General Intro

* css code can be placed inline with html code, in the html file but in a separate section of the same file or in an external file.
* when inline it goes within the relevant tag, when in file it goes in a **<style>** tag in the header section, when external it goes in a .css file
* for external css files you have to place a link to the file in the html: <**link** href="pathToCssFile" type="text/css" rel="stylesheet">
* rel — this attribute describes the relationship between the HTML file and the CSS file. Because you are linking to a style sheet, the value should be set to style sheet.
* to select an HTML element and style it in CSS use the what is inside the HTML tag followed by curly brackets e.g. **p {property: value;}**
* to select a class and style it in CSS use dot then the class name followed by curly brackets e.g. **.className {property: value;}**
* to select a id and style it in css use hash then the id name followed by curly brackets e.g. **#idName {property: value;}**
* IDs are meant to style only one unique element they should be used sparingly and only on elements that need to always appear the same.
* **Specificity** is the order by which the browser decides which CSS styles will be displayed. A best practice in CSS is to style elements while using the lowest degree of specificity, so that if an element needs a new style, it is easy to override.
* IDs are the most specific selector in CSS, followed by classes, and finally, tags. If IDs are used in conjunction with classes and tags the ID css will always override the others.
* you can combine multiple selectors in one css style to be more specific by using **chaining** : **h1.special { property: value;}**
* to select nested html elements first select the mother class then the nested item: **.className h1 {property: value;}**
* you can override *any* style no matter how specific it is if you use the important command. This should only be used in emergency as it will override any other css rule: **p {color: blue !important;}**
* if you have multiple selectors applying the same css styles you can combine them into one comma separated statement: **h1, .menu { font-family: Georgia; }**
* background-image property will set the element's background to display an image. **background-image: url ("images/mountains.jpg");**

Text/Fonts

* font-family: indicates the font type. Default value: times new roman. Client needs to have the font installed for it to render. Limit the font types to 2-3 per page for best performance. let the font name be in quotes to cover spaces in the name.
* font-size: indicates the size of the font. It can be set in pixels or em units
* font-weight: indicates how bold the text will appear. It can be bold or normal. Some fonts are compatible with numerical weights: 400 is the default font-weight of most text, 700 signifies a bold font-weight, 300 signifies a light font-weight. Scale 100-900
* font-style: used to italicize text **h3 { font-style: italic; }**
* You can increase the spacing between words in a body of text using the word-spacing property. The default amount of space between words is usually 0.25em. **h1 { word-spacing: 0.3em; }**
* The technical term for adjusting the spacing between letters is called "kerning". Kerning can be adjusted with the **letter-spacing**property in CSS.
* text-align: positions the text with respect to its parent. It can be left, right or centre
* text-transform: change text to upper or lowercase
* We can use the line-height property to set how tall we want the line containing our text to be, regardless of the height of the text. A unit-less number, such as 1.2. This number is an absolute value that will compute the line height as a ratio of the font size.
* If a client does not have the fonts you specify installed then a fall back font will be used. You can specify the fall back options using: **h1 { font-family: "Garamond", "Times", serif; }** it will go left to right till an installed font is found.
* To link to online fonts use fonts.google.com and **<link>** to the fonts in the **<head>** of your html file. You can also link to online fonts by using **@font-face{}** at the top of your css file. Alternatively you could use **@import url(//fonts.googleapis.com/css?family=Open+Sans);** in your css file to link to the relevant font.
* Lastly for local fonts with the relative path in your CSS file. **@font-face {font-family: "Glegoo"; src: url(../fonts/Glegoo-Regular.ttf) format('truetype');}**

