**Web Development by Maximilian Schwarzmuller & Manuel Lorenz**

**Day 1:**

* HTML stands for Hypertext Mark-up Language.
* <h1>content</h1>, h1 tag is used to input heading or a title.
* The reason our website does not shows a domain or IP is that it is not connected to the server. So, it is being showed by a path in a local machine.

**Day 2:**

* Reason behind using Visual Studio is, it’s easier to debug.
* Opening and closing tags are highlighted.
* Shows the definition of the header when a mouse aims on it.
* Redirects to a website for further info.
* A lot of options to edit the code.
* HTML is a mark-up language.
* <h1>content</h1> is called an **HTML element**.
* **Element** “tell” the browser which kind of content is displayed (element, text, images, lists, buttons, etc...)
* Opening or closing tags are called **HTML tags**.
* Content between opening and closing tags is **HTML Content**.
* Content between angular brackets is called **HTML Name**.
* There are many HTML elements because there are a lot of things we can describe.
* Reason behind using HTML elements: without extra annotation, content often has no clear meaning like how will you tell it's a title or what.
* Also, it is used for visitors of the website, to search engine crawlers, to visitors using Assistive technologies (e.g., screen reader).
* We can use “ <https://developer.mozilla.org/en-US/docs/Web/HTML/Element> ” this website for further reference.
* We use <p> element for not visual element but rather for semantic reference. <p> refers to paragraph.
* We can add **HTML Attribute** to further describe the behavior of the content.
* Format: Attribute=""
* For <h1 style="value”>, where "*style*” is an HTML Attribute.
* Value can be font-family which means its type like Calibri or Comic Sans MS e.g., <h1 style="font-family: sant-serif">
* This font-family example is called CSS. “ <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference> ” can be used for CSS reference
* Text-align is used to align the text in center, left, right, etc...

**Day 3:**

* We use hexadecimal to get mix colors.
* Hexadecimal is of six digits.
* Represented by “#”
* Separated in groups of two (total three groups): red, green, and blue.
* Their combinations are then used to create different colors.
* You can use color picker on google or use some good website to pick a hexadecimal code.
* You can also use different notations if you don’t like hex notation like *hsla (hue-saturation-lightness-alpha)* and *rgba (red-green-blue-alpha)*. “ <https://github.com/academind/100-days-of-web-development/blob/02-html-css-basics/extra-files/colors.pdf> ”
* You can separate the code in multiple lines to make it look better. It can be done by using the tag after the line and before the line. Visual Studio will automatically indent it.
* You can also use **Shift+Alt+F** to make it the same and prettier faster.
* Dev tools are very useful to play and experimenting around with a certain page, even ours.
* By adding a new <p></p> element, you can add a line break and get a new paragraph.
* For adding links to the text, we can use an anchor tag, <a></a>.
* Instead of copying same style attribute everywhere, we can define a global tag at the top and assign it using CSS.
* Format:   
  <style>   
     Attribute\_name{   
         content   
    }   
  </style>
* You can use TAB to get tags while searching them.
* We use <head>content</head> to indicate metadata that will be displayed on tabs and on search engine.
* We use <body>content</body> to indicate content of that respective webpage.
* <!Doctype html> indicates that we are using html version 5 which is the latest one.
* <!-- can be used to comment some unused data and -->, it will serve as closing tag (<!-- hi by shy -->).
* In CSS, **/\*** content bro **\*/** can be used to comment out.
* **Inline Styles** means you assign element styles via the "style” attribute.
* **CSS Selector** means the part of the CSS rule that defines for which element the rule applies.
* We use title element to display the content on the tab or on the search engine.

