Summer Internship Report

FULL STACK WEB DEVELOPMENT INTERNSHIP

DAILY ANALYSIS REPORT

6/4/2020 - 13/4/2020

Amala Peter

3rd year Computer Science and Engineering Graduate

NSS College of Engineering,Palakkad

Asst.Prof Albert Sunny

IIT Palakkad

CONTENTS

Contents Page No

Chapter 1 :FRONT END DEVELOPMENT

1.2 CSS
- 1.2.6 How to add CSS 4
  - 1.2.6.1 External CSS 4
  - 1.2.6.2 Internal CSS 4
  - 1.2.6.3 Inline CSS 5
- 1.2.7 Visibility and z-index 6
- 1.2.8 FlexBox 8
- 1.2.9 CSS Units 15
  - 1.2.9.1 Absolute Units 16
  - 1.2.9.2 Relative Units 16
- 1.2.10 Responsive Web Design :- Media Queries 18
- 1.2.11 More on CSS Selectors 22
  - 1.2.11.1 Attribute and nth child Pseudo Selectors 25
  - 1.2.11.2 Before and After Pseudo Selectors 28
- 1.2.12 Shadow 32
  - 1.2.12.1 Box shadow 32
  - 1.2.12.2 Text shadow 33
- 1.2.13 Variables and Custom Properties 36
- 1.2.14 Animations and Keyframes 38
- 1.2.15 Transitions 43
- 1.2.16 Transform property 49

1.3 CSS Grid 53
- 1.3.1 Introduction 53
- 1.3.2 Creating basic Grid 54
- 1.3.3 Spanning Multiple Rows and Columns 58
- 1.3.4 Autofill/Autofit and Minmax 61
- 1.3.5 Layout using Grid Template area 65
- 1.3.6 Media Queries 70

1.4 Javascript 76
- 1.4.1 Introduction 76
- 1.4.2 Writing in-browser JavaScript and Developer Console 76
- 1.4.3 Variables,Data Types and Operators 78
- 1.4.4 Strings 83
- 1.4.5 String Functions 86
- 1.4.6 Scope,If-else conditionals & Switch Case 89

1.4.7 Arrays and Objects 94
1.4.8 Functions 99

REFERENCES 100

1.2 CSS

1.2.6 HOW TO ADD CSS

There are three ways of inserting a style sheet:

- External CSS
- Internal CSS
- Inline CSS

1.2.6.1 External CSS

With an external style sheet, you can change the look of an entire website by changing just one file! Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

SOURCE CODE:-

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

mystyle.css

```
body {
  background-color: lightblue;
}

h1 {
  color: navy;
```

```
  margin-left: 20px;
}
```

1.2.6.2 Internal CSS

An internal style sheet may be used if one single HTML page has a unique style. The internal style is defined inside the <style> element, inside the head section.

SOURCE CODE:-

```
<!DOCTYPE html>
<html>
<head>
<style>
body {

  background-color: linen;
}
h1 {

  color: maroon;
 margin-left: 40px;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

1.2.6.3 Inline CSS

An inline style may be used to apply a unique style for a single element. To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

SOURCE CODE:-

```
<!DOCTYPE html>
<html>
```

<body>

<h1 style="color:blue;text-align:center;">This is a heading</h1>
This is a paragraph.

</body>
</html>

1.2.7 VISIBILITY AND Z INDEX

Visibility: The visibility property specifies whether or not an element is visible.

SYNTAX= visibility: visible | hidden | collapse | initiaL | inherit ;

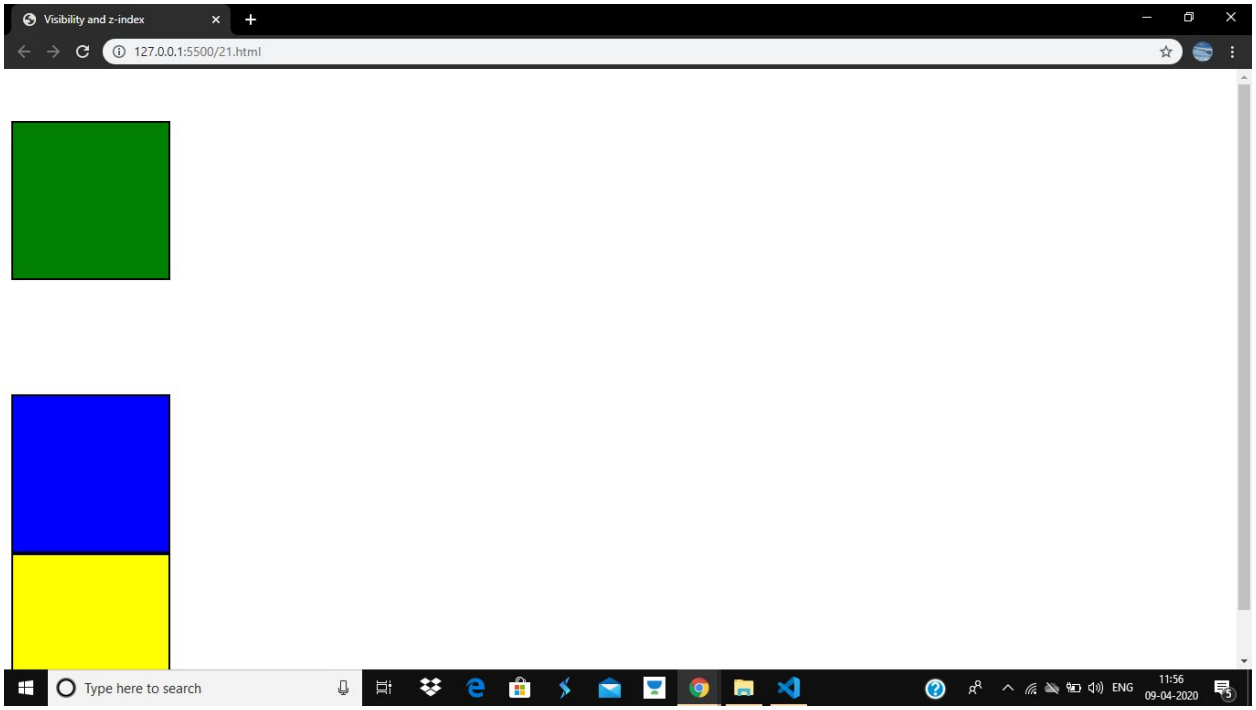
Z-index: The z-index property specifies the stack order of an element.An element with greater stack order is always in front of an element with a lower stack order.z-index only works on positioned elements (position: absolute, position: relative, position: fixed, or position: sticky).

SOURCE CODE :-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Visibility and z-index</title>
    <style>
        .box{
            width: 170px;
            height: 170px;
            border: 2px solid black;
        }
        #box-1{
            position: relative;
            top: 49px;
            z-index: -35;
            background-color: green;        }
        #box-2{
            position: relative;
```

```
            top: 14px;
            /* z-index  will work only for position: relative, absolute,
fixed or sticky; */
            z-index: -165;
            /* will hide the element and the space */
            /* display: none;  */
            /* will hide the element but will show its empty space */
            visibility:hidden;
            background-color: red;
            }
        #box-3{
            background-color: blue;         }
        #box-4{ background-color: yellow;         }
    </style>
</head>
<body>
    <div class="box" id="box-1"></div>
    <div class="box" id="box-2"></div>
    <div class="box" id="box-3"></div>
    <div class="box" id="box-4"></div>
</body>
</html>
```

OUTPUT :-



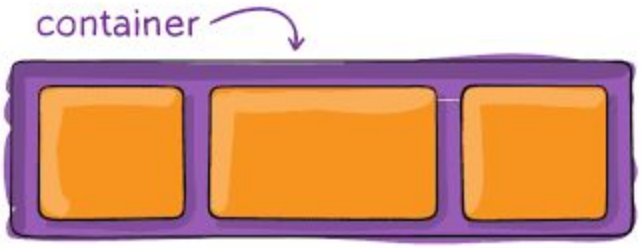
1.2.8 FlexBox

There were four layout modes:

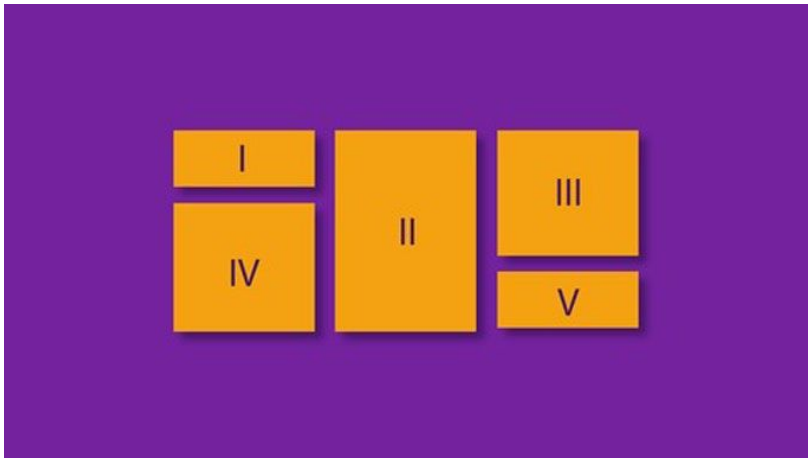
- Block, for sections in a webpage
- Inline, for text
- Table, for two-dimensional table data
- Positioned, for explicit position of an element

The Flexible Box Layout Module makes it easier to design flexible responsive layout structure without using float or positioning.

Flex Container :-



Flexbox Elements:-



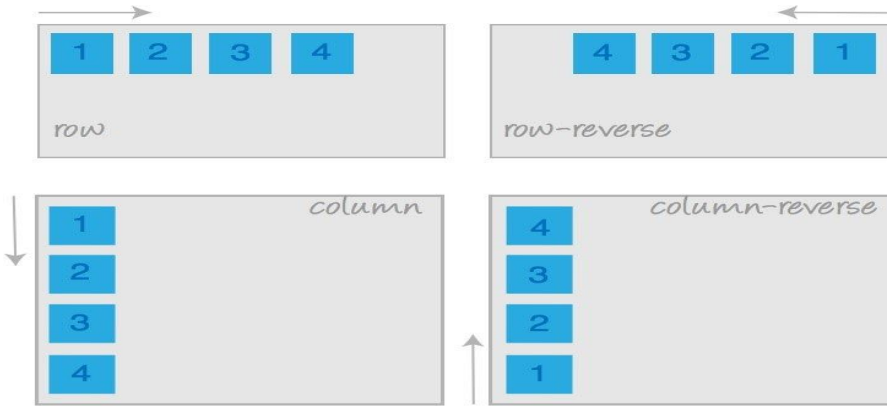
SOURCE CODE:-

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

The flex container properties are:

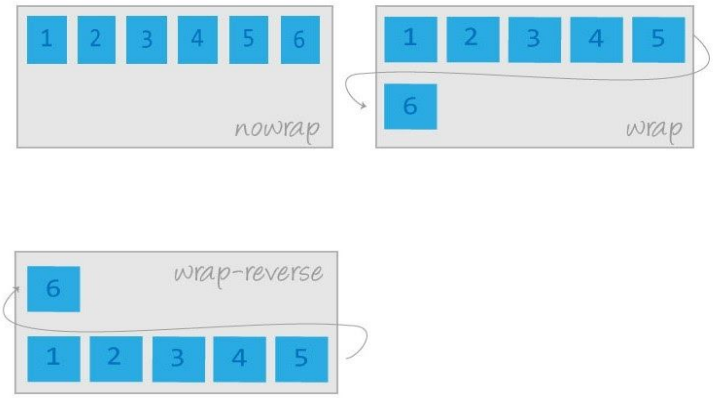
- Flex-direction

```
.flex-container {
  display: flex;
  flex-direction: column;
}
```

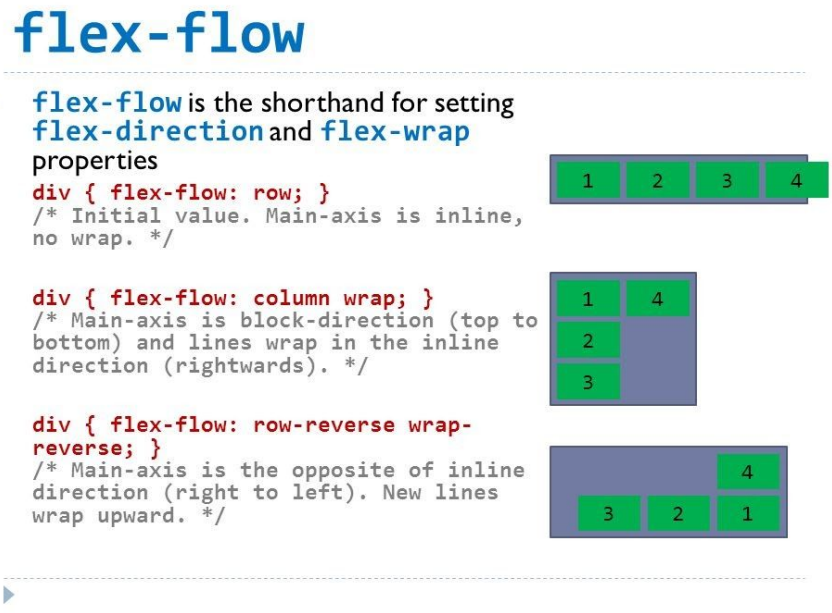


- Flex-wrap

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
}
```

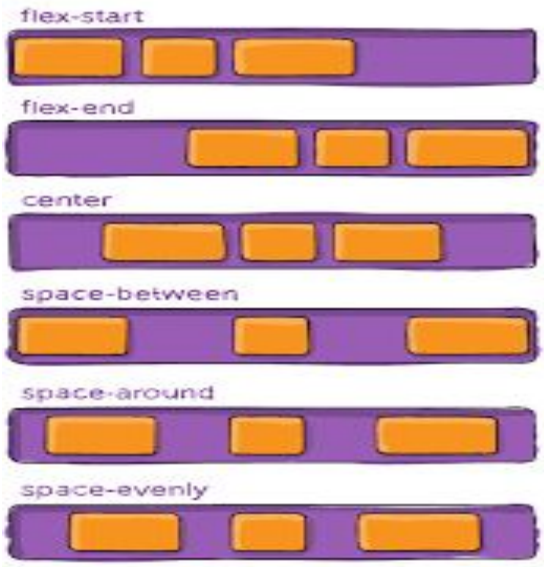


- Flex-flow



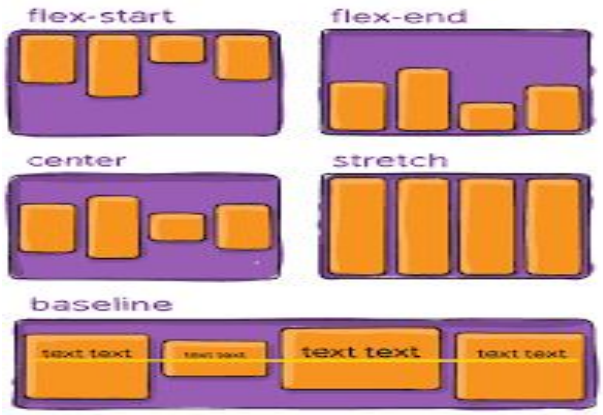
- Justify-content

```
.flex-container {
  display: flex;
  justify-content: center;
}
```



- Align-items

```
.flex-container {
  display: flex;
  height: 200px;
  align-items: center;

}
```



- align-content

```
.flex-container {
  display: flex;
  height: 600px;
  flex-wrap: wrap;
  align-content: space-between;
}
```

SOURCE CODE:-

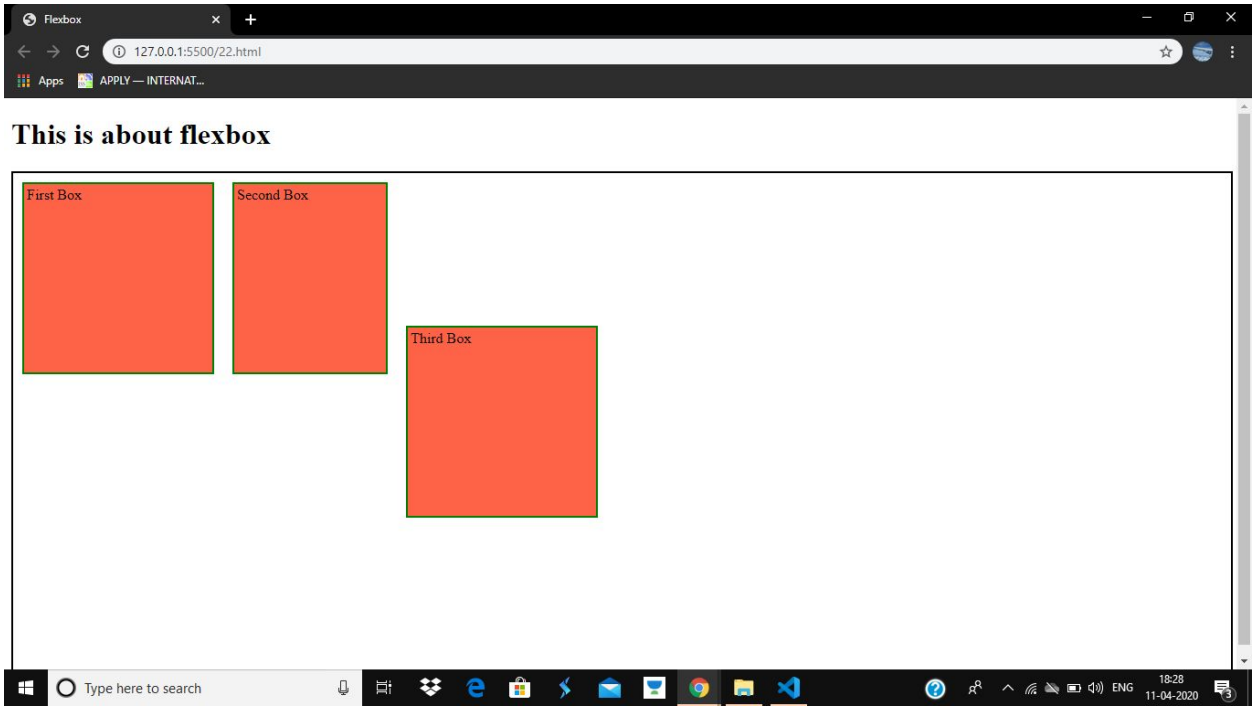
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Flexbox</title>
    <style>
        .container{
            height: 544px;
            width: 100%;
            border: 2px solid black;
```

```
                display: flex; /* Initialize the container as a flex box
*/

                /* Flex properties for a flex container */

                /* flex-direction: row; (Default value of flex-direction
is row) */
                /* flex-direction: column;
                flex-direction: row-reverse;
                flex-direction: column-reverse; */


                /* flex-wrap: wrap; (Default value of flex-direction is
no-wrap) */
                /* flex-wrap: wrap-reverse; */

                /* This is a shorthand for flex-direction: and
flex-wrap: ;; */
                /* flex-flow: row-reverse wrap; */

                /* justify-content will justify the content in
horizontal direction */
                /* justify-content: center; */
                /* justify-content: space-between; */
                /* justify-content: space-evenly; */
                /* justify-content: space-around; */

                /* justify-content will justify the content in vertical
direction */
                /* align-items: center; */
                /* align-items: flex-end; */
                /* align-items: flex-start; */
                /* align-items: stretch; */
            }
            .item{
                width: 200px;
                height: 200px;
```

```
                background-color: tomato;
                border: 2px solid green;
                margin: 10px;
                padding: 3px;
            }

            #item-1{
                /* Flex properties for a flex item */
                /* Higher the order, later it shows up in the container
*/
                /* order: 2; */

                /* flex-grow: 2;
                flex-shrink: 2; */

            }
            #item-2{
                /* flex-grow: 3;
                flex-shrink: 3 ; */
                flex-basis: 160px;
                /* when flex-direction is set to row flex-basis: will
control width */
                /* when flex-direction is set to column flex-basis: will
control height */
            }
            #item-3{
                /* flex: 2 2 230px; */
                align-self: flex-start;
                align-self: flex-end;
                align-self: center;

                }

        </style>
    </head>
    <body>
        <h1>This is about flexbox</h1>
```

```
        <div class="container">
            <div class="item" id="item-1">First Box</div>
            <div class="item" id="item-2">Second Box</div>
            <div class="item" id="item-3">Third Box</div>
            <!-- <div class="item" id="item-4">Fourth Box</div>
            <div class="item" id="item-5">Fifth Box</div>
            <div class="item" id="item-6">Sixth Box</div> -->
        </div>
    </body>
    </html>
```

OUTPUT:-



1.2.9 CSS Units

CSS has several different units for expressing a length. Many CSS properties take "length" values, such as width, margin, padding, font-size, etc. Length is a number followed by a length unit, such as 10px, 2em, etc. A whitespace cannot appear between the number and the unit. However, if the value is 0, the unit can be omitted. For some CSS properties, negative lengths are allowed. There are two types of length units: absolute and relative.

