COLLAGE OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY OF JEDDAH

كلية علوم و هندسة الحاسب

جامعة جدة

CSS: Cascading Style Sheets 2

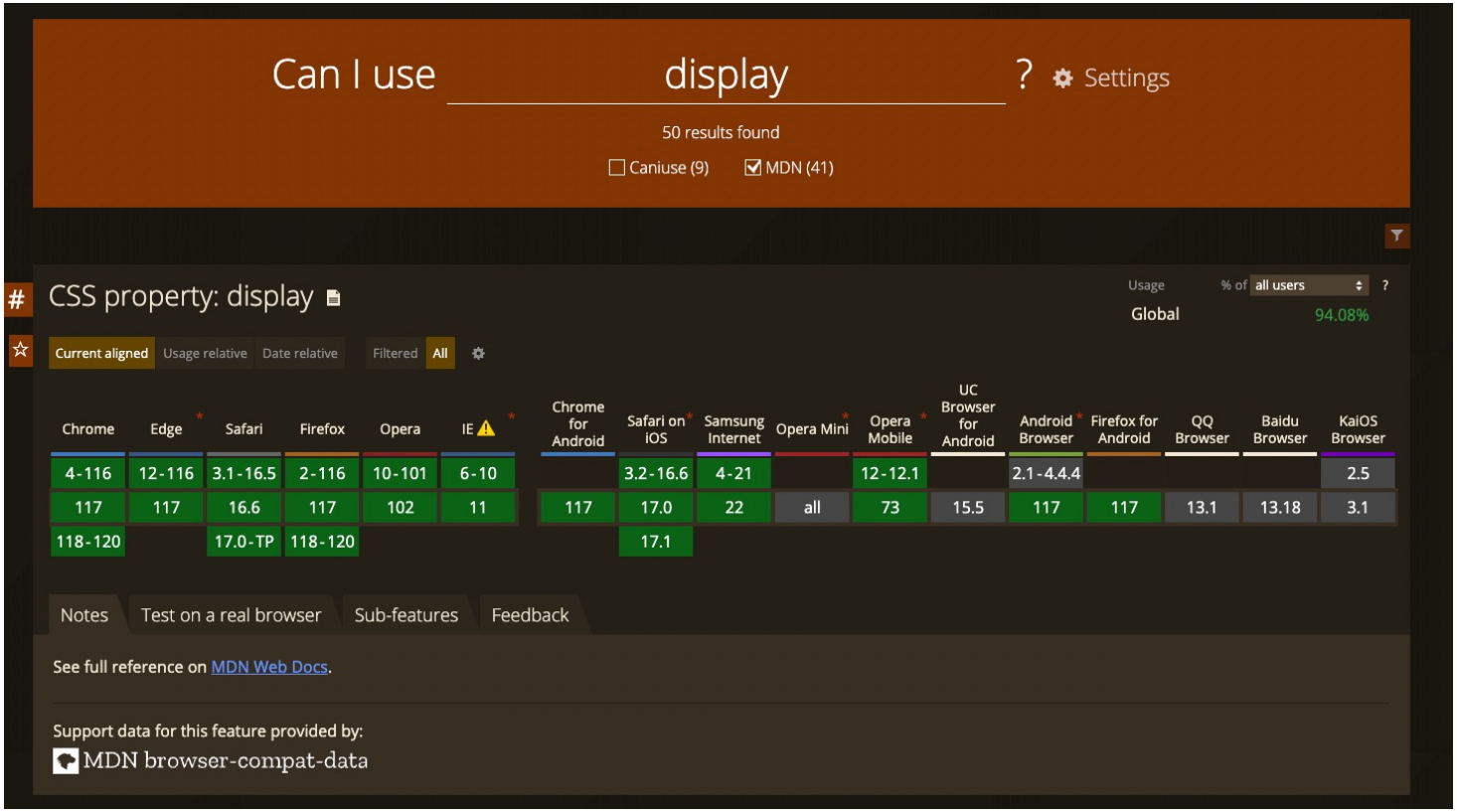
CCSW 321 (Web Development)

What will be covered

- CSS Layout.
- The Box Model.
- Positioning.
- Responsive Web Design.
- Media Queries.
- Advanced CSS Layouts: Grid and Flexbox.

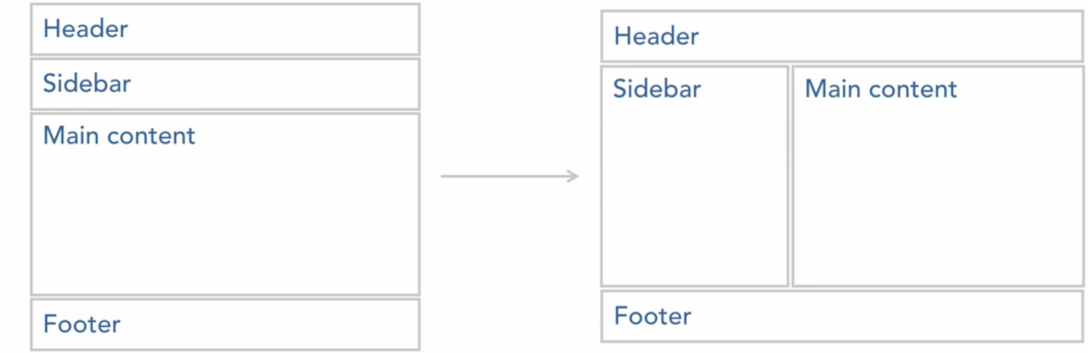
CSS Support

- How can I check if a specific property is supported?
- You can browse to the website https://caniuse.com/ and type down the name of the CSS property to see the list of browsers that supports it.



CSS Layouts

- Normal flow: Elements appear as stacked on top of each other.
- Elements are taken out of their normal flow when CSS is used to create custom CSS layout where columns and rows are used and the placement of elements in specific areas of the page is needed.
- One way to accomplish custom CSS layouts is through the understanding the box model, the display property, and positioning of elements.



CSS Layouts

- More advanced techniques to do custom CSS layouts exist such as **the Flexbox**, and **Grid** layouts. However, keep in mind that they have not yet reached the W3C "recommendation" status (https://www.w3.org/ Style/CSS/). We will cover them at the end of this chapter.

