Fundamentals of Web Development

Third Edition by Randy Connolly and Ricardo Hoar

Chapter 3

(In the book this is chapters 4 and 5)

CSS part 1: Selectors and Basic Styling

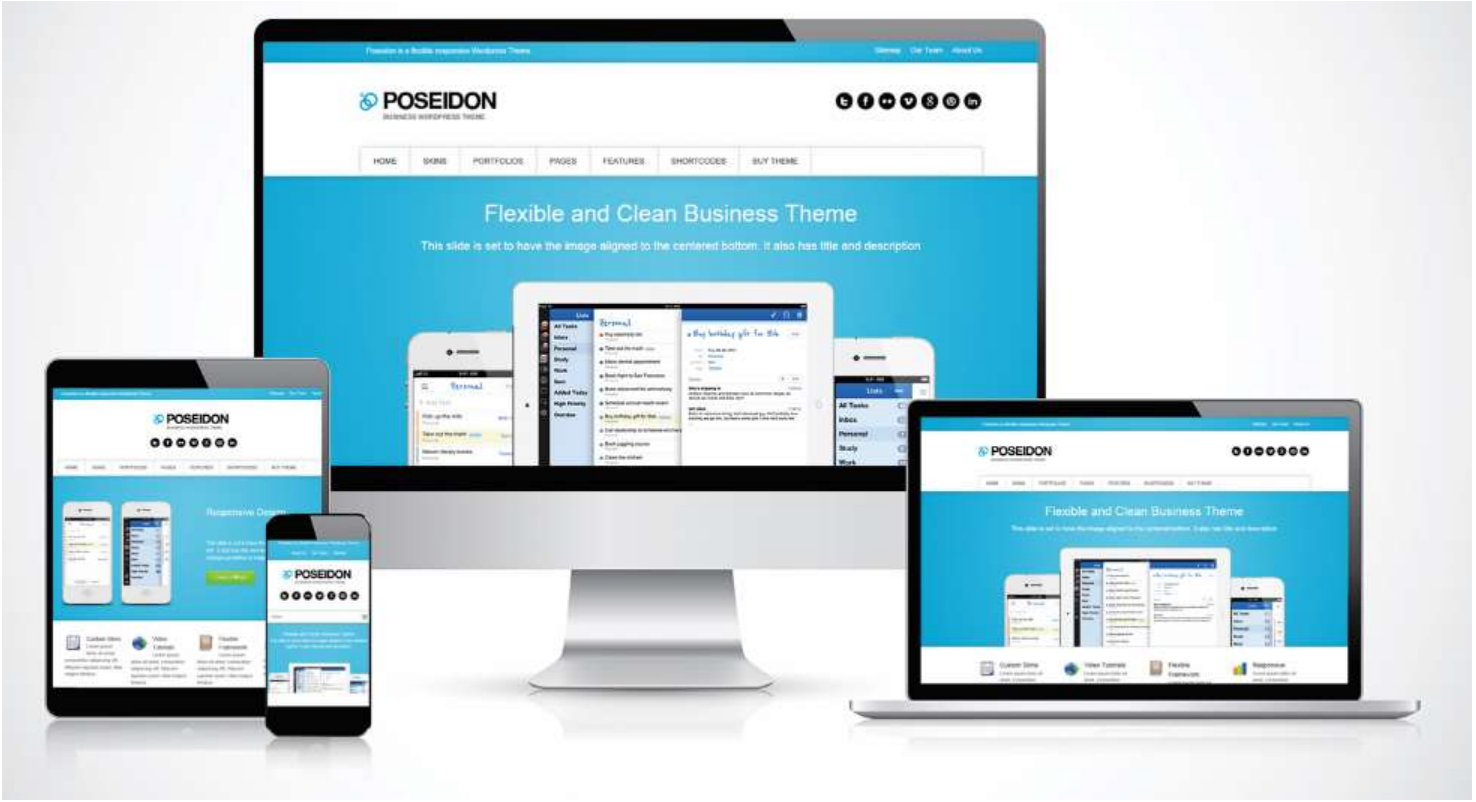
What Is CSS?

- CSS is a W3C standard for describing the **appearance** and the **layout** of an HTML document.
- With CSS, we can assign font properties, colors, sizes, borders, background images, positioning and even animate elements on the page.
- CSS can be added:
  - directly to any HTML element (via the style attribute),
  - within the <head> element,
  - or, most commonly, in a separate text file that contains only CSS.

Benefits of CSS

- **Main engineering concepts: Separation of concerns + Reuse**
- **Improved control over formatting.**
- **Improved site maintainability.** All formatting can be centralized into one CSS file
- **Improved accessibility.** By keeping presentation out of the HTML, screen readers and other accessibility tools work better.
- **Improved page-download speed.** Each individual HTML file will contain less style information and markup, and thus be smaller.
- **Improved output flexibility.** CSS can be used to adapt a page for different output media. This approach to CSS page design is often referred to as **responsive design**.

CSS allows Responsive Design

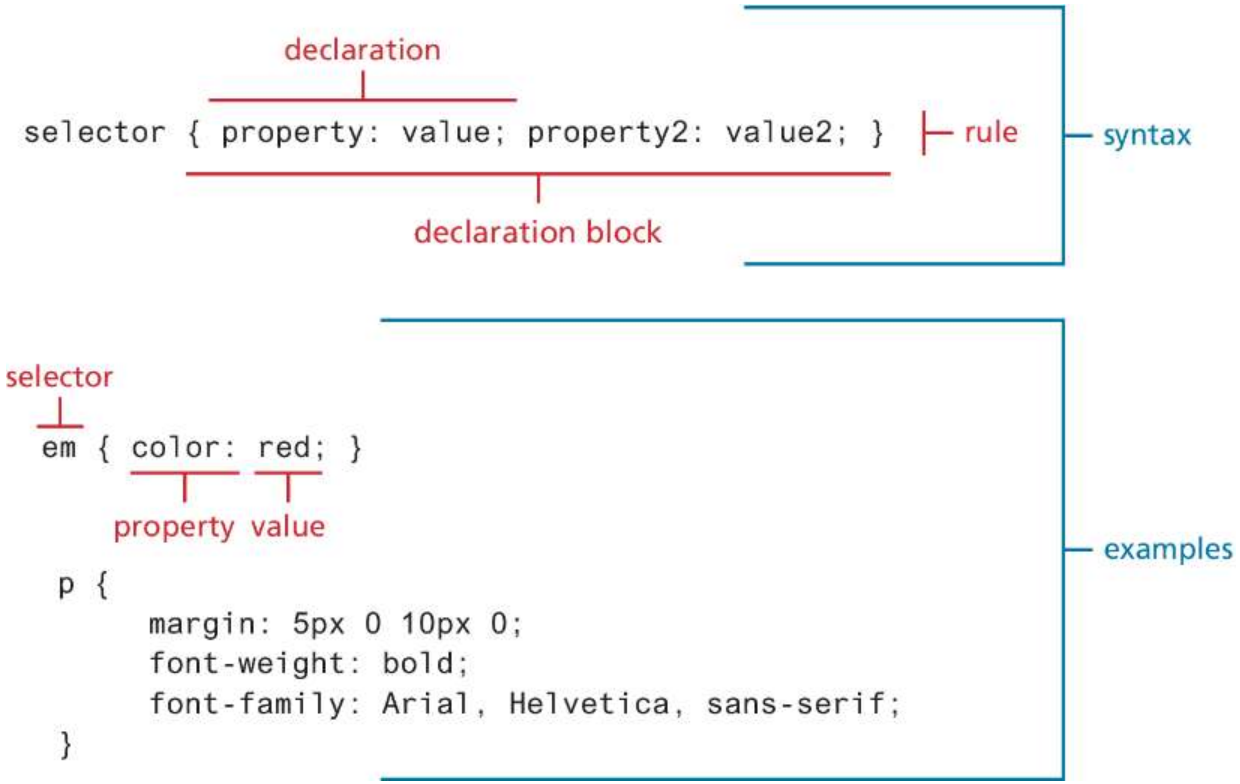


CSS Versions

- W3C decided to adopt CSS, and by the end of 1996, the CSS Recommendation was published
- A year later, the CSS2 was published
- CSS2.1, did not become an official W3C Recommendation until June 2011
- At the same time the CSS2.1 standard was being worked on, a different group at the W3C was working on a CSS3 draft
- IMPORTANT: the different browsers have not always kept up with the W3C.
- For this reason, CSS has a reputation for being a frustrating language.
- To read: https://ishadeed.com/article/cross-browser-development/

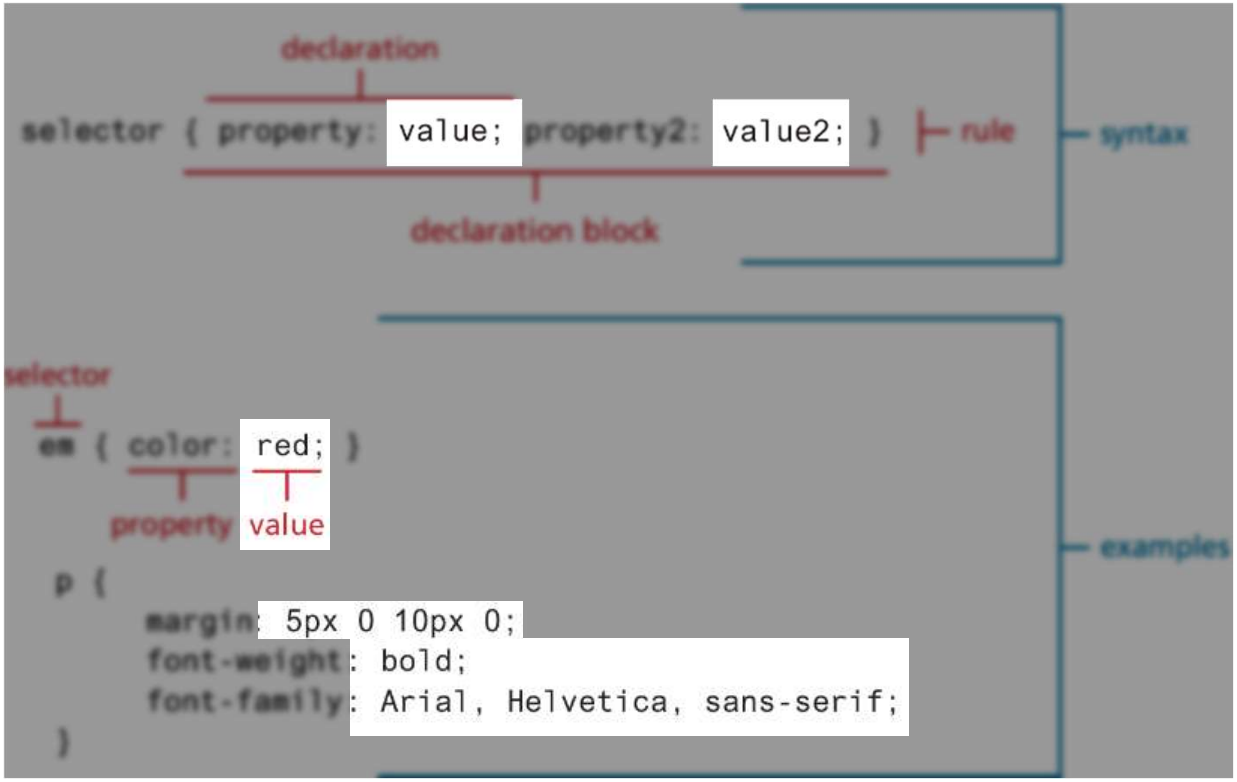
CSS Syntax

- A CSS document consists of one or more **style rules**.
- A rule consists of a **selector** that identifies the HTML element or elements that will be affected, followed by a series of **property:value pairs** called **the declaration**
- The series of declarations is also called the **declaration block**.



CSS Syntax: Values

- Each individual CSS declaration must also contain a **value**
- The unit of any given value is dependent upon the property.
- Some property values are from a predefined list of keywords. Others are values such as length measurements, percentages, numbers without units, color values, and URLs.



Color Values

Method	Description	Example
Name	Use one of 17 standard color names. CSS3 has 140 standard names.	color: red; color: green; /* CSS3 only */
RGB	Uses three different numbers between 0 and 255 to describe the red, green, and blue values of the color.	color: rgb(255,0,0); color: rgb(255,105,180);
Hexadecimal	Uses a six-digit hexadecimal number to describe the red, green, and blue value of the color	color: #FF0000; color: #FF69B4;
RGBa	This defines a partially transparent background color. The “a” stands for “alpha,” which is a term used to identify a transparency	color: rgba(255,0,0,0.5); (check: *rgbacolorpicker.com*)
HSL	Allows you to specify a color using Hue Saturation and Light values. This is available only in CSS3. HSLA is also available as well.	color: hsl(0,100%,100%); color: hsla(330,59%,100%,0.5);

Common Units of Measure Values

Units of measure in CSS are either

- **relative units**, in that they are based on the value of something else, or
- **absolute units**, in that they have a real-world size.

Some example measures:

- **px** Pixel. In CSS2 this is a relative measure, while in CSS3 it is absolute (1/96 of an inch -> 4px ~ 1.06mm)
- **em** Equal to the computed value of the font-size property of the element on which it is used (2em means 2 times the size of the current font)
- **vw** Relative to 1% of the width of the viewport (the browser window size). If the viewport is 30cm wide, 1vw = 0.3cm.
- **in** Inches (absolute)
- **cm** Centimeters (absolute)

Location of Styles

CSS style rules can be located in three different locations.

- Inline Styles
- Embedded Style Sheet
- External Style Sheet

These three are not mutually exclusive, in that you could place your style rules in all three.

Inline Styles

Inline styles are style rules placed within an HTML element via the style attribute

- An inline style only affects the element it is defined within and overrides any other style definitions for properties used

```
<h1>Share Your Travels</h1>
<h2 style="font-size: 24pt">Description</h2>
...
<h2 style="font-size: 24pt; font-weight: bold;">Reviews</h2>
```

LISTING 4.1 Inline styles example

Embedded Style Sheet

Embedded style sheets (also called internal styles) are style rules placed within the <style> element (inside the <head> element of an HTML document)

- While better than inline styles, using embedded styles is also discouraged.

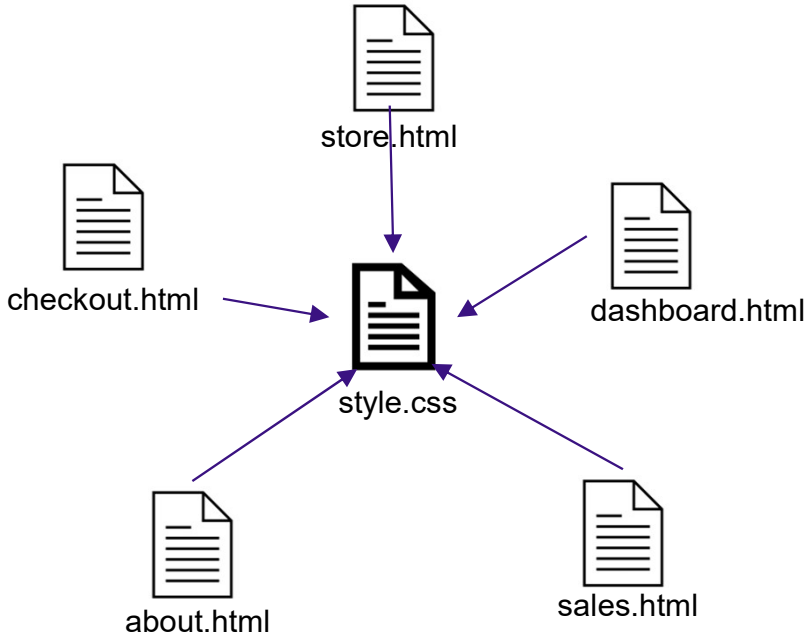
```
<head>
  <meta charset="utf-8">
  <title>Chapter 4</title>
  <style>
            h1 { font-size: 24pt; }
            h2 {
              font-size: 18pt;
              font-weight: bold;
            }
  </style>
  </head>
<body>
```

LISTING 4.2 Embedded styles example

External Style Sheet

External style sheets are style rules placed within an external text file with the **.css** extension.

- When you change an external style sheet, all HTML documents that reference that style sheet will **automatically use the updated version**.
- The browser is able to **cache** the external style sheet, which can improve performance as well.




```
<head>
   <meta charset="utf-8">
   <title>Chapter 4</title>
   <link rel="stylesheet" href="styles.css" />
</head>
```

LISTING 4.3 Referencing an external style sheet

Selectors

When defining CSS rules, you will need to use a selector.

- Selectors tell the browser which elements will be affected by the property values
- Selectors allow you to select individual or multiple HTML elements.
- Three basic selector types have been around since the earliest CSS2 specification.
  - Element Selectors
  - Class Selectors
  - Id Selectors

Element Selectors

Element selectors select all instances of a given HTML element.

You can also select all elements by using the **universal element selector** (*)

You can select a group of elements by separating the different element names with commas.

```
/* commas allow you to group selectors */
p, div, aside {
  margin: 0;
  padding: 0;
}
/* the above single grouped selector is equivalent to the following: */
p {
  margin: 0;
  padding: 0;
}
div {
  margin: 0;
  padding: 0;
}
aside {
  margin: 0;
  padding: 0;
}
```

LISTING 4.4 Sample grouped selector

Class and ID Selectors

A **class selector** allows you to target different HTML elements labeled with the same class

We use a period (.) followed by the class name.

An **ID selector** allows you to target a specific element by its id attribute, which takes the form:

hash (#) followed by the id name.

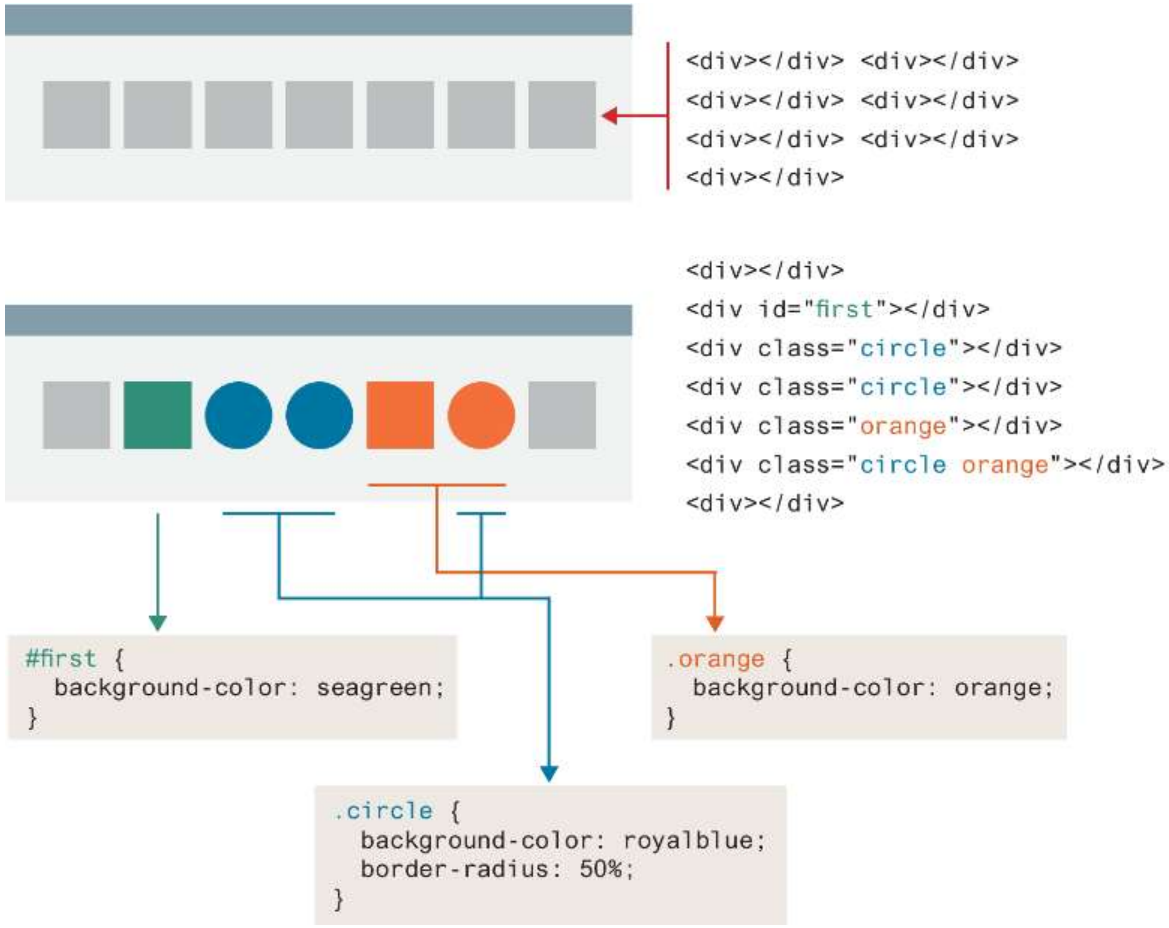
The difference between "class" and "id":

"id" is **unique** in a document and used to target only one element, whereas "class" can be used to target many elements.

Class and ID Selector Example

Consider this figure, the original HTML can be modified to add class and id values, which are then styled in CSS.

- Id selector **#first** matches the div with id “first”
- Class selectors .**orange** and .**circle**, match all divs with those class values.
- Notice that an element can be tagged with multiple classes (priority goes to the last rule in the document).



Note: in reality the divs would appear vertically

Source code: chapter03/01_selectors

Attribute Selectors

An **attribute selector** provides a way to select HTML elements either by the presence of an element attribute or by the value of an attribute.

- Attribute selectors can be helpful in the styling of hyperlinks and form elements
- These selectors can be combined with the element id and class selectors.
- You use square brackets to create attribute selectors

Attribute Selectors

	Matches	Example
[ ]	A specific attribute.	**[title]** Matches any element with a title attribute
[=]	A specific attribute with a specific value.	**a[title="posts from this country"]** Matches any <a> element whose title attribute is exactly “posts from this country”
[~=]	A specific attribute whose value matches at least one of the words in a space-delimited list of words.	**[title~="Countries City"]** Matches any title attribute that contains the word “Countries“
[^=]	A specific attribute whose value begins with a specified value.	**a[href^="mailto"]** Matches any <a> element whose href attribute begins with “mailto“
[*=]	A specific attribute whose value contains a substring.	**img[src*="flag"]** Matches any <img> element whose src attribute contains somewhere within it the text “flag“
[$=]	A specific attribute whose value ends with a specified value.	**a[href$=".pdf"]** Matches any <a> element whose href attribute ends with the text “.pdf“

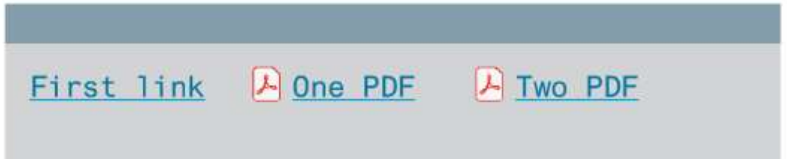
Attribute Selector Example

Here we show an attribute selector to show PDF files differently than other links

```
a[href$=".pdf"] {
   background: url(pdf_icon.svg) no-repeat left center;
   padding-left:19px;
}
```

This attribute selector selects only those <a> elements whose href attribute value ends with the characters ".pdf".

```
<a href="abc.html">First link</a>
<a href="sample.pdf">One PDF</a>
<a href="another.pdf">Two PDF</a>
```



Pseudo-Element and Pseudo-Class Selectors

A **pseudo-element selector** is a way to select something that does not exist explicitly **as an element** in the HTML document tree.

