CSS3 - Summary

Topics already covered in CSS

- The following are CSS3 features which have been covered in CSS
  - `box-sizing`
  - `box-shadow`
  - `border-radius`
  - *Media queries*

Basics

- *Multi-column* layout
  - We can get a newspaper-like layout where content can be arranged in columns
  - This differes from multi-column layouts created using floats, flex box etc. - in CSS3 multi-column layout, the content at the end of the first column continues at the beginning of the second column, and so on (every column'c ontent continues with the next column). This is not the case with multic-olumn layout created using floats, flex box etc. which arrange independent content in columns.
  - Some of the properties are
    - `column-count` - The number of columns in the layout
    - `column-gap` - The gap between the columns (similar to margins)
    - `column-rule` - Used to specify the divider between columns
    - There are other multi-column related properties and a shorthand `column` property that can be used to set many of the other properties.
  - Example

```
..newspaper-layout {
  column-count: 4;
  column-gap: 30px;
  column-rule: 1px solid crimson;
}
```

- *Text shadows* can be specified using `text-shadow`
  - Multiple shadows can be separated by commas
  - The parameters are x offset, y offset, blur radius (how much it spreads before it tapers out in a gradient-like fashion) and color of the shadow

```
p {
  text-shadow: 2px 2px 4px rgba( 128, 192, 255, 0.5 ), -20px -30px 4px lightpink;
}
```

- *Custom fonts*
  - The browser has at its disposal the fonts installed in the udnerlying platform (like Windows, Mac OS X etc.)
  - We can now use fonts that are not supported by the underlying platform
  - The `@font-face` rule can be used to specify font files for a custom font
    - Font files are specified in various formats - ttf, otf, eot etc. Every browser supports some format - ttf is very common.
    - Using `font-family` we give a custom name for the font
    - Using ```src`` property we set a font using files in various formats - this ensures the font loads in every browser.

```
@font-face {
  font-family: 'MyWebFont';
```

```
  src: url('webfont.eot'); /* IE9 Compat Modes */
  src: url('webfont.eot?#iefix') format('embedded-opentype'), /* IE6-IE8 */
      url('webfont.woff2') format('woff2'), /* Super Modern Browsers */
      url('webfont.woff') format('woff'), /* Pretty Modern Browsers */
      url('webfont.ttf')  format('truetype'), /* Safari, Android, iOS */
      url('webfont.svg#svgFontName') format('svg'); /* Legacy iOS */
}
```

- We apply a custom font using the name of the font (```font-family`` is used, like while applying built-in fonts)

```
p {
  font-family: 'MyWebFont', Garamond, 'Times New Roman', serif;
  font-weight: 400; /* apply the font-weight : 400 variant of the font */
}
```

- There are many websites that host font files and the CSS files that define the fonts using `@font-face` rules. Example Google Fonts
- Select the fonts and their variants, and add a link to the CSS file for sleected fonts.

```
<link href="https://fonts.googleapis.com/css?family=Playfair+Display:400,700|Raleway:400,700&d
```

- Note that these websites also allow you to download the font files and define your own CSS with `@font-face` rules.

- Multiple background images
  - We can now set multiple background images for an element - they are specified as comma-separated values. The first background image mentioned appears on top, the next appears below it, and so on.

```
p {
  background-image: url( "images/abstract_bg_image.jpg" ), url( "images/pexels-photo-1103970.jpeg"
}
```

  - If background-color is also set, it appears below the background images

```
p {
  background-image: url( "images/abstract_bg_image.jpg" ), url( "images/pexels-photo-1103970.jpeg"
  background-color: crimson;
}
```

  - The rest of background properties can also specify multiple comma-separated values - they are applied to images in the order specified for `background-image` property.

```
p {
  background-color: crimson;
  background-image: url( "images/abstract_bg_image.jpg" ), url( "images/pexels-photo-1103970.jpeg"
  /* set comma-separated background properties now - first for first background image, second for
  background-repeat: no-repeat, repeat; /* repeat behavior for first image, then second image *
}
```

- *Transforms*
  - Transforms alter the box for an element
  - There are *2d* and *3d* transforms
  - The 2d transforms are - *scale*, *translate*, *rotate*, *skew*
  - There are *3d* versions of these transforms too (but using it is more complicated)
  - The `transform` property is used to specify the kind of transformations to apply to an element
  - The `scale()` function "magnifies" the element box in both X and Y directions (independently). The content is proportionately magnified. Values between 0 and 1 will shrink instead of grow the element in that direction.

```
div {
  transform: scale(2, 3); /* 2 times wider and 3 times as tall */
}
```

