

UNIT – 1

Syllabus:

CSS 3: Syntax structure, using style sheets, Box model, Grid, Flexbox. Responsive Web Design using Media Queries, use of viewport, Transition, Animation. CSS Framework: Bootstrap.

XML: Introduction, Syntax, Validating XML with Document type definition and XML Schemas.

CASCADING STYLE SHEETS (CSS)

➤ **INTRODUCTION:**

- A Style sheet is a set of stylistic rules that expresses the Presentation and Layout of Structured documents (Web Pages).
- *“Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.”*
- CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.
- It is the language for describing the presentation of Web pages, including colors, layout, and fonts, thus making our web pages more presentable to the users.
- CSS **consists of set of rules** that determines how the content of elements within your document should be formatted.
- Collection of CSS rules is called as “Stylesheet”

Advantages of CSS:

- Reduced Complexity and Repetition :
 - Supports reusability – Same stylesheet can be used in multiple pages
 - Saves the time of developer.
- Reduced Document Size.
- Once a visitor to your site has downloaded the CSS style with the first page of your site that uses it, subsequent pages will be quicker to load (because the browser retains a copy of the CSS style sheet and the rules do not have to be downloaded for every page). This also puts less strain on the server because the pages it sends out are smaller.
- Reduced Clutter:
 - Can change the appearance of several pages by altering style sheet rather than individual page
- Multiple style sheets can be included in a single page.

- Improves formatting capability of a HTML page.
- One style sheet can import and use styles from other style sheets
 - @import cssfilename

➤ **CSS RULES:**

- CSS works by allowing you to associate rules with the elements that appear in a web page. These rules govern how the content of those elements should be rendered.
 - For example, we can specify that the contents of all < p > elements should be displayed in gray using the Arial typeface, and that all < h1 > elements should be in red using the Times New Roman typeface.

- **Syntax:**

```
Selector  {
            property1 : value ;
            property2 : value;
            .....
        }
```

- CSS rule is made up of 2 parts:

1. Selector
2. Declaration

- **Selector :** Element/ set of elements to which declaration must be applied to
- **Declaration:**
 - (i). *Property:* CSS Property that is to be applied
 - (ii). *Value:* Value of CSS property
- **Example:**

```
h1  _____➔      Selector
{
    font-family : arial;
    color : blue;
    text-align : center;
}
                    } Declaration
```

➤ **CSS SELECTORS:**

- CSS selectors are used to determine which HTML element, or set of elements, to apply the CSS declarations/rules to.
- **Basic CSS selectors:**
 - Element name/type Selector
 - Class Selector
 - Id Selector

- Grouping Selector
- Universal Selector
- Pseudo Class Selector

- **Element name/type Selector**

- Selects HTML elements based on the element name and applies CSS rules.
- **Syntax:**

```
element-name {
    property1 : value; property2 : value;
    .....
}
```

- **Example:**

Here, all **<p>** elements on the page will be center-aligned, with a red text color:

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

- **Class Selector/ Stylesheet 'Class':**

- Selects HTML elements with a specific class attribute.
- Allows us to define multiple styles for the same type of HTML element.
- It is used with a period character '.' followed by the class name.
- **Syntax 1:**

```
selector.classname {
    property1 : value1;    property2 : value; ...
}
```

- **Example:**

- **<p>** elements with class="center" will be red and center-aligned:

CSS Rule:

```
p.center {
    text-align: center;
    color: red;
}
```

HTML Code: **<p class="center">** This is a Paragraph **</p>**

- **Syntax 2:** To define a style that can be used by multiple HTML elements remove tag name/selector.

```
.classname {
    property1 : value1;    property2 : value; .....
}
```

- **Example:**

All HTML elements with class="center" will be red and center-aligned:

CSS Rule:

```
.center {  
    text-align: center;  
    color: red;  
}
```

HTML Code: `<p class="center"> This Paragraph is red in color </p>`

`<h1 class="center"> This Heading is red in color </h1>`

- **Id Selector:**

- Uses the id attribute of an HTML element to select a specific element.

- There should be only one element with a given ID in a document.

- **Syntax:**

```
#idname {  
    property1 : value1;    property2 : value; ....  
}
```

- **Example:**

The CSS rule below will be applied to the HTML element with id="para1":

CSS rule:

```
#para1 {  
    text-align: center;  
    color: red;  
}
```

HTML Code: `<p id="para1"> This Paragraph is Red in color </p>`

- **Grouping Selector:**

- Used when we want to apply same styles to multiple elements.

- The grouping selector selects all the HTML elements with the same style definitions.

- To group selectors, each selector is separated by a space.

- **Syntax:**

```
Selector 1, Selector 2,....  
{  
    property1 : value1;    property2 : value; ...  
}
```

- **Example:**

```
h1, h2, p {  
    text-align: center;  
    color: red;  
}
```

- **Universal Selector:**

- The universal selector is an asterisk (*)
- Selects ***all*** the HTML elements on the page.
- Used:
 - When we want a rule to be applied to all elements present in a web page.
 - To specify default values that will apply to the whole of the document (such as a font - family and font - size)
- **Syntax:**

```
* {  
    property1 : value; property2 : value;  
    .....  
}
```

- **Example:**

The CSS rule below will affect every HTML element on the page:

```
* {  
    text-align: center;  
    color: blue;  
}
```

- **Pseudo-class Selector:**

- A pseudo-class is used to define a special state of an element.
- For example, it can be used to:
 - Style an element when a user mouses over it
 - Style visited and unvisited links differently
 - Style an element when it gets focus
- **Syntax:**

```
selector:pseudo-class {  
    property: value;  
}
```

- **Some pseudo-classes:**

- :link - adds style to an unvisited link
- :visited – adds style to a visited link
- :active – adds style to an active link
- :hover – add style to an element when we mouse over it
- :focus – adds style to an element when it has focus
- :first-child – adds style to the first child of an element
- :last-child – adds style to the last child of an element
- :nth-child(N) – adds style to an nth child of an element. N can be a number or keyword(even/odd)

- **Example 1:**

When user moves his mouse over the div element its background color will be set to blue.

```
div:hover {  
    background-color: blue;  
}
```

- **Example 2:**

Specify a background color for every <p> element that is the second child of its parent:

```
p:nth-child(2) {  
    background-color: red;  
}
```

All CSS Simple Selectors:

Selector	Example	Example description
<u>#id</u>	#firstname	Selects the element with id="firstname"
<u>.class</u>	.intro	Selects all elements with class="intro"
<u>element.class</u>	p.intro	Selects only <p> elements with class="intro"
<u>*</u>	*	Selects all elements
<u>element</u>	p	Selects all <p> elements
<u>element,element,..</u>	div, p	Selects all <div> elements and all <p> elements

➤ **TYPES OF CSS:**

- When a browser reads a style sheet, it will format the HTML document according to the information in the style sheet.
- There are three ways of inserting a style sheet:
 1. Inline style sheet
 2. Internal/Embedded style sheet
 3. External style sheet

1. INLINE STYLE SHEET:

- Inline styles are placed directly inside a specific HTML element in the code.
- The style is applied at the occurrence of the HTML element by using “**style**” attribute in the relevant tag.
- The style attribute can contain any number of CSS Properties.
- Inline styles cannot be reused at all

- **Syntax:**

```
<element style="property1:value;property2:value;... "> --- </element>
```

- **Example:**

```
<html>
<body>
<h1>This is Normal Text</h1>
<p style="color:red;font-size:30pt;text-align:center">This Text is
Styled</p>
</body>
</html>
```

Output:

This is Normal Text

This Text is Styled

2. INTERNAL STYLE SHEET:

- An internal style sheet may be used if one single page has a unique style.
- Internal styles are defined within the **<style>** element, inside the **<head>** section of an HTML page.
- All the desired selectors along with the properties and values are included in the header section between **<style>** and **</style>** tags.

- **Example:**

```
<html>
<head>
<style>
    body {
        background-color:pink;
    }
    h1 {
        color: maroon;
        font-family: verdana;
    }
</style>
</head>
<body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
</body>
</html>
```

Output:

This is a heading

This is a paragraph.

3. EXTERNAL STYLE SHEET:

- External Style Sheets are useful when we need to apply particular style to more than one web page.
- The central idea in this type of style sheet is that the desired style is stored in an external **.css** file.
- The name of the external **.css** file has to be mentioned on our web pages. Then the styles defined in the **.css** file will be applied to all those web pages.
- **<link>** tag is used to link the external style sheet to a web page.

Example:

Mystyle.css:

```
h1 {
    color:blue;
    background-color:cyan;
    text-decoration:underline;
}
p {
    color:orange;
    background-color:black;
}
```

Ext.html:

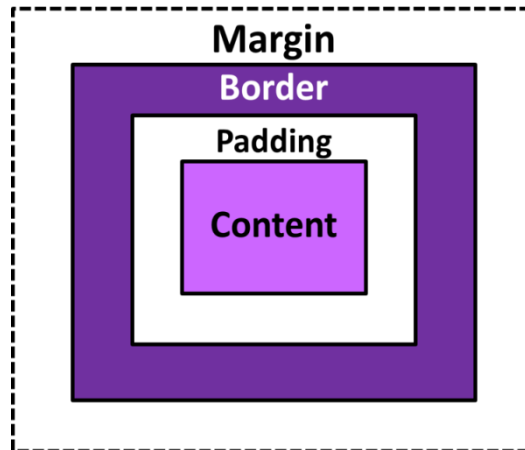
```
<html>
<head>
    <link rel="stylesheet" href="Mystyle.css">
</head>
<body>
    <h1>This is Heading 1</h1>
    <p>This is Paragraph1</p>
</body>
</html>
```

Output:

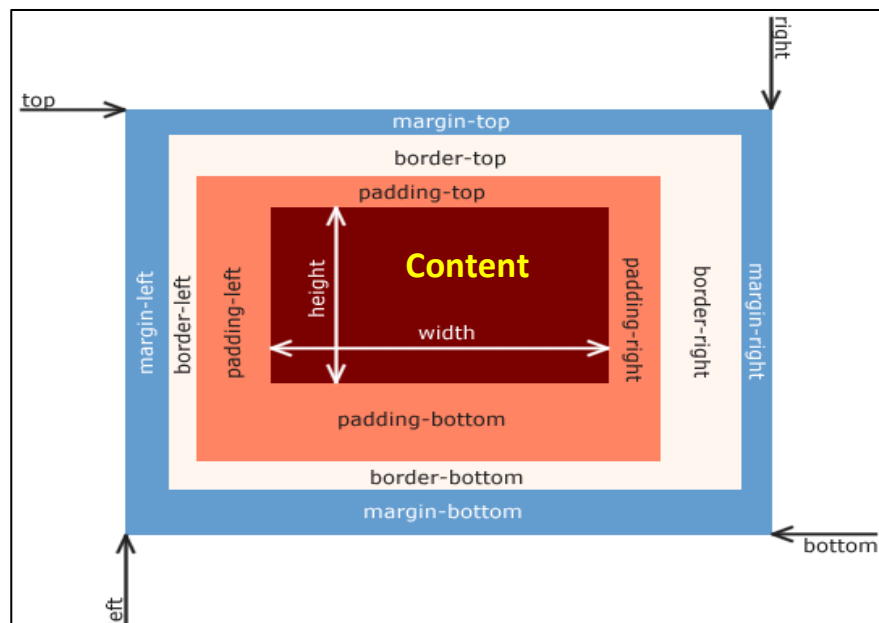


➤ **CSS BOX MODEL:**

- The box model is a very important concept in CSS because it determines how elements are positioned within the browser window.
- When laying out a document, the browser's rendering engine represents each element as a rectangular box according to the standard CSS basic box model.
- The CSS box model is essentially a box that wraps around every HTML element.
- Every box is composed of four areas(or parts), defined by their respective edges: Content, Padding, Border and Margin.



1. **Content** - Contains the "real" content of the element, such as text, an image.
2. **Padding** - Clears an area around the content. It is the transparent space between the content of the box and its border. If an element has the background color it will be visible through its padding area.
3. **Border** - A border that goes around the padding and content
4. **Margin** - Clears an area outside the border. It is an empty area used to separate the element from its neighbours. It is not affected by the element's background color and is transparent.