Box/Structure

* The **box model** comprises the set of properties, which define parts of an element that take up space on a web page.
* width and height — specifies the width and height of the content area.
* padding — specifies the amount of space between the content area and the border.
* border — specifies the thickness and style of the border surrounding the content area and padding. **border: 1px dotted red**
* margin — specifies the amount of space between the border and the outside edge of the element.
* Pixels allow you to set the exact size of an element's box (width and height). When the width and height of an element are set in pixels, it will be the same size on all devices
* a border doesn't have to be square. You can modify the corners of an element's border box with the border-radius property.
* padding property lets you specify exactly how much padding there should be on each side of the content in a single declaration. **padding: 6px 11px 4px 9px;** clockwise rotation from top. This is the same for margins as well.
* To center content use: **width: 400px; margin: 0 auto;** you cannot center without a width specified. the top/bottom margins will be zero and the side margins will be automatically set to center
* Horizontal margins add, so the total space between the borders of adjacent elements is equal to the sum of the right margin of one element and the left margin of the adjacent element.
* Vertical margins collapse, so the space between vertically adjacent elements is equal to the larger margin.
* min-width — this property ensures a minimum width of an element's box.
* max-width — this property ensures a maximum width of an element's box. Similar exist for height as well. All these help in controlling the experience for clients on different devices.
* The overflow property controls what happens to content that spills, or overflows, outside its box. It can be set to one of the following values (i.e. set to the parent of the desired element:
* hidden - when set to this value, any content that overflows will be hidden from view.
* scroll - when set to this value, a scrollbar will be added to the element's box so that the rest of the content can be viewed by scrolling.
* visible - when set to this value, the overflow content will be displayed outside of the containing element. Note, this is the default value.
* The first rule in an external css file should be: **\* {margin: 0; padding: 0; }** it resets the default margin and padding values of all HTML elements to ensure you are working with a clean slate.
* An element with **display: none**will be completely removed from the web page. An element with **visibility: hidden**, however, will not be visible on the web page, but the space reserved for it will.
* The default box sizing model **content-box** comes with sizing issues so it is better to reset the box sizing model in the beginning to border-box which auto sizes your content for you so that dimensions remain accurate. Use: **\* {box-sizing: border-box;}**
* One way to modify the default position of an element is by setting its position property to relative. This value allows you to position an element *relative* to its default static position on the web page. **position: relative; top: 20px; left: 50px;**
* When an element's position is set to absolute all other elements on the page will *ignore* the element and act like it is not present on the page. **position: absolute; top: 20px; left: 50px;**
* We can *fix* an element to a specific position on the page (regardless of user scrolling) by setting its position to fixed. **position: fixed; top: 20px; left: 50px;** This technique is often used for navigation bars on a web page.
* When boxes on a web page have a combination of different positions, the boxes (and therefore, their content) can overlap with each other, making the content difficult to read or consume. The **z-index**property controls how far "back" or how far "forward" an element should appear on the web page by the integer you assign. The highest one comes to the front of the page. The z-index property does *not* work on static elements so you have to set the position to relative or absolute before using it on an element.
* The default **display** for some tags, such as <em>, <strong>, and <a>, is called ***inline*** *(on the same line)*. Inline elements have a box that wraps tightly around their content, only taking up the amount of space necessary to display their content and not requiring a new line after each element. The height and width of these elements cannot be specified in the CSS document.
* The CSS **display** property provides the ability to make any element an inline element. This includes elements that are not inline by default such as paragraphs, divs, and headings**. h1 { display: inline; }**
* Some elements are not displayed in the same line as the content around them. These are called ***block****-level* elements. These elements fill the entire width of the page and, unless specified, are the heights necessary to accommodate the content inside of them. **strong { display: block; }**
* **Inline-block** display combines features of both inline and block elements. Inline-block elements can appear next to each other and we can specify their dimensions using the width and height properties. Images are the best example of default inline-block elements.
* If you're simply interested in moving an element as far left or as far right as possible on the page, you can use the **float** property. Floated elements must have a width specified otherwise, the element will assume the full width of its containing element, and changing the float value will not yield any visible results. **float: right;**
* The float property can also be used to float multiple elements at once. However, when multiple floated elements have different heights, it can affect their layout on the page. Specifically, elements can "bump" into each other and not allow other elements to properly move to the left or right. The **clear** property specifies how elements should behave when they bump into each other on the page. It can take on one of the following values: left — the left side of the element will not touch any other element within the same containing element. right — the right side of the element will not touch any other element within the same containing element. both — neither side of the element will touch any other element within the same containing element. none — the element can touch either side. **clear: left;**

Color

* color: this sets the foreground color of an element
* background color: this sets the background color of an element
* opacity is the measure of how transparent an element is. It's measured from 0 to 1
* **HSL** stands for hue (the color itself), saturation (the intensity of the color), and lightness (how light or dark a color is).
* Hue ranges from 0 to 360 and saturation and lightness are both represented as percentages like this: **hsl(200, 20%, 50%).**
* You can add opacity to color in RGB and HSL by adding a fourth value, a, which is represented as a decimal ranging from 0 – 1 : **hsl(200, 20%, 50%, 0.5).**