**Day 4:**

* We can use a {} tag for styling anchor tag.
* “text-decoration” element (CSS) can be used to remove or add the underline.
* **Pseudo Selector** means a CSS selector like: hover that allows you to set styles for an element that fulfills a certain condition (e.g., A mouse cursor hovers over it).
* CSS: When we place “:hover” after any element, it means the element will change its color when we aim on it e.g., a:hover {}.
* CSS stands for Cascading-Style Sheets which means it can style to multiple elements at once.
* We can create a new file with .css and put all our styles in it.
* **<link>** can then be used to link to that file.
* **<link>** is called **self-closing tag** or **white element** which is **only** controlled by attributes e.g., it needs her=”” attribute.
* / can also be used to turn the tag into self-closing tag (*It’s optional)* e.g., <link her=”daily-challenges.css” rel=”stylesheet” />
* “her” in <link> attribute links both files not making an underline below it.
* “rel=” needs to be used to tell the browser relation to which type of sheet it is e.g., <link her="daily-challenges.css" rel="stylesheet">
* For selecting particular elements and them a different style, we can give them an ID e.g., <p id="todays-challenge">
* Then, for adding this ID, we use **ID Selector** (CSS) instead of **Type Selector** which is denoted “#” e.g., #todays-challenge {
* color: #ba8df5b2
* }
* **"font-weight"** can be used to make the text small, normal, bold or extra bold where values can also be used (CSS).
* **"font-size"** can be used to change the dimensions of the text in pixels (px, good absolute unit) by targeting *device-independent screen pixels.* "rem” and "%” can also be used which are relative.
* Px is not relative except device resolution because they don't dynamically grow or shrink. Only depends on the website but **not** on-screen width.
* CSS can change everything and therefore you don't use HTML for styling (there is default styling) but rather describe your content and to provide correct semantic around your content.
* We can use *google fonts* to change to something different. For this, we use the code that the web (google fonts) provides for adding particular font.
* We then write the name of that font where it’s in *apostrophe.* We can also place another font after the comma which will be used as second hand in case first one’s server shut down e.g., font-family: "Montserrat Alternates", sans-serif;

**Day 5:**

* The browser reads the code by excluding line breaks. whitespaces and indentations but we rather add theme to make them human readable.
* If we want output special characters that are in HTML semantics like “<”, we have two ways:   
   1. We can use <pre>content</pre> so the browser will output without excluding whitespaces or line breaks.   
    2. We can also use some special shortcuts like using maths formula e.g., &gt for “>” and &lt for “<”.
* For adding images, we use <img> which is a self-closing tag. Hence, it needs attributes. For this we use “src” (source) to specify the image locations.
* We should also use “alt” which will display some text in case the server fails (maybe ours) and it will be read out to blind people e.g., <img src="trophy-challenge.png" alt="Trophy for motivating to complete the challenge.">
* You **must** place “;” after every line in CSS.
* “img {}” is used to style image element.
* Width and height are used to describe image dimensions.
* “border-radius” can be used round the picture.
* By using text-align with image, it doesn’t work because it aligns the content *not* the tag.
* So, instead, we can align <body> to center because text-align will align its content as required.
* For adding spacing and background color we can use body tag because it will apply to the whole content.
* Reason why we say Cascading Style Sheets is because more than one CSS rule may apply to the same element.

**Day 6:**

* By using the extension “live server” we are able to get the page automatically reloaded without *reloading it*.
* <ol></ol> and <ul></ul> are used to create **Ordered List** and **Unordered List**.
* <li></li> will be used within this “ol” or “ul” tag to define **List Elements**.
* In CSS, while defining li {} element, "list-style" attribute is used to define which type of list it is e.g., square, none, bullets etc.
* *“Sub or specific attribute always win”*
* Parent-Child relationship in HTML means elements nested within one element like html is parent and body and head tag are children.
* Rule applied to parent may affect child as if it has inherited sometimes e.g., color:aliceblue applied to body will <p>, <a> element etc too.
* **Inheritance** simply means selected container rules applied to descendants.
* **Cascading** means multiple rules can be applied to the same element.
* When multiple **same** rules are applied to the same element, *“latest rule always wins.”*
* If we place ol {} below li{} element, rules of li {} will still win because concept of **specificity** which means more specific selector’s rule wins over less specific selector.
* We can also add “background-color” to multiple tags to define their colors even in anchor element. It can also be added in :hover e.g., in anchor tag to make it more attractive *instead* of an underline.
* Same for border-radius which can make elements look more beautiful.