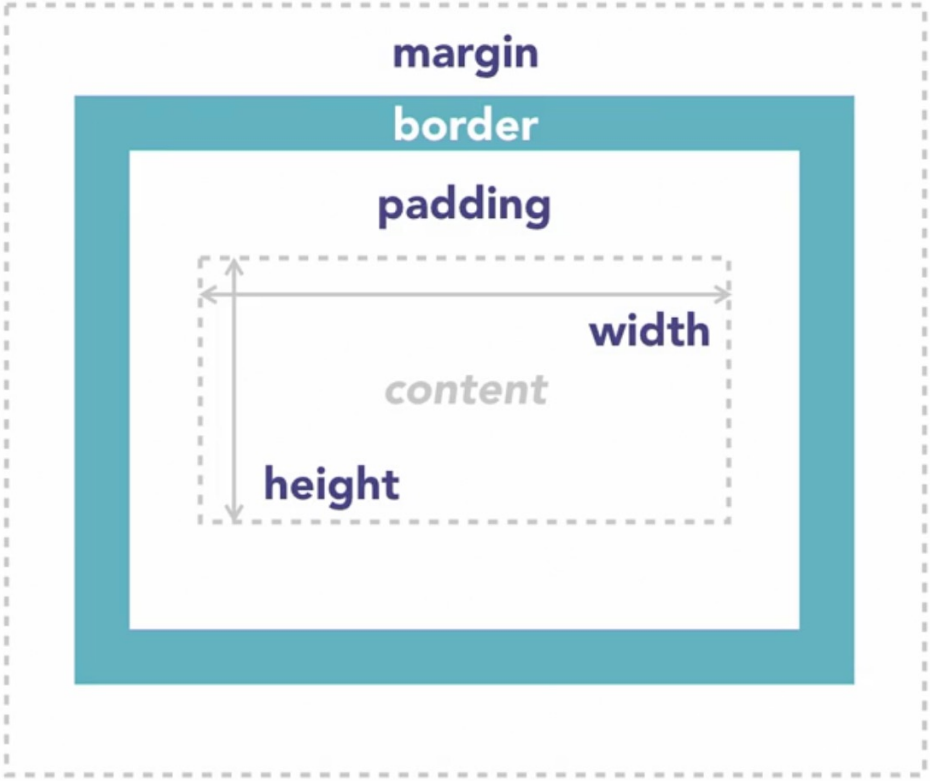
- Alternatively, advance layouts can also be achieved through CSS frameworks such as Bootstrap.

Stable drafts	Sta-tus	Upcom-ing	
CSS Snapshot 2023	NOTE	NOTE	
CSS Backgrounds and Borders Level 3	CRD	CR	
CSS Conditional Rules Level 3	CR	CR	
CSS Multi-column Layout Level 1	CR	PR	
CSS Values and Units Level 3	CR	PR	
CSS Flexible Box Layout Level 1	CR	PR	
CSS Counter Styles Level 3	CR	PR	

Abbreviation	Full name
FPWD	First Public Working Draft
WD	Working Draft
CR	Candidate Recommendation
PR	Proposed Recommendation
REC	Recommendation
SPSD	Superseded Recommendation

The Box Model

- The CSS box model is a way of representing the layout of an HTML element as a collection of boxes with content, padding, border, and margin areas.



The Box Model

- The **content area** is where the actual content of the element is displayed.
- The **padding** property determines the distance between the content inside an element and the inside of the element's border.
- The **padding** can be specified in the same way, using padding-top, padding-right, padding-left and padding-bottom.
- The **border area** is the edge of the box. Borders can be styled using the border property.
- The **margin** property sets the space between the outside of an element's border and all other content on the page.
- The **margins** for individual sides of an element can be specified by using the properties margin-top, margin-right, margin-left and margin-bottom.

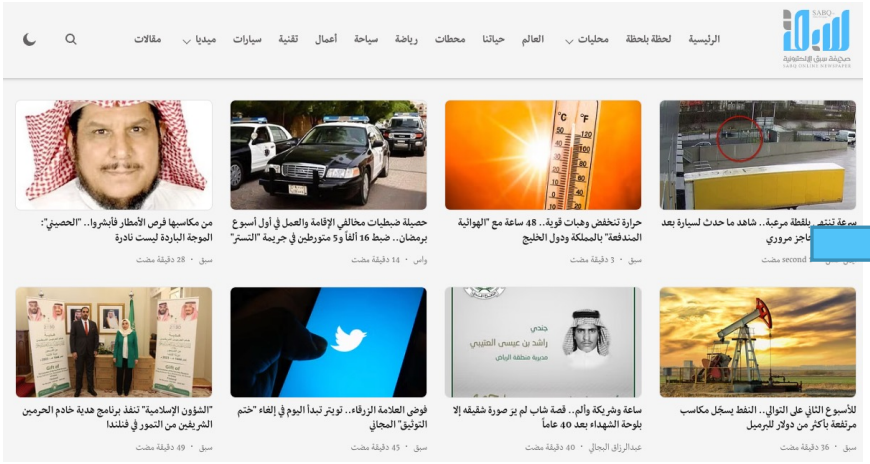
The Display Property

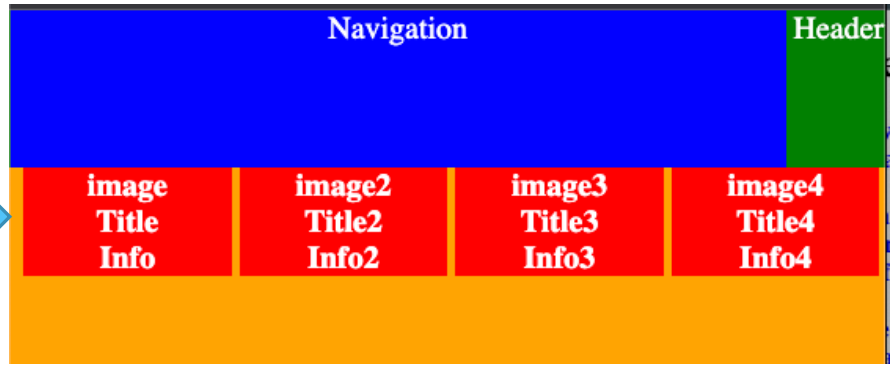
- The display property in CSS controls how an element is displayed on a web page. It has many possible values including block, inline, inline-block, and none.

property	Explanation	Width/Height
display: block;	Takes up the full width of its container and forces subsequent elements to appear on a new line	Can be set.
display: inline;	only takes up as much width as necessary to contain its content	Cannot be set
display: inline-block;	Similar to an inline element, but can have width and height properties applied to it	Can be set.
display: none;	Hides element from page	NA

The Display Property

- The display property provides us with the first step towards custom CSS layout, the creation of boxes.
- All website layouts can be thought of as a collection of boxes. Once the outline is defined, we can arrange content inside each box as we desire using positioning.





This structure can easily be created by setting the display property to “inline-block” and setting a width for each box.

Positioning

- The position property can be used to change the flow of the document by positioning elements relative to its current position, its containing element, or the browser viewport.
- There are four main values for the position property in CSS: static, relative, absolute, and fixed.
- **Static** is the default value for the position property, and it means that the element is positioned according to the normal flow of the document (i.e., the order in which they appear in the HTML).