- The actual space that an element's box might take on a web page is calculated as:

Box Size	CSS Properties
Total Width	width + padding-left + padding-right + border-left + border-right + margin-left + margin-right
Total Height	height + padding-top + padding-bottom + border-top + border-bottom + margin-top + margin-bottom

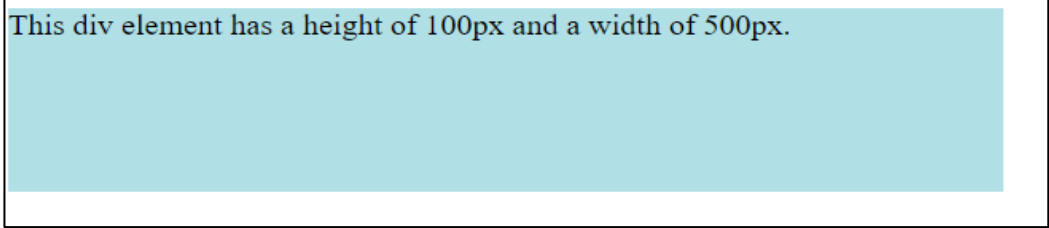
CSS Dimension Properties:

- CSS has several dimension properties, such as ***width, height, max-width, min-width, max-height, and min-height*** that allow you to control the width and height of an element.
- The CSS ***height*** and ***width*** properties are used to set the height and width of the content area of an element.
- This width and height does not include paddings, borders, or margins.
- The width and height properties can take the following values:
 - auto - This is default. The browser calculates the height and width
 - length - Defines the height/width in px, cm etc.
 - % - Defines the height/width in percent of the containing block

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
  div {
    height: 100px;
    width: 500px;
    background-color: powderblue;
  }
</style>
</head>
<body>
  <div>This div element has a height of 100px and a width of 500px.</div>
</body>
</html>
```

Output:



This div element has a height of 100px and a width of 500px.

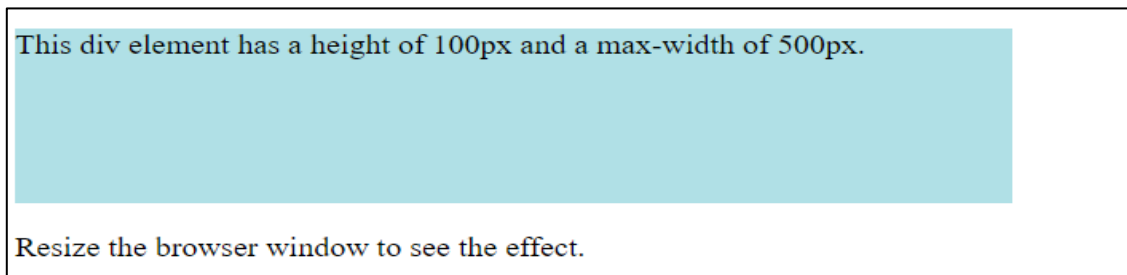
- The ***max-width*** property is used to set the maximum width of an element.
- The ***max-height*** property is used to set the maximum width of an element.

- **Example:**

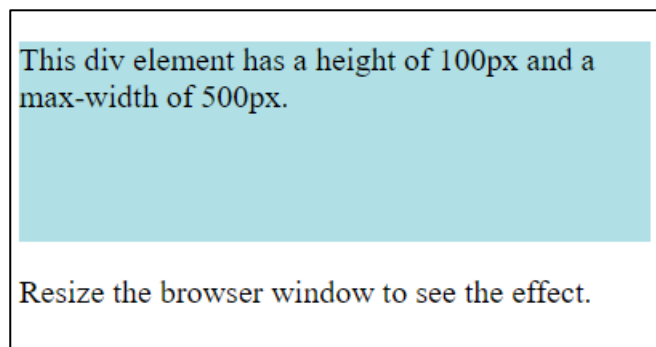
```
<!DOCTYPE html>
<html>
<head>
<style>
  div {
    max-width: 500px;
    height: 100px;
    background-color: powderblue;
  }
</style>
</head>
<body>
<div>This div element has a height of 100px and a max-width of 500px.</div>
<p>Resize the browser window to see the effect.</p>
</body>
</html>
```

Output:

When the browser is not resized:



When the browser is resized:



CSS Padding Properties:

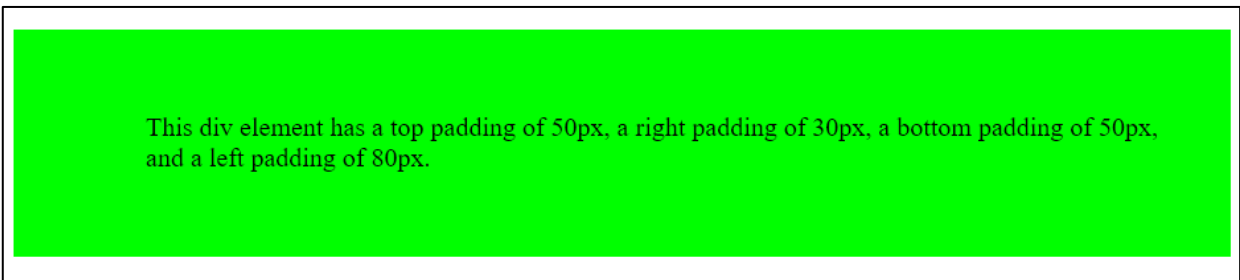
- Padding is used to create space around an element's content, inside of any defined borders.
- The CSS padding properties are used to generate space around an element's content, inside of any defined borders.

- CSS has properties for specifying the padding for each side of an element:
 - ***padding-top***
 - ***padding-right***
 - ***padding-bottom***
 - ***padding-left***
- All the padding properties can have the following values:
 - length - specifies a padding in px, pt, cm, etc.
 - % - specifies a padding in % of the width of the containing element

○ **Example:**

```
<!DOCTYPE html>
<html>
<head>
<style>
    div {
        background-color: lime;
        padding-top: 50px;
        padding-right: 30px;
        padding-bottom: 50px;
        padding-left: 80px;
    }
</style>
</head>
<body>
<div>This div element has a top padding of 50px, a right padding of 30px, a
bottom padding of 50px, and a left padding of 80px.</div>
</body>
</html>
```

Output:



This div element has a top padding of 50px, a right padding of 30px, a bottom padding of 50px, and a left padding of 80px.

- The ***padding*** property is a shorthand property for the following individual padding properties: padding-top, padding-right, padding-bottom, padding-left
- If the ***padding*** property has four values:


```
padding: 25px 50px 75px 100px;
```

 - top padding is 25px
 - right padding is 50px
 - bottom padding is 75px

- left padding is 100px
- Example:

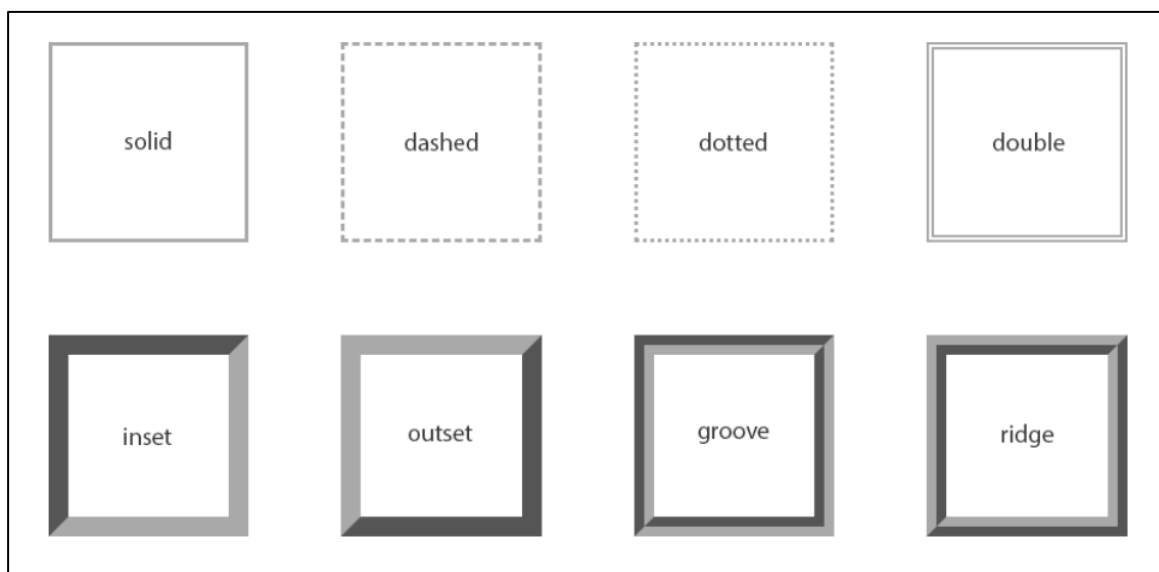
```
<html>
<head>
<style>
  div {
    padding: 25px 50px 75px 100px;
    background-color: pink;
  }
</style>
</head>
<body>
<h2>The padding shorthand property - 4 values</h2>
<div>This div element has a top padding of 25px, a right padding of 50px, a
bottom padding of 75px, and a left padding of 100px.</div>
</body>
</html>
```

Output:

This div element has a top padding of 25px, a right padding of 50px, a bottom padding of 75px, and a left padding of 100px.

CSS Border Properties:

- The CSS border properties allow you to define the border area of an element's box.
- Borders appear directly between the margin and padding of an element.
- The ***border-style*** property specifies what kind of border to display.
- The border-style property can have the following values: none, hidden, solid, dashed, dotted, double, inset, outset, groove, and ridge.



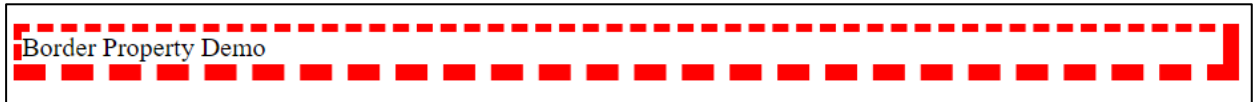
- The ***border-width*** property specifies the width of the border area.
 - It is a shorthand property for setting the thickness of all the four sides (top, right, bottom, left) of an element's border at the same time.
 - **Examples:**
 - `border-width: 10px; /* 10px on all the 4 border sides*/`
 - `border-width: 5px 20px; /* 5px top and bottom, 20px on the left and right*/`
 - `border-width: 25px 10px 4px 35px; /* 25px top, 10px right, 4px bottom and 35px left */`
 - The ***border-color*** property specifies the color of the border area.
 - This is also a shorthand property for setting the color of all the four sides of an element's border.
 - **Examples:**
 - `border-color: green;`
 - `border-color: red green blue yellow; /* red top, green right, blue bottom and yellow left */`
- The ***border*** property is a shorthand property for the following individual border properties:
 - `border-width`
 - `border-style` (required)
 - `border-color`
- **Examples:**
 - `border: 5px solid red;`
 - `border-left: 6px solid red;`
 - `border-bottom: 6px solid red;`
- **Example:**

```
<!DOCTYPE html>
<html>
<head>
  <style>
    p {
      border-style: dashed;
      border-color: red;
      border-width: 5px 10px 10px 5px;
    }
  </style>
</head>
<body>
<h2>The border Property</h2>
<p>Border Property Demo</p>
```

</body>

</html>

Output:



- **CSS Margin Properties:**

- The CSS margin properties allow you to set the spacing around the border of an element's box.
- An element's margin is not affected by its background-color, it is always transparent.
- CSS has properties for specifying the margin for each side of an element:
 - ***margin-top***
 - ***margin-right***
 - ***margin-bottom***
 - ***margin-left***
- All the margin properties can have the following values:
 - auto - the browser calculates the margin. Used to horizontally center the element within its container.
 - length - specifies a margin in px, pt, cm, etc.
 - % - specifies a margin in % of the width of the containing element
- **Example:**

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  border: 1px solid black;
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
  background-color: orange;
}
</style>
</head>
<body>
<div>This div element has a top margin of 100px, a right margin of 150px,
a bottom margin of 100px, and a left margin of 80px.</div>
</body>
</html>
```

Output:

This div element has a top margin of 100px, a right margin of 150px, a bottom margin of 100px, and a left margin of 80px.

- The ***margin*** property is a shorthand property for the following individual margin properties:
 - margin-top
 - margin-right
 - margin-bottom
 - margin-left

Example:

margin: 25px 50px 75px 100px;
top margin is 25px
right margin is 50px
bottom margin is 75px
left margin is 100px

➤ CSS Layout:

- CSS page layout techniques allow us to take elements contained in a web page and control where they're positioned relative to the following factors:
 - their default position in normal layout flow,
 - the other elements around them,
 - their parent container, and
 - the main viewport/window.
- Two such page layout techniques are:
 - Flexbox
 - Grid
- **'display'** property:
 - The display property is the most important CSS property for controlling layout.
 - It specifies if/how an element is displayed.
 - Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline.

