Advanced CSS

Section 1: CSS Basics

1.1 What is CSS and How to write CSS?

CSS stands for Cascading Style Sheets. This allows us to create rules that specify how HTML elements should appear. CSS is used to make websites/web apps to appear aesthetically pleasing to the user.

Interesting Facts: CSS1 was first created in 1996. CSS2 was released in 1998. Finally, CSS3 first draft was released in 1999. Since 1999 it has continued developing and is still the current version as at April 2020.
CSS is split in separate modules like colours, backgrounds, animations, flex box, etc. and CSS creators/developers are working on developing those modules instead of creating another version i.e. CSS4. This is why the current CSS3 is hugely different from when it was released in 1999.

There are three methods of writing CSS which are inline, internal CSS and external CSS. It is best practice to use the external CSS method when writing CSS code because it is more readable, reusable and easier to maintain. The external CSS can be used across multiple HTML files by adding a link element tag in the HTML file referencing the .css file location to import the stylesheet.

The syntax below demonstrates how to write CSS:

```
h3 {
  background-color: blue;
  color: white;
}
```

The text represented in red is called the selector. This can be a HTML element and/or element attributes (such as id and classes) and/or pseudo selectors.

The opening and closing brackets wraps all the styles to be applied to the selector.

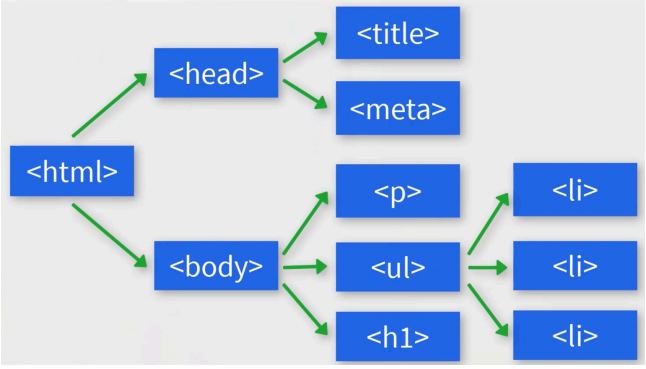
The text in blue is called properties which we want to style for the selector. Finally, the text in yellow is the values for the property.

The property and value pairs are actually called declarations. After each declaration we need to place a semi-colon (exception for the last declaration which does not require a semi-colon but it is recommended to add a semi-colon to all declarations).

1.2 HTML Elements Tree?

The HTML document is represented as a family tree. The family tree describes the relationships between family members and uses terms such as parent, child and sibling. A member of the family can be a parent to other while being a child of another family

member and a sibling of another family member. Below is a example representation of a HTML Elements tree for a .html document.



All HTML documents open with a <html> tag and all other element tags are contained within this element. Therefore, this is a parent element. The <html> element tag has not parents itself as well as no siblings.

Moving one level deeper we have the <head> and <body> element tags. These tags exists side by side which makes them siblings. They both also have the same parent <html> but they also contain children elements which make them parents themselves.

The <head> element has two children <meta> and <title> which are siblings to one another.

The <body> element has children of <p>, <ul> and <h1> which are all siblings of one another. Moving further down a level the <ul> has three children of <li> of its own making it a parent.

This is how we can view the structure of HTML documents as an HTML Element Tree. It is vital to have this knowledge when dealing with different types of selectors in CSS.

1.3 CSS Selectors

To style an element we require to select the element and there are multiple ways to access elements in CSS. Below is example syntax for selecting elements:

```
h1, h2 {
    color: green;
}

ul li {
    color: green;
}

#list-item {
    color: red;
}

.item {
    color: blue;
}

input[type="email"] {
    background-color: orange;
}
```

Comma (,) is used to select multiple elements and style them with the same declarations i.e. styles.

We can be more specific to reach child elements by using its parent as seen with the <li> element using its parent <ul> to select it.

The pound (#) symbol is used to select elements by their id attribute.

The period (.) symbol is used to select elements by their class attribute.

We can select an attribute value using square bracket ([ ]) notation.

It is important to understand the precedence of selectors. Execution of the code occurs line by line from top to bottom. However, specific selectors have higher precedence over less specific selectors and therefore the top to bottom rule does not always apply. In the example to the right, the ul li block of code has higher precedence over the li block of code and therefore the link items will have a text colour of green and not yellow.

Classes allow us to target multiple elements to share the same style while id allow us to target a specific element only because id must be unique.
Class and id attributes have a higher precedence over element selectors while the id selector has higher precedence over class selectors.