- You can select the **first line** or **first letter** of any HTML element using a pseudo-element selector.

A **pseudo-class selector** targets either a particular state or a variety of family relationships

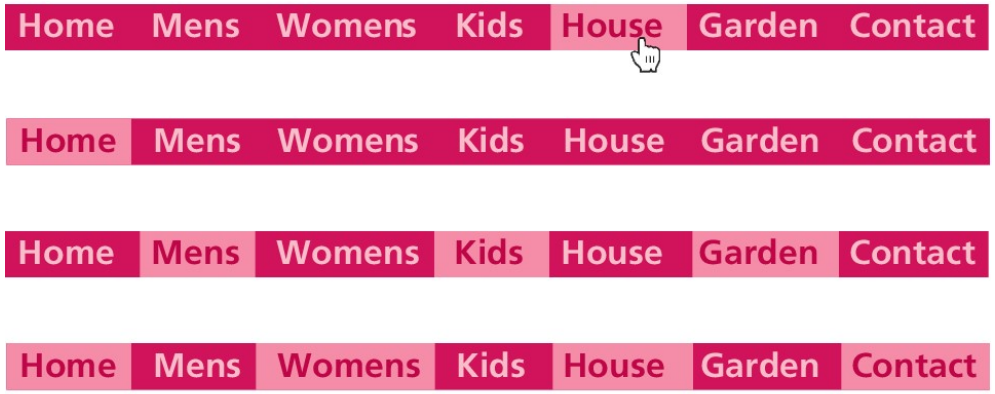
- This type of selector is for targeting link states and for adding hover styling for other elements

Common Pseudo-Class and Pseudo-Element Selectors

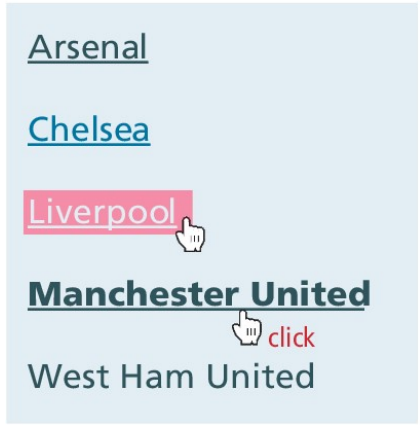
- **a:link** Selects links that have not been visited.
- **a:visited** Selects links that have been visited.
- **:hover** Selects elements that the mouse pointer is currently above.
- **:active** Selects an element that is being activated by the user. A typical example is a link that is being clicked.
- **:first-child** Selects an element that is the first child of its parent.
- **:first-letter** Selects the first letter of an element.
- **:first-line** Selects the first line of an element.

Source code: chapter03/02_selectors_2

Styling a link using pseudo-class selectors



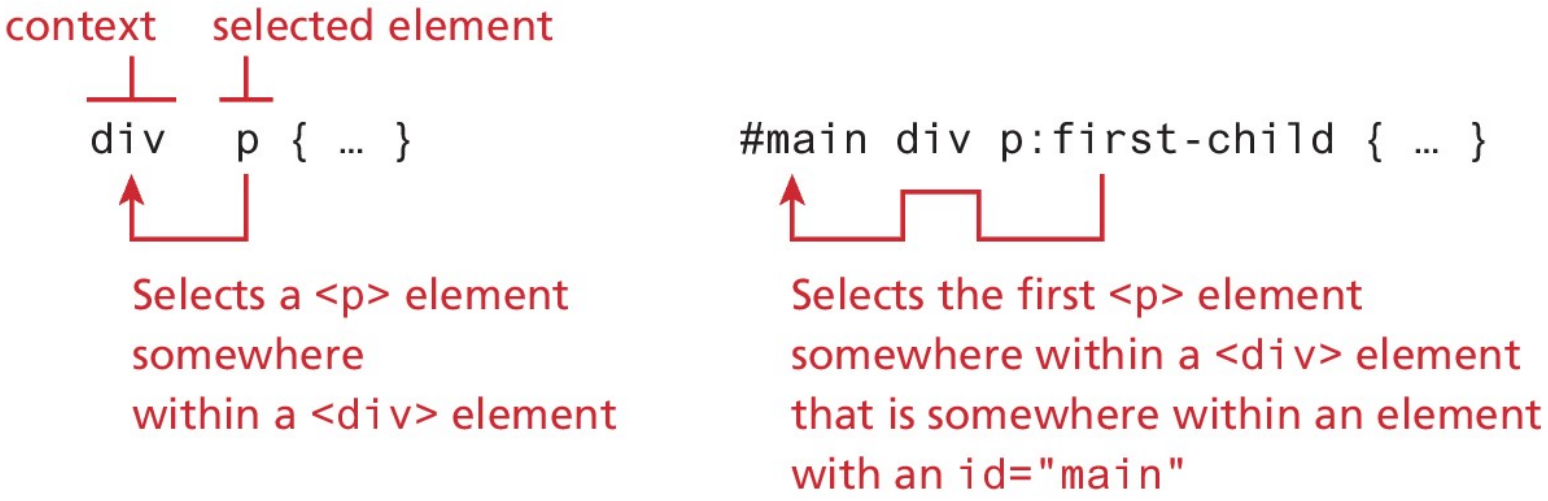

```
li:hover { ... }
li:first-child { ... }
li:nth-child(2n) { ... }
li:nth-child(2n-1) { ... }
```




```
a:link { ... }
a:visited { color: royalblue }
a:hover { color: lavender; background-color: hotpink}
a:active { font-weight: bold }
a:link:last-child { text-decoration: none}
```

Pseudo selectors can be combined

Syntax of a descendant selection

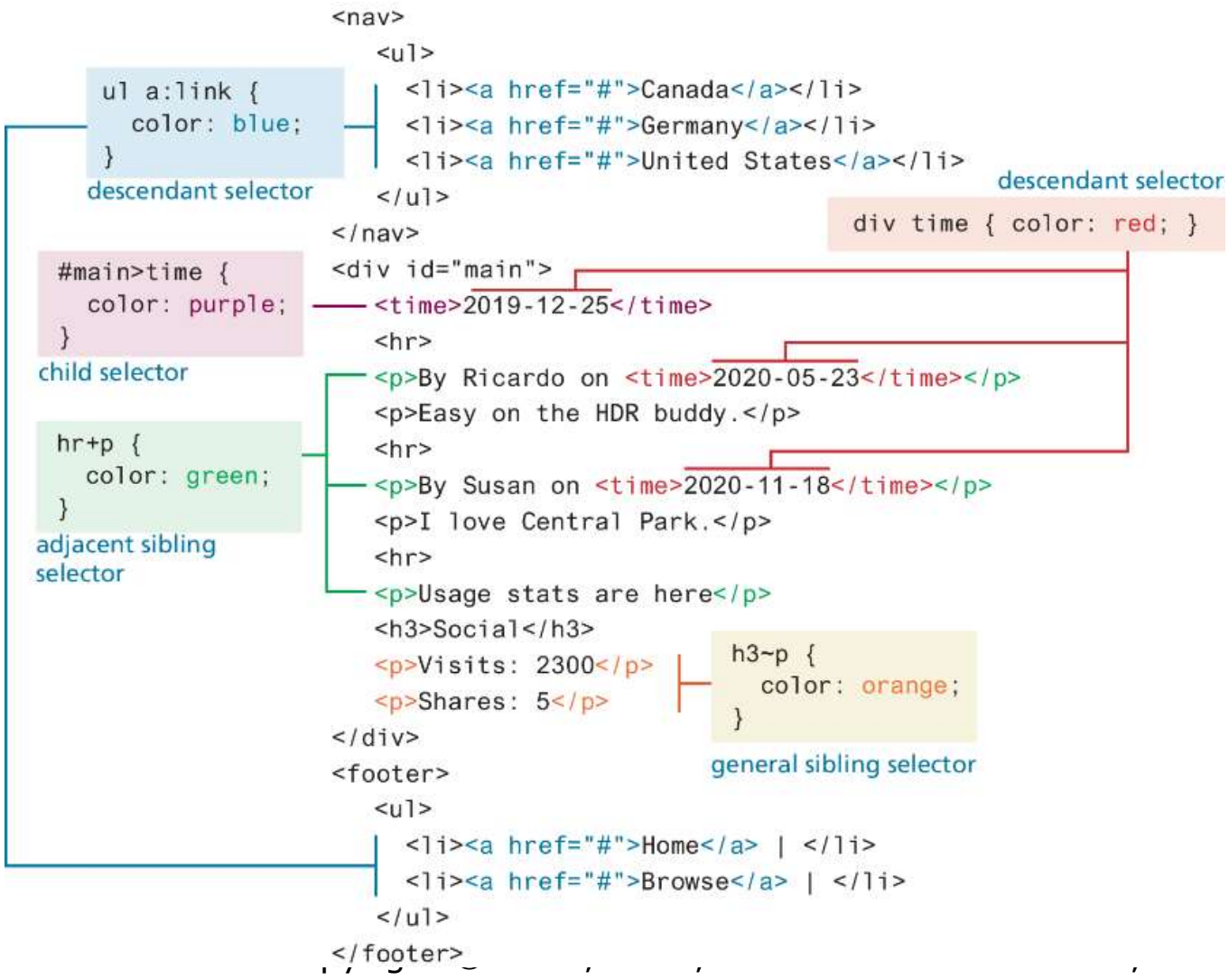


Contextual Selectors

A **contextual selector** (in CSS3 also called **combinators**) allows you to select elements based on their *ancestors*, *descendants*, or *siblings*.

- A **descendant selector** matches all elements that are contained within another element. The character used to indicate **descendant** selection is a space " "
- A **child selector** matches a specified element that is a **direct child** of the specified element. The character used is a ">"
- An **Adjacent selector** matches a specified element that is the next sibling of the specified element. The character used is a "+"
- A **General sibling selector** matches All **following** elements that shares the same parent as the specified element. The character used is a "~"

Contextual selectors in action



The Cascade: How Styles Interact

The “Cascade” in CSS refers to how conflicting rules are handled.

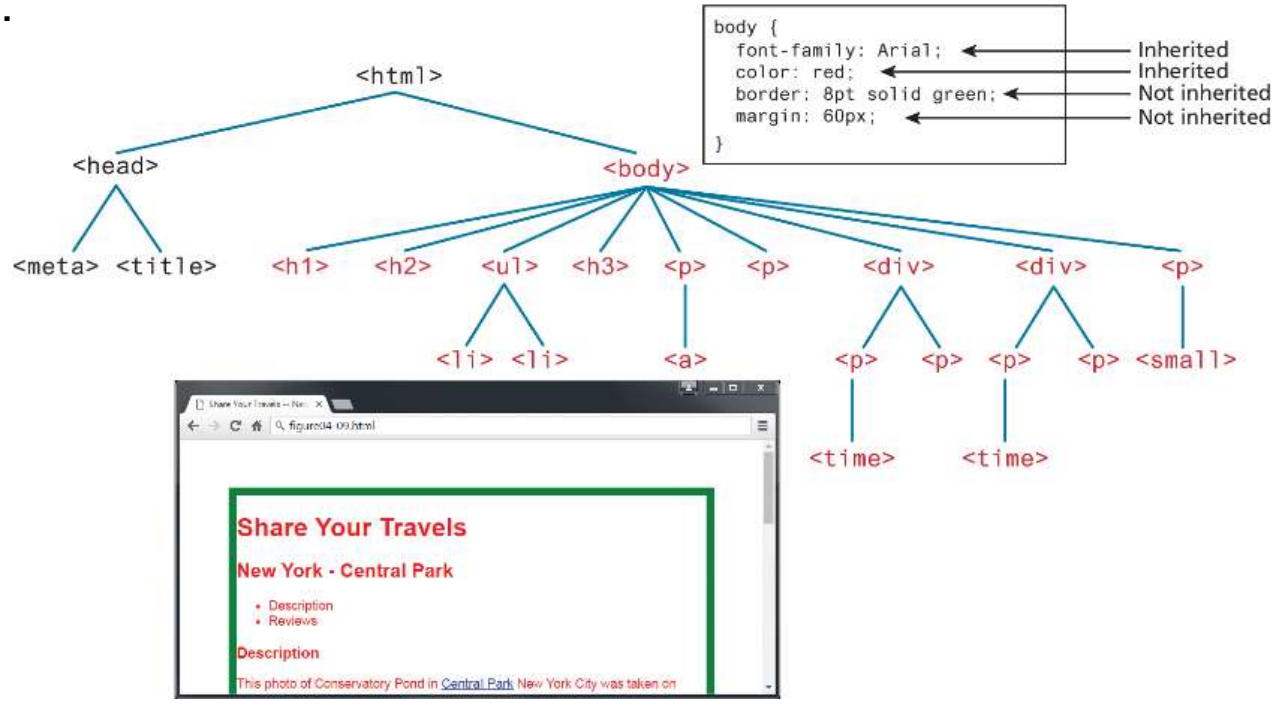
CSS uses the following cascade principles to help it deal with conflicts:

- inheritance,
- specificity, and
- location.

The Cascade: Inheritance

Inheritance is the principle that many CSS properties affect their descendants as well as themselves.

- Font, color, list, and text properties are inheritable;
- Layout, sizing, border, background, and spacing properties are not.
- it is also possible to inherit properties that are normally not inheritable using **inherit** *{color: inherit;}*

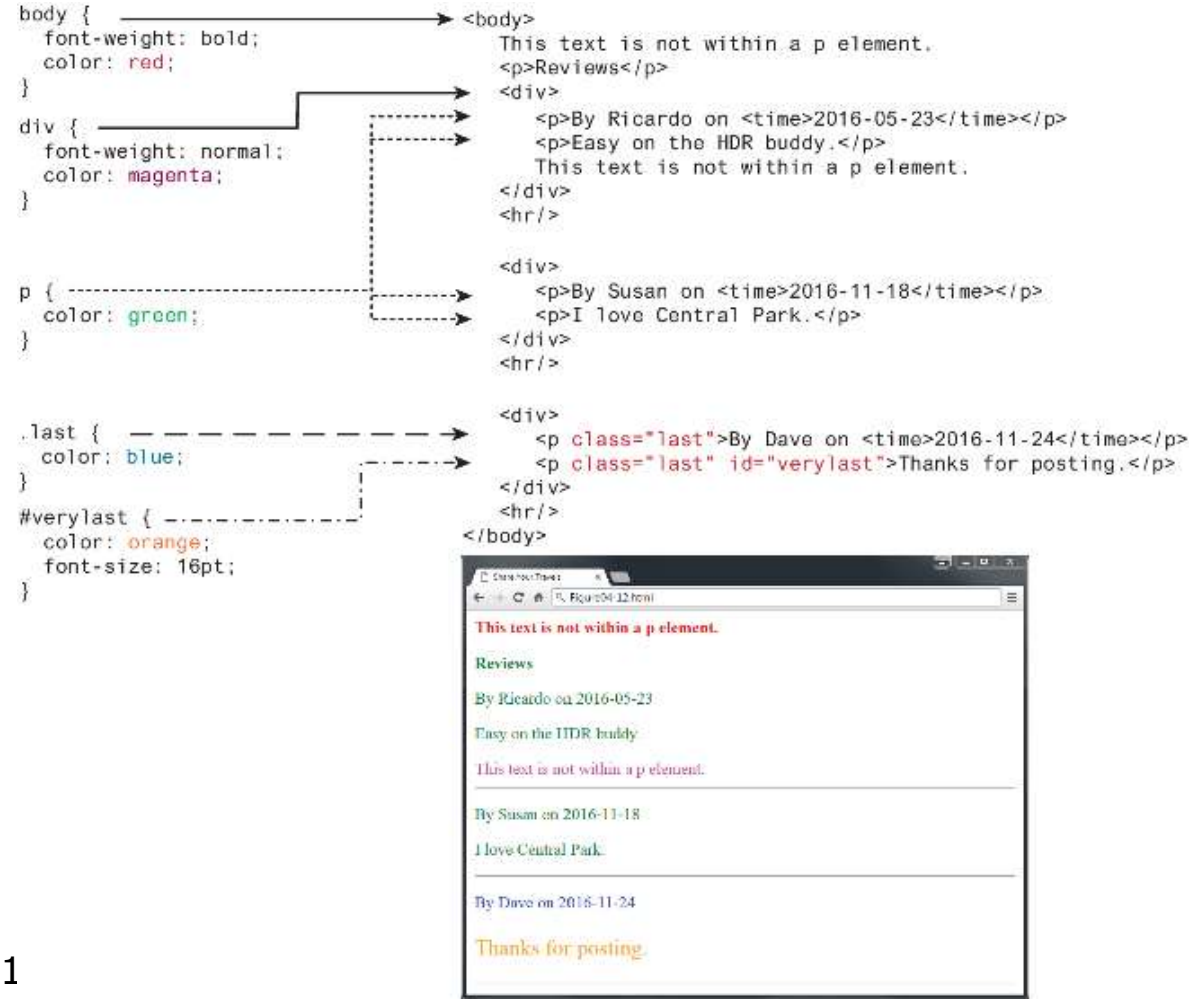


The Cascade: Specificity

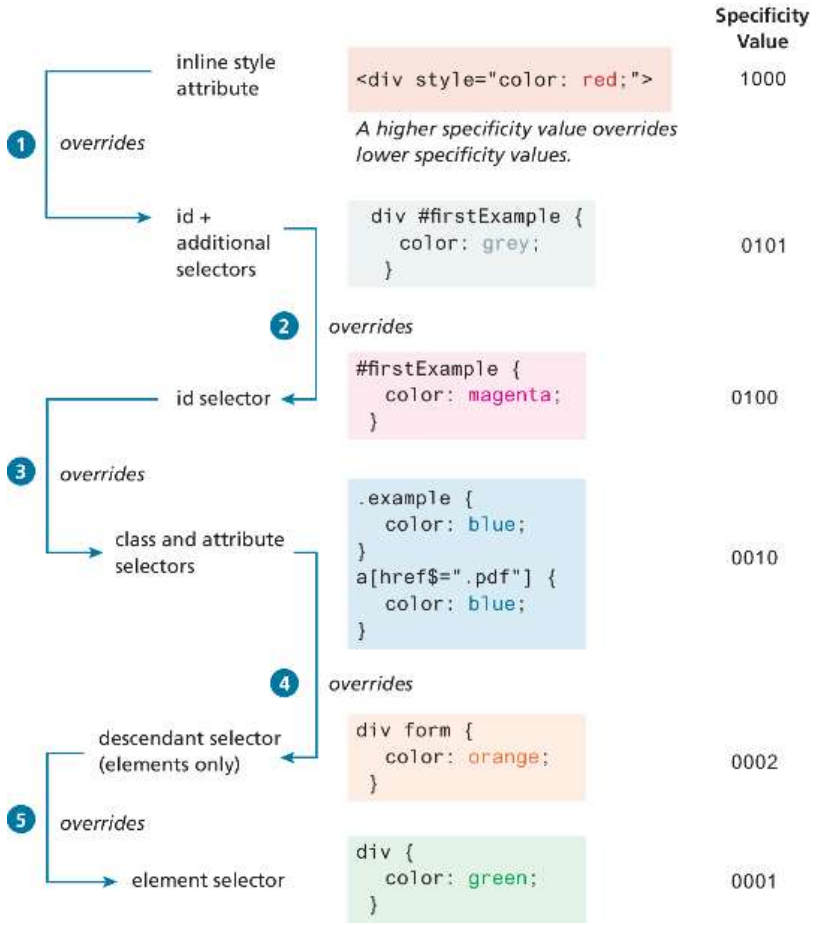
Specificity is how the browser determines which style rule takes precedence when more than one style rule could be applied.

- The more specific selector takes precedence
- Class selectors take precedence over element selectors, and id selectors take precedence over class selectors

Note: <body> color and font-weight properties are overridden by the more specific and <p> selectors.



Simplified Specificity algorithm



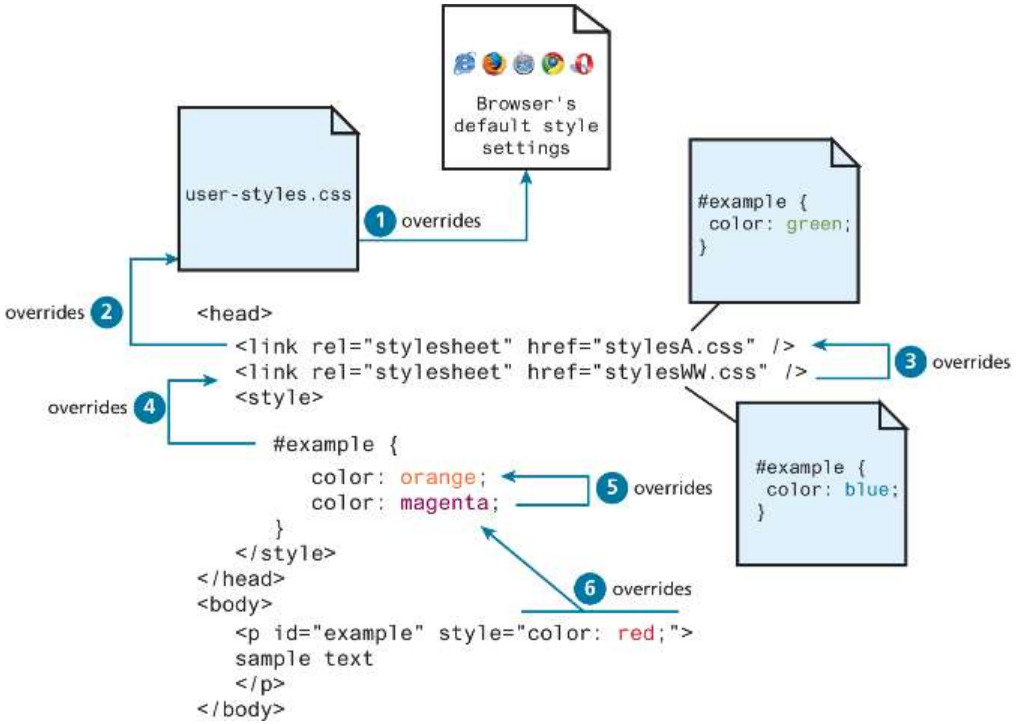
The Cascade: Location

Finally, when inheritance and specificity cannot determine style precedence, the principle of **location** will be used.

When rules have the same specificity, then the latest are given more weight.