1.2.9.1 Absolute Units

The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.

1.2.9.2 Relative Units

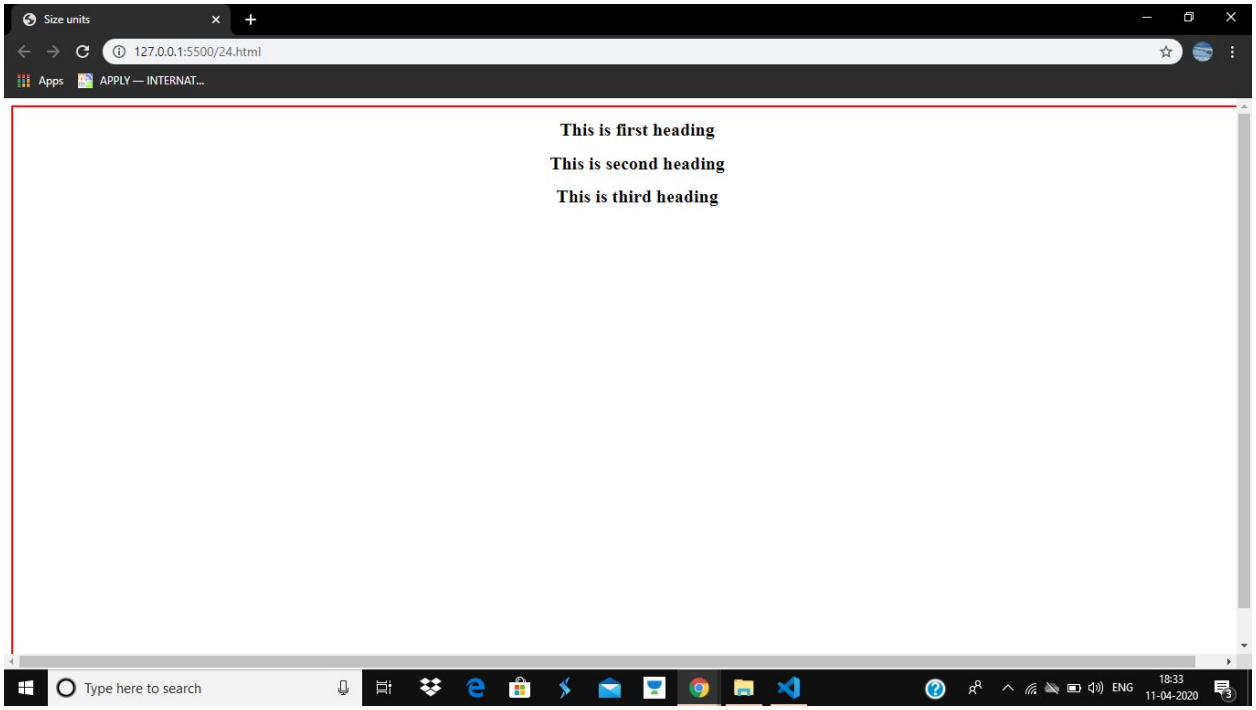
Relative length units specify a length relative to another length property. Relative length units scale better between different rendering mediums.

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Size units</title>
    <style>
        html{
            font-size: 25px;
        }
        .container{
            width:400px;
```

```
            /* height: 344px; */
            height: 100vh;
            width: 100vw;
            font-size: 10px;
            border :2px solid red;
        }
    h1{
        text-align: center;
    }
    #first{
        /* font-size: 3em;
        padding: 3em; */

    }
    #second{
        /* font-size: 3rem;
        padding: 3rem; */

    }

    </style>
</head>
<body>
    <div class="container">
        <h1 id="first">This is first heading</h1>
        <h1 id="second">This is second heading</h1>
        <h1 id="third">This is third heading</h1>
    </div>
</body>
</html>
```

OUTPUT:-



1.2.10 Media Queries

Media query is a CSS technique introduced in CSS3. It uses the @media rule to include a block of CSS properties only if a certain condition is true.

If the browser window is 600px or smaller, the background color will be light blue:

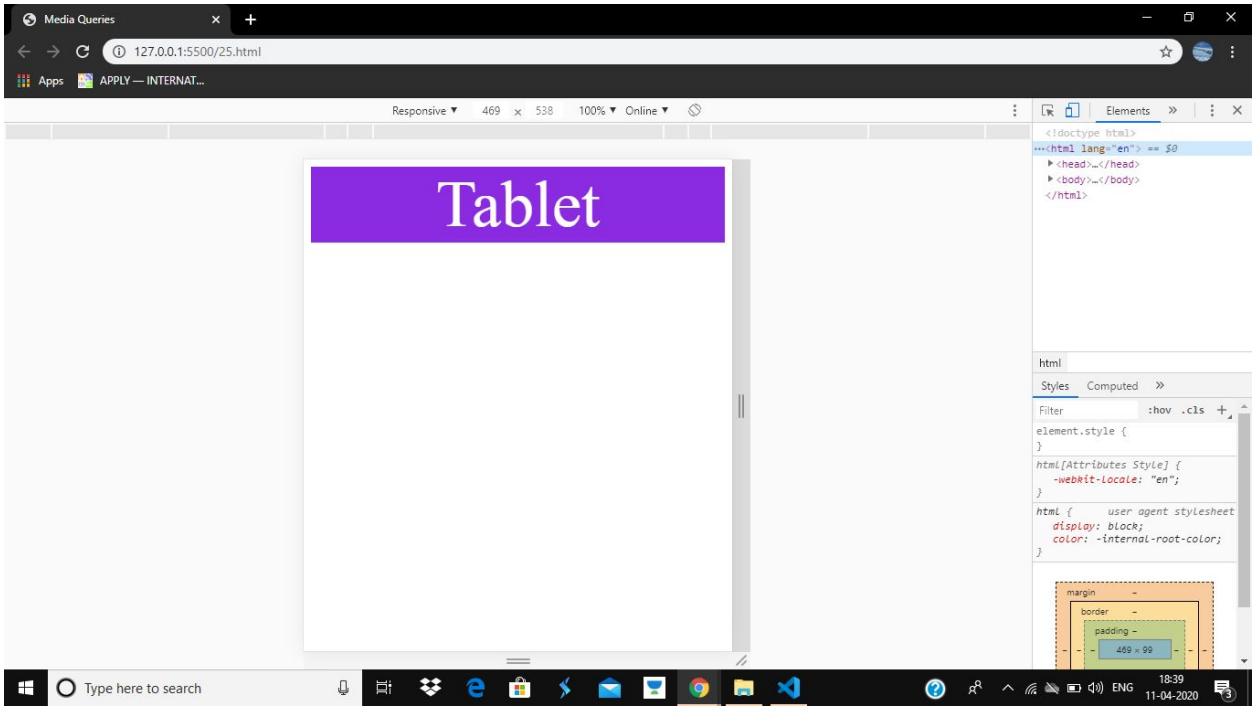
```
@media only screen and (max-width: 600px) {
  body {
        background-color: lightblue;
  }
}
```

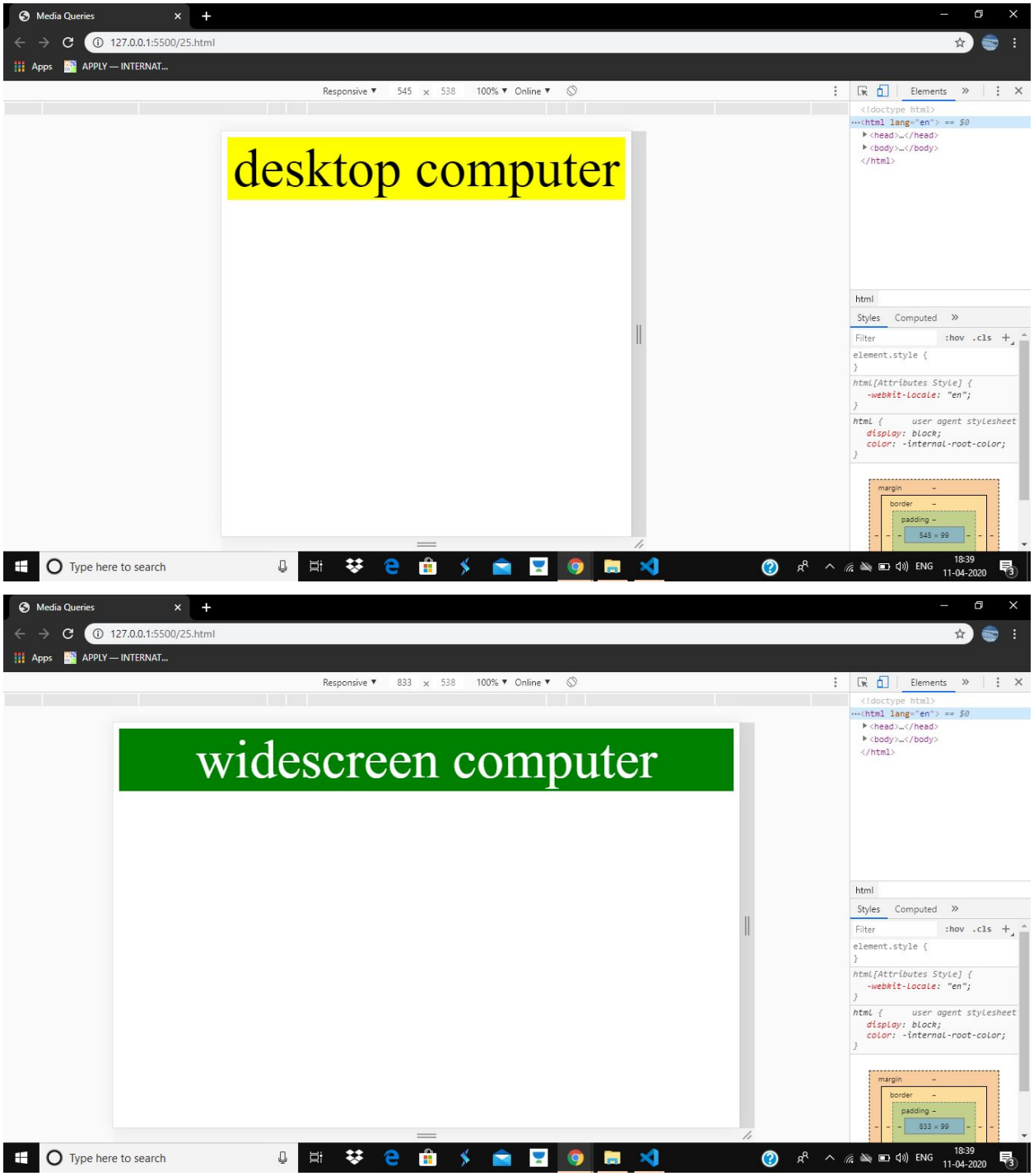
SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Media Queries</title>
    <style>
        .box{
            font-size: 72px;
            text-align: center;
            background-color: red;
            color: white;
            display: none;
        }

        @media only screen and (max-width:300px){
            #box-1{
                display: block;
                background-color: cyan;
```

```
            }
        }

        @media only screen and (min-width: 300px) and (max-width:500px) {
            #box-2{
                display: block;
                background-color: blueviolet;
            }
        }

        @media (min-width: 500px) and (max-width:800px) {
            #box-3{
                display: block;
                color: black;
                background-color: yellow;
            }
        }

        @media (min-width: 800px)  {
            #box-4{
                display: block;
                background-color: green;
            }
        }

    </style>
</head>
<body>
    <div class="box" id="box-1"> iPhone</div>
    <div class="box" id="box-2">Tablet </div>
    <div class="box" id="box-3"> desktop computer </div>
    <div class="box" id="box-4">widescreen computer</div>
</body>
</html>
```

OUTPUT:-


Media Queries
127.0.0.1:5500/25.html
Apps
APPLY — INTERNAT...
Responsive
469 × 538
100% Online
Tablet
Elements
<!doctype html>
<html lang="en"> == $0
<head>…</head>
<body>…</body>
</html>
html
Styles
Computed
Filter
:hov .cls
element.style {
}
html[Attributes Style] {
-webkit-locale: "en";
}
html { user agent stylesheet
display: block;
color: -internal-root-color;
}
margin
border
padding
469 × 99
Type here to search
ENG
18:39
11-04-2020



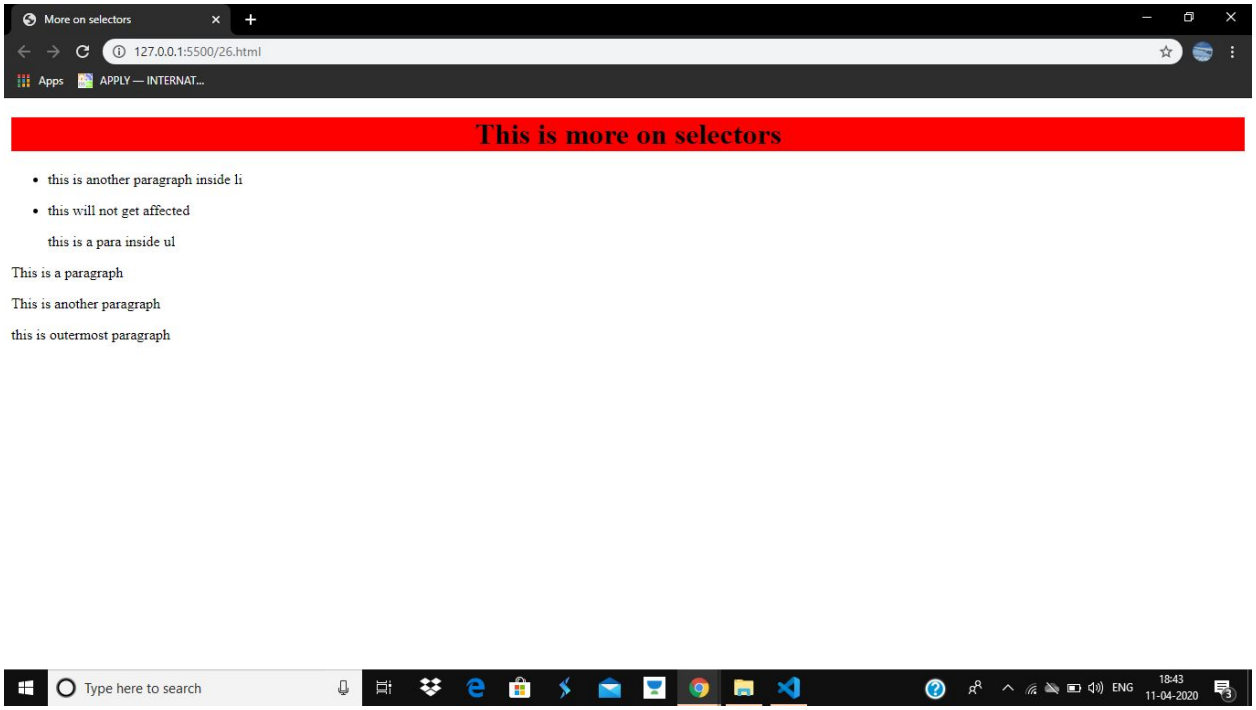
1.2.11 More on CSS Selectors

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>More on selectors</title>
</head>
<style>
    h1{
        background-color: red;
        color: black;
        font-weight: bold;
        text-align: center;
    }

    /* if p is contained by any li which is contained by div */
    /* div li p{
        color: yellow;
        background-color: green;
        font-weight: bold;
    } */

    /* if p is right inside div then this CSS will be applied */
    /* div > p{
        color: yellow;
        background-color: green;
        font-weight: bold;
    } */

    /* if p is right after div i.e p is the next sibling of div*/
    /* div + p{
        color: white;
        background-color: rgb(238, 137, 137);
    } */

</style>
```

```
<body>
    <h1>This is more on selectors</h1>
    <div class="container">
        <div class="row">
            <ul>
                <li class="item"><p> this is another paragraph inside
li</p></li>
                <li>this will not get affected</li>
                <p>this is a para inside ul</p>
            </ul>
            <p>This is a paragraph</p>
        </div>
        <p>This is another paragraph</p>
    </div>
    <p>this is outermost paragraph</p>
</body>
</html>
```

OUTPUT:-



1.2.11.1 Attribute and nth child Pseudo Selectors

Attribute Selector:- The [attribute] selector is used to select elements with a specified attribute.

