HTML5, CSS3 & Javascript

Chapter 2:
Advanced HTML and CSS

Chapter Objectives

In this chapter, we will discuss:

- Standardizing presentation with HTML and CSS
  - How HTML lays out a web page
  - Additional HTML elements
  - Advanced styling with CSS
  - Semantic tags
  - Forms and validation
  - HTML5 API's

Chapter Concepts

HTML Layout

The Document Object Model

More HTML and CSS

Advanced CSS

Semantic Tags

Forms and Validation

HTML5 API's

Chapter Summary

Normal Flow Layout

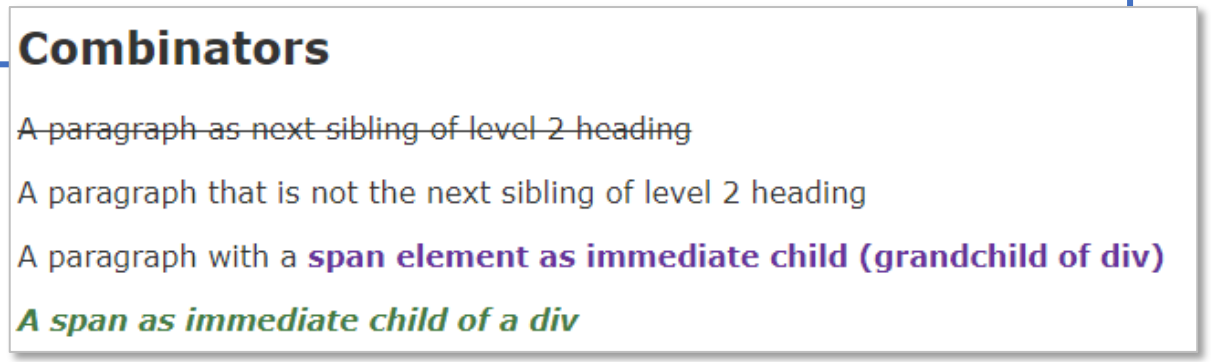
- In the Normal Flow model, the browser positions elements in the order they are seen
- The `display` property determines how elements interact with the normal flow model:

`block`	The element has line breaks before and after. It normally occupies all the available width, with height being set according to contents. Can specify `height` and `width` properties.
`inline`	The element has no line breaks before or after. The `height` and `width` are both set by the contents and cannot be specified.
`inline-block`	Formatted as `inline`, but can specify `height` and `width`.
`none`	The element takes up no space in the page layout. Compare to `visibility: hidden` where the element still takes up space.

The Div and Span Tags

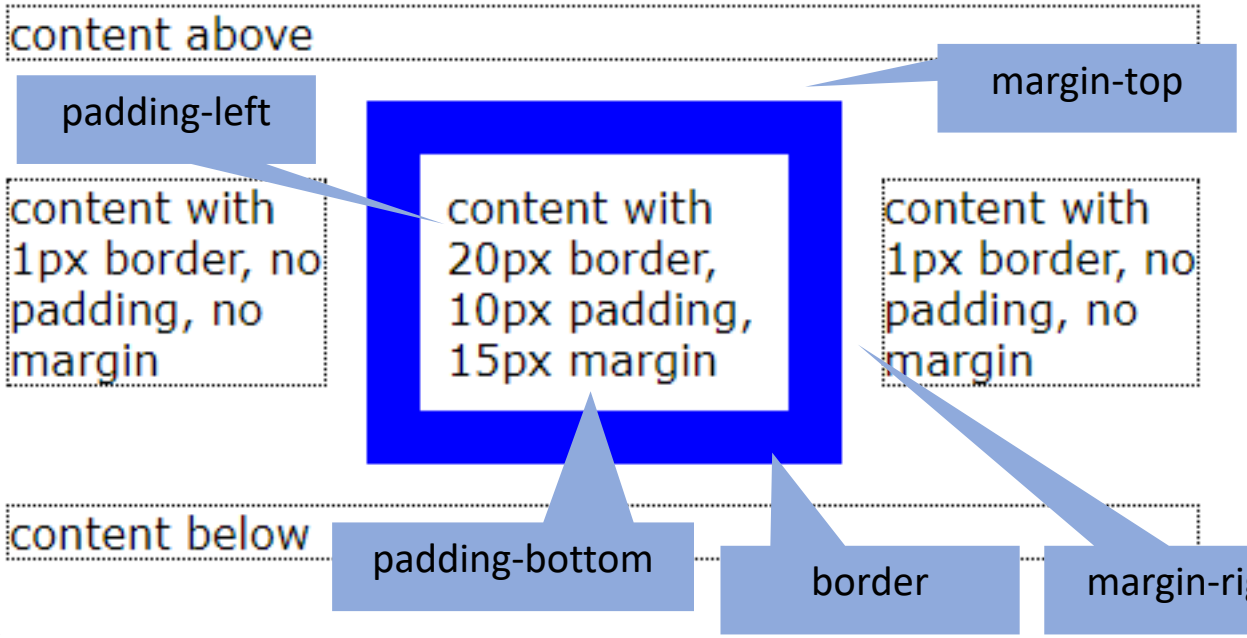
- The `div` element is used to break an HTML document into sections
  - Typically to give it a specific style or allow programmatic access
  - It has no specific thematic meaning and should be an element of last resort
  - By default, `div` elements are `block` elements
- The `span` element is similar, but by default, `span` elements are `inline`

```
<div class="text">
    <h2>Combinators</h2>
    <p>A paragraph as next sibling of level 2 heading</p>
    <p>A paragraph that is not the next sibling of level 2 heading</p>
    <p>A paragraph with a <span>span element as immediate child (grandchild of div)</span></p>
    <span>A span as immediate child of a div</span>
</div>
```



CSS Box Model

- CSS box model allows borders to be added around elements
  - Wraps HTML elements and provides borders, padding, and margins




```
#borderDemo {
    margin: 15px;
    border: 20px solid blue;
    padding: 10px;
}
```

margin, border, and padding are summary properties, allowing all to be set at once

- \
- By default, all sizes (`height`, `width`, or content sizing) apply to the content alone
  - Border and padding are added afterwards
  - Ideal in most cases, but can be hard to line up elements
  - Can change to `box-sizing: border-box`, which includes border and padding in size

Positioning Within Normal Flow

- A detailed discussion of positioning is beyond the scope of this course
- Can control position within normal flow using:
  - `position` property
    - Allows positioning relative to other elements or fixed on page
    - Specify position with `left`, `right`, `top`, `bottom`, and `z-index`
  - `float` property
    - Allows positioning to left or right of containing element
    - Text and inline elements flow around the element (e.g., an image in text)
- In many cases, these techniques are better replaced with Flex Box or Grid Layout

Flex Box and Grid Layout

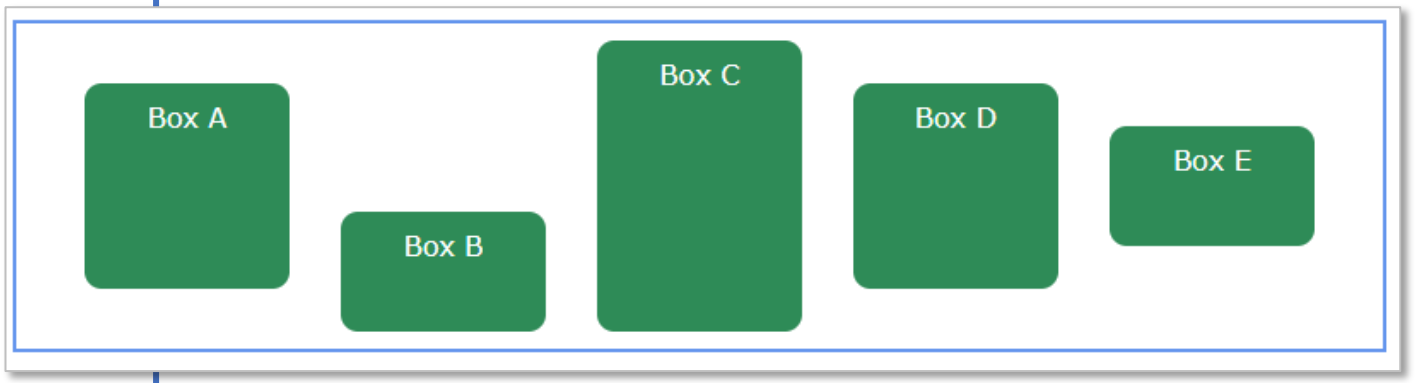
- It can be hard to align items on the page using the box model
  - CSS provides two additional layout models
  - Specify `display: flex` or `display: grid` on a containing element
- **Flex Box** is designed to lay out items in one dimension (e.g., equally across page)
  - There is one principal direction (`flex-direction`) along which items are laid out
  - They may also be aligned along the other direction (`align-items`)
  - Control whether items resize or wrap in the available space (`flex`, `flex-wrap`)
  - https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox
- **Grid Layout** is designed to control the overall page layout with items presented in a grid
  - Specify a grid template that includes rows and columns (`grid-template-columns`)
  - Control how rows and columns resize to fit content (`minmax`)
  - Control how items are positioned in the grid (`grid-column-start`, `grid-area`)
  - https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout

Flex Box Example

```
#container {
    display: flex;
    width: 80%;
    justify-content: space-evenly;
    align-items: center;
}

#b {
    align-self: flex-end;
}

.item {
    max-width: 100px;
    flex-grow: 1;
    flex-basis: auto;
    text-align: center;
}
```

How to arrange items in the container. In this case, the principal axis is horizontal (default) and items are spaced evenly in that direction while being center-aligned on the vertical axis.