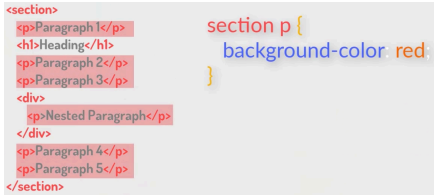
Disclaimer: It is possible to add the same id on multiple elements and they will be styled the same as we would have with classes. However, it is good practice to keep id attribute values unique while using classes to share attribute values.

Note: There are different ways of selecting elements but the common practice is to select using classes or ids instead of using the element names. In some cases a combination can be used.

1.3 CSS Combinators

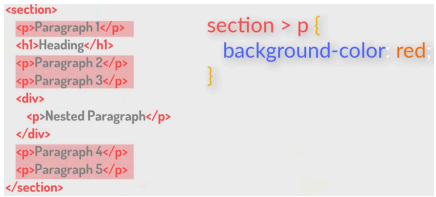
Combinators defines the relationship between selectors. In CSS there are four different combinators. Using the example HTML markup to the right we can explore the different combinator selectors in more detail:

Descendant (space) selector - matches all descendant elements of specified element.



All of the paragraph elements including the nested paragraph will be selected using the descendant selector and will be styled the same with a red background colour i.e. selecting all descendants.

Child (>) selector - selects all immediate children of the specified element.



The background colour for all paragraph elements will change to red except for the nested paragraph because it is not an immediate child of the <selector> parent element.

Adjacent Sibling (+) selector - selects the immediately following sibling of the specified element.

```
<section>
  <p>Paragraph 1</p>
  <h1>Heading</h1>
  <p>Paragraph 2</p>
  <p>Paragraph 3</p>
  <div>
    <p>Nested Paragraph</p>
  </div>
  <p>Paragraph 4</p>
  <p>Paragraph 5</p>
</section>
```

```
h1 + p {
  background-color: red;
}
```

The background colour will change only for the paragraph element which is placed immediately after the <h1> element.

General Sibling (~) selector - selects all siblings placed after a specified element.

```
<section>
  <p>Paragraph 1</p>
  <h1>Heading</h1>
  <p>Paragraph 2</p>
  <p>Paragraph 3</p>
  <div>
    <p>Nested Paragraph</p>
  </div>
  <p>Paragraph 4</p>
  <p>Paragraph 5</p>
</section>
```

```
h1 ~ p {
  background-color: red;
}
```

The background colour will be changed for all paragraphs which are placed after the <h1> element except for the nested paragraph element.

1.4 CSS Colours

There are multiple ways to choose/select colours in CSS.

The first method is by the colour name and all modern browsers support 140 standard colour names.

```
background-color: red;
```

The second method is by choosing rbg values which stands for red, blue and green and specifying a number between 0-255 for the three colours.

```
rgb(red, green, blue);
rgb(255, 0, 0);
```

The third method is selecting the rgba value . This is similar to the rgb values but adds a fourth value which is the alpha channel value. This allows us to select an opacity for the colour. The alpha value ranges between 0 and 1 i.e. 0-100% opacity.

```
rgba(255, 0, 0, 0.5)
```

The fourth method is to use hexadecimal values which start with a pound sign (#) followed by three values in pairs. The intensity of the colour can be controlled by values between 0 - f.

```
#ff0000 - red;
#00ff00 - green;
#0000ff - blue;
```

```
(#f00)
(#0f0)
(#00f)
```

Hexadecimals can be written in a shorten range using only three values but this will provide the least amount of ranged colours to select from.

Note: Colours have a huge effect on people's mood for example red is seen as a colour that grabs the user's attention and has an irritative effect. Green is seen as a colour related to nature and therefore is restful for the user's eyes. Blue is seen as a colour that relaxes, refreshes and produces peaceful feelings and moods. It is not recommended to use many colours on a webpage but a selection of colours that complement each other.