Positioning

- **Relative** allows you to position an element relative to its normal position in the document flow. You can use the top, right, bottom, and left properties to adjust the element's position.
- **Fixed** allows you to position an element relative to the viewport (i.e. the browser window). The element will remain in the same position even if the user scrolls the page. You can use the top, right, bottom, and left properties to adjust the element's position.

Positioning

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 4.10: positioning2.html -->
4  <!-- Relative positioning of elements. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Relative Positioning</title>
9        <style type = "text/css">
10          p           { font-size: 1.3em;
11                        font-family: verdana, arial, sans-serif; }
12          span        { color: red;
13                        font-size: .6em;
14                        height: 1em; }
15          .super      { position: relative;
16                        top: -1ex; }
17          .sub        { position: relative;
18                        bottom: -1ex; }
19          .shiftleft  { position: relative;
20                        left: -1ex; }
21          .shiftright { position: relative;
22                        right: -1ex; }
23       </style>
24    </head>
```

Fig. 4.10 | Relative positioning of elements. (Part 1 of 3.)

Positioning

```
25      <body>
26         <p>The text at the end of this sentence
27            <span class = "super">is in superscript</span>.</p>
28
29         <p>The text at the end of this sentence
30            <span class = "sub">is in subscript</span>.</p>
31
32         <p>The text at the end of this sentence
33            <span class = "shiftleft">is shifted left</span>.</p>
34
35         <p>The text at the end of this sentence
36            <span class = "shiftright">is shifted right</span>.</p>
37      </body>
38   </html>
```

Fig. 4.10 | Relative positioning of elements. (Part 2 of 3.)

Positioning

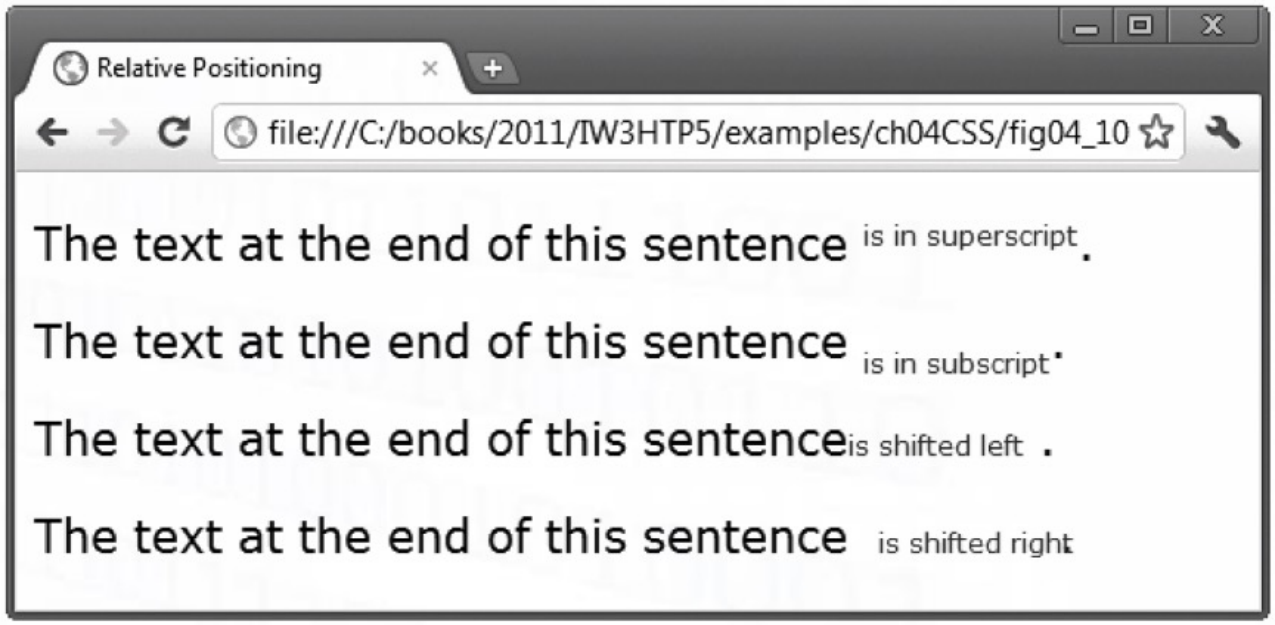


Fig. 4.10 | Relative positioning of elements. (Part 3 of 3.)

Positioning

- **Absolute** allows you to position an element relative to its nearest positioned ancestor (i.e. an element with a position value other than static). If there is no positioned ancestor, the element is positioned relative to the initial containing block (usually the **body element**).
- You can use the top, right, bottom, and left properties to adjust the element's position

Positioning

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 4.9: positioning.html -->
4  <!-- Absolute positioning of elements. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Absolute Positioning</title>
9        <style type = "text/css">
10          .background_image { position: absolute;
11                              top: 0px;
12                              left: 0px;
13                              z-index: 1; }
14          .foreground_image { position: absolute;
15                              top: 25px;
16                              left: 100px;
17                              z-index: 2; }
18          .text             { position: absolute;
19                              top: 25px;
20                              left: 100px;
21                              z-index: 3;
22                              font-size: 20pt;
23                              font-family: tahoma, geneva, sans-serif; }
24       </style>
25    </head>
```

Fig. 4.9 | Absolute positioning of elements. (Part 1 of 3.)

Positioning

```
26      <body>
27         <p><img src = "background_image.png" class = "background_image"
28            alt = "First positioned image" /></p>
29
30         <p><img src = "foreground_image.png" class = "foreground_image"
31            alt = "Second positioned image" /></p>
32
33         <p class = "text">Positioned Text</p>
34      </body>
35   </html>
```

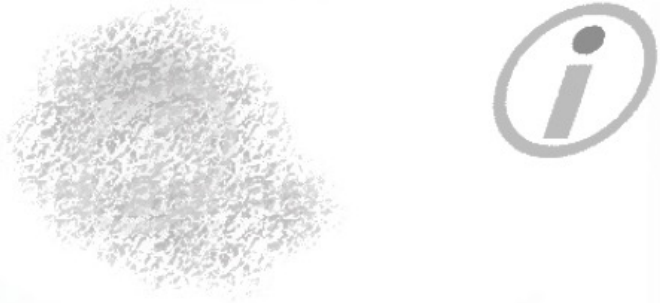


Fig. 4.9 | Absolute positioning of elements. (Part 2 of 3.)