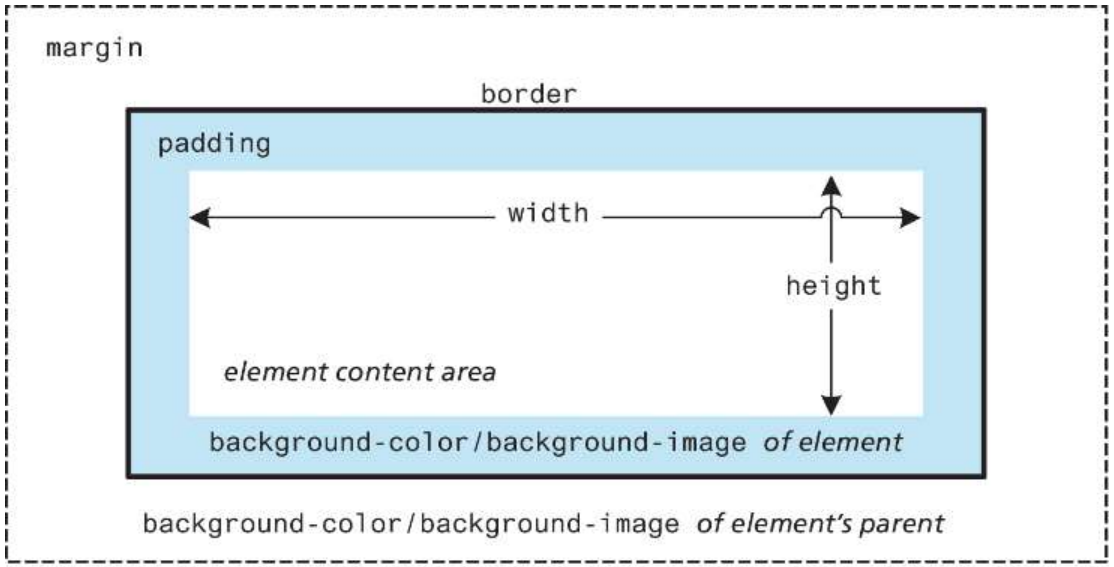
For instance, an inline style will override one defined in an external author style sheet or an embedded style sheet.

Note: browser user styles are less and less used today.



The Box Model

In CSS, all HTML elements exist within an **element box.** In order to become proficient with CSS, you must become familiar with the box model.



Block Elements

Block-level elements such as <p>, , <h2>, <ul>, and <table> are each contained on their own line.

- without styling, two block-level elements can't exist on the same line
- Block-level elements use the normal CSS box model

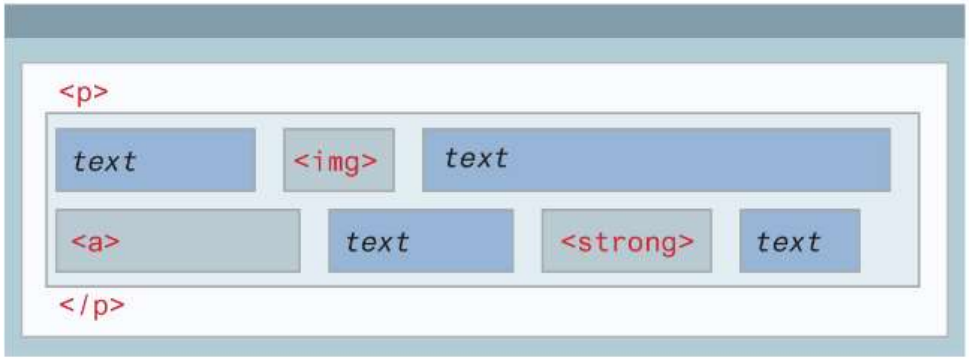
Inline Elements

Inline elements do not form their own blocks but instead are displayed within lines.

- Normal text in an HTML document is inline, as are elements such as , <a>, <img>, and .
- When there isn't enough space left on the line, the content moves to a new line

Inline Element Example

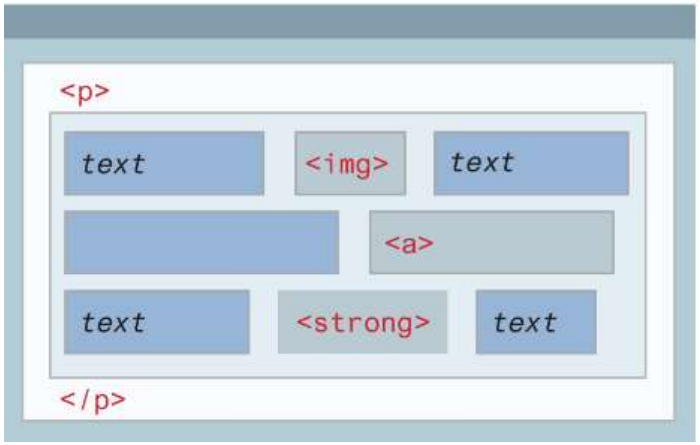
```
<p>
This photo <img src="photo-con.png"> of Conservatory Pond in
<a href="http://www.centralpark.com">Central Park</a> was
taken with a <strong>Canon EOS 30D</strong> camera.
</p>
```



Inline content is laid out horizontally left to right within its container.

Once a line is filled with content, the next line will receive the remaining content, and so on.

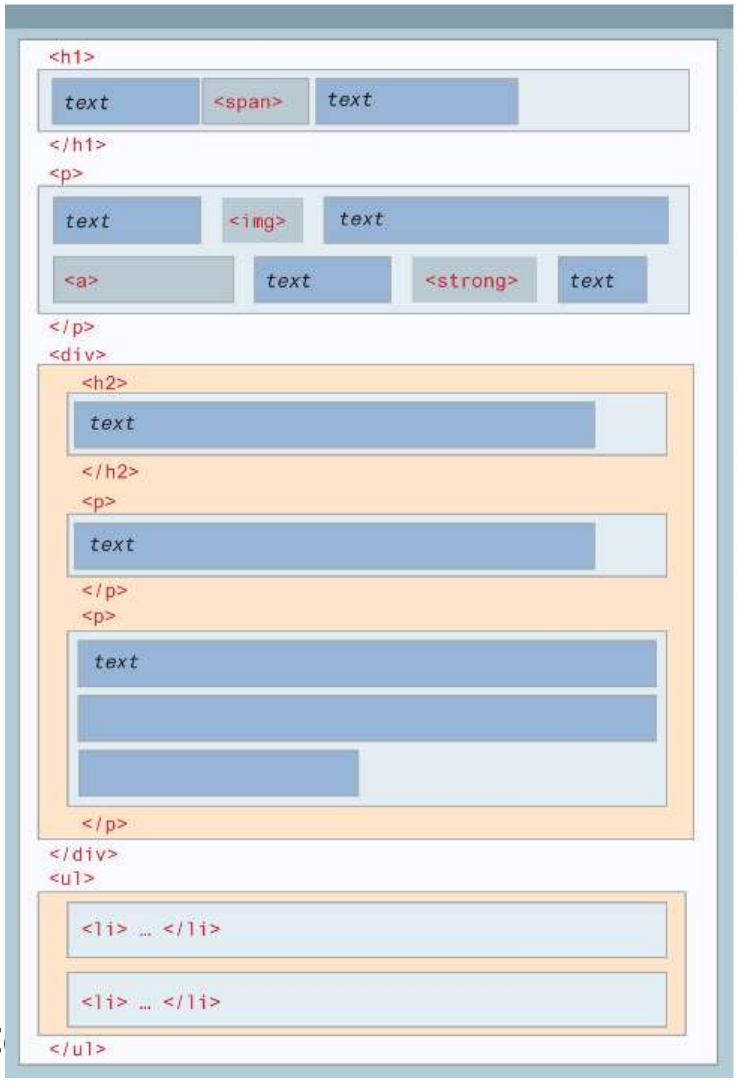
Here the content of this <p> element is displayed on two lines.



If the browser window resizes, then inline content will be "reflowed" based on the new width.

Here the content of this <p> element is now displayed on three lines.

Block and inline elements together



A document consists of block-level elements stacked from top to bottom.

Within a block, inline content is horizontally placed left to right.

Some block-level elements can contain other block-level elements (in this example, a can contain other blocks).

In such a case, the block-level content inside the parent is stacked from top to bottom within the container ( ).

Border and Shadow Examples

```
border-style: solid;
```



```
border-style: dotted;
```

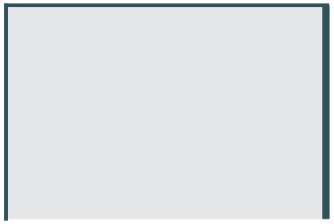


```
border-style: dashed;
```



```
border-width: 0;
```

```
border-width: 1px;
```



```
border-width: 1px 2px 0 1px;
```

```
border: solid 2px gold;
```

```
border-radius: 3px;
border-width: 1px;
```



```
border-radius: 3px;
border-width: 0;
```

```
box-shadow: 2px 2px gray;
```

```
box-shadow: 2px 2px 2px 2px gray;
```

```
/* offset-x | offset-y | blur-radius | spread-radius | color */
box-shadow: 2px 2px 2px 1px rgba(0, 0, 0, 0.2);
```

```
/* Any number of shadows, separated by commas */
box-shadow: 3px 3px red, -1em 0 0.4em olive;
```

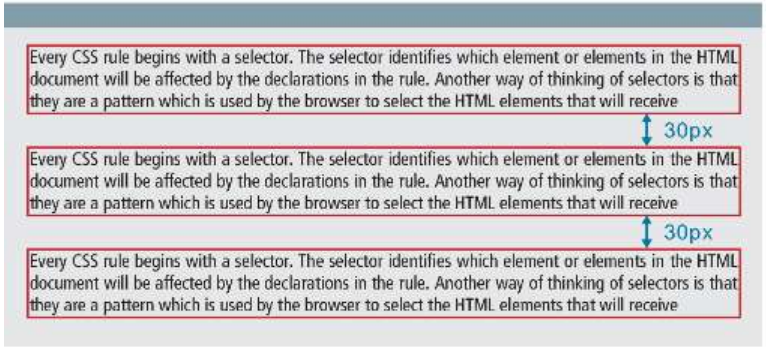
Pearson

Margins and Padding

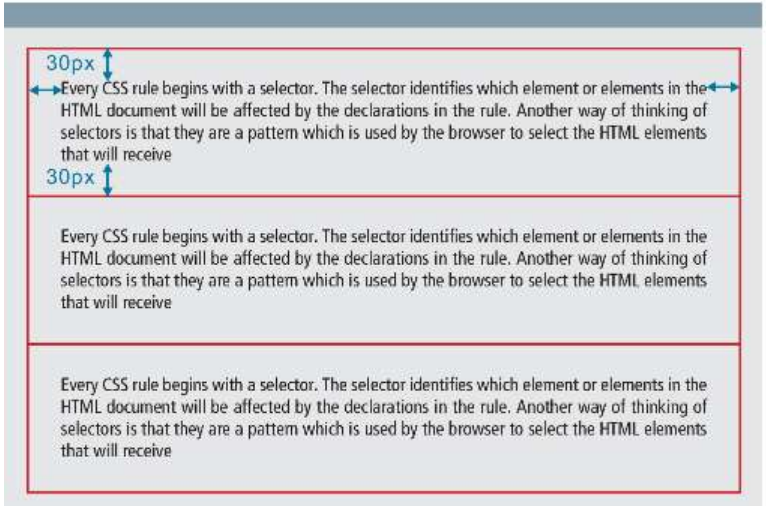
Margins add spacing around an element's content

Padding adds spacing within elements.

Borders divide the margin area from the padding area.




```
p {
  border: solid 1pt red;
  margin: 30px;
  padding: 0;
}
```




```
p {
  border: solid 1pt red;
  margin: 0;
  padding: 30px;
}
```

Collapsing margins

The W3C defines **collapsing margins** as:

- *In CSS, the adjoining margins of two or more boxes (which might or might not be siblings) can combine to form a single margin. Margins that combine this way are said to collapse, and the resulting combined margin is called a collapsed margin.*

What this means is that when the **vertical** margins of two elements touch, only the largest margin value of the elements will be displayed

Horizontal margins, on the other hand, **never** collapse.

Box Model: All or one

NOTE
With border, margin, and padding properties, it is possible to set the properties for one or more sides in a single property, or individually

```
border-top-color: red; /* sets just the top side */
border-right-color: green; /* sets just the right side */
border-bottom-color: yellow; /* sets just the bottom side */
border-left-color: blue; /* sets just the left side */
```

Alternately, we can set all four sides to a single value via:

```
border-color: red; /* sets all four sides to red */
```

Or we can set all four sides to different values via:

```
border-color: red green orange blue;
```

Another shortcut is to use just two values; in this case the first value sets top and bottom, while the second sets the right and left.

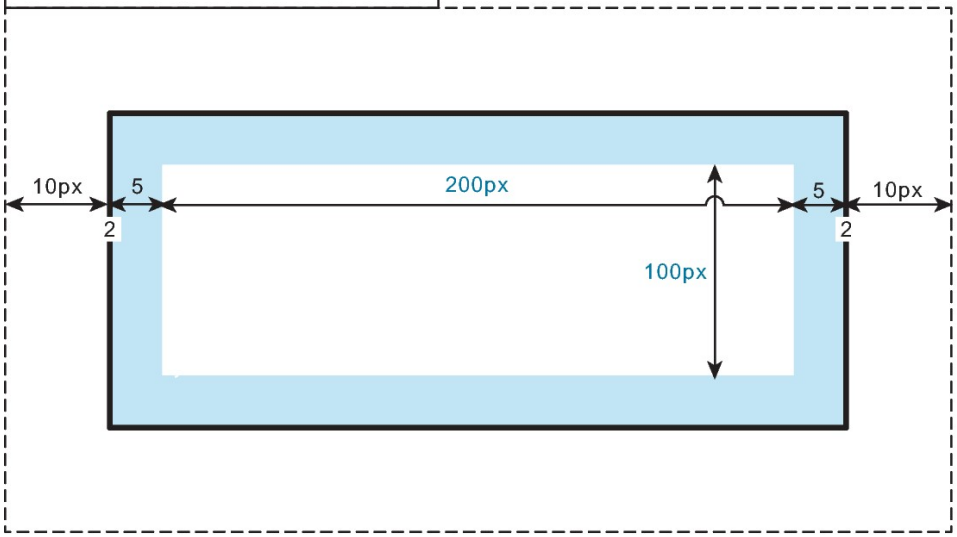
```
border-color: red yellow; /* top+bottom=red, right+left=yellow */
```

Box Dimensions

- Block-level elements have **width** and **height** properties.
- They also have a **min-width, min-height, max-width**, and **max-height** properties as well to help when the width or a height might be specified as a % of its parent container
- Since the width and the height only refer to the size of the content area, the **total size** of an element is equal to the size of its content area plus the sum of its padding, borders, and margins.

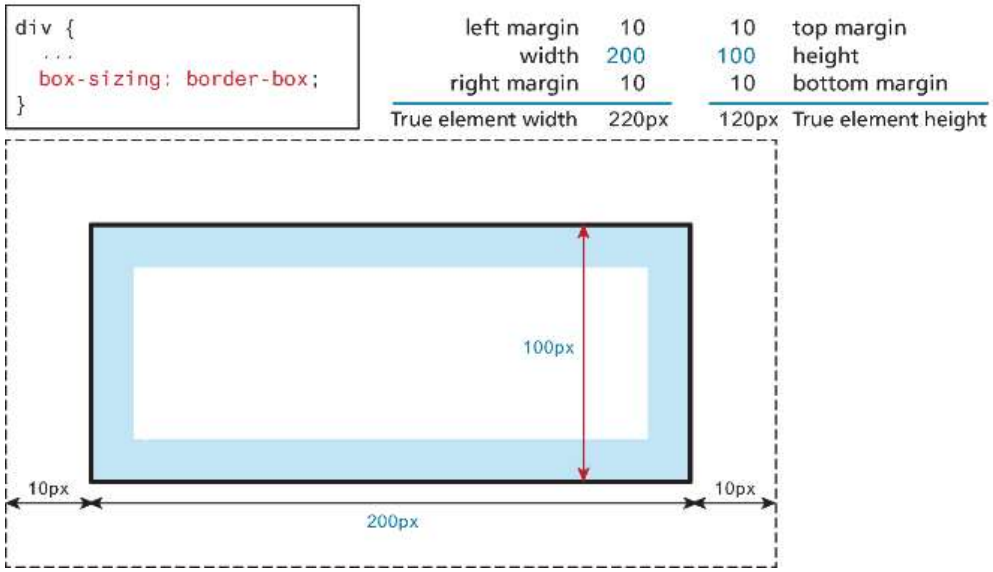
```
div {
  box-sizing: content-box;
  width: 200px;
  height: 100px;
  padding: 5px;
  margin: 10px;
  border: solid 2pt black;
}
```

left margin	10	10	top margin
left border	2	2	top border
left padding	5	5	top padding
width	200	100	height
right padding	5	5	bottom padding
right border	2	2	bottom border
right margin	10	10	bottom margin
True element width	234px	134px	True element height



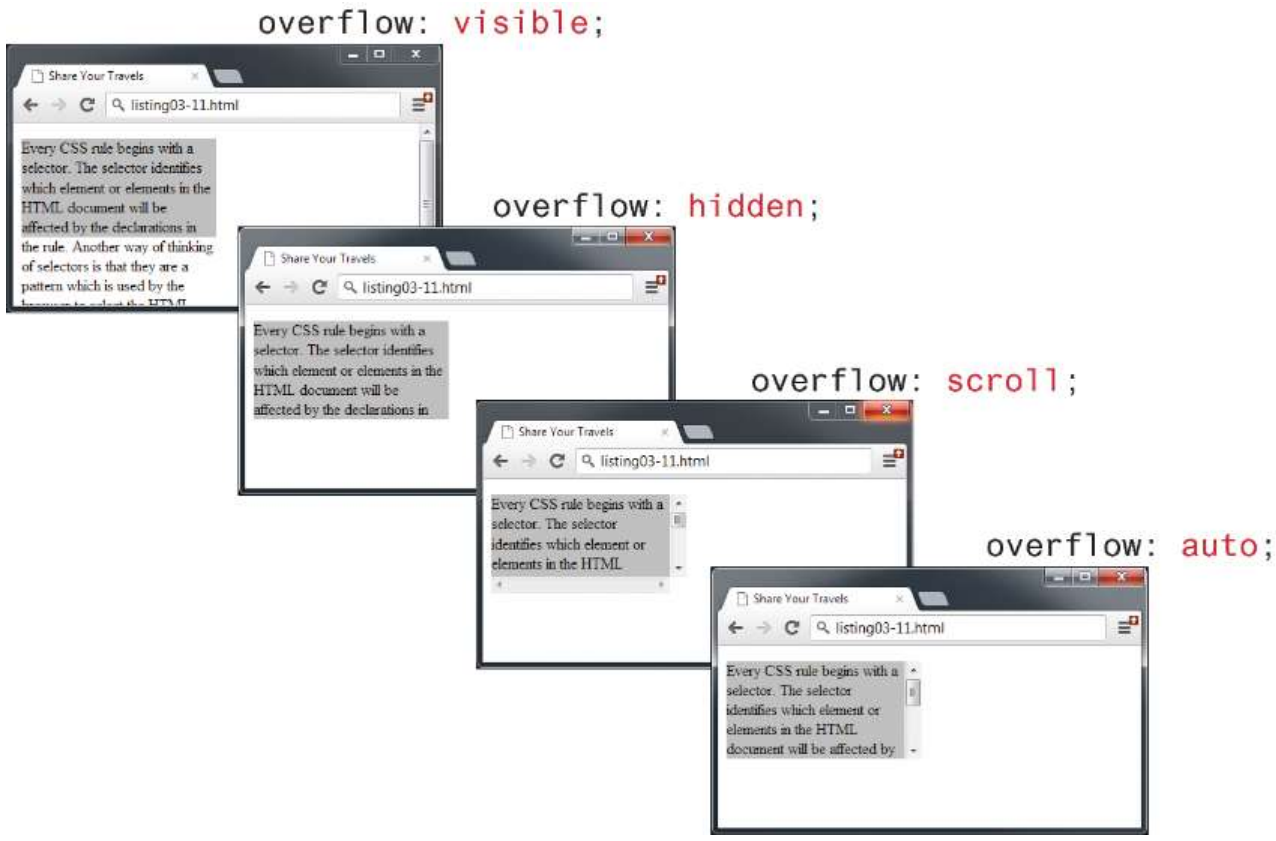
The border-box approach

To reduce complexity of adding margins, content and borders, you can use the newer border-box approach that is more intuitive.



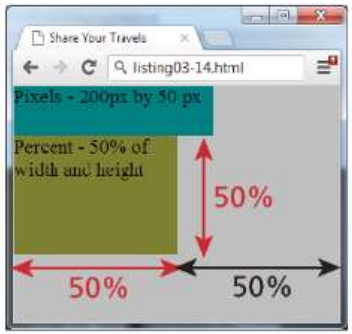
Overflow Property

It is possible to control what happens with the content if the box's width and height are not large enough to display the content using the **overflow** property

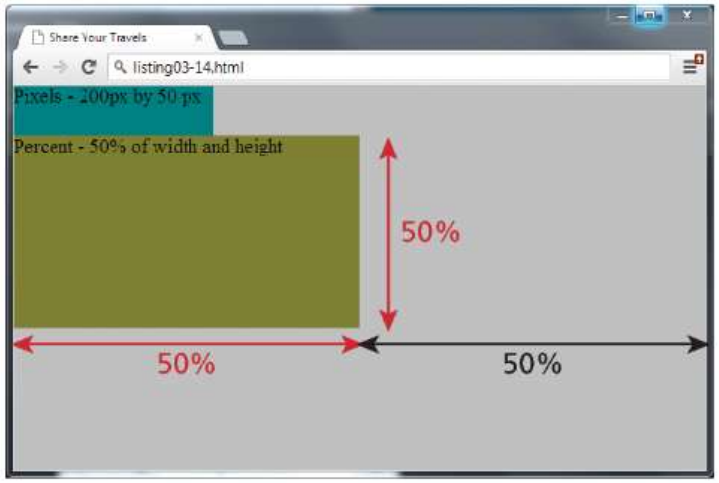


Box sizing via percent

```
<style>
  html,body {
     margin:0;
     width:100%;
     height:100%;
     background: silver;
  }
  .pixels {
     width:200px;
     height:50px;
     background: teal;
  }
  .percent {
     width:50%;
     height:50%;
     background: olive;
  }
```




```
<body>
   <div class="pixels">
     Pixels - 200px by 50 px
   </div>
   <div class="percent">
     Percent - 50% of width and height
   </div>
</body>
```



Source code: chapter03/04_box_sizing

CSS Text Styling

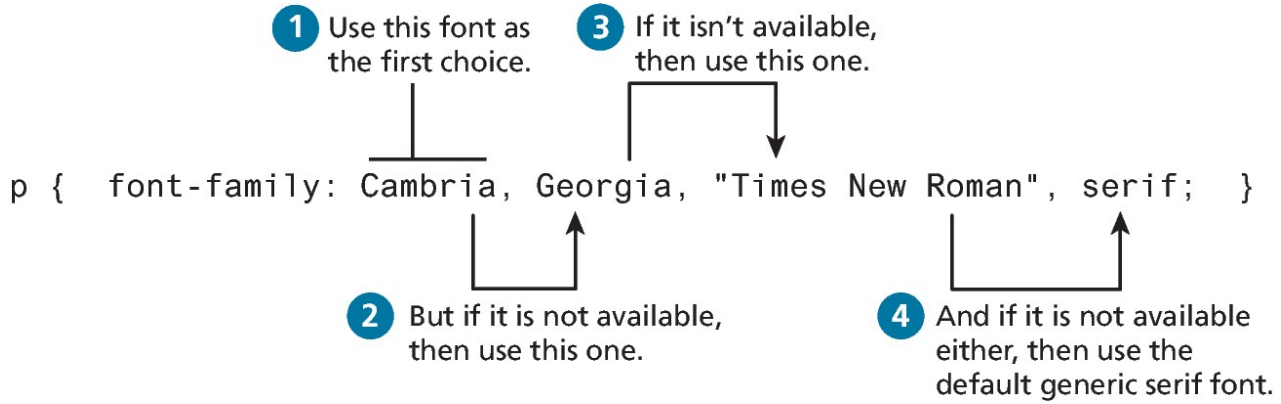
CSS provides two types of properties that affect text.

