CSS

Basics

- CSS - Cascading Style Sheets
- Use *mdn website for any reference*
- It is a language that is used to describe the *style* of a html document
- Basic syntax can be given as :

```
* {
        padding : 0;                    /* boiler plate code for general css */
        margin  : 0;
}
```

```
h1 {                    /* h1 is called selector */
        color : red;        /* color - property */
}                       /* red - value */
```

Including Style 3 Ways :

- *Inline*

```
<h1 style="color:red"> Lorem ipsum </h1>
```

- *Style tag*

```
<style>
        h1 {
                color:red;
        }
</style>
```

- *External Stylesheet*
  - Writing CSS in a separate document & linking it with HTML file
  - We use this method mostly
  - To link CSS file to HTML we use this command or code inside of head tag such as :
    `<link rel ="stylesheet" href="style.css">`

Color Property

- Used to set the color of *foreground*
- We can simply use it as follows

```
color:red;
color:pink;
color:blue;
color:green;
```

Background Color Property

- Used to set the color of *background*

```
background-color: red;
background-color: pink;
background-color: blue;
background-color: green;
```

Color System

- *RGB*
  - The three primary colors of CSS are RGB . Every other color emerges from the mix of these three colors.
  - color : `rgb(255, 0, 0)` *red*
  - color : `rgb(0, 255, 0)` *green*
  - color : `rgb(0, 0, 255)` *blue*
- *HEX*
  - Hexadecimal
  - color : `#ff0000` *red*
  - color : `#00ff00` *green*
  - color : `#0000ff` *blue*

Selectors

- *Universal selector*
  - `* { }`
- *Element selector*
  - `h1 { }`
- *Id selector*
  - `# myId { }`
- *Class selector*
  - `.myClass { }`

Text Properties

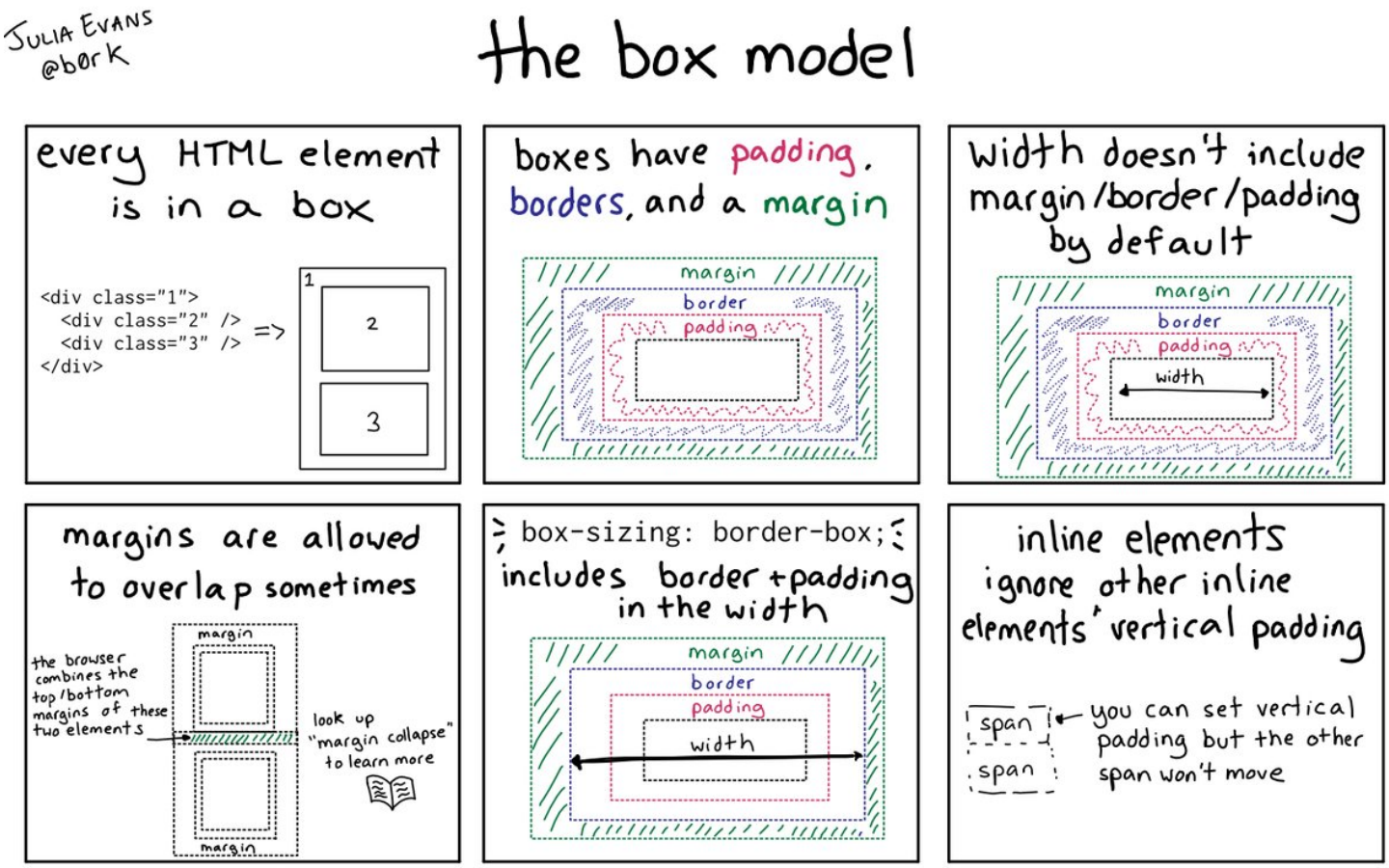
- *text-align*
  - text - align : left / right / center
- *text - decoration*
  - text - decoration : underline / overline / line-through
  - can also include additional properties such as `text-decoration : red underline;` or `text-decoration : blue wavy underline;` etc..
- *font - weight*
  - font - weight : normal / bold / bolder / lighter
  - font - weight : 100 - 900
- *font - family*
  - font - family : aerial
  - font - family : Arial, roboto
- *line - height*
  - line - height : 2px
  - line - height : normal
- *text - transform*
  - text - transform : uppercase / lowercase / capitalize / none

