

A WEB PAGE, STEP-BY-STEP

The demonstration in this chapter has five steps that cover the basics of page production:

Step 1: Start with content. As a starting point, we'll write up raw text content and see what browsers do with it.

Step 2: Give the document structure. You'll learn about HTML element syntax and the elements that set up areas for content and metadata.

Step 3: Identify text elements. You'll describe the content using the appropriate text elements and learn about the proper way to use HTML.

Step 4: Add an image. By adding an image to the page, you'll learn about attributes and empty elements.

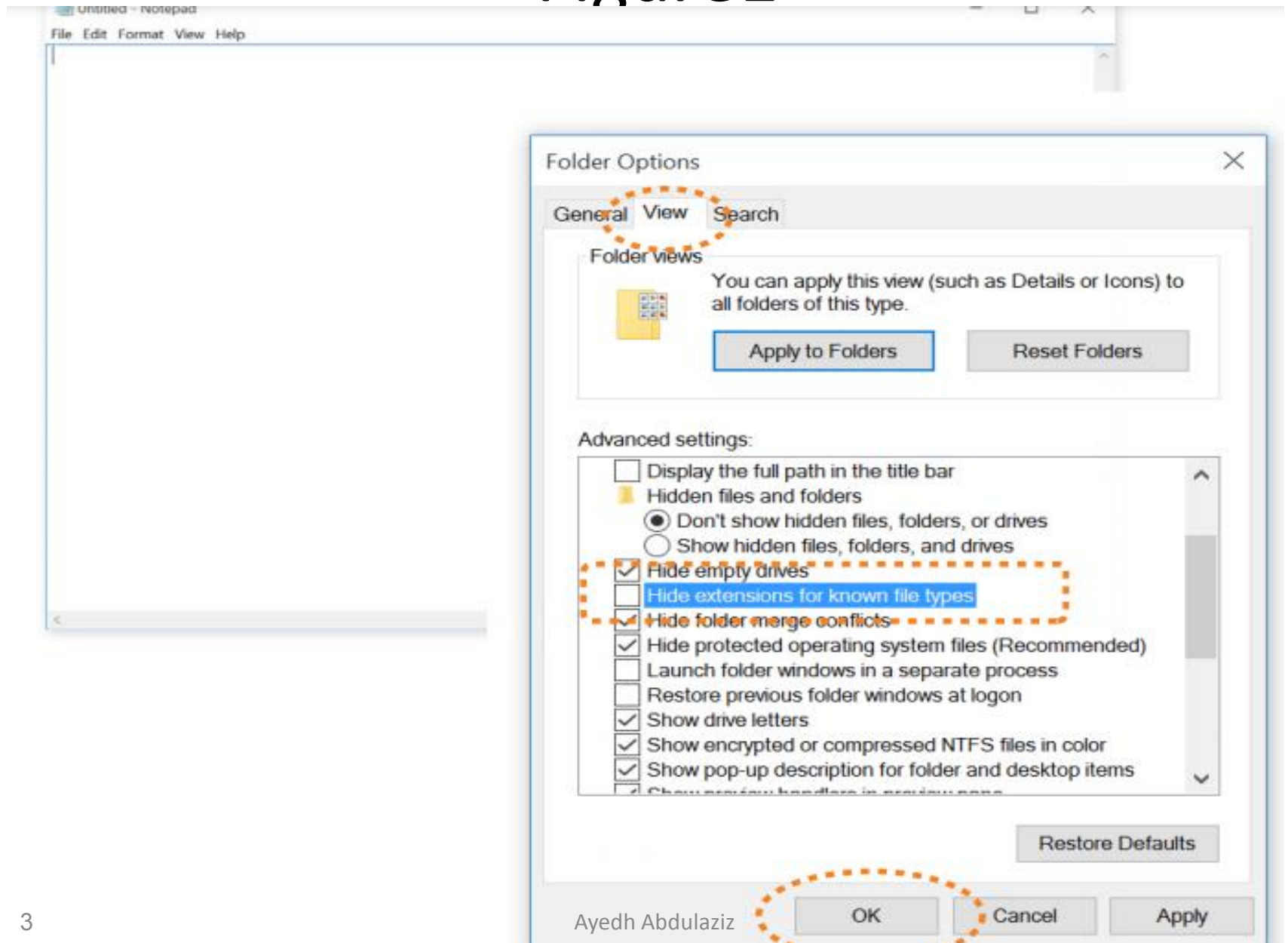
Step 5: Change how the text looks with a style sheet. This exercise gives you a taste of formatting content with Cascading Style Sheets

LAUNCH A TEXT EDITOR

Creating a New Document in Notepad (Windows) These are the steps to creating a new document in Notepad on Windows 10 (FIGURE 1):

- 1. Search for “Notepad” to access it quickly. Click on Notepad to open a new document window, and you’re ready to start typing. 1
- 2. Next, make the extensions visible. This step is not required to make HTML documents, but it will help make the file types clearer at a glance. Open the File Explorer, select the View tab, and then select the Options button on the right. In the Folder Options panel, select the View tab again. 2
- 3. Find “Hide extensions for known file types” and uncheck that option. 3
- 4. Click OK to save the preference 4, and the file extensions will now be visible

Figure1



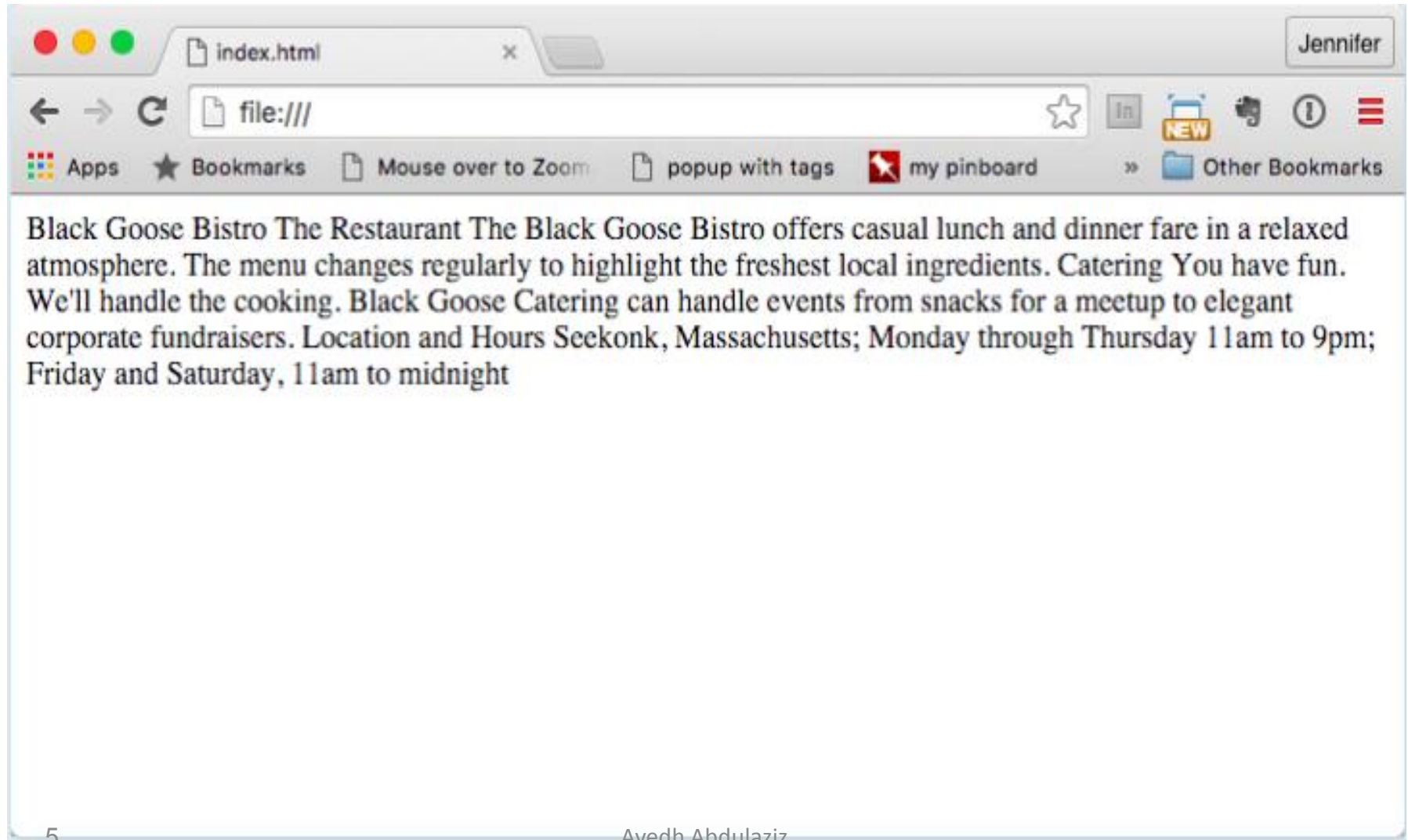
STEP 1: START WITH CONTENT

Now that we have our new document, it's time to get typing. A web page is all about content, so that's where we begin our demonstration:

Our page isn't looking so good (FIGURE 2). The text is all run together into one block—that's not how it looked when we typed it into the original document. There are a couple of lessons to be learned here. The first thing that is apparent is that the browser ignores line breaks in the source document. The sidebar "What Browsers Ignore" lists other types of information in the source document that are not displayed in the browser window.

Second, we see that simply typing in some content and naming the document .html is not enough. While the browser can display the text from the file, we haven't indicated the structure of the content. That's where HTML comes in. We'll use markup to add structure: first to the HTML document itself (coming up in Step 2), then to the page's content (Step 3). Once the browser knows the structure of the content, it can display the page in a more meaningful way

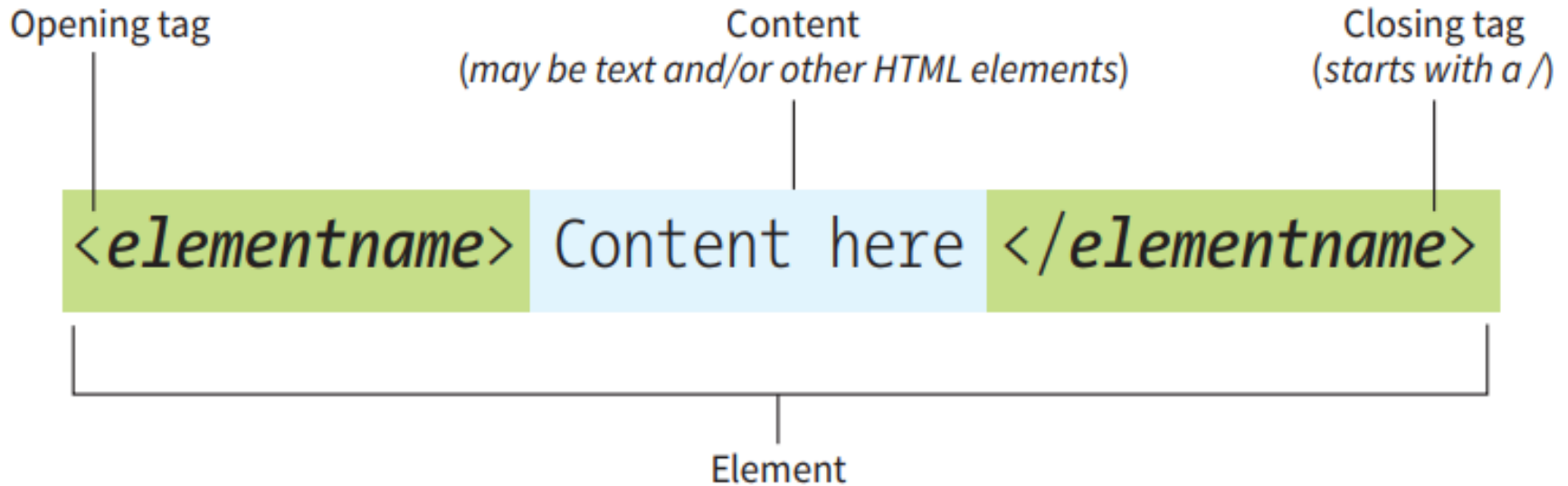
Figure2



STEP 2: GIVE THE HTML DOCUMENT STRUCTURE

- We have our content saved in an HTML document—now we're ready to start marking it up.
- The Anatomy of an HTML Element Back in presentation 2 you saw examples of elements with an opening tag (for a paragraph, for example) and a closing tag (). Before we start adding tags to our document, let's look at the anatomy of an HTML element (its syntax) and firm up some important terminology. A generic container element is labeled in FIGURE 3