- **Font properties** that affect the font and its appearance.
- **Paragraph properties** that affect the text in a similar way no matter which font is being used
- Font properties include
  - font, font-family, font-style, font-size, font-variant, font-weight

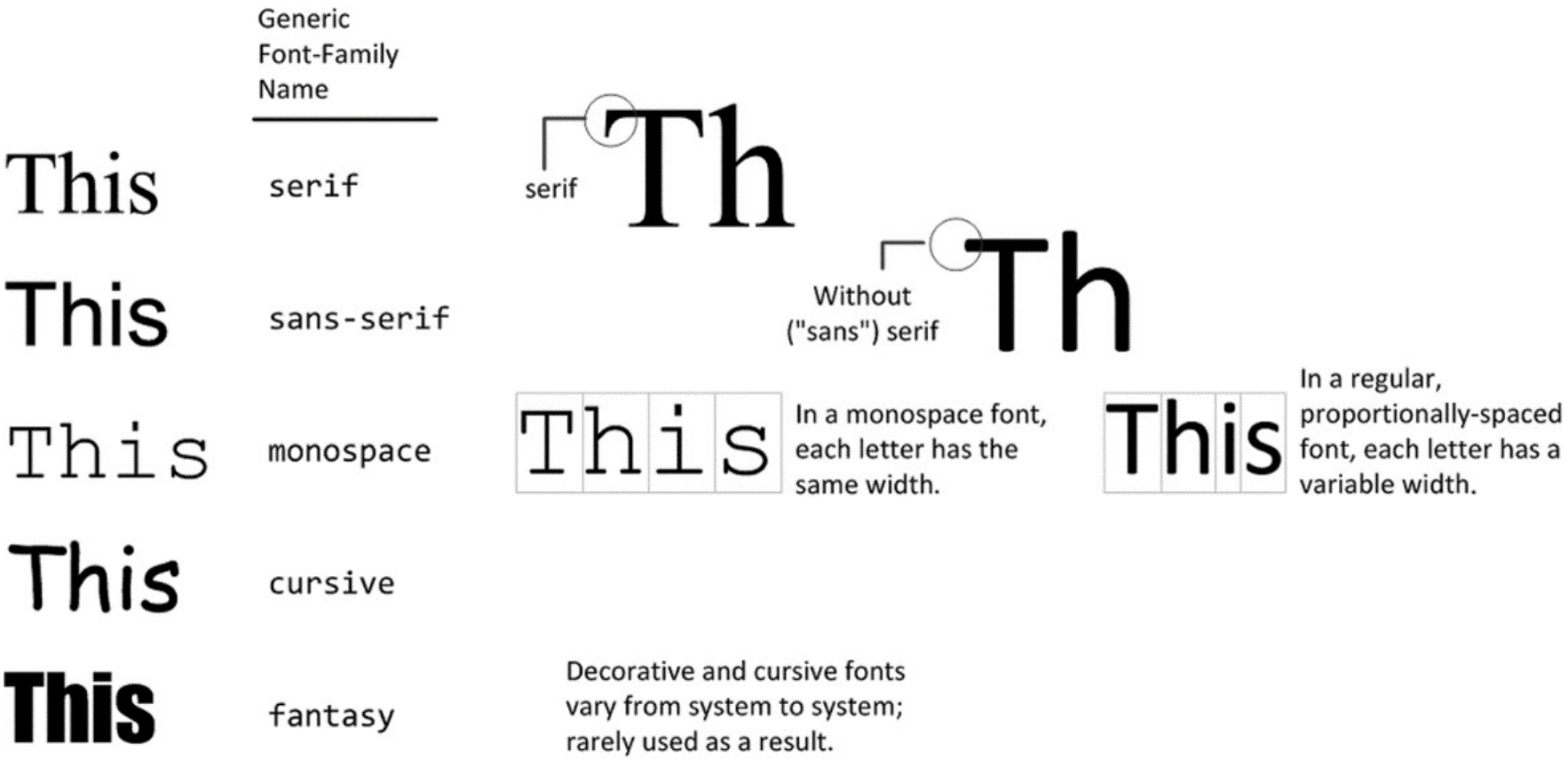
Font Family

Just because a given font is available on the web developer's computer does not mean it will be available for all users.

For this reason, it is conventional to supply a so-called **web font stack**— a series of alternate fonts (fall back fonts) to use in case the original font choice is not on the user's computer



Different font families



Font Sizes

If we wish to create web layouts that work well on different devices, we should learn to use relative units such as **em units** or **percentages** for our font sizes

- it can quickly become difficult to calculate actual sizes when there are nested elements
  - For this reason, CSS3 now supports a new relative measure, the **rem** (for root em unit). This unit is always relative to the size of the root element (i.e., the <html> element).

@import a font available online

DIVE DEEPER

You can use a font even if it is not installed on the user's computer. Open source font sites such as Google Web Fonts (https://fonts.google.com) and Font Squirrel (http://www.fontsquirrel.com/) have online fonts available for public use:

To use Droid Sans, you can add the following to your <head> section

```
<link href="https://fonts.googleapis.com/css?family=Droid+Sans" rel="stylesheet" type="text/css">
```

Or, add the following import inside one of your CSS files:

```
@import url('https://fonts.googleapis.com/css2?family=Roboto+Slab:wght@400&display=swap');
```

Pearson

Source code: chapter03/05_import_font

Font Weight

CSS Syntax

```
font-weight: normal|bold|bolder|lighter|number|initial|inherit;
```

Property Values

Value	Description
normal	Defines normal characters. This is default
bold	Defines thick characters
bolder	Defines thicker characters
lighter	Defines lighter characters
100 200 300 400 500 600 700 800 900	Defines from thin to thick characters. 400 is the same as normal, and 700 is the same as bold
initial	Sets this property to its default value. Read about *initial*

Text properties

Every CSS rule begins with a selector. The selector identifies which element or elements in the HTML document will be affected by the declarations in the rule.

```
line-height: normal;
```

Every CSS rule begins with a selector. The selector identifies which element or elements in the HTML document will be affected by the declarations in the rule.

```
line-height: 1.5;
```

Every CSS rule begins with a selector. The selector identifies which element or elements in the HTML document will be affected by the declarations in the rule.

```
text-indent: 4em;
```

Every CSS rule begins with a selector.

```
text-align: left;
```

Every CSS rule begins with a selector.

```
text-align: center;
```

Every CSS rule begins with a selector.

```
text-align: right;
```

Every CSS rule begins with a selector.

```
letter-spacing: 3px;
```

```
vertical-align: top;
```

```
vertical-align: middle;
```

```
vertical-align: bottom;
```

EVERY RULE BEGINS WITH A SELECTOR

```
text-transform: uppercase;
```

every rule begins with a selector

```
text-transform: lowercase;
```

selector

```
text-decoration: underline;
```

selector

```
text-decoration: none;
```

CSS Variables

CSS Variables (also called custom properties) are a new feature that let you

- define variables (which must begin with a double hyphen --x) at the top of your CSS file usually within a special **:root** pseudo-class selector, and
- reference those variables as property values using the **var()** CSS function.

Duplication of colors, spacing, and borders, often happens numerous times within a single file.

Example Part 1 (duplicate values)

```
header {
  background-color: #431c5d;
  color: #e05915;
  padding: 4px;
  box-shadow: 6px 5px 20px 1px rgba(0,0,0,0.22);
  margin: 0;
}
header button {
  background-color: #e05915;
  border-radius: 5px;
  border-color: #e6e9f0;
  padding: 4px;
  color: #e6e9f0;
  font-size: 18px;
  margin-top: 9px;
}
```

```
#results {
  background-color: #431c5d;
  font-size: 18px;
  border-radius: 5px;
  padding: 4px;
  box-shadow: 6px 5px 20px 1px rgba(0,0,0,0.22);
}
```

LISTING 4.8 Duplicate property values in CSS

Example Part 2 (CSS variables)

```
:root {
  --bg-color-main: #431c5d;
  --bg-color-secondary: #e05915;
  --fg-color-main: #e6e9f0;
  --radius-boxes: 5px;
  --padding-boxes: 4px;
  --fontsize-default: 18px;
  --shadow-color: rgba(0,0,0,0.22);
  --dropshadow: 6px 5px 20px 1px var(--shadow-color);
}
header {
  background-color: var(--bg-color-main);
  color: var(--bg-color-secondary);
  padding: var(--padding-boxes);
  box-shadow: var(--dropshadow);
  margin: 0;
}
```

```
header button {
  background-color: var(--bg-color-secondary);
  border-radius: var(--radius-boxes);
  border-color: var(--fg-color-main);
  padding: var(--padding-boxes);
  color: var(--fg-color-main);
  font-size: var(--fontsize-default);
  margin-top: calc( --fontsize-default / 2 );
}
#results {
  background-color: var(--bg-color-main);
  font-size: var(--fontsize-default:);
  border-radius: var(--radius-boxes);
  padding: var(--padding-boxes);
  box-shadow: var(--dropshadow);
}
```

LISTING 4.9 Using CSS Variables

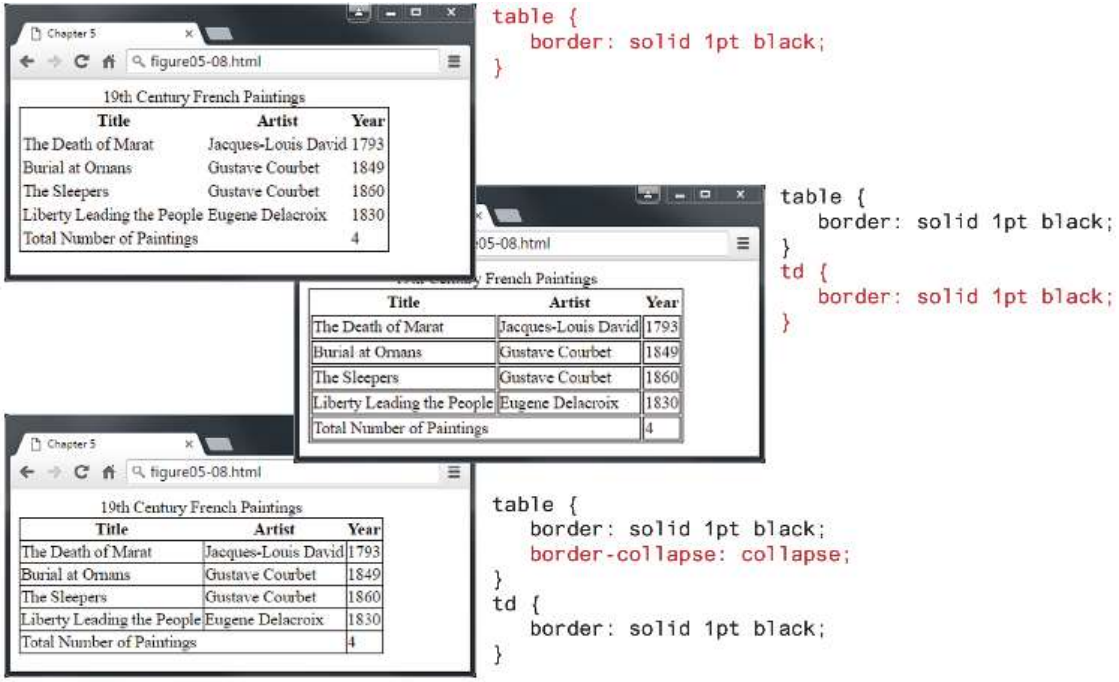
Styling Tables (Borders)

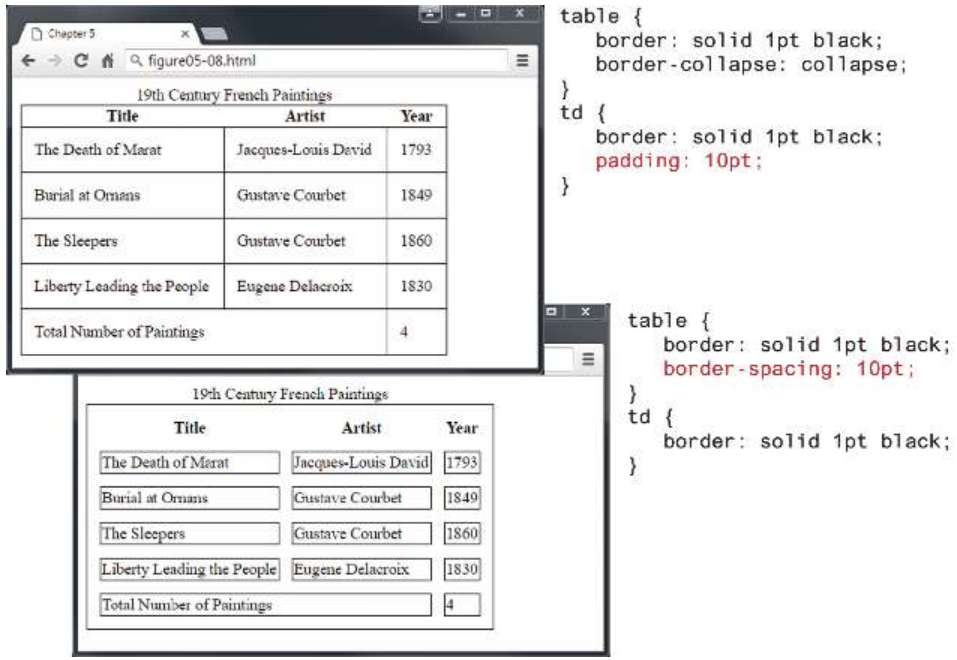
Borders can be assigned to the **<table>, <th>** and the **<td>** element. Interestingly, borders cannot be assigned to the <tr>, <thead>, <tfoot>, and <tbody> elements.

Notice as well the **border-collapse** property. This property selects the table's border model.

- The default is the **separated** model or value (each cell has its own unique borders)
- The **collapsed** border model makes adjacent cells share a single border.

Styling table borders





Boxes and Zebras

While there is almost no end to the different ways one can style a table, there are a number of common approaches. We will look at two of them here.

- The first of these is a box format, in which we simply apply background colors and borders in various ways


```
caption {
   font-weight: bold;
   padding: 0.25em 0 0.25em 0;
   text-align: left;
   text-transform: uppercase;
   border-top: 1px solid #DCA806;
}
table {
   font-size: 0.8em;
   font-family: Arial, sans-serif;
   border-collapse: collapse;
   border-top: 4px solid #DCA806;
   border-bottom: 1px solid white;
   text-align: left;
}
```

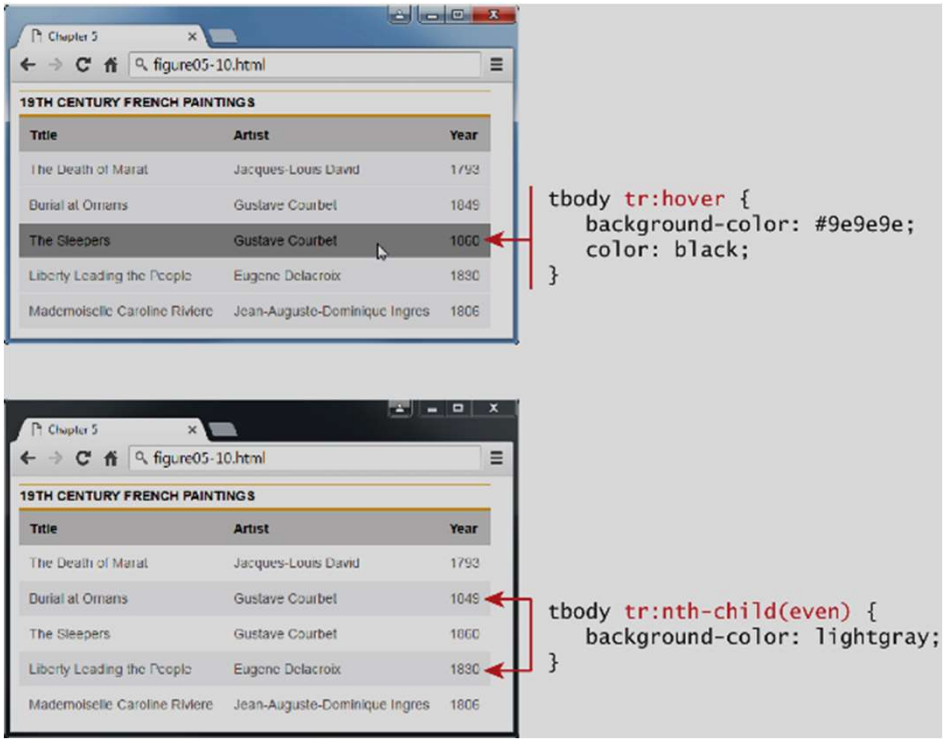

```
thead tr {
   background-color: #CACACA;
}
th {
   padding: 0.75em;
}
```


```
tbody tr {
   background-color: #F1F1F1;
   border-bottom: 1px solid white;
   color: #6E6E6E;
}
tbody td {
   padding: 0.75em;
}
```

Boxes and Zebras (ii)

We can then

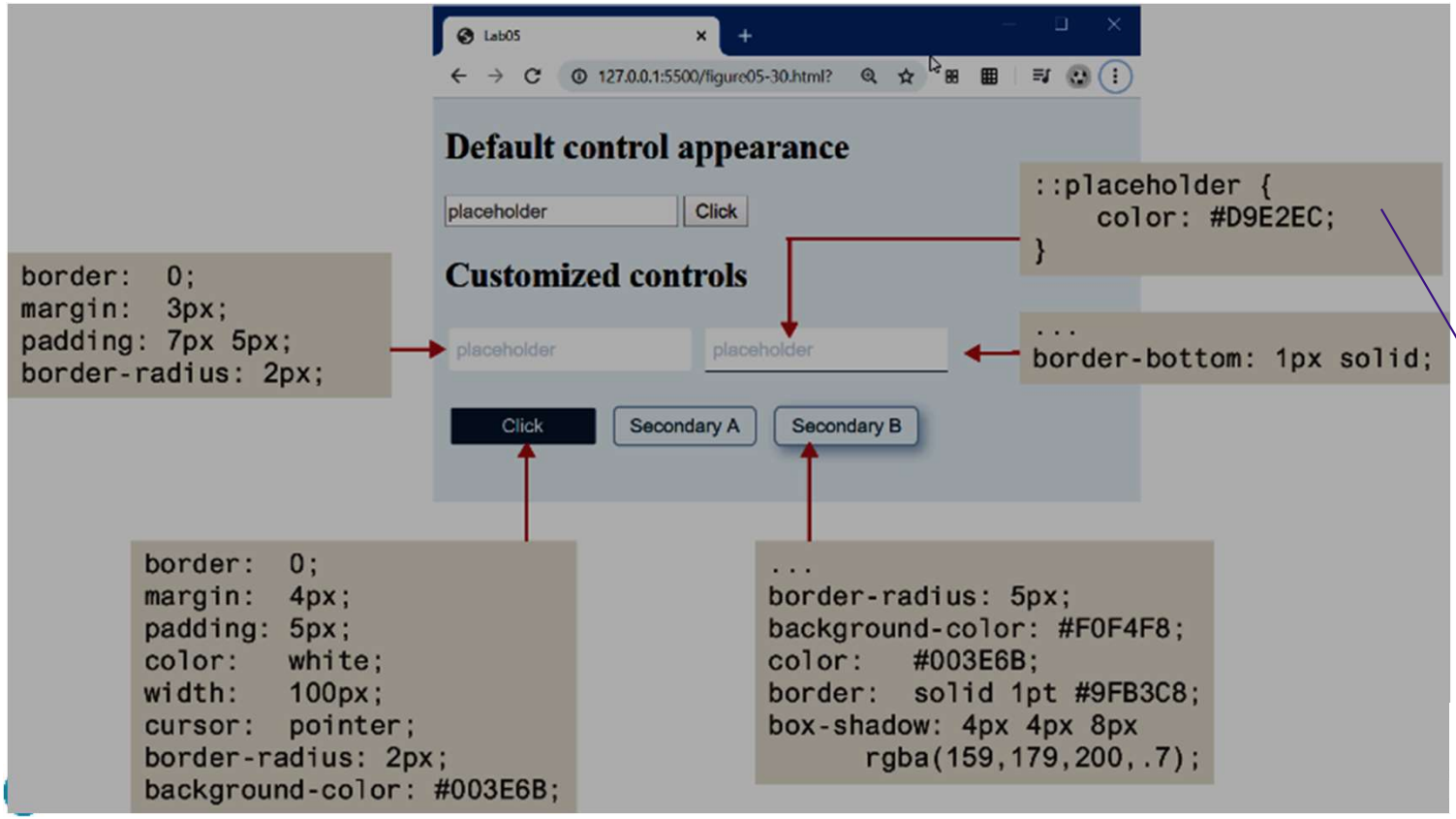
- add special styling to the :hover pseudo-class of the <tr> element to highlight a row when the mouse cursor hovers over
- use the pseudo-element nth-child to alternate the format of every second row.