```
a[target] {
  background-color: yellow;
}
```

Nth child Pseudo Selector:-The nth child selector matches every element that is the *n*th child, regardless of type, of its parent. *n* can be a number, a keyword, or a formula.

Syntax:-

```
:nth-child(number) {
  css declarations;
}
```

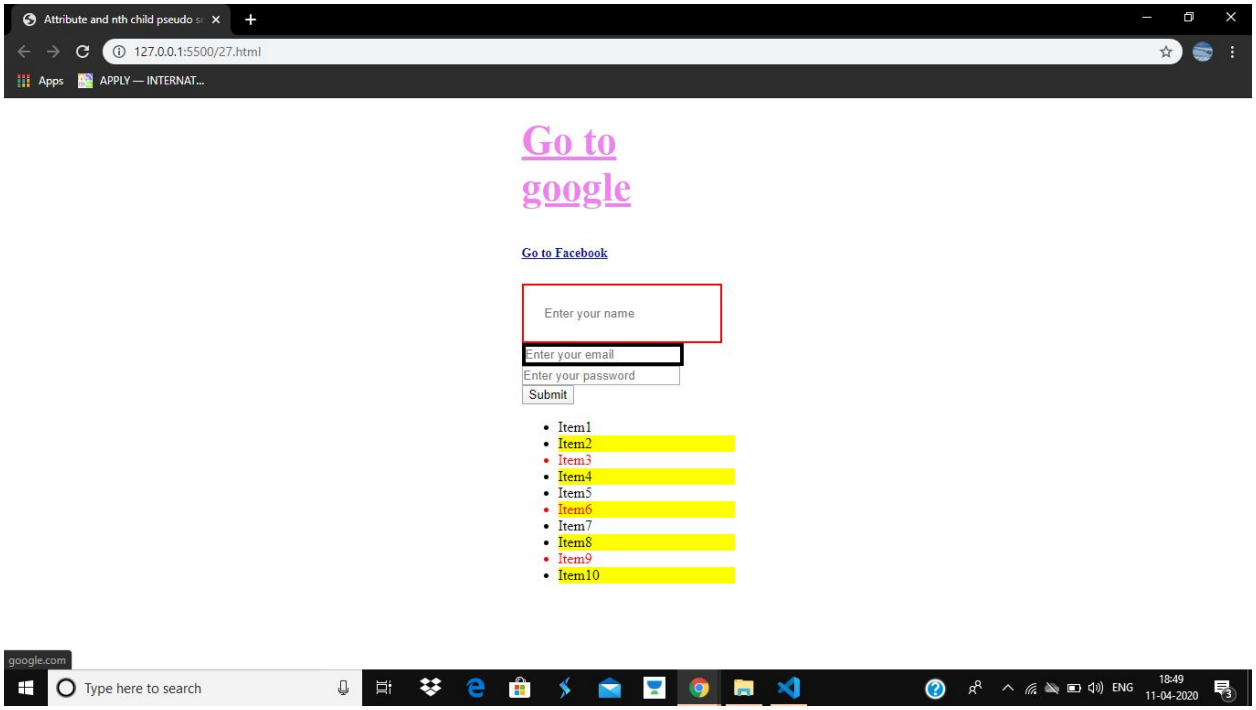
SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Attribute and nth child pseudo selectors</title>
    <style>
        .container {
            display: block;
            width: 233px;
            margin: auto;
        }

        input {
            display: block;
        }

        input[type='text'] {
            padding: 23px;
```

```
            border: 2px solid red;
        }

        a[target] {
            font-size: 44px;
            color: violet;
        }

        a[target='_self'] {
            font-size: 14px;
            color: rgb(13, 22, 151);
        }

        input[type='email'] {
            color: yellow;
            border: 4px solid black;
        }

        /* This will apply css for third child */
        /* li:nth-child(3){
        color: cyan;
    }

    li:nth-child(2n+0){
        color: red;
    } */

        li:nth-child(3n+3) {
            color: red;
        }

        /* Odd child */
        /* li:nth-child(odd){
        background-color: yellow;
    } */

        /* Even child */
```

```
        li:nth-child(even) {
            background-color: yellow;
        }
    </style>
</head>

<body>
    <div class="container">
        <h1><a href="http://google.com" target="_blank">Go to
google</a></h1>
        <h1><a href="http://facebook.com" target="_self">Go to
Facebook</a></h1>
        <form action="" class="form-control">
            <input type="text" placeholder="Enter your name">
            <input type="email" placeholder="Enter your email">
            <input type="password" placeholder="Enter your password">
            <input type="submit" value="Submit">
        </form>
        <ul>
            <li class="item" id="item-1">Item1</li>
            <li class="item" id="item-2">Item2</li>
            <li class="item" id="item-3">Item3</li>
            <li class="item" id="item-4">Item4</li>
            <li class="item" id="item-5">Item5</li>
            <li class="item" id="item-6">Item6</li>
            <li class="item" id="item-7">Item7</li>
            <li class="item" id="item-8">Item8</li>
            <li class="item" id="item-9">Item9</li>
            <li class="item" id="item-10">Item10</li>
        </ul>
    </div>
</body>

</html>
```

OUTPUT:-



1.2.11.2 Before and After Pseudo Selectors

Before:The ::before selector inserts something before the content of each selected element(s).

Syntax:-

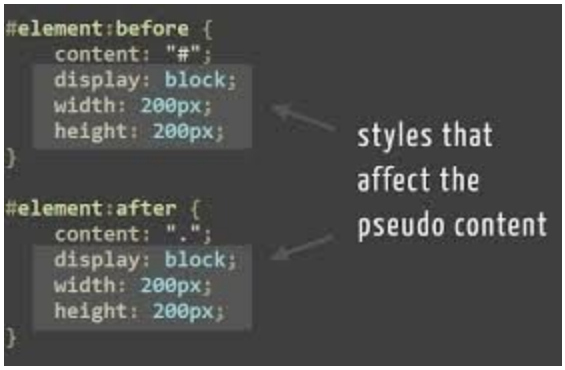
```
::before {
  css declarations;
}
```

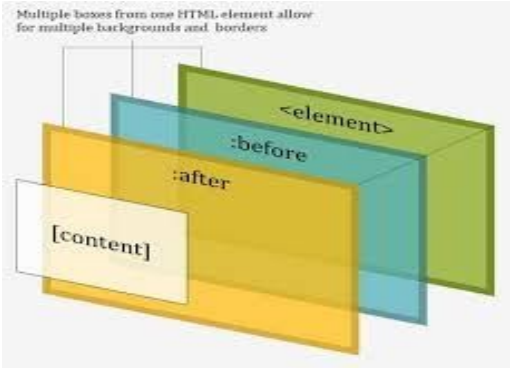
Use the content property to specify the content to insert.

After:-Use the ::after selector to insert something after the content.

Syntax:-

```
::after {
  css declarations;
}
```





SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Before and after pseudo selector</title>
    <link
href="https://fonts.googleapis.com/css?family=Bree+Serif&display=swap"
rel="stylesheet">
    <style>
        body {
            margin: 0;
            padding: 0;
            background-color: black;
            color: white;
        }

        header::before{
            background:
url('https://source.unsplash.com/collection/190727/1600x900') no-repeat
center center/cover;
            content: "";
            position: absolute;
```

```
        top:0;
        left: 0;
        width: 100%;
        height: 100%;
        z-index: -1;
        opacity: 0.3;
    }

    .navigation {
        font-family: 'Bree Serif', serif;
        font-size: 20px;
        display: flex;
    }

    li {
        list-style: none;
        padding: 20px 23px;
    }

    section {
        height: 344px;
        font-family: 'Bree Serif', serif;
        margin: 3px 230px;
        display: flex;
        flex-direction: column;
        align-items: center;
        justify-content: center;
    }

    h1 {
        font-size: 4rem;
    }

    p {
        text-align: center;
    }
```

```
        /* section::after{
        content:"this is a content"
    } */

    </style>
</head>

<body>
    <header>
        <nav class="navbar">
            <ul class="navigation">
                <li class="item">Home</li>
                <li class="item">About</li>
                <li class="item">Services</li>
                <li class="item">Contact Us</li>
            </ul>
        </nav>
    </header>
        <section>
            <h1> Welcome to Coding World</h1>
            <p>Lorem ipsum dolor, sit amet consectetur adipisicing elit.
Provident error ratione doloribus sed dolorum,
                ipsum cumque reprehenderit dignissimos architecto veniam
optio sint aliquam consectetur corrupti vero
                similique velit. Possimus eum consequatur delectus quia
magni.</p>
        </section>

</body>

</html>
```

OUTPUT:-

1.2.12 Shadow

With CSS you can add shadow to text and to elements.
Two type of shadows:

- Box shadow
- Text shadow

1.2.12.1 Box shadow

The CSS box-shadow property applies shadow to elements.

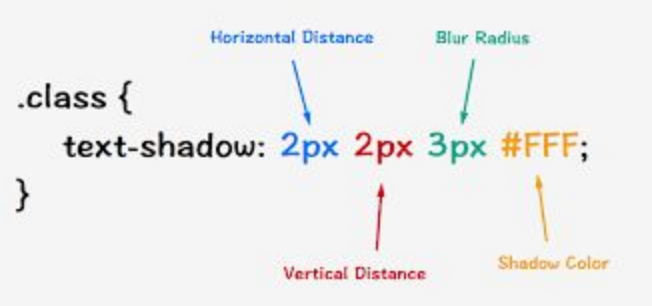
```
div {
  box-shadow: 10px 10px grey;
}
```

1.2.12.2 Text shadow

The CSS text-shadow property applies shadow to text.

Example:-

```
h1 {
  text-shadow: 2px 2px;
}
```



SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Box shadow and text shadow</title>
</head>
<style>
.container{
    display: flex;
}
.card{
    padding: 23px 12px;
    margin: 23px 12px;
    /* border: 2px solid red; */
    background-color: burlywood;
    /* box-shadow: offset-x offset-y color; */
    /* box-shadow: offset-x offset-y blur-radius color; */
    /* box-shadow: offset-x offset-y blur-radius spread-radius color; */

    /* box-shadow: 10px 13px green; */
    /* box-shadow: -10px -13px green; */
    /* box-shadow: 7px 5px 10px green;
    box-shadow: -7px -5px 10px green; */
    /* box-shadow: -7px -5px 10px 34px green; */
    /* box-shadow: -7px -5px 10px 34px rgba(71, 172, 172, 0.5); */
    box-shadow: inset 3px 5px green;

    box-shadow: 3px 5px green, 4px 6px red;
}
.card h2{
    /* text-shadow: 3px 4px red; */
    /* text-shadow: 3px 2px 2px white; */
    text-shadow: -3px -2px 2px white;
}
</style>
<body>
    <div class="container">
        <div class="card" id="card-1">
            <h2>iuef</h2>
```

```
            <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
Molestiae velit saepe atque tenetur odit, illum eius perspiciatis iste
fuga debitis est nulla dolor alias sed quam ipsa earum! Eum aliquam saepe
delectus odit maxime.</p>
        </div>
        <div class="card" id="card-2">
            <h2>qwhhfu</h2>
            <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
Veritatis placeat doloribus molestiae velit ipsum, aliquam officia ratione
excepturi in officiis, incidunt quo est pariatur tempore ex, distinctio
nostrum! Sint non doloribus rem obcaecati sunt.</p>
        </div>
        <div class="card" id="card-3">
           <h2>euwfu</h2>
           <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. In
tenetur molestiae, placeat quas perferendis quibusdam atque omnis
distinctio obcaecati dolor, tempora unde deserunt iure nam. Iste labore
eveniet esse deserunt?</p>
        </div>
    </div>
</body>
</html>
```

OUTPUT:-

1.2.13 Variables and Custom Properties

The var() function can be used to insert the value of a custom property.Variables in CSS should be declared within a CSS selector that defines its scope. For a global scope you can use either the :root or the body selector.The variable name must begin with two dashes (--) and is case sensitive!

Syntax:-

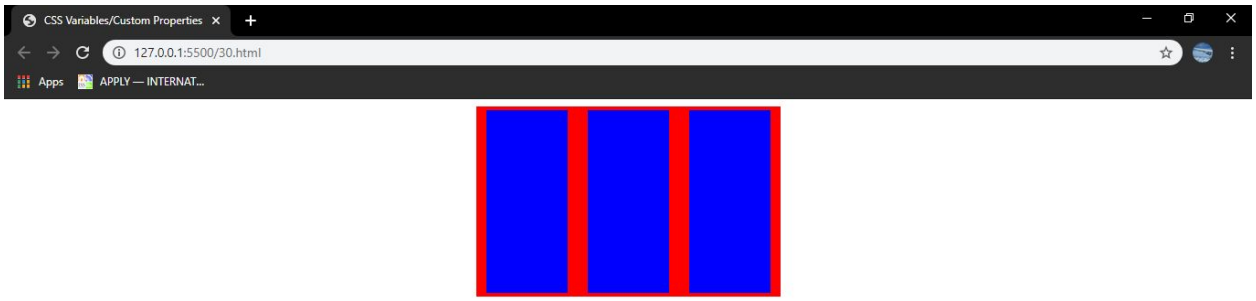
var(*custom-name, value*)

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>CSS Variables/Custom Properties</title>
    <style>
        :root{
            --primary-color: blue;
            --danger-color: red;
            --maxw: 333px;
        }
        .box{
            width:200px;
            height: 200px;
            background-color: var(--primary-color);
            border: 2px solid var(--danger-color);
            box-shadow: 3px 3px var(--box-color);
            margin: 2px 9px;
        }
        .container{
            max-width: var(--maxw);
            margin: auto;
            background-color: var(--danger-color);
            display: flex;
            align-items: center;
            justify-content: center;
            /* background-color: var(--box-color); */
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="box"></div>
        <div class="box"></div>
        <div class="box"></div>
```

```
    </div>
</body>
</html>
```

OUTPUT:-



1.2.14 Animations and Keyframes

CSS allows animation of HTML elements without using JavaScript or Flash! An animation lets an element gradually change from one style to another.

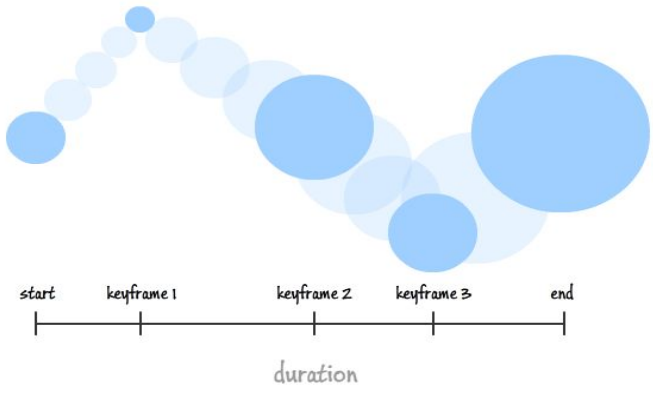
To use CSS animation, we must first specify some keyframes for the animation.Keyframes hold what styles the element will have at certain times.

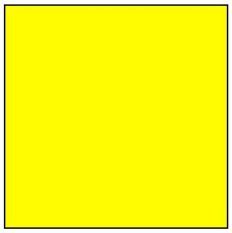
The @keyframes Rule

When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.

To get an animation to work, you must bind the animation to an element.

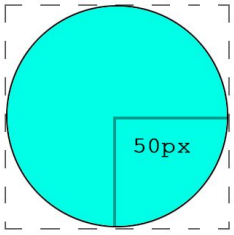
The animation-duration property defines how long an animation should take to complete. If the animation-duration property is not specified, no animation will occur, because the default value is 0s.





Orignal

```
width: 100px;
height: 100px;
border: 1px solid black;
background: yellow;
```



Target

```
border-radius: 50px;
background: teal;
```

```
@keyframes animationName {
    0% { border-radius: 0; background: yellow; }
  100% { border-radius: 50px; background: teal; }
}

.animateClass {
    animation-name: animationName;
    animation-fill-mode: forwards;
    animation: normal 3000ms ease;
}
```

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Keyframes and Animations</title>
</head>
<style>
    .container {
        background-color: greenyellow;
    }

    .box {
        background-color: green;
        width: 250px;
        height: 250px;
```

```
        position: relative;
        /* animation-name: harry1; */
        animation-name: harry2;
        animation-duration: 8s;
        animation-iteration-count: 1;
        /* animation-fill-mode: alternate; */
        /* animation-timing-function: ease-in-out; */
        /* animation-delay: 3s; */
        /* animation-direction: reverse;  */

        /* These properties can be set using this shorthand */
                        /*   animation:   animation-name   animation-duration
animation-timing-function     animation-delay     animation-iteration-count
animation-fill-mode; */
        /* animation: harry 5s ease-in 1s 12 backwards; */

    }

    @keyframes harry2 {
        0%{
            top:0px;
            left:0px;
        }
        25%{
            top: 250px;
            left: 0px;
        }
        75%{
            top: 250px;
            left: 250px;
        }
        100%{
            top: 0px;
            left: 250px;
        }

    }
```

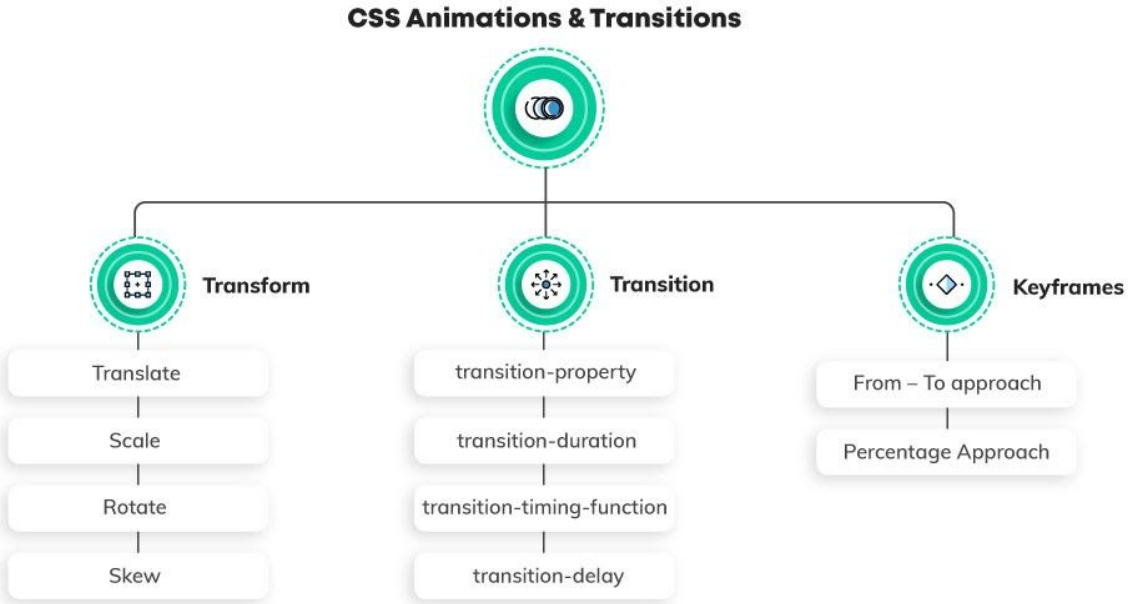
```
    @keyframes harry1 {
        from {
            width: 200px;
        }

        to {
            width: 1400px;
        }
    }
</style>

<body>
    <div class="container">
        <div class="box">
            This is a box
        </div>
    </div>
</body>

</html>
```

1.2.15 Transitions

CSS transitions allows you to change property values smoothly, over a given duration.To create a transition effect, must specify two things:

- the CSS property you want to add an effect to
- the duration of the effect

If the duration part is not specified, the transition will have no effect, because the default value is 0.The transition effect will start when the specified CSS property changes value.