This item will be aligned differently



How items change size when the container changes size. In this case, they will all vary at the same rate (`flex-grow`) and size will be set by the item size (`flex-basis`)

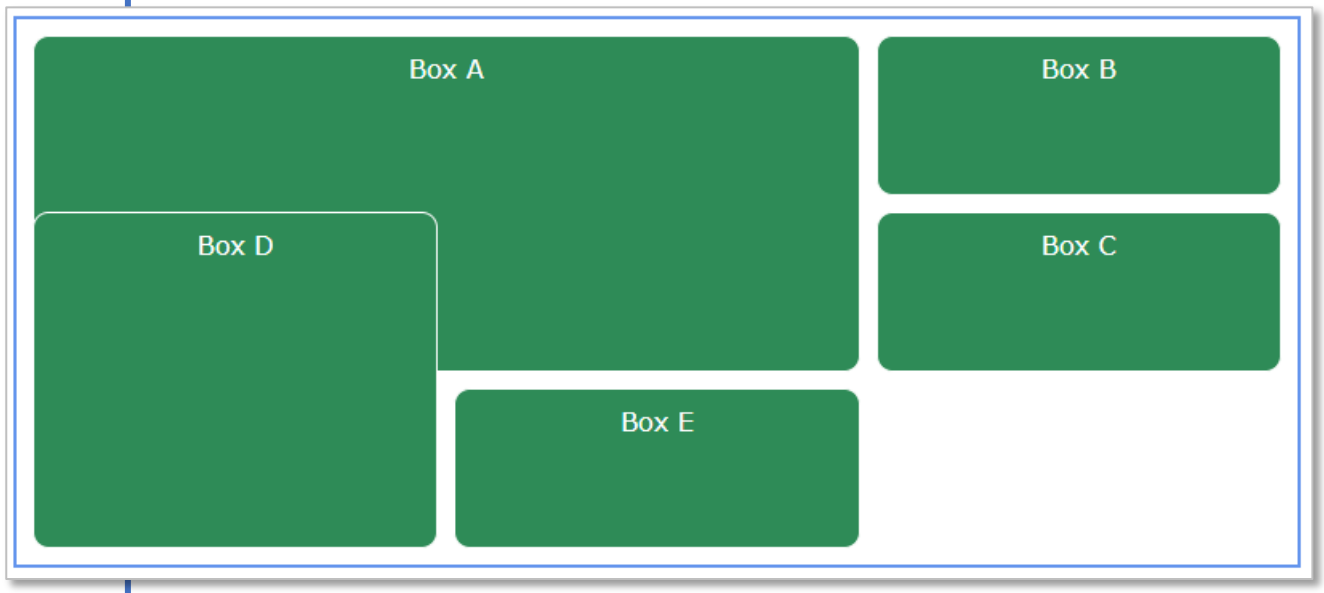
Grid Layout Example

```
#container {
    display: grid;
    width: 80%;
    grid-template-columns: repeat(3, 1fr);
    grid-auto-rows: 100px;
    grid-gap: 10px 10px;
}

#a {
    grid-area: 1 / 1 / 3 / 3;
}

#d {
    grid-row: 2 / 4;
    grid-column: 1 / 2;
}
```

How to arrange items in the container. In this case, defines a grid with 3 columns, each an equal fraction of the space. Rows are 100px high. Rows and columns have a 10px gap.

This item will span rows and columns 1 and 2

This item will span rows 2 and 3, but will fit in column 1 (alternative syntax)



Flex layout fro responsive layouts

Flex layout for responsive layouts

Chapter Concepts

HTML Layout

The Document Object Model

More HTML and CSS

Advanced CSS

Semantic Tags

Forms and Validation

HTML5

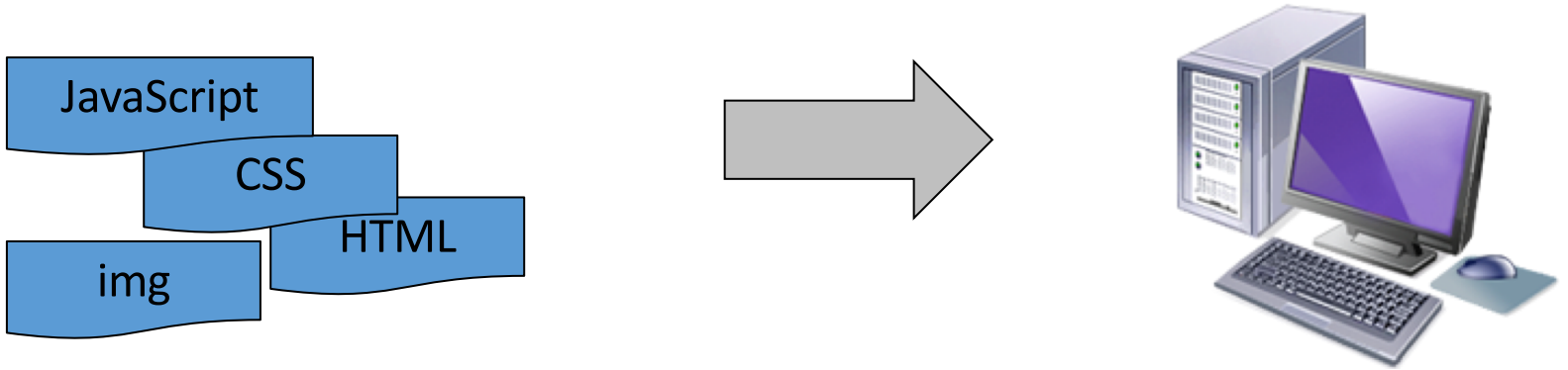
Chapter SummaryAPI's

What's in a Version?

- Earlier HTML standards had version numbers:
  - HTML 2.0, 3.2, 4.0, 4.01
  - HTML4 had 3 variations: strict, transitional, and frameset
  - After HTML4, W3C started to focus on a stricter approach (XHTML)
- WHATWG was formed because browser writers were dissatisfied with the W3C process
  - They considered it too slow
  - They felt the W3C was heading in the wrong direction with XHTML
- WHATWG maintains the HTML Living Standard (what browsers actually do)
  - https://html.spec.whatwg.org/multipage/
  - There are no version numbers and never will be!
- W3C, having abandoned XHTML, periodically publishes HTML standards
  - E.g., HTML 5.2 https://www.w3.org/TR/html/
  - Significantly based on snapshots of the WHATWG Living Standard