Units in CSS

- *Absolute*
  - Pixels (px)
    - 96 px = 1 inch (2.54cm)
    - `font-size : 2px;`
- *Relative*
  - Percentage *%*
    - It is often used to define a *size as relative to an element's parent* object
    - Ex : `width : 33%;` , `margin - left : 50%;`

unit	relative to
`em`	Font size of the parent, in case of typographical like *font-size* and, font size of the element itself, in the case of other properties like *width* suppose if it's `2em` it means double of the parent font-size
`rem (root em)`	Font size of the root element
`vh`	relative to 1% viewport (or our browser) height
`vw`	relative to 1% viewport (or our browser) width

Box model in CSS



- *Height*
  - By default, it sets the content area *height* of the element

```
div {
        height : 50px;
}
```

- *Width*

- By default, it sets the content area *width* of the element

```
div {
        width : 50px;
}
```

Border Properties

- Border
  - Used to set an element's *border*
  - Ex: `border-width : 2px;` , `border-style : solid / dotted / dashed ;` , `border-color : black;`
- Border *shorthand*
  - Short form to use multiple properties at once such as
  - `border : 2px solid black;` this is a blend of the above three border properties written at once
- Border *radius*
  - Used to *round the corners* of an element's outer border edge
  - such as `border-radius : 10px;` or
  - `border-radius : 50%;` (this becomes a circle if our width and height are of same pixels)

Padding Properties

- Padding-left
  - `padding-left : 25px (can be any value);`
- Padding-right
  - `padding-right : 25px;`
- Padding-top
  - `padding-top : 25px;`
- Padding-bottom
  - `padding-bottom : 25px;`
- Padding *shorthand*
  - `padding : 1px 2px 3px 4px;` (writing all of the above individual padding properties into a single property)
  - the order is : *(top | right | bottom | left)* in clockwise for a box