Properties:-

- Transition-delay

The transition-delay property specifies a delay (in seconds) for the transition effect.

Example:-

```
div {
  transition-delay: 1s;
}
```

- Transition-duration

The transition-duration property specifies how many seconds (s) or milliseconds (ms) a transition effect takes to complete.

Syntax:-

transition-duration: time | initial | inherit ;

- Transition-property

The transition-property property specifies the name of the CSS property the transition effect is for (the transition effect will start when the specified CSS property changes).

A transition effect could typically occur when a user hover over an element.

Syntax :-

transition-property: none | all | property | initial | inherit ;

- Transition-timing-function

The transition-timing-function property specifies the speed curve of the transition effect.The transition-timing-function property can have the following values:

- ease - specifies a transition effect with a slow start, then fast, then end slowly (this is default)
- linear - specifies a transition effect with the same speed from start to end
- ease-in - specifies a transition effect with a slow start
- ease-out - specifies a transition effect with a slow end
- ease-in-out - specifies a transition effect with a slow start and end

- cubic-bezier(n,n,n,n) - lets you define your own values in a cubic-bezier function

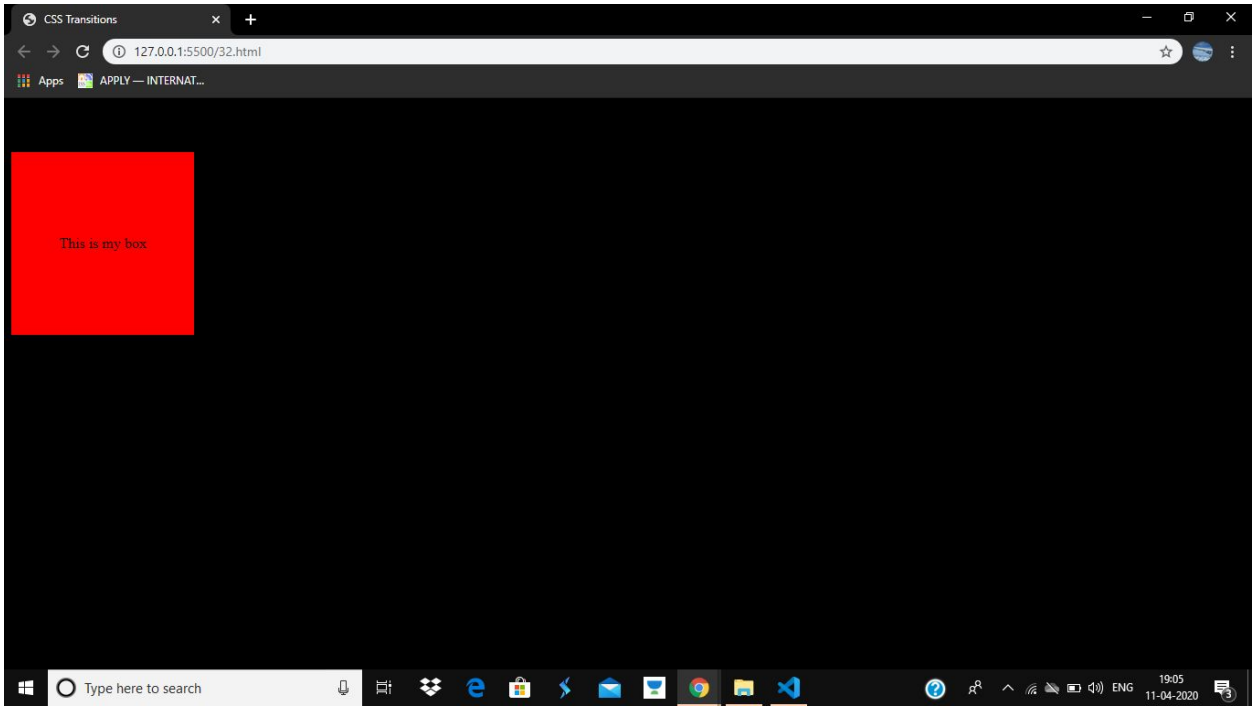
SOURCE CODE:-

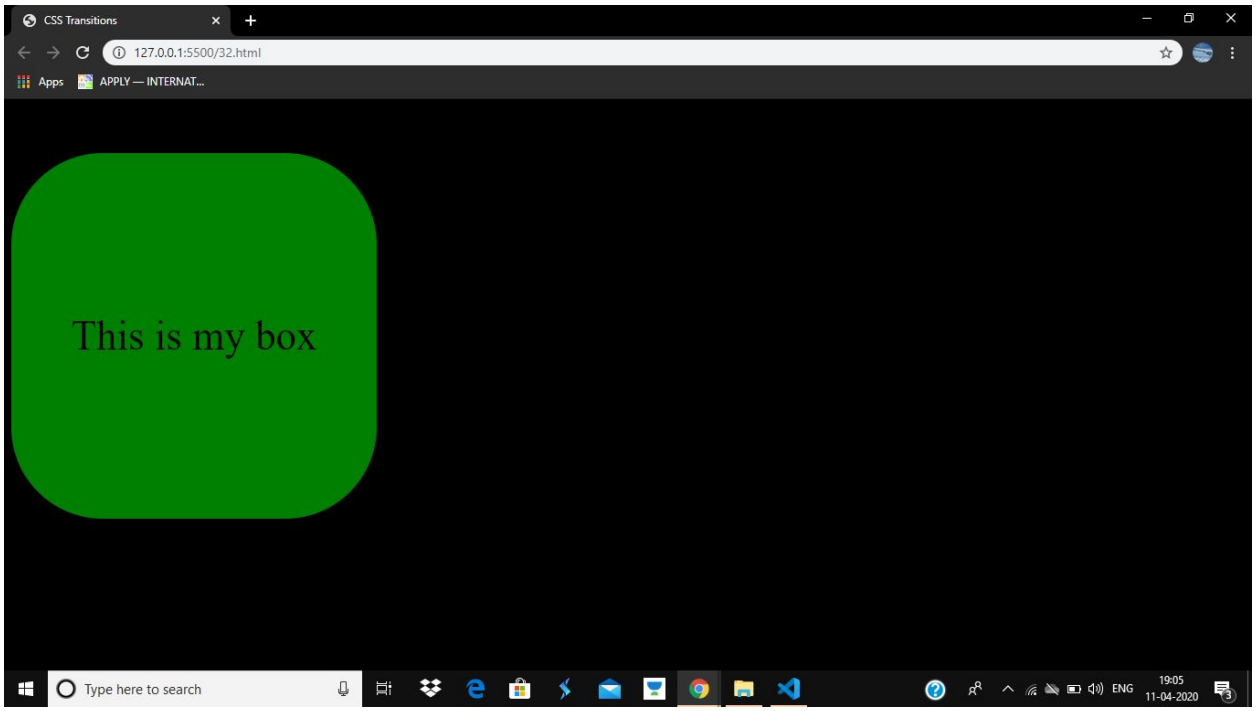
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>CSS Transitions</title>
</head>
<style>
    body{
        background-color: black;
    }
    #box{
        display: flex;
        height: 200px;
```

```
        width: 200px;
        background-color: red;
        justify-content: center;
        align-items: center;
        /* transition-property: background-color;
        transition-duration: 1s;
        transition-timing-function: ease-in-out;
        transition-delay: 2s; */

        /* Transition short hand property in one line */
        /* transition: background-color 1s ease-in-out 2s; */


        transition: all 1s ease-in-out .3s;
    }

    #box:hover{
        background-color: green;
        height: 400px;
        width: 400px;
        border-radius: 100px;
        font-size: 45px;
```

```
    }


</style>
<body>
    <h3>This is CSS Transition </h3>
    <div class="container">
        <div id="box">
            This is my box
        </div>
    </div>
</body>
</html>
```

OUTPUT:-


CSS Transitions
127.0.0.1:5500/32.html
Apps
APPLY — INTERNAT...
This is my box
Type here to search
ENG
19:05
11-04-2020


CSS Transitions
127.0.0.1:5500/32.html
Apps
APPLY — INTERNAT...
This is my box
Type here to search
ENG
19:05
11-04-2020

1.2.16 Transform property

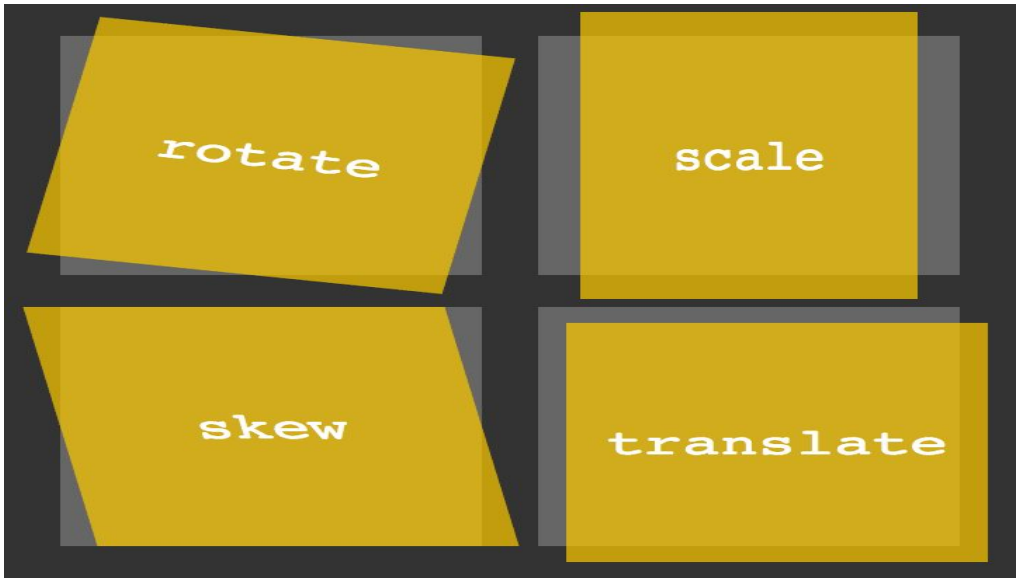
The transform property applies a 2D or 3D transformation to an element. This property allows you to rotate, scale, move, skew, etc., elements.

Syntax:-

```
transform: none | transform-functions | initial | inherit ;
```

Example:-

```
div.a {
  transform: rotate(20deg);
}

div.b {
  transform: skewY(20deg);
}

div.c {
  transform: scaleY(1.5);
}
```



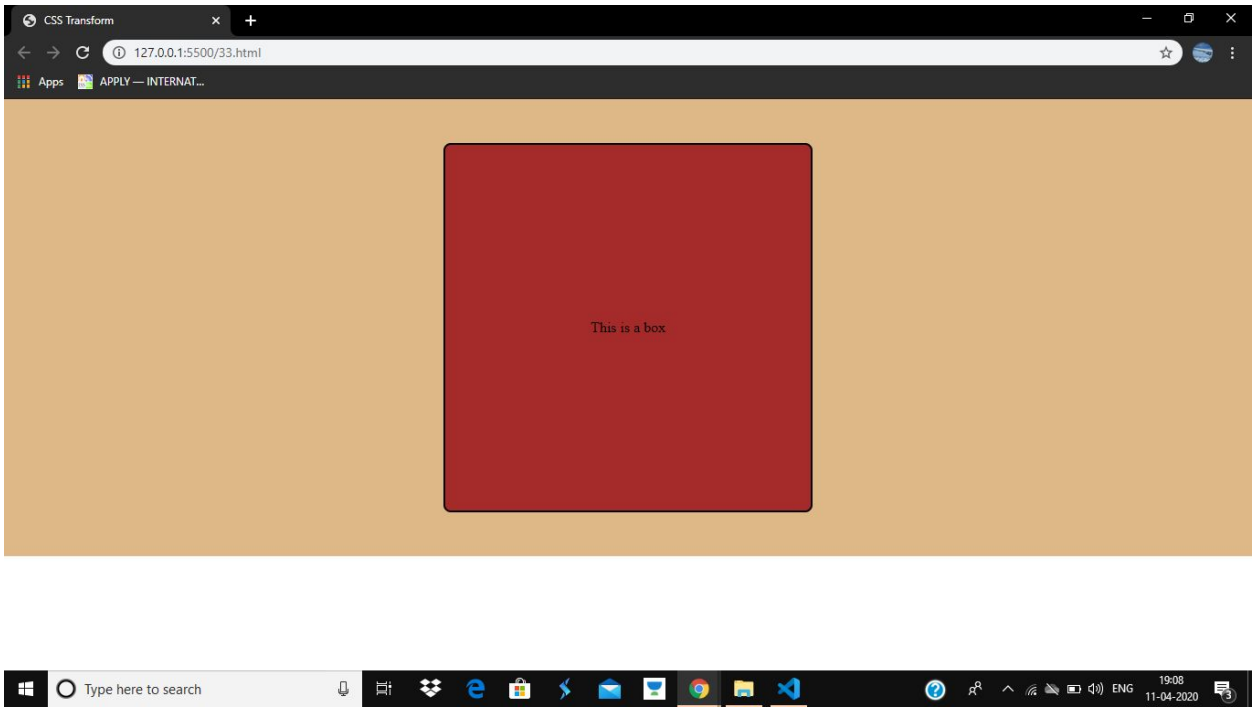
SOURCE CODE:-

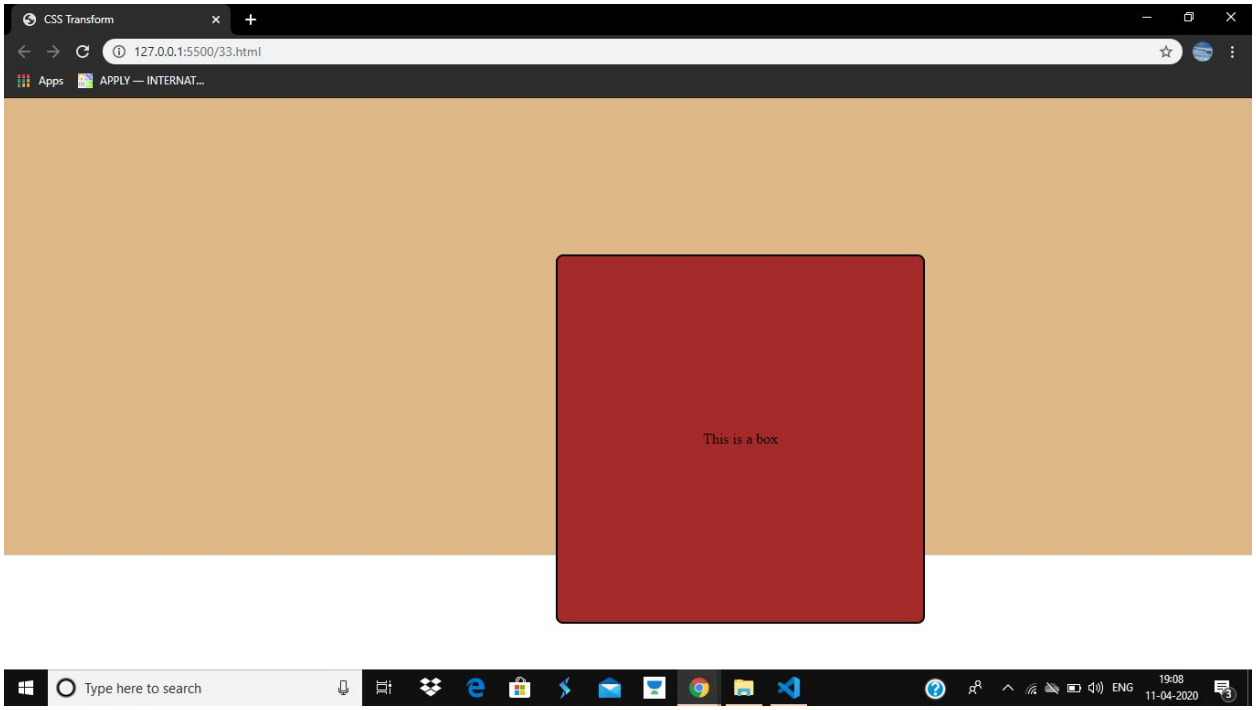
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>CSS Transform</title>
    <style>
        *{
            margin: 0px;
            padding: 0px;
        }
        .container{
            height: 80vh;
            background-color: burlywood;
            display: flex;
            justify-content: center;
            align-items: center;

        }
        .box{
            display: flex;
            align-items: center;
```

```
            justify-content: center;
            background: brown;
            border: 2px solid black;
            border-radius: 8px;
            height: 400px;
            width: 400px;
            transition: all 0.5s ease-in-out;
        }
        .box:hover{
            /* transform: rotate(360deg); */
            /* transform: skew(40deg); */
            /* transform: scale(2); */
            /* transform: translateX(123px); */
            /* transform: translateY(123px); */
            transform: translate(123px, 123px);


        }
    </style>
</head>
<body>
    <div class="container">
        <div class="box">This is a box</div>
    </div>
```

```
</body>

</html>
```