- **CSS Syntax:**

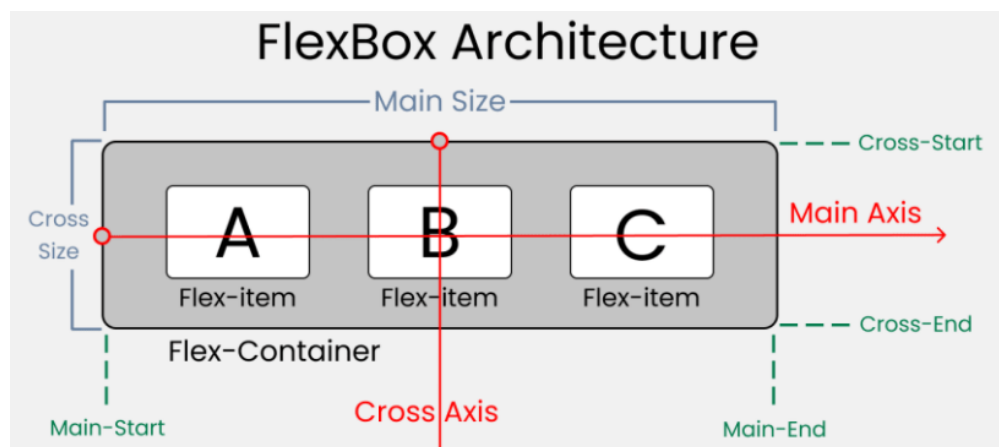
- **display: value;**

- *Property Values:* **inline | block | inline-block | flex | grid | inline-grid | inline-flex**

Value	Description
inline	Displays an element as an inline element. Any height and width properties will have no effect. Examples of inline elements: <code></code> , <code><a></code> , <code></code>
block	Displays an element as a block. It starts on a new line, and takes up the whole width Examples of block-level elements: <code><div></code> , <code><h1></code> - <code><h6></code> , <code><p></code>
flex	Displays an element as a block-level flex container
grid	Displays an element as a block-level grid container
inline-block	Displays an element as an inline-level block container. The element itself is formatted as an inline element, but you can apply height and width values
inline-flex	Displays an element as an inline-level flex container
inline-grid	Displays an element as an inline-level grid container

➤ **FLEXBOX LAYOUT:**

- CSS Flexbox is the short name for the Flexible Box Layout.
- It is a CSS 3 web layout model.
- It was designed as a one-dimensional layout model and is used to lay things out in one dimension; either as a row or as a column at a time.
- The flex layout allows responsive elements within a container to be automatically arranged depending upon screen size (or device).

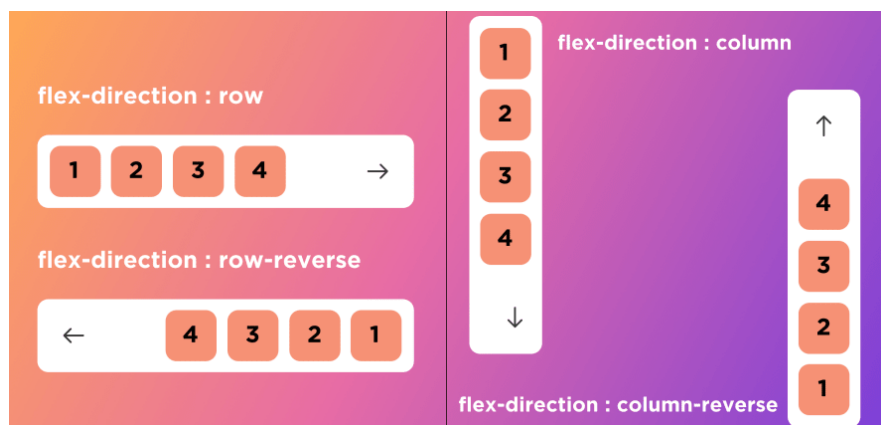


- Two key terminologies in Flexbox are the main axis and the cross axis.
 - A flex container's main axis(horizontal) is the primary axis along which these flex items are laid out, and the cross-axis(vertical) is perpendicular to it.
- **Flex Container:**
 - To use flexbox, apply **`display: flex`** or **`display: inline-flex`** to the parent element (container) of the elements you want to layout; all its direct children then become *flex items*.
 - *Example:*

```
.flex-container {
    display: flex;
}
```
- The flex container properties are:
 - flex-direction
 - flex-wrap
 - flex-flow
 - justify-content
 - align-items
 - align-content
- **The flex-direction Property:**
 - Defines in which direction the container wants to stack the flex items.
 - *Syntax:*

```
flex-direction: row | row-reverse | column | column-reverse;
```

 - row (default): stacks the flex items horizontally from left to right.
 - row-reverse: stacks the flex items horizontally but from right to left.
 - column - stacks the flex items vertically from top to bottom.
 - column-reverse - stacks the flex items vertically but from bottom to top.



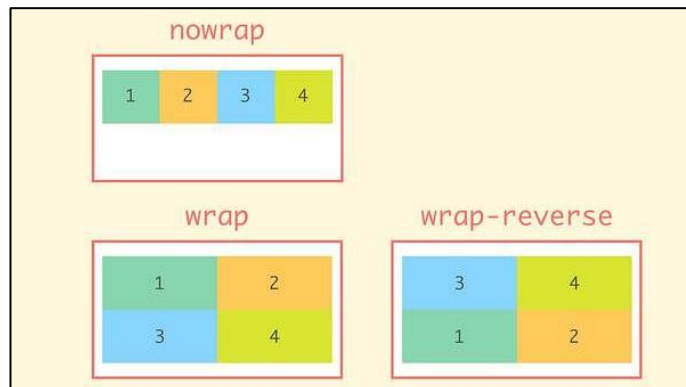
- **The flex-wrap Property:**

- Specifies whether flex items are forced onto one line or can wrap onto multiple lines.

- *Syntax:*

flex-wrap: nowrap | wrap | wrap-reverse;

- nowrap (default): flex items are laid out in a single line which may cause the flex container to overflow.
- wrap : flex items break into multiple lines.
- wrap-reverse: flex items are wrapped in reverse order



- **The flex-flow Property:**

- This property is a shorthand for the following CSS properties:

- flex-direction
- flex-wrap

- *Example:*

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

- **The justify-content Property:**

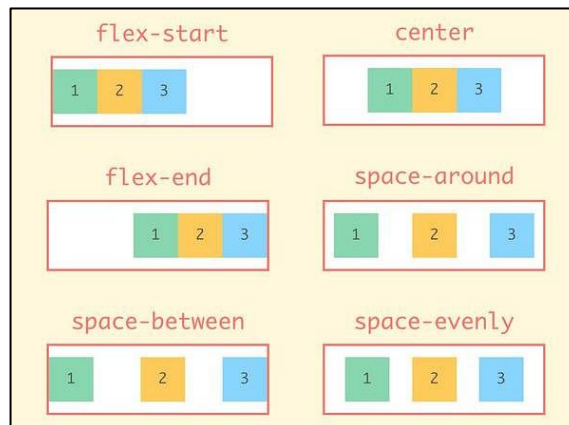
- It is used to align the flex items horizontally.

- *Syntax:*

justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly

- flex-start (default) - Items are positioned at the beginning of the container.
- flex-end - Items are positioned at the end of the container.
- center - Items are positioned in the center of the container.
- space-between - Items will have space between them.

- space-around - Items will have space before, between, and after them.
- space-evenly - Items will have equal space around them.



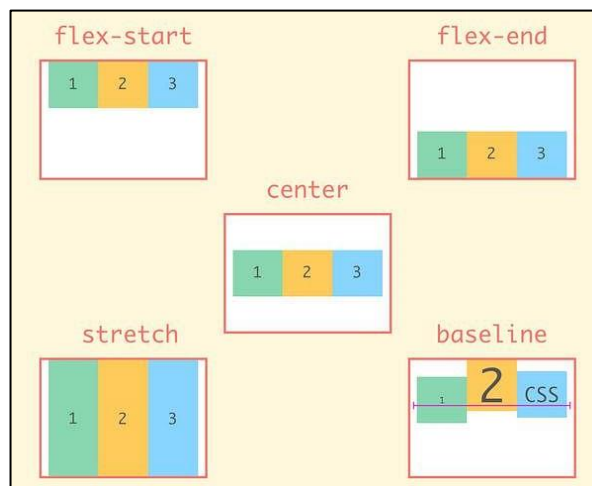
- **The align-items Property:**

- It is used to align the flex items vertically

- *Syntax:*

align-items: stretch | center | flex-start | flex-end | baseline;

- stretch (Default) - Items are stretched to fit the container
- center - Items are positioned at the center of the container
- flex-start - Items are positioned at the beginning of the container
- flex-end - Items are positioned at the end of the container
- baseline - Items are positioned at the baseline of the container



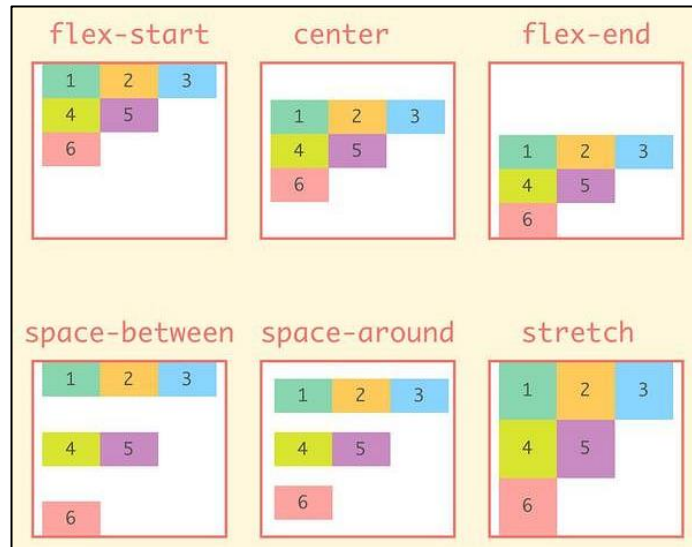
- **The align-content Property:**

- This property modifies the behavior of the flex-wrap property.
- It is similar to align-items, but instead of aligning flex items, it aligns flex lines.

- *Syntax:*

align-content: stretch | center | flex-start | flex-end | space-between | space-around;

- stretch (Default): Lines stretch to take up the remaining space
- center: Lines are packed toward the center of the flex container
- flex-start: Lines are packed toward the start of the flex container
- flex-end: Lines are packed toward the end of the flex container
- space-between: Lines are evenly distributed in the flex container
- space-around: Lines are evenly distributed in the flex container, with half-size spaces on either end
- space-evenly: Lines are evenly distributed in the flex container, with equal space around them



- The flex item properties are:
 - order
 - flex-grow
 - flex-shrink
 - flex-basis
 - flex
 - align-self

Property	Description
order	Specifies the order of the flex items inside the same container. The value must be an integer, default value is 0.
flex-grow	Specifies how much a flex item will grow relative to the rest of the flex items inside the same container. The value must be a number, default value is 0.
flex-shrink	Specifies how much a flex item will shrink relative to the rest of the flex items inside the same container. The value must be a number, default value is 1.
flex-basis	Specifies the initial length of a flex item

flex	A shorthand property for the flex-grow, flex-shrink, and the flex-basis properties
align-self	Specifies the alignment for the selected item inside the flexible container. It overrides the default alignment set by the container's align-items property. Values: flex-start, flex-end, center, baseline, stretch, auto

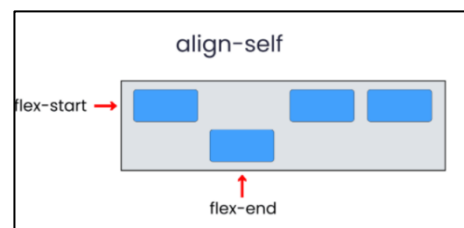
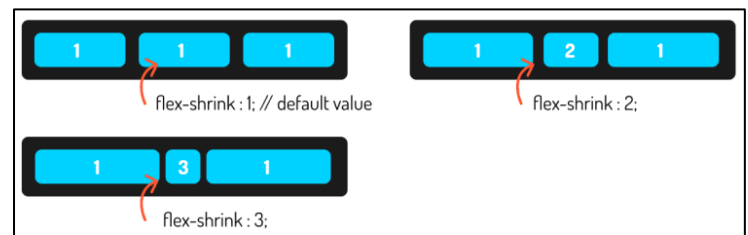
(default)



order: 1



order: 3

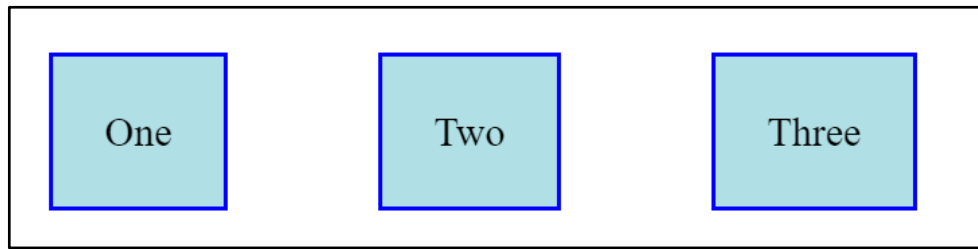


Examples:

```
1. <html>
    <head>
    <style>
        .container {
            display: flex;
        }
        .container div {
            margin: 30px;
            background-color: powderblue;
            padding: 20px;
            border: 2px solid blue;
        }
    </style>
    </head>
    <body>
        <div class="container">
            <div>One</div>
            <div>Two</div>
            <div>Three</div>
```

