). We can use min-content to wrap the words.
Minmax function: grid-template-rows: repeat(2, minmax(150px,min-content));
 grid-template-columns: minmax(200px, 300px) repeat(3,1fr);
96. If using the 1fr it can make a cell bigger than other that have 1fr because the rule says that the column width will not be smaller than content min.
97. Responsive layouts with auto-fit and auto-fill.
98. The ultimate way to create a responsive grid: grid-template-columns: repeat(auto-fit,minmax(200px,1fr));

grid-auto-rows:150px; . The auto-fit creates as many as columns can be added provided by the min-max function that has 1fr an a minimum of 200px. If there is 1000px width we will have 5 boxes per row, if we have only 750 than we have 3 boxes per row with 250px… And all of this continue to adapt itself until we have only one box per row with a width of min 200px.

99. **Nexter project**
100. Overall layout: How to build a complex and modern layout using advanced CSS grid techniques; - How to choose different row and column track sizes for different types of content
101. Feature section: -how and why to create grids inside of grids; -how to create a responsive component without media queries; -how to build a small component using CSS Grid.
102. Story section: -how to deal with overlapping grid items; -Why images are special and behave differently than other grid items; -How to decide if flexbox is a better tool in creatin situations.
103. Building the homes section: How to build a rather complex component using a mix of CSS Grid properties, overlapping and flexbox.
104. Building gallery section: -how to create a complex grid-looking gallery; -Using object-fit together with images for grid items.
105. Header section: how to manage vertical spacing in a responsive layout using CSS Grid techniques; -How to use:: before and ::after as grid items
106. Grid support: When we need to support older browser we should use the progression enhancement technique.
107. Next: -subgrids, individual grid column/row gap, select each row, column individualy.