Figure3



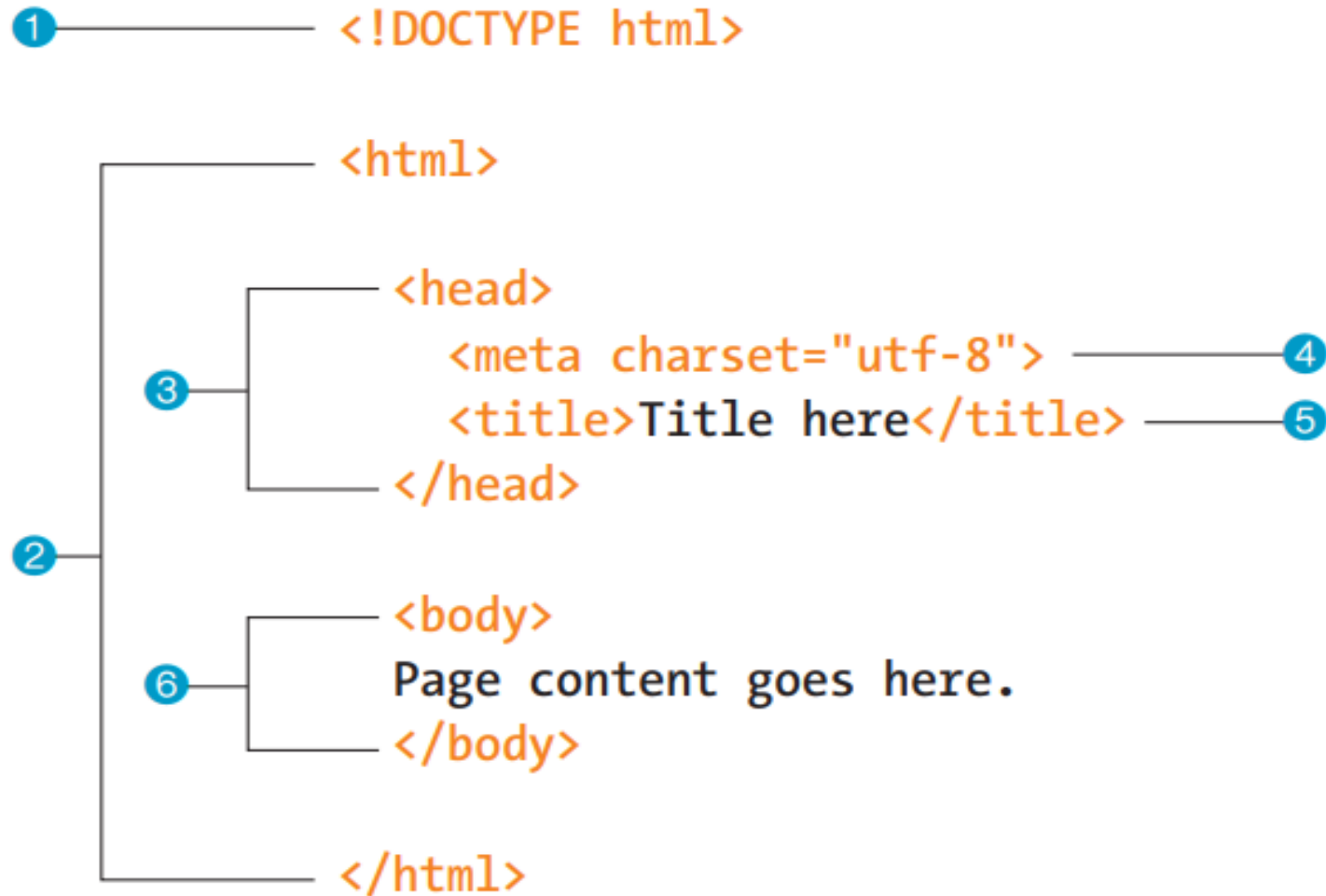
Example:

`<h1>Black Goose Bistro</h1>`

Basic Document Structure

- FIGURE 4 shows the recommended minimal skeleton of an HTML document. I say “recommended” because the only element that is required in HTML is the title. But I feel it is better, particularly for beginners, to explicitly organize documents into metadata (head) and content (body) areas. Let’s take a look at what’s going on in this minimal markup example.

figure4



Basic Document Structure1

I don't want to confuse things, but the first line in the example isn't an element at all. It is a document type declaration (also called DOCTYPE declaration) that lets modern browsers know which HTML specification to use to interpret the document. This DOCTYPE identifies the document as written in HTML5.

Meta elements provide document metadata, information about the document. In this case, it specifies the character encoding (a standardized collection of letters, numbers, and symbols) used in the document as Unicode version UTF-8 (see the sidebar "Introducing Unicode"). I don't want to go into too much detail on this right now, but know that there are many good reasons for specifying the charset in every document, so I have included it as part of the minimal document markup. Other types of metadata provided by the meta element are the author, keywords, publishing status, and a description that can be used by search engines.

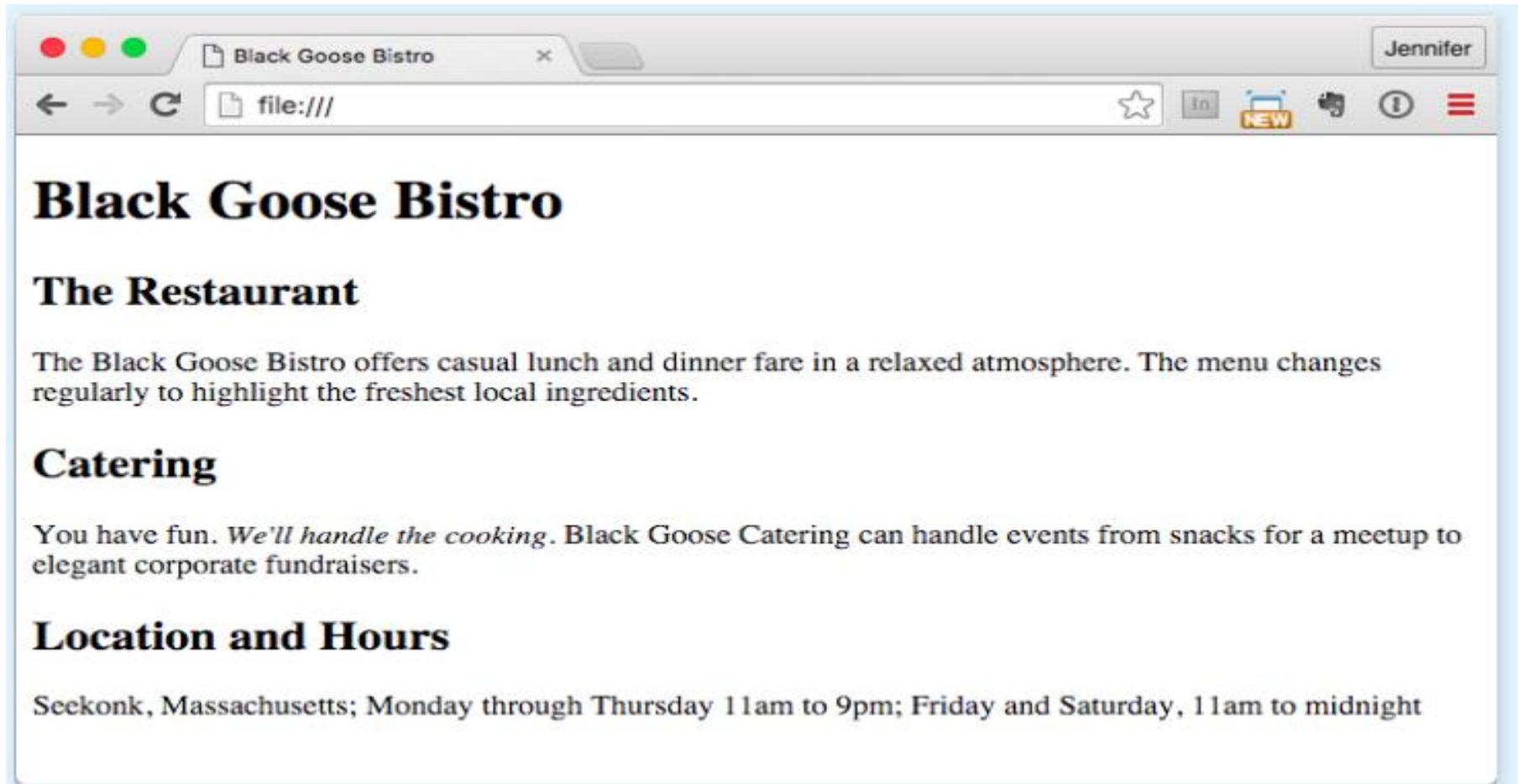
STEP 3: IDENTIFY TEXT ELEMENTS

With a little markup experience under your belt, it should be a no-brainer to add the markup for headings and subheads (h1 and h2), paragraphs (p), and emphasized text (em) to our content. However, before we begin, I want to take a moment to talk about what we're doing and not doing when marking up content with HTML

Defining text elements

1. Open the document `index.html` in your text editor, if it isn't open already. 2. The first line of text, "Black Goose Bistro," is the main heading for the page, so we'll mark it up as a Heading Level 1 (h1) element. Put the opening tag, `<h1>`, at the beginning of the line and the closing tag, `</h1>`, after it, like this: `<h1>Black Goose Bistro</h1>`
3. Our page also has three subheads. Mark them up as Heading Level 2 (h2) elements in a similar manner. I'll do the first one here; you do the same for "Catering" and "Location and Hours." `<h2>The Restaurant</h2>`
4. Each h2 element is followed by a brief paragraph of text, so let's mark those up as paragraph (p) elements in a similar manner. Here's the first one; you do the rest: `<p>The Black Goose Bistro offers casual lunch and dinner fare in a relaxed atmosphere. The menu changes regularly to highlight the freshest local ingredients.</p>`
5. Finally, in the Catering section, I want to emphasize that visitors should just leave the cooking to us. To make text emphasized, mark it up in an emphasis element (em) element, as shown here: `You have fun. We'll handle the cooking.` Black Goose Catering can handle events from snacks for a meetup to elegant corporate fundraisers.
6. Now that we've marked up the document, let's save it as we did before, and open (or reload) the page in the browser. You should see a page that looks much like the one in FIGURE 4. If it doesn't, check your markup to be sure that you aren't missing any angle brackets or a slash in a closing tag.

Figure4



Block and Inline Elements

Although it may seem like stating the obvious, it's worth pointing out that the heading and paragraph elements start on new lines and do not run together as they did before. That is because by default, headings and paragraphs display as block elements. Browsers treat block elements as though they are in little rectangular boxes, stacked up in the page. Each block element begins on a new line, and some space is also usually added above and below the entire element by default, the edges of the block elements are outlined in red.

STEP 4: ADD AN IMAGE

What fun is a web page with no images? In EXERCISE 2, we'll add an image to the page with the `img` element, Adding Images, but for now, they give us an opportunity to introduce two more basic markup concepts: empty elements and attributes.

Attributes

Attributes Let's get back to adding an image with the empty `img` element. Obviously, an tag is not very useful by itself—it doesn't indicate which image to use. That's where attributes come in. Attributes are instructions that clarify or modify an element. For the `img` element, the `src`(short for "source") attribute is required, and specifies the location (URL) of the image file. The syntax for an attribute is as follows:

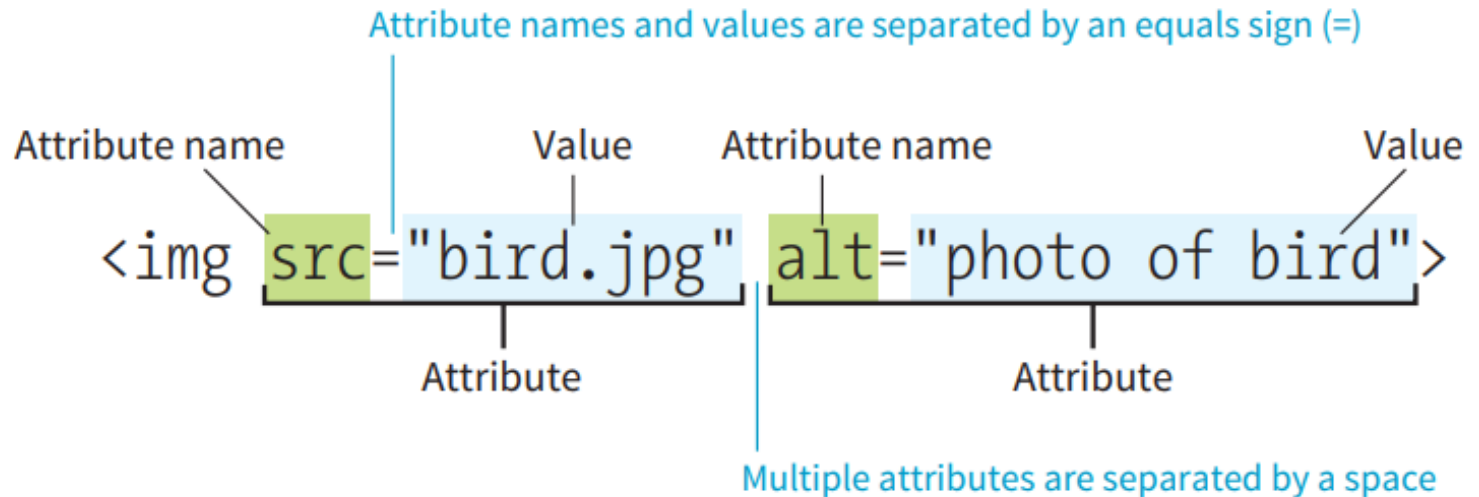
attribute name="value"

Attributes go after the element name, separated by a space. In non-empty elements, attributes go in the opening tag only: Content You can also put more than one attribute in an element in any order. Just keep them separated with spaces:

<element attribute1="value" attribute2="value">

EXERCISE 2

```
<h1><br>Black Goose  
Bistro</h1>
```



STEP 5: CHANGE THE LOOK WITH A STYLE SHEET

Once your content is ready to go (you've proofread it, right?) and you've added the markup to structure the document (, html, head, title, meta charset, and body), you are ready to identify the elements in the content. This lecture introduces the elements you have to choose from for marking up text.

There probably aren't as many of them as you might think, and really just a handful that you'll use with regularity. That said, this chapter is a big one and covers a lot of ground.

`<p>...</p>`Paragraph element

Paragraphs may contain text, images, and other inline elements (called phrasing content), but they may not contain headings, lists, sectioning elements, or any elements that typically display as blocks by default. HEADINGS `<h1>h1h2h3h4h5h6</h1>`

Example heading and paragraph

- This example shows the markup for four heading levels. Additional heading levels would be marked up in a similar manner.
- `<h1>Type Design</h1>`
- `<h2>Serif Typefaces</h2>`
- `<p>Serif typefaces have small slabs at the ends of letter strokes.`
- `In general, serif fonts can make large amounts of text easier to`
- `read.</p>`
- `<h3>Baskerville</h3>`
- `<h4>Description</h4>`
- `<p>Description of the Baskerville typeface.</p>`
- `<h4>History</h4>`
- `<p>The history of the Baskerville typeface.</p>`
- `<h3>Georgia</h3>`
- `<p>Description and history of the Georgia typeface.</p>`
- `<h2>Sans-serif Typefaces</h2>`
- `<p>Sans-serif typefaces do not have slabs at the ends of strokes.</p>`

THEMATIC BREAKS (HORIZONTAL RULE)

hr is an empty element—you just drop it into place where you want the thematic break to occur, as shown in this example and FIGURE 5:

- `<h3>Times</h3>`
- `<p>Description and history of the Times typeface.</p>`
- `<hr>`
- `<h3>Georgia</h3>`
- `<p>Description and history of the Georgia typeface.</p>`
- FIGURE 5

Times

Description and history of the Times typeface.