OUTPUT:-

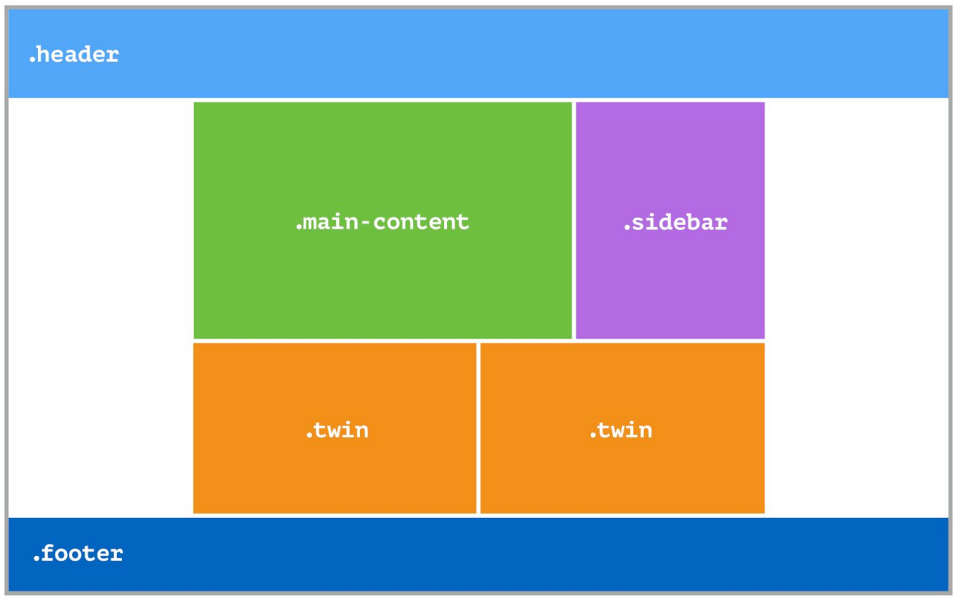




1.3 CSS Grid

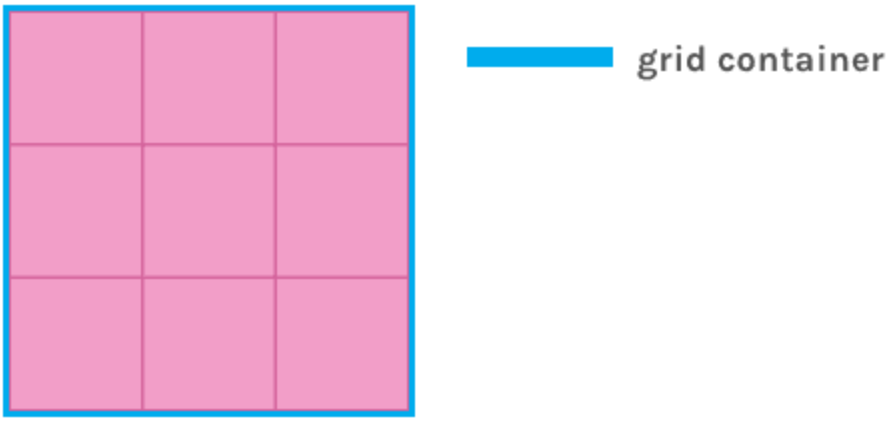
1.3.1 Introduction

CSS Grid Layout (aka “Grid”), is a two-dimensional grid-based layout system that aims to do nothing less than completely change the way we design grid-based user interfaces. CSS has always been used to lay out our web pages, but it’s never done a very good job of it. First, we used tables, then floats, positioning and inline-block, but all of these methods were essentially hacks and left out a lot of important functionality (vertical centering, for instance). Flexbox helped out, but it’s intended for simpler one-dimensional layouts, not complex two-dimensional ones (Flexbox and Grid actually work very well together). Grid is the very first CSS module created specifically to solve the layout problems we’ve all been hacking our way around for as long as we’ve been making websites.



1.3.2 Creating basic Grid

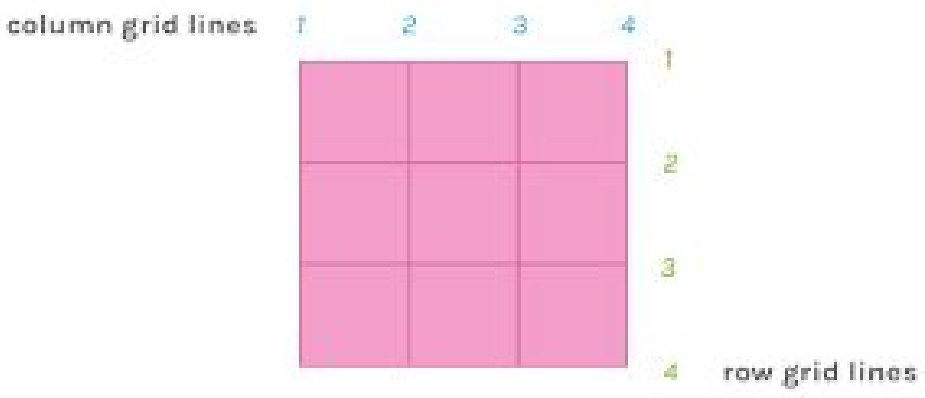
The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.A grid layout consists of a parent element, with one or more child elements.



An HTML element becomes a grid container when its display property is set to grid or inline-grid.

Example:-

```
.grid-container {
  display: grid;
}
```



Grid Columns :-The vertical lines of grid items are called *columns*.

Grid Rows:-The horizontal lines of grid items are called *rows*.

Grid Gaps:-The spaces between each column/row are called *gaps*.

You can adjust the gap size by using one of the following properties:

- grid-column-gap
- grid-row-gap
- grid-gap

Grid Lines:-

The lines between columns are called *column lines*.
The lines between rows are called *row lines*.

SOURCE CODE:-

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>CSS Grid </title>

    <style>
```

```
    .container{
        display: grid;
        /* grid-template-columns: 300px 100px 100px; */
        /* grid-template-columns: 300px auto 100px; */
        /* grid-template-columns: 1fr 4fr 1fr; */
        grid-template-columns: repeat(3, auto);
        grid-gap: 2rem;
    }
    .item{
        /* height: 100px;
        width: 100px; */
        background-color: red;
        border: 2px solid black;
        padding: 15px 5px;
    }
    </style>
</head>
<body>
    <div class="container">
        <div class="item">This is Item-1</div>
        <div class="item">This is Item-2</div>
        <div class="item">This is Item-3</div>
        <div class="item">This is Item-4</div>
        <div class="item">This is Item-5</div>
        <div class="item">This is Item-6</div>
        <div class="item">This is Item-7</div>
```

```
        <div class="item">This is Item-8</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
        <div class="item">This is Item-9</div>
    </div>
</body>
</html>
```

OUTPUT:-

1.3.3 Spanning Multiple Rows and Columns

While placing items to the parent grid with grid-column or grid-row, we can use the span keyword to avoid specifying end lines when items should span more than one column or row.

Example: spans 2 rows and 3 columns

```
.item {
  grid-column: 2 / 5;
  grid-row: 1 / 3;
}
```

Using span keyword:-

```
.item {
  grid-column: 2 / span 3;
  grid-row: 1 / span 2;
}
```

If multiple lines have same name,we can define the start and end lines like this:-

```
.item {
  grid-column: col 2 / col 7;
  grid-row: content 6 / content 10;
```

```
}
```

The items starts on the column with second line named col and ends in 7th line also named col.Similarly it starts on 6th line named row and ends on 10th line named row:-

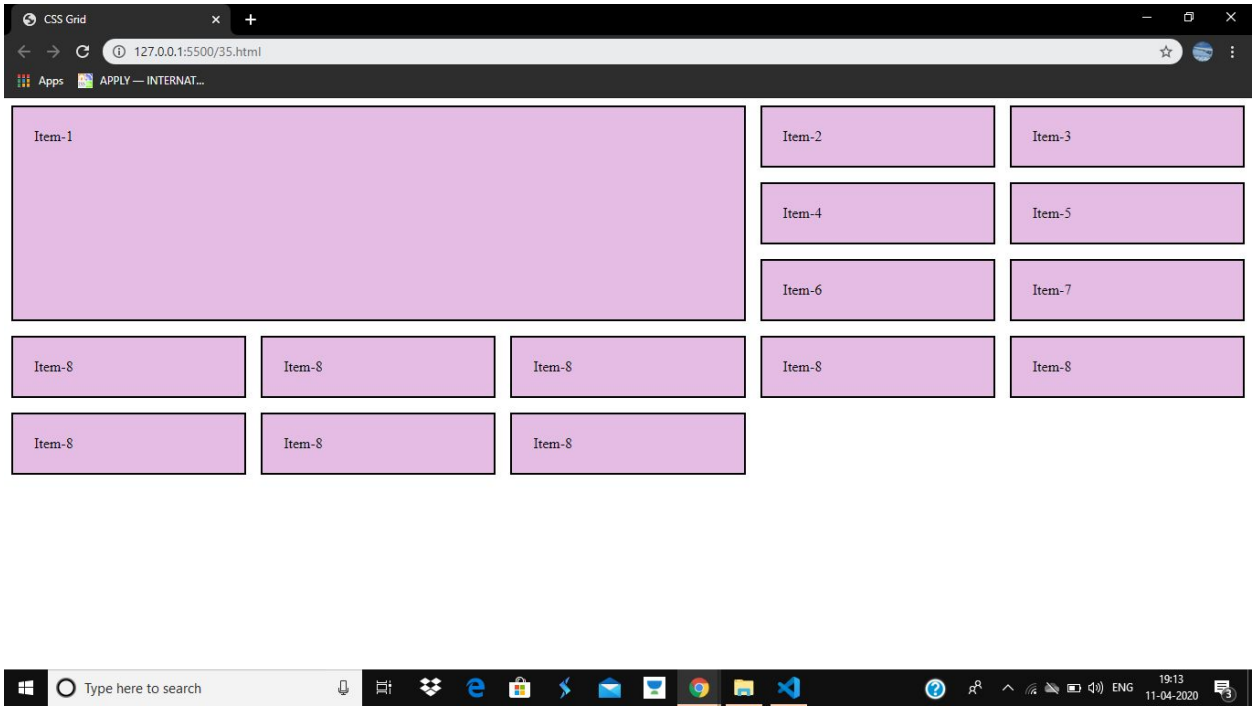
```
.item {
  grid-column: col 2 / span col 5;
  grid-row: content 6 / span content 4;
}
```

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>CSS Grid</title>
    <style>
    .container{
        display: grid;
        grid-template-columns: repeat(5, 1fr);
        grid-template-rows: repeat(4, 1fr);
        /* grid-column-gap: 7rem;
        grid-row-gap: 1rem; */
        grid-gap: 1rem;


    }
    .box{
        border: 2px solid black;
        background-color: rgb(228, 188, 228);
        padding: 23px;
    }
    .box:first-child{
        /* grid-column-start: 1;
        grid-column-end: 3;
```

```
        grid-row-start: 1;
        grid-row-end: 3; */
        grid-column: 1 / span 3;
        grid-row: 1 / span 3;
    }
    </style>
</head>
<body>
    <div class="container">
        <div class="box">Item-1</div>
        <div class="box">Item-2</div>
        <div class="box">Item-3</div>
        <div class="box">Item-4</div>
        <div class="box">Item-5</div>
        <div class="box">Item-6</div>
        <div class="box">Item-7</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
    </div>
</body>
</html>
```

OUTPUT:-



1.3.4 Autofill/Autofit and Minmax

To achieve wrapping, we use the auto-fit or auto-fill keywords.

Example:-

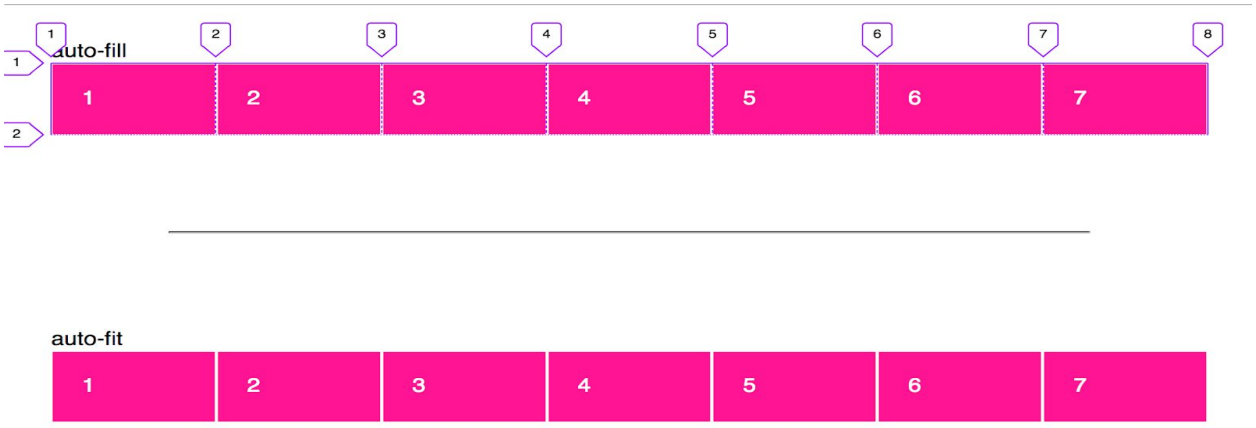
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

These keywords tell the browser to handle the column sizing and element wrapping , so that the elements will wrap into rows when the width is not large enough to fit them in without any overflow. The fraction unit we used also ensures that, in case the width allows for a fraction of a column to fit but not a full column, that space will instead be distributed over the column or columns that already fit, making sure we aren't left with any empty space at the end of the row.

At first glance of the names, it might seem like auto-fill and auto-fit are opposites. But in fact, the difference between is quite subtle.

auto-fill FILLS the row with as many columns as it can fit. So it creates implicit columns whenever a new column can fit, because it's trying to FILL the row with as many columns as it can. The newly added columns can and may be empty, but they will still occupy a designated space in the row.

auto-fit FITS the CURRENTLY AVAILABLE columns into the space by expanding them so that they take up any available space. The browser does that after FILLING that extra space with extra columns (as with auto-fill) and then collapsing the empty ones.



Minmax:-

The **minmax()** CSS function defines a size range greater than or equal to min and less than or equal to max. It is used with CSS Grids.

Example:-

If we want a column to be between 200px and 500px and then a 1fr column:

```
.container {
  display: grid;
  grid-template-columns: minmax(200px, 500px) 1fr;
}
```

The value for min has to be smaller than the value for max. Fr units can't be used for the min value, but they can absolutely be used for the max value. In fact, using a 1fr as the max value will ensure that the track expands and takes up the available space:

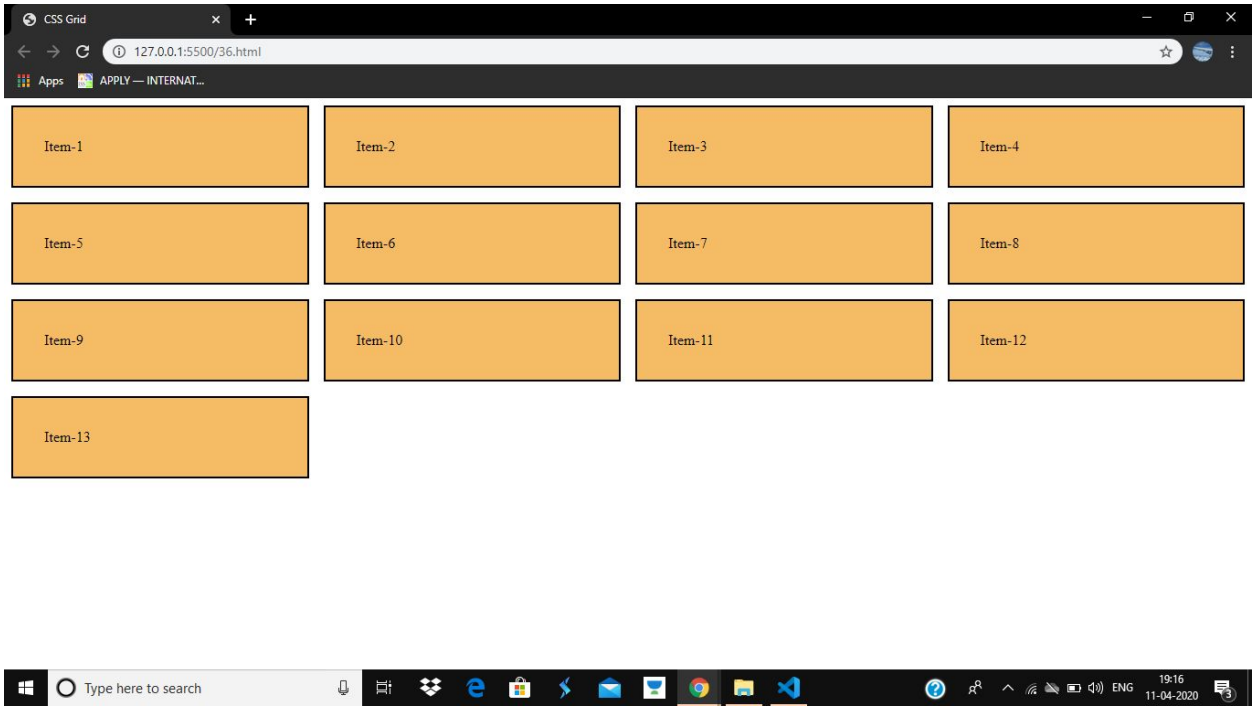
```
.container {
  display: grid;
  grid-template-columns: minmax(250px, 1fr) 1fr;
  grid-template-rows: 100px 100px;
}
```

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>CSS Grid</title>
    <style>
    .container{
        display: grid;
        grid-template-columns: repeat(5, 1fr);
        grid-template-rows: repeat(4, 1fr);
        /* grid-column-gap: 7rem;
        grid-row-gap: 1rem; */
        grid-gap: 1rem;


    }
    .box{
        border: 2px solid black;
        background-color: rgb(228, 188, 228);
        padding: 23px;
    }
    .box:first-child{
        /* grid-column-start: 1;
        grid-column-end: 3;
        grid-row-start: 1;
        grid-row-end: 3; */
        grid-column: 1 / span 3;
        grid-row: 1 / span 3;
    }
    </style>
</head>
<body>
    <div class="container">
```