**Day 6:**

* By using the extension “live server” we are able to get the page automatically reloaded without *reloading it*.
* <ol></ol> and <ul></ul> are used to create **Ordered List** and **Unordered List**.
* <li></li> will be used within this “ol” or “ul” tag to define **List Elements**.
* In CSS, while defining li {} element, "list-style" attribute is used to define which type of list it is e.g., square, none, bullets etc.
* *“Sub or specific attribute always win”*
* Parent-Child relationship in HTML means elements nested within one element like html is parent and body and head tag are children.
* Rule applied to parent may affect child as if it has inherited sometimes e.g., color:aliceblue applied to body will <p>, <a> element etc too.
* **Inheritance** simply means selected container rules applied to descendants.
* **Cascading** means multiple rules can be applied to the same element.
* When multiple **same** rules are applied to the same element, *“latest rule always wins.”*
* If we place ol {} below li{} element, rules of li {} will still win because concept of **specificity** which means more specific selector’s rule wins over less specific selector.
* We can also add “background-color” to multiple tags to define their colors even in anchor element. It can also be added in :hover e.g., in anchor tag to make it more attractive *instead* of an underline.
* Same for border-radius which can make elements look more beautiful.

**Day 7:**

* **The CSS Box Model:**
  1. **Padding** means the space content gets between the box and border of our content.
  2. It also has box property to define the borders of the box using **“border”** which has three types: ‘width’ ‘type of border’ ‘color’... Type of border e.g., solid or dashed etc.
  3. Margin is the space between the content of the box and the element e.g., between h1 and content of <a> tag.
  4. *Padding* has four types to define its size which size of content from the box as top, right, bottom and left. *You can remember it using a clock*.Can be *applied* to margin also.
  5. **“auto”** property can be used to align the text automatically like for margin it would be aligned in center if you apply it on right and left hand side.
* In body element, <h1> tag is defined as *main heading* of the page. <ol> is *core content* of the page while the <p> tag at end is called *footer.*
* For further reference, refer to: “ <https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Document_and_website_structure> ”
* **Selectors**: These include **Type** (element name), **ID** (#id), **Group** (element name, element name) and **Class** (.class).
* **Combinators**: These are the combination of selectors. There are two types of combinators:
  1. **Descendant** which means e.g., “li p” where all p with li as ancestor will be applied the rules e.g., header p {   
         margin-bottom: 36px;   
     }
  2. **Child** which means e.g., “h2 > p” where only p which are direct children of h2 will be applied the rules.
* For some elements if we replicate them next to each other, they are either next to each other or in a different line.
* To understands this concept, we use the difference between two terms called **"Block (level) vs Inline Elements”**.
* Block element uses an enter element so each element will be created in a new line which means both things are true i.e., *new line* and *full width*.
* Whereas these two things are not true for *inline elements* which means they are added in line with the element and only use the space when needed.
* <a>, <button>, <img> and <span> are some examples of inline elements.