---

Margin Properties

- Margin-left
  - `margin-left : 25px (can be any value);`
- Margin-right
  - `margin-right : 25px;`
- Margin-top
  - `margin-top : 25px;`
- Margin-bottom
  - `margin-bottom : 25px;`
- Margin *shorthand*
  - `margin : 1px 2px 3px 4px;` (writing all of the above individual padding properties into a single property)
  - the order is : *(top | right | bottom | left)* in clockwise for a box

---

Display Properties

- *display : inline / block / inline-block / none*
- *inline* : Takes only the space required by the element. (no margin / padding)

- *block* : Takes full space available in width
- *inline-block* : Similar to inline but we can set margin and padding
- *none* : To remove the element from document flow
- *visibility*
  - `visibility : hidden;`

> Note
>
> When visibility is set to none , space for the element is reserved. But for display set to none , no space is reserved or blocked for the element.

Alpha Channel

- opacity (0 to 1)
- managing the shade of the color or opacity like how in normal rgb the range lies from *0 - 255* in alpha the range of the color lies from *0 - 1*
- *RGBA*
  - color : rgba (255, 0, 0, 0.5); *light red*
  - color : rgba (255, 0, 0, 1); *full red*

Position Properties

- The position CSS property sets *how an element is positioned* in a document
  - `position : static / relative / absolute / fixed;`
  - *static* : default position (the top, right, bottom, left and z-index properties have no effect)
  - *relative* : element is relative to itself (the top, right, bottom, left and z-index will work)
  - *absolute* : positioned relative to it's closest positioned ancestor (removed from flow)
  - *fixed* : positioned relative to browser (removed from flow)
  - *sticky* : positioned based on user's scroll position

Z - Index

- It decides the *stack level* of elements
- Overlapping elements with a larger z - index will cover those with a smaller one
  - `z-index : auto(0);`
  - `z-index : 1 / 2 / 3 / .....;`
  - `z-index : -1 / -2 / -3 / ......;`

> Note
>
> If our position is in *static* mode then we can't set the z-index. Normally at the start our position will always be static. So if we wants to use z-index we have to change our position from static to *relative* or so

---

Background Image & Size

- Used to set an image as a background
  - `background-image : url("image.jpg");`
- To set the size of the background image per say
  - `background-size : cover / contain / auto`
  - we mostly use the *cover* ones rather than the other two

Flex Box

- *flexible box layout*
  - It is a one-dimensional layout method for arranging items in row or columns
  - Firstly, if we want to use flexbox properties we have set the display to *display : flex;*

- *Flexbox Direction*
  - It sets how items are placed in the flex container, along which axis and direction
  - There will be two axis for the container namely *main axis (generally left to right (i.e. row-wise))* and *cross axis (generally top to bottom (i.e. column-wise))*
  - This direction property only used for the *containers* not for the individual elements
    - `flex-direction : row ;` (default) in this *main axis : left - right* & *cross axis : top - bottom*
    - `flex-direction : row-reverse;` in this *main axis : right - left* & *cross axis : top - bottom*
    - `flex-direction : column;` in this *main axis : top - bottom* & *cross axis : left - right*
    - `flex-direction : column-reverse;` in this *main axis : bottom - top* & *cross axis : left - right*

- *Flex Properties*
  - Flex properties for *flex containers*
    - `justify-content :`
      - alignment along the main axis
      - `flex-start / flex-end / center / space-evenly / space-around / space-between /`
    - `flex-wrap : nowrap / wrap / wrap-reverse`
    - `align-items :`
      - alignment along the cross axis
    - `align-content :`
      - alignment of space between & around the content
  - Flex properties for *flex items*
    - `align-self :` alignment of individual along the cross axis
    - `flex-grow :` how much a flex item will grow relative to the rest of the flex items if space is available
    - `flex-shrink :` how much a flex item will shrink relative to the rest of the items if space is available

> Note
>
> To make items / elements align horizontally use *justify-center*
> To make items / elements align vertically use *align-items*

> Question
>
> Which has higher priority `align-items` or `align-self` ?
>
> Ans: *align-items* applies for the `container` and *align-self* applies for the `individual flex items` . Therefore `align-self` has higher priority.