Positioning

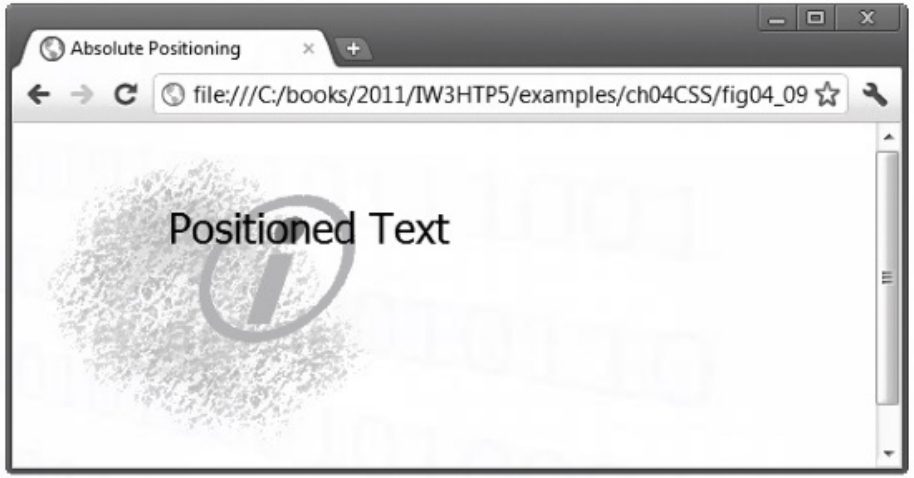


Fig. 4.9 | Absolute positioning of elements. (Part 3 of 3.)

Positioning and z-index

- **The z-index** controls the stacking order of elements that overlap. Elements with a higher z-index value appear on top of elements with a lower value.
- The static positioning ignores the z-index property.
- For elements with non-static positioning (e.g., relative), the element with the higher z-index will be placed higher in the hierarchy (i.e., will be placed on top of others).
- If z-index is set to the same value for all elements, then the positioning type will determine which element shows on the top.
- In general, the items will appear from top to bottom depending on the following order: The ” fixed”, then “absolute”, and finally “relative”, Keep in mind “static” positioning will always be at the bottom.

Responsive Web Design

- The goal of responsive design is to create a consistent user experience across all devices, ensuring that users can access and use your website or web application no matter what device they are using.
- Responsive design is becoming increasingly important as more and more users access the internet on mobile devices, and is now considered a best practice in web design

- Responsive design requires careful planning and design, as well as testing across a range of devices and screen sizes..

Responsive Web Design

- Responsive design typically involves designing for the smallest screen size first, and then scaling up to larger screen sizes.
- Responsive design can be achieved using a variety of tools and techniques, including the use of media queries to build custom CSS for the various sizes, as well as the use of CSS frameworks like Bootstrap and Foundation.

Portability Tip 4.1

To ensure that your style sheets work in various web browsers, test them on many client web browsers, and use the W3C CSS Validator.

الأفضل نستخدم relative length measurement

Good Programming Practice 4.1

Whenever possible, use relative-length measurements. If you use absolute-length measurements, your document may not scale well on some client browsers (e.g., smartphones).

Responsive Web Design

- Media queries are typically used for creating responsive designs that adapt to different screen sizes and devices.
- Media queries are a CSS technique that allows you to apply different styles to an element based on certain conditions, such as the size of the screen or the device being used to view the page.
- The @media rule specifies one or more conditions that must be met in order for the styles within the rule to be applied.

```
/* Styles for all screens */
h1 {
  font-size: 24px;
}

/* Styles for screens with a minimum width of 768px */
@media screen and (min-width: 768px) {
  h1 {
    font-size: 36px;
  }
}
```

Media type

Condition

Responsive Web Design

- To create a block of styles that apply to a single media type such as screen, use @media print and enclose the style rules in curly braces.

```
/* Styles for all screens */
h1 {
  font-size: 24px;
}

/* Styles for screens with a minimum
width of 768px */
@media screen and (min-width: 768px) {
  h1 {
    font-size: 36px;
  }
}
```

Media type

Condition

Responsive Web Design

- Examples of media types in CSS include:
  - **all**: This is the default media type and applies to all devices and media types.
  - **print**: Applies to printers or print preview mode in web browsers.
  - **screen**: Applies to computer screens, tablets, smartphones, and other devices with a screen.
  - **handheld**: Designed for mobile Internet devices.
  - **speech**: Allow the programmer to give a speech-synthesizing web browser more information about the content of the web page
- Most common media type for a web page is the screen media type, which is a standard computer screen.

Responsive Web Design

- Common media query conditions include:
  - **width**: This condition allows you to target screens with a specific width. For example, you could use **min-width** to target screens with a minimum width of a certain size, or **max-width** to target screens with a maximum width of a certain size.
  - **height**: This condition allows you to target screens with a specific height. Similar to width, you can use **min-height** and **max-height** to create specific conditions.
  - **aspect-ratio**: This condition allows you to target screens with a specific aspect ratio (the ratio of width to height). For example, you could use aspect-ratio: 16/9 to target screens with a 16:9 aspect ratio, which is common for widescreen displays.
  - **orientation**: This condition allows you to target screens in a specific orientation, either portrait or landscape.

Responsive Web Design

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 4.16: mediatypes.html -->
4  <!-- CSS media types. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Media Types</title>
9        <style type = "text/css">
10          @media all
11          {
12             body  { background-color: steelblue; }
13             h1    { font-family: verdana, helvetica, sans-serif;
14                     color: palegreen; }
15             p     { font-size: 12pt;
16                     color: white;
17                     font-family: arial, sans-serif; }
18          } /* End @media all declaration. */
```

Fig. 4.16 | CSS media types. (Part 1 of 4.)

Responsive Web Design