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

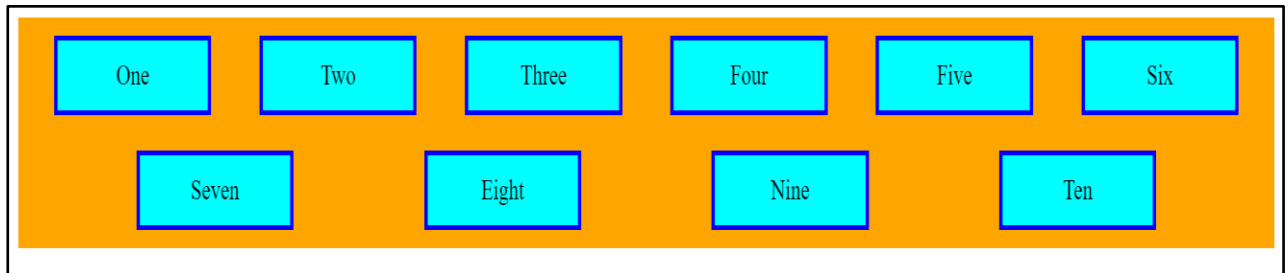
Output:



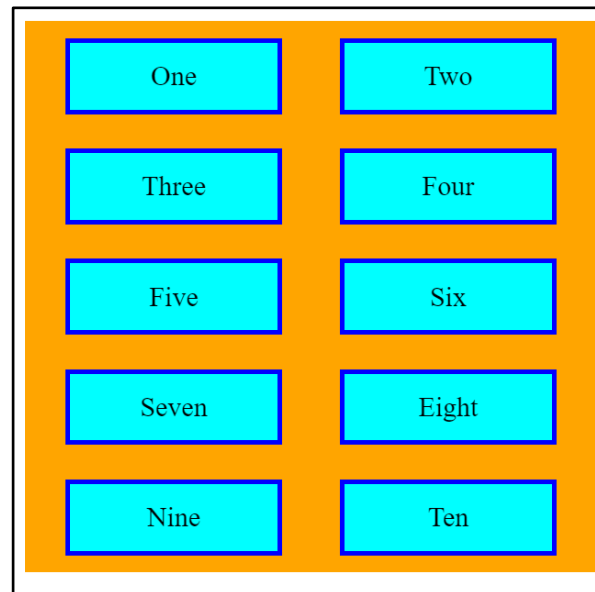
```
2. <html>
    <head>
    <style>
        .container {
            display: flex;
            flex-direction: row;
            flex-wrap: wrap;
            justify-content: space-evenly;
            background-color: orange;
        }
        .container div
        {
            margin: 10px;
            background: cyan;
            padding: 10px;
            border: 3px solid blue;
            width: 100px;
            text-align: center;
        }
    </style>
</head>
<body>
    <div class="container">
        <div>One</div>
        <div>Two</div>
        <div>Three</div>
        <div> Four </div>
        <div> Five </div>
        <div> Six </div>
        <div> Seven </div>
        <div> Eight </div>
        <div> Nine </div>
        <div> Ten </div>
    </div>
</body>
```

Output:

When the browser is not re-sized:

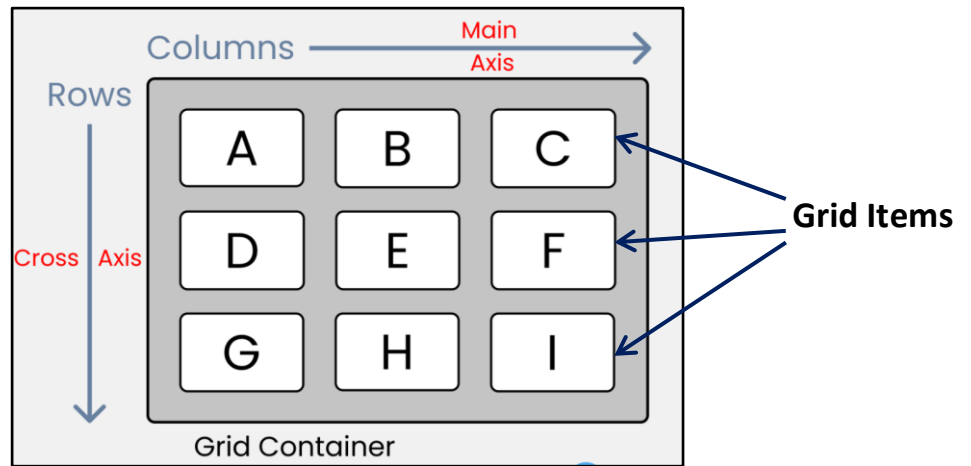


When the browser is re-sized:



➤ **GRID LAYOUT:**

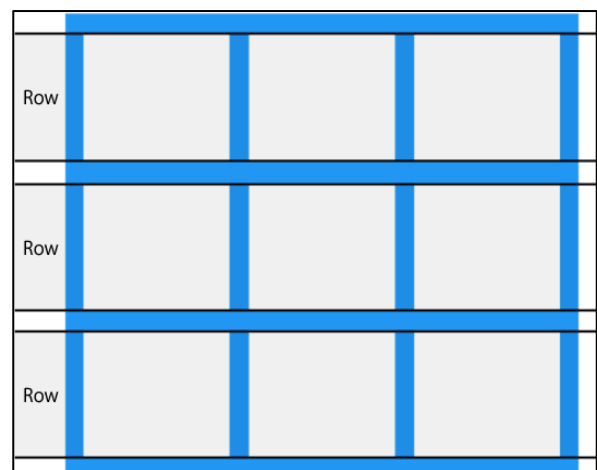
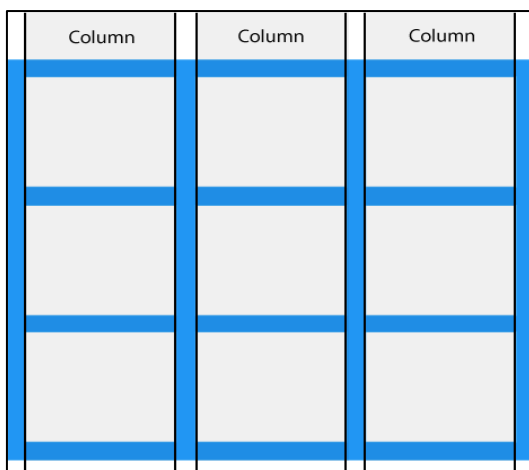
- The CSS Grid Layout offers a two-dimensional grid layout system, with rows and columns, making it easier to design web pages.
- It is a CSS 3 web layout model.
- The Grid layout allows responsive elements within a container to be automatically arranged depending upon screen size (or device).
- A grid layout consists of a parent element (Grid Container), with one or more child elements (Grid Items).



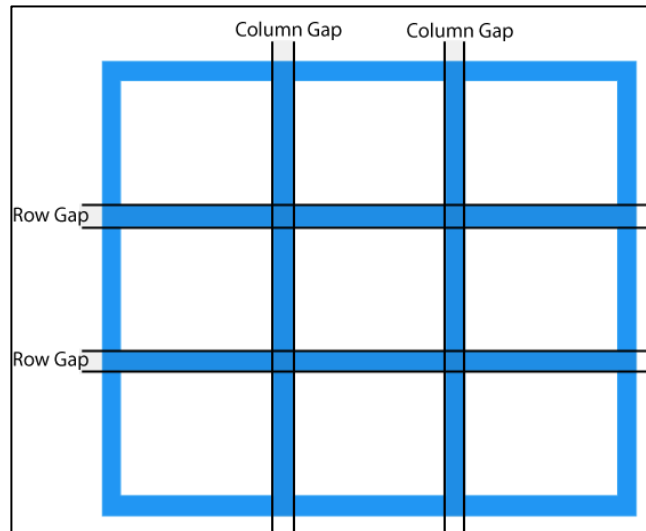
BASIC TERMINOLOGY:

- **Grid Container:**
 - To make an HTML element behave as a grid container, you have to set the ***display*** property to ***grid*** or ***inline-grid***.
 - *Syntax:*

```
.container {
  display: grid | inline-grid;
}
```
 - All direct children of the grid container automatically become *grid items*.
 - Grid containers consist of grid items, placed inside columns and rows.
- **Grid Columns:** The vertical lines of grid items are called *columns*.
- **Grid Rows:** The horizontal lines of grid items are called *rows*.

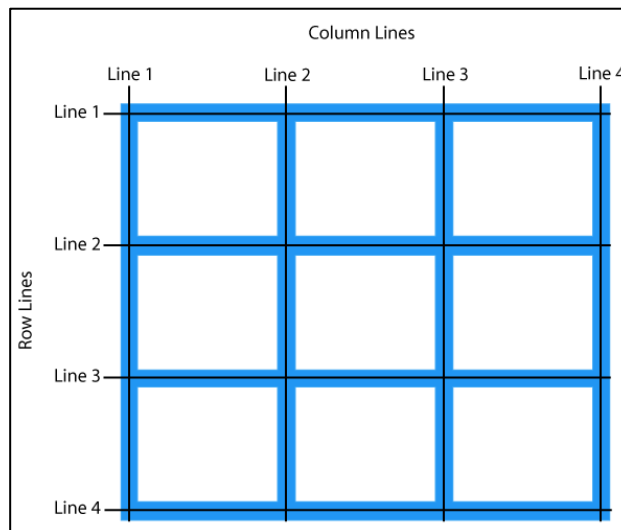


- **Grid Gaps:** The spaces between each column/row are called gaps or gutters.



- **Grid Lines:**

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



- **Grid Track:** A grid track is the space between two adjacent grid lines.
- **The fr unit:** Fr is a fractional unit. It represents a fraction of the available space in the grid container. Example: 1fr = 1 part of the available space.

CSS GRID CONTAINER/PARENT PROPERTIES:

1. grid-template-columns:

- Defines the number of columns in a grid layout, and it can define the width of each column.
- The value is a space-separated-list, where each value defines the width of the respective column.
- *Examples:*
 - `grid-template-columns: 90px 50px 120px;`
 - `grid-template-columns: 1fr 2fr 1fr;`

- `grid-template-columns: 100px auto 200px;`
- `grid-template-columns: auto auto auto;`

2. **grid-template-rows:**

- Defines the height of each row in a grid layout, and it can define the height of each row.
- The value is a space-separated-list, where each value defines the height of the respective row.
- *Examples:*
 - `grid-template-rows: 1fr 2fr;`
 - `grid-template-rows: 50px 100px;`
 - `grid-template-rows: 100px auto;`

3. **column-gap:**

- Defines the size of the gap between the columns in a grid layout.
- Syntax: `column-gap: length;`

4. **row-gap:**

- Defines the size of the gap between the rows in a grid layout.
- Syntax: `row-gap: length;`

5. **gap property:**

- defines the size of the gap between the rows and columns in a grid layout, and is a shorthand property for the following properties:
 - `row-gap`
 - `column-gap`
- Syntax: `gap: row-gap column-gap;`

6. **grid-template-areas:**

- Specifies areas within the grid layout.
- We can name grid items by using the `grid-area` property and then reference to the name in the `grid-template-areas` property.
- A period (`.`) refer to a grid item with no name.
- Syntax: `grid-template-areas: none|itemnames;`

7. **justify-items:**

- Used to position grid-items inside grid containers along the X-Axis (row).
- This value applies to all grid items inside the container.
- Syntax: `justify-items: start | end | center | stretch;`

8. **align-items:**

- Used to position grid-items inside the grid container along the Y-Axis (column)
- This value applies to all grid items inside the container.