Source code: chapter03/06_style_tables

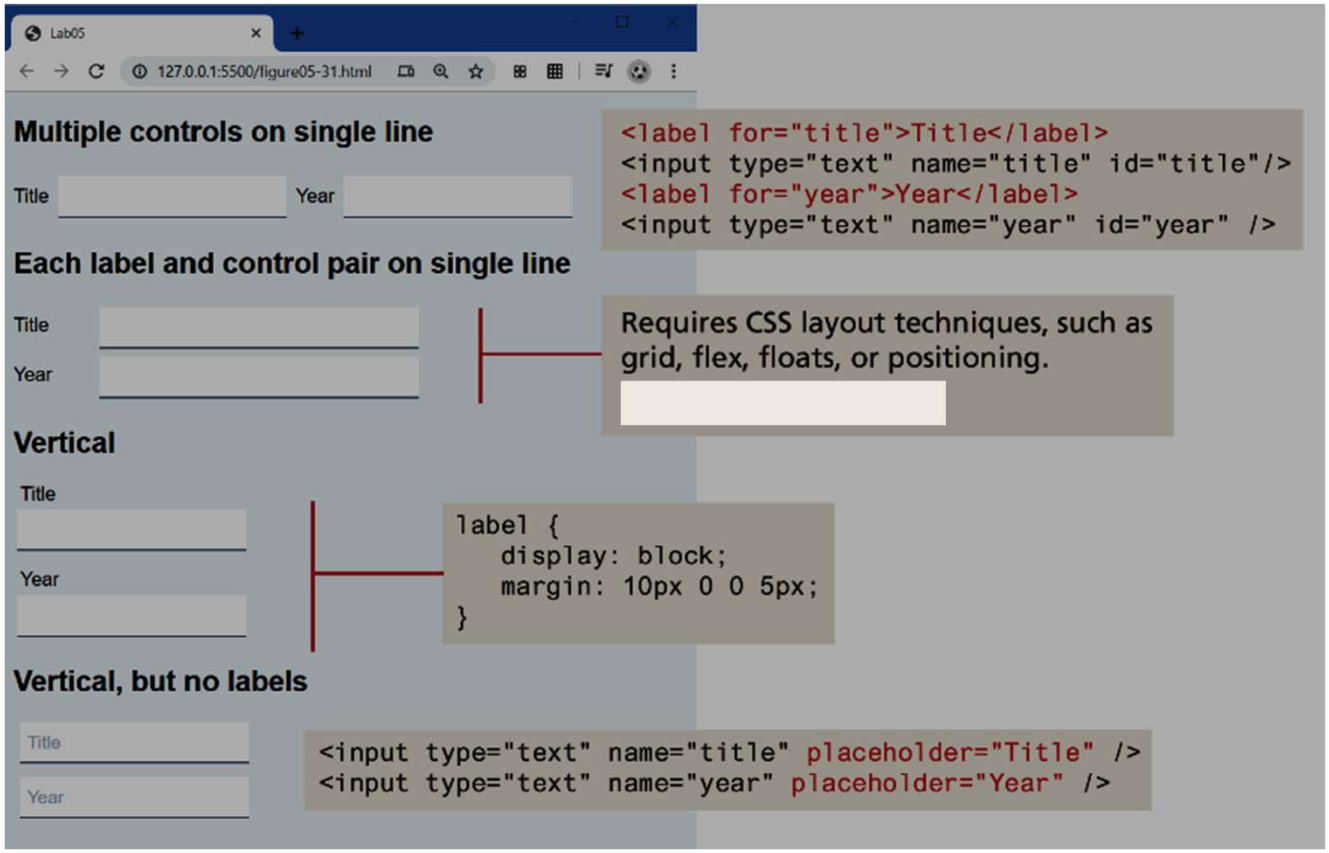
Styling Form Elements

Let's begin with the common text and button controls. A common styling change is to eliminate the borders and add in rounded corners and padding.



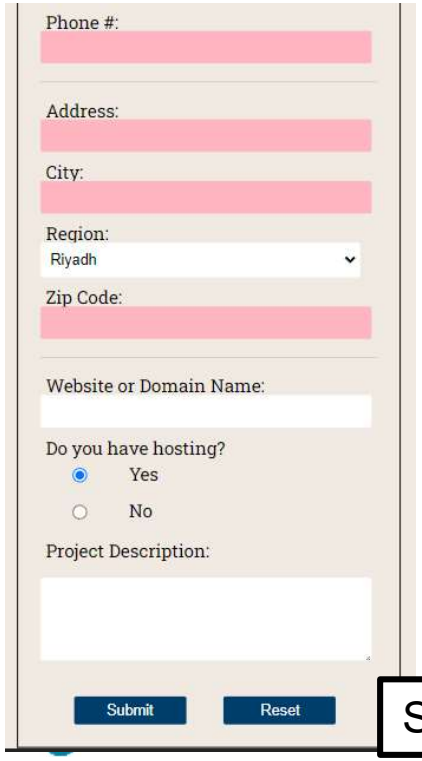
https://www.w3schools.com/cssref/css_selectors.asp

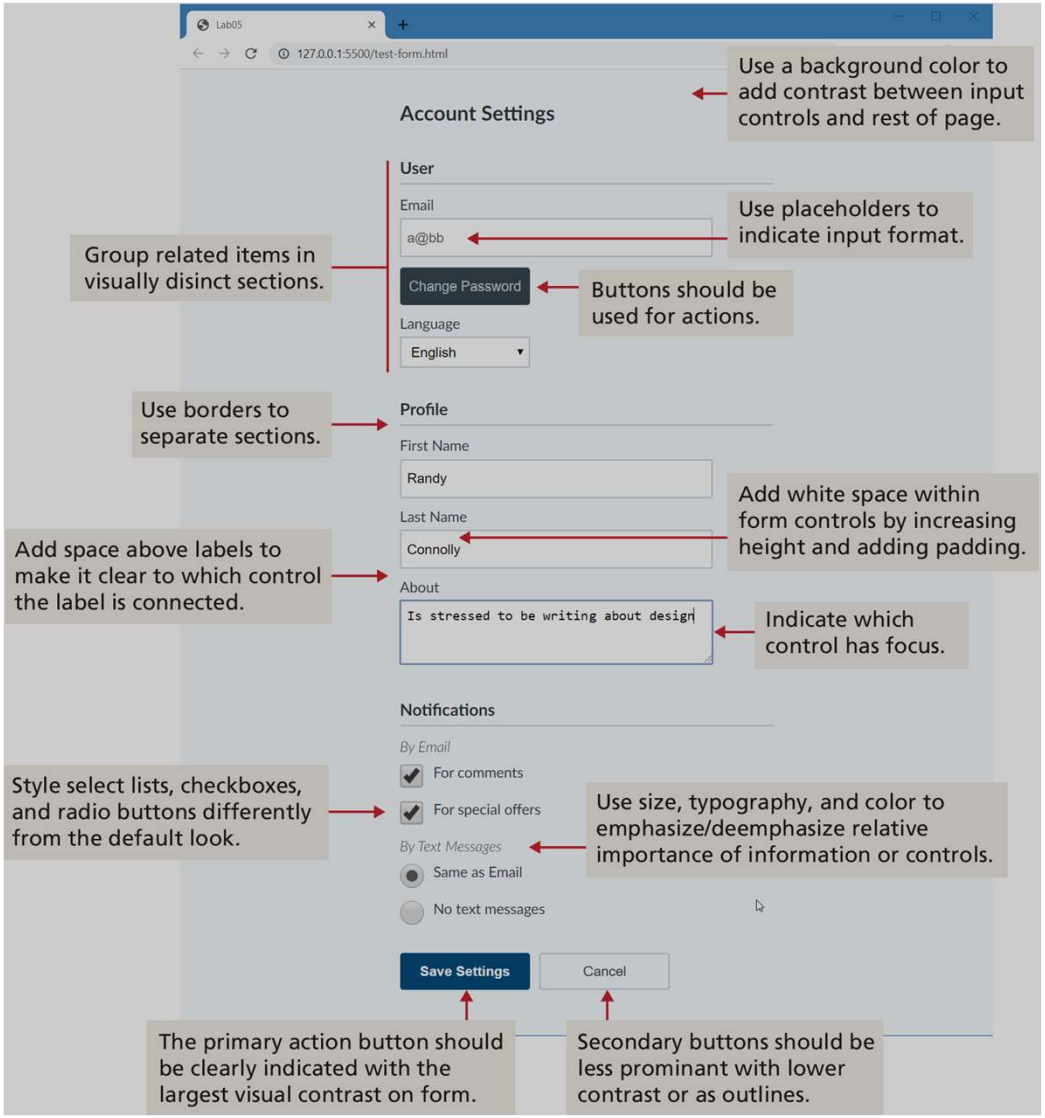
Working with labels



Form Design

A well-designed form communicates to a user that the site values their time and data.





Source code: chapter03/07_style_forms

Fundamentals of Web Development

Third Edition by Randy Connolly and Ricardo Hoar

Chapter 2

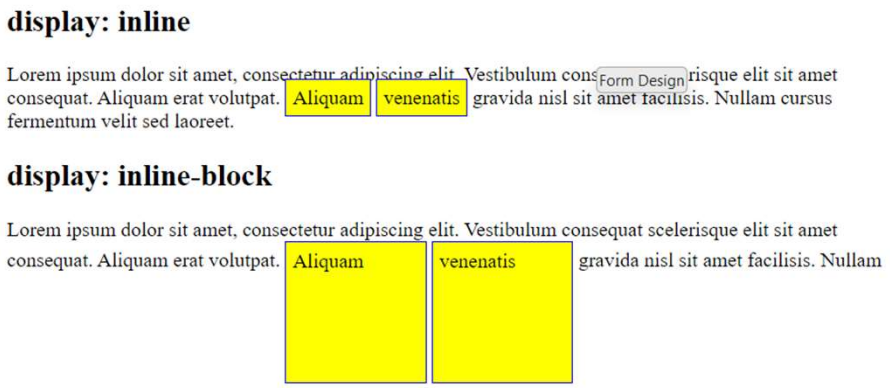
CSS Part 2: Layout

In this chapter you will learn . . .

- Approaches to CSS layout using CSS flexbox and grid models
- What responsive web design is and how to construct responsive designs

Older Approaches to CSS Layout

- The display property in CSS provides a mechanism for the developer to change an element to block, inline, or inline-block.



- For the first 20 years of CSS, designers had to “hack” together multi column layouts using **floats** and/or **positioning**. If you have to support legacy CSS you need to know the basics of floats and positioning.
- Newer approaches (flexbox and grid display modes) make column layouts much easier to implement.

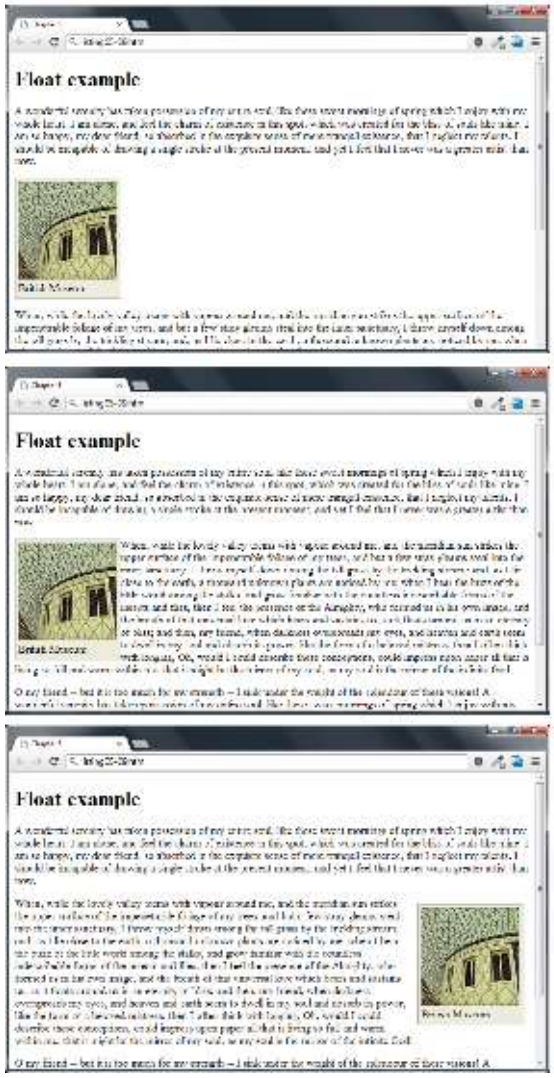
Floating Elements

It is possible to displace an element out of its position in the normal flow via the CSS **float property**.

An element can be floated to the **left** or floated to the **right** and content will wrap around the floated element

A floated block-level element should have a **width** specified

The **clear property** is used to prevent other elements in the same container to touch the current element.




```
<h1>Float example</h1>
<p>A wonderful serenity has taken ...</p>
<figure>
   <img src="images/828.jpg" alt="..." />
   <figcaption>British Museum</figcaption>
</figure>
<p>When, while the lovely valley …</p>
```

```
figure {
   border: 1pt solid #A8A8A8;
   background-color: #EDEDDD;
   margin: 0;
   padding: 5px;
   width: 150px;
}
```

Notice that a floated block-level element **should** have a width specified.

```
figure {
   …
   width: 150px;
   float: left;
}
```

```
figure {
   …
   width: 150px;
   float: right;
   margin: 10px;
}
```

Pearson

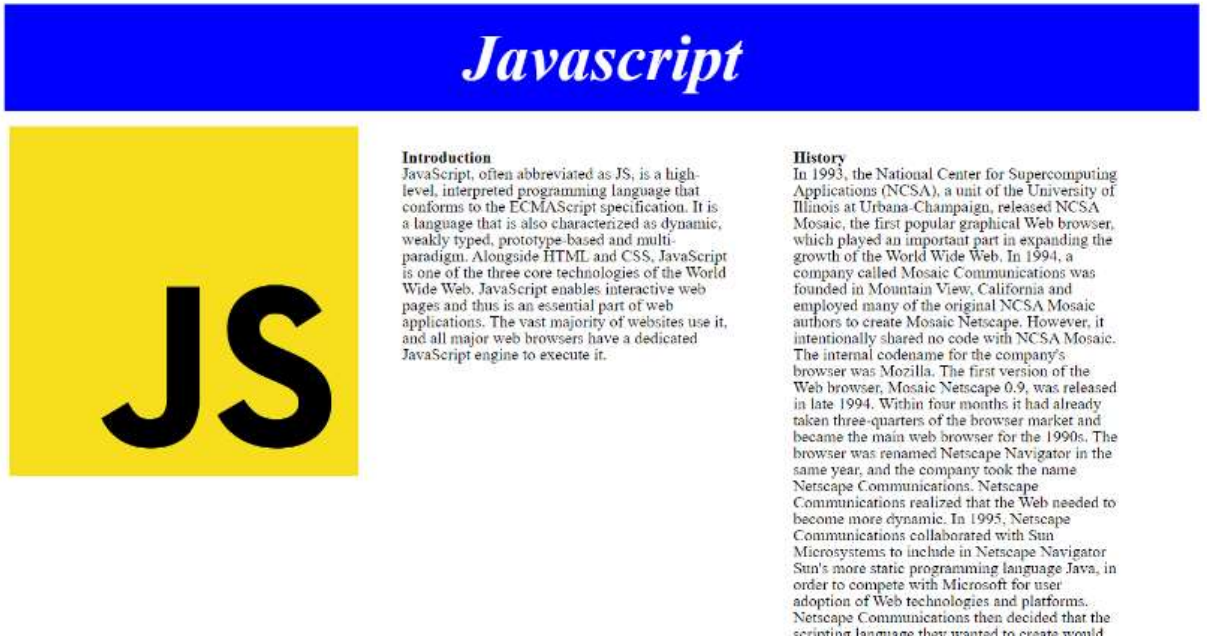
Float example

```
<body>
    <header>
      <h1 id="site-title">Javascript</h1>
    </header>
    <main>
      <img src="javascript.png" alt="javascript photo">
      <article>
        <h2>Introduction</h2>
        <p> JavaScript, often abbreviated …..</p>
      </article>
      <article>
        <h2>History</h2>
        <p>In 1993, the National ….</p>
      </article>
      <article class="clear-lf">
        <h2>Features</h2>
          …
      </article>
      ….
    </main>
    <footer>
      <article>
        <hr>
        <p>….</p>
      </article>
    </footer>
</body>
```

```
body {
  padding: 3%;
}

img {
  float: left;
  width: 29%;
  margin: 10px;
}
```

```
article {
  float: left;
  width: 27%;
  padding: 2em;
}

.clear-lf {
  clear: left;
}
```

```
#site-title {
  font-style: oblique;
  font-size: 4rem;
  text-align: center;
  background-color: blue;
  color: white;
  margin: 5px;
  padding: 20px;
}
```



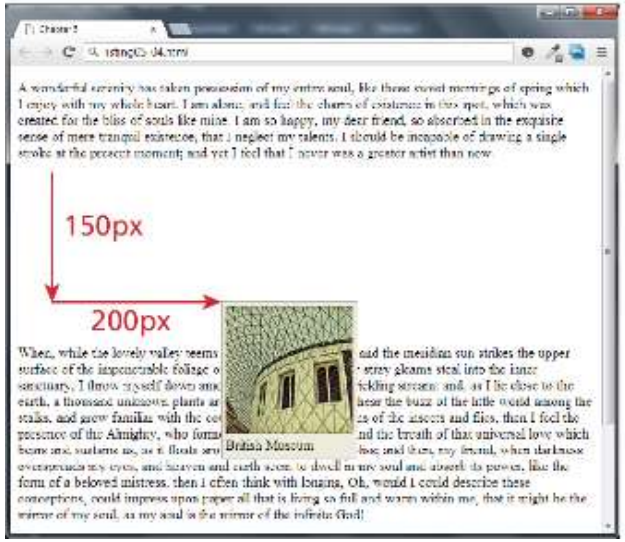
Positioning Elements

The **position** property is used to specify the type of positioning. The **left**, **right**, **top**, and **bottom** properties are used to indicate the distance the element will move (top and left properties).

- **absolute** The element is removed from normal flow and positioned in relation to its first parent that has a position other than static. If no such element is found, the containing block is <html>.
- **fixed** The element is fixed in a specific position in the window even when the document is scrolled.
- **relative** The element is moved relative to where it would be in the normal flow.
- **static** The element is positioned according to the normal flow. **This is the default**.
- **sticky** The element is positioned according to the normal flow, and then offset relative to its nearest scrolling ancestor. This is used to allow an item to scroll, and then stay fixed in position once its scroll position is reached.

Relative Positioning


```
<p>A wonderful serenity has taken ... </p>

<figure>
   <img src="images/828.jpg" alt="British Museum" />
   <figcaption>British Museum</figcaption>
</figure>

<p>When, while the lovely valley …
```

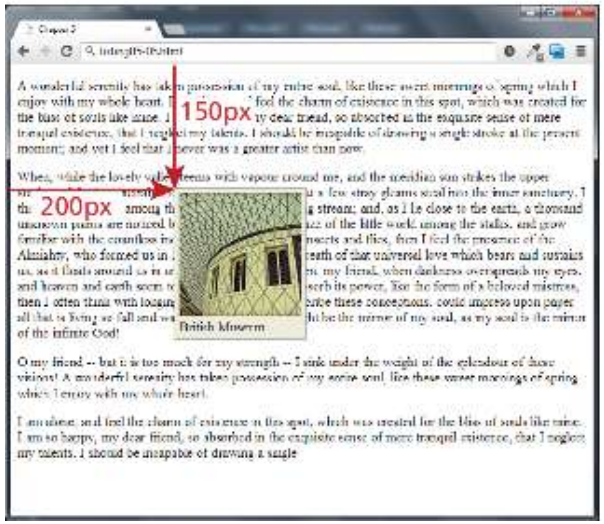



```
figure {
   border: 1pt solid #A8A8A8;
   background-color: #EDEDDD;
   padding: 5px;
   width: 150px;
   position: relative;
   top: 150px;
   left: 200px;
}
```

Pearson

Absolute positioning


```
<p>A wonderful serenity has taken possession of my …

<figure>
   <img src="images/828.jpg" alt="British Museum" />
   <figcaption>British Museum</figcaption>
</figure>

<p>When, while the lovely valley …
```

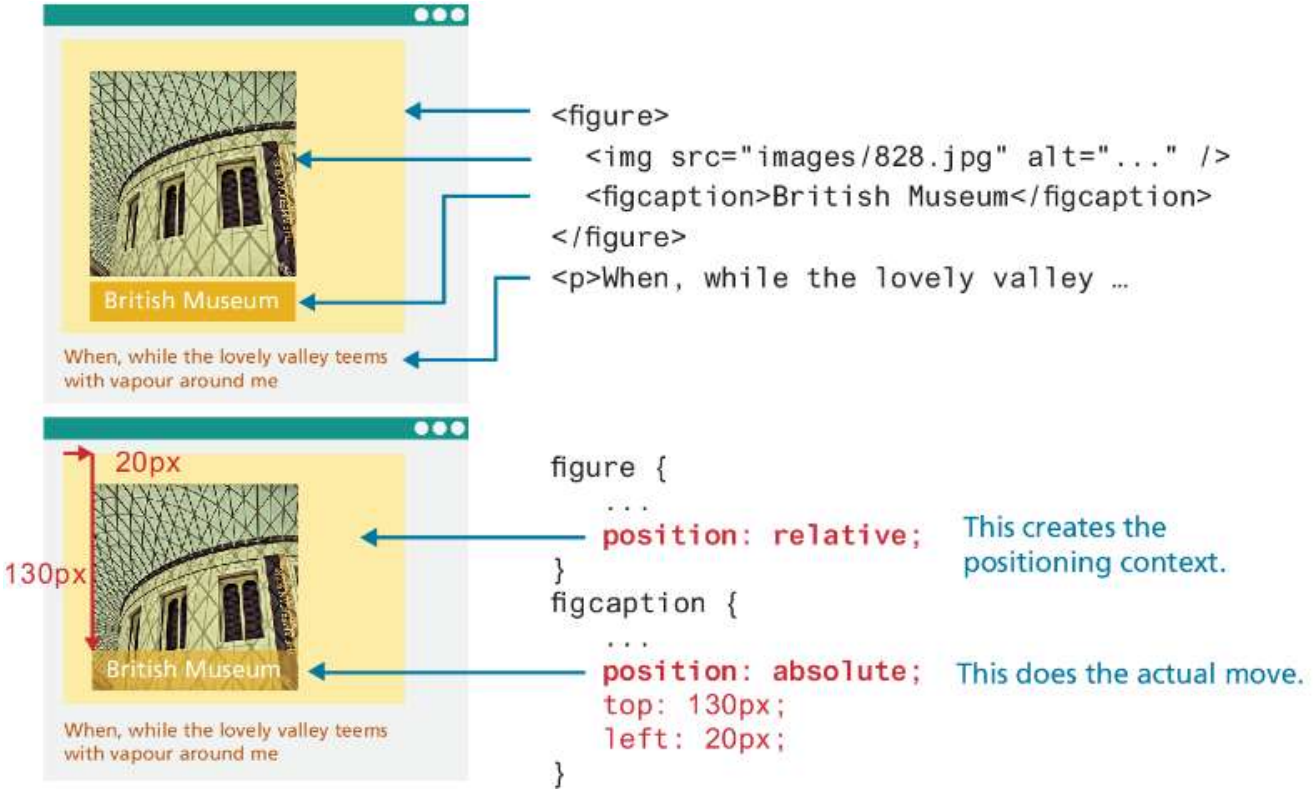



```
figure {
   margin: 0;
   border: 1pt solid #A8A8A8;
   background-color: #EDEDDD;
   padding: 5px;
   width: 150px;
   position: absolute;
   top: 150px;
   left: 200px;
}
```

Overlapping and Hiding Elements

One of the more common design tasks with CSS is to place two elements on top of each other, or to selectively hide and display elements.