Georgia

Description and history of the Georgia typeface.

LISTS

Humans are natural list makers, and HTML provides elements for marking up three types of lists:

- **Unordered lists**

Collections of items that appear in no particular order

- **Ordered lists**

Lists in which the sequence of the items is important

- **Description lists**

Lists that consist of name and value pairs, including but not limited to terms and definitions

Unordered Lists

- `...`

Unordered list

- `...`

List item within an unordered list.

You can put any type of content element within a list item (li):

``

`Serif`

`Sans-serif`

`Script`

`Display`

`Dingbats`

``

- 
- Serif
 - Sans-serif
 - Script
 - Display
 - Dingbats

With style sheets, you can give the same unordered list many looks.

Serif
Sans-serif
Script
Display
Dingbats

 SERIF

 SANS-SERIF

 SCRIPT

 DISPLAY

 DINGBATS

Serif

Sans-serif

Script

Display

Dingbats

SERIF

SANS-SERIF

SCRIPT

DISPLAY

DINGBATS

Ordered Lists

- `...`

Ordered list

- `...`

List item within an ordered list

Ordered list elements must contain one or more list item elements, as shown in this example and in FIGURE 6:

``

`Gutenberg develops moveable type (1450s)`

`Linotype is introduced (1890s)`

`Photocomposition catches on (1950s)`

`Type goes digital (1980s)`

``

FIGURE 6

1. Gutenberg develops moveable type (1450s)
2. Linotype is introduced (1890s)
3. Photocomposition catches on (1950s)
4. Type goes digital (1980s)

OL1

If you want a numbered list to start at a number other than 1, you can use the start attribute in the ol element to specify another starting number, as shown here:

```
<ol start="17">  
  <li>Highlight the text with the text tool.</li>  
  <li>Select the Character tab.</li>  
  <li>Choose a typeface from the pop-up menu.</li>  
</ol>
```

The resulting list items would be numbered 17, 18, and 19, consecutively

Nesting Lists:

```
<ol>  
  <li></li>  
  <li>  
    <ul>  
      <li></li>  
      <li></li>  
      <li></li>  
    </ul>  
  </li>  
</ol>
```

Description Lists

Here is an example of a list that associates forms of typesetting with their descriptions (FIGURE 6):

<dl>

<dt>Linotype</dt>

<dd>Line-casting allowed type to be selected, used, then recirculated into the machine automatically. This advance increased the speed of typesetting and printing dramatically.</dd>

<dt>Photocomposition</dt>

<dd>Typefaces are stored on film then projected onto photo-sensitive paper. Lenses adjust the size of the type.</dd>

<dt>Digital type</dt>

<dd><p>Digital typefaces store the outline of the font shape in a format such as Postscript. The outline may be scaled to any size for output.</p>

<p>Postscript emerged as a standard due to its support of graphics and its early support on the Macintosh computer and Apple laser printer.</p>

</dd>

</dl>

Figure6

Linotype

Line-casting allowed type to be selected, used, then recirculated into the machine automatically. This advance increased the speed of typesetting and printing dramatically.

Photocomposition

Typefaces are stored on film then projected onto photo-sensitive paper. Lenses adjust the size of the type.

Digital type

Digital typefaces store the outline of the font shape in a format such as Postscript. The outline may may be scaled to any size for output.

Postscript emerged as a standard due to its support of graphics and its early support on the Macintosh computer and Apple laser printer.

Figure

The figure element identifies content that illustrates or supports some point in the text. A figure may contain an image, a video, a code snippet, text, or even a table—pretty much anything that can go in the flow of web content.

```
<figure>...</figure>
```

Related image or resource

```
<figcaption>...</figcaption>
```

Text description of a figure

```
<figure>
```

```

```

```
<figcaption>Fig.1 - Trulli, Puglia, Italy.</figcaption>
```

```
</figure>
```

The HTML <picture> Element

The HTML <picture> element gives web developers more flexibility in specifying image resources.

The <picture> element contains one or more <source> elements, each referring to different images through the src set attribute. This way the browser can choose the image that best fits the current view and/or device.

Each <source> element has a media attribute that defines when the image is the most suitable.

Example

Show different images for different screen sizes:

```
<picture>
```

```
<source media="(min-width: 650px)"  
srcset="img_food.jpg">
```

```
<source media="(min-width: 465px)"  
srcset="img_car.jpg">
```

```

```

```
</picture>
```

HTML:BLOCK&& INLINE

- **The block-level elements :**

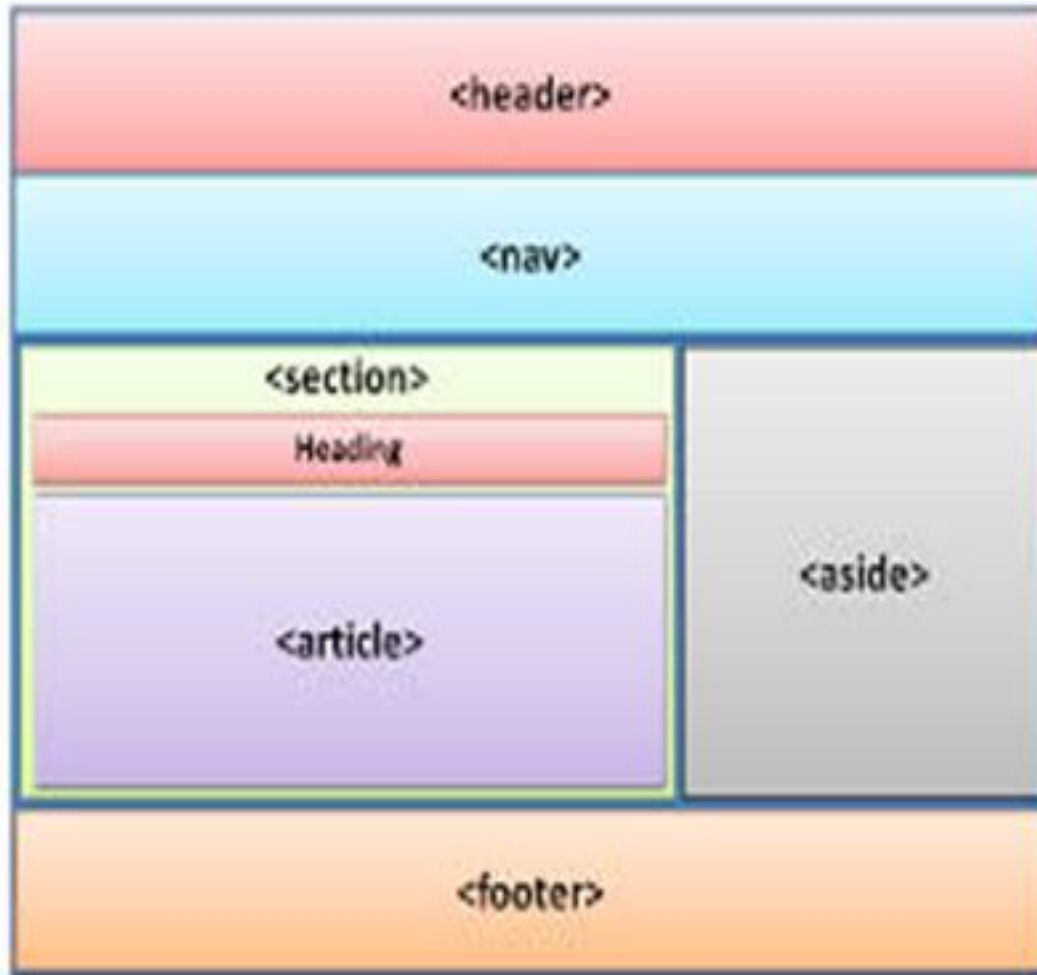
<address><article><aside><blockquote><canvas><dd><div><dl><dt><fieldset><figcaption><figure><footer><form><h1><h6><header><hr><main>
<nav><noscript><p><pre><section><table>
<tfoot>

- **Inline Elements:**

<a><abbr><acronym><bdo><big>
<button><cite><code><dfn><i><input><kbd><label><map><object><output><q><samp><script><select><small><sub><sup><textarea><time><tt><var>

ORGANIZING PAGE CONTENT

So far, the elements we've covered handle very specific tidbits of content: a paragraph, a heading, a figure, and so on. Prior to HTML5, there was no way to group these bits into larger parts other than wrapping them in a generic division (div) element (I'll cover div in more detail later). HTML5 introduced new elements that give semantic meaning to sections of a typical web page or application (see Note), including main content (main), headers (header), footers (footer), sections (section), articles (article), navigation (nav), and tangentially related or complementary content (aside). Curiously, the spec lists the old address element as a section as well, so we'll look at that one here too.



Main Content

Web pages these days are loaded with different types of content: mastheads, sidebars, ads, footers, more ads, even more ads, and so on.

```
<main>...</main>
```

Primary content area of page or app

In other words, headers, sidebars, and other elements that appear across multiple pages in a site should not be included in the main section:

```
<body>
```

```
<header>...</header>
```

```
<main>
```

```
<h1>Humanist Sans Serif</h1>
```

```
<!-- code continues -->
```

```
</main>
```

```
</body>
```

footer, header

`<header>...</header>`

Introductory material for page, section, or article

`<footer>...</footer>`

Footer for page, section, or article. In the following example, the document header includes a logo image, the site title, and navigation:

```
<header>
```

```
<h1>Nuts about Web Fonts</h1>
```

```
<nav><ul> <li><a href="/">Home</a></li>
```

```
  <li><a href="/">Blog</a></li>
```

```
<li><a href="/">Shop</a></li>
```

```
</ul> </nav></header></body>
```

Navigation and Aside (Sidebars)

The aside element identifies content that is separate from, but tangentially related to, the surrounding content.

`<aside>...</aside>` Tangentially related material.

`<nav>...</nav>` Primary navigation links.

The nav element gives developers a semantic way to identify navigation for a site.

```
<nav> <ul>
```

```
<li><a href="/">Serif</a></li>
```

```
<li><a href="/">Sans-serif</a></li>
```

```
<li><a href="/">Script</a></li>
```

```
<li><a href="/">Display</a></li>
```

```
<li><a href="/">Dingbats</a></li></ul></nav>
```

Addresses

`<address>...</address>` Contact information.

Last, and well, least, is the address element that is used to create an area for contact information for the author or maintainer of the document.

`<address>`

Contributed by `Jennifer Robbins`,

`O'Reilly Media`

`</address>`

Contributed by [Jennifer Robbins](#), [O'Reilly Media](#)

Dates and times

The time element allows us to mark up dates and times in a way that is comfortable for a human to read, but also encoded in a standardized way that computers can use . The **time** element indicates dates, times, or date-time combos.

The **datetime** attribute specifies the date and/or time information in a standardized time format illustrated in FIGURE 7

+ or – hours ahead or behind Greenwich Mean Time

A “T” always precedes time information

Here are a few examples of valid values for datetime:

Time only: 9:30 p.m.

```
<time datetime="21:30">9:30p.m.</time>
```

Date only: June 19, 2016

```
<time datetime="2016-06-19">June 19, 2016</time>
```

Date and time: Sept. 5, 1970, 1:11a.m.

```
<time datetime="1970-09-05T01:11:00">Sept. 5, 1970,  
1:11a.m.</time>
```

Date and time, with time zone information: 8:00am on July 19, 2015, in Providence, RI

```
<time datetime="2015-07-19T08:00:00-05:00">July  
19, 2015, 8am, Providence RI</time>
```

GENERIC ELEMENTS (DIV AND SPAN)

What if none of the elements we've talked about so far accurately describes your content? After all, there are endless types of information in the world, but as you've seen, not all that many semantic elements.

`<div>...</div>`

Generic block-level element

`...`

Generic inline element

We're going to spend a little time on `div` and `span` elements, as well as the **id** and **class attributes**, to learn how authors use them to structure content.

Example

In this example, a div element is used as a container to group an image and two paragraphs into a product “listing”:

```
<div class="listing">
```

```

```

```
<p><cite>The Complete Manual of Typography</cite>, James  
Felici</p>
```

```
<p>A combination of type history and examples of good and bad type  
design.</p> </div>
```

```
EXP2:ID <div id="news">
```

```
<h1>New This Week</h1>
```

```
<p>We've been working on...</p>
```

```
<p>And last but not least,... </p>
```

```
</div>
```

Define a Phrase with span

In this example, each telephone number is marked up as a span and classified as “tel”:

```
<ul>  
<li>John: <span class="tel">999.8282</span></li>  
  <li>Paul: <span class="tel">888.4889</span></li>  
<li>George: <span class="tel">888.1628</span></li>  
<li>Ringo: <span class="tel">999.3220</span></li>  
</ul>
```

- John: 999.8282
- Paul: 888.4889
- George: 888.1628
- Ringo: 999.3220

ADDING LINKS

If you're creating a page for the web, chances are you'll want to link to other web pages and resources, whether on your own site or someone else's. Linking, after all, is what the web is all about. In this chapter, we'll look at the markup that makes links work—links to other sites, to your own site, and within a page. There is one element that makes linking possible: the anchor (a).

`<a>...` Anchor element (hypertext link)

Example

Here is an example that creates a link to the O'Reilly Media site: `Go to the O'Reilly Media site` To make an image a link, simply put the `img` element in the anchor element: ``

By the way, you can put any HTML content element in an anchor to make it a link, not just images.