- Syntax: `align-items: start | end | center | stretch;`

9. justify-content:

- Used to position grid inside the grid container along the X-Axis (row).
- The grid's total width has to be less than the container's width for the justify-content property to have any effect.
- Syntax: `justify-content: start | end | center | stretch | space-around | space-between | space-evenly;`

10. align-content:

- Used to position grid inside the grid container along the Y-Axis (column).
- Syntax: `align-content: start | end | center | stretch | space-around | space-between | space-evenly;`

CSS GRID ITEMS PROPERTIES:

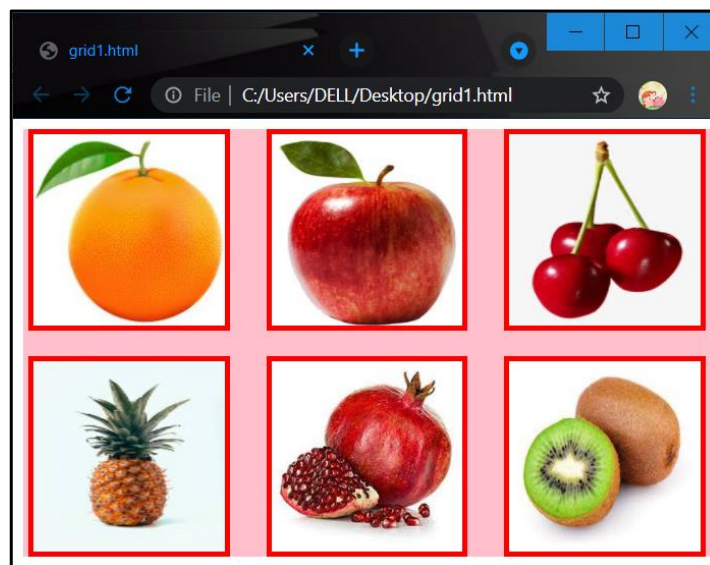
- A grid container contains grid items.
 - By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.
- 1. grid-column-start:**
 - Defines on which column-line the item will start.
 - Syntax: `grid-column-start: auto|span n|column-line;`
 - 2. grid-column-end:**
 - Defines how many columns an item will span, or on which column-line the item will end.
 - Syntax: `grid-column-end: auto|span n|column-line;`
 - 3. grid-column:**
 - specifies a grid item's size and location in a grid layout, and is a shorthand property for the following properties:
 - `grid-column-start`
 - `grid-column-end`
 - Syntax: `grid-column: grid-column-start / grid-column-end;`
 - 4. grid-row-start:**
 - defines on which row-line the item will start.
 - Syntax: `grid-row-start: auto|row-line;`
 - 5. grid-row-end:**
 - defines how many rows an item will span, or on which row-line the item will end
 - Syntax: `grid-row-end: auto|row-line|span n;`
 - 6. grid-row:**
 - Specifies a grid item's size and location in a grid layout, and is a shorthand property for the following properties:
 - `grid-row-start`
 - `grid-row-end`
 - Syntax: `grid-row: grid-row-start / grid-row-end;`
 - 7. grid-area property or naming grid items:**
 - When we use grid-row and grid-column property together it becomes a grid-area. So, grid-area is also a shorthand property for grid-row and grid-column property.
 - The grid-area property can also be used to assign a name to a grid item. Named grid items can then be referenced to by the grid-template-areas property of the grid container.

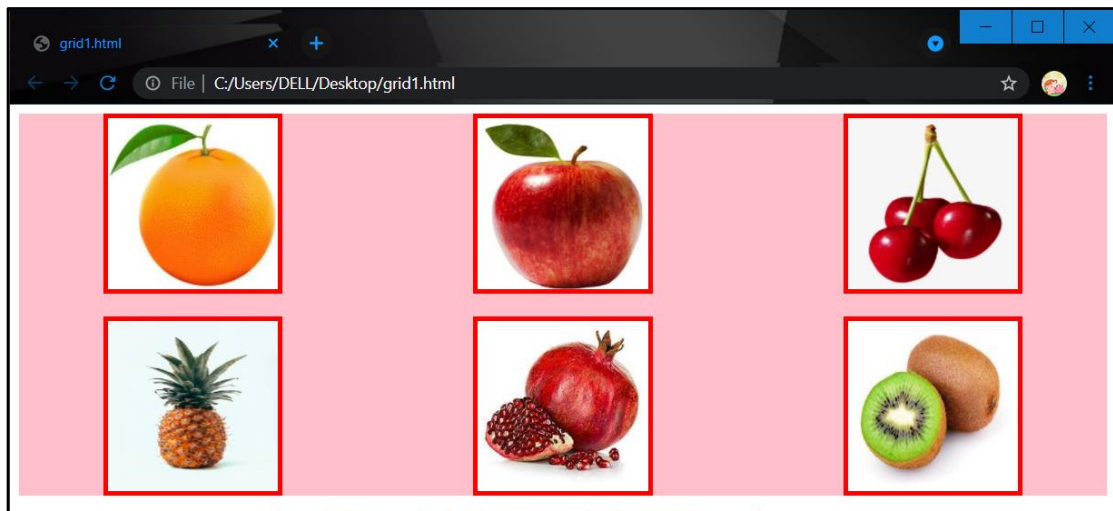
- Syntax: `grid-area: grid-row-start / grid-column-start / grid-row-end / grid-column-end | itemname;`

Examples:

```
1. <html>
  <style>
    .container
    {
      display:grid;
      grid-template-columns: 1fr 1fr 1fr;
      gap: 20px;
      justify-items:center;
      background-color:pink;
    }
    .container img
    {
      width: 150px;
      height: 150px;
      border: 4px solid red;
    }
  </style>
  <body>
    <div class="container">
      
      
      
      
      
      
    </div>
  </body>
</html>
```

Output:





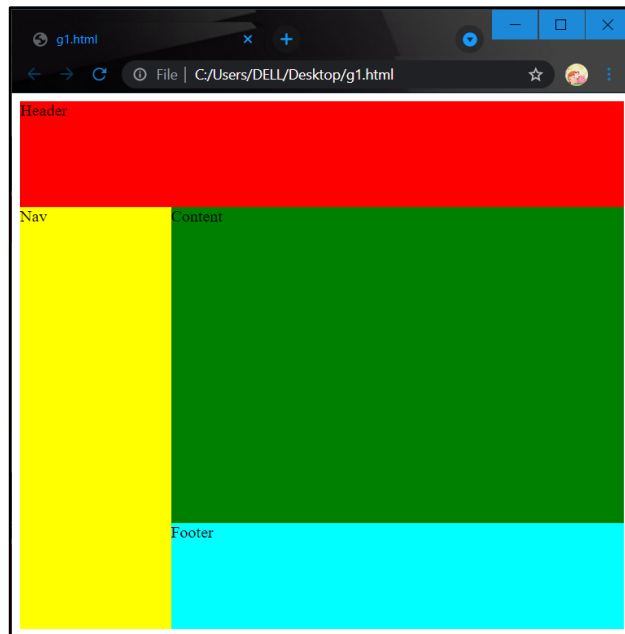
```

2. <html>
    <head>
    <style>
        .container
        {
            display: grid;
            grid-template-columns: 1fr 3fr;
            grid-template-rows: 20% 60% 20%;
            height: 100%;
        }
        .i1
        {
            background-color:red;
            grid-column-start: 1;
            grid-column-end: 3;    // grid-column: 1/3; OR grid-column: 1/span 2
        }
        .i2
        {
            background-color: yellow;
            grid-row: 2/span 2;
        }
        .i3
        {
            background-color: green;
            grid-column: 2/span 2;
        }
        .i4
        {
            background-color: cyan;
            grid-column: 2/span 2;
        }
    </style>
    </head>
    <body>

```

```
<div class="container">
  <div class="i1">Header</div >
  <div class="i2">Nav</div>
  <div class="i3">Content</div>
  <div class="i4">Footer</div>
</div>
</body>
</html>
```

Output:



➤ **RESPONSIVE WEB DESIGN:**

- Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices (desktops, tablets, and phones).
- It is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.
- A responsive web design will automatically adjust for different screen sizes and viewports.



VIEWPORT:

- The viewport is the user's visible area of a web page.
- In web browser terms, it is generally the same as the browser window, excluding the UI, menu bar, etc.
- It is the part of the document you are viewing.
- The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.
- To create a responsive website, add **<meta>** tag to all your web pages.

Setting The Viewport:

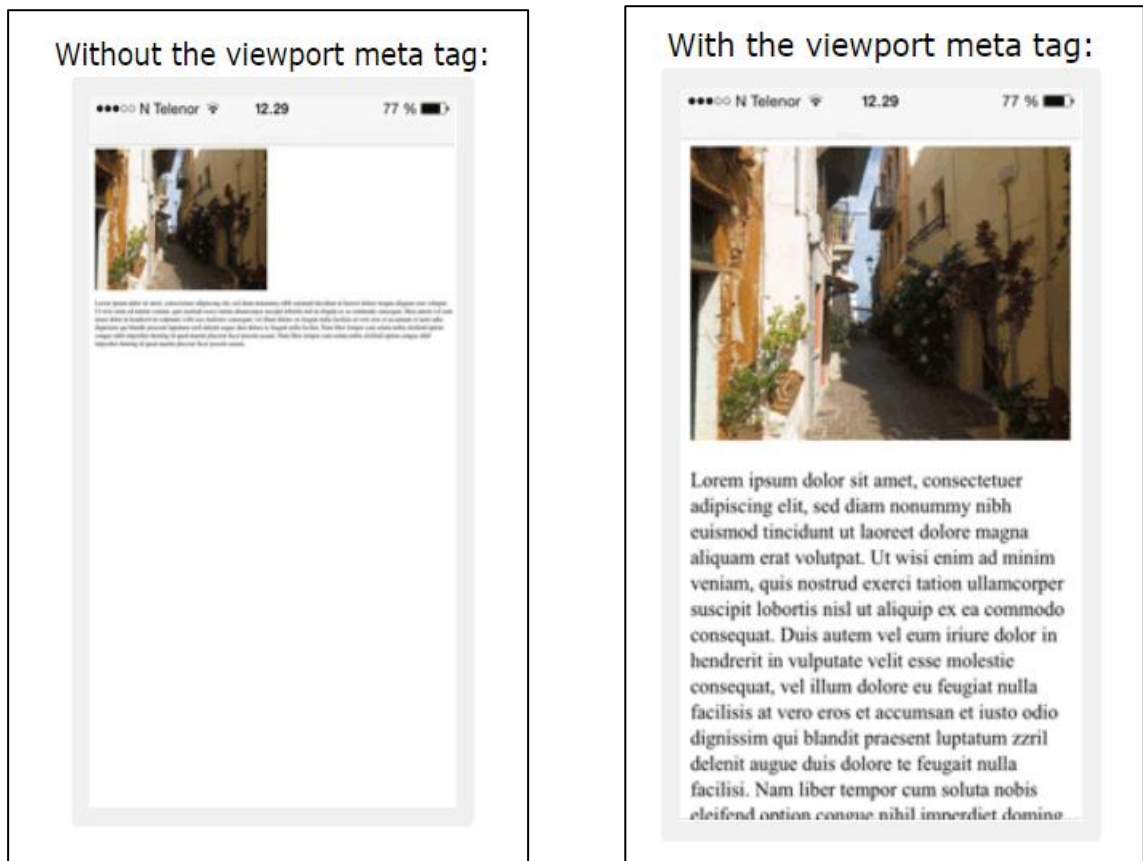
- HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.

Syntax:

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

- This will set the viewport of your page, which will give the browser instructions on how to control the page's dimensions and scaling.
- The *width=device-width* part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

- The *initial-scale=1.0* part sets the initial zoom level when the page is first loaded by the browser.



Responsive Images:

- Responsive images are images that scale nicely to fit any browser size.
- Using the width Property
 - If the CSS width property is set to 100%, the image will be responsive and scale up and down.
 - Syntax: ``
- Using the max-width Property
 - If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size.
 - Syntax: ``

Responsive Text:

- The text size can be set with a "vw" unit, which means the "viewport width".
- That way the text size will follow the size of the browser window.
- Viewport is the browser window size. 1vw = 1% of viewport width.
- Example: `<h1 style="font-size:10vw">Hello World</h1>`

MEDIA QUERIES:

- Media query is a CSS technique introduced in CSS3.
- In addition to resize text and images, it is also common to use media queries in responsive web pages.
- With media queries you can define completely different styles for different browser sizes.
- Media queries can be used to check many things, such as:
 - width and height of the viewport
 - width and height of the device
 - orientation (is the tablet/phone in landscape or portrait mode)
 - resolution
- It uses the *@media* rule to include a block of CSS properties only if a certain condition is true.
- **@media rule:**
 - The **@media** rule is used in media queries to apply different styles for different media types/devices.
 - *Syntax:*
**@media not|only mediatype and (mediafeature and|or|not mediafeature) {
CSS-Code;
}**
 - **not:** The not keyword inverts the meaning of an entire media query.
 - **only:** The only keyword prevents older browsers that do not support media queries with media features from applying the specified styles. It has no effect on modern browsers.
 - **and:** The and keyword combines a media feature with a media type or other media features.