Grids

* To set up a grid, you need to have both a ***grid container***and *grid items*. The grid container will be a parent element that contains grid items as children and applies overarching styling and positioning to them.
* To turn an HTML element into a grid container, you must set the element's display property to **grid** (for a block-level grid) or **inline-grid** (for an inline grid). Then, you can assign other properties to lay out the grid.
* By default, grids contain only one column. If you were to start adding items, each item would be put on a new row; you need to explicitly define the number of rows and columns in our grid.
* We can define the columns of our grid by using the CSS property **grid-template-columns**: .**class { display: grid; width: 500px; grid-template-columns: 100px 200px; }** This property creates two changes. First, it defines the number of columns in the grid; in this case, there are two. Second, it sets the width of each column. The first column will be 100 pixels wide and the second column will be 200 pixels wide. To create rows use the same format with **grid-template-rows**
* Remember that rows are defined as a percentage of the grid's height, and columns are defined as a percentage of its width.
* Alternatively the property **grid-template**can be used: **.class { display: grid; width: 1000px; height: 500px; grid-template: 200px 300px / 20% 10% 70%; }** . When using grid-template, the values before the slash will determine the size of each row. The values after the slash determine the size of each column.
* When using % or px to specify row and column sizes it is possible for the contents to overflow beyond the available space. To ensure this doesn’t happen you can use **fr** which is the fraction unit: **height: 400px; grid-template: 2fr 1fr 1fr / 1fr 3fr 1fr;**
* The **repeat** function will duplicate the specifications for rows or columns a given number of times. **repeat(5, 1fr)**would split your table into five equal rows or columns. **grid-template-columns: repeat(2, 20px 50px)** will create four columns where the first and third columns will be 20 pixels wide and the second and fourth will be 50 pixels wide.
* To prevent a row or column from getting too big or too small the **minmax()** function can help: **.grid { display: grid; grid-template-columns: 100px minmax(100px, 500px) 100px; }** In this example, the first and third columns will always be 100 pixels wide, no matter the size of the grid. The second column, however, will vary in size as the overall grid resizes. The second column will always be between 100 and 500 pixels wide. Do not set a fixed width or the function will not work.
* The CSS properties **grid-row-gap**and **grid-column-gap**will put blank space between every row and column in the grid: **grid-template-columns: repeat(3, 1fr); grid-column-gap: 10px;** our grid will have three columns with two ten-pixel gaps between them. Alternatively **grid-gap** that can set the row and column gap at the same time. **grid-gap: 20px 10px;** will set the distance between rows to 20 pixels and the distance between columns to 10 pixels.
* **grid-row-start** and **grid-row-end**, we can make single grid items take up multiple rows. **.item { grid-row-start: 1; grid-row-end: 3; }** the HTML element of class item will take up two rows in the grid, rows 1 and 2. Row grid lines and column grid lines start at 1 and end at a value that is 1 greater than the number of rows or columns the grid has. For example, if a grid has 5 rows, the grid row lines range from 1 to 6. Alternatively **.item { grid-row: 4 / 6; }** does similar in one line. For columns the same format is used with **grid-column-start** keywords etc. Another way to get the same result is to use the span property: **.item { grid-column: 4 / span 2; }** it is the same as **.item { grid-column: 4 / 6; }**
* The **grid-area** property will set the starting and ending positions for both the rows and columns of an item in one line: **.item { grid-area: 2 / 3 / 4 / span 5; }** grid-area takes four values separated by slashes. grid-row-start / grid-column-start / grid-row-end / grid-column-end
* **The grid-template-areas**property allows you to name sections of your web page to use as values in the grid-row-start, grid-row-end, grid-col-start, grid-col-end, and grid-area properties. **grid-template-areas: "head head";** **grid-template-rows: 300px 120px 800px 120px; grid-template-columns: 1fr 3fr;** followed by **header { grid-area: head; }** means The header element spans the first row and both columns
* **justify-items**is a property that positions grid items along the inline, or row, axis. This means that it positions items from left to right across the web page. It accepts these values: start — aligns grid items to the left side of the grid area, end — aligns grid items to the right side of the grid area, center — aligns grid items to the center of the grid area, stretch — stretches all items to fill the grid area
* We can use **justify-content**to position the entire grid along the row axis. It accepts these values: start — aligns the grid to the left side of the grid container, end — aligns the grid to the right side of the grid container, center — centers the grid horizontally in the grid container, stretch — stretches the grid items to increase the size of the grid to expand horizontally across the container, space-around — includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element, space-between — includes an equal amount of space between grid items and no space at either end, space-evenly — places an even amount of space between grid items and at either end
* **align-items** is a property that positions grid items along the block, or column axis. This means that it positions items from top to bottom. It accepts these values: start — aligns grid items to the top side of the grid area, end — aligns grid items to the bottom side of the grid area, center — aligns grid items to the center of the grid area, stretch — stretches all items to fill the grid area
* **align-content**positions the rows along the column axis, or from top to bottom. It accepts these positional values: start — aligns the grid to the top of the grid container, end — aligns the grid to the bottom of the grid container, center — centers the grid vertically in the grid container, stretch — stretches the grid items to increase the size of the grid to expand vertically across the container, space-around — includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element, space-between — includes an equal amount of space between grid items and no space at either end, Space-evenly — places an even amount of space between grid items and at either end
* **justify-self**specifies how an individual element should position itself with respect to the row axis. This property will override justify-items for any item on which it is declared.
* **align-self**specifies how an individual element should position itself with respect to the column axis. This property will override align-items for any item on which it is declared.
* Something called the ***implicit* grid** takes over. The implicit grid is an algorithm built into the specification for CSS Grid that determines default behavior for the placement of elements when there are more than fits into the grid specified by the CSS. The default behavior of the implicit grid is as follows: items fill up rows first, adding new rows as necessary. New grid rows will only be tall enough to contain the content within them.
* **grid-auto-rows**specifies the height of implicitly added grid rows. **grid-auto-columns** specifies the width of implicitly added grid columns. grid-auto-rows and grid-auto-columns accept the same values as their explicit counterparts, grid-template-rows and grid-template-columns:
* **grid-auto-flow**specifies whether new elements should be added to rows or columns. It accepts these values: row — specifies the new elements should fill rows from left to right and create new rows when there are too many elements (default), column — specifies the new elements should fill columns from top to bottom and create new columns when there are too many elements, dense — this keyword invokes an algorithm that attempts to fill holes earlier in the grid layout if smaller elements are added, You can pair row and column with dense, like this: grid-auto-flow: row dense;.