```
        <div class="box">Item-1</div>
        <div class="box">Item-2</div>
        <div class="box">Item-3</div>
        <div class="box">Item-4</div>
        <div class="box">Item-5</div>
        <div class="box">Item-6</div>
        <div class="box">Item-7</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
        <div class="box">Item-8</div>
    </div>
</body>
</html>
```

OUTPUT:-

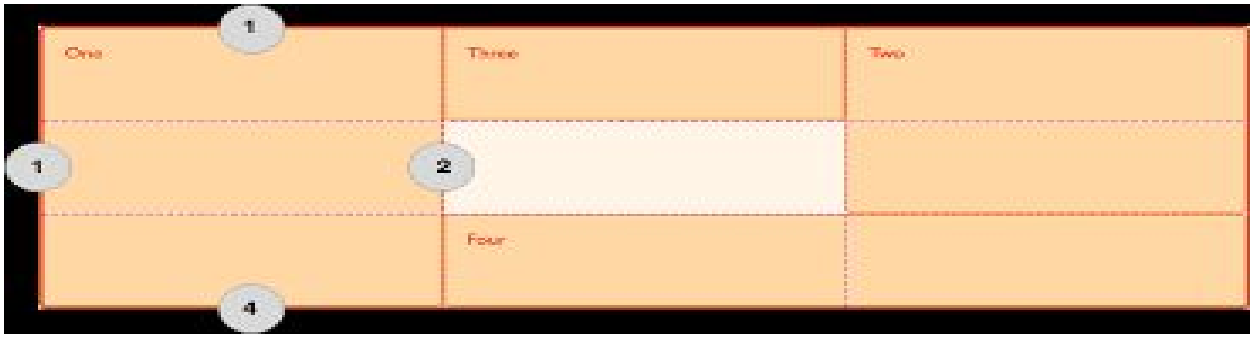


1.3.5 Layout using Grid Template area

The grid-template-areas property specifies areas within the grid layout.

Syntax:-

grid-template-areas: none | item names ;



SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>CSS Grid </title>
    <style>
    .container{
        display: grid;
        grid-gap: 1rem;
        grid-template-areas:
        'navbar navbar navbar navbar'
        'section section section aside'
        'footer footer footer footer ';
    }
    .item{
        background-color: yellow;
```

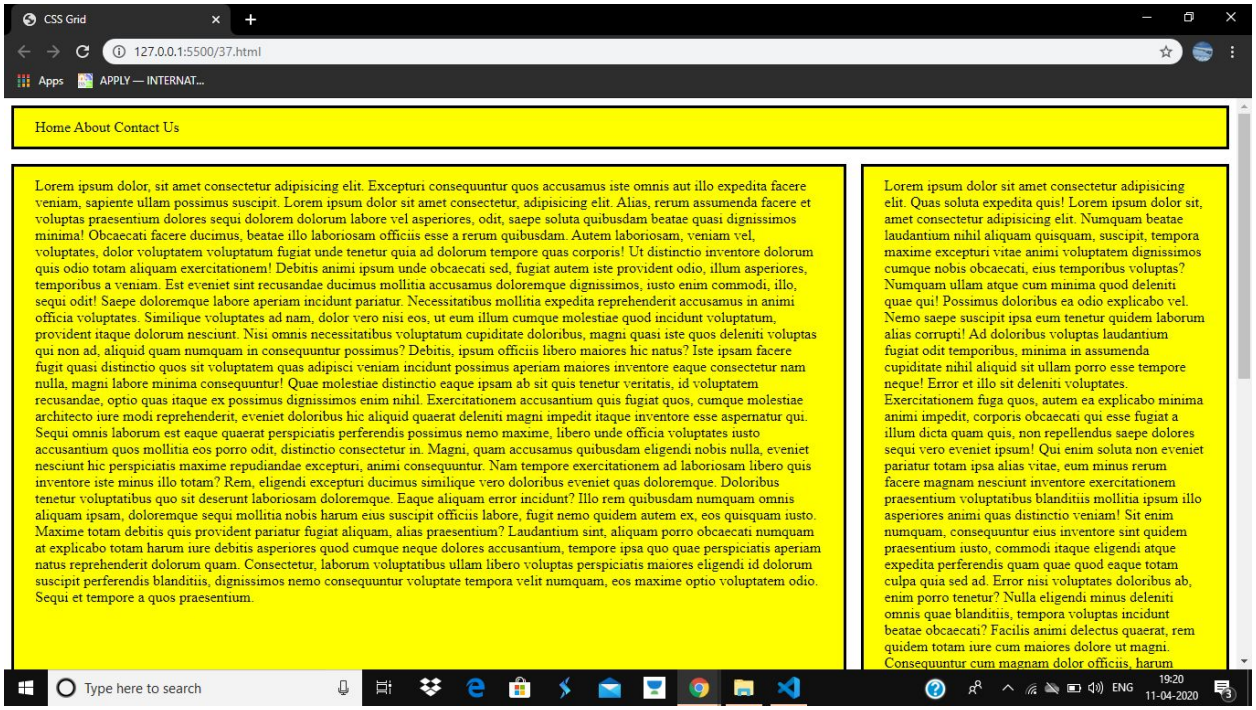
```
        border: 3px solid black;
        padding: 12px 23px;
    }
    #navbar{
        grid-area: navbar;
    }
    #section{
        grid-area: section;
    }
    #aside{
        grid-area: aside;
    }
   footer{
        grid-area: footer;
    }
    </style>
</head>
<body>
    <div class="container">
        <div id="navbar" class="item">

            Home About Contact Us
        </div>
        <div id="section" class="item">
            Lorem ipsum dolor, sit amet consectetur adipisicing elit.
Excepturi consequuntur quos accusamus iste omnis aut illo expedita facere
veniam, sapiente ullam possimus suscipit.
            Lorem ipsum dolor sit amet consectetur, adipisicing elit.
Alias, rerum assumenda facere et voluptas praesentium dolores sequi
dolorem dolorum labore vel asperiores, odit, saepe soluta quibusdam beatae
quasi dignissimos minima! Obcaecati facere ducimus, beatae illo laboriosam
officiis esse a rerum quibusdam. Autem laboriosam, veniam vel, voluptates,
dolor voluptatem voluptatum fugiat unde tenetur quia ad dolorum tempore
quas corporis! Ut distinctio inventore dolorum quis odio totam aliquam
exercitationem! Debitis animi ipsum unde obcaecati sed, fugiat autem iste
provident odio, illum asperiores, temporibus a veniam. Est eveniet sint
recusandae ducimus mollitia accusamus doloremque dignissimos, iusto enim
```

```
commodi, illo, sequi odit! Saepe doloremque labore aperiam incidunt
pariatur. Necessitatibus mollitia expedita reprehenderit accusamus in
animi officia voluptates. Similique voluptates ad nam, dolor vero nisi
eos, ut eum illum cumque molestiae quod incidunt voluptatum, provident
itaque dolorum nesciunt. Nisi omnis necessitatibus voluptatum cupiditate
doloribus, magni quasi iste quos deleniti voluptas qui non ad, aliquid
quam numquam in consequuntur possimus? Debitis, ipsum officiis libero
maiores hic natus? Iste ipsam facere fugit quasi distinctio quos sit
voluptatem quas adipisci veniam incidunt possimus aperiam maiores
inventore eaque consectetur nam nulla, magni labore minima consequuntur!
Quae molestiae distinctio eaque ipsam ab sit quis tenetur veritatis, id
voluptatem recusandae, optio quas itaque ex possimus dignissimos enim
nihil. Exercitationem accusantium quis fugiat quos, cumque molestiae
architecto iure modi reprehenderit, eveniet doloribus hic aliquid quaerat
deleniti magni impedit itaque inventore esse aspernatur qui. Sequi omnis
laborum est eaque quaerat perspiciatis perferendis possimus nemo maxime,
libero unde officia voluptates iusto accusantium quos mollitia eos porro
odit, distinctio consectetur in. Magni, quam accusamus quibusdam eligendi
nobis nulla, eveniet nesciunt hic perspiciatis maxime repudiandae
excepturi, animi consequuntur. Nam tempore exercitationem ad laboriosam
libero quis inventore iste minus illo totam? Rem, eligendi excepturi
ducimus similique vero doloribus eveniet quas doloremque. Doloribus
tenetur voluptatibus quo sit deserunt laboriosam doloremque. Eaque aliquam
error incidunt? Illo rem quibusdam numquam omnis aliquam ipsam, doloremque
sequi mollitia nobis harum eius suscipit officiis labore, fugit nemo
quidem autem ex, eos quisquam iusto. Maxime totam debitis quis provident
pariatur fugiat aliquam, alias praesentium? Laudantium sint, aliquam porro
obcaecati numquam at explicabo totam harum iure debitis asperiores quod
cumque neque dolores accusantium, tempore ipsa quo quae perspiciatis
aperiam natus reprehenderit dolorum quam. Consectetur, laborum
voluptatibus ullam libero voluptas perspiciatis maiores eligendi id
dolorum suscipit perferendis blanditiis, dignissimos nemo consequuntur
voluptate tempora velit numquam, eos maxime optio voluptatem odio. Sequi
et tempore a quos praesentium.
        </div>
        <div id="aside" class="item">Lorem ipsum dolor sit amet
consectetur adipisicing elit. Quas soluta expedita quis!
```

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Numquam beatae laudantium nihil aliquam quisquam, suscipit, tempora maxime excepturi vitae animi voluptatem dignissimos cumque nobis obcaecati, eius temporibus voluptas? Numquam ullam atque cum minima quod deleniti quae qui! Possimus doloribus ea odio explicabo vel. Nemo saepe suscipit ipsa eum tenetur quidem laborum alias corrupti! Ad doloribus voluptas laudantium fugiat odit temporibus, minima in assumenda cupiditate nihil aliquid sit ullam porro esse tempore neque! Error et illo sit deleniti voluptates. Exercitationem fuga quos, autem ea explicabo minima animi impedit, corporis obcaecati qui esse fugiat a illum dicta quam quis, non repellendus saepe dolores sequi vero eveniet ipsum! Qui enim soluta non eveniet pariatur totam ipsa alias vitae, eum minus rerum facere magnam nesciunt inventore exercitationem praesentium voluptatibus blanditiis mollitia ipsum illo asperiores animi quas distinctio veniam! Sit enim numquam, consequuntur eius inventore sint quidem praesentium iusto, commodi itaque eligendi atque expedita perferendis quam quae quod eaque totam culpa quia sed ad. Error nisi voluptates doloribus ab, enim porro tenetur? Nulla eligendi minus deleniti omnis quae blanditiis, tempora voluptas incidunt beatae obcaecati? Facilis animi delectus quaerat, rem quidem totam iure cum maiores dolore ut magni. Consequuntur cum magnam dolor officiis, harum ipsum blanditiis dicta amet iusto quos, sed eaque ipsa atque provident voluptas rem. Nesciunt minima esse, facilis recusandae quod adipisci magnam assumenda necessitatibus! Dolor blanditiis exercitationem quas vero officia quia ipsam, vel maiores distinctio minus autem eveniet consequuntur magni. Id nostrum, sunt cum nesciunt dolore quisquam eligendi dolorem recusandae, beatae quis eveniet, aut velit itaque neque necessitatibus quasi laudantium commodi illo adipisci minima! Id suscipit quae omnis. Veritatis, temporibus. Delectus, eligendi! Magnam itaque veritatis reprehenderit eos odit, explicabo recusandae incidunt vitae voluptate saepe hic blanditiis pariatur officia. Necessitatibus suscipit quo nostrum! Expedita nostrum dolore perspiciatis id, itaque repudiandae tempora doloribus amet quas, dicta quam! Nam dolorem voluptatum incidunt iure cum eos enim voluptate, iusto voluptatibus modi repudiandae quo, ipsum, quos qui soluta. Eos architecto, adipisci rem ipsam unde reiciendis officiis eveniet, suscipit eaque quae tempora omnis earum ipsum sapiente nostrum neque itaque consequuntur maiores. Saepe officia dolores inventore! Natus et aliquid dolorem eius delectus,

```
architecto dolor non atque beatae aperiam ipsum numquam quae deserunt
voluptates voluptate maxime, cum quis ex quia quaerat placeat illo.
Distinctio, atque, adipisci officia alias excepturi facere iure minus non
voluptas dolore quidem architecto impedit nostrum optio delectus expedita
possimus hic, repudiandae iste ipsa eveniet tempore. Impedit, cupiditate.
        </div>
        <footer class="item">Lorem, ipsum dolor sit amet consectetur
adipisicing elit. Provident quo libero cumque.</footer>
        <!-- <div class="item">Item-1</div>
        <div class="item">Item-2</div>
        <div class="item">Item-3</div>
        <div class="item">Item-4</div>
        <div class="item">Item-5</div>
        <div class="item">Item-6</div>
        <div class="item">Item-7</div>
        <div class="item">Item-8</div>
        <div class="item">Item-9</div>
        <div class="item">Item-10</div>
        <div class="item">Item-11</div>
        <div class="item">Item-12</div>
        <div class="item">Item-13</div>
        <div class="item">Item-14</div> -->
    </div>
</body>
</html>
```

OUTPUT:-



1.3.6 Media Queries

Media queries are commonly used to control responsive layouts on websites. Organizing layouts this way is intuitive: On a wide desktop display, we want to present information in columns, and as screen width diminishes below a threshold, we stack elements vertically. With modern CSS, solutions to this problem have become easier than in the past. No longer must we use kludgey rules like display: table; to achieve the layout of our dreams. CSS modules like flexbox and several clever frameworks have made grids easy to achieve with minimal code, but with CSS grid we can write our grid rules once, and achieve the desired layout at any screen size with a single rule, and without any framework.

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

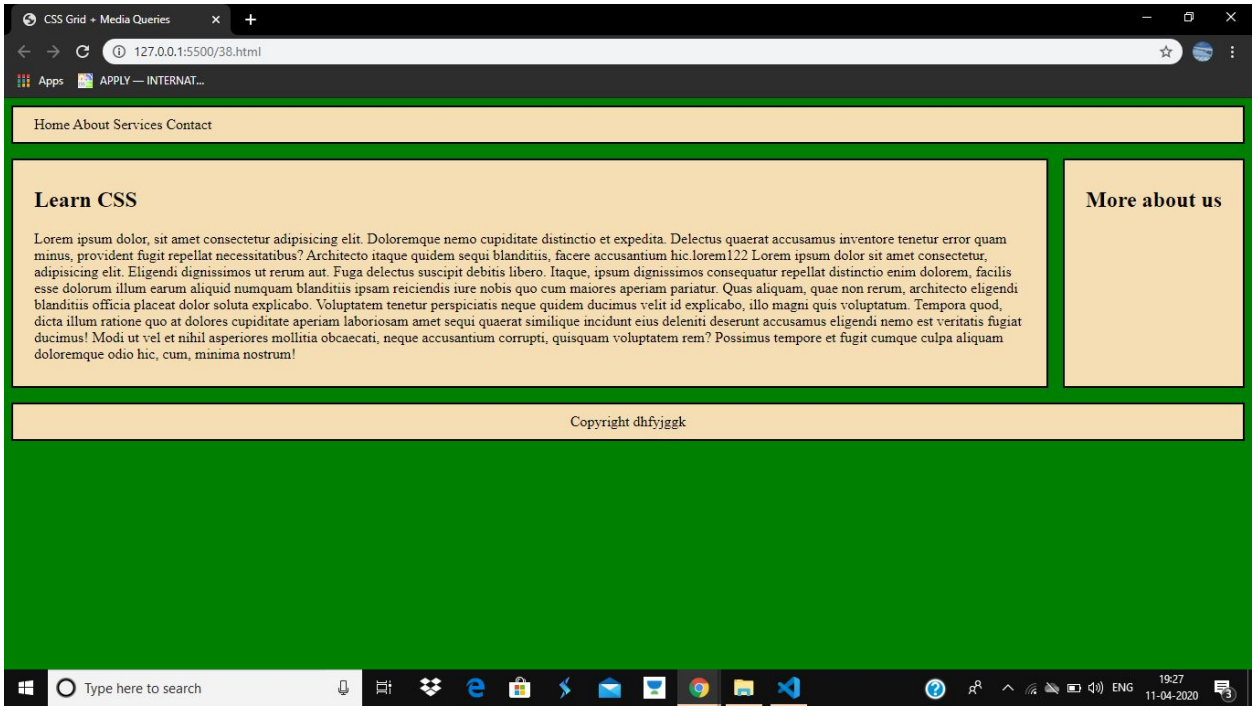
```
    <title>CSS Grid + Media Queries</title>
</head>
<style>
    .container {
        display: grid;
        grid-gap: 1rem;
        grid-template-areas:
            'navbar navbar navbar navbar'
            'section section section aside'
            'footer footer footer footer ';
    }

    @media only screen and (max-width:300px) {
        body {
            background-color: red;
        }
        .container {
            grid-template-areas:
                'navbar navbar navbar navbar'
                'section section section section'
                'aside aside aside aside'
                'footer footer footer footer ';
        }
        aside{
            display: none;
        }
        span{
            display: block;
            text-align: center;
        }
    }

    @media only screen and (min-width: 300px) and (max-width:500px) {
        body {
            background-color: blue;
        }
        .container {
```

```
            grid-template-areas:
                'navbar navbar navbar navbar'
                'section section section section'
                'aside aside aside aside'
                'footer footer footer footer ';
        }
        aside{
            display: none;
        }
        span{
            display: block;
            text-align: center;
        }
    }

    @media (min-width: 500px) and (max-width:800px) {
        body {
            background-color: yellow;
        }

        .container {
            grid-template-areas:
                'navbar navbar navbar navbar'
                'section section section section'
                'aside aside aside aside'
                'footer footer footer footer ';
        }
        span{
            display: block;
            text-align: center;
        }
    }

    @media (min-width: 800px) {
        body {
            background-color: green;
        }
```

```
    }

    .bdr {
        border: 2px solid black;
        padding: 10px 23px;
        background-color: wheat;
    }

    nav {
        grid-area: navbar;
    }

    section {
        grid-area: section;
    }

    aside {
        grid-area: aside;
    }

    footer {
        grid-area: footer;
        text-align: center;
    }
</style>

<body>
    <div class="container ">
        <nav class="bdr">
            <span>Home</span>
            <span>About</span>
            <span>Services</span>
            <span>Contact</span>
        </nav>
        <section class="bdr">
            <h2>Learn CSS </h2>
```

```
            <p>Lorem ipsum dolor, sit amet consectetur adipisicing elit.
Doloremque nemo cupiditate distinctio et expedita. Delectus quaerat
accusamus inventore tenetur error quam minus, provident fugit repellat
necessitatibus? Architecto itaque quidem sequi blanditiis, facere
accusantium hic.lorem122 Lorem ipsum dolor sit amet consectetur,
adipisicing elit. Eligendi dignissimos ut rerum aut. Fuga delectus
suscipit debitis libero. Itaque, ipsum dignissimos consequatur repellat
distinctio enim dolorem, facilis esse dolorum illum earum aliquid numquam
blanditiis ipsam reiciendis iure nobis quo cum maiores aperiam pariatur.
Quas aliquam, quae non rerum, architecto eligendi blanditiis officia
placeat dolor soluta explicabo. Voluptatem tenetur perspiciatis neque
quidem ducimus velit id explicabo, illo magni quis voluptatum. Tempora
quod, dicta illum ratione quo at dolores cupiditate aperiam laboriosam
amet sequi quaerat similique incidunt eius deleniti deserunt accusamus
eligendi nemo est veritatis fugiat ducimus! Modi ut vel et nihil
asperiores mollitia obcaecati, neque accusantium corrupti, quisquam
voluptatem rem? Possimus tempore et fugit cumque culpa aliquam doloremque
odio hic, cum, minima nostrum!</p>
        </section>
        <aside class="bdr">
            <h1>More about us</h1>
        </aside>
    </div>
    <footer class="bdr">Copyright dhfyjggk</footer>

</body>

</html>
```

OUTPUT:-




```
        .item {
            grid-column: col 2 / col 7;
            grid-row: content 6 / content 10;
        }

.item {
    grid-column: col 2 / col 7;
    grid-row: content 6 / content 10;
}
```

1.4 javascript

1.4.1 Introduction

JavaScript is the programming language of HTML and the Web and is easy to learn.JavaScript is one of the 3 languages all web developers must learn:

1. HTML:- to define the content of web pages

2. CSS:- to specify the layout of web pages

3. JavaScript:- to program the behavior of web pages

Web pages are not the only place where JavaScript is used. Many desktop and server programs use JavaScript. Node.js is the best known. Some databases, like MongoDB and CouchDB, also use JavaScript as their programming language.