- **Media Types:**

Value	Description
all	Default. Used for all media type devices
print	Used for printers
screen	Used for computer screens, tablets, smart-phones etc.
speech	Used for screenreaders that "reads" the page out loud

- **Some Media Features:**

- *height* : The viewport height.
- *max-height*: The maximum height of the display area, such as a browser window.

- *max-width*: The maximum width of the display area, such as a browser window.
- *min-height*: The minimum height of the display area, such as a browser window.
- *min-width*: The minimum width of the display area, such as a browser window.
- *orientation*: The orientation of the viewport (landscape or portrait mode)
- *width*: The viewport width.

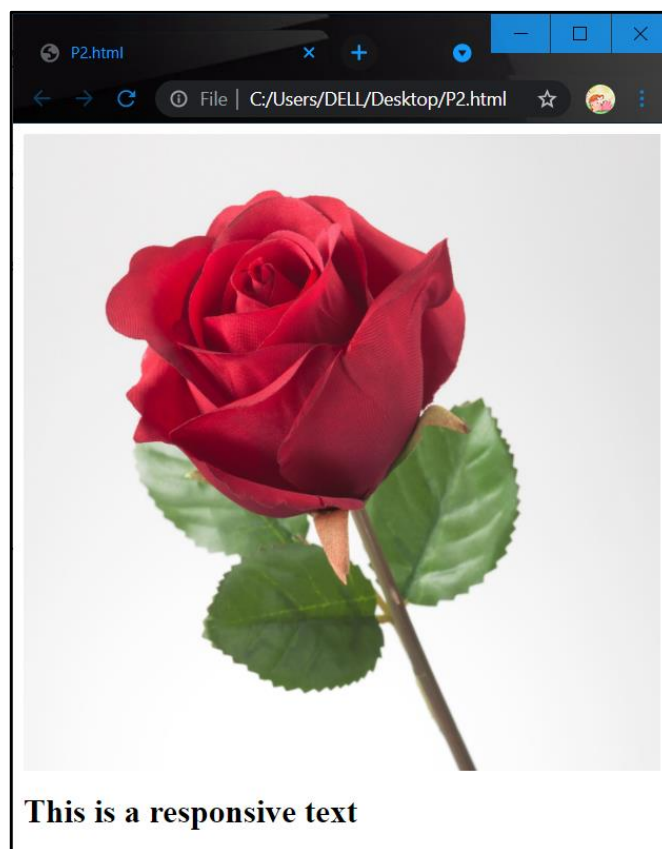
Examples:

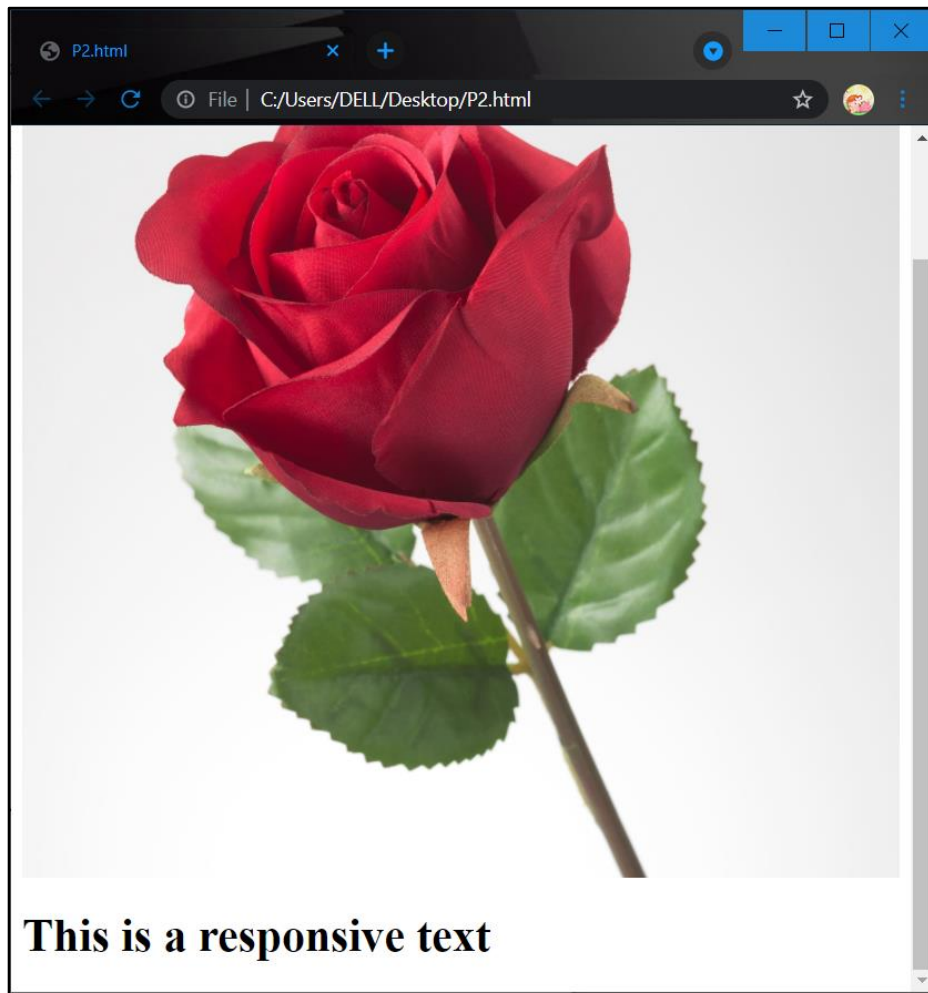
1. Responsive Text and Images:

```
<html>
<head>
    <meta name="viewport" content="width=device-width,initial-
scale=1.0">
</head>
<body>

<h1 style="font-size:5vw">This is a responsive text</h1>
</body>
</html>
```

Output:





2. Media Queries:

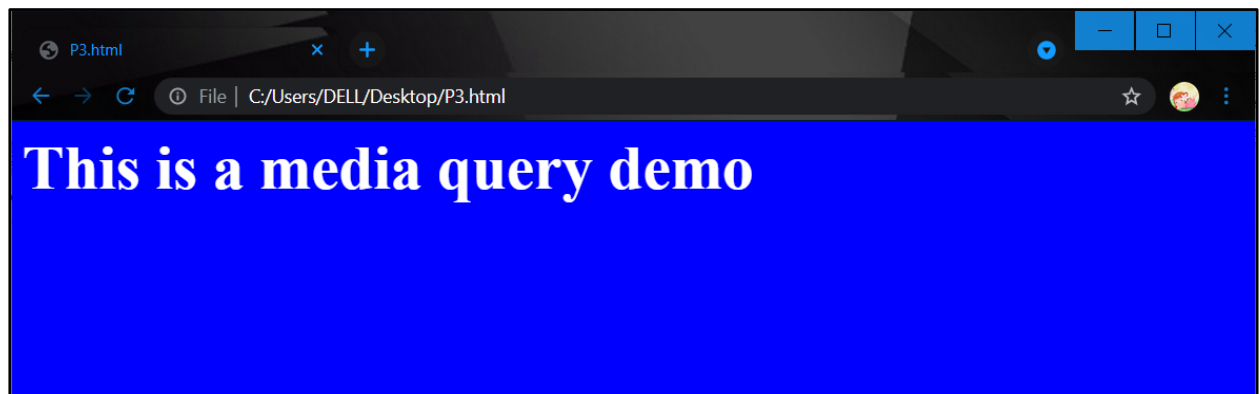
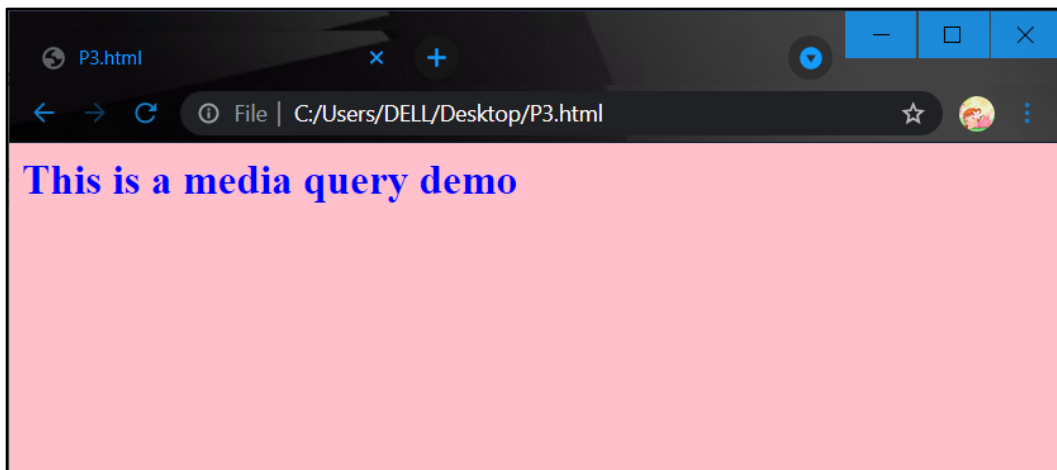
```
<html>
<head>
  <meta name="viewport" content="width=device-width,initial-
    scale=1.0">
<style>
  body
  {
    background-color: blue;
  }
  h1
  {
    color:white;
    font-size:5vw;
  }
  @media screen and (max-width:700px) //breakpoint
  {
    body
    {
      background-color:pink;
    }
    h1
```

```

        {
            color:blue;
            font-size:4vw;
        }
    }
</style>
</head>
<body>
    <h1>This is a media query demo</h1>
</body>
</html>

```

Output:



3. Media Queries with multiple breakpoints:

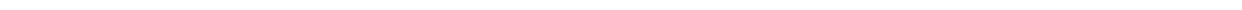
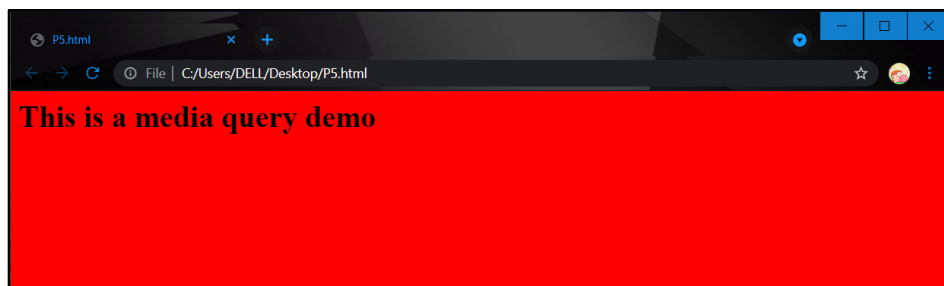
```

<html>
<head>
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
<style>
    body
    {
        background-color: orange;
    }
    @media screen and (max-width:1000px) //breakpoint 1

```

```
{
  body
  {
    background-color:red;
  }
}
@media screen and (max-width:600px) //breakpoint 2
{
  body
  {
    background-color: cyan;
  }
}
</style>
</head>
<body>
  <h1>This is a media query demo</h1>
</body>
</html>
```

Output:



➤ **CSS TRANSITIONS:**

- CSS transitions allow you to change property values smoothly, over a given duration.
- CSS transitions let you decide which properties to animate (by listing them explicitly), when the animation will start (by setting a delay), how long the transition will last (by setting a duration), and how the transition will run.
- They provide a way to control animation speed when changing CSS properties. Instead of having property changes take effect immediately, you can cause the changes in a property to take place over a period of time.

Defining transitions:

- CSS Transitions are controlled using the shorthand *transition* property.
- To create a transition effect, you 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.