To colour your websites/web apps special attention should be taken to the:

- Perfect dominant colour
- Perfect colour scheme
- Perfect background colour
- Colours in the correct places

1.5 Inheritance

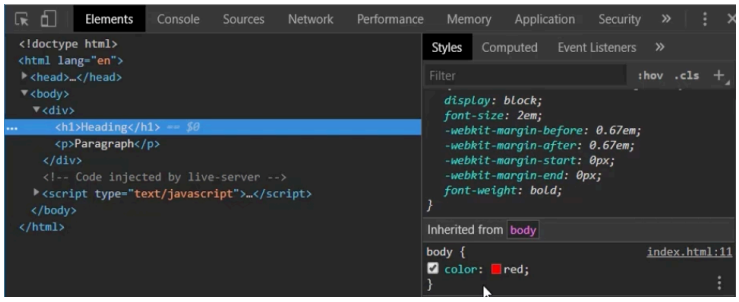
Inheritance is an important topic in CSS. The term itself describes the relationship between parent and child elements. In CSS each child element inherits its style from the parent element.

```
<body>
    <div>
        <h1>Heading</h1>
        <p>Paragraph</p>
    </div>
</body>
```

The example HTML markup to the left can be used to demonstrate the inheritance principle.

```
body {
    color: red;
}
```

The heading and paragraph elements will change text colour to red despite not being reference directly. This is because they inherited the <body> element because they are placed inside of the <body> element. This does not matter whether <body> is the direct or indirect parent.

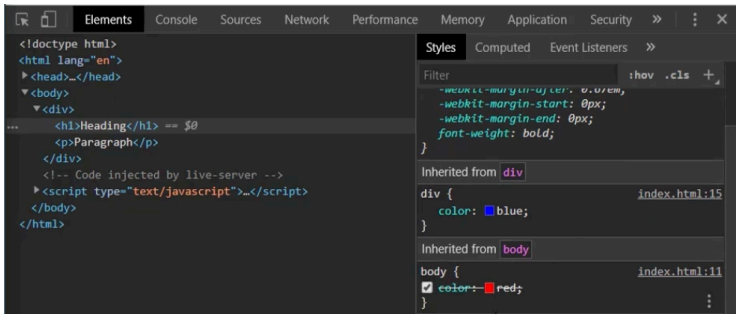


This can also be seen in the developer tools where the colour has been inherited from the <body> parent element for the , <h1> and <p> element.

```
body {
    color: red;
}

div {
    color: blue;
}
```

In this example the <h1> and <p> elements will inherit the blue text colour from their parent element style which will replace the red text colour style from the <body> element. This is due to CSS specificity and precedence.



This can also be seen in the developer tools Elements Styles tab where the colour: red now has a strike through which indicates that it is no longer applied. All the elements now inherit from their direct parent which is the element.

Therefore, closer parents have higher precedence to inherit its styles to their child elements. However, if the child element has its own style it will not inherit from any parent elements.

This is how inheritance operates in CSS i.e. child element inherits styles from its parent element. By default it is the closer parent element that has higher precedence to inherit styles to the child element, unless the child element has its own styles then the inherited style is overwritten and no longer applied.

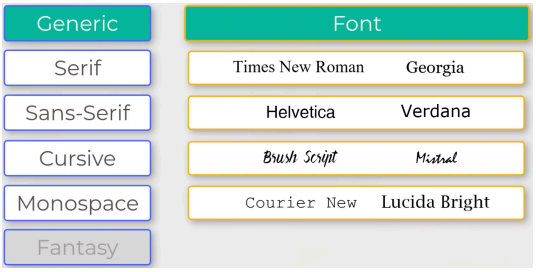
1.6 Text Formatting

We can use CSS to format text for example the background-color property allows us to change the background colour for the text element (e.g <p> or <h1>) while the color property allows us to change the text colour. There are other text formatting properties available to use:

Text Formatting Properties Examples	Description
background-color: #23af23;	Change background colour
color: #fff;	Change text colour
font-size: 20px;	Change font size from the default browser's 16px
text-align: right;	Align text. The four common values are left, right, center and justify. The browsers default value for text-align is left. The text-align property only works with block level elements and does not work for inline elements
display: inline;	Transform elements from block to inline and vice versa
font-family: Helvatica, sans-serif;	Change the font style of the element. If a font has a space in its name this must be wrapped in double quotes e.g. "Times New Roman". If the first font is available it will be applied else it will fall back on the next font within the the list
font-style: normal;	Change the style of the font
font-weight: bold;	Change the weight of the font

Disclaimer: The text-align property does not work with inline elements; however, we can manipulate this behaviour by wrapping the inline element with a block element such as a element. Styling the block element with the text-align property allows us to indirectly target the inline element text and apply the text-align styling to the text.

You would often work with inline elements and so it would be good to remember this technique which is often used.



In CSS we have two types of font-family names:
Generic - a group of font families with similar look and style i.e. sherif, sans-serif etc.
Font - defines the specific font family i.e. Times New Roman, Arial, Helvatica, etc.