In such case, relative positioning is used to create the **positioning context** for a subsequent absolute positioning move.



Overlapping and Hiding Elements (ii)

Consider an image that is the same size as the underlying one is placed on top of the other image using absolute positioning.

You can hide this image using the **display** property

```
<figure>
  <img src="828.jpg" alt="..." />
  <figcaption>British Museum</figcaption>
  <img src="new-banner.png" class="overlayed" />
</figure>
```

new-banner.png

Transparent area

```
.overlayed {
   position: absolute;
   top: 10px;
   left: 10px;
}
```


```
.hide {
   display: none;
}
```

This is the preferred way to hide: by adding this additional class to the element.

```
<img src="new-banner.png" class="overlayed hide" />
```

Pearson

Flexbox Layout


```
.media-image {
   float: left;
   margin-right: 10px;
}
.media-body {
   margin-left: 160px;
}
```

```
<div class="media">
   <img class="media-image" src="c.jpg" >
   <div class="media-body">
      <h2>Fall in Calgary</h2>
      <p>Nunc nec fermentum dolor...</p>
      <p>Mauris porta arcu id...</p>
      <p>Phasellus vel felis purus...</p>
   </div>
</div>
```

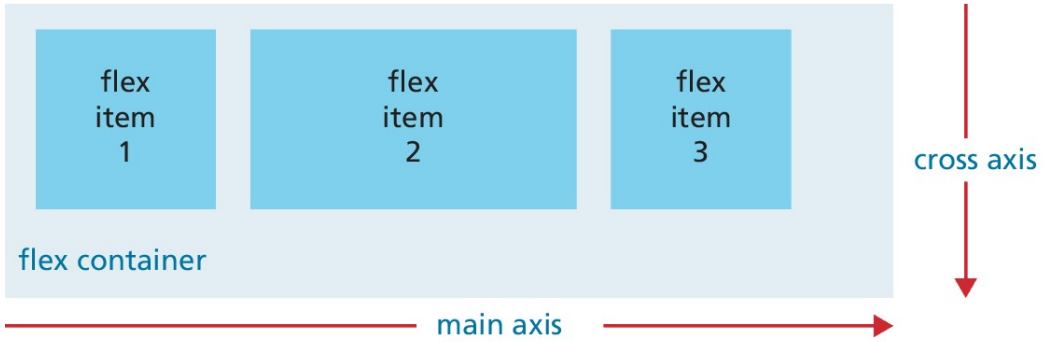
Prior to flexbox, one would create such a layout within a container using floats plus margins. The problem with this approach is that margins needed to be in pixels and had to exactly match image size. If image size changed (or you wanted same kind of style elsewhere), you had to modify the style.


```
.media {
   display: flex;
   align-items: flex-start;
}
.media-image {
   margin-right: 1em;
}
```

Using flexbox, we now have a much more generalized **(and thus reusable)** style.

Flex containers and items

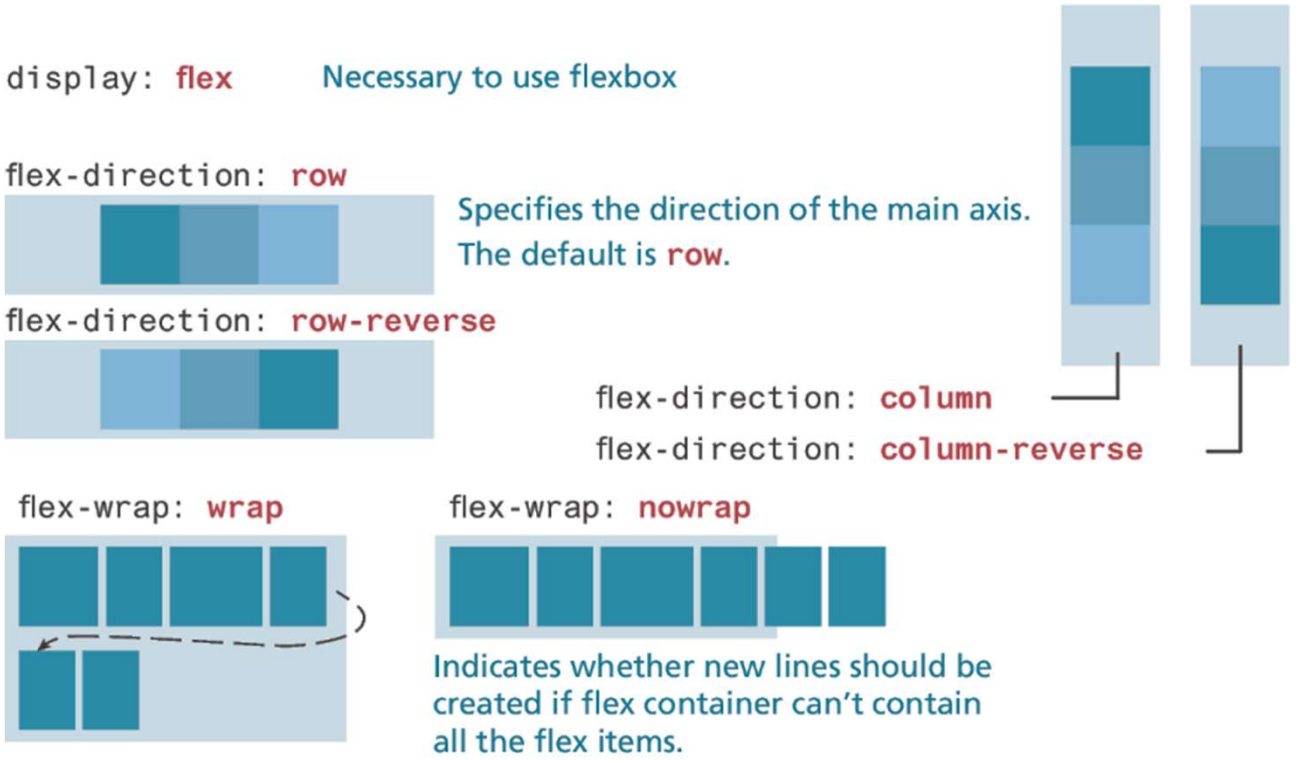
There are two places in which you will be assigning flexbox properties: the **flex container** and the **flex items** within the container.



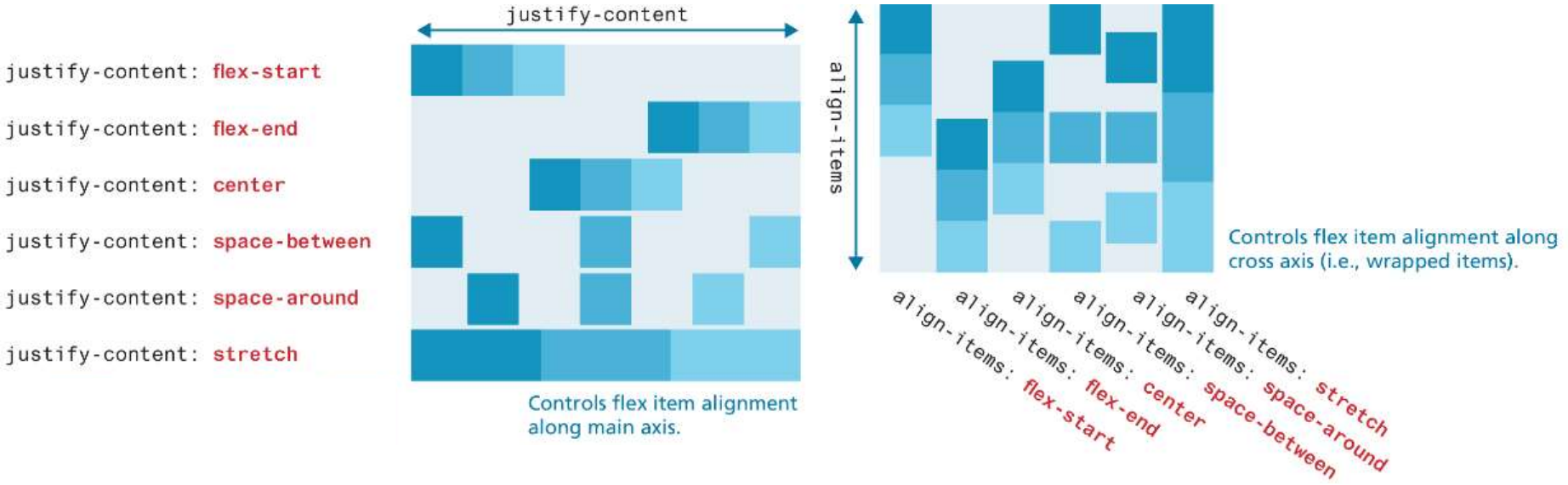

```
<div class="container">
  <div>flex item 1</div>
  <span>flex item 2</span>
  <div>flex item 3</div>
</div>
```

Both block-level and inline elements will become flex items.

The flexbox container properties



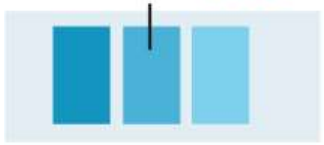
The flexbox container properties (ii)



The flexbox item (child) properties

The flex-basis property determines the initial size of the flex item before the remaining space is distributed.

flex-basis: auto



The default auto value means that the size is determined by the width and height.

flex-basis: 200px

You can specify a width using px, %, or other measurement units.

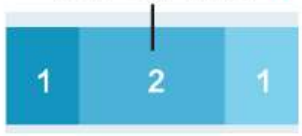
flex-grow: 2

Defines the growth factor of an element relative to the other items.

width=*n*

width=*n* × 2

flex-shrink: 2



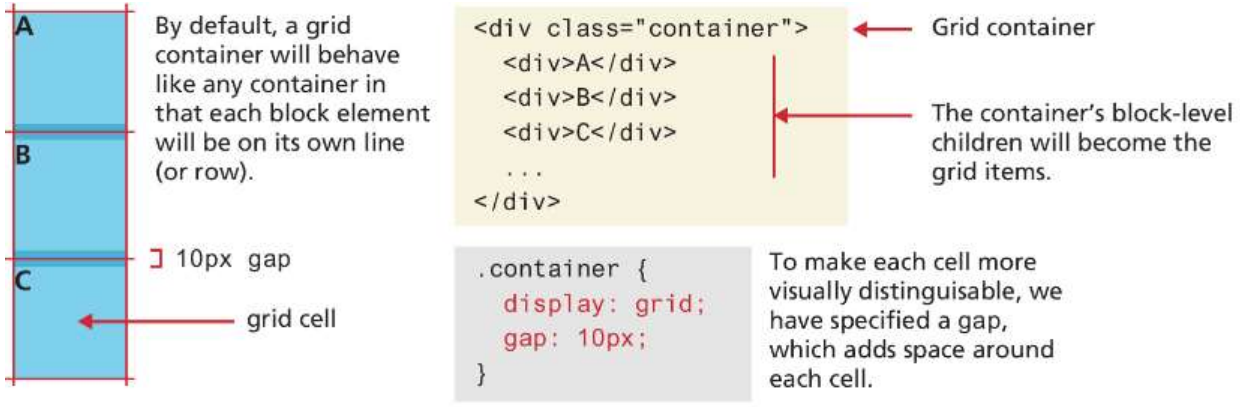
Defines how much an item will shrink when not enough space in container.

Pearson

Grid Layout

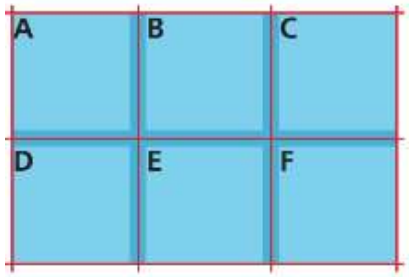
Grid layout is adjustable, powerful, and, compared to floats, positioning, and even flexbox, is relatively easy to learn and use!

- Each block-level child in a parent container whose **display** property is set to **grid** will be automatically placed into a grid cell



Specifying Grid Structure

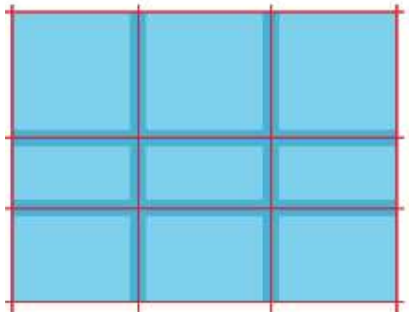
grid-template-columns is used for adding columns by specifying each column's width using the **fr** unit.




```
.container {
  display: grid;
  gap: 10px;
  grid-template-columns: 1fr 1fr 1fr;
}
```

This makes each column equal to an equal fraction of the available space.

You can specify the number of columns per row/line via the `grid-template-column` property.



```
.container {
  ...
  height: 300px;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 80px 30px 50px;
}
```

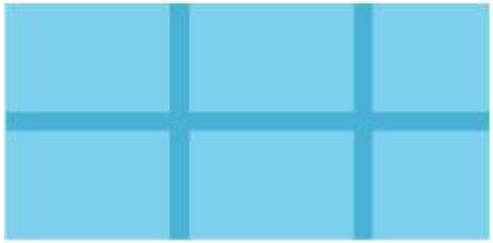
While you often do not need to set a row height, you can make rows taller than their actual content.

Specifying column widths

Column widths can be specified

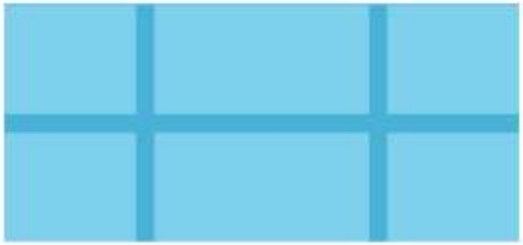
The CSS **repeat()** function provides a way to specify repeating patterns of columns.

```
grid-template-columns: 70px 70px 45px;
```



Each column can have its own unique width value, in px, %, em, etc.

```
grid-template-columns: 50px auto 50px;
```



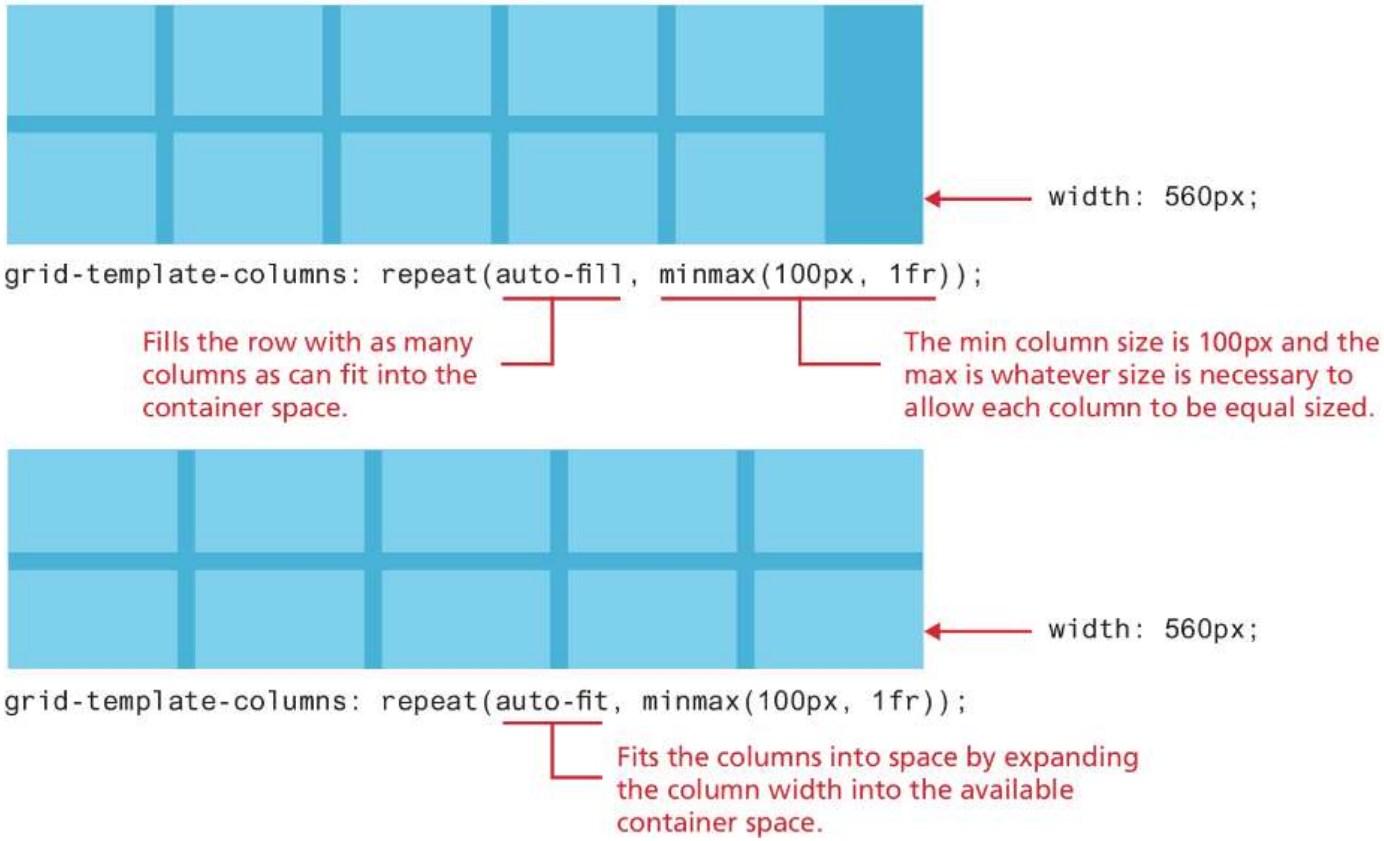
An auto value indicates the width will fill the remaining space.

```
grid-template-columns: repeat(3, 1fr);
```

The repeat function can be used to defining a repeating pattern.

```
grid-template-columns: repeat(2, 30px 50px) 70px;
```

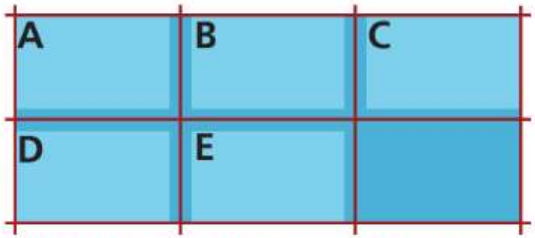
Specifying column widths (ii)



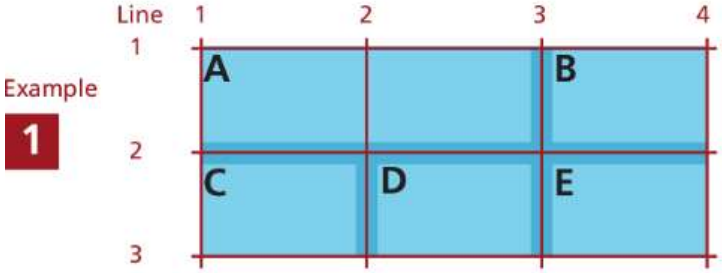
Explicit Grid Placement Example 1

```
<div class="container">
  <div class="a">A</div>
  <div class="b">B</div>
  <div class="c">C</div>
  <div class="d">D</div>
  <div class="e">E</div>
</div>
```

```
.container {
  display: grid;
  gap: 10px;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(2,200px);
}
```



With implicit layout, grid items are placed automatically.




```
.a {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

The start and end numbers refer to the line number not the column number.

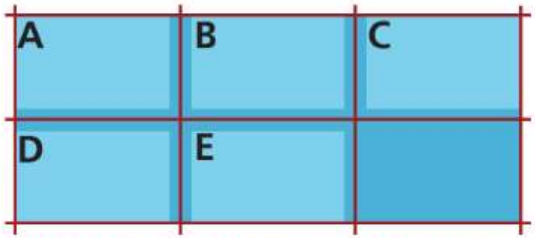
The same effect also possible using either of the following:

```
grid-column: 1 / 3;
grid-column: 1 / span 2;
```

Explicit Grid Placement Example 2

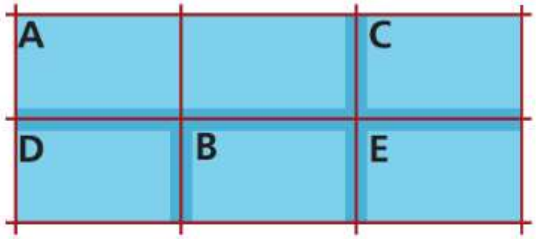
```
<div class="container">
  <div class="a">A</div>
  <div class="b">B</div>
  <div class="c">C</div>
  <div class="d">D</div>
  <div class="e">E</div>
</div>
```

```
.container {
  display: grid;
  gap: 10px;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(2,200px);
}
```



With implicit layout, grid items are placed automatically.

2

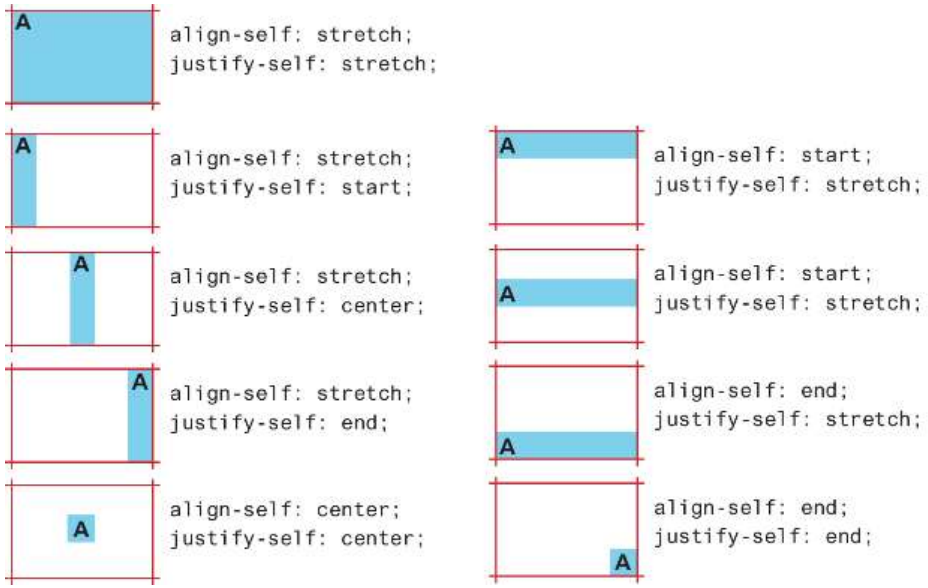