CSS Transition Properties:

The following table lists all the CSS transition properties:

Property	Description
transition	A shorthand property for setting <i>the transition-property, transition-duration, transition-timing-function, and transition-delay.</i>
transition-delay	Specifies a delay (in seconds) for the transition effect. Syntax: <code>transition-delay: time</code>
transition-duration	Specifies how many seconds or milliseconds a transition effect takes to complete. Syntax: <code>transition-duration: time</code>
transition-property	Specifies the name of the CSS property the transition effect is for. Syntax: <code>transition-property: none all property</code>
transition-timing-function	Specifies the speed curve of the transition effect Syntax: <code>transition-timing-function: linear ease ease-in ease-out ease-in-out step-start step-end steps(int,start end) cubic-bezier(n,n,n,n)</code> <ul style="list-style-type: none">• <i>ease</i> - specifies a transition effect with a slow start, then

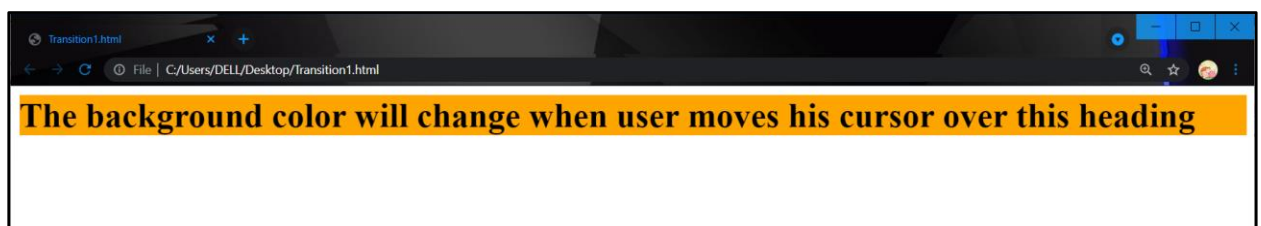
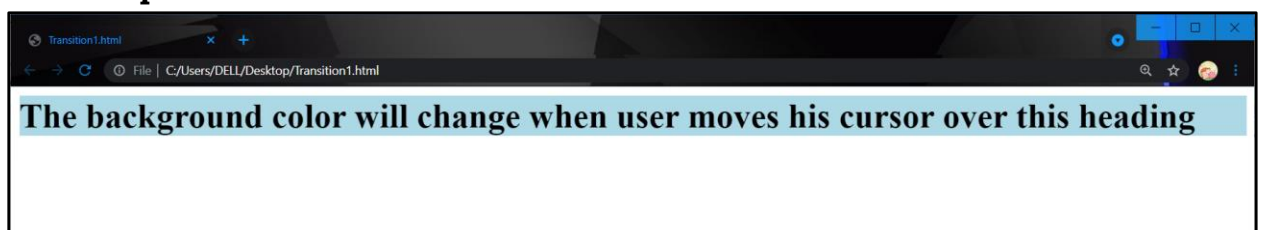
	<p>fast, then end slowly (this is default)</p> <ul style="list-style-type: none"> • <i>linear</i> - specifies a transition effect with the same speed from start to end • <i>ease-in</i> - specifies a transition effect with a slow start • <i>ease-out</i> - specifies a transition effect with a slow end • <i>ease-in-out</i> - specifies a transition effect with a slow start and end • <i>cubic-bezier(n,n,n,n)</i> - lets you define your own values in a cubic-bezier function
--	--

Examples:

1.

```
<html>
<head>
<style>
h1
{
    background-color:lightblue;
    transition-property: background-color;
}
h1:hover
{
    background-color:orange;
}
</style>
</head>
<body>
<h1>The background color will change when user moves his cursor over this heading</h1>
</body>
</html>
```

Output:

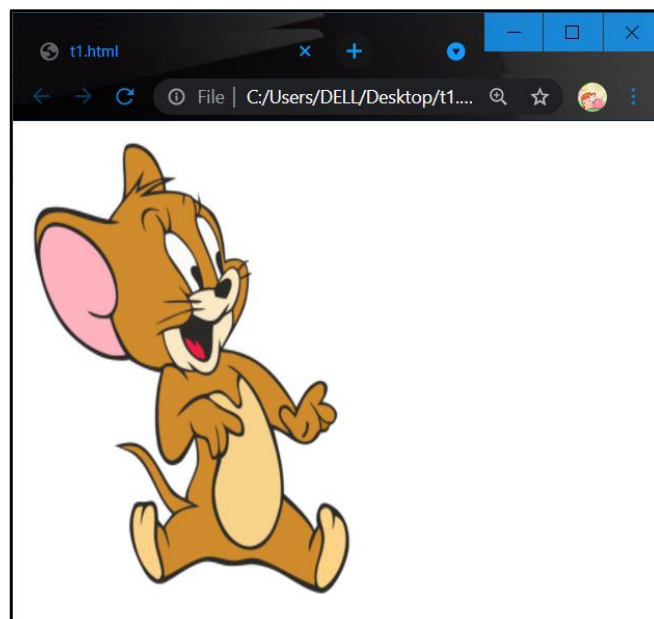
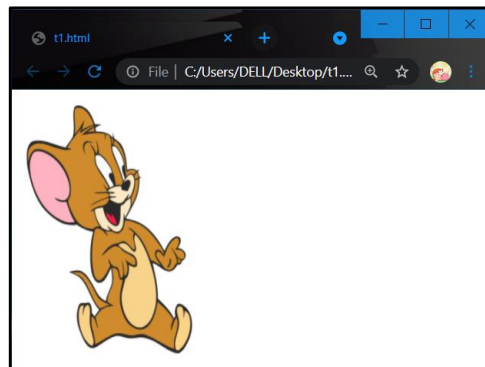


2.

```
<html>
<head>
<style>
img
{
    transition-duration: 10s;
    transition-property: width height;
    width: 100px;
    height: 150px;
}
img:hover
{
    width: 300px;
    height: 400px;
}
</style>
</head>
<body>

</body>
</html>
```

Output:



➤ **CSS ANIMATIONS:**

- CSS3 allows animation of HTML elements without using JavaScript or Flash!.
- CSS animations make it possible to animate transitions from one CSS style configuration to another.
- “An animation lets an element gradually change from one set of CSS styles to another.”
- During the animation, you can change the set of CSS styles many times.
- CSS Animations consist of two components:
 - A style describing the CSS animation and
 - A set of keyframes that indicate the start and end states of the animation’s style, as well as possible intermediate waypoints.
- Steps in creating Animations:
 1. Configuring the animation using animation properties.
 2. Defining the animation sequence using keyframes.

Configuring the animation:

- To create a CSS animation sequence, style the element you want to animate with the animation property or its sub-properties. This lets you configure the timing, duration, and other details of how the animation sequence should progress.
- Use the animation properties to control the appearance of the animation, and also to bind the animation to selectors.

CSS Animation Properties:

Property	Description
animation	A shorthand property for setting all the animation properties. <code>animation: name duration timing-function delay iteration-count direction fill-mode play-state;</code>
animation-delay	Specifies a delay for the start of an animation in seconds. <code>animation-delay: time</code> Seconds(s) or Milliseconds(s) and default is 0
animation-direction	Specifies whether an animation should be played forwards, backwards or in alternate cycles. Values: <ul style="list-style-type: none">• normal - The animation is played as normal (forwards). This is default• reverse - The animation is played in reverse direction (backwards)

	<ul style="list-style-type: none"> • alternate - The animation is played forwards first, then backwards • alternate-reverse - The animation is played backwards first, then forwards <p>animation-direction: normal reverse alternate alternate-reverse</p>
animation-duration	Specifies how long time an animation should take to complete one cycle. Default value is 0 seconds
animation-fill-mode	<p>Specifies a style for the element when the animation is not playing (before it starts, after it ends, or both).</p> <p>animation-fill-mode: none forwards backwards both</p> <p>Values:</p> <ul style="list-style-type: none"> • none - Default value. Animation will not apply any styles to the element before or after it is executing. • forwards - The element will retain the style values that is set by the last keyframe (depends on animation-direction and animation-iteration-count). • backwards - The element will get the style values that is set by the first keyframe (depends on animation-direction), and retain this during the animation-delay period. • both - The animation will follow the rules for both forwards and backwards, extending the animation properties in both directions.
animation-iteration-count	<p>Specifies the number of times an animation should be played.</p> <p>animation-iteration-count: <i>number</i> infinite</p> <p>Values: number, infinite</p>
animation-name	<p>Specifies the name of the @keyframes animation.</p> <p>animation-name: <i>keyframename</i></p>
animation-play-state	<p>Specifies whether the animation is running or paused.</p> <p>animation-play-state: paused running</p>
animation-timing-function	<p>Specifies the speed curve of the animation.</p> <p>animation-timing-function: linear ease ease-in ease-out ease-in-out step-start step-end steps(int,start end) cubic-bezier(<i>n,n,n,n</i>)</p>

	<p>Values:</p> <ul style="list-style-type: none"> • ease - Specifies an animation with a slow start, then fast, then end slowly (this is default) • linear - Specifies an animation with the same speed from start to end • ease-in - Specifies an animation with a slow start • ease-out - Specifies an animation with a slow end • ease-in-out - Specifies an animation with a slow start and end • cubic-bezier(n,n,n,n) - Lets you define your own values in a cubic-bezier function
--	---

Defining the animation sequence using keyframes:

- Once you've configured the animation's timing, you need to define the appearance of the animation. This is done by establishing two or more keyframes using the `@keyframes` at-rule.
- To use CSS animation, you must first specify some keyframes for the animation.
- Keyframes hold what styles the element will have at certain times.
- Each keyframe describes how the animated element should render at a given time during the animation sequence.
- Keyframes are paired with the animation property to set the duration, timing function, delay and direction.
- keyframes use a `<percentage>` to indicate the time during the animation sequence at which they take place. 0% indicates the first moment of the animation sequence, while 100% indicates the final state of the animation.

@keyframes rule:

- The `@keyframes` CSS at-rule controls the intermediate steps in a CSS animation sequence by defining styles for keyframes along the animation sequence.
- To use keyframes, create a **`@keyframes`** rule with a name that is then used by the animation-name property to match an animation to its keyframe declaration.

Syntax:

`@keyframes animationname {keyframes-selector {css-styles;}}`

- Each @keyframes rule contains a style list of keyframe selectors, which specify percentages along the animation when the keyframe occurs, and a block containing the styles for that keyframe.

Value	Description
animationname	Required. Defines the name of the animation.
keyframes-selector	Required. Percentage of the animation duration. values: 0-100% from (same as 0%) to (same as 100%) You can have many keyframes-selectors in one animation.
css-styles	Required. One or more legal CSS style properties