**Day 8:**

* In inline elements, margin elements don’t work up and down, instead it works left and right.
  1. If full <a> element is in a header tag, only its content will be submerged as under it.
  2. Second difference between them is that if we add padding attribute to inline element like <a>, it may overlap the block element while if we add padding to block element like <p>, it will not overlap.
* To overcome first and second difference/problem, we can use **“display”** property which can be set to *inline-block element* to combine both worlds.
* In cases such as adding margin on <h1> and <p> element next to each other, so between the bottom margin of <h1> tag and the margin above the <p> tag, larger margin always wins e.g., if we add 12px to <h1> element and 8px to <p>, so, between them 12px of margin will be applied **NOT** 8px.
* This is the case for block elements not for inline elements.
* We have two types of inline elements: **replaced** and **non-replaced.**
* Replaced elements such as <img> which is replaced by an image while non-replaced like <a> which is not replaced.
* **“box-shadow”** is the property which can give a box, the 3d effect. It has four types described as a format i.e., ‘offset in x-axis (+ve denoted as right)’ ‘offset in y-axis (+ve denoted as down)’ ‘blur (number of pixels to be blurred all around’ ‘color’.
* Types of selectors:
* **Tag type selector**
  1. CSS: p {...}
  2. Would select this HTML element for example: <p>Some text...</p>
  3. This selector selects all HTML elements that are of this tag type
  4. **ID selector**
  5. CSS: #some-id {...}
  6. Would select this HTML element for example: <h1 id="some-id">...</h1>
  7. This selector selects the element that has this ID on it (should only be once per page)
  8. **Class selector**
  9. CSS: .some-class {...}
  10. Would select this HTML element for example: <h1 class="some-class">...</h1>
  11. This selector selects all HTML elements that have this class on them
  12. **Attribute selector** *(new)*
  13. CSS: [src] {...}
  14. Would select this HTML element for example: <img src="...">
  15. This selector selects all elements that have this HTML attribute on them
  16. **Universal selector** *(new)*
  17. CSS: \* {...}
  18. Would select this HTML element for example: <p>.... </p><img ...>
  19. This selector selects ALL HTML elements (directly, not through inheritance but as if you would target them all individually)
  20. **Grouping selector / selector list**
  21. CSS: p,. some-class {...}
  22. Would select this HTML element for example: <p>...</p><h2 class="some-class">...</h2>
  23. This selector selects all elements that match the individual selectors in that list
  24. **Combined selector**
  25. CSS: p.some-class {...}
  26. Would select this HTML element for example: <p class="some-class">...</p>
  27. This selector selects all elements that meet both conditions (i.e., "<p>" elements with "some-class" class on it, in this example)
  28. **Pseudo selector**
  29. CSS: a:hover {...}
  30. Would select this HTML element for example: <a>...</a> *(when the user hovers over it)*
  31. This selector selects all elements that meet this "pseudo state" (i.e., all links that are hovered in this example)

**Day 9 & 10:**

* Notes, HTML and CSS link: “ <https://github.com/academind/100-days-of-web-development/blob/04-html-css-first-practice-summary/extra-files/html-css-basics-summary.pdf> ”
* In <a> tag, we can also add the attribute **"target”** which means we can tell the browser to open the link in the same tab or in the new tab etc.
* For opening it in the same tab, we use **"\_self”** and for opening it in the same tab we use **"\_blank”** and for opening it in new tab.
* <span> element has no semantic meaning.
* To make it more clear even more screen reader, we can use <strong> or <em> instead <span>.
* To make it more semantic, we can add another element under <main> section which is known as <section>.

**Day 11:**

* **Deploying a website:** Moving the website code onto a remote machine (server) that serves the site to visitors.
* **Hosting:** The remote machine (“server”) hosts (stores) the website code and serves it to visitors.
* A remote machine (“server”) holds the website code and sends it to browser that sent a request to that remote machine.
* For hosting, we can also use netlify drop.
* To add a favicon, we <link> element along with the attributes: rel, her and type e.g., <link her="images/trophy-favicon.ico" rel="icon" type="images/x-icon" class="link">
* For favicon we can use this link: “ [https://favicon.io](https://favicon.io/) ”
* **Relative Path** e.g., images/my-image.jpg means it will look for files and folders *relative from the source file* (i.e., the file that contains that path).
* **Absolute Path** e.g., /images/my-image.jpg means it will look for files and folders *relative from the root folder* (i.e., the main project folder that is being served).
* **Git version control** is used for web development projects organized in “repositories”.
* **GitHub** is cloud hosting provider for “Git repositories” and it is great for Collaboration in & contribution to web development projects.
* For human user interaction with computer, we use GUI (Graphical User Interface) and CLI (Command Line Interface).
* Reason behind using CLI is that it is the most common way to efficiently work with Git and GitHub.
* In cmd, “**dir”** is used to see all the files in the folder
* You can “**cd..”** to move one folder up, “**cd file path”** can be used to move one folder down.
* *Absolute* and *relative* path can both be used.
* **"cls”** can be used to clear the screen.
* **“mkdir”** can be used to create a new folder.
* **“echo”** can be used to create new files along with describing its type using **“>”** e.g., *echo first file > test.txt* where “first file” is the content in the file and “test.txt” is its type as text and “test” as name.
* **“del”** can be used to delete a *file.*
* **"rmdir”** can be used to delete a *folder*.