```
.b {
  grid-row: 2;
  grid-column: 2;
}
```

Grid cells can be placed into any row and column.

Cell properties

- **align-self and justify-self** control the cell content's horizontal and vertical alignment within its grid container.

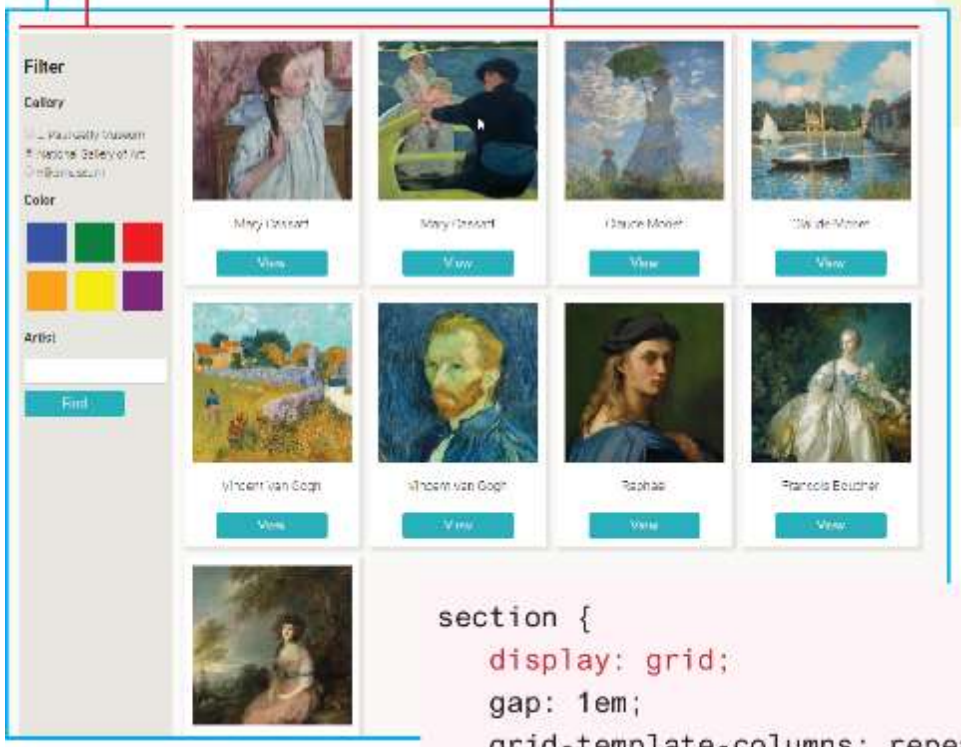


- You can similarly control cell alignment within a grid container using **align-items** and **justify-items**

Nested Grid

```
main {
   display: grid;
   grid-template-columns: 1fr 4fr;
}
```

```
<main>
  <aside>
    <h2>Filter</h2>
    ...
  </aside>
  <section>
    <div class="card">...
    <div class="card">...
    ...
  </section>
</main>
```

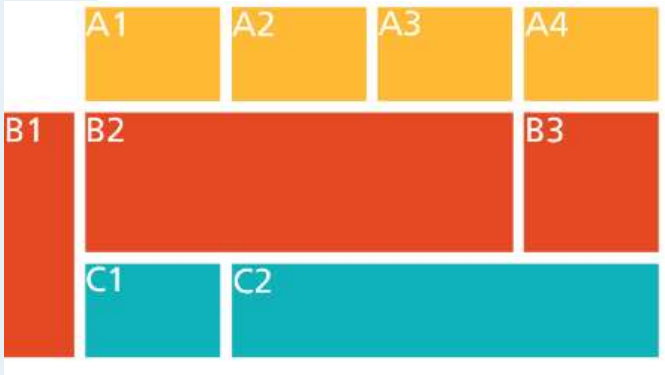



```
section {
   display: grid;
   gap: 1em;
   grid-template-columns: repeat(auto-fit, 220px);
}
```

Pearson

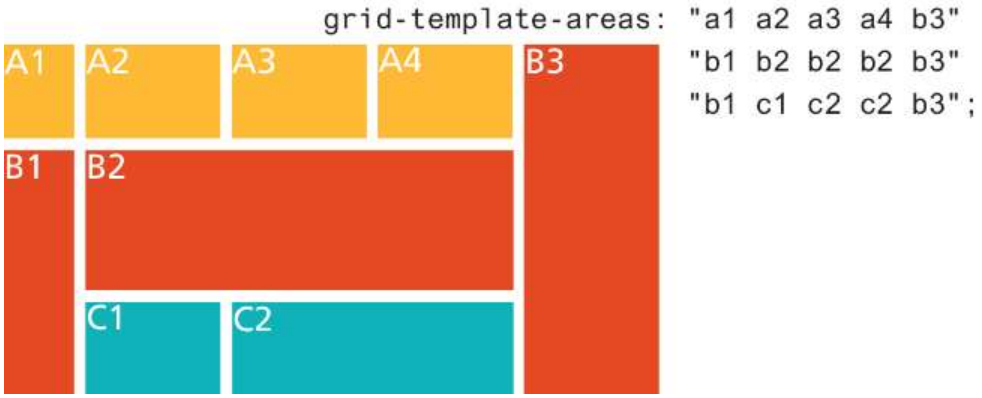
Grid Named Areas

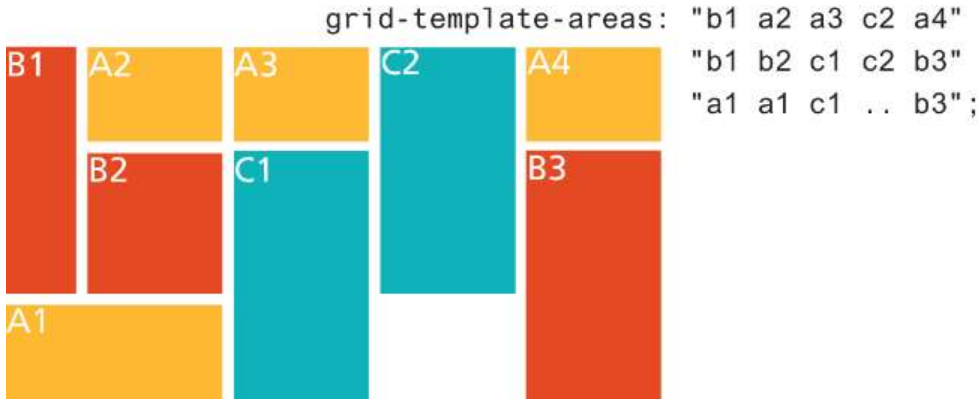
```
<style>
.container {
  grid-gap: 10px;
  display: grid;
  grid-template-rows: 100px 150px 100px;
  grid-template-columns: 75px 1fr 1fr 1fr 1fr;
  grid-template-areas: ". a1 a2 a3 a4"
                       "b1 b2 b2 b2 b3"
                       "b1 c1 c2 c2 c2";
}
.a1 { grid-area: a1; }
.a2 { grid-area: a2; }
.a3 { grid-area: a3; }
.a4 { grid-area: a4; }
.b1 { grid-area: b1; }
.b2 { grid-area: b2; }
.b3 { grid-area: b3; }
.c1 { grid-area: c1; }
.c2 { grid-area: c2; }
</style>


<section class="container">
  <div class="yellow a1">A1</div>
  <div class="yellow a2">A2</div>
  <div class="yellow a3">A3</div>
  <div class="yellow a4">A4</div>
  <div class="orange b1">B1</div>
  <div class="orange b2">B2</div>
  <div class="orange b3">B3</div>
  <div class="cyan c1">C1</div>
  <div class="cyan c2">C2</div>
</section>
```



LISTING 7.2 Using grid areas

Grid Areas (ii)

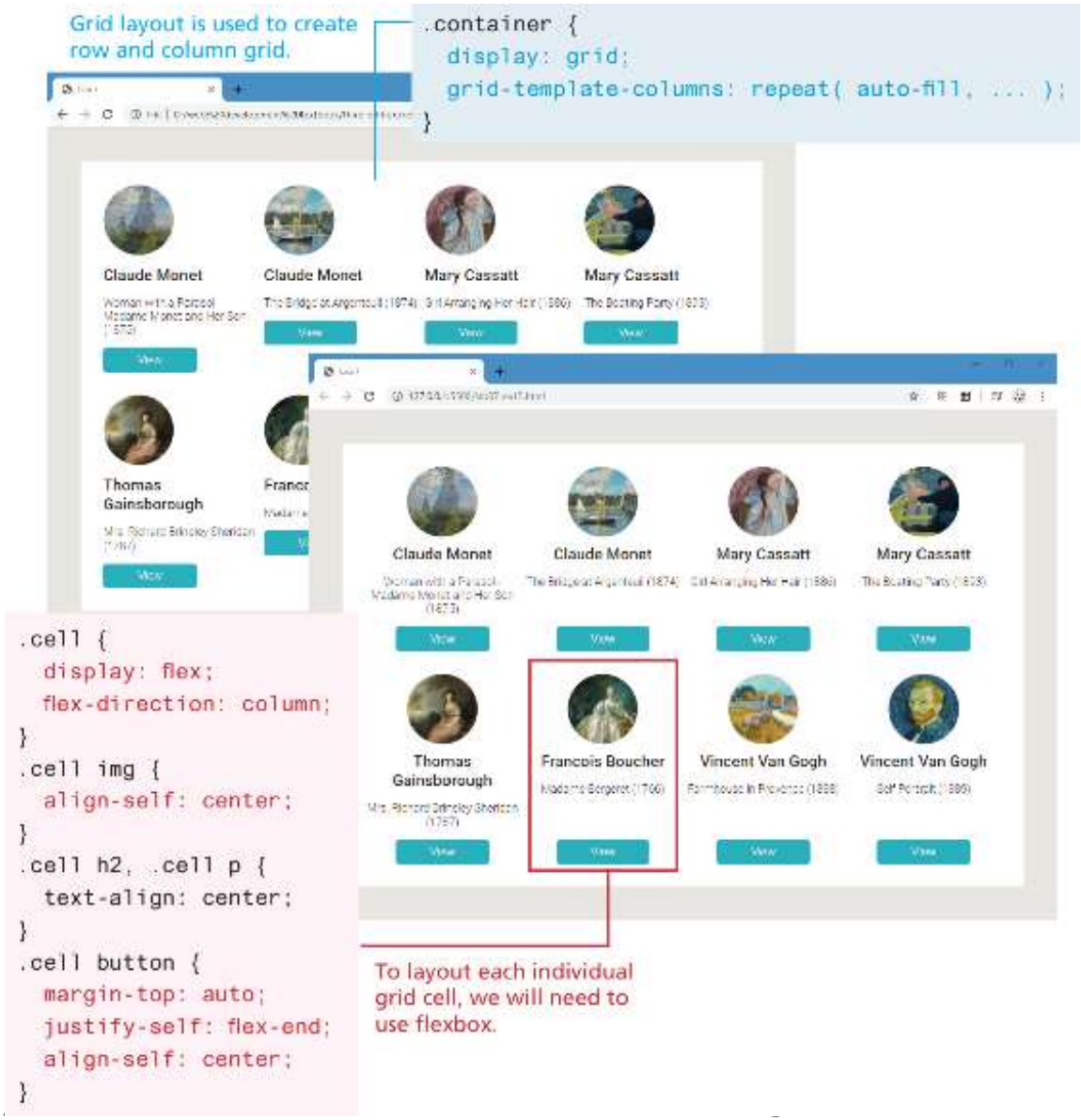




LISTING 7.2 Using grid areas

Grid and Flexbox Together

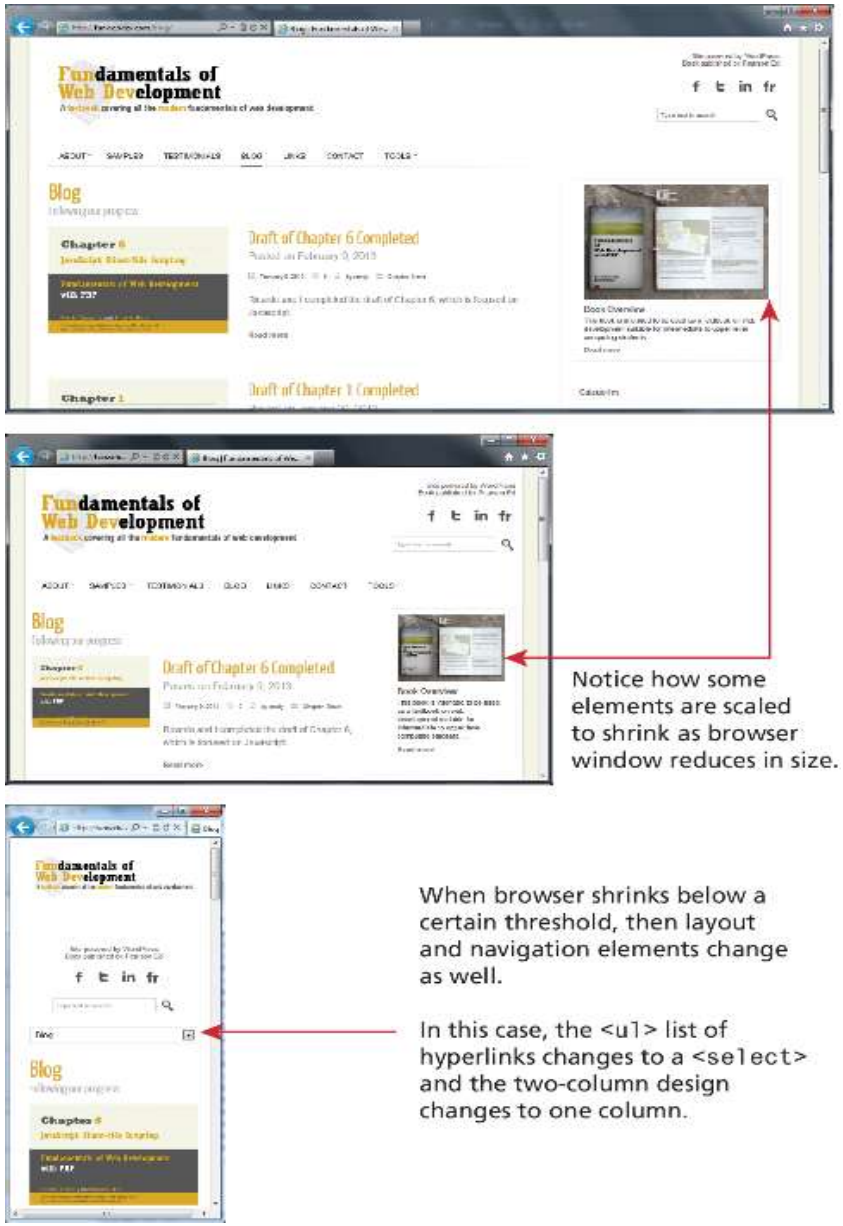
- **grid** and **flexbox** each have their strengths and these strengths can be combined
- **grid** layout is ideal for constructing the layout structure of your page
- **flexbox** is ideal for laying out the contents of a grid cell.



Responsive Design

In a **responsive design**, the page "responds" to changes in the browser size that go beyond simple percentage scaling of widths.

- smaller images will be served and
- navigation elements will be replaced as the browser window shrinks



Mobile First Design

NOTE

Mobile-first design suggests that the first step in the design and implementation of a new website should be the design and development of its mobile version (rather than as an afterthought as is often the case).

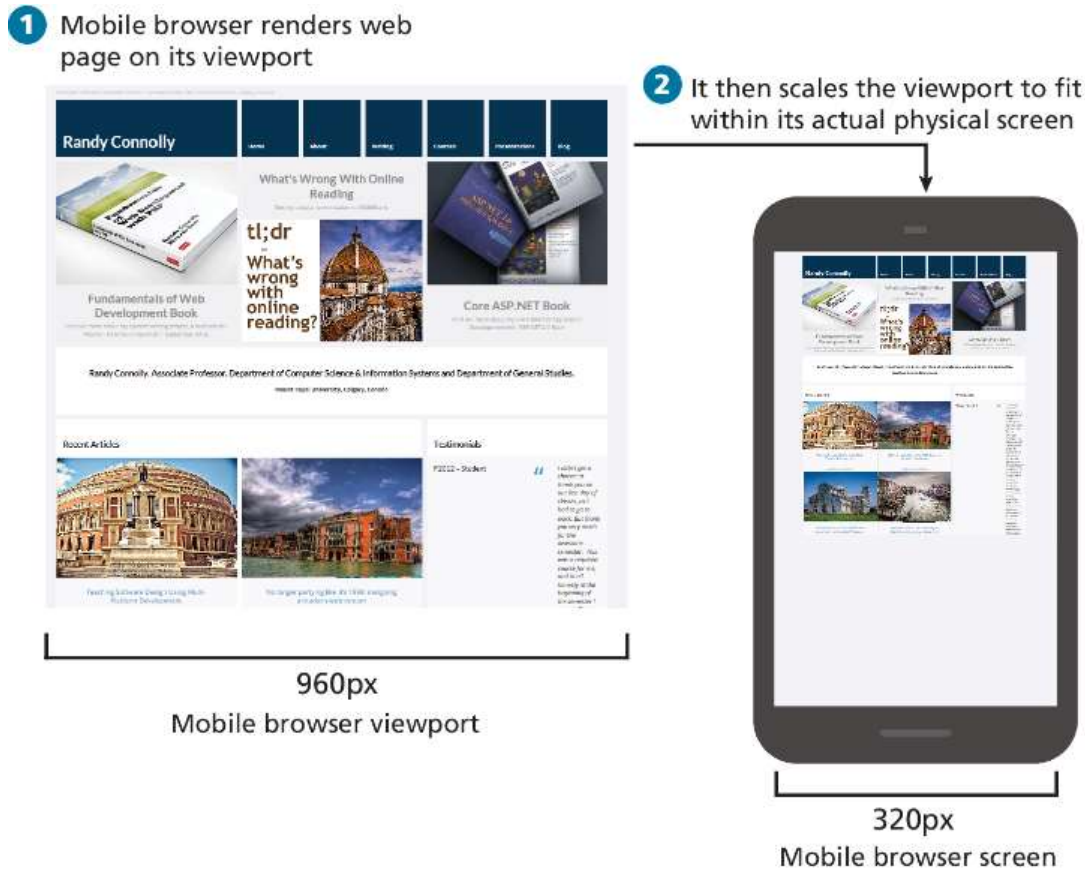
The rationale for the mobile-first approach lies not only in the increasingly larger audience whose principal technology for accessing websites is a smaller device such as a phone or a tablet, focusing first on the mobile platform also forces the designers and site architects to focus on the most important component of any site: the content.

Viewports

The browser's **viewport** is the part of the browser window that displays web content.

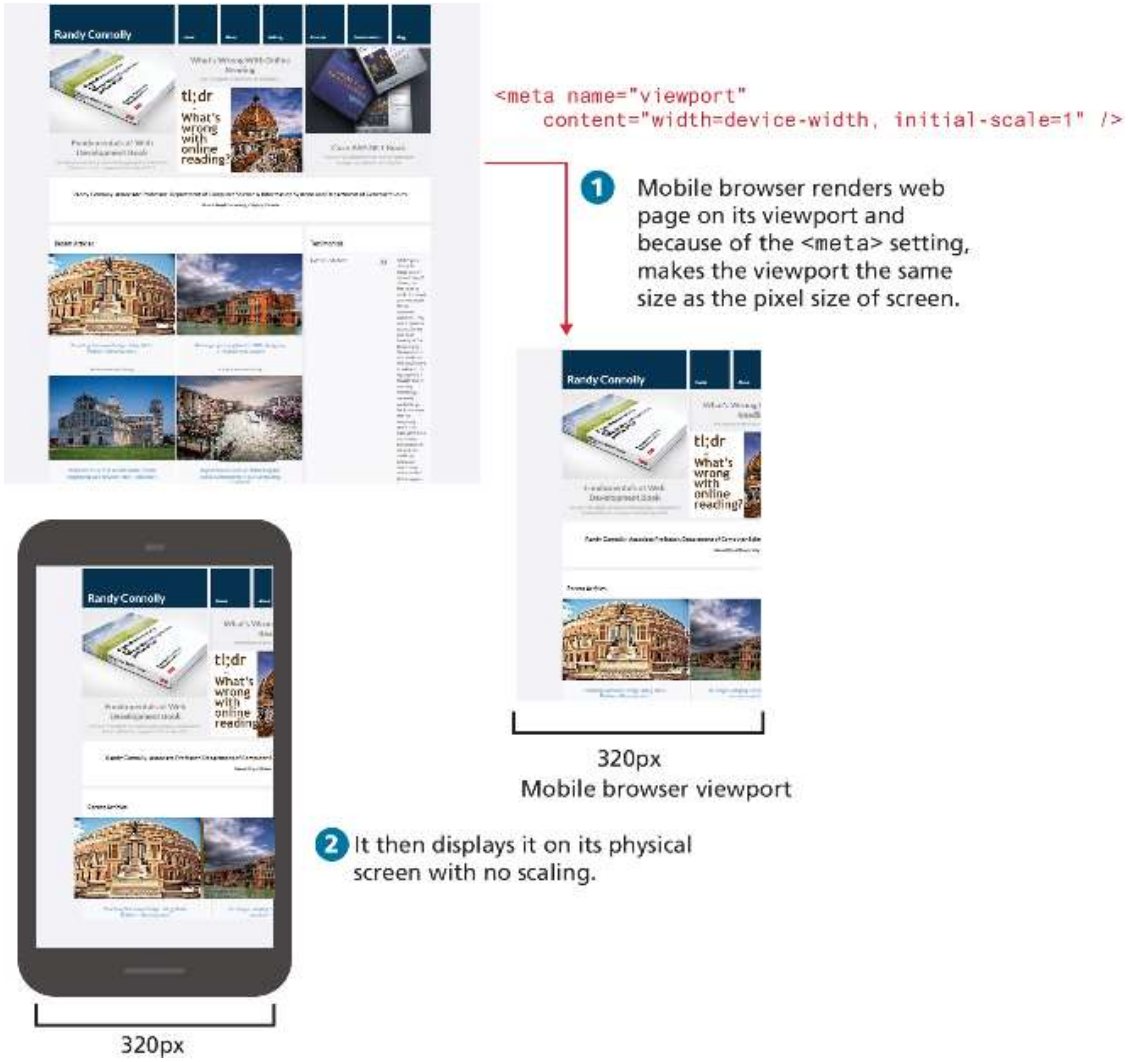
Mobile browsers will by default scale a web page down to fit the width of the screen.

Generally, results in a viewing experience that works but is very difficult to read and use.



Setting Viewports

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



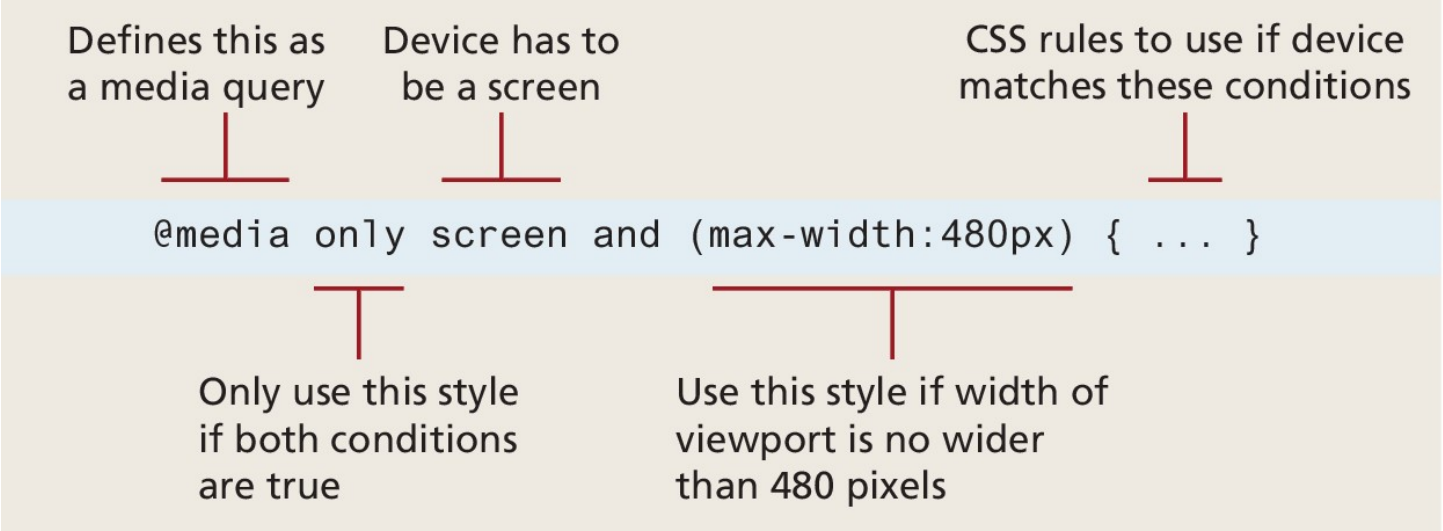
Fig, Setting the Viewport

The **width** attribute controls the size of the viewport, while **initial-scale** sets the zoom level.

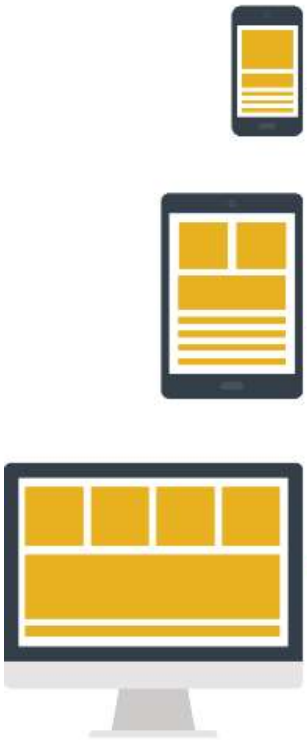
Media Queries

CSS media queries are a way to apply style rules based on the medium that is displaying the file

Contemporary responsive sites will typically provide CSS rules for phone displays first, then tablets, then desktop monitors, an approach called **progressive enhancement**



Media queries in action



styles.css

```
/* CSS rules for phones */
@media only screen and (max-width:480px)
{
  #slider-image { max-width: 100%; }
  #flash-ad { display: none; }
    ...
}

/* CSS rules for tablets */
@media only screen and (min-width: 481px)
    and (max-width: 768px)
{
    ...
}

/* CSS rules for desktops */
@media only screen and (min-width: 769px)
{
    ...
}
```

Instead of having all the rules in a single file, we can put them in separate files and add media queries to <link> elements.