```
19          @media print
20          {
21             body  { background-color: white; }
22             h1    { color: seagreen; }
23             p     { font-size: 14pt;
24                     color: steelblue;
25                     font-family: "times new roman", times, serif; }
26          } /* End @media print declaration. */
27       </style>
28    </head>
29    <body>
30       <h1>CSS Media Types Example</h1>
31
32       <p>
33          This example uses CSS media types to vary how the page
34          appears in print and how it appears on any other media.
35          This text will appear in one font on the screen and a
36          different font on paper or in a print preview. To see
37          the difference in Internet Explorer, go to the Print
38          menu and select Print Preview. In Firefox, select Print
39          Preview from the File menu.
40       </p>
41    </body>
42 </html>
```

Fig. 4.16 | CSS media types. (Part 2 of 4.)

Responsive Web Design

a) Background color appears on the screen.

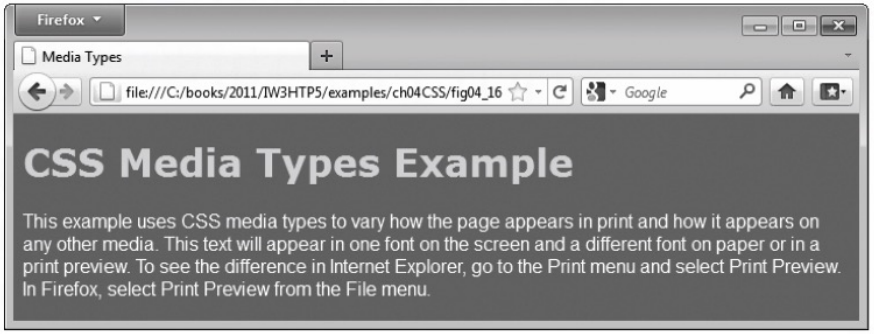


Fig. 4.16 | CSS media types. (Part 3 of 4.)

b) Background color is set to white for the `print` media type.

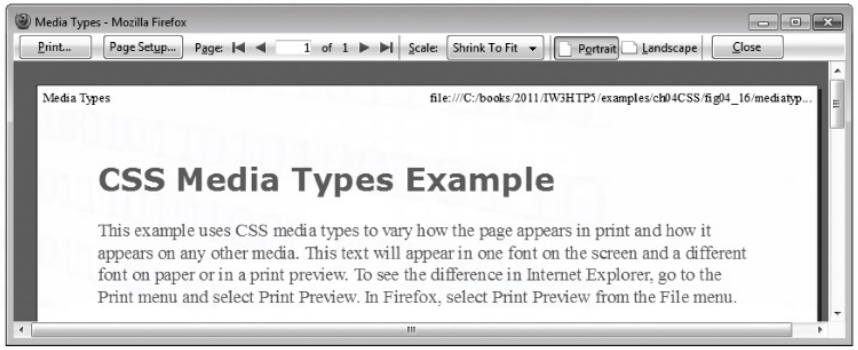
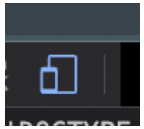
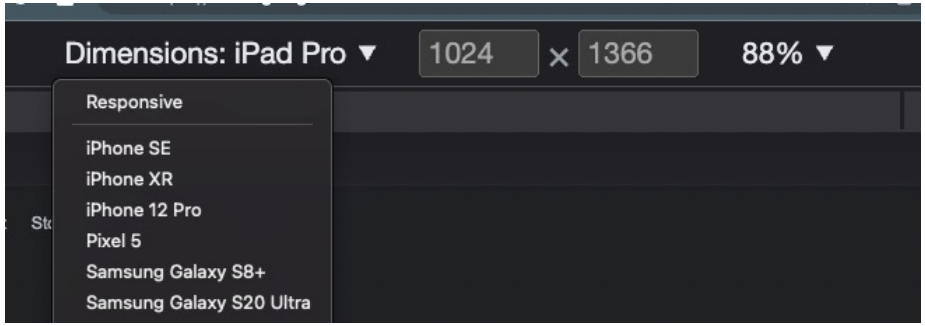


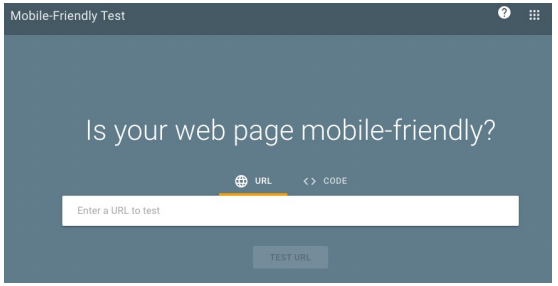
Fig. 4.16 | CSS media types. (Part 4 of 4.)

Responsive Web Design

- Testing your website on different sizes can be done using Google's chrome developer tools, simply open a page and right-click to select "inspect".
- You will find the icon 'device toolbar' icon. 
- Now you can set the device you want to test, or specify the exact screen size.



- Testing whether your website is mobile friendly with Google:
  - https://search.google.com/test/mobile-friendly



CSS Example

Can you create this simple page?

Step0. Find common design elements that you can create with classes (e.g., text size).

Step1. Create the title (blue box). Note the text color, border, text-alignment.

Step2. Create the passenger information (green box). Note text and image are side by side and text is centered vertically with the image.

Step3. Create the extra information box (with 3x columns). Notice multi-column structure, text alignment, box alignment, border type.

CSS Example

Can you create this simple page?

Step4. Use media query to **1)** fix any display issues for smaller screens and **2)** hide the 'extra information' box when page is printed.

Issue on smaller screens

Show only necessary info for page print

1. Logo.
2. Passenger Info.
3. Barcode.

CSS Example: HTML

Step1

Step2

Step3

```
<body>
  <header>
    <p class='lrgtxt'>Saudi Airlines</p>
    <p class='smltxt'>SkyTeam Member. Happy to serve.</p>
  </header>
  <section id='info'>
    <div>
      <p class='smltxt'>Name: Moayad Alshangiti</p>
      <p class='smltxt'>Fursan: 10554646</p>
    </div>
    <img src="media/barcode.webp" alt="picture of a barcode">
  </section>
  <section id="extra">
    <article>
      <p class='lrgtxt'>Luggage</p>
      <p class='smltxt'>Lorem ipsum dolor sit amet, erant elitr detracto sed
ei, eum elitr similique ne, dicit adipisci sit ne. Malorum reprehendunt ius
ea, ei vel falli dicit aperiri. Sale definiebas instructior eam ut, at per
rationibus voluptatum, movet petentium gubergren vim at.  </p>
    </article>
     <article>
      <p class='lrgtxt'>Forbidden</p>
      <p class='smltxt'>Lorem ipsum dolor sit amet, erant elitr detracto sed
ei, eum elitr similique ne, dicit adipisci sit ne. Malorum reprehendunt ius
ea, ei vel falli dicit aperiri. Sale definiebas instructior eam ut, at per
rationibus voluptatum, movet petentium gubergren vim at.   </p>
    </article>
     <article>
      <p class='lrgtxt'>On Time!</p>
      <p class='smltxt'>Lorem ipsum dolor sit amet, erant elitr detracto sed
ei, eum elitr similique ne, dicit adipisci sit ne. Malorum reprehendunt ius
ea, ei vel falli dicit aperiri. Sale definiebas instructior eam ut, at per
rationibus voluptatum, movet petentium gubergren vim at.   </p>
    </article>
  </section>
  <!-- Scripts here -->
  <script src="js/scripts.js"></script>
</body>
```

CSS Example: CSS

Step0

```
/*    shared classes*/
    .lrgtxt{
      font-size: 4em;
    }

    .smltxt{
      font-size: 2em;
    }
```

```
/*    custom classes*/
```

Step1

```
    header{
      background-color: blue;
      text-align: center;
      color:white;
      margin: 2%;
      border: 2px solid black;
    }
```

Step2