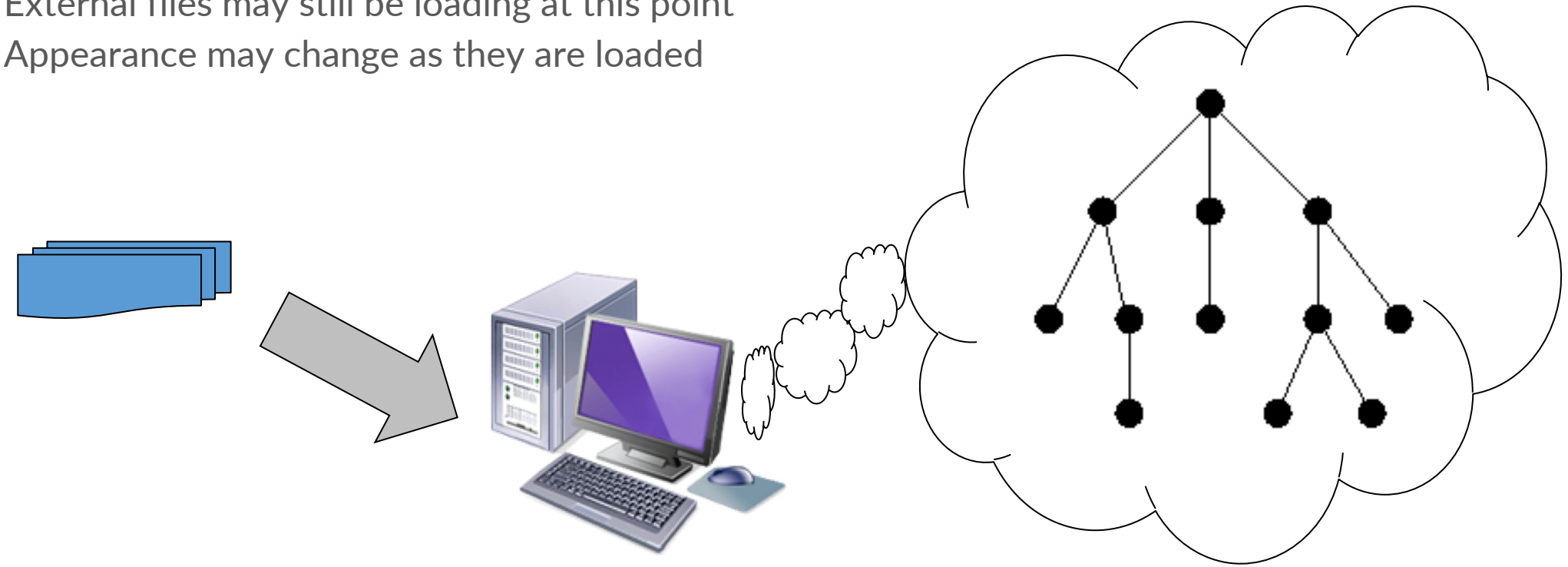
How HTML Works

- The web browser receives a document
  - Document is written in HTML
  - May have style specified using CSS (in-line or in external files)
  - May have behavior specified using JavaScript (in-line or in external files)
  - Refers to images, which are external files
- The browser starts to parse the document, constructing a Document Object Model (DOM)
  - Script files are retrieved immediately, when they are encountered
    - Possible to change this to make them load in parallel
  - Other files are loaded in parallel as parsing of the document continues



How HTML Works (continued)

- Script files are executed immediately
  - May be changed to force them to wait for the DOM to be created
  - Any script may change the DOM by adding, removing, or updating elements
  - Important to write scripts so they do not interact with DOM elements that have not been parsed yet
- Finally, the DOM is “rendered” (displayed to the user)
  - External files may still be loading at this point
  - Appearance may change as they are loaded

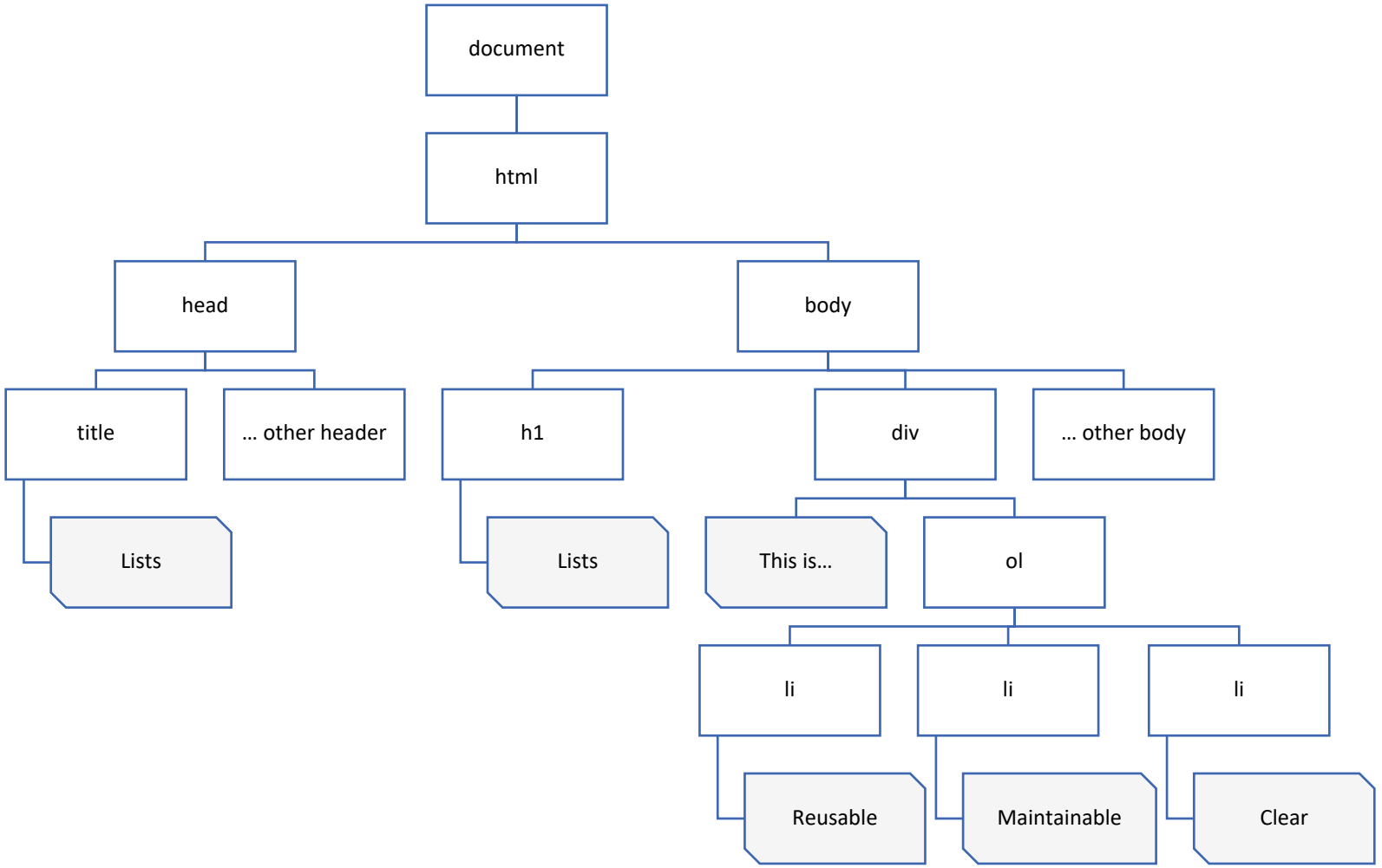


Older Browsers

- The original HTML5 was designed as a superset of HTML
  - Pages degrade gracefully when browsers do not support features
  - Still usable in older browsers (not all features will be present)
- As the name “HTML Living Standard” acknowledges, browsers move at their own pace
  - Before deciding to use a feature, check compatibility on websites such as:
    - https://caniuse.com/ (and check browser usage at https://caniuse.com/usage-table)
    - https://developer.mozilla.org
- Do not attempt to check browser version – check feature support instead
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent
  - A common library to do this is Modernizr
- Provide compatibility using polyfills or shims
  - JavaScript libraries that provide new features in older browsers
  - CSS workarounds that minimize incompatible implementations
  - https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/HTML_and_CSS

The Document Object Model (DOM)

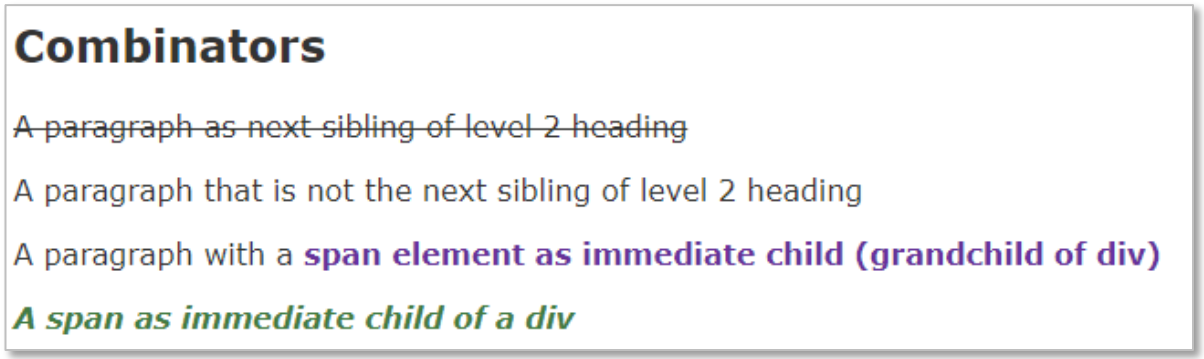
```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Lists</title>
    … other header content
</head>

<body>
    <h1>Lists</h1>
    <div>
        This is an ordered list:
        <ol>
            <li>Reusable</li>
            <li>Maintainable</li>
            <li>Clear</li>
        </ol>
    </div>
    … other body content
</body>

</html>
```



Relationship-Based Selectors (Combinators)

- Selectors can be used to select elements based on their relationships to other elements
  - `E F`
    - All F that are descendants of E (at any level)
  - `E > F`
    - All F that are immediate children of E
  - `E + F`
    - All F that are the next sibling of E

```
div span {
    color: rebeccapurple;
    font-weight: bold;
}

div > span {
    color: #437c43;
    font-style: italic;
}

h2 + p {
    text-decoration: line-through;
}
```