**Day 12:**

* In **git**, **working directory** contains in current project folder and **repository** contains all tracked folders and files.
* **Commit** has code snapshots and **branch** where all commits are stored. All commits are stored in *main/master* and it is the core location of this master branch.
* New branches can also be created.
* We can terminal integrate in visual studio to use git.
* To use git commands, start by typing git e.g., git init.
* **“git init”** can be used to initialize a repository.
* **“git branch –m <name>”** can be used to change the name of the repository.
* **“git add <file>”** can be used to add a file to stage area in git.
* **“git commit –m "<message>””** can be used to add commit message to the file.
* **"git config –-global –-edit”** can be used to edit the credentials but we need to link git part...
* To exit menu of above line, press esc, x, one after another.
* **“git config –-global user.name "<name>””** or **“git config –-global user.email email”** to add our username and email.
* **"git log”** can be used to check if log is created and when it is created.
* Instead of using *git add <file> <file>* and so on if needed, we can simply type in **“git add.”**.

**Day 13:**

* **“git checkout <branch name>”** can be used to switch between branches and **“git checkout –b <branch name>”** can be used to create and *immediately* switch to that branch.
* **“git merge <branch name>”** can be used to merge a branch into *current one*.
* If we add a change to a new branch and to old the branch and then, try to merge them, we then would need to accept which change should accept because it results in a **conflict**.
* **“git rm <branch name or specific path>”** can be used to delete a file and to immediately commit while if you delete it *manually*, you need to commit it.
* **“git reset --hard HEAD~number”** can be used to delete previous changes to the *number* specified after *“HEAD~”*.
* **“git branch –D <branch name>”** can be used to delete the branch specified by path or *direct* name.
* **“git checkout -- . /<branch name or specified path>”** can be used to revert it to normal (i.e., revered the *unstaged changes*)
* **“git reset -- ./<branch name or specified path>”** can be to revert staged *but* uncommitted changes to the file.
* **Advantages** of GitHub:
  1. Cloud Storage
  2. Portfolio Page
  3. Collaboration
  4. Contribution
* GitHub repositories are called **remote repositories** while Git repositories are call **local repositories**.
* Both are required to establish a connection in between.
* **"git remote add <url identifier>(optional) < url>”** can be used to connect to the url.
* **“git push <url identifier> <branch name>”** can be used to push this repository to the url where “<branch name>” is the branch we want to add.
* **“git credential reject”** *=> ENTER*, then, **“host=github.com”** *=> ENTER*, then, **“protocol=https”** *=> ENTER*, then, *double => ENTER* would erase all the credentials stored.

**Day 14:**

* **“git clone <url> .”** would copy its content *directly* to the folder. “.” is required to add the contents directly instead copying root folder along with.
* **"git pull <url identifier> <branch>”** or **"git pull”** can be to pull and merge the data into the file.