1.4.2 Writing in-browser JavaScript and Developer Console

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually.
JavaScript in Chrome

Here are the steps to turn on or turn off JavaScript in Chrome −

- Click the Chrome menu at the top right hand corner of your browser.
- Select Settings.
- Click Show advanced settings at the end of the page.
- Under the Privacy section, click the Content settings button.
- In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

What can in-browser Javascript do ?

- Add new HTML and change existing HTML
- React to events - Response from server,Key press,mouse movement
- Ajax Request
- Get and Set cookies & Use local storage

What can’t in-browser JavaScript can do ?

- Read/Write to/from computer harddisk

- Should operate on same origin policy

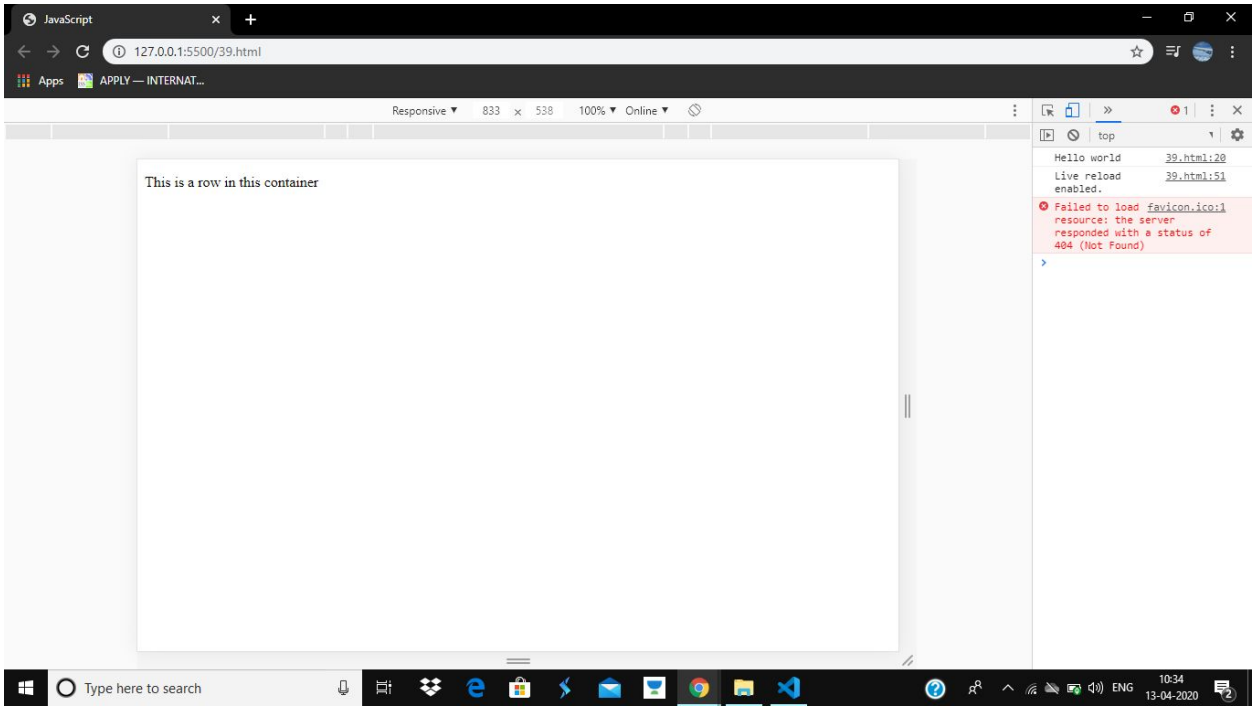
What makes JavaScript so unique?

- HTML Support
- Simple API
- Supported by major modern browsers

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript </title>

</head>
<body>
    <div class="container">
        <div class="row">
            <p>
                This is a row in this container
            </p>
        </div>
    </div>
    <script>
        //Write your js here
        console.log('Hello world');

    </script>
</body>
</html>
```

OUTPUT:-



1.4.3 Variables,Data Types and Operators

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After the first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

A JavaScript local variable is declared inside a block or function. It is accessible within the function or block only.

```
function abc(){
var x=10;//local variable
}
```

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with a window object is known as a global variable.

```
1. var data=200;//global variable
2. function a(){
3. document.writeln(data);
4. }
5. function b(){
6. document.writeln(data);
7. }
8. a();//calling JavaScript function
9. b();
```

Data Types in JavaScript :- Data types basically specify what kind of data can be stored and manipulated within a program.

There are six basic data types in JavaScript which can be divided into three main categories: primitive (or *primary*)
composite (or *reference*)
special data types.

String, Number, and Boolean are primitive data types. Object, Array, and Function (which are all types of objects) are composite data types. Whereas Undefined and Null are special data types.

Primitive data types can hold only one value at a time, whereas composite data types can hold collections of values and more complex entities.

Operators in JavaScript :-

Operator	Description
+	Addition
-	Subtraction
*	Multiplication

**	Exponentiation
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Properties:-

- Dynamic Typing
- Operators :- Unary Operator(one operand),Binary Operators(two operands)
- Operand(:-entities on which operator operates) and operator(function performed)

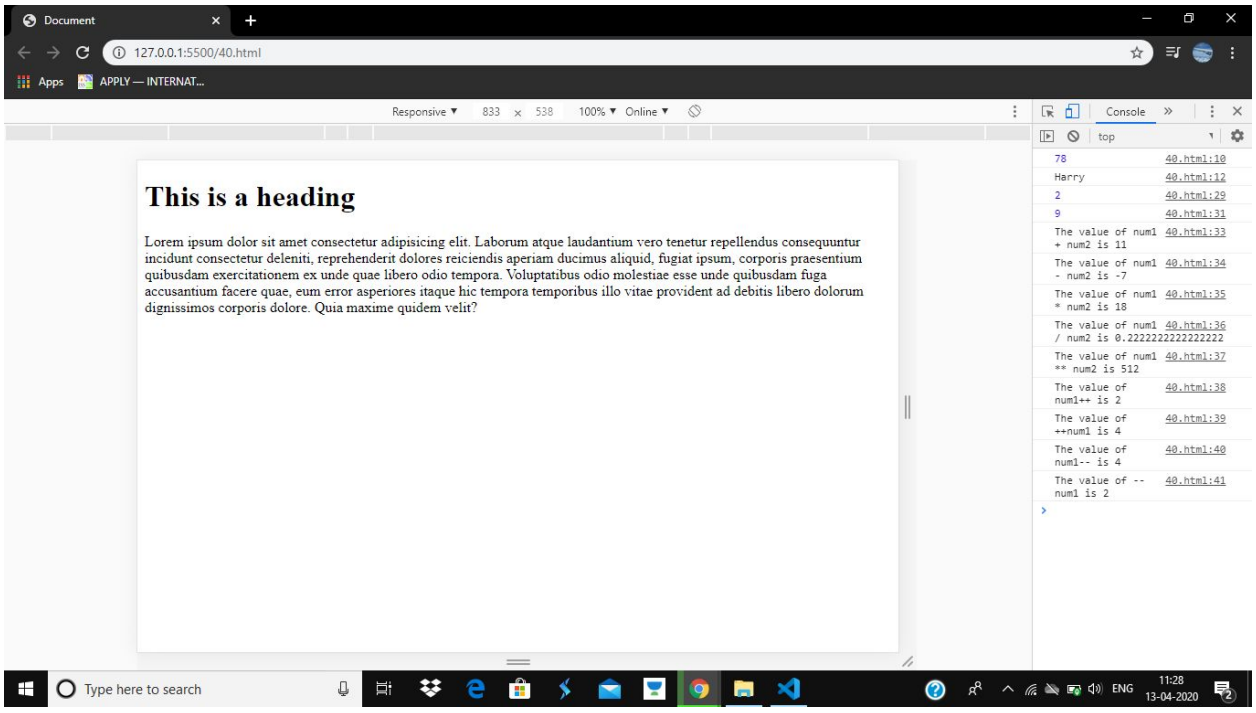
SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <script>
        var a = 78;
        console.log(a);
        var b = "Harry";
        console.log(b);
        c = 34.55;
        // console.log(c);

        // Operators in JavaScript
        // Operand - entities on which operators operate
```

```
        // In 3 + 4 '+' is the operator and 3,4 are operands
        // 1. unary operator - It has single operand (x = -x)
        // Examples of unary operator
        c = -c;
        // console.log(c);
        // 2. binary operator - It has two operand (x = x+6)
        // Examples of binary operator
        c = 456+8;
        // console.log(c);

        var num1 = 2;
        console.log(num1);
        var num2 = 9;
        console.log(num2);
        // Arithmetic operators in action in JavaScript
        console.log("The value of num1 + num2 is "+ (num1 + num2));
        console.log("The value of num1 - num2 is "+ (num1 - num2));
        console.log("The value of num1 * num2 is "+ (num1 * num2));
        console.log("The value of num1 / num2 is "+ (num1 / num2));
        console.log("The value of num1 ** num2 is "+ (num1 ** num2));
        console.log("The value of num1++ is "+ (num1++));
        console.log("The value of ++num1 is "+ (++num1));
        console.log("The value of num1-- is "+ (num1--));
        console.log("The value of --num1 is "+ (--num1));









    </script>
</head>
<body>
    <div class="container">
        <h1>This is a heading</h1>

        <div class="content">
```

```
            <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
Laborum atque laudantium vero tenetur repellendus consequuntur incidunt
consectetur deleniti, reprehenderit dolores reiciendis aperiam ducimus
aliquid, fugiat ipsum, corporis praesentium quibusdam exercitationem ex
unde quae libero odio tempora. Voluptatibus odio molestiae esse unde
quibusdam fuga accusantium facere quae, eum error asperiores itaque hic
tempora temporibus illo vitae provident ad debitis libero dolorum
dignissimos corporis dolore. Quia maxime quidem velit?</p>
        </div>
    </div>
</body>
</html>
```

OUTPUT:-



1.4.4 Strings

JavaScript strings are used for storing and manipulating text.A JavaScript string is zero or more characters written inside quotes.

```
var x = "John Doe";
```

We can use single or double quotes:

```
var carName1 = "Volvo XC60";  // Double quotes
var carName2 = 'Volvo XC60';  // Single quotes
```

We can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
var answer1 = "It's alright";
```

To find the length of a string, use the built-in length property:

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

The backslash (\) escape character turns special characters into string characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Other escape sequence:

Code	Result
\b	Backspace

\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

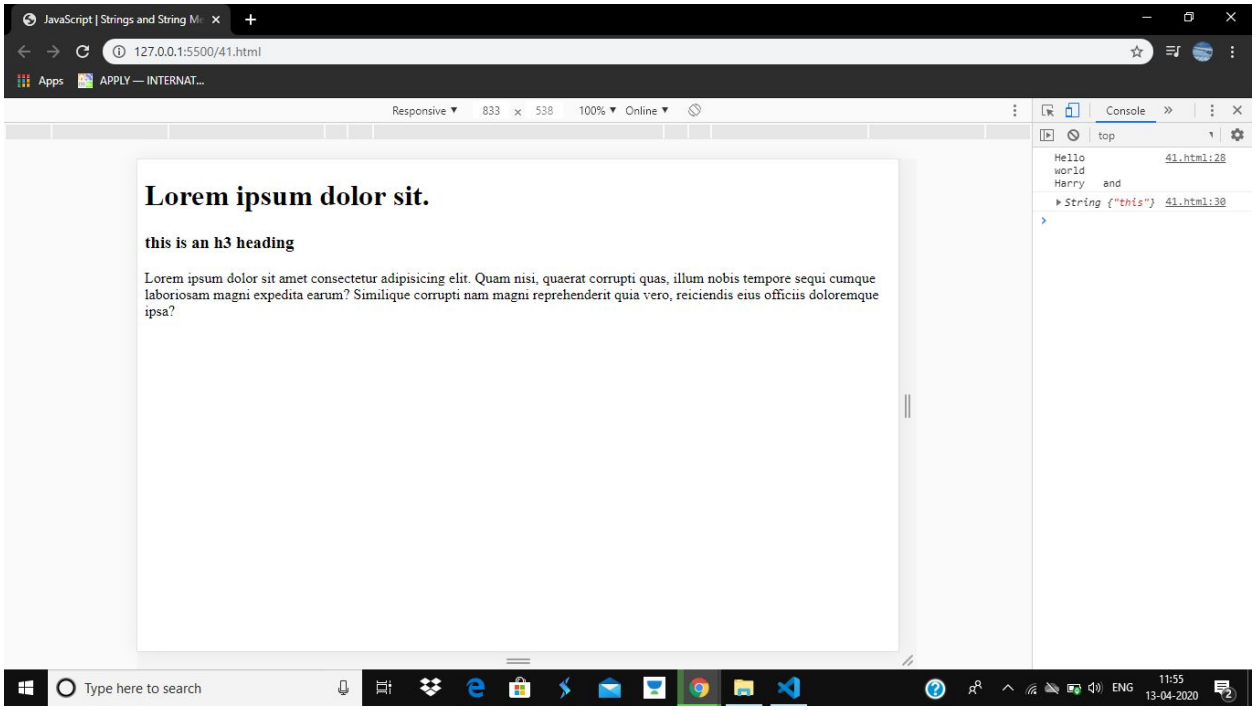
document.getElementById("demo").innerHTML ="Hello Dolly!";

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript | Strings and String Methods</title>
</head>

<body>
    <div class="container">
        <h1>Lorem ipsum dolor sit.</h1>
        <div id="content"></div>
        <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Quam
nisi, quaerat corrupti quas, illum nobis tempore sequi cumque laboriosam
magni expedita earum? Similique corrupti nam magni reprehenderit quia
vero, reiciendis eius officiis doloremque ipsa?</p>
    </div>
    <script>
    // var string = "this";
```

```
    var string = 'thi"s';
    var name = 'fjy';
    var channel = 'fjegfkew';
    var message = 'idfgreureui';
    var temp = `${name} is a 'nice' person "and" he has a channel called
${channel}`;
    // console.log(string + name + message);
    // console.log(temp);
    // var len = name.length;
    // console.log(`Length of name is ${len}`)

    // console.log("Hello \\nworld\nHarry\tand");
    var y = new String("this");
    console.log(y);
    document.getElementById('content').innerHTML = '<h3>this is an h3
heading</h3>'

    </script>
</body>
</html>
```

OUTPUT:-



1.4.5 String Functions

String methods help you to work with strings.Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

String Length:-The length property returns the length of a string

Finding a String in a String

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

The lastIndexOf() method returns the index of the last occurrence of a specified text in a string:

```
var pos = str.lastIndexOf("John");
```

The search() method searches a string for a specified value and returns the position of the match.

The slice() Method:- slice() extracts a part of a string and returns the extracted part in a new string.The method takes 2 parameters: the start position, and the end position (end not included).

If a parameter is negative, the position is counted from the end of the string.If you omit the second parameter, the method will slice out the rest of the string:

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);
```

The substring() Method:- substring() is similar to slice().The difference is that substring() cannot accept negative indexes.

The substr() Method:- substr() is similar to slice().The difference is that the second parameter specifies the length of the extracted part.

Replacing String Content:- The replace() method replaces a specified value with another value in a string:

Converting to Upper and Lower Case

A string is converted to upper case with toUpperCase():

A string is converted to lowercase with toLowerCase():

```
var text1 = "Hello World!";       // String
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower
```

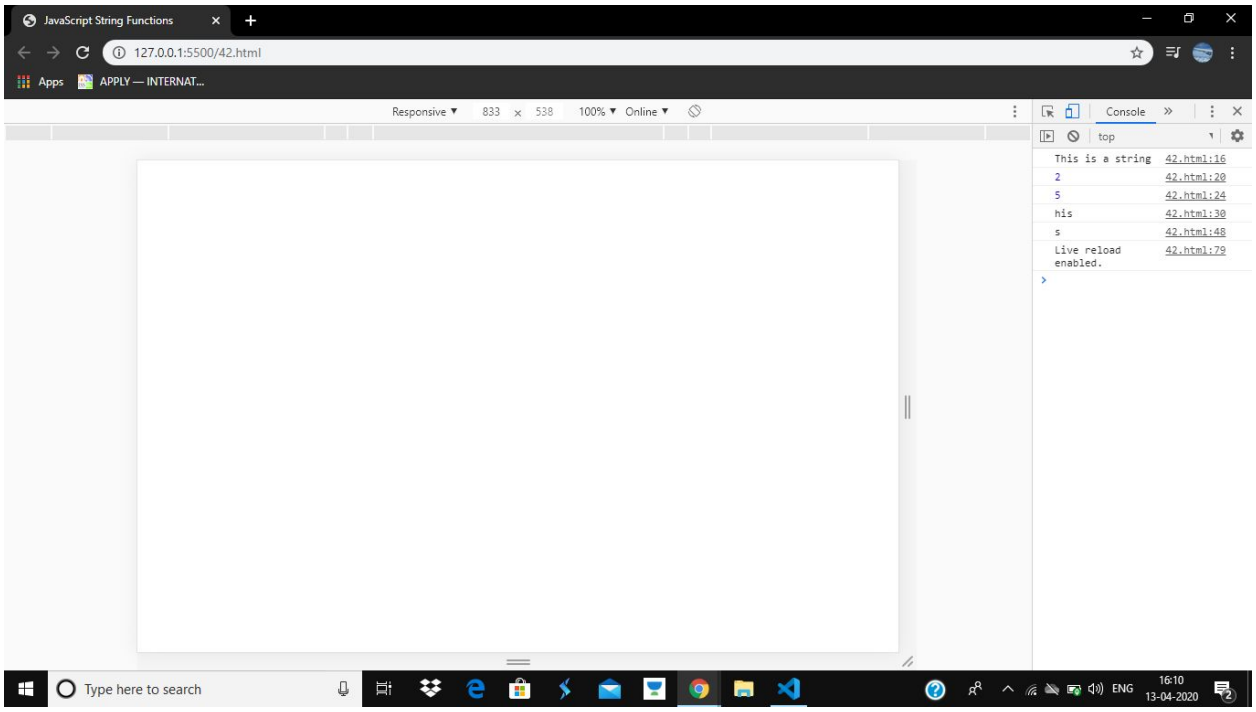
SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>JavaScript String Functions</title>
    <style>
```

```
    </style>
</head>
<body>

    <script>
    var str = "This is a string";
    console.log(str);

    // First occurence of a substring
    var position = str.indexOf('is');
    console.log(position)

    // Last occurence of a substring
    position = str.lastIndexOf('is');
    console.log(position)

    // Substring from a string
    // var substr = str.slice(1,7);
    // var substr = str.substring(1,7);
    var substr1 = str.substr(1,3);
    console.log(substr1)

    // var replaced = str.replace('string', 'Harry');
    // console.log(str)
    // console.log(replaced)

    // console.log(str.toUpperCase());
    // console.log(str.toLowerCase());
    // var newString = str.concat('New String')
    // console.log(newString)
    // var strWithWhitespaces = "   this contains        whitespaces   ";
    // console.log(strWithWhitespaces)
    // console.log(strWithWhitespaces.trim())

    // var char2 = str.charAt(2);
    // var char2 = str.charCodeAt(2); // Not very important
    // console.log(char2)
```

```
    console.log(str[3])

    </script>
</body>
</html>
```

OUTPUT:-



1.4.6 Scope,If-else conditionals & Switch Case

Scope:- Scope determines the accessibility (visibility) of variables.JavaScript Function Scope
In JavaScript there are two types of scope:

- Local scope
- Global scope

JavaScript has function scope: Each function creates a new scope.