- The `translate()` function moves the element's box by specified X and Y offsets - negative values can also be given (when negative, results in move to left for X direction, and move up for Y direction)

```
div:nth-of-type(2) {
  transform: translate( 300px, 400px );
}
```

- The `rotate()` function rotates the element in the X-Y plane. You can rotate the element clockwise or anti-clockwise (+/- value). The values are specified in angle units like degrees.

```
div:nth-of-type(3) {
  transform: rotate( -45deg );
}
```

- The `skew()` function "shears" the element in either X or Y directions. The values are specified in angle units.

```
div:nth-of-type(4) {
  transform: skew( 0deg, 45deg );
}
```

- There are other properties related to transforms, eg. `tranform-origin` which decides the "pivot point" about which rotation is applied (default is the center of the box where diagonals meet)

```
div:nth-of-type(3) {
  transform: rotate( -45deg );
  transform-origin: top right; /* rotate about the top right corner of the box */
}
```

- You can specify multiple transformations on an element - simply separate the functions called with spaces.

```
div:nth-of-type(5) {
  transform: rotate( 30deg ) skew( 0deg, 45deg ) translate( 400px, -700px );
}
```

- *Transition*
  - A CSS property value for an element can change with time. For example
    - A new value for the property is assumed on hover (due to hover styles for the element)
    - A new CSS class was applied after some time (through JavaScript code say) - this caused a change in some property's value
  - When a CSS property changes we can have it change gradually. For example, when thw width of an element changes from 100px to 200px, we can have it change over a second - the width shall assume intermediate values like 101px, 102px, ... 199px in that 1 second time duration.
  - This gradual change in property value is called *interpolation*
  - The `transition` property can be used to control the transition behavior like delay before transition starts, transition duration, *easing function* (decides the "velocity" of the transition at various points in time during the transition), etc.
  - Many CSS property changes can be "transitioned", but some cannot be (you can check the documentation for that property).
  - In the example below, we set any change in any property (`all`) to take place over a duration of 1 second. However, we override the behavior for width, setting it to change over 500 milliseconds. We similarly have height change over 2 seconds. On **hover**, these changes take place. `line-height` would change from 100px to 200px over 1 second, width changes from 100px to 200px over 500 milliseconds, and height changes from 100px to 200px over 2 seconds.

```
<head>
  <style>
      div {
          width: 100px;
          height: 100px;
          background-color: crimson;
          line-height: 100px;
          text-align: center;
          color: white;
          transition: all 1s, width 500ms, height 2s;
```

```
      }

      div:hover {
          width: 200px;
          height: 200px;
          line-height: 200px;
      }
  </style>
</head>
<body>
  <div>hello</div>
</body>
```

- *Animation*
  - Animations are more powerful than transitions
  - Animations are defined using `@keyframes` rule
    - A keyframes rule is defined independent of the element(s) on which the animation shall be applied
    - A name is given for an animation rule
    - The rule defines keyframes, i.e. the state of the animation (the CSS properties that apply to that element) at various points in time as he animation progresses.
    - The time point in specified in terms of percentage - for example, *50%* means *halfway through the duration of the animation*.
    - In between keyframes, the CSS properties that are animated are interpolated (just like transitions).
    - An animation is applied to an element using the `animation` property (a shorthand property). There are corresponding "longhand" properties that may be applied instead
    - The features that can be controlled are duration of the animation (in milliseconds, seconds etc.), delay in applying the animation, animation direction (forwards, backwards, alternate), easing function, animation count (how many times the animation should repeat, or even if it should keep going forever - infinite) etc.
    - Multiple animation keyframes may be applied to the same element with different values for the individual animation related CSS properties (comma-separated the way multiple background images are set).

      ```
      div {
        position: relative;
        top: 0px;
        left: 0px;
        width: 100px;
        height: 100px;
        background-color: crimson;
        line-height: 100px;
        text-align: center;
        color: white;
        animation: move forwards 2, rotate linear infinite;
        animation-delay: 2s, 0s;
        animation-duration: 3s, 3s;
      }

      @keyframes rotate {
        0% {
            transform: rotate( 0deg );
        }
        100% {
            transform: rotate( 3600deg );
        }
      }

      @keyframes move {
        0% {
            top: 0px;
            left: 0px;
        }
        25% {
            top: 0px;
            left: calc( 100vw - 100px - 16px );
        }
        50% {
            top: calc( 100vh - 100px - 16px );
            left: calc( 100vw - 100px - 16px );
        }
      ```

```
    75% {
        top: calc( 100vh - 100px - 16px );
        left: 0px;
    }
}
```

- *Flex box* layout