THE HREF ATTRIBUTE

Opening anchor tag

```
<a href="https://www.amazon.com/Bequet-Gourmet-Caramel-24oz-Celtic/dp/B00GZEU10Y/ref=sr_1_1_a_it?ie=UTF8&qid=1467055107&sr=8-1&keywords=bequet">Bequet Caramels</a>
```

URL

Linked text

Closing anchor tag

Ayedh Abdulaziz

LINKING TO PAGES ON THE WEB

```
<ul> <li><a  
href="http://www.foodnetwork.com/">The  
Food Network</a></li>  
  <li>Epicurious</li> </ul>
```

[The Food Network](http://www.foodnetwork.com/)

Epicurious

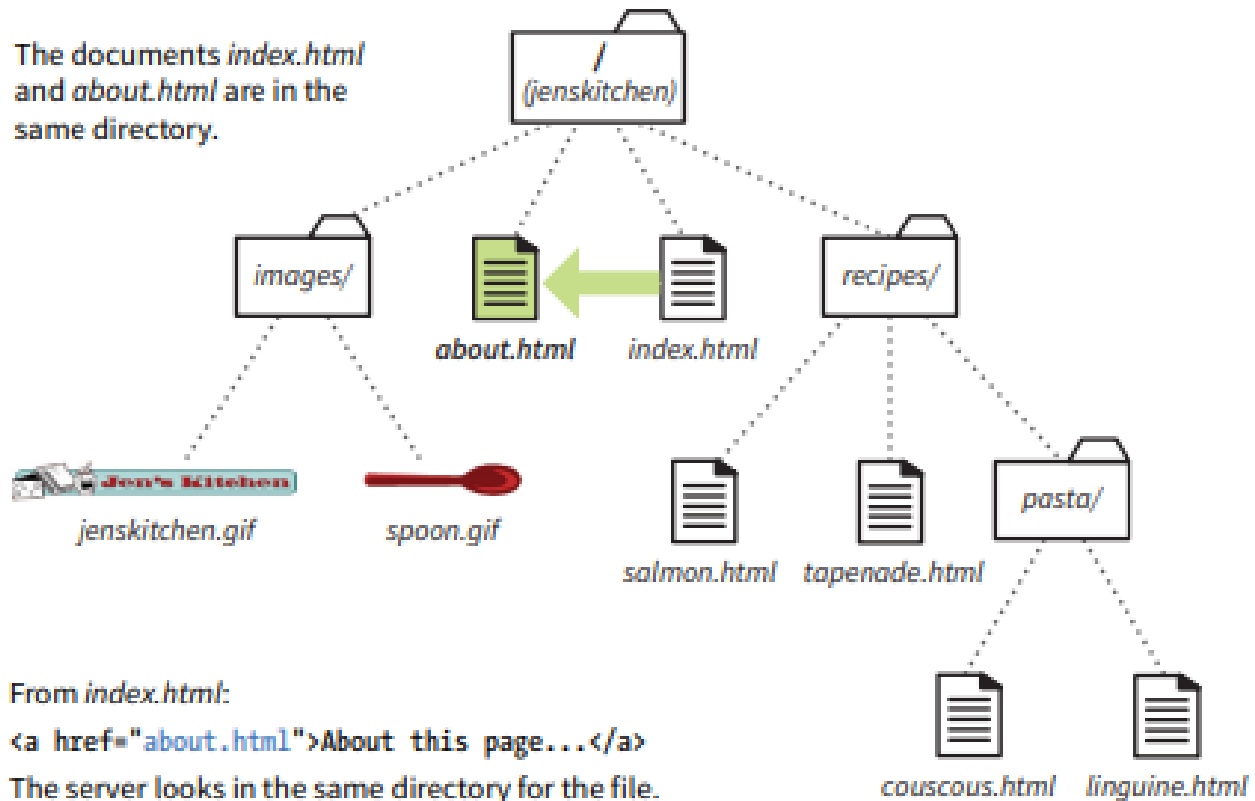
LINKING WITHIN YOUR OWN SITE

```
<a href="about.html">About the site...</a>
```

[About the site...](#)

Figure

The documents *index.html* and *about.html* are in the same directory.



Linking to a Specific Point in a Page

Step 1: Identifying the destination

Identify the destination by using the id attribute.

```
<h2 id="startH">H</h2>
```

```
<dl>
```

```
<dt>hexadecimal</dt>
```

Step 2: Linking to the destination

```
<p>... F | G | <a href="#startH">H</a> | I | J ...</p> ... F  
| G | H | I | J ...
```

Create a link to the destination. The # before the name is necessary to identify this as a fragment and not a filename. ... | F | G | [H](#) | I | J ...

Linking to a Fragment in Another Document

You can link to a fragment in another document by adding the fragment name to the end of the URL (absolute or relative). For example, to make a link to the “H” heading of the glossary page from another document in that directory, the URL would look like this: `See the Glossary, letter H`

You can even link to specific destinations in pages on other sites by putting the fragment identifier at the end of an absolute URL, like so: `See the Glossary, letter H`

HTML Links - The target Attribute

The target attribute specifies where to open the linked document.

The target attribute can have one of the following values:

- `_self` - Default. Opens the document in the same window/tab as it was clicked
- `_blank` - Opens the document in a new window or tab
- `_parent` - Opens the document in the parent frame
- `_top` - Opens the document in the full body of the window

```
<a href="https://www.w3schools.com/" target="_blank">  
Visit W3Schools!</a>
```

Link to an Email Address:

```
<a href="mailto:someone@example.com">Send email</a>
```

Make an external link

```
<ul>  
<li><a href="http://www.foodnetwork.com/">The Food  
Network</a></li>  
<li>Epicurious</li>  
</ul>
```

[The Food Network](#)

Epicurious

A sample mailto link is shown here:

```
<a href="mailto:alklecker@example.com">Contact Al Klecker</a>
```

TELEPHONE LINKS

```
<a href="tel:+01-800-555-1212">Call us free at (800) 555-1212</a>
```

[Call us free at \(800\) 555-1212](#)

Link Titles and Button as a Link

To use an HTML button as a link, you have to add some JavaScript code.

JavaScript allows you to specify what happens at certain events, such as a click of a button:

Example

```
<button onclick="document.location='default.asp'">HTML  
Tutorial</button>
```

Link Titles

```
<a href="https://www.w3schools.com/html/" title="Go to  
W3Schools HTML section">Visit our HTML Tutorial</a>
```

Create a Bookmark in HTML

First, use the id attribute to create a bookmark:

```
<h2 id="C4">Chapter 4</h2>
```

Then, add a link to the bookmark ("Jump to Chapter 4"), from within the same page:

- Example

```
<a href="#C4">Jump to Chapter 4</a>
```

You can also add a link to a bookmark on another page:

```
<a href="html_demo.html#C4">Jump to Chapter 4</a>
```

Linking to a Higher Directory

```
<p><a href="../index.html">[Back to home  
page]</a></p> \[Back to home page\]
```

TABLE MARKUP

HTML tables were created for instances when you need to add tabular material (data arranged into rows and columns) to a web page.

`<table>...</table>`

Tabular content (rows and columns)

`<tr>...</tr>`

Table row

`<th>...</th>`

Table header

`<td>...</td>`

Table cell data

Menu item	Calories	Fat
Chicken noodle soup	120	2
Caesar salad	400	26

Table

```
<table>
  <tr>
    <th>Menu item</th>
    <th>Calories</th>
    <th>Fat (g)</th>
  </tr>
  <tr>
    <td>Chicken noodle soup</td>
    <td>120</td>
    <td>2</td>
  </tr>
  <tr>
    <td>Caesar salad</td>
    <td>400</td>
    <td>26</td>
  </tr></table>
```

Menu item	Calories	Fat (g)
Chicken noodle soup	120	2
Caesar salad	400	26

HTML Table - Cell that Spans Many Rows

To make a cell span more than one row, use the rowspan attribute:

```
<table style="width:100%">
  <tr>
    <th>Name:</th>
    <td>Bill Gates</td>
  </tr>
  <tr>
    <th rowspan="2">Telephone:</th>
    <td>55577854</td>
  </tr>
  <tr>
    <td>55577855</td>
  </tr>
</table>
```

Name:	Bill Gates
Telephone:	55577854
	55577855

HTML Table - Cell that Spans Many Columns

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th colspan="2">Telephone</th>
  </tr>
  <tr>
    <td>Bill Gates</td>
    <td>55577854</td>
    <td>55577855</td>
  </tr>
</table>
```

Name	Telephone	
Bill Gates	55577854	55577855

HTML Table - Add a Caption

To add a caption to a table, use the <caption> tag

```
<table style="width:100%">
  <caption>Monthly savings</caption>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>February</td>
    <td>$50</td>
  </tr>
</table><thead>...</thead>
```

Table header row group

```
<tbody>...</tbody>
```

Table body row group

```
<tfoot>...</tfoot>
```

Table footer row group

The colgroup element

- <table>
- <colgroup>
- <col span="2" style="background-color:red">
- <col style="background-color:yellow">
- </colgroup>
- <tr>
- <th>ISBN</th>
- <th>Title</th>
- <th>Price</th>
- </tr>
- <tr>
- <td>3476896</td>
- <td>My first HTML</td>
- <td>\$53</td>
- </tr>
- <tr>
- <td>5869207</td>
- <td>My first CSS</td>
- <td>\$49</td>
- </tr>
- </table>

ISBN	Title	Price
3476896	My first HTML	\$53
5869207	My first CSS	\$49

HTML Iframe Syntax

The HTML <iframe> tag specifies an inline frame.

An inline frame is used to embed another document within the current HTML document.

```
<iframe src="url" title="description">
```

Tip: It is a good practice to always include a title attribute for the <iframe>. This is used by screen readers to read out what the content of the iframe is.

```
<iframe src="demo_iframe.htm" height="200" width="300" title="Iframe Example"></iframe>
```

Image Maps

The HTML <map> tag defines an image map. An image map is an image with clickable areas. The areas are defined with one or more <area> tags

```

```

```
<map name="workmap">  
  <area shape="rect" coords="34,44,270,350" alt="Computer"  
  href="computer.htm">  
  <area shape="rect" coords="290,172,333,250" alt="Phone"  
  href="phone.htm">  
  <area shape="circle" coords="337,300,44" alt="Coffee" href  
  ="coffee.htm">  
</map>
```

Shape

You must define the shape of the clickable area, and you can choose one of these values:

- rect - defines a rectangular region
- circle - defines a circular region
- poly - defines a polygonal region
- default - defines the entire region

You must also define some coordinates to be able to place the clickable area onto the image.

•

FORMS

HOW FORMS WORK There are two parts to a working form. The first part is the form that you see on the page itself that is created using HTML markup. Forms are made up of buttons, input fields, and drop-down menus (collectively known as form controls) used to collect information from the user. Forms may also contain text and other elements.

`<form>...</form>`

Interactive form

MAILING LIST SIGNUP

Get news about the band such as tour dates and special MP3 releases sent to your own in-box.

Name:

Email:

Name = Sally Strongarm
Email = strongarm@example.com

Data


Web application
(stores data in database)


Response
(HTML)

THANKS

You are now on the band mailing list. Can't wait to see you at the shows.

[Go back to the main page](#)

Example

- <!DOCTYPE html>
- <html>
- <head>
- <title>Mailing List Signup</title>
- <meta charset="utf-8">
- </head>

<body>

<h1>Mailing List Signup</h1>

<form action="/mailinglist.php" method="POST">

<fieldset>

<legend>Join our email list</legend>

<p>Get news about the band such as tour dates and special MP3 releases sent to your own in-box.</p>

<label for="firstlast">Name:</label>

<input type="text" name="fullname" id="firstlast">

<label for="email">Email:</label>

<input type="text" name="email" id="email">

<input type="submit" value="Submit">

</fieldset>

</form>

</body></html>

The Name Attribute for <input> Notice that each input field must have a name attribute to be submitted. If the name attribute is omitted, the value of the input field will not be sent at all.

```
<form action="/action_page.php">  
  <label for="fname">First name:</label><br>  
  <input type="text" id="fname" value="John">  
<br><br>  
  <input type="submit" value="Submit">  
</form>
```

The action Attribute

The action attribute provides the location (URL) of the application or script that will be used to process the form. The action attribute in this example sends the data to a script called mailinglist.php:

```
<form action="/mailinglist.php"  
method="POST">...</form>
```

The method Attribute

The method attribute specifies how the information should be sent to the server. Let's use this data gathered from the sample form as an example.

fullname = Sally Strongarm

Email = strongarm@example.com

The GET method With the GET method, the encoded form data gets tacked right onto the URL sent to the server. A question mark character separates the URL from the following data, as shown here: [get](#)

<http://www.bandname.com/maillinglist.php?name=Sally+Strongarm&email=strongarm%40example.com>

The POST method

When the form's method is set to POST, the browser sends a separate server request containing some special headers followed by the data. In theory, only the server sees the content of this request, and thus it is the best method for sending secure information such as a home address or other personal information.

The Target Attribute

- The target attribute specifies where to display the response that is received after submitting the form.
- The target attribute can have one of the values in **slide 48**

The Autocomplete Attribute

The autocomplete attribute specifies whether a form should have autocomplete on or off.

When autocomplete is on, the browser automatically complete values based on values that the user has entered before. `<form action="/action_page.php" autocomplete="on">`

The name Attribute

The name attribute provides the variable name for the control. In this example, the text gathered by a textarea element is defined as the “comment” variable:

```
<textarea name="comment" rows="4" cols="45" placeholder="Leave us a comment."></textarea>
```

THE GREAT FORM CONTROL ROUNDUP

- This is the fun part—playing with the markup that adds form controls to the page. This section introduces the elements used to create the following:
- Text-entry controls
- Specialized text-entry controls
- Submit and reset buttons
- Radio and checkbox buttons
- Pull-down and scrolling menus
- File selection and upload control
- Hidden controls
- Dates and times
- Numerical controls
- Color picker control

Black Goose Bistro | Pizza-on-Demand

Our 12" wood-fired pizzas are available for delivery. Build your custom pizza and we'll deliver it within an hour.

Your Information

Name:

Address:

Telephone Number:

Email:

Delivery instructions:

No more than 400 characters long.

Design Your Dream Pizza:

Pizza specs

Crust (*Choose one*):

- ☐ Classic white
- ☐ Multigrain
- ☐ Cheese-stuffed crust
- ☐ Gluten-free

Toppings (*Choose as many as you want*):

- ☒ Red sauce
- ☐ White sauce
- ☐ Mozzarella Cheese
- ☐ Pepperoni
- ☐ Mushrooms
- ☐ Peppers
- ☐ Anchovies

Number

How many pizzas:

Size, Maxlength, Minlength, placeholder