**Day 15:**

* **<nav></nav>** can be used to tell the browser that this is the **navigation bar** where we will provide links to other sites.
* **"text-transform”** property can be used to describe text as required.
* **“text-shadow”** attribute/property can be used to add shadow to text.
* If we set *display* to **flex**, which is part of the **flex box**. The flex layout allows responsive elements within a container to be automatically arranged depending upon screen size.
* **“flex-direction”** can be used to define its arrangement like row, column, column-reverse etc.
* In *“flex-direction: column”*:
  + **“Main Axis”** in flex box means as y-axis (i.e., top to bottom etc) and **“Cross-axis”** as x-axis.
* In “*flex-direction: row”*:
  + **“Main Axis”** in flex box means as x-axis (i.e., top to bottom etc) and **“Cross-axis”** as y-axis.
* **“flex-wrap”** is a property that sets whether it should adjust its width or not which mean that it can be set to **nowrap** and **wrap**.
* **"flex-flow"** is the combination of *flex-direction* and *flex-wrap* e.g., flex-flow: row nowrap;
* **“align-items”** can be used to align the items as needed e.g., center.
* **"justify-content"**, specifically used in flex box tells the browser to e.g., put space in between the contents e.g., space-between, space-around, space-evenly, center, flex-start and flex-end etc.
* **“background-image”** can be used to specify a background image with url describing its location in inverted commas e.g., background-image: url("/images/places/ocean.jpg");
* **“background-position”** can be used to set the image according to us e.g., center.
* **“background-size”** can be used to set the size of the image e.g., 200px. *Cover* is a shortcut to cover the image within the section or width.

**Day 16:**

* **“position”** can be used to define how the element will be positioned e.g., *static* or *relative* etc. If we set it to *absolute*, then, it will be positioned right after the first element which has relative position *applied* to it. If we set it to *fixed*, it will be visible all through the page irrespective or relative being applied.
* **“top”** or **“bottom”** etc can be used to define relative position of the e.g., element e.g., 20px.
* **"box-sizing"** can be used to describe the size of the box e.g., content-box or border-box.

**Day 17:**

* **“object-fit”** can be used to fit in with aspect ratio and it’s described by *fill*, *contain* or *cover* where cover is the best one as combines fill and contain.

**Day 18:**

* "background” is a property that we can use for linear gradient which is described by three properties: degrees, color, color e.g., background: linear-gradient (0deg, rgb (227, 255, 253), rgb (202, 243, 240));

**Day 19:**

* **“overflow”** can be used to clip the content of the box or something like that and it can be set to e.g., *hidden* or *visible* etc.
* **“CSS Grid”** can be used to set the contents as a grid and is an easy way describe similar to *flex box* like this display: grid;
* To separate the content into columns, we can use **“grid-template-columns"** and describe it with fraction denoted by **“fr”** and describe the space like *1fr 1fr* where first *1fr* means 1st column and second *1fr* means second column.
* **"li:first-of-type"** or **"li:nth-oftype(number)”** to select particular list element within that ordered/unordered list.
* **"grid-column"** property can be used to tell the content how much column space should it occupy and it works with grids on the browser described e.g., grid-column: 1 / 3;
* It can also be described like span <columns> which tells how many columns to occupy e.g., grid-column: 1 / span 2;

**Day 20:**

* We can use Unicode block for defining or something like that and for that, we can refer to hex value which is referred by **“&#x”** e.g., <a her="https://en.wikipedia.org/wiki/London">Explore &#x2192</a>
* Reference for utf website: “<https://en.wikipedia.org/wiki/Arrows_(Unicode_block)>”
* <https://academind.com/tutorials/the-position-property>
* <https://academind.com/tutorials/flexbox-basics-container>
* <https://academind.com/tutorials/css-grid-vs-flexbox>
* <https://academind.com/tutorials/flexbox-flex-items>

**Day 21:**

* 4 types of units along with pros and cons:
  + **“px”:** Easy to Understand and translatable (Pro).   
            Limited User focus and not scalable (Con).
  + **“%”:** Relative to parent element size (Pro).   
        Hard to manage due to cascading nature (Con).
  + **“em (ephemeral unit)”:** Size is relative to font size (Pro).   
             Hard to manage due to cascading (Con).
  + **“rem (root element's font-size)”:** Size is relative to root font-size. Preferred choice if applicable.
* “%” tells the browser that x% where "x” denotes font size compared to the parent element.
* 1em means 100%.
* In text case, “*em”* and “*%”* means the same.
* In other cases, like in padding case, it depends on *actual* element size.
* “%” always depends on parent element.
* “em” and “rem” depends on font-size always.