Disclaimer: The browser's default font family is set to Times New Roman. This can be changed by clients in the browsers font settings. The Fantasy generic is not commonly used in in web development.

Google Fonts (https://fonts.google.com/) is the most popular website for importing web fonts to your projects. There are two different ways to import fonts from Google Fonts:

1. The <link> element tag allows us to import files from external websites into our HTML document using the href attribute to point to the website URL and the rel attribute set to stylesheet. This should be placed in the <head> element tag.

```
<link href="https://fonts.googleapis.com/css?family=Montserrat:300i,400,500,500i,600i,700,800" rel="stylesheet">
```

2. The CSS @import url( ) syntax can be used inside the .css file to import the font directly into the CSS file.

```
@import url('https://fonts.googleapis.com/css?family=Montserrat:300i,400,500,500i,600i,700,800');
```

The font can then be used as normal in CSS using the font-family property as though the font existed on the client's machine, demonstrated to the right.

```
h1 {
    font-family: 'Montserrat', Helvetica, sans-serif;
    font-style: normal;
}
```

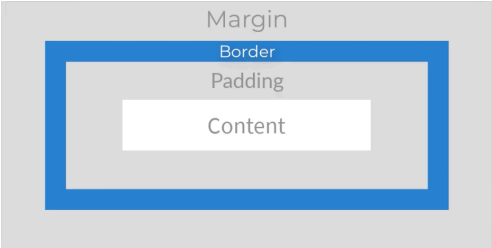
The font-weight property allows us to change the weight of the font e.g. lighter, light, normal, bold, bolder, 100, 200, 300, etc. while the font-style property allows us to change the style of the text eg. italic, normal, oblique, unset, etc.

Disclaimer: Not all fonts will support the number values between 100 - 900. The font-style can be used for Google Fonts to change the weight of the font using the numeric values.

Important Note: To create notes in CSS the syntax is to wrap the commented text in the following: /* Commented Text Appears Here */

1.7 Box Model

The term box model is used when dealing with layout of HTML elements. Generally, all HTML elements can be considered as boxes. The below diagram provides a visual representation of the box model which consists of four different parts:



The content of the element can be a text or images where we can manage its size using the width and height properties.

The padding clears an area around the content and the padding is transparent.

The border goes around the padding and content and can be considered a wrapper for them. By default the border is invisible but its style can be defined by calling the border style declarators.

The margin clears the area outside of the border and is also transparent.

Disclaimer: The box model for the element can be seen inside of the browser's developer tools which is a very useful tool when designing websites/web apps CSS styles.

There are two types of elements: block and inline. Block elements take up the full width that is available inside of the browser. In the case of inline elements they take up the width that is required to display the content of an element.

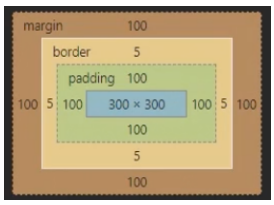
Below are example declarations to change the box model of elements:

Declaration Example	Description
margin-top: 30px; margin-right: 50px; margin-bottom: 70px; margin-left: 90px;	Change the margin area for the particular sides of the element box model
margin: 30px 50px 70px 90px; margin: 10px 20px; margin: 0;	Shorthand syntax for the four margin properties the first example is for each sides, the second example is for the top/bottom and right/left sides and the last example is for all sides with the same value
border-width: 5px;	Change the width of the border. This property requires the border-style to display the border
border-style: solid;	Set the style of the border to display by default this is set to none but can be changed to dashed, dotted, double, groove, hidden, solid, etc.
border-color: red;	Change the border colour (by default is black)
border: 5px solid red;	Shorthand syntax for the three border styles
padding-top: 30px; padding-right: 50px; padding-bottom: 70px; padding-left: 90px;	Change the padding area for the particular sides of the element box model
padding: 30px 50px 70px 90px; padding: 10px 20px; padding: 100px;	Shorthand syntax for the four padding properties the first example is for each sides, the second example is for the top/bottom and right/left sides and the last example is for all sides with the same value
width: 300px;	Change the width of the content

Declaration Example	Description
height: 300px;	Change the height of the content
box-sizing: border-box;	Defines how the width and height of the element will be calculated. the default value is content-box where the padding and border are not included in the elements total width and height. This can be changed to border-box to include the padding and border in the calculation