Attribute Selectors

- `E[attr]`
  - All E with attribute `attr`
- `E[attr=value]`, `E[attr=value i]`
  - All E with attribute `attr` having `value` (exactly or case insensitive)
- `E[attr^=value]`
  - All E with attribute `attr` starting with `value`
- `E[attr$=value]`
  - All E with attribute `attr` ending with `value`
- `E[attr*=value]`
  - All E with attribute `attr` containing `value`

Cascading and Specificity

- Style rules “cascade” in four ways:
  - More local styles are used in preference to external styles
  - Children inherit their parents’ styles
  - More specific rules take precedence over less specific rules
  - The last style specified at a given specificity wins
- Local styles
  - Inline styles over internal document styles over external styles
- Children inherit their parents’ styles
  - A rule for `html` will apply the rule to every element that doesn’t have a conflicting rule
- More specific rules
  - Rules specified for `#id` take precedence over `.class` over `element`
  - Pseudo-classes and attribute selectors make rules more specific
  - `element.class` is more specific than either `element` or `.class`
  - https://css-tricks.com/strategies-keeping-css-specificity-low/
  - The * selector has high specificity and is less efficient than a rule for `html`

ig ID

Chapter Concepts

HTML Layout

The Document Object Model

More HTML and CSS

Advanced CSS

Semantic Tags

Forms and Validation

HTML5 API's

Chapter Summary

iframe

- Embed an external web page

```
<a href="colors.html" target="main">Colors</a>

<iframe name="main" src="blank.html"></iframe>
```

- Supports a `sandbox` attribute to specify a list of space-separated permissions:
  - `allow-same-origin`: allow frame access to resources from same origin
  - `allow-scripts`: run scripts, but not open popups windows
  - `allow-top-navigation`: navigation in top-level browsing context (`target="_top"`)
    - See next slide
  - `allow-popups`: allow frame to open popup windows (`target="_blank"`)
    - Any attempt to navigate to a disallowed context will be treated as a popup

Link Targets

- `a` element supports an optional `target` attribute
  - Indicates where to open the link

"" or omitted	Current "browsing context" (current tab or `iframe`)
`_blank`	A new tab, the tab remains anonymous (no `name`)
`_self`	Current tab or `iframe`
`_parent`	Parent if there is one, current tab or iframe if there isn't
`_top`	Top-level context (usually the tab)
`name`	If `name` doesn't exist, create a tab with that name. If `name` exists and either has same origin, or was opened by this context, then open in `name`.

  - Note that browser configuration or `iframe` parameters may change the behavior
    - Pop-up blocker
    - Sandboxing

Tables

- Display tabular data

Table Row

Table Header

Table Data

```
<table class="outline-only">
    <caption>Countries of the Americas whose capital city has changed</caption>
    <tr>
        <th>Country</th>
        <th>Old capital</th>
        <th>New capital</th>
    </tr>
    <tr>
        <td>Bermuda</td>
        <td>St George</td>
        <td>Hamilton</td>
    </tr>
    <tr>
        <td>Brazil</td>
        <td>Rio de Janeiro, RJ</td>
        <td>Bras&iacute;lia, DF</td>
    </tr>
    <tr> <td>Honduras</td> <td>Comayagua</td> <td>Tegucigalpa</td> </tr>
    <tr> <td>USA</td> <td>New York, NY</td> <td>Washington, DC</td> </tr>
</table>
```

Countries of the Americas whose capital city has changed

Country	Old capital	New capital
Bermuda	St George	Hamilton
Brazil	Rio de Janeiro, RJ	Brasília, DF
Honduras	Comayagua	Tegucigalpa
USA	New York, NY	Washington, DC

Blue lines show outline of individual elements

Table Styles

- Each cell (`td` or `th` element) is treated individually when applying styles

- `border`: cell border is not shared with adjacent cells, consider:
  - `border-collapse`
  - `border-spacing` distance between borders of adjacent cells if border is not collapsed

- Extremely common to apply different styles to alternate rows

```
table, th, td {
    border: 1px solid black;
    border-spacing: 2px;
    padding: 2px;
}


/* Collapsed borders. Default is separate */
.collapse {
    border-collapse: collapse;
}

/* Striped rows for readability */
.collapse tr:nth-child(even) td {
    background: #dedede;
}
```

Countries of the Americas whose capital city has changed

Country	Old capital	New capital
Bermuda	St George	Hamilton
Brazil	Rio de Janeiro, RJ	Brasília, DF
Honduras	Comayagua	Tegucigalpa
USA	New York, NY	Washington, DC

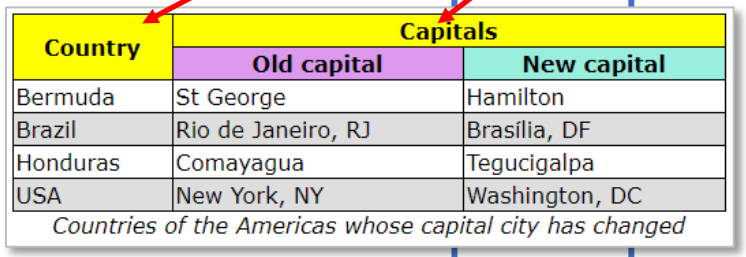
Countries of the Americas whose capital city has changed

Country	Old capital	New capital
Bermuda	St George	Hamilton
Brazil	Rio de Janeiro, RJ	Brasília, DF
Honduras	Comayagua	Tegucigalpa
USA	New York, NY	Washington, DC

Table Structure

- Merge cells using `colspan` or `rowspan`
- All content of table is in implied `tbody`
  - For more control, specify an explicit `thead` and `tbody`

```
/* Striped rows for readability, but only rows
in the tbody (not in the thead) */
.interesting tbody tr:nth-child(even) td {
    background: #dedede;
}

th.old-capital {
    background: #dd99ee;
}

th.new-capital {
    background: #99eedd;
}
```



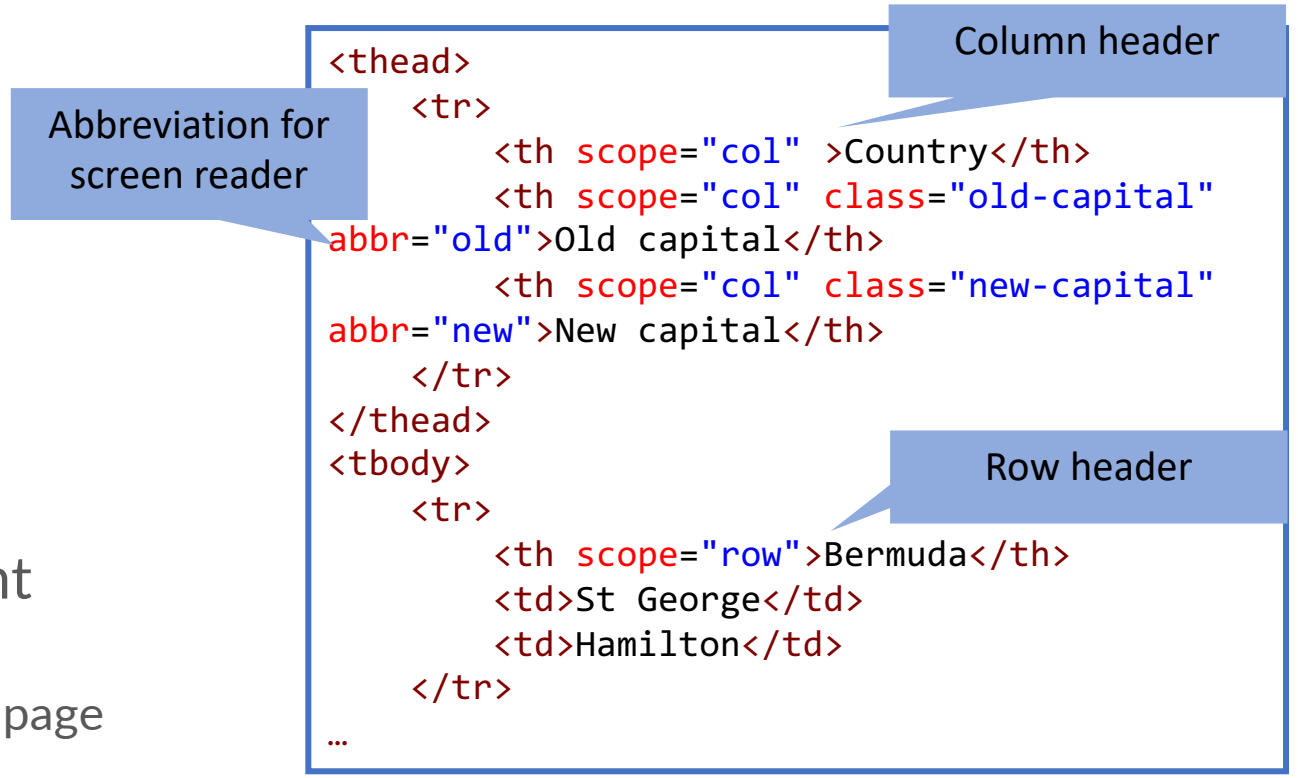
Country	Capitals	
	Old capital	New capital
Bermuda	St George	Hamilton
Brazil	Rio de Janeiro, RJ	Brasília, DF
Honduras	Comayagua	Tegucigalpa
USA	New York, NY	Washington, DC