```
<li><label>Favorite color: <input type="text"
name="favcolor" value="Red" maxlength="50"
placeholder="50 words or less"></label></li>
```

The size

The size attribute specifies the length of the input field in number of visible characters.

Other Attribute

<input type="search">

Search field

<input type="email">

Email address

<input type="tel">

Telephone number

<input type="url">

Location (URL)

<input type="password">

Password text control

<input type="submit">

Submits the form data to the server

<input type="reset">

Resets the form controls to their default settings

<input type="image">

Image buttons

<input type="button">

Custom input button

<button>...</button>

The button element

Drop-Down Suggestions

`<datalist>...</datalist>` Drop-down menu input

Drop-Down Suggestions

The `datalist` element allows the author to provide a drop-down menu of suggested values for any type of text input.

Within the `datalist` element, suggested values are marked up as `option` elements. Use the `list` attribute in the input element to associate it with the id of its respective `datalist`.

In the following example ,a `datalist` suggests several education level options for a text input:

`<p>Education completed: <input type="text" list="edulevel" name="education">`

`<datalist id="edulevel">`

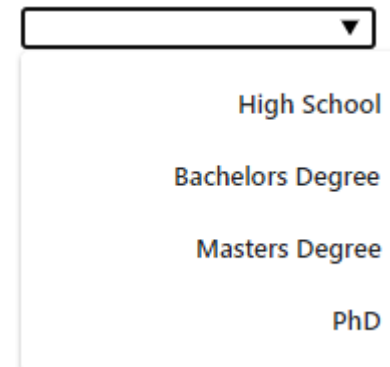
`<option value="High School">`

`<option value="Bachelors Degree">`

`<option value="Masters Degree">`

`<option value="PhD">`

`</datalist>`



A visual representation of a web form. It features a text input field with a dropdown arrow. The dropdown menu is open, showing four suggestions: "High School", "Bachelors Degree", "Masters Degree", and "PhD".

Radio buttons and Checkbox

Radio buttons are added to a form via the input element with the type attribute set to “radio.” Here is the syntax for a minimal radio button:

```
<input type="radio" name="variable" value="value">
```

```
<p>How old are you?</p>
```

```
<ol> <li><input type="radio" name="age" value="under24" checked>  
under 24</li>
```

```
<li><input type="radio" name="age" value="25-34"> 25 to 34</li>
```

```
<li><input type="radio" name="age" value="35-44"> 35 to 44</li>
```

```
<li><input type="radio" name="age" value="over45"> 45+</li>
```

```
</ol>
```

Radio buttons (`input type="radio"`) Checkboxes (`input type="checkbox"`)

How old are you?

- ☒ under 24
- ☐ 25 to 34
- ☐ 35 to 44
- ☐ 45+

What type of music do you listen to?

- ☒ Punk rock
- ☒ Indie rock
- ☐ Hip Hop
- ☐ Rockabilly

Menu

- `<select>...</select>`

Menu control

- `<option>...</option>`

An option within a menu

- `<optgroup>...</optgroup>`

A logical grouping of options within a menu

`<p>What is your favorite 80s band?`

`<select name="EightiesFave" size="6" multiple>`

`<option>The Cure</option>`

`<option>Cocteau Twins</option>`

`<option>Tears for Fears</option>`

`<option>Thompson Twins</option>`

`<option value="EBTG">Everything But the Girl</option>`

`<option>Depeche Mode</option>`

`<option>The Smiths</option>`

`<option>New Order</option>`

`</select>`

`</p>` simply specify the number of lines you'd like to be visible using the **size** attribute

The **multiple** attribute allows users to make more than one selection from the scrolling list.

Grouping menu options

You can use the `optgroup` element to create conceptual groups of options. The required `label` attribute provides the heading for the group (see Note). FIGURE shows how option groups are rendered in modern browsers.

```
<select name="icecream" size="7" multiple>
```

```
  <optgroup label="traditional">
```

```
    <option>vanilla</option>
```

```
    <option>chocolate</option>
```

```
  </optgroup>
```

```
  <optgroup label="fancy">
```

```
    <option>Super praline</option>
```

```
    <option>Nut surprise</option>
```

```
    <option>Candy corn</option>
```

```
  </optgroup>
```

```
</select>
```



Type="file"

```
<form action="/client.php" method="POST"
enctype="multipart/form-data">
```

```
<label>Send a photo to be used as your online
icon <em>(optional) </em><br>
```

```
<input type="file" name="photo"></label>
</form>
```

Send a photo to be used as your online icon (*optional*):

Choose File No file chosen

The <output> Element

The <output> element represents the result of a calculation (like one performed by a script). Example

Perform a calculation and show the result in an <output> element:

```
<form action="/action_page.php"
  oninput="x.value=parseInt(a.value)+parseInt(b.value)">
  0
  <input type="range" id="a" name="a" value="50">
  100 +
  <input type="number" id="b" name="b" value="50">
  =
  <output name="x" for="a b"></output>
  <br><br>
  <input type="submit">
</form>
```

`<input type="date">`

Date input control

`<input type="time">`

Time input control

`<input type="datetime-local">`

Date/time control

`<input type="month">`

Specifies a month in a year

`<input type="week">`

Specifies a particular week in a year

The new date- and time-related input types are as follows:

<input type="date" name="name" value="2017-01-14">

Creates a date input control, such as a pop-up calendar, for specifying a date (year, month, day). The initial value must be provided in ISO date format (YYYY-MM-DD).

<input type="time" name="name" value="03:13:00">

Creates a time input control for specifying a time (hour, minute, seconds, fractional sections) with no time zone indicated. The value is provided as hh:mm:ss.

<input type="datetime-local" name="name" value="2017-01-14T03:13:00">

Creates a combined date/time input control with no time zone information (YYYY-MM-DDThh:mm:ss)

<input type="month" name="name" value="2017-01">

Creates a date input control that specifies a particular month in a year (YYYY-MM).

<input type="week" name="name" value="2017-W2">

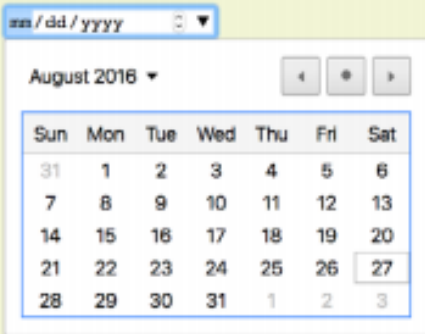
Creates a date input control for specifying a particular week in a year using an ISO week numbering format (YYYY-W#).

input type="time"



12:06 -- + -

input type="date"

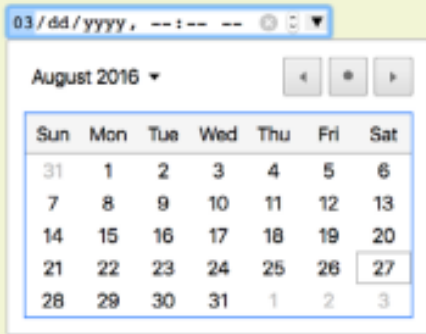


mm/dd/yyyy

August 2016

Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	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

input type="datetime-local"

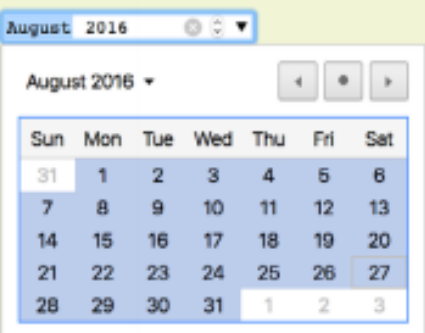


03/mm/dd/yyyy, --:--

August 2016

Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	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

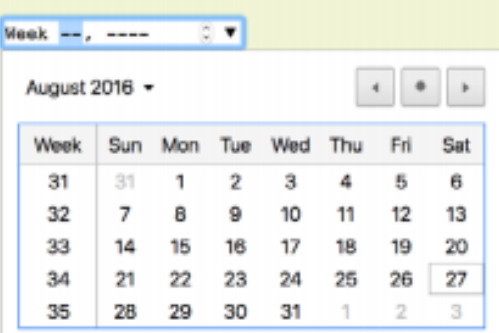
input type="month"



August 2016

Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	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

input type="week"



Week --, ----

August 2016

Week	Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	31	1	2	3	4	5	6
32	7	8	9	10	11	12	13
33	14	15	16	17	18	19	20
34	21	22	23	24	25	26	27
35	28	29	30	31	1	2	3

Numerical Inputs

```
<input type="number">
```

Number input

```
<input type="range">
```

Slider input

```
<label>Number of guests <input type="number" name="guests"
min="1"max="6"></label>
```

```
<label>Satisfaction (0 to 10) <input type="range"
name="satisfaction" min="0" max="10" step="1"></label>
```

```
input type="number"
```

Number of guests:

```
input type="range"
```

Satisfaction (from 0 to 10):

Ayedh Abdulaziz

Color picker

Non-supporting browsers—currently all versions of IE, iOS Safari, and older versions of Android—display the default text input instead.

```
<label>Your favorite color: <input type="color"  
name="favorite"> </label>
```



Multimedia

In the early days, web browsers were extremely limited in what they were able to render, so they relied on plug-ins to help them display media that they couldn't handle natively. Java applets, Flash movies, RealMedia (an old web video and audio format), and other media required third-party plug-ins in order to be played in the browser. Heck, even JPEG images once required a plug-in to display.

`<object>...</object>`

Represents external resource

`<param>` Parameters of an object

`<embed></embed>`

``

Example

Embed is an empty element that points to an external resource with the src attribute:

```
<embed type="video/quicktime" src="movies/hekboy.mov"
width="320" height="256">
```

```
<object type="video/quicktime" data="movies/hekboy.mov"
width="320"
height="256">
```

```
<param name="autostart" value="false">
```

```
<param name="controller" value="true">
```

```
</object>
```

How Media Formats Work

- MPEG-4 container + H.264 video codec + AAC audio codec
- WebM container + VP8 video codec + Vorbis audio codec.
- Ogg container + Theora video codec + Vorbis audio codec and mp3

How Media Formats Work

- MPEG-4 container + H.264 video codec + AAC audio codec
- WebM container + VP8 video codec + Vorbis audio codec.
- Ogg container + Theora video codec + Vorbis audio codec and mp3

Adding a Video to a Page

```
<video>...</video>
```

Adds a video player to the page

```
<video src="highlight_reel.mp4" width="640"  
height="480"
```

```
poster="highlight_still.jpg" controls autoplay>
```

Your browser does not support HTML5 video. Get
the [MP4 video](highlight_reel.mp4)

```
</video>
```

Attribute preload, loop, muted

Example

```
<video id="video" controls  
poster="img/poster.jpg">  
<source src="clip.webm" type="video/webm">  
<source src="clip.mp4" type="video/mp4">  
<source src="clip.ogv" type="video/ogg">  
<a href="clip.mp4">Download the MP4 of the  
clip.</a>  
</video>
```

Adding Audio to a Page

<p>Play "Percussion Gun" by White Rabbits</p>

<audio id="whiterabbits" controls preload="auto">

<source src="percussiongun.mp3" type="audio/mp3">

<source src="percussiongun.ogg" type="audio/ogg">

<source src="percussiongun.webm" type="audio/webm">

<p>Download "Percussion Gun":</p>

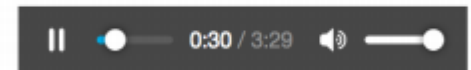
MP3

Ogg Vorbis

</audio>

<audio src="jetfighter.mp3" autoplay loop></audio>

Play "Percussion Gun" by White Rabbits



INTRODUCING CASCADING STYLE SHEETS

You've heard style sheets mentioned quite a bit already, and now we'll finally put them to work and start giving our pages some much-needed style. Cascading Style Sheets (CSS) is the W3C standard for defining the presentation of documents written in HTML, and in fact, any XML language. Presentation, again, refers to the way the document is delivered to the user, whether shown on a computer screen, displayed on a cell phone, printed on paper, or read aloud by a screen reader.

Benefit css:

- Precise type and layout controls
- More accessible sites
- Less work.

Writing the Rules

The following example contains two rules. The first makes all the h1 elements in the document green; the second specifies that the paragraphs should be in a large, sans-serif font. Sans-serif fonts do not have a little slab (a serif) at the ends of strokes and tend to look more sleek and modern.