Media Queries

- Helps create a *responsive* website

```
@media (width : 600px) {
        div {
                background-color : red;
        }
}
```

```
@media (min-width : 600px) {
        div {
                background-color : red;
        }
}
```

```
@media (min-width : 200px) and (max-width : 300px) {
        div {
                background-color : red;              /* combined type query like settting a range */
        }
}
```

Transitions

- Transitions enable u to define the transition between two states of an element.
- `transition-property :` property u want to transition (font-size, width etc.)
- `transition-duration :` 2s / 4ms . . .
- `transition-timing-function :` ease-in / ease-out / linear / steps . . .
- `transition-delay :` 2s / 4ms . . .

```
div {
        height: 100px;
        width : 100px;
        background-color : blue;
        border : 2px solid black;
        transition-property : all;
        transition-duration : 2s;
        transition-timing-function : steps(5);
        transition-delay : 1s;
}

div: hover {
        background-color : red;          /* when on hover on div the color of div gets changed */
        color : white;
}

div: active {
        background-color : aqua;        /*when on clicked on div, it'z color gets changed */
        color : black;
}
```

- Transition *shorthand*
  - To use all of the above or combining multiple properties into a single command such as :
  - `transition : font-size 2s ease-in 0.2s` this is just an example
  - order is *property name* | *duration* | *timing-function* | *delay*

CSS Transform

- Used to apply *2D & 3D transformations* to an element
- *rotate*
  - `transform : rotate(45 deg);`
  - `rotate : 45 deg;`
  - `rotateX : 45 deg;`
  - `rotateY : 45 deg;`
  - `rotateZ : 45 deg;`

- *scale*
  - `transform : scale(2);` increases the size to two times *here both X (width) & Y (height) axis are getting scaled*
  - `transform : scale(0.5);` increases the size to half times
  - `transform : scale(1,2);` increases the size *1 time on X axis (width) and 2 times on Y axis (height)*
  - `transform : scaleX(0.5);`
  - `transform : scaleY(0.5);`
- *translate*
  - `translate : translate(20px);` will translate on both *X & Y axis*
  - `translate : translate(20px, 50px);`
  - `translate : translateX(20px);`
  - `translate : translateY(-200px);` will translate on *Y axis but downward direction*
- *skew*
  - `transform : skew(30deg);`

Animation

- To animate CSS elements

```
@keyframe myName {                /* here myName is an animation name (can be anything) */
        from {font-size : 20px;}
        to {font-size : 40px;}
}
```

- To apply the actual animation we need to use the animation properties
  - *Animation Properties*
    - `animation-name`
    - `animation-duration`
    - `animation-timing-function`
    - `animation-delay`
    - `animation-iteration-count`
    - `animation-direction` : normal / reverse / alternate / . . .
  - animation *shorthand*
    - `animation : colorAnimate 5s ease-in 3s infinite normal;`
    - above order is *animation : name | duration | timing-function | delay | iteration-count | direction*

```
div {
        height: 100px;
        width : 100px;
        background-color : blue;
        border : 2px solid black;
        position : absolute;
        top : 200px;
        left : 200px;
        animation-name : colorAnimate;
        animation-duration : 5s;
        animation-timing-function : ease-in;
        animation-iteration-count : 3;
        /* animation-direction : normal; */

}

@keyframes colorAnimate {
        from {background-color : red;}
        to {background-color : green;}
        /* from {left :0px;}
```

```
        to {left : 200px;} */
}
```

% in Animation

- Another way to write the *from* and *to*

```
@keyframes myName {
        0% {font-size : 20px;}
        50% {font-size : 30px;}
        100% {font-size : 40px;}
}
```

> Loader Animation

```
/*make a div with class name as loader in html*/

.loader {
        height : 200px;
        width : 200px;
        border-radius : 50%;
        border : 20px solid #023047;
        border-top : 20px solid #219EBC;
        animation : spin 3s ease-in 0s infinte normal;
}

@keyframes spin {
        from {
                transform : rotate(0deg);
        }
        to {
                transform : rotate(360deg);
        }
}
```