```
    #info{
      margin: 2%;
      background-color: green;
       border: 2px solid black;
       text-align: center;
       color: white;
    }
```

Step2

```
#info div, #info img{
  display: inline-block;
  width: 45%;
  vertical-align: middle;
}

#info img{
  padding-top: 2%;
}
```

Step3

```
#extra{
  margin: 2%;
  text-align: center;
}

#extra article{
  display: inline-block;
  width: 30%;
  border: 1px dashed black;
}
```

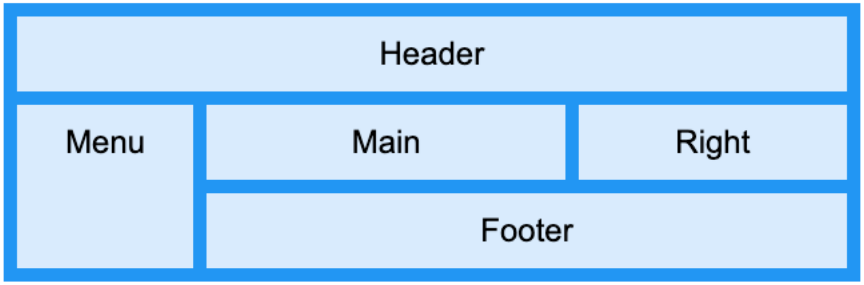
Step4

```
@media (max-width:900px){
  .lrgtxt{
  font-size: 2em;
  }
  .smltxt{
    font-size: 1em;
  }
}

@media print{
  #extra{
    display: none;
  }
}
```

Advanced CSS Layouts: Grid and Flexbox

What is CSS Grid?

- CSS Grid is a two-dimensional layout system in CSS.
- It allows us to create grid-based layouts with rows and columns, providing **precise control** over the placement and alignment of elements.
- It's perfect for creating complex layouts with rows and columns.
- CSS Grid is inherently responsive, making it ideal for adapting layouts to different screen sizes.



Advanced CSS Layouts: Grid and Flexbox

What is CSS Grid?

- Key Features of CSS Grid:
  - **Grid Container**: The parent element that holds the grid.
  - **Grid Items**: The child elements placed within the grid.
  - **Rows and Columns**: The structure of the grid is defined by rows and columns.

- **Grid properties**
  - **grid-template-columns:** Define the number of columns in the grid.
  - **grid-template-rows**: Define the number of rows in the grid.
  - **grid-gap**: Sets the spacing between the grid items.
  - **grid-row**: Specifies where to start and end the grid item (used when **merging rows**).
  - **grid-column**: Specifies where to start and (used when **merging columns**).

Advanced CSS Layouts: Grid and Flexbox

Use fr unit

What is the ‘fr’ unit?

- The "fr" unit in CSS stands for "**fractional unit**" or "flexible unit." It's a relatively **new unit** introduced with the CSS Grid Layout and CSS Flexible Box Layout (Flexbox) specifications to create flexible and responsive layouts.
- The "fr" unit divides the available space into fractions. For example, if you have two columns with 1fr each, they will share the available space **equally**.
- You can also use proportions with "fr." For example, **2fr 1fr** would allocate **twice as much** space to the **first** column as compared to the **second** column.
- "fr" units can be combined with other units like percentages or fixed values to create complex and responsive grid layouts.

Advanced CSS Layouts: Grid and Flexbox

- Example

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Grid Example</title>
    <style>
    /* Create a grid container */
    .container {
        display: grid;
        grid-template-columns: 1fr 1fr 1fr; /* Three columns with equal width */
        grid-gap: 10px; /* Spacing between grid items */
    }

    /* Style grid items */
    .item {
        background-color: #3498db;
        color: #ffffff;
        padding: 20px;
        text-align: center;
    }

    </style>
</head>
<body>
    <div class="container">
        <div class="item">1</div>
        <div class="item">2</div>
        <div class="item">3</div>
        <div class="item">4</div>
        <div class="item">5</div>
        <div class="item">6</div>
    </div>
</body>
</html>
```

1. Enabling Grid.

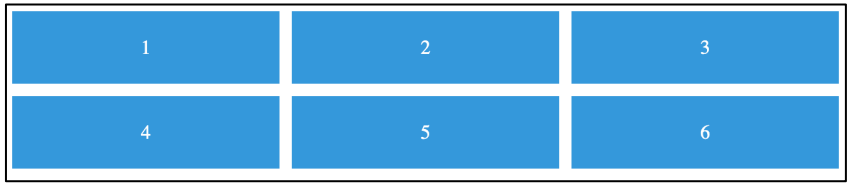
2. Creating three columns with equal spacing.

3. Create a space between grid items of 10px.

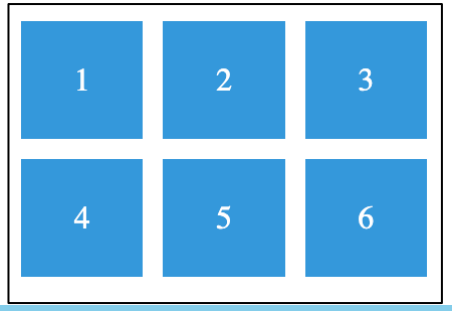
Grid Container

Grid Items

Result in large screen



Result in small screen

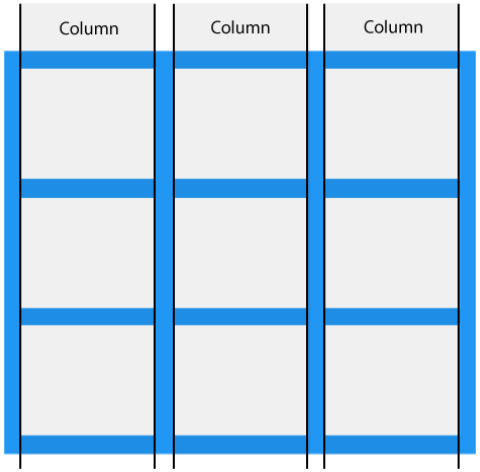


Advanced CSS Layouts: Grid and Flexbox

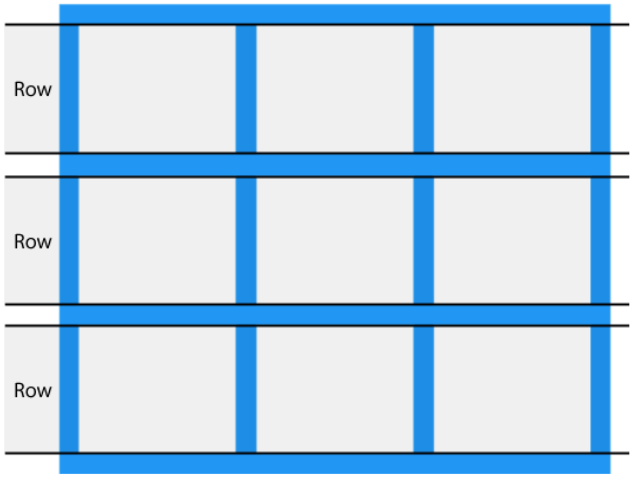
What is a Grid?