```
h1 { color: green; }
```

```
p { font-size: large; font-family: sans-serif; }
```

declaration

selector { property: value; }

declaration block

```
selector {  
  property1: value1;  
  property2: value2;  
  property3: value3;  
}
```

Attaching the Styles to the Document

- External style sheets

```
<link rel="stylesheet" href="assets/css/main.css">
```

- Embedded style sheets

```
<style>
```

```
h1 {  
  color: green;  
}
```

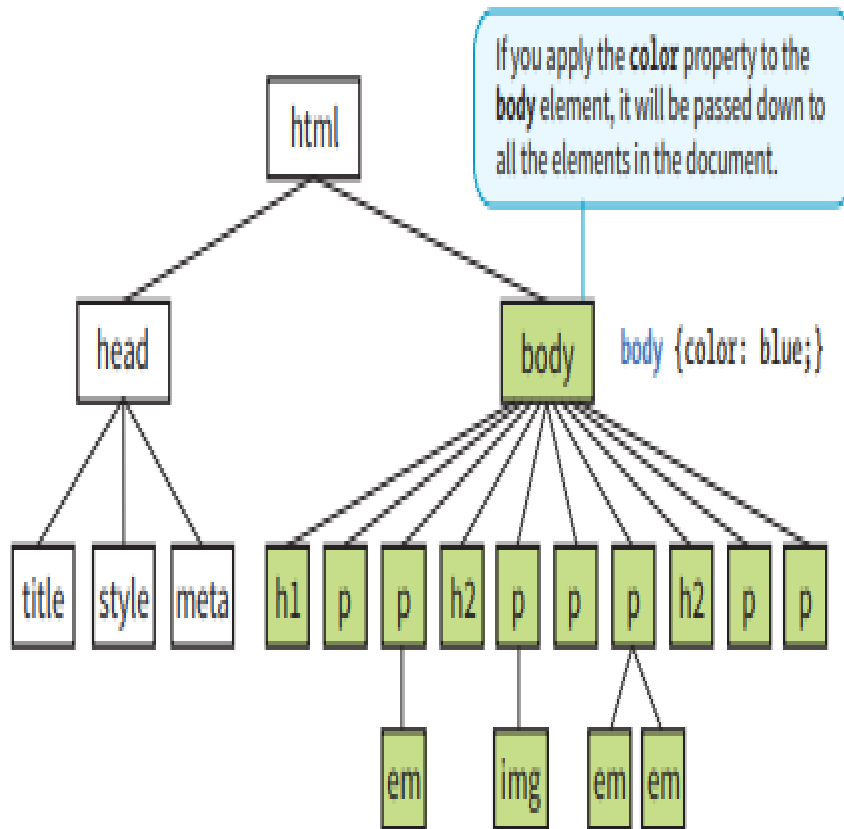
```
p {  
  font-size: large;  
  font-family: sans-serif;  
  text-align: center;  
}
```

```
</style>
```

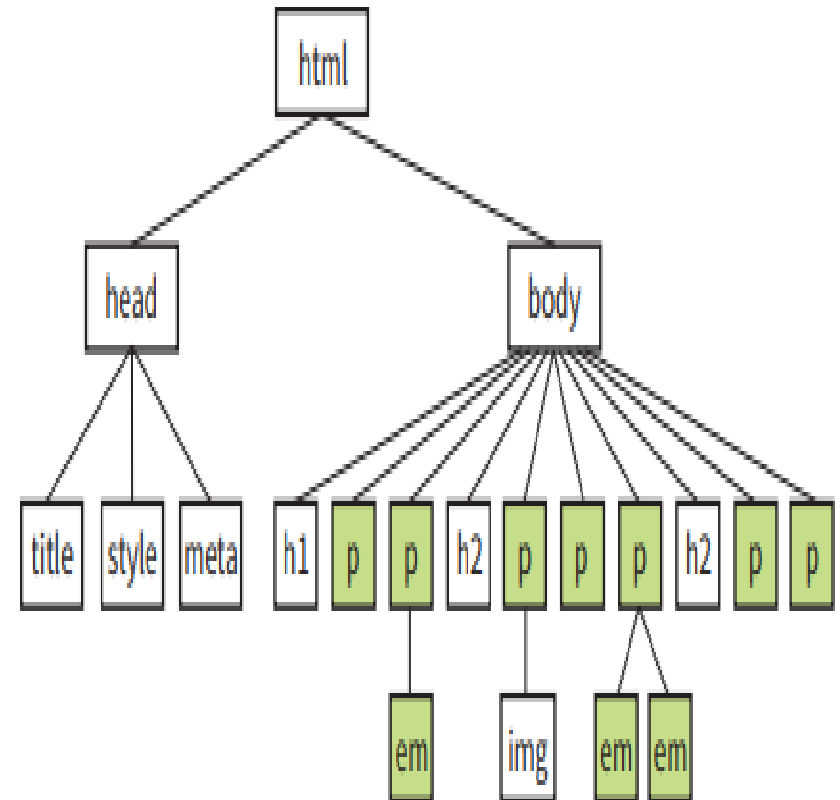
- Inline styles

```
<h1 style="color: red">Introduction</h1>
```

Structure



(The color will show for the image only if it has a border applied to it.)



`p {font-size: large; font-family: sans serif;}`

CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style. The element selector selects HTML elements based on the element name.

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

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

- The CSS id Selector

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

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

```
<p id="para1">Hello World!</p>
```

The CSS class Selector

In this example all HTML elements with class="center" will be red and center-aligned:

```
.center {  
  text-align: center;  
  color: red;} <h1 class="center">Red and center-aligned heading</h1>
```

```
<p class="center">Red and center-aligned paragraph.</p>
```

Selector2

In this example only `<p>` elements with `class="center"` will be red and center-aligned:

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

Two classes in only element `<p>`

the `<p>` element will be styled according to `class="center"` and to `class="large"`

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

```
p.large {  
  font-size: 300%;  
}
```

```
<h1 class="center">This heading will not be affected</h1>
```

```
<p class="center">This paragraph will be red and center-aligned.</p>
```

```
<p class="center large">This paragraph will be red, center-aligned, and in a large font-size.</p>
```


Selector3

The universal selector (*) selects all HTML elements on the page.

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

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

The CSS Grouping Selector

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

To group selectors, separate each selector with a comma.

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

Select all elements inside <div> elements and set their background color to yellow:

```
div * {  
  background-color: yellow;}  
97
```

selectors4

Descendant selectors are indicated in a list separated by a character space.

```
li em { color: olive; }
```

Select and style every <p> element where the parent is a <div> element:

```
div > p {  
  background-color: yellow;  
}
```

```
<div>
```

```
  <h2>My name is Donald</h2>
```

```
  <p>I live in Duckburg.</p>
```

```
</div>
```

```
<div>
```

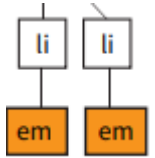
```
  <span><p>I will not be styled.</p></span>
```

```
</div>
```

CSS *element+element* Selector

Select and style the first <p> element that are placed immediately after <div> elements:

```
div + p {  
  background-color: yellow;  
} <div><p></p></div><p>yes</p>
```



- `p ~ ul` Selects every element that are preceded by a <p> element

CSS Backgrounds

- background-color
- background-image
- background-repeat
- background-attachment
- background-position

The background-color: property specifies the background color of an element.

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

With CSS, a color is most often specified by:

- a valid color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

```
h1,div,span {  
  background-color: green;
```

```
opacity: 0.3;  
}
```

Background

Set the background image for a page:

```
body {  
  background-image: url("paper.gif");  
}
```

```
p {  
  background-image: url("paper.gif");  
}
```

background-repeat: no-repeat; value:Repeat-x, Repeat-y

background-position: right top;

background-attachment: scroll; value:Fixed

Use the shorthand property to set the background properties in one declaration:

```
body {  
  background: #ffffff url("img_tree.png") no-repeat right top;  
}
```

Background Position

The background-position property specifies the position of the origin image in the background.

background-position

Values: length measurement | percentage | left | center | right | top | bottom

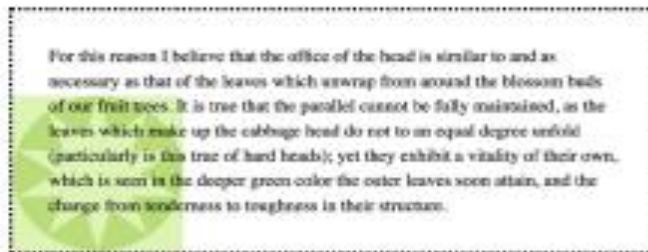
Default: 0% 0% (same as left top)

Applies to: all elements

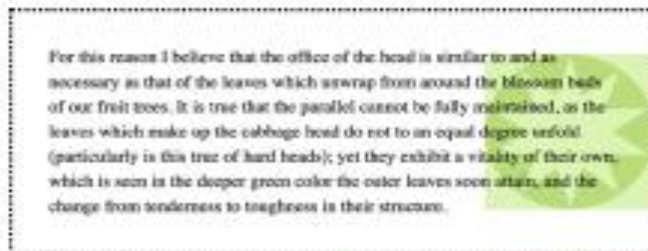
Keywords are typically used in pairs, as in these examples: background-position: left bottom; background-position: right center;

background-position right 50px bottom 50px;

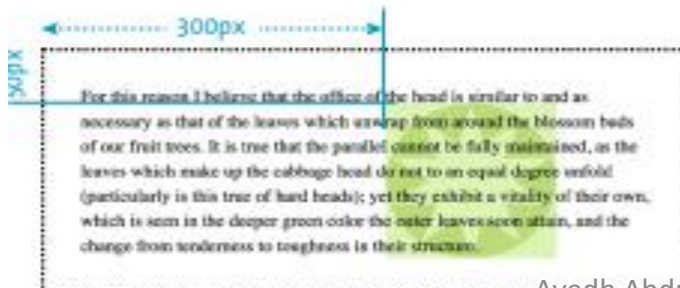
This four-part syntax is not supported by IE 8 and earlier, Safari and iOS ,Safari 6 and earlier, and Android 4.3 and earlier.



`background-position: left bottom;`



`background-position: right center;`
or
`background-position: right;`



`background-position: 300px 50px;`

Multiple Backgrounds

```
body {  
  background-image: url(image1.png), url(image2.png),  
  url(image3.png);  
  background-position: left top, center center, right bottom;  
  background-repeat: no-repeat, no-repeat, no-repeat;  
  ...  
}  
body {  
  background:  
  url(image1.png) left top no-repeat,  
  url(image2.png) center center no-repeat,  
  url(image3.png) right bottom no-repeat;  
}
```



Text Decoration

The text-decoration property is used to set or remove decorations from text. The value text-decoration: none; is often used to remove underlines from links

```
a {  
  text-decoration: none;  
}
```

The other text-decoration values are used to decorate text:

```
h1 {  
  text-decoration: overline;  
}
```

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

```
h3 {  
  text-decoration: underline;  
}
```


Text Transformation

The text-transform property is used to specify uppercase and lowercase letters in a text.

It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word: Example

```
p.uppercase {  
  text-transform: uppercase;  
}
```

```
p.lowercase {  
  text-transform: lowercase;  
}
```

```
p.capitalize {  
  text-transform: capitalize;  
}
```

Links

Styling Links

Links can be styled with any CSS property (e.g. color, font-family, background, etc.)

In addition, links can be styled differently depending on what **state** they are in.

The four links states are:

a:link - a normal, unvisited link

a:visited - a link the user has visited

a:hover - a link when the user mouses over it

a:active - a link the moment it is clicked

Example:

Link

```
/* unvisited link */
a:link {
    color: red;
}
/* visited link */
a:visited {
    color: green;
}
/* mouse over link */
a:hover {
    color: hotpink;
}
/* selected link */
a:active {
    color: blue;
}
```

Text Decoration

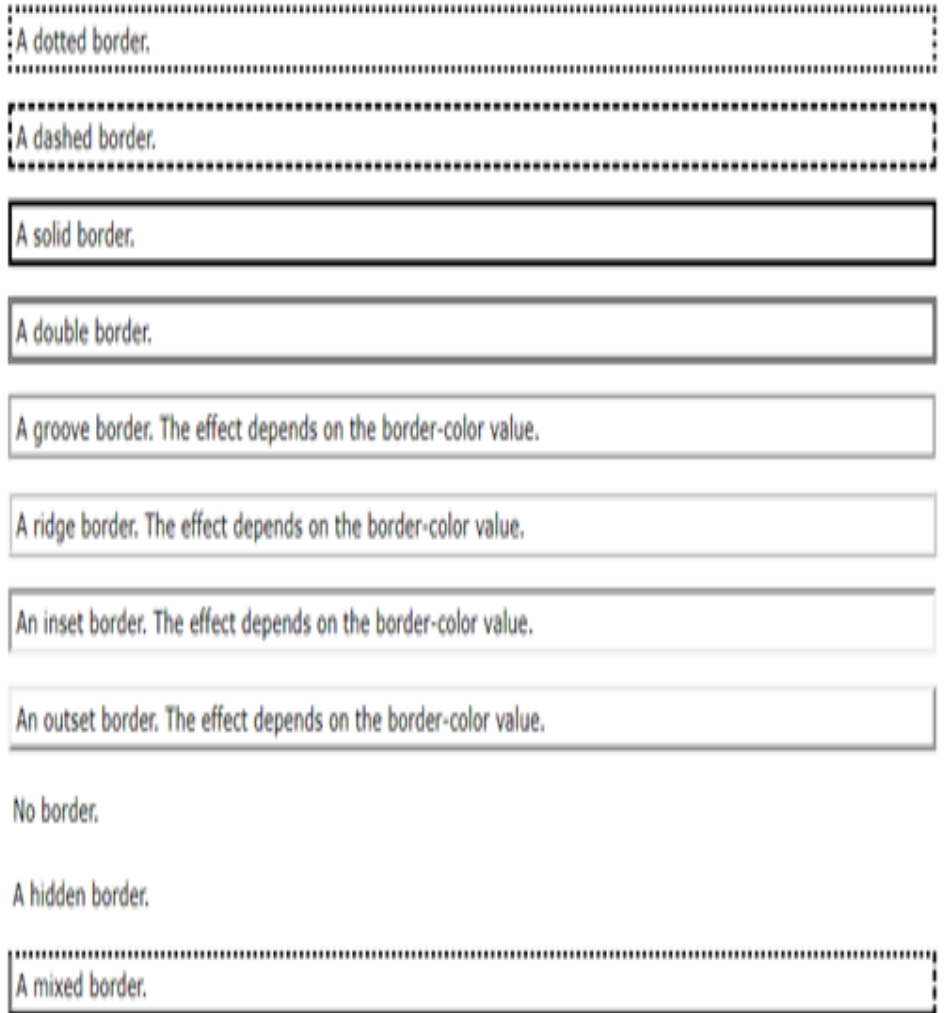
```
a:link {
    text-decoration: none;
}
a:visited {
    text-decoration: none;
}
a:hover {
    text-decoration: underline;
}
a:active {
    text-decoration: underline;
}
```

Background color

```
a:link {
    background-color: yellow;
}
a:visited {
    background-color: cyan;
}
a:hover {
    background-color: lightgreen;
}
a:active {
    background-color: hotpink;
}
```

CSS Borders

p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}
p.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style:
dotted dashed solid double;}



Border-radius

The border-width property specifies the width of the four borders.

- The border-color property is used to set the color of the four borders.
- The border property is a shorthand property for the following individual border properties:

border-width

border-style (required)

border-color

```
p {  
  border: 5px solid red;  
}
```

- CSS Rounded Borders

The border-radius property is used to add rounded borders to an element:

```
p {  
  border: 2px solid red;  
  border-radius: 8px;  
}
```

Rounder border

④ `border-radius: 36px 40px 60px 20px / 12px 10px 30px 36px;`

Ⓐ



`border-top-right-radius: 100px 50px;`

Ⓑ



`border-top-right-radius: 50px 20px;`
`border-top-left-radius: 50px 20px;`

Ⓒ



`border-radius: 60px / 40px;`

Ⓓ



`border-radius:`
`36px 40px 60px 20px/12px 10px 30px 36px;`

CSS Margins

The CSS margin properties are used to create space around elements, outside of any defined borders.

Margin - Individual Sides

CSS has properties for specifying the margin for each side of an element:

margin-top

margin-right

margin-bottom

margin-left

Set different margins for all four sides of a <p> element:

```
p {  
  margin-top: 100px;  
  margin-bottom: 100px;  
  margin-right: 150px;  
  margin-left: 80px;  
}
```

Margin: four values:

If the margin property has four values:

- **margin: 25px 50px 75px 100px;**

top margin is 25px

right margin is 50px

bottom margin is 75px

left margin is 100px

If the margin property has one value:

margin: 25px;

all four margins are 25px

- The auto Value

You can set the margin property to auto to horizontally center the element within its container.

CSS Padding

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

CSS has properties for specifying the padding for each side of an element:

- padding-top
- padding-right
- padding-bottom
- padding-left

Set different padding for all four sides of a <div> element:

```
div {  
  padding-top: 50px;  
  padding-right: 30px;  
  padding-bottom: 50px;  
  padding-left: 80px;  
}
```

If the padding property has four values:

padding: 25px 50px 75px 100px;

- top padding is 25px
- right padding is 50px
- bottom padding is 75px
- left padding is 100px

SPECIFYING BOX DIMENSIONS

width

Values: length | percentage | auto

Default: auto

Applies to: block-level elements and replaced Inline elements (such as images)

Height

Values: length | percentage | auto

Default: auto

Applies to: block-level elements and replaced inline elements (such as images)

box-sizing

Values: content-box | border-box

Default: content-box

Applies to: all elements

Sizing the Content Box

By default (that is, if you do not include a box-sizing rule in your styles), the width and height properties are applied to the content box. That is the way all current browsers interpret width and height values, but you can explicitly specify this behavior by setting `box-sizing: content-box`

In the default content box model, the width and height values are applied to the content area only.