Variables defined inside a function are not accessible (visible) from outside the function.

Local JavaScript Variables:- Variables declared within a JavaScript function, become LOCAL to the function.Local variables have Function scope:- They can only be accessed from within the function.

```
function myFunction() {
  var carName = "Volvo";
}
```

Global JavaScript Variables:- A variable declared outside a function, becomes GLOBAL.A global variable has global scope:- All scripts and functions on a web page can access it.

```
var carName = "Volvo";
function myFunction() {
}
```

Conditional Statements:- Conditional statements are used to perform different actions based on different conditions.

Conditional Statements:- In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

The if Statement:- Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

```
if (condition) {
}
```

The else Statement:- Use the else statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

The else if Statement:- Use the else if statement to specify a new condition if the first condition is false.

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

The switch statement is used to perform different actions based on different conditions.The JavaScript Switch Statement use the switch statement to select one of many code blocks to be executed.

Syntax:-

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

The break Keyword:- When JavaScript reaches a break keyword, it breaks out of the switch block.This will stop the execution of inside the block.It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

The default Keyword:- The default keyword specifies the code to run if there is no case match

Switching Details

If multiple cases match a case value, the first case is selected.
If no matching cases are found, the program continues to the default label.
If no default label is found, the program continues to the statement(s) after the switch.

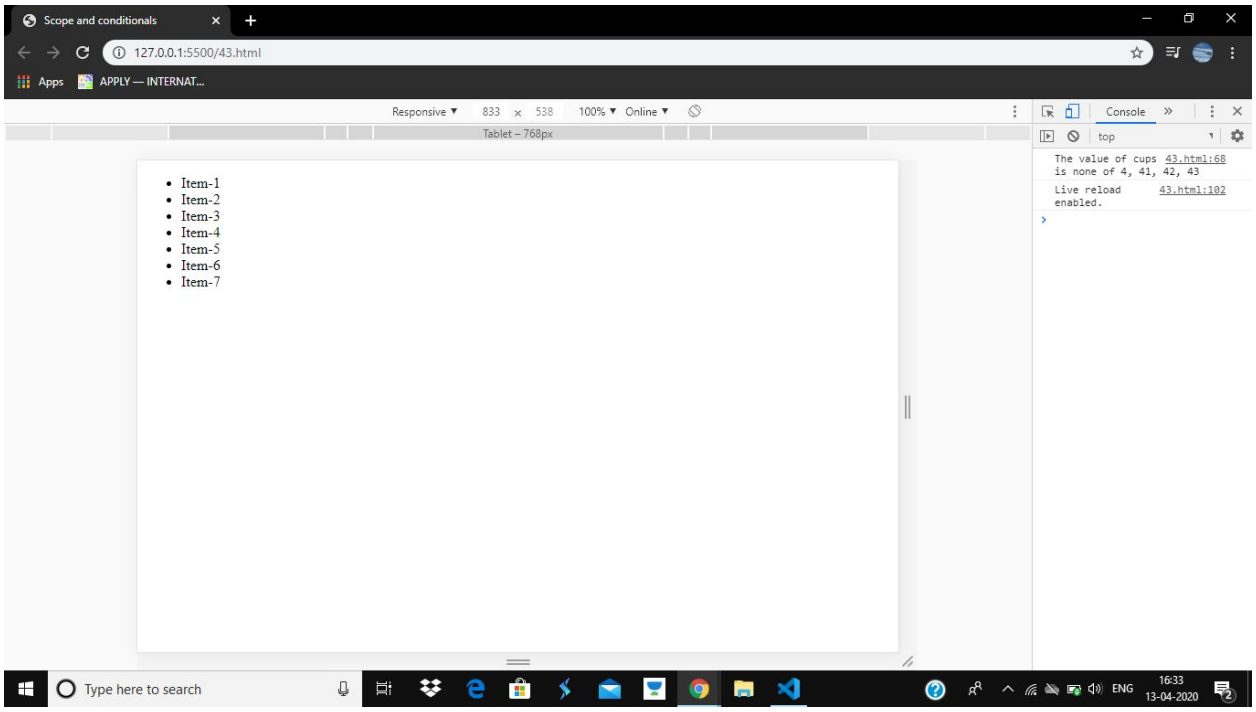
SOURCE CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Scope and conditionals</title>
</head>
<body>
    <div>
        <ul>
            <li>Item-1</li>
            <li>Item-2</li>
            <li>Item-3</li>
            <li>Item-4</li>
            <li>Item-5</li>
            <li>Item-6</li>
            <li>Item-7</li>
        </ul>
    </div>
    <script>
    // var string1 = "This is a string";
    // var string1 = "This is a string2";
    // console.log(string1);
    // let a = "u";
    // {
    // let a = "u6";
    // console.log(a)
    // }
    // console.log(a)
```

```
// const a = "This cannot be changed";
// a = "I want to change this. This cannot be changed";
// console.log(a);
// let age = 5;
// if(age>18){
//     console.log("You can drink water");
// }
// else if(age==2){
//     console.log("Age is 2")
// }
// else if(age==5){
//     console.log("Age is 5")
// }
// else{
//     console.log("You can drink Cold Drink");
// }

const cups = 47;
switch (cups) {
    case 4:
        console.log("The value of cups is 4")
        break;

    case 41:
        console.log("The value of cups is 41")
        break;

    case 42:
        console.log("The value of cups is 42")
        break;

    case 43:
        console.log("The value of cups is 43")
        break;

    default:
        console.log("The value of cups is none of 4, 41, 42, 43")
```

```
            break;
    }



    </script>
</body>
</html>
```

OUTPUT:-



1.4.7 Arrays and Objects

Arrays:-

JavaScript arrays are used to store multiple values in a single variable.

var cars = ["Saab", "Volvo", "BMW"];

What is an Array?

An array is a special variable, which can hold more than one value at a time.If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

An array can hold many values under a single name, and you can access the values by referring to an index number.

Syntax:

var *array_name* = [*item1*, *item2*, ...];

Using the JavaScript Keyword new: - var cars = new Array("Saab", "Volvo", "BMW");

Access the Elements of an Array:- You access an array element by referring to the index number.This statement accesses the value of the first element in cars:

```
var name = cars[0];
```

Changing an Array Element:- This statement changes the value of the first element in cars:

```
cars[0] = "Opel";
```

Arrays are Objects:- Arrays are a special type of objects. The typeof operator in JavaScript returns an "object" for arrays.But, JavaScript arrays are best described as arrays.Arrays use numbers to access its "elements". In this example, person[0] returns John:

Array:

```
var person = ["John", "Doe", 46];
```

Objects use names to access its "members". In this example, person.firstName returns John:
Object:

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

Array Elements Can Be Objects:- JavaScript variables can be objects. Arrays are special kinds of objects.Because of this, we can have variables of different types in the same Array.we can have objects,functions,arrays in an Array.

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars
```

Objects:-

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the new keyword)
- Numbers can be objects (if defined with the new keyword)
- Strings can be objects (if defined with the new keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Primitives:- A primitive value is a value that has no properties or methods.It is data that has a primitive value.

JavaScript defines 5 types of primitive data types:

- string
- number
- boolean
- null
- undefined

Primitive values are immutable (they are hardcoded and therefore cannot be changed).

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"

3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

JavaScript variables can contain single value and many values.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Objects written as name value pairs are similar to:

- Associative arrays in PHP
- Dictionaries in Python
- Hash tables in C
- Hash maps in Java
- Hashes in Ruby and Perl

Object Methods:- Methods are actions that can be performed on objects.An object method is an object property containing a function definition.

Creating a JavaScript Object:- With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

- Define and create a single object, using an object literal.
- Define and create a single object, with the keyword new.
- Define an object constructor, and then create objects of the constructed type.

Using an Object Literal

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Using the JavaScript Keyword new

```
var person = new Object();
person.firstName = "John";
```

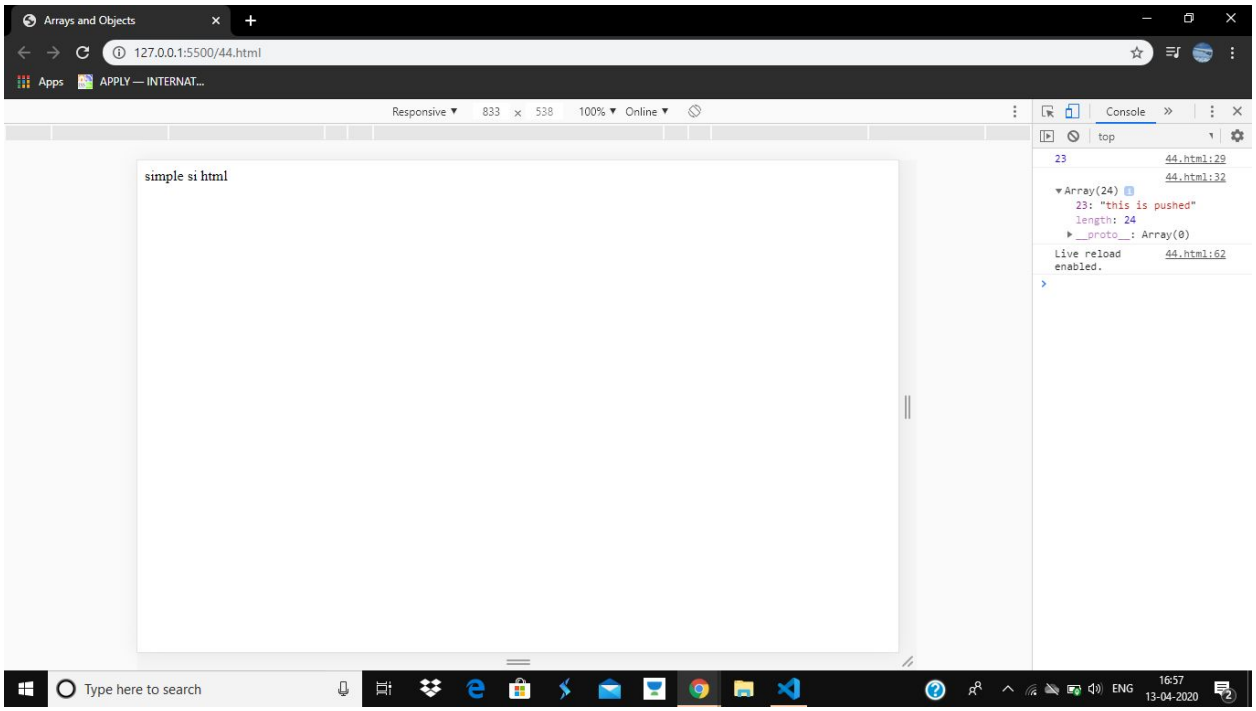
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";

SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Arrays and Objects</title>
</head>
<body>
    <div class="container">simple si html</div>
    <script>
        let myVar = 34;
        let myVar2 = "string";
        let myVar3 = true;
        let myVar4 = null;
        let myVar5 = undefined;

        // let employee = {
        //     name: "Harry",
        //     salary: 10,
        //     channel: "CodeWithHarry",
        //     "channel 2": "ProgrammingWithHarry",
        // }
        // console.log(employee);

        // let names = [1, 2, 4, "Harry", undefined];
        // let names = new Array(41, 2, 4, "Harry", undefined);
        let names = new Array(23);
        console.log(names.length);
        names = names.sort();
        names.push("this is pushed");
        console.log(names);
```

```
    </script>
</body>
</html>
```

OUTPUT:-



1.4.8 Functions

A JavaScript function is a block of code designed to perform a particular task.It is executed when "something" invokes it (calls it).

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

JavaScript Function Syntax:- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).The parentheses may include parameter names separated by commas:

(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: {}

function *name*(*parameter1, parameter2, parameter3*) {
  // *code to be executed*
}

Function parameters are listed inside the parentheses () in the function definition.Function arguments are the values received by the function when it is invoked.Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation:- The code inside the function will execute when "something" invokes (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return:- When JavaScript reaches a return statement, the function will stop executing.If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.Functions often compute a return value. The return value is "returned" back to the "caller":The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

Local Variables:- Variables declared within a JavaScript function, become LOCAL to the function.Local variables can only be accessed from within the function.

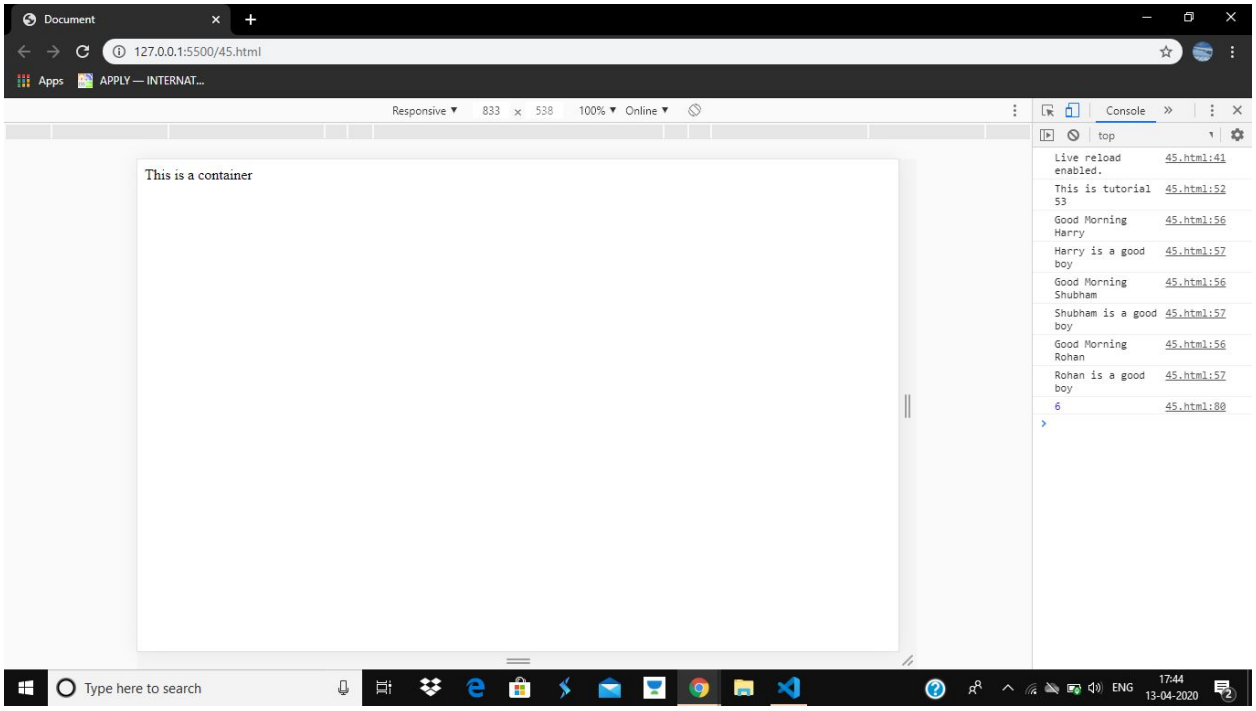
function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}
SOURCE CODE:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
```

```
</head>
<body>
    <div class="container">
        This is a container
    </div>
</body>
<script>
console.log("This is hdfkanklnf");

function greet(name, greetText="Greetings from JavaScript"){
    let name1 = "Name1";
    console.log(greetText + " " + name);
    console.log(name + " is a good boy");
}

function sum(a,b,c){
    let d = a + b + c;
    return d;
    // This line will never execute (Unreachable code)
    // console.log("Function is returned");
}


let name = "Harry";
let name1 = "Shubham";
let name2 = "Rohan";
let name3 = "Sagar";
let greetText1 = "Good Morning";
greet(name, greetText1);
greet(name1, greetText1);
greet(name2, greetText1);
// let returnVal = greet(name3);
// console.log(returnVal)

let returnVal = sum(1,2,3);
console.log(returnVal)
// console.log(name + " is a good boy");
```

```
// console.log(name1 + " is a good boy");
// console.log(name2 + " is a good boy");
// console.log(name3 + " is a good boy");
</script>
</html>
```

OUTPUT:-



REFERENCES:-

CSS :-

https://www.w3schools.com/css/

https://www.codewithharry.com/

https://en.wikipedia.org/wiki/Web_development

https://www.udemy.com/courses/development/web-development/

Javascript:-

https://www.w3schools.com/css/

https://www.codewithharry.com/