- It is a set of **grid items** that are directly children of a **grid container**.
- It consists of columns and rows.

Columns

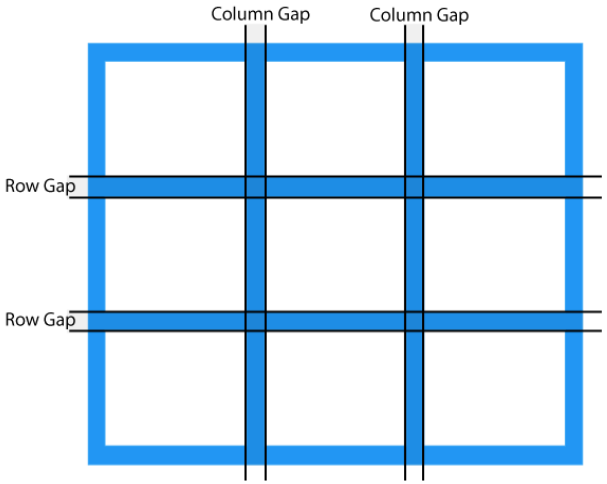


Rows



grid-template-columns and **grid-template-row** can set # of columns and rows

Gap

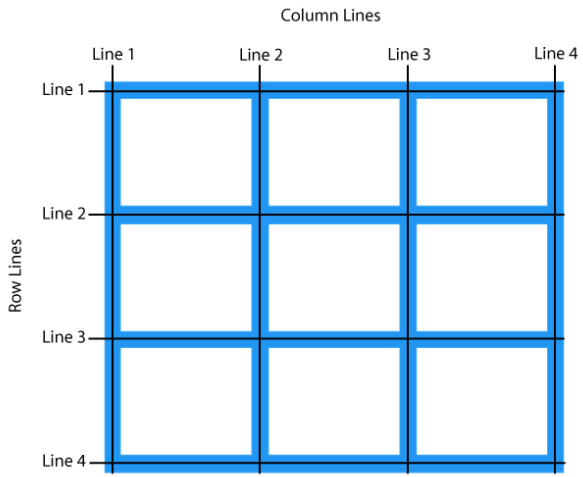


This can be controlled by property gap

Advanced CSS Layouts: Grid and Flexbox

Complex Layout?

- **grid-row**: is a property used to merge rows.
- **grid-column**: is a property used to merge cols.
- You simply need set the grid item CSS to specify the **start line** and **end line** in format "start / end"



Grid Lines

1	2	3
4	5	6
7	8	

```
.item1 {grid-column: 1/3;}
```

Grid Lines

1		2
3	4	5
6	7	8

Advanced CSS Layouts: Grid and Flexbox

What is Flexbox?

- CSS Flexible Box Layout, or Flexbox, is a **one-dimensional** layout model.
- It's excellent for **aligning items** within a container in a **row or column** layout.
- Flexbox **simplifies** responsive design by automatically adjusting item layout.
- Flexbox will adjust layout based on content and available space.

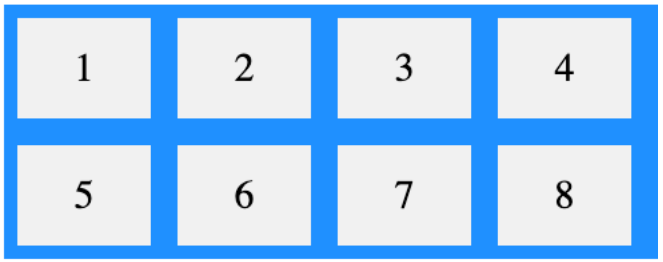
Flex item will **wrap** to the next row if there is not enough space.

Large Screen

Flexible Boxes

Smaller Screen

Flexible Boxes



Advanced CSS Layouts: Grid and Flexbox

Differences Between Grid and Flexbox

- **Grid** is two-dimensional, while **Flexbox** is one-dimensional.
- **Grid** creates both rows and columns, whereas **Flexbox** works along a single axis.
- **Grid** is suitable for complex layouts, while **Flexbox** is ideal for distributing space among items.
- **Grid** is great for overall page layout, and **Flexbox** is perfect for item alignment.

Advanced CSS Layouts: Grid and Flexbox

What is Flexbox?

- Key properties:
  - **display: flex**: Converts an element into a flex container.
  - **flex-direction**: Defines the layout direction (row or column).
    - flex-direction:row; /* will stack flex items horizontally (from left to right)*/
    - flex-direction:column;/*will stack flex items vertically (from top to bottom)*/
  - **flex-wrap**: handle wrapping when items exceed the container's size.
    - flex-wrap:no-wrap;/* items will not wrap */
    - flex-wrap:wrap; /* item will wrap, i.e., move to the next line if no needed*/

Advanced CSS Layouts: Grid and Flexbox

- Example (w/ wrap and row direction)

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  flex-wrap: wrap;
  flex-direction:row;
  background-color: DodgerBlue;
}

.flex-container > div {
  background-color: #f1f1f1;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>Flexible Boxes</h1>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>

</body>
</html>
```

1. Enabling Flex Layout.
2. Enabling "wrap".
3. Setting flow to horizontal.

Flex Container

Flex Items

Result in large screen

Flexible Boxes

1 2 3 4

Result in mid screen

Flexible Boxes

1 2 3

4

Result in small screen

Flexible Boxes

1

2

3

4

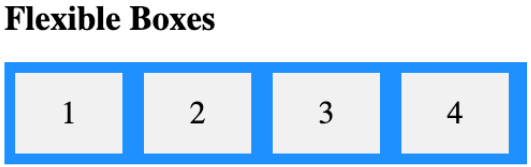
Advanced CSS Layouts: Grid and Flexbox

- Example (without wrap)

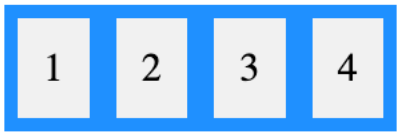
```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  flex-wrap: no-wrap;
  flex-direction:row;
  background-color: DodgerBlue;
}

.flex-container > div {
  background-color: #f1f1f1;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>Flexible Boxes</h1>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>