```
<link rel="stylesheet" href="mobile.css"  media="screen and (max-width:480px)" />
<link rel="stylesheet" href="tablet.css"
    media="screen and (min-width:481px) and (max-width:768px)" />
<link rel="stylesheet" href="desktop.css" media="screen and (min-width:769px)" />
```

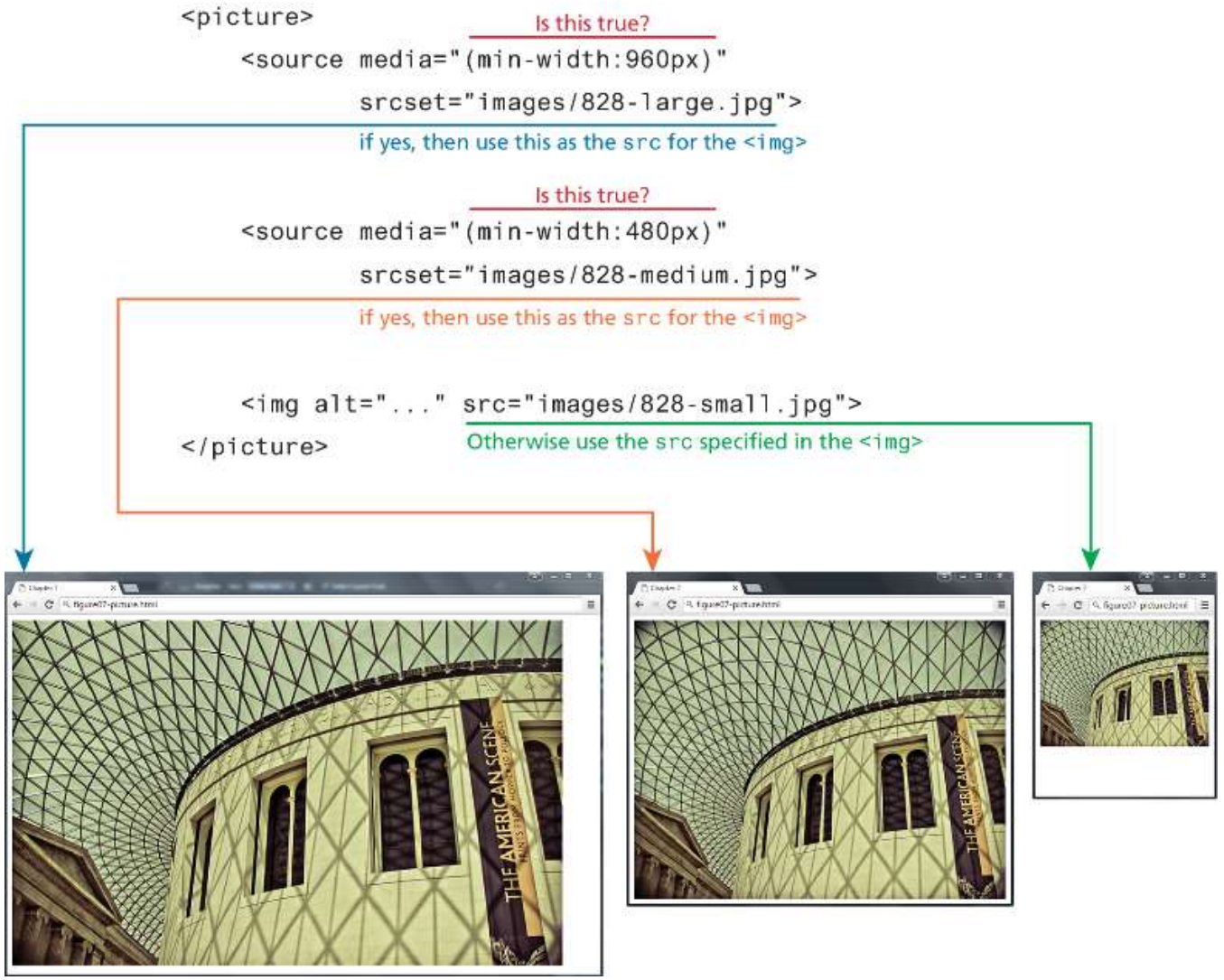
Pearson

The <picture> element

Making images scale in size is straightforward, but does not change the downloaded size of the image

```
img {
    max-width: 100%;
}
```

HTML5.1 defines the new <picture> element that lets the designer specify multiple <img> elements. The browser determines which <img> to use based on the viewport size.



Pearson

Bootstrap (optional)

A quick Introduction

What is a CSS Framework?

A **CSS framework** is a set of CSS classes or other software tools that make it easier to use and work with CSS.

- Early CSS frameworks became popular as they helped create complex grid-based layouts.
- More sophisticated subsequent CSS Frameworks such as
  - Bootstrap (**https://getbootstrap.com/**)
  - Tailwind
  - Material and
  - Foundation (**https://get.foundation/**)
- They provide much more than a grid system; they provide a comprehensive set of predefined CSS classes, which makes it easier to construct a consistent and attractive web interface.

Bootstrap 5

You can add Bootstrap to any app by adding the following code to the top of your HTML (CDN : Content Delivery Network):

```
<!-- Latest compiled bootstrap 5 CSS -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/css/bootstrap.min.css" rel="stylesh
eet">

<!-- Latest compiled Bootstrap JavaScript -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/js/bootstrap.bundle.min.js"></scri
pt>
```

Define the top-level container type

Bootstrap 5 is designed to be responsive to mobile devices. Mobile-first styles are part of the core framework. 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">
```

Include all the body elements in a single div element and add the container class to the div:

1.The `.container` class provides a responsive **fixed width container** (for each breakpoint)
2.The `.container-fluid` class provides a **full width container**, spanning the entire width of the viewport

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

   ……content goes here
 </div>
</body>
```

Fig. class .*container* width and breakpoints

	Extra small <576px **xs**	Small ≥576px **sm**	Medium ≥768px **md**	Large ≥992px **lg**	Extra Large ≥1200px **xl**	XXL ≥1400px **xxl**
max-width	100%	540px	720px	960px	1140px	1320px

Container style

Spacing (margins and padding) :

By default, containers have left and right padding, with no top or bottom padding. Therefore, we often use spacing utilities, such as extra padding and margins to make them look even better. Spacing have 6 predefined levels. For example, `.pt-5` means "add a large top padding“, (sides: t : top, b: bottom, s: start, e: end, x, y)

```
<div class="container pt-5"></div>
```

Here, `.my-5` means "add a large margin on the top and bottom of 3rem“. `.p-5` means "add a large padding of 3rem":

```
<div class="container p-5 my-5"></div>
```

Adding a Nav bar

- Classes navbar, container, navbar-brand, navbar-nav, nav-item and nav-link are used to specify the role of the element.
- navbar-expand-md defines the responsive behavior: The navbar expands at the L breakpoint.
- Classes bg-dark, navbar-dark specify styles.

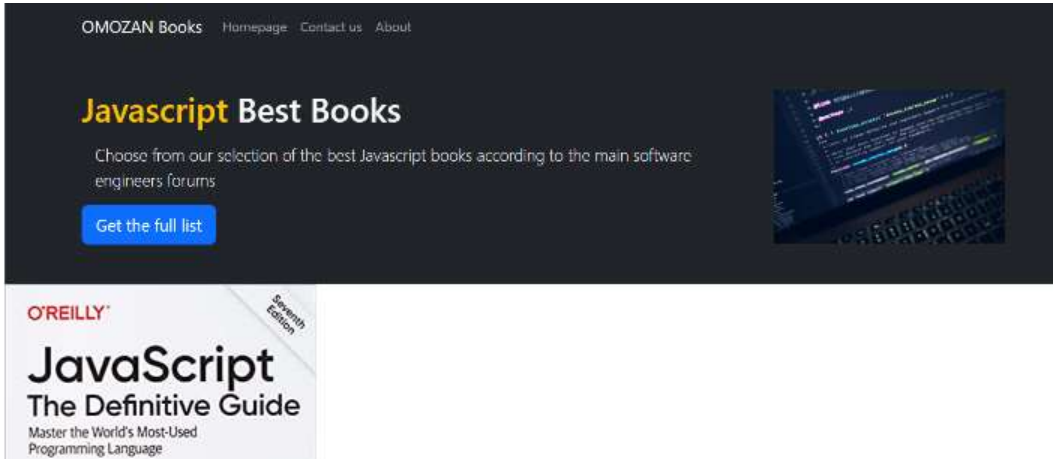
```
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
  <div class="container">
    <a href="#" class="navbar-brand">Best Javascript Books</a>
    <div class="collapse navbar-collapse">
      <ul class="navbar-nav">
        <li class="nav-item"><a href="#" class="nav-link">Homepage</a></li>
        <li class="nav-item"><a href="#contact-us" class="nav-link">Contact us</a></li>
        <li class="nav-item"><a href="#about" class="nav-link">About</a></li>
      </ul>
    </div>
  </div>
</nav>
```

Add a hamburger menu (toggle)

Add data-bs-toggle="collapse" and a data-bs-target to the element to automatically assign control of one or more collapsible elements. The data-bs-target attribute accepts a CSS selector to apply the collapse to. Be sure to add the class collapse to the collapsible element.

```
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
   <div class="container">
        <a href="#" class="navbar-brand">Best Javascript Books</a>
        <button class="navbar-toggler"
             type="button"
             data-bs-toggle="collapse"
             data-bs-target="#navmenu">
             <span class="navbar-toggler-icon"></span>
       </button>
       <div class="collapse navbar-collapse" id="navmenu">
             <ul class="navbar-nav">
               <li class="nav-item"><a href="#" class="nav-link">Homepage</a></li>
               <li class="nav-item"><a href="#contact-us" class="nav-link">Contact us</a></li>
               <li class="nav-item"><a href="#about" class="nav-link">About</a></li>
             </ul>
       </div>
   </div>
</nav>
```

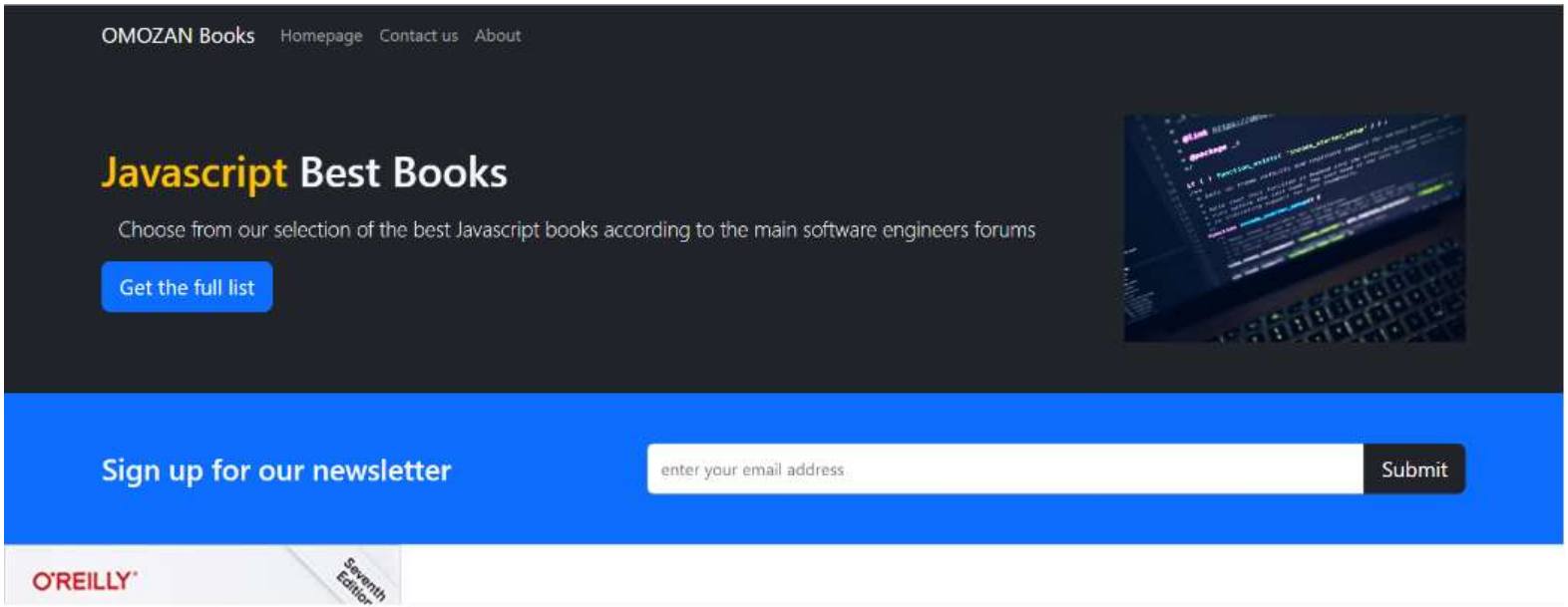
Adding a showcase



- Add paddings: [pt / pb / pl / pr / px / py / p] - 1 to 5 : px-1 is the smallest padding pixels on left and right (same for margins).

```
<section class="bg-dark text-light p-5 text-center text-sm-start">
  <div class="container">
    <div class="d-sm-flex align-items-center justify-content-between">
      <div>
        <h1><span class="text-warning">Javascript</span>  Best Books</h1>
        <p class="m-3 lead">
          Choose from our selection of the best Javascript books according to the main software engineers forums
        </p>
        <button class="btn btn-primary btn-lg">Get the full list</button>
      </div>
      <img class="img-fluid w-25" src="img/site/screen-black.jpg" alt="computer">
    </div>
  </div>
</section>
```

Add a newsletter section

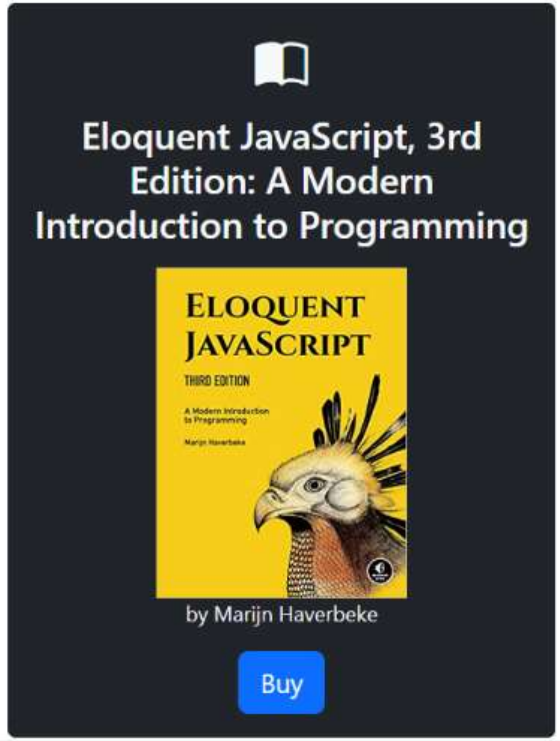



```
<section class="bg-primary text-light p-5">
  <div class="container">
    <div class="d-md-flex justify-content-between align-items-center">
      <h3 class="mb-3 mb-md-0">Sign up for our newsletter</h3>
      <div class="input-group">
        <input type="text" class="form-control" placeholder="enter your email address">
        <button class="btn btn-dark btn-lg" type="button">Submit</button>
      </div>
    </div>
  </div>
</section>
```

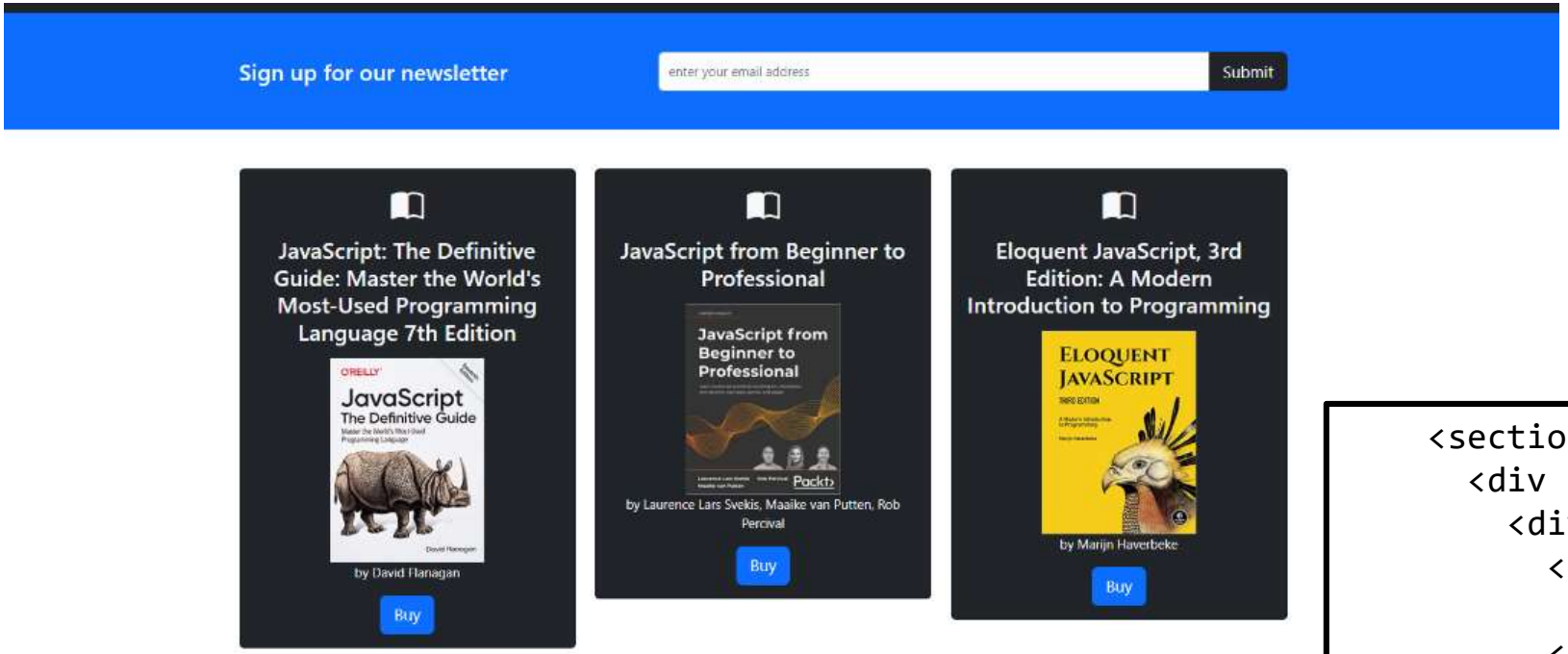
Adding a card

Add a link to Bootstrap icons CDN:

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.3.0/font/bootstrap-icons.css">
```




```
<div class="card h-100 bg-dark text-light">
  <div class="card-body text-center">
    <div class="h1 mb-3">
       <i class="bi bi-book-half"></i>
    </div>
    <h3 class="card-title mb-3">Eloquent JavaScript, 3rd
                Edition: A Modern Introduction to Programming</h3>
    <img class="card-image img-fluid w-50" src="img/books/eloquent-javascript.jpg"
                    alt="Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming">
    <p class="card-text">by Marijn Haverbeke</p>
    <button class="btn btn-primary btn-lg">Buy</button>
  </div>
</div>
```



Creating the grid

```
<section class="p-5">
  <div class="container">
    <div class="row text-center">
      <div class="col-md">
        ……  card goes here
      </div>
      <div class="col-md">
        ……  card goes here
      </div>
      <div class="col-md">
        ……  card goes here
      </div>
    </div>
  </div>
</section>
```

Finalizing the grid

- Add a line break to push cards to a new line:

```
<div class="w-100 p-3"></div>
```