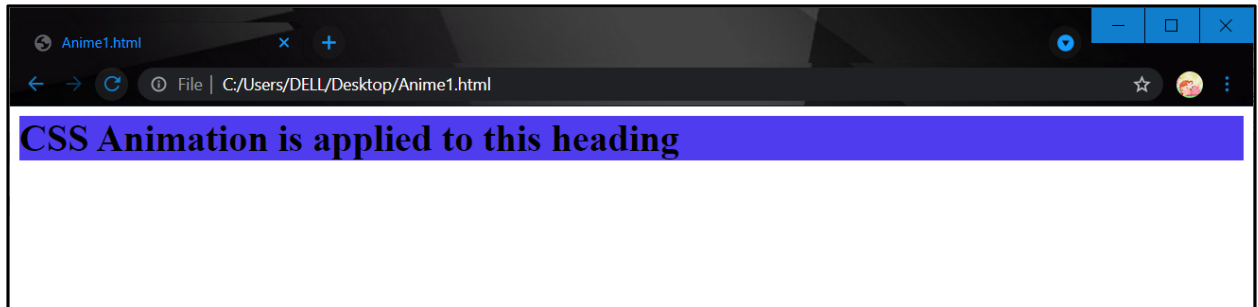
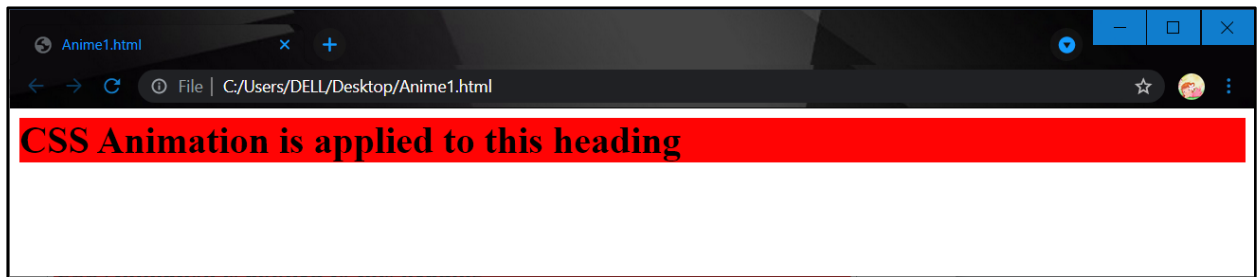
Examples:

```

1.  <html>
    <head>
    <style>
    h1
    {
        animation-name : color;
        animation-duration : 10s;
        animation-iteration-count : infinite;
        animation-delay: 1s;
    }
    @keyframes color
    {
        0% {background-color: blue;}
        25% {background-color: pink;}
        50% {background-color: red;}
        75% {background-color: cyan;}
        100% {background-color: green;}
    }
    </style>
    </head>
    <body>
    <h1>CSS Animation is applied to this heading</h1>
    </body>
    </html>

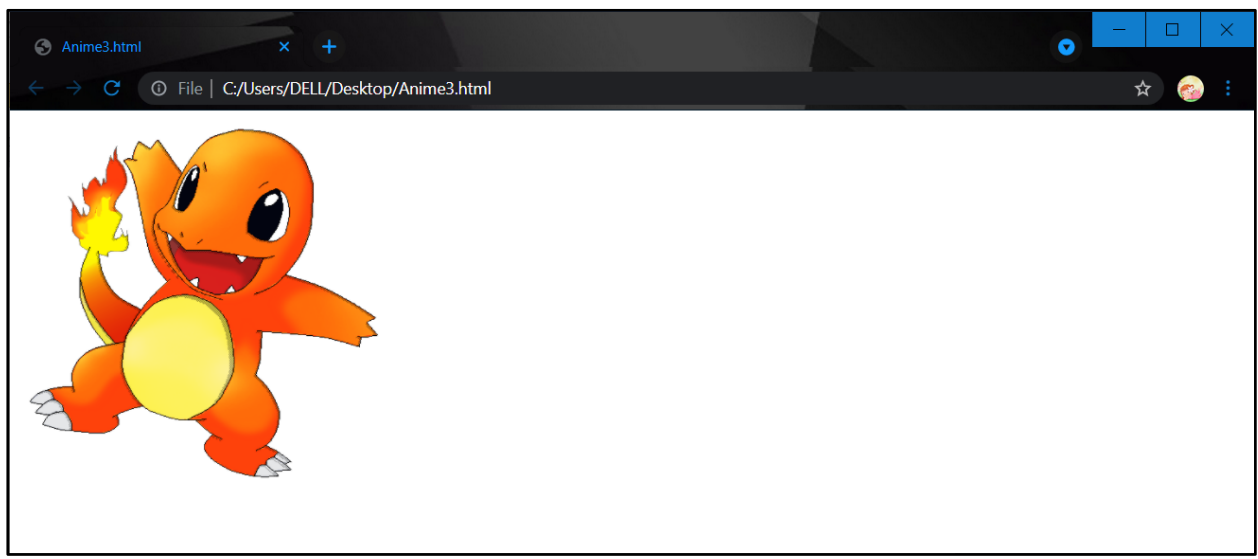
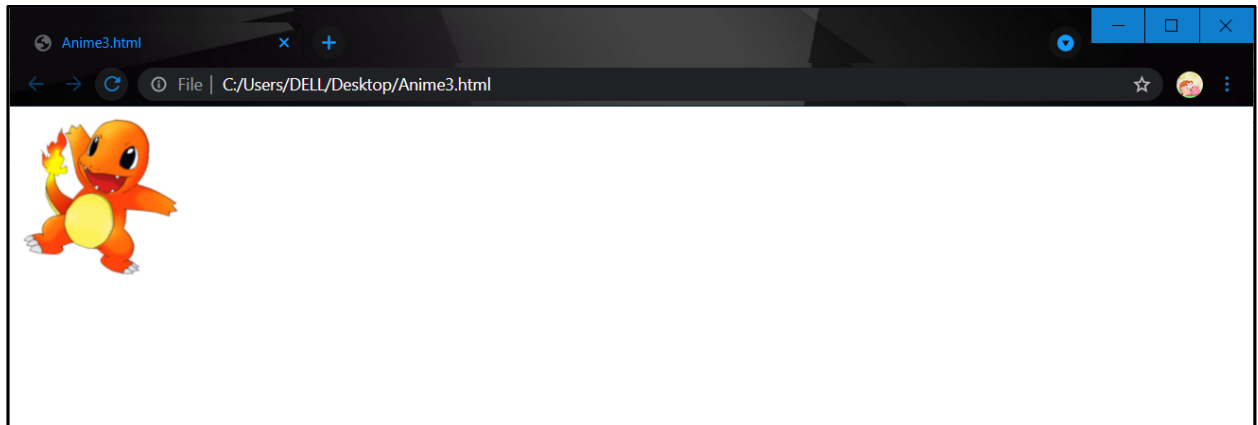
```

Output:



```
2. <html>
    <head>
    <style>
    img
    {
        width: 100px;
        height: 100px;
        animation-name: change;
        animation-duration: 20s;
    }
    @keyframes change
    {
        10% {width:150px; height:150px}
        25% {width:200px; height:200px; }
        50% {width:250px; height:250px; }
        75% {width:300px; height:300px; }
        100% {width:350px; height:350px; }
    }
    </style>
    </head>
    <body>
    
    </body>
    </html>
```

Output:



➤ **CSS FRAMEWORK: BOOTSTRAP**

- Bootstrap was developed by Mark Otto and Jacob Thornton at Twitter, and released as an open source product in August 2011 on GitHub.
- In June 2014 Bootstrap was the No.1 project on GitHub.
- Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.
- It is completely free to download and use – Open Source.
- Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation and other interface components., as well as optional JavaScript plugins.
- It gives you the ability to easily create responsive designs.
- **Bootstrap 5** is the newest version of Bootstrap; with new components, faster stylesheet and more responsiveness.
- Bootstrap 5 supports the latest, stable releases of all major browsers and platforms.

- The main differences between Bootstrap 5 and Bootstrap 3 & 4, is that Bootstrap 5 has switched to *Vanilla JavaScript* instead of jQuery.

➤ **Advantages of Bootstrap:**

- Easy to use: Anybody with just basic knowledge of HTML and CSS can start using Bootstrap.
- Responsive features: Bootstrap's responsive CSS adjusts to phones, tablets, and desktops.
- Mobile-first approach: Bootstrap is developed mobile first, a strategy in which code is optimized for mobile devices first and then components are scaled up as necessary using CSS media queries.
- Browser compatibility: Bootstrap 5 is compatible with all modern browsers.

There are two ways to start using Bootstrap 5 on your own web site:

1. Include Bootstrap 5 from a CDN:

jsDelivr provides CDN support for Bootstrap's CSS and JavaScript.

- `<!-- Latest compiled and minified CSS -->`
`<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet">`
- `<!-- Latest compiled JavaScript -->`
`<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"></script>`

2. Download Bootstrap 5 from <https://getbootstrap.com/>

➤ **Designing a Web page using Bootstrap 5:**

1. Add the HTML5 doctype:

- Bootstrap 5 uses HTML elements and CSS properties that require the HTML5 doctype.
- Always include the HTML5 doctype at the beginning of the page, along with the lang attribute and character set.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Bootstrap 5 Example</title>
    <meta charset="utf-8">
  </head>
</html>
```

2. Responsive meta tag:

- Bootstrap 5 is mobile-first and is designed to be responsive to mobile devices. To ensure proper rendering and touch zooming, add the following `<meta>` tag inside the `<head>` element.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

- The `width=device-width` part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).
- The `initial-scale=1` part sets the initial zoom level when the page is first loaded by the browser.

3. Include Bootstrap

4. Containers:

- There are two container classes to choose from:
 1. The **.container** class provides a responsive fixed width container
 2. The **.container-fluid** class provides a full width container, spanning the entire width of the viewport.

➤ **Bootstrap Containers:**

- Containers are the most basic layout element in Bootstrap.
- Bootstrap 5 also requires a containing element to wrap site contents.
- There are two main containers:
 1. Fixed Container.
 2. Fluid Container.

1. **Fixed Container (Default Container):**

- **.container** class is used to create a responsive, fixed-width container.



- Its max-width changes on different screen sizes

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra Large ≥1200px	XXLarge ≥1400px
max-width	100%	540px	720px	960px	1140px	1320px

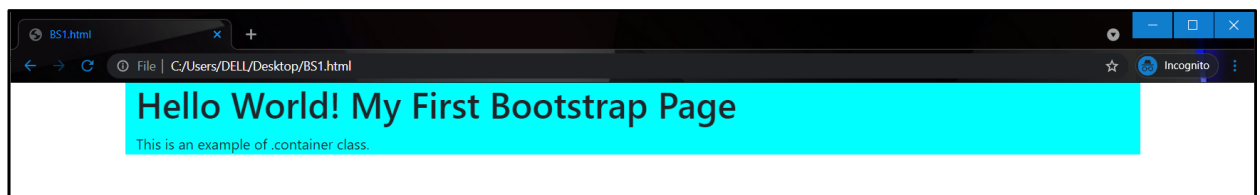
- Syntax:

```
<div class="container">
```

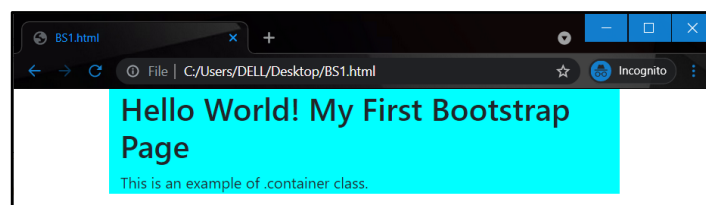
- ...
- ```
</div>
```
- Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.m
in.css" rel="stylesheet">
 <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bun
dle.min.js"></script>
</head>
<body>
<div class="container" style="background-color:cyan">
 <h1>Hello World! My First Bootstrap Page</h1>
 <p>This is an example of .container class.</p>
</div>
</body>
</html>
```

Output:

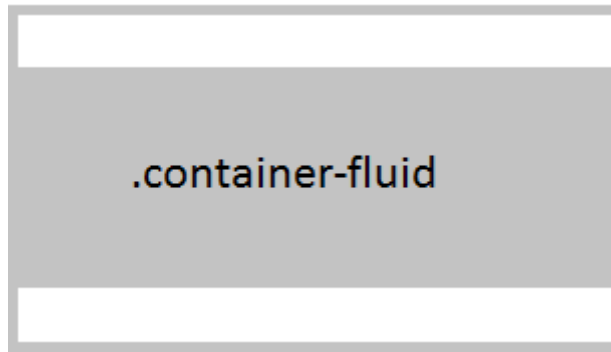


*When the browser is resized:*



## 2. Fluid Container:

- Use the **.container-fluid** class to create a full width container that will always span the entire width of the screen.
- The width of a fluid container is always 100%.



- Syntax:

```
<div class="container-fluid">
```

```
...
```

```
</div>
```

- Example:

```
<!DOCTYPE html>
```

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

```
<head>
```

```
<meta charset="utf-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

```
<link
```

```
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.m
in.css" rel="stylesheet">
```

```
<script
```

```
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bun
dle.min.js"></script>
```

```
</head>
```

```
<body>
```

```
<div class="container-fluid" style="background-color:orange ">
```

```
<h1>Hello World! My First Bootstrap Page</h1>
```

```
<p>This is an example of .container-fluid class.</p>
```

```
</div>
```

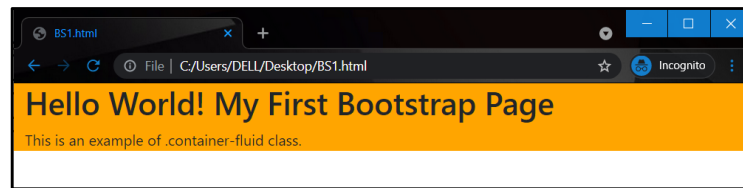
```
</body>
```

```
</html>
```

Output:



*When the browser is resized:*



### **Bootstrap 5 Text/Typography:**

- Bootstrap 5 styles HTML headings (<h1> to <h6>) with a bolder font-weight and a responsive font-size.
- Can use .h1 to .h6 classes on other elements to make them behave as headings.
- **Example:**

```
<!DOCTYPE html>
```

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

```
<head>
```

```
 <title>Bootstrap Example</title>
```

```
 <meta charset="utf-8">
```

```
 <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
 <link
```

```
 href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.css" rel="stylesheet">
```

```
 <script
```

```
 src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/js/bootstrap.bundle.min.js"></script>
```

```
</head>
```

```
<body>
```

```
<div class="container">
```

```
 <p>The font-size of each Bootstrap heading depends on the screen size. Try to
 resize the browser window to see the effect.</p>
```

```
 <h1>h1 Bootstrap heading</h1>
```

```
 <h2>h2 Bootstrap heading</h2>
```

```
 <h3>h3 Bootstrap heading</h3>
```

```
 <h4>h4 Bootstrap heading</h4>
```

```
 <h5>h5 Bootstrap heading</h5>
```

```
 <h6>h6 Bootstrap heading</h6>
```

```
 <p class="h1">Paragraph as h1 Bootstrap heading</p>
```

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

**Output:**

The font-size of each Bootstrap heading depends on the screen size. Try to resize the browser window to see the effect.

**h1 Bootstrap heading**

**h2 Bootstrap heading**

**h3 Bootstrap heading**

**h4 Bootstrap heading**

**h5 Bootstrap heading**

**h6 Bootstrap heading**

**Paragraph as h1 Bootstrap heading**

---