Countries of the Americas whose capital city has changed

```
<table class="simple interesting yellow-headers">
    <caption>Countries of the Americas whose capital
city has changed</caption>
    <thead>
        <tr>
            <th rowspan="2">Country</th>
            <th colspan="2">Capitals</th>
        </tr>
        <tr>
            <th class="old-capital">Old capital</th>
            <th class="new-capital">New capital</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Bermuda</td>
            <td>St George</td>
            <td>Hamilton</td>
        </tr>
    …
    </tbody>
</table>
```

Accessibility Considerations for Tables

- For screen readers:
  - Provide a caption for the table
  - Annotate header cell to specify whether it is for a row or a column
  - Avoid complex tables with many merged cells
  - Specify an abbreviation for long headers
    - Because screen readers repeat header before reading each value
- Do not use tables to lay out page content
  - Use them to present tabular data
  - Use Flex Box and Grid Layout to control the page



Try It Now: Tables

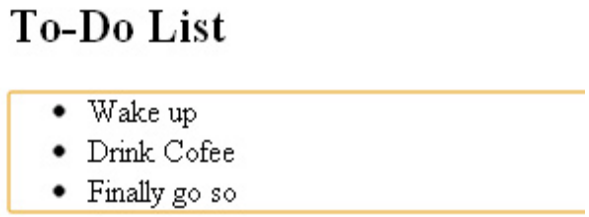
10 **min**

- Launch `Ch02\AdvancedStyleExamples\index.html`
  - Click on “Tables”
  - View the output
- Open the files `Ch02\AdvancedStyleExamples\tables.html` and `Ch02\AdvancedStyleExamples\css\style.css`
  - Make changes to the layout and formatting
  - See the impact of changes in the browser

Editable contents on a web page

- A new attribute can be applied to elements, called **contenteditable**.
- As the name implies, this allows the user to edit any of the text contained within the element, including its children.

```
<ul contenteditable="true">
    <li> Wake up </li><li> Drink Coffee </li>
    <li> Finally go so </li>
</ul>
```



- It’s a step towards an interface more similar to a classic desktop GUI.

Chapter Concepts

HTML Layout

The Document Object Model

More HTML and CSS

Advanced CSS

Semantic Tags

Forms and Validation

HTML5 API's

Chapter Summary

Advanced CSS Effects

- CSS Level 3 added some new features
  - Backward compatible since CSS always ignores anything unrecognized
  - Note that CSS Level 3 (or CSS3) is really a collection of standards
    - CSS3 is a useful shorthand, but not an official standard
- Enables CSS to produce effects previously only achievable with graphic tools such as PhotoShop or animation tools like Flash:
  - Advanced borders (rounded corners, shadows)
  - Advanced text decoration (wavy underlines)
  - Gradients
  - Create new shapes through transformations
  - Add “life” to interactive elements through transitions

Advanced CSS Selectors – Attribute Selectors

- E[foo]
  - Selects an E element with a foo attribute.
- E[foo="bar"]
  - Selects an E element whose foo attribute value is exactly equal to bar
- E[foo="bar" i]
  - Selects an E element whose foo attribute value is exactly equal to bar, regardless of its case. Basically, using i allows you to specify "case-sensitive" when specifying the value.
- E[foo~="bar"]
  - Selects an E element whose foo attribute value is a list of whitespace-separated values, one of which is exactly equal to bar.
- E[foo^="bar"]
  - Selects an E element whose foo attribute value begins exactly with the string bar.
- E[foo$="bar"]
  - Selects an E element whose foo attribute value ends exactly with the string bar.
- E[foo*="bar"]
  - Selects an E element whose foo attribute value contains the substring bar.

Advanced CSS Selectors – Combinators

- E F
  - Selects an F element descendant of an E element.
- E > F
  - Selects an F element child of an E element.
- E + F
  - Selects an F element immediately preceded by an E element.
- E ~ F
  - Selects an F element preceded by an E element.

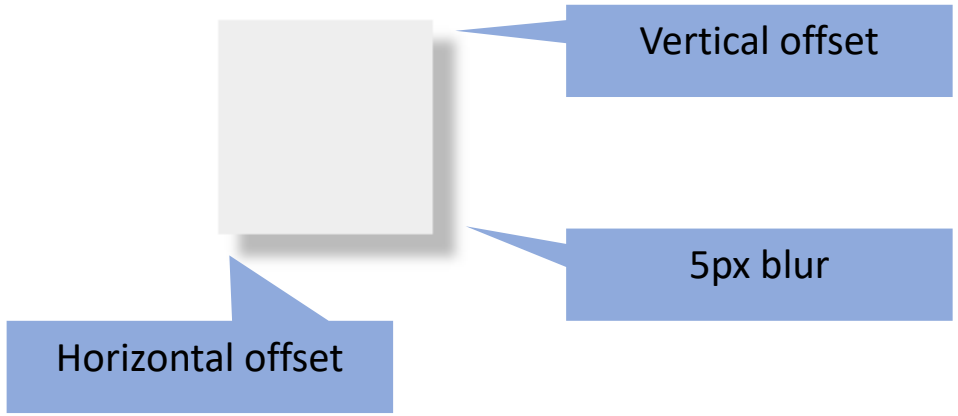
Advanced CSS Selectors – Combinators

- E:first-child
- E:last-child
- E:only-child
- E:nth-child(n)
- E:nth-last-child(n)
- E:nth-of-type(n)
- E:nth-last-of-type(n)
- E:first-of-type
- E:last-of-type
- E:only-of-type

Hierchical selectors

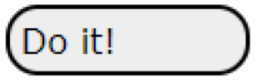
Borders

- Box shadow
  - Specify horizontal offset, vertical offset, and blur of shadow




```
#shadow {
    width: 100px;
    height: 100px;
    background-color: #eeeeee;
    box-shadow: 10px 5px #bbb;
}
```

- Round rectangle
  - Use border-radius




```
#rounded-corners {
    border: 2px solid black;
    padding: 5px;
    background: #eeeeee;
    width: 100px;
    border-radius: 15px;
}
```

Linear Gradient

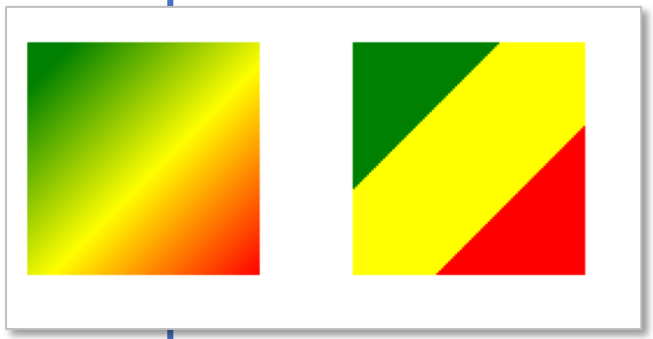
- Linear gradient property declaration format is:
  - `background: linear-gradient(direction, color1, color2, color3, …)`
  - `direction` is a pre-defined string, starting `to`, or an angle

```
#linear-gradient {
    height: 100px;
    width: 100px;
    background: linear-gradient(to bottom right, green 10%, yellow, red);
}

#linear-stripes {
    height: 100px;
    width: 100px;
    background: linear-gradient(to bottom right, green 32%, yellow 32% 68%, red 68%);
}
```

Direction of gradient

Hints move the transition away from the mid-point between the colors

Overlapping hints cause hard boundaries between colors



- Also, `repeating-linear-gradient`
- `conic-gradient` arranges colors around a circle

Radial Gradient

- Linear gradient property declaration format is:
  - `background: radial-gradient(shape, color1, color2, color3, …)`
  - `shape` is `circle` or `ellipse`

```
#radial-gradient {
    height: 100px;
    width: 100px;
    background: radial-gradient(circle, #0055A4, white, #EF4135, white);
}

#radial-stripes {
    height: 100px;
    width: 100px;
    background: radial-gradient(circle, #0055A4 24%, white 24% 48%, #EF4135 48% 71%, white 71%);
}
```

Can also move the center point using, e.g., `circle at top`

But if you really want to create a roundel, use SVG instead

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="-3 -3 6 6">
    <circle fill="#EF4135" r="3" />
    <circle fill="#FFFFFF" r="2" />
    <circle fill="#0055A4" r="1" />
</svg>
```

Transitions

- In normal use, style property changes occur instantly
- Transitions allow changes to happen gradually with intermediate states created automatically
  - Do not use this for animation, it is intended for “one-off” changes

- http://cubic-bezier.com/#0,0,0,0