**Day 22:**

* Desktop Version Websites have:
  1. Traditional approach.
  2. Office based audience.
  3. Feature-rich website.
* Mobile Version Website have:
  1. Functional approach.
  2. Lifestyle/news focused audience.
  3. “Content first”.
* **Media Queries** can be used to adjust the website according to the device.
* If we create mobile-based websites first, then, **“min-width”** is used otherwise **“max-width"** is used.
* For **portrait devices** are smartphone (480px) and tablet (768px). For landscape device are notebook (1024px), computer (1200px) and tv (more than 1200 px).

**Day 23:**

* We can add media querie from the following format if we are moving from desktop to mobile version: @media (max-width: 768px) {
* #latest-products ul {
* grid-template-columns: 1fr;
* }
* }
* We would use min-width if we are moving from mobile to desktop version.
* Media querie should be added after the code to be altered and *only* required code needs to changed not all of it needs to be re-added.
* **"display”** being set to *none* makes that element invisible.
* **“Internal Links”** is used to add links within the website (adds defined id to url e.g., her=”#id”.
* **“The Target Selector”** is used to activate css rule if defined is selected in URL e.g., #id: target.
* **“Inline Elements”** need some content or text to be displayed onto the browser.
* **“space-around":** items are evenly distributed in the line with equal space around them.
* **“space-evenly”:** items are distributed so that the spacing between any two adjacent alignment subjects, before the first alignment subject, and after the last alignment subject is the same.
* "z-index" property can be used to position the elements on *z-axis.* It is automatically by default is set to 0. If we use "z-index: 1”, it wouldn't work since this property *only* applies to the elements set to the position other than static e.g.   position: relative;
* z-index: 1;

**Day 24:**

* Three things to remember when you start building a website from scratch:
  1. Add *different features* step by step.
  2. Think about the *core information* that should be transferred.
  3. Less is more – *Don’t overlay your style*.
* Don't over add the fonts and over add the font-weight. Add to emphasize text. That's it!
* Grey is always required for background as the least choice.
* Grey is also required for text because it looks more natural than black.
* Having a primary color looks a lot good because it makes your website more stunning.

**Day 25:**

* **“CSS variables/CSS custom properties”** is used to declare variables.
* It starts with the format of *"--”*, then, *“name”*, then *"-”* and so on... e.g. --color-grey-100: rgb (236, 236, 236);
* And then you can use it by the command of **“var(variablename)”** e.g., background-color: var (--color-grey-100);
* In color, we use values like 100 to 900 indicating their lightness to darkness and same in size except, we generally use 1 to 10 and so on...
* Variables can be used for any property but if we mistakenly add a wrong property to the element (which doesn’t apply to it), it will be ignored.
* Three types of selectors:
  1. **“html”:** selects html element i.e., CSS rules are applied to html element and inherited to nested elements inside the html element.
  2. **“:root”:** selects element which is the root of the element i.e., CSS rules are applied to root element and inherited to nested elements inside the root element.
  3. **“\*”:** selects all elements of the html document i.e., CSS rules are applied to all **elements** (specificity *must* be considered though).
* **Transformation:** Move/change appearance of element e.g., when hovering.
* **Transition:** Smooth transition from initial to transformed state (code is added to the event triggering the transition) e.g.,transform: scale (1.01);
* Where *“2”* means its size will be doubled and *“2”* means **200%**. Second e.g., will be transform: rotate(180deg);
* For *transition smoothening,* it is applied to *“initial state”* of the element, **not** an event triggering the transition.
* It is done by using four properties i.e., *property (e.g., Transform or background-color)*, *duration*, *timing function*, *delay* where delay means how much delay will be there before transition e.g., transition: transform 0.5s ease 1s;
* **SVGs (Scalable Vector Graphics)** is ❓ XML based markup language to describe two-dimensional vector graphics ❓. ✔ They are text-based description of scalable images that can be rendered by the browser ✔.
* “hericons” can be used to get pre-defined svg icons.