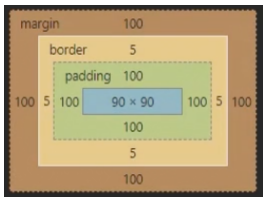
The below demonstrates the different in box-sizing calculation of content-box vs border-box for an element that has a height and width of 300px, padding of 100px and a border of 5px.

```
border: 5px #f00 solid;
padding: 100px;
width: 300px;
height: 300px;
box-sizing: content-box;
```



This has a total width of 510px (300px + 100px + 100px + 5px + 5px) and a total height of 510px (300px + 100px + 100px + 5px + 5px).

```
border: 5px #f00 solid;
padding: 100px;
width: 300px;
height: 300px;
box-sizing: border-box;
```



This has a total width of 300px (90px + 100px + 100px + 5px + 5px) and a total height of 300px (90px + 100px + 100px + 5px + 5px).

As seen in the above the box-sizing property affects how the content width and height is calculated i.e. whether or not to take into account the padding and border values to the total size of the content width or height.

Disclaimer: The margin property can be used to centre an element by setting the left and right margin sides to auto as seen in the example to the right.

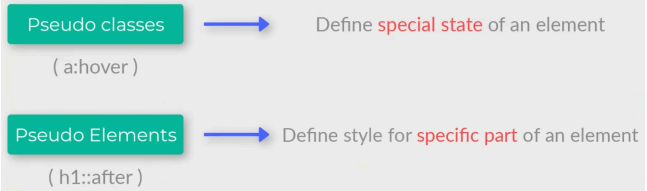
```
margin: 100px auto;
```

This is a good technique to remember for centring an element horizontally. Using auto for the top and bottom value will centre vertically.

Understanding how the box model works in CSS is very important because it will help you to control the layout and position of elements on the webpage.

1.8 Pseudo Classes & Pseudo Elements

Using pseudo classes we are able to create a more dynamic webpage while pseudo elements allows us to create additional content for an element and also style them.



There are a few pseudo classes we can use with the link element tag for example:

- The :link pseudo class allows us to style the link before the user visits a link
- The :visited pseudo class allows us to change the style of links after the user has visited a link
- The :hover pseudo class allows us to style the link when the user hovers over the link
- The :active pseudo class allows us to style the link while the links is being clicked by the user

The link pseudo classes should preserve this order i.e. :visited should come after :link, :hover after :visited and :active after :hover - this is because CSS code is executed top to bottom.

Some of the other commonly used pseudo classes are with child elements for example:

- The :first-child pseudo class allows us to target and style the first child element
- The :last-child pseudo class allows us to target and style the last child element
- The :nth-child( ) pseudo class allows us to target and style the nth child element. This method takes in one argument i.e. a number which is the nth element to select (this is not zero based indexing)

Some of the commonly used pseudo elements are:

- The ::first-letter pseudo element allows us to select the first letter of a text element and style it
- The ::before pseudo element allows us to insert some content before the contents of the selected element
- The ::after pseudo element allows us to insert some content after the contents of the selected element

1.9 Measurement Units

It is important to understand the measurement units in CSS because some declarations such as the width, height, padding, etc. can take values measured in various units. In general CSS supports two types of units which are absolute and relative. Absolute units are fixed units while relative units specify the length relative to another property. Relative units scale better between different rendering mediums.

Disclaimer: Although there are different absolute units as developers we tend to use the pixels (px) unit only and it is not recommended to use the other absolute measurement units.

1em = 16px.

The general the default is that relative unit is closely related to the parent element's size. For example where the parent element is 20px, 3em is equal to 60px (3 x 20px). The calculation is based on the closest parent element.

The rem unit is always calculated from the html root element size. By default the root html element size is 16px and it is recommended to use em and rem relative units.

The percentage unit calculates the size according to the closest parent element.

The vw and vh unit calculates the size based on the viewport of the browser's window size i.e. the visible view of the window. This is calculated for the width and height respectively. The vh and vw do not take into account the margin and border spacing.

1.10 Positions

Positions is a very important aspect in CSS because it is how we align elements on the page. There are five different types of positions: static, relative, absolute, fixed and sticky.

HTML elements have a static position by default and are positioned in the normal flow of the page. Elements with a position of static are not positioned in a special way and are also not affected by top, right, bottom and left properties.
In the case of static positions we can change the position of elements using margins, floats, flexbox and CSS grid.

A element with the position relative is positioned relative to its normal position. We are able to move the element using the top, right, bottom, left properties relative to its original position. In the example on the right the box will move downwards 20px from the top from its normal position. This will cause the element to cover another element.