```
#transition div {
    width: 100px;
    transition-property: width;
    transition-duration: 2s;
    margin-left: 0px;
}

#transition:hover div {
    width: 200px;
}

#transition1 {
    transition-timing-function: ease-in-out;
}

#transition2 {
    transition-timing-function: cubic-bezier(0.68,
-0.55, 0.265, 1.55);
}

#transition3 {
    transition-timing-function: linear;
}
```

Apply transition when width changes

Change width on mouseover of parent

How “fast” to apply the change (easing function)

Transformations

- Change the appearance of an existing object
  - Can create new shapes
  - Can rotate text or image
- Can be combined with transitions, but this is not their primary purpose
  - Primary purpose is to give more control over presentation of individual object

```
#transform div {
    width: 50px;
    height: 50px;
    background: green;
    margin: 25px 6px;
    display: inline-block;
}

#transform:hover #transform1 {
    transform: translateY(-20px);
}

#transform:hover #transform2 {
    transform: rotate(30deg);
}

#transform:hover #transform3 {
    transform: skewX(-30deg);
}
```

Draw squares

On mouseover, transform squares

Exercise 2.1: Applying Advanced CSS Styling Effects

20 **min**

- Continue working with your solution to the previous exercise
- Change the way the photographs are displayed:
  - Make their width 300px.
  - Add a drop-shadow.
  - Make sure the first one is outside the normal flow of text and on the left. The second one should be on the right. Make sure there is some spacing between the images and the text.
  - Add a transition so that the images are displayed at their full width (600px) when the mouse pointer is over them.
- Add a suitable gradient to the page background.
- To see your changes, you will need to reload the web page.

Optional Exercise 2.2: Static Web Pages

30 min

- Open the project Ch02\StaticWebPage and create a static web page with a separate CSS file.
- Your web page should look somewhat like the one below, but in particular:
  - There should be an overall page header containing a heading and links.
  - There should be a copyright message at the foot of the page.
  - The rest of the page should be divided into two distinct parts with some indication of which part is which (e.g., a border).
  - The first of the two parts should be further subdivided into a section with photos and another with bullets. Align the photos seamlessly.
  - The second of the two parts should contain other photos or text. If you use photos, arrange them differently from those in the previous section.
  - Optionally, create additional pages for the links and/or clicking the pictures.
- Do not make changes at random: have a plan and aim to stick to it as closely as possible. If you need ideas, you can try to reproduce the sample page exactly.

Optional Exercise 2.2: Static Web Pages

HANDS-ON EXERCISE

30 **min**

Static Web Page

Option One Option Two

First Part of the Page

Section one contains some photos:

Section two contains interesting things:

- CompuServe introduced the GIF format in 1987. Unknown to them, Unisys owned the patent to the Lempel-Ziv-Welch (LZW) compression technique on which GIF was based. In 1994, Unisys tried to charge a fee for using GIFs, with the result that the PNG format was invented as an alternative. In 2004, the last patents on GIF technology expired, allowing it to be used for free.
- In English, the normal adjective order is: opinion, size, physical quality, shape, age, colour, origin, material, type and purpose. Native speakers are never taught this order explicitly yet use it without conscious thought.
- In Switzerland, it is illegal to own only one guinea pig. A 2008 animal rights law protects these social animals from loneliness.
- Bananas are curved because they grow towards the sun. They initially grow upwards before gravity turns them towards the ground, however they then undergo a process called "negative geotropism", where they turn towards the sun.

More Interesting things

Copyright © 2019 The Company

Chapter Concepts

HTML Layout

The Document Object Model

More HTML and CSS

Advanced CSS

Semantic Tags

Forms and Validation

HTML5 API's

Chapter Summary

Adding Document Structure with Semantic Elements

- Historically, web authors used a variety of means to signal the meaning of their content
  - E.g., `<div class="footer">`
  - Easy for humans to understand, not useful for screen readers
- HTML now contains many elements known as semantic elements
  - Convey better meaning of role of an element in a document
- Examples include:
  - `header`, `footer`
  - `nav`
- Benefits of new elements include:
  - More maintainable HTML
  - Easier to style with CSS
  - Better search engine rankings

`body`, `head`, and Overall Structure

- `body` and `head` are ***not*** semantic elements, they are required parts of an html document
  - May be omitted, but all practical documents include `body` and `head`
  - If included, there may be exactly one of each
  - `head` must precede `body`
- The heading levels, `h1-h6`, represent headings for a section of a document
  - Also not semantic elements, but level should not be chosen for presentation reasons
  - Should start at a high level and work down if hierarchical headings are required
- `<header>`
  - Used to contain the headline for a page or section (typically an `h1-h6`)
  - May also contain navigation information or other useful introductory material
  - Can have several header elements per document
- `<footer>`
  - Contains summary information for a page or section

Semantic Elements

- `<nav>`
  - Contains navigation links
  - Type of navigation may be determined by where it is
    - Global site navigation as direct child of body
    - In-page navigation within an article or section
  - Not all links need to be in a nav
- `<article>`
  - Represents self-contained content, e.g., for syndication
  - Should make sense on its own regardless of other content on the page
  - Nested articles make sense in some cases (e.g., replies to a blog post)
- `<section>`
  - Contains content that should be grouped together
  - May be used to break up articles
  - Use when thematically correct, not as a styling convenience

Semantic Elements (continued)

- `<main>`
  - Dominant contents of a document
  - Must not be more than one unless hidden
- `<aside>`
  - Content related to the page but can exist independently
  - Typically used for sidebars
- `<figure>`
  - Used to display images, audio, tables, code examples, etc.
  - Enables addition of a caption
  - Figures may be nested to create a montage
- `<figcaption>`
  - Used to define caption for a figure
  - Must be first or last child of a `<figure>`

Semantic Elements Example

- The purpose is to define the structure of the document
- Semantic elements do not have default styles
  - However, they are defined as block-level elements

```
<body>
    <header>
        <h1>Semantic Elements</h1>
        <nav><a href="">Top-level Navigation</a></nav>
    </header>
    <article>
        <header>
            <h2>First Article</h2>
        </header>
        <section>
            <p>Something interesting.</p>
        </section>
        <section>
            <p>Something else interesting.</p>
        </section>
    </article>

    <footer>
        Copyright &copy; 2019 The Company
    </footer>
</body>
```

Header for page

Header for article

Sections within article

Footer for page

Exercise 2.3: Creating an HTML5 Page Using Semantic Tags

20 **min**

- The idea is to produce the web page shown, or something very similar. You can either work from the exercise starter, or your solution to Exercise 3.2, if you did it.
- Use the following semantic tags to replace divs or other elements:
  - `header`: to contain the page level heading and the links.
  - `nav`: to contain the top-level links.
  - `article`: to replace the two parts of the page.
  - `section`: to differentiate the two sections of the first part.
  - `header`: to contain each of the headings inside individual articles or sections.
  - `footer`: to contain the copyright message.
  - `figure`: to contain the images in one article. In the solution, we have chosen to do this for the second section. Give each figure a `figcaption`.
  - Add an `aside` with a suitable message.
- Add borders to the semantic tags: black for header, red for article, blue for section, and green for aside.

Chapter Concepts

HTML Layout

The Document Object Model

More HTML and CSS

Advanced CSS

Semantic Tags

Forms and Validation

HTML5 API's

Chapter Summary

HTML Forms

- HTML forms are collections of input controls that have a single Submit action
  - A submit button sends form data to the `action`, a URL on the server
  - Most forms have `method="post"`, which sends form data in the body
  - Not all input controls must be in forms, but if the intention is to send the data to a server, then a form is appropriate
  - Controls have a `name` and this is the key when values are sent to the server
- Input controls may have a label associated with them
  - Strongly prefer a label over a simple text element
  - Screen readers can understand the relationship between the label and the control
  - Either explicit with `for` keyword

```
<label for="name">Name:</label>
<input type="text" id="name" name="userName">
```

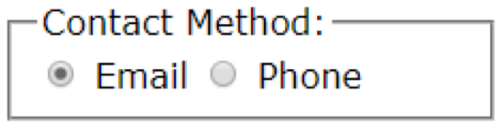
  - Or implicit by enclosing the control