**Day 26:**

* Websites often request user input which is int terms of forms e.g., contact form, checkout form, login form, and comment form etc.
* For using forms, some key elements are:
  1. **“<input type=” ...”>”:** The type attribute controls which type of element will be shown (e.g., single line text input, number input, email address input).
  2. **“<textarea>”:** A multi-line text input field.
  3. **“<select>”:** A dropdown multiple-choice or multi-select input.
  4. **“<button>”:** A button to reset or submit the forms.
* **"<input>”** *(inline)* is a self-closing tag where *action (redirects us to specified url)* and *method (allows to talk about the type of http request that should be sent e.g., GET (to fetch a file) or POST (usually used to store the data in a file))* can be used to tell the browser what to do with the data e.g., <form action="/use-data " method="POST">
* *“Action”* being set to "/” means it will be redirected to the home page.
* **"<button>”** *<inline-block>* is a tag that can be used to do specific action.
* **"<form>”** can be used to tell the browser that it is a **“form”**.
* **"name=""”** attribute can be used to add name where the data will be *stored*.
* **POST:** Any data that is submitted will be part of the request itself.
* **GET:** Key idea is not post something but encodes the data in the url.

**Day 27:**

* To correct the size of “input box”, we need to use box-sizing: border-box;
* We can use "font” as *inherit* to overwrite browser-default styles e.g., font: inherit;
* **“elementname:focus”** where **“:focus”** means that when we will be focused on the field, styles specified will be applied.
* Different types of *<input>* *type* attribute which are like *radio* (can be used to present multiple options where only one option can be selected simultaneously), *checkbox* (can be used to present a *toggle* (yes/no) to the user), *file* (allows the user to pick up the file), *text*, *email*, *number*, *password*, and *date*.
* In *type=“number”*, **“step”** property can be set to how much increments should there be when we click on up and down arrows in that field.
* Multiple *"radio”* input elements can be set to a name="" which means that we will be able group them together.
* *“input”* elements can have an attribute of value which will tell the browser about which value it is e.g., <input type="checkbox" name="contact" id="contact-email" value="email">
* “Cursor” CSS element can be used to tell the browser to change cursor shape as soon as the cursor hits on the button.

**Day 28 & 29:**

* **“<textarea></textarea>”** element is used to allow the user to multiple lines of code freely and it has a closing tag e.g., <textarea name="user-message" id="your-message" cols="30" rows="10"></textarea>
* **“<select></select>”** can be used to make a dropdown for the users. Within this element, we use **“<option></option>”** property for telling the options we want in the dropdown. **“value”** attribute within *<option>* property sends server, the value is sent to the server, when that particular option is clicked e.g.

<select name="user-favorite-color" id="your-favorite-color">

            <option value="blue">Blue</option>

            <option value="orange">Orange</option>

            <option value="black">Black</option>

</select>

* We can **change** the function performed by the button by using type attribute e.g., reset, button and submit and
* <button type="submit">Submit</button>
* If we don’t want to validate under the elements of parent element form, we use **“novalidate”** attribute which is a **boolean attribute**.
* For not leaving any *input* empty, we can use a boolean attribute which is **“required”** e.g., <input type="email" name="user-email" id="your-email" required />
* We can also limit the length entered into the element by using **“minlength”** or **“maxlength”** attribute e.g., <input type="password" name="user-password" id="your-password" required minlength="7" maxlength="20" />
* To limit the value according to your field you can use **“min”** or **“max”** e.g., <input type="date" name="user-date" id="your-date" required min="1921-01-01" max="2003-07-21"/>
* **Month format** in coding is **“yyyy-mm-dd"**.
* To place an example at an input box as an extra hint, we can use **“placeholder”** e.g., <input type="text" name="user-name" id="your-name" required placeholder="e.g., Salman Adnan" />
* To stop text area box from resizing, we can use “resize” e.g., resize: none;

For further reference use: “<https://github.com/academind/100-days-of-web-development>”