```
p
{ background: #f2f5d5;
  width: 500px;
  height: 150px;
  padding: 20px;
  border: 5px solid gray;
  margin: 20px;
}
```

The resulting width of the visible element box ends up being 550 pixels: the content plus 40px padding (20px left and right) and 10px of border (5px left and right).

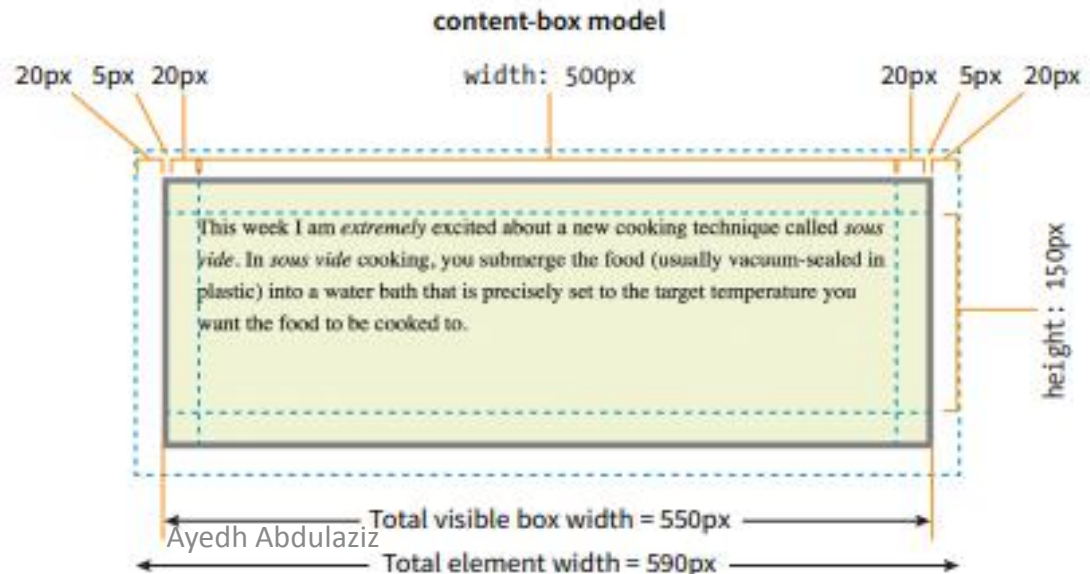
Box-sizing

- Visible element box =

$5\text{px} + 20\text{px} + 500\text{px width} + 20\text{px} + 5\text{px} = 550\text{ pixels}$

When you throw in 40 pixels of margin, the width of the entire element box is 590 pixels. Knowing the resulting size of your elements is critical to getting layouts to behave predictably.

- Element box = $20\text{px} + 5\text{px} + 20\text{px} + 500\text{px width} + 20\text{px} + 5\text{px} + 20\text{px} = 590\text{ pixels}$



Using the border-box Model

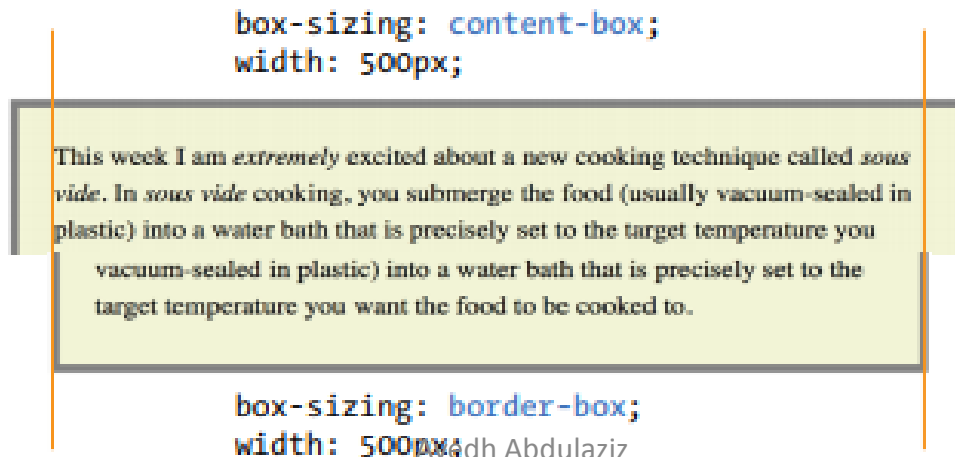
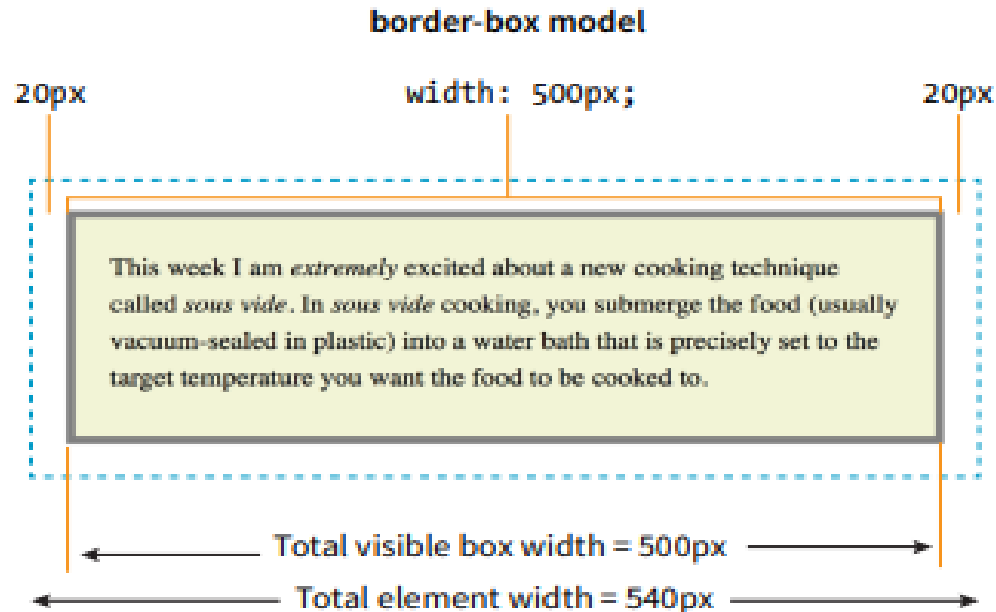
The other way to specify the size of an element is to apply width and height dimensions to the entire visible box, including the padding and border.

```
p {  
  box-sizing: border-box;  
  width: 500px;  
  height: 150px;  
}
```

```
p {  
  Max-width: 500px; or min  
  max-height: 150px; or min  
}
```

Now the width of the visible box is 500 pixels (compare to 550 pixels in the content-box model), and the total element width is 540px.

Border-box



Handling Overflow

When an element is sized too small for its contents, you can specify what to do with the content that doesn't fit by using the overflow property.

overflow

Values: visible | hidden | scroll | auto

Default: visible

Applies to: block-level elements and replaced inline elements (such as images)

Visible: The default value is visible, which allows the content to hang out over the element box so that it all can be seen.

Hidden: When overflow is set to hidden, the content that does not fit gets clipped off and does not appear beyond the edges of the element's content area.

Scroll: When scroll is specified, scrollbars are added to the element box to let users scroll through the content. Be aware that they may become visible only when you click the element to scroll it.

Auto: The auto value allows the browser to decide how to handle

Figure

visible

Applying the masks to the glasses is the most labor-intensive part of the process. Not only do you have to measure, place, and burnish on each mask, but you also need to completely cover the remainder of the glass in heavy paper. Any exposed areas (even inside) will get scratched by the flying sand, so it has to be a good seal.

hidden

Applying the masks to the glasses is the most labor-intensive part of the process. Not only do you have to measure, place, and burnish on each mask, but you also need to completely cover the remainder of the glass

scroll

labor-intensive part of the process. Not only do you have to measure, place, and burnish on each mask, but you also need to completely cover the remainder of the glass in heavy paper. Any exposed areas (even

auto (short text)

Applying the masks to the glasses is the most labor-intensive part of the process.

auto (long text)

Applying the masks to the glasses is the most labor-intensive part of the process. Not only do you have to measure, place, and burnish on each mask, but you also need to completely cover the remainder of the glass

CSS Table attribute

```
table {  
  width: 50%;  
}  
  
th {  
  height: 70px;  
}  
  
table, td, th {  
  border: 1px solid black;  
}  
  
th, td {  
  padding: 15px;  
  text-align: left;  
}
```

```
table {  
  border-collapse: collapse;  
  width: 100%;  
}  
  
td {  
  height: 50px;  
  vertical-align: bottom or  
  top ;  
}
```

Ayedh Abdulaziz

Responsive Table

```
<div style="overflow-  
x:auto;">
```

```
<table>  
... table content ...  
</table>
```

```
</div>
```


ASSIGNING DISPLAY TYPES

As long as we're talking about boxes and the CSS layout model, this is a good time to introduce the display property.

display Values: inline | block | list-item | table | none.....

Default: inline

Applies to: all elements

Block-level Elements

<div>

<h1> - <h6>

<p>

<form>

<header>

<footer>

<section>

Inline Elements

<a>

Example:

```
<style>
li {
  display: inline;
}
</style>
<body>
<p>Display a list of links as a horizontal menu:</p>
<ul>
  <li><a href="/html/default.asp" target="_blank">HTML</a></li>
  <li><a href="/css/default.asp" target="_blank">CSS</a></li>
  <li><a href="/js/default.asp" target="_blank">JavaScript</a></li>
</ul>
```

Display a list of links as a horizontal menu:

HTML CSS JavaScript

Hide an Element - display:none or visibility:hidden?

Hiding an element can be done by setting the display property to none. The element will be hidden, and the page will be displayed as if the element is not there:

```
<style>
h1.hidden {
  display: none;
}
</style>
</head>
<body>
<h1>This is a visible heading</h1>
<h1 class="hidden">This is a hidden heading</h1>
<p>Notice that the h1 element with display: none; does not take up any
space.</p>
```

This is a visible heading

Notice that the h1 element with display: none; does not take up any space.

Visibility: hidden?

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

Visible: Default value. The element is visible

Hidden: The element is hidden (but still takes up space)

Collapse: Only for table rows (`<tr>`), row groups (`<tbody>`), columns (`<col>`), column groups (`<colgroup>`). This value removes a row or column, but it does not affect the table layout. The space taken up by the row or column will be available for other content.

initial: Sets this property to its default value

Inherit: Inherits this property from its parent element

Visibility: Initial