```
<label>Name:
    <input type="text" id="name" name="userName">
</label>
```

Structure Within Forms

- Within a form, controls can be grouped using `<fieldset>`
  - Optional first child can be a `<legend>`
    - If supplied, appears in the `<fieldset>` frame and is used by screen readers

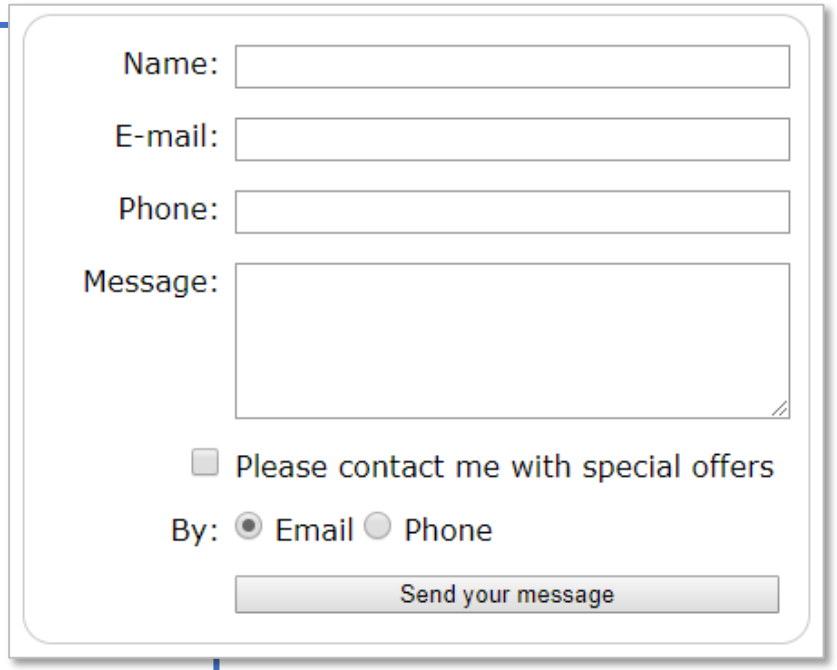
```
<fieldset>
    <legend>Contact Method:</legend>
    <input type="radio" id="byEmail" name="contact" value="email" checked>
    <label for="byEmail">Email</label>
    <input type="radio" id="byPhone" name="contact" value="phone">
    <label for="byPhone">Phone</label>
</fieldset>
```



  - Can use `<fieldset>` as a purely thematic grouping by hiding the border
    - Advantageous for screen readers
  - Radio buttons do not have to be related by a `<fieldset>`
    - Radio groups are defined by the `name`, not the `<fieldset>`

HTML Form Example

```
<form action="results.html">
    <label for="name" class="col1">Name:</label>
    <input type="text" id="name" name="userName" class="col2">
    <label for="email" class="col1">E-mail:</label>
    <input type="email" id="mail" name="userMail" class="col2">
    <label for="phone" class="col1">Phone:</label>
    <input type="phone" id="phone" name="userPhone" class="col2">
    <label for="msg" class="col1">Message:</label>
    <textarea id="msg" name="userMessage"></textarea>
    <input type="checkbox" id="optin" name="optIn" class="col1">
    <label for="optin" class="col2">Please contact me with
special offers</label>
    <div class="col1">By:</div>
    <fieldset class="col2">
        <input type="radio" id="byEmail" name="contact" value="email" checked>
        <label for="byEmail">Email</label>
        <input type="radio" id="byPhone" name="contact" value="phone">
        <label for="byPhone">Phone</label>
    </fieldset>
    <button type="submit" class="col2">Send your message</button>
</form>
```

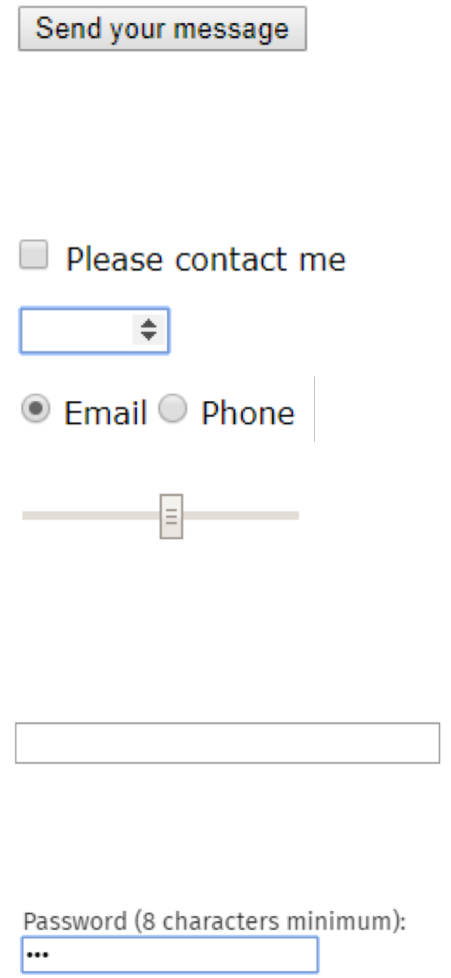


HTML5 Forms

- HTML5 provides new features that simplify building robust interactive forms
  - Specific input fields for certain data types
  - Declarative validation and error messages using element attributes
- As a result, the variety of input types has increased
  - For example:
    - Number
    - Email
    - Telephone number
    - URL
    - Date
    - Time
    - Color
    - Search

HTML Input Types

type="…"	Description
`button`, `reset`, `submit`	A simple button, one that resets the form, or one that submits the form.
`color`, `date`, `file`, `time`	Provide special selector according to the type of value.
`checkbox`	A simple on/off choice. `value` attribute specifies the "on" value, otherwise `on`.
`number`	Numeric input. Generally has a spinner on the right. Supports `min` and `max`.
`radio`	Grouped into a radio group where controls have a common `name` attribute. `value` attribute specifies the "on" value, otherwise it is `on`.
`range`	A numeric value between `min` and `max`, use where precision is not important.
`tel`	A telephone number, but no validation is usually imposed. User agents, such as mobile browsers, can provide special interaction.
`text`, `search`, `email`, `url`	Simple single line text input. `search` is functionally identical, but may be styled differently. The `email` and `url` versions validate according to the appropriate standard (e.g., email addresses must have an @ sign), but not that the value exists.
`hidden`	Value is submitted with the form, but not visible.
`password`	Single line text input with obscured text.



Date, Time, and Color Pickers

- Date

Start date:

22/07/2018

July 2018

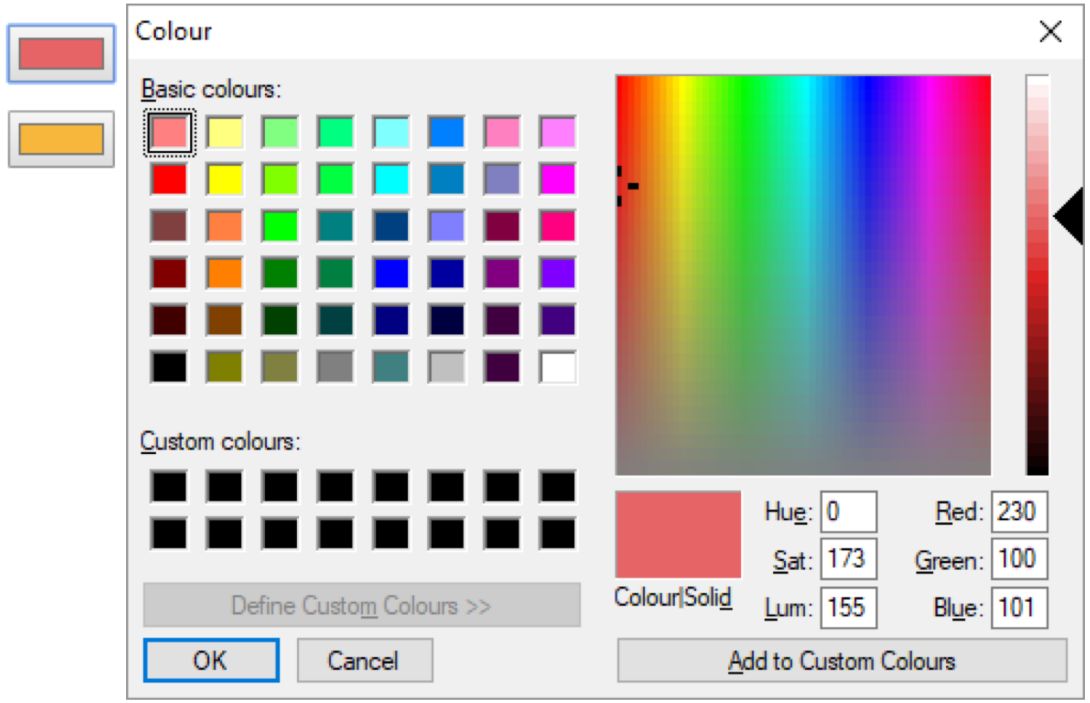
Mon	Tue	Wed	Thu	Fri	Sat	Sun
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

- Time

Choose a time for your meeting:

09:03 Office hours are 9am to 6pm

- Color



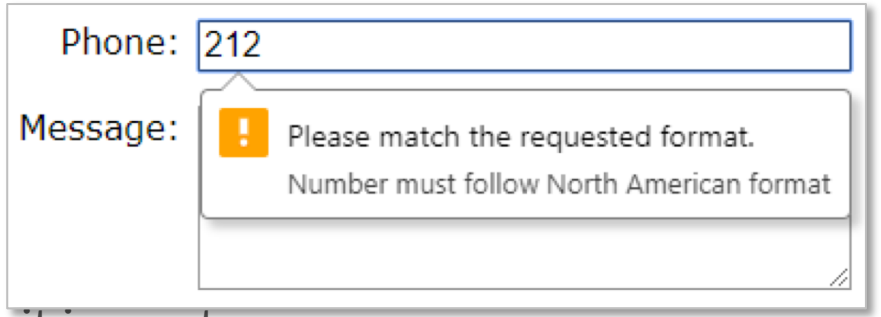
- Also variations for date and time, month and year, and week number

Other Form Controls

- `<textarea>`
  - A multi-line field
  - By default, `<textarea>` uses a monospaced font
- `<select>`
  - A `<select>` control is a drop-down list
  - It contains a series of `<option>` entries representing the individual values
  - It may contain `<optgroup>` entries to group `<option>` values
- `<button>`
  - A clickable button
  - Supports an optional `type` that takes a value `submit` (default), `reset`, or `button`
  - Functionally equivalent to an input control with `type="button"`
  - Allows more versatile styling than input control

Form Validation

- Some input types have built-in validation
  - `required` indicates that an input element must have a value
  - `title` can be used to customize the error message



- Otherwise validation can be added through attributes:
  - `pattern` is a regular expression that must be matched
  - `minlength` and `maxlength` for text fields
  - `placeholder` is text that will appear faintly inside the field when it is empty
    - Do *not* use this instead of a label