Changing the size of a relative element does not move the position of the element and will not overlap other elements as it does with the top, right, bottom and left properties.

A element with position absolute is positioned relative to the closest positioned parent element. Setting the position of the element to absolute will jump the element up out of its normal flow of the page and it will sit behind the parent element. This allows us to move the element without breaking other elements normal flow. The element exists separately from all the other elements and therefore it does not have any effect on the other elements positions. Again we can move elements with position absolute using the top, right, bottom and left properties.
If we want to move an element according to its closest parent element that parent should have any of the positions except for static. Therefore, if ancestor parents have position static then the element with position absolute will always move according to the <body> element.
If an element has a position absolute but no width and height property assigned to it the element acts as if it was an inline element i.e. it only takes the width that is required to display its content.

The z-index property in CSS allows us to control which elements should be at the top of the other elements. By default each element has its index set to 0 and in order to place an element on top of the other elements the z-index must be greater than 0. The highest value is always placed on top of the lower values. Using the z-index property is not enough and would require to be used with the position properties other than static.

A element with a fixed position will jump out of the normal flow similar to position absolute but it's position will be relative to the viewport. This means that it will always stay in the same place even if the page was scrolled. This is frequently used with navigations where we would want to fix it at the top of the page.

A element with a sticky position is positioned according to the user's scroll position. When using position sticky we also need to define the top position of the element. As soon as the distance between the top edge of the browser viewport and the sticky element reaches the top value, the sticky element will keep its position and distance from the top and will remain there until it's parent element is no longer visible on the page i.e. the parent becomes hidden when scrolling down the page.

1.11 Overflow

There may be situations where the content of an element does not fit in its area and it is displaying outside the elements box. This is the case where you would use the CSS overflow property.

This specified whether to clear the content or add a scrollbars where the content of an element is too big to fit in a specified area. The overflow property has four different values: visible (default), hidden, auto and scroll.

The visible property value will display the overflowing content of the element.
The hidden value will hide the overflowing content but we would be unable to view this overflowing content.
The auto value fixes the issue with the hidden value by adding a scrollbar to the element box allowing us to view the overflowing content inside of the element's box. The scrollbar will appear automatically only when the content exceeds the element box.
The scroll value will always display the vertical and horizontal scrollbars regardless of whether it is needed or not.

The overflow-x and overflow-y allows us to create the vertical/horizontal scrollbars separately.

Disclaimer: The overflow property only works for block level elements with a specified height.

1.12 Floats

Floats is one of the many ways to align elements to a page. The float property allows you to take an element out from the normal flow of the page and place it as far to the left or right of the containing element as possible. When using the float property you should also use width property to indicate how wide the float element should be (*else results can be inconsistent as the box will take up the full width of the containing element like it is in the normal flow*).

The containing element will automatically lose its height from it's child elements when the child elements are floated. To get back the height of the containing element we would use overflow: auto declaration.

```
<body>
    <div class="container">
        <div class="left"></div>
        <div class="right"></div>
    </div>
</body>
```

```
.container {
    background-color: #ccc;
    border-top: 20px solid blue;
    border-bottom: 20px solid black;
    /* overflow: auto; */
}

.left {
    background-color: orangered;
    width: 100px;
    height: 100px;
    float: left;
}

.right {
    background-color: green;
    width: 100px;
    height: 100px;
    float: right;
}
```

An alternative method to fix the height issue of the containing element is to create a new element after the floated child element and target that element with a clear: both declaration. This is not an efficient method to solving the issue because we always need to create a new element.

```
<body>
    <div class="container">
        <div class="left"></div>
        <div class="right"></div>
        <div class="clear"></div>
    </div>
</body>
```

```
.clear {
    clear: both;
}
```

To avoid creating new elements to fix the height issue of the containing element the is a third alternative way known as the clear fix hack. This is the more modern approach. We would assign to the container element a class of clearfix and in the CSS file select the class and use the ::after pseudo element with the following three: content: ‘’; display: block’ and clear: both; declarations as seen in the example below.

```
<body>
    <div class="container clearfix">
        <div class="left"></div>
        <div class="right"></div>
    </div>
</body>
```

```
.clearfix::after {
    content: "";
    display: block;
    clear: both;
}
```

Disclaimer: Floats are becoming less popular because of CSS Flexbox and CSS Grid system are getting support in more modern web browsers and allow us to to control layout of elements in a much more easier and convenient way.

END