</body>
</html>
```

1. Enabling Flex Layout.
2. Disabling “wrap”.
3. Setting flow to horizontal.

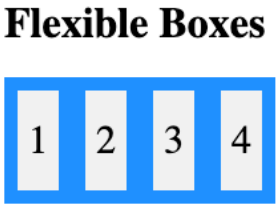
Result in large screen



Result in mid screen



Result in small screen



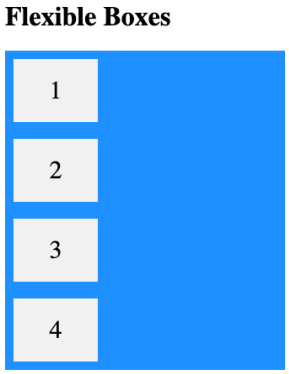
Advanced CSS Layouts: Grid and Flexbox

- Example (without wrap and direction column)

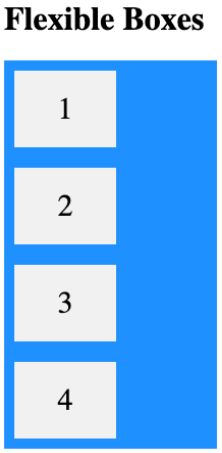
```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  flex-wrap: no-wrap;
  flex-direction: column;
  background-color: DodgerBlue;
}

.flex-container > div {
  background-color: #f1f1f1;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>Flexible Boxes</h1>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>

</body>
</html>
```

1. Enabling Flex Layout.
2. Disabling "wrap".
3. Setting flow to vertical.

Result in large screen



Result in small screen



Advanced CSS Layouts: Grid and Flexbox

What is Flexbox?

- What happens if there is **extra** space? (how to grow elements)
- What happens if there is **no** space? (how to shrink elements)
- Key properties:
  - **flex**: Combines flex-grow, flex-shrink, and flex-basis.
    - **flex-grow**: Specifies how items grow relative to each other.
    - **flex-shrink**: Specifies how items shrink relative to each other.
    - **flex-basis**: Defines the initial size of an item.
  - **justify-content**: Controls alignment along the main axis.
  - **align-items**: Controls alignment along the cross axis.

Advanced CSS Layouts: Grid and Flexbox

- Flex: growRule shrinkRule initalSize;

```
flex:0 0 100px;
```

Each box is 100px, do not grow or shrink;

Issue 1: when there is extra space, it will be empty.

Flexible Boxes


```
flex:1 0 100px;
```

Each box is 100px, grow equally to fill space

Flexible Boxes

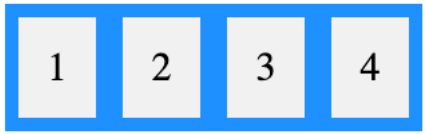
Issue 2: when there is not enough space, layout will be cut and user must scroll to see content

Flexible Boxes


```
flex:1 1 100px;
```

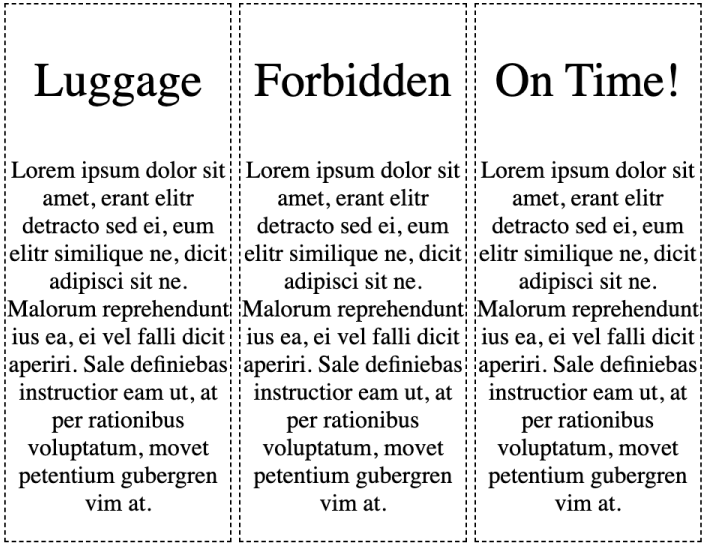
Each box is 100px, grow equally to fill space and shrink equally when space is limited

Flexible Boxes



CSS Example

How can we achieve the multi-column setup we had earlier with Flex or Grid?



Replace this snippet

```
#extra article{
  display: inline-block;
  width: 30%;
  border: 1px dashed black;
}
```

with this snippet for **Flex**

```
#extra article{
  border: 1px dashed black;
}
#extra{
  display: flex;
  gap:5px;
}
```

Or with this snippet for **Grid**

```
#extra article{
  border: 1px dashed black;
}
#extra{
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  gap:5px;
}
```

No Width Calculations: You do not have to worry about setting the width of each element.

In class activity (optional)

- Can you create the following layout using advance layout (**Flex** and/or **Grid**)?
  - **No** need to use any media queries.
  - **No** need to set any width for any element.

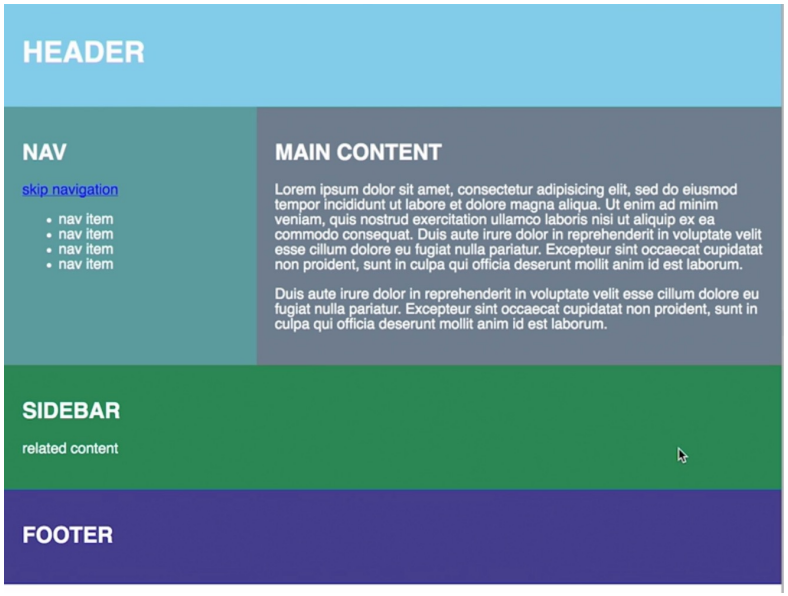
Full screen:

See next slide for more instructions

In class activity (optional)

- Can you create the following layout using Flex or Grid?

Mid size screen (responsive):



Small size screen (responsive):

Any questions?
Please feel free to raise your hands and ask.