```
<input type="phone" id="phone" name="userPhone"
    required
    pattern="[2-9][0-9]{2}-[0-9]{3}-[0-9]{4}"
    placeholder="XXX-XXX-XXXX"
    title="Number must follow North American format">
```

- Pseudo-classes `:invalid` and `:valid` can also be used for visual feedback

Exercise 2.4: Creating an HTML Form with Validation

20 **min**

1. You are responsible for building a form for a corporate support website. It will capture customer details and the details of their issue.
2. Open the file Ch02\Forms\form-exercise.html.
   a) Complete the TODO items.
   b) Don't worry about the appearance, just concentrate on meeting the requirements.
3. Test your form and check that the results page shows the values you expect.

Chapter Concepts

HTML Layout

The Document Object Model

More HTML and CSS

Advanced CSS

Semantic Tags

Forms and Validation

HTML5 API's

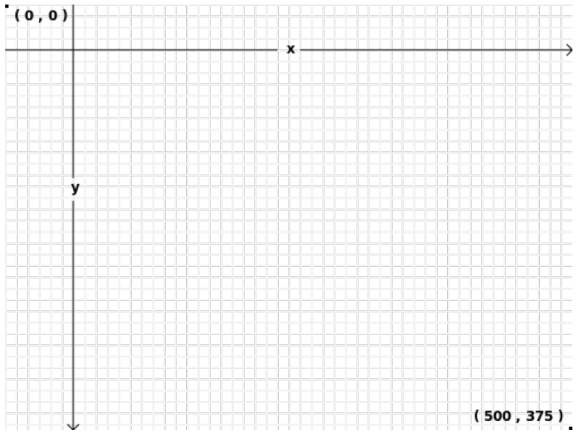
Chapter Summary

Canvas

- HTML5 defines the **canvas** element as "a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics, or other visual images on the fly."
- A canvas is **a rectangle in your page where you can use JavaScript to draw anything you want**.

```
<canvas width="640" height="480"></canvas>
```

- The canvas is a two-dimensional grid.
  - The coordinate (0, 0) is at the upper-left corner of the canvas.
  - Along the X-axis, values increase towards the right edge of the canvas.
  - Along the Y-axis, values increase towards the bottom edge of the canvas.



Local storage for Web Application

Client Application

- XML file
- INI file
- SQLite
- DBMS
- Specific application file (binary or text)

Web Application

- Cookie
- Nothing else? :/

Using Cookies for persistent data

- Cookies are included with every HTTP request Redundant transmission of the same information
- Cookies are **unencrypted** (unless the entire web application is served over SSL)
- Limited to about 4 KB of data
- They are not really useful for a web 2.0 application

What is really needed ?

- A lot of storage space on the client that **persists beyond a page refresh** and is **not transmitted to the server**.

Web Storage

Session Storage

- It is intended for **short-lived data** and shared only with pages from the same domain opened in the same window/tab and does **not persist after the window/tab is closed**.
- Every time a user opens a page in a new window/tab, the browser creates a new session storage database.

Local Storage

- This mechanism allows **storing data for more than a single session**. **Storage area is persistent** and not limited to the lifetime of the window/tab.

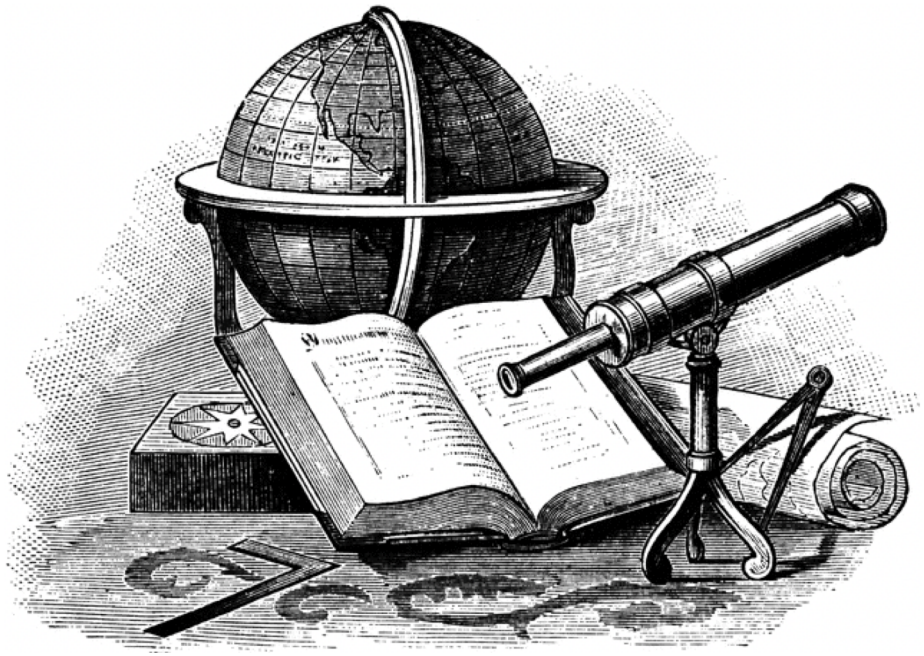
These ones are exposed through the **sessionStorage** and **localStorage** global objects (attribute of the window object), respectively.

Web Storage

- The **sessionStorage** methods can be also applied to localStorage
- An application can store **up to 5 MB** of data
- All modern browsers support Web Storage

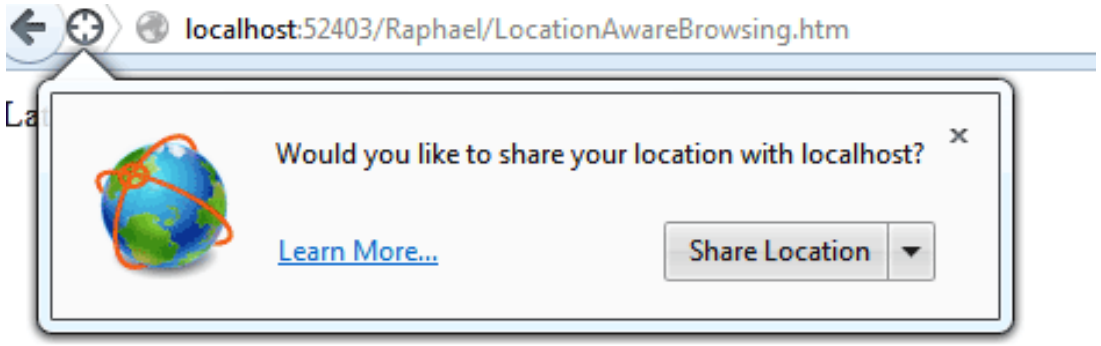
Geolocation

- Before HTML5, geolocation was already possible through the analysis of data from:
  - IP address
  - Wireless network connection (LAN)
  - Mobile phone tracking through triangulation of *hello messages* from **B**ase **T**ransceiver **S**tations
  - Tracking latitude and longitude (WGS84) via *Global Positioning System*



Geolocation

- The geolocation API lets you **share your location with trusted web sites**
- Latitude and longitude are available to JavaScript on the page
- *"User Agents must not send location information to Web sites without the express permission of the user"*



Chapter Concepts

HTML Layout

The Document Object Model

More HTML and CSS

Advanced CSS

Semantic Tags

Forms and Validation

HTML5 API's

Chapter Summary

Chapter Summary

In this chapter, we have discussed:

- Standardizing presentation with HTML and CSS
  - How HTML lays out a page
  - Additional HTML elements
  - Advanced styling with CSS
  - Semantic tags
  - Forms and validation