```
<style>
div {
  text-align: center;
  border: 1px solid blue;
}
```

```
h1 {
  text-align: initial;
}</style>
</head>
<body>
```

```
<div>
  <h1>Initial</h1>
  <p>The header above and this paragraph is inside a DIV element, The DIV element has the color property
  set to "red". The header element has its color property set to "initial", which in this case is "black".</p>
</div>
```

<p>Note: The "initial" keyword is not supported in Internet Explorer 11 and earlier versions, or in Opera 15 and earlier versions.</p>

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

Result

Initial

The header above and this paragraph is inside a DIV element, The DIV element has the color property set to "red". The header element has its color property set to "initial", which in this case is "black".

FLOATING AND POSITIONING

At this point, you've learned dozens of CSS properties that let you change the appearance of text elements and the boxes they generate. But so far, we've merely been formatting elements as they appear in the flow of the document.

NORMAL FLOW:

In the CSS layout model, text elements are laid out from top to bottom in the order in which they appear in the source, and from left to right in left-to-right.

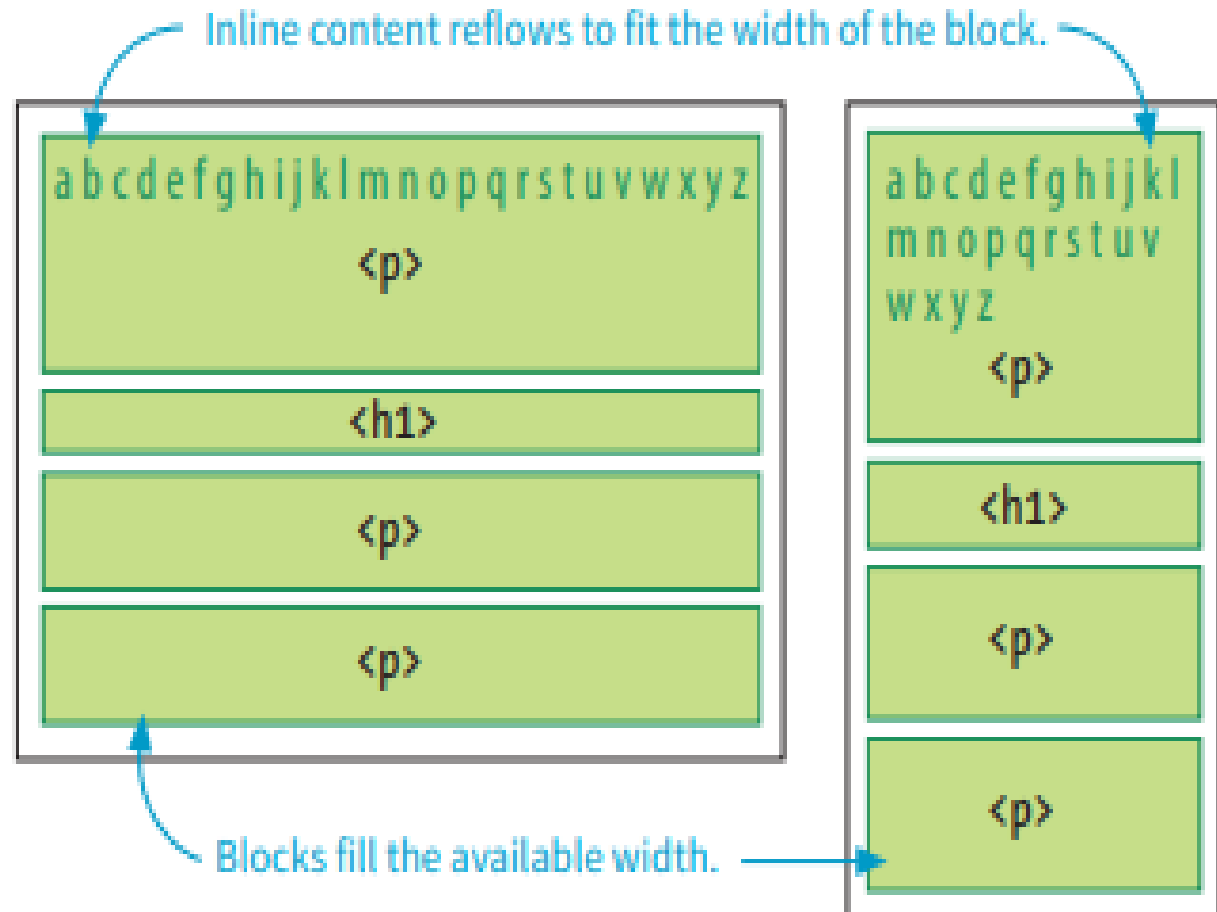
Block elements stack up on top of one another and fill the available width of the browser window or other containing element.

Inline elements and text characters line up next to one another to fill the block elements.

Figure

Blocks are laid out in the order in which they appear in the source.

Each block starts on a new line.



FLOATING

Simply stated, the float property moves an element as far as possible to the left or right, allowing the following content to wrap around it. It is a unique feature built into CSS with some interesting behaviors.

float

Values: left | right | none

Default: none

Applies to: all elements

Inherits: no.

FIGURE next slide shows how the paragraph and the contained image are rendered by default (top) and how it looks when the float property is applied (bottom).

Float2

THE MARKUP

<p> After the cream is frozen rather stiff,...

THE STYLES

```
img {  
  float: right;  
}
```

You can begin to see how the box properties work together to improve page layout.

```
img {  
  float: right;  
  margin: 1em;  
}
```

Inline image in the normal flow



After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub. Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid.

Space next to image is held clear

Inline image floated to the right

After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub. Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid.



Image moves over, and text wraps around it

Indicates outer margin edge (dotted line does not appear in the browser)

After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub. Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid.



Floating an inline text element

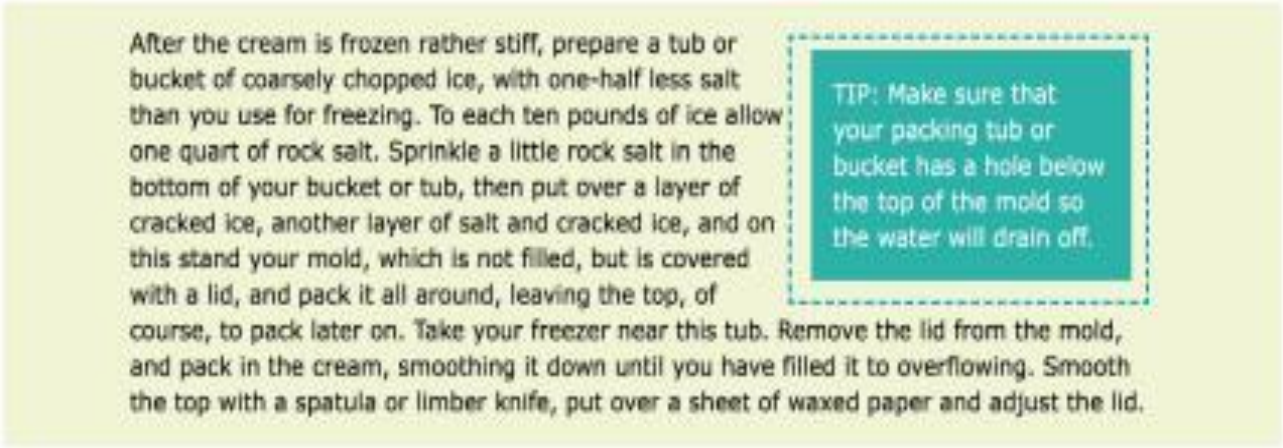
THE MARKUP

`<p>TIP: Make sure that your packing tub or bucket`

`has a hole below the top of the mold so the water will drain off.After the cream is frozen rather stiff, prepare a tub or bucket of...</p>`

THE STYLES

```
span.tip {  
  float: right;  
  margin: 1em;  
  width: 200px;  
  color: #fff;  
  background-color: lightseagreen;  
  padding: 1em;  
}
```



After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub. Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid.

TIP: Make sure that your packing tub or bucket has a hole below the top of the mold so the water will drain off.

Floating block elements

Let's look at what happens when you float a block within the normal flow. In this example, the whole second paragraph element is floated to the left .

THE MARKUP

```
<p>If you wish to pack ice cream...</p>
```

```
<p id="float">After the ice cream is rather stiff,...</p>
```

```
<p>Make sure that your packing tub or bucket...</p>
```

```
<p>As cold water is warmer than the ordinary...</p>
```

THE STYLES

```
p { border: 2px red solid; }
```

```
#float
```

```
{ float: left;
```

```
width: 300px;
```

```
margin: 1em;
```

```
background: white; }
```

If you wish to pack ice cream and serve it in forms or shapes, it must be molded after the freezing. The handiest of all of these molds is either the brick or the melon mold.

After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub.

the cold water spigot, then quickly wipe the top and bottom and remove the lid. Loosen the pudding with a limber knife, hold the mold a little slanting, give it a shake, and nine times out of ten it will come out quickly, having the perfect shape of the can or mold. If the cream still sticks and refuses to come out, wipe the mold with a towel wrung from warm water. Hot water spoils the gloss of puddings, and unless you know exactly how to use it, the cream is too much melted to garnish.

Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid. Have a strip of muslin or cheese cloth dipped in hot paraffin or suet and quickly bind the seam of the lid. This will remove all danger of salt water entering the pudding. Now cover the mold thoroughly with ice and salt.

Make sure that your packing tub or bucket has a hole below the top of the mold, so that the salt water will be drained off. If you are packing in small molds, each mold, as fast as it is closed, should be wrapped in wax paper and put down into the salt and ice. These must be filled quickly and packed.

As cold water is warmer than the ordinary freezing mixture, after you lift the can or mold, wipe off the salt, hold it for a minute under

If you wish to pack ice cream and serve it in forms or shapes, it must be molded after the freezing. The handiest of all of these molds is either the brick or the melon mold.

After the cream is frozen rather stiff, prepare a tub or bucket of coarsely chopped ice, with one-half less salt than you use for freezing. To each ten pounds of ice allow one quart of rock salt. Sprinkle a little rock salt in the bottom of your bucket or tub, then put over a layer of cracked ice, another layer of salt and cracked ice, and on this stand your mold, which is not filled, but is covered with a lid, and pack it all around, leaving the top, of course, to pack later on. Take your freezer near this tub.

the cold water spigot, then quickly wipe the top and bottom and remove the lid. Loosen the pudding with a limber knife, hold the mold a little slanting, give it a shake, and nine times out of ten it will come out quickly, having the perfect shape of the can or mold. If the cream still sticks and refuses to come out, wipe the mold with a towel wrung from warm water. Hot water spoils the gloss of puddings, and unless you know exactly how to use it, the cream is too much melted to garnish.

Remove the lid from the mold, and pack in the cream, smoothing it down until you have filled it to overflowing. Smooth the top with a spatula or limber knife, put over a sheet of waxed paper and adjust the lid. Have a strip of muslin or cheese cloth dipped in hot paraffin or suet and quickly bind the seam of the lid. This will remove all danger of salt water entering the pudding. Now cover the mold thoroughly with ice and salt.

Make sure that your packing tub or bucket has a hole below the top of the mold, so that the salt water will be drained off. If you are packing in small molds, each mold, as fast as it is closed, should be wrapped in wax paper and put down into the salt and ice. These must be filled quickly and packed.

As cold water is warmer than the ordinary freezing mixture, after you lift the can or mold, wipe off the salt, hold it for a minute under the

Clearing Floated Elements

If you're going to be floating elements around, it's important to know how to turn the text wrapping off and get back to normal flow as usual. You do this by clearing the element that you want to start below the float. Applying the `clear` property to an element prevents it from appearing next to a floated element and forces it to start against the next available "clear" space below the float.

`clear`

Values: `left` | `right` | `both` | `none`

Default: `none`

Applies to: block-level elements only

Inherits: no

```
img {
  float: left;
  margin-right: .5em;
}
h2 {
  clear: left;
  margin-top: 2em;
}
```

```
<style>
```

```
img {
  float: left;
}
```

```
p.clear {
  clear: both;
}
```

```
<h1>The clear Property</h1>
```

```

```

```
<p>This is some text. This is some text. This is some text. This is
some text. This is some text.</p>
```

```
<p class="clear">This is also some text. This is also some text. This is also some
text..</p>
```

```
<p><strong>Remove the "clear" class to see the effect.</strong></p>
```

The clear Property



This is some text. This is some text. This is some text. This is some text. This is some text. This is some text. This is some text.

This is also some text. This is also some text. This is also some text. This is also some text. This is also some text. This is also some text. This is also some text.

POSITIONING BASICS

CSS provides several methods for positioning elements on the page. They can be positioned relative to where they would normally appear in the flow, or removed from the flow altogether and placed at a particular spot on the page.

You can also position an element relative to the viewport, and it will stay put while the rest of the page scrolls.

Types of Positioning position

Values: static | relative | absolute | fixed

Default: static

Applies to: all elements

Inherits: no

Position Values

static

This is the normal positioning scheme in which elements are positioned as they occur in the normal document flow.

relative

Relative positioning moves the element box relative to its original position in the flow. The distinctive behavior of relative positioning is that the space the element would have occupied in the normal flow is preserved as empty space.

absolute

Absolutely positioned elements are removed from the document flow entirely and positioned with respect to the viewport or a containing. Unlike relatively positioned elements, the space they would have occupied is closed up. In fact, they have no influence at all on the layout of surrounding elements.

Fixed

The distinguishing characteristic of fixed positioning is that the element stays in one position in the viewport even when the document scrolls. Fixed elements are removed from the document flow and positioned relative to the viewport rather than another element in the document

Specifying Position

Once you've established the positioning method, the actual position is specified with some combination of up to four offset properties.

top, right, bottom, left

Values: length | percentage | auto

Default: auto Applies

to: positioned elements (where position value is relative, absolute, or fixed) Inherits: no

em

```
{ position: relative;
```

```
top: 2em; /* moves element down */
```

```
left: 3em; /* moves element right */
```

```
background-color: fuchsia;
```

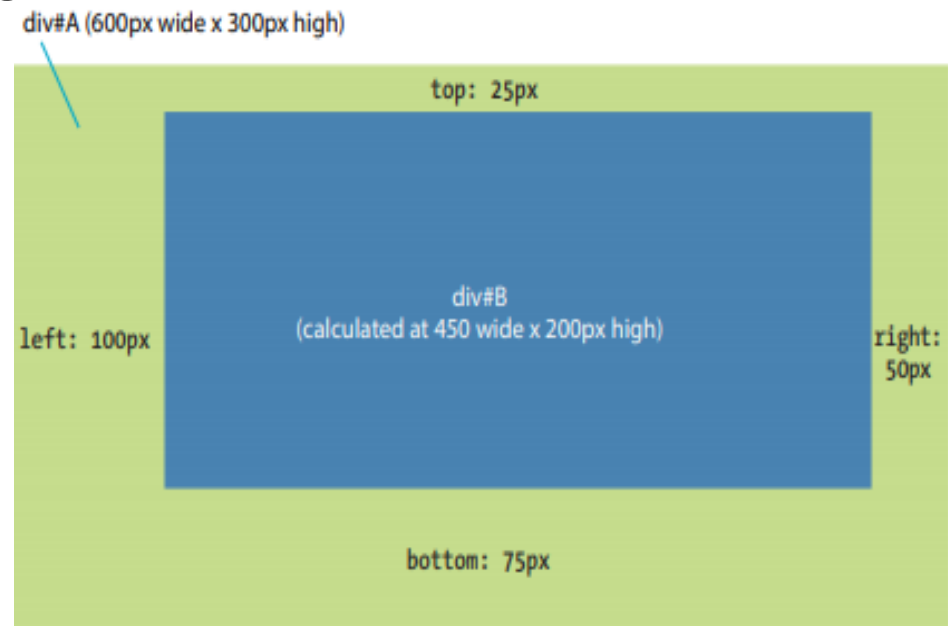
```
}
```

THE MARKUP

```
<div id="A">  
  <div id="B">&nbsp;</div>  
</div>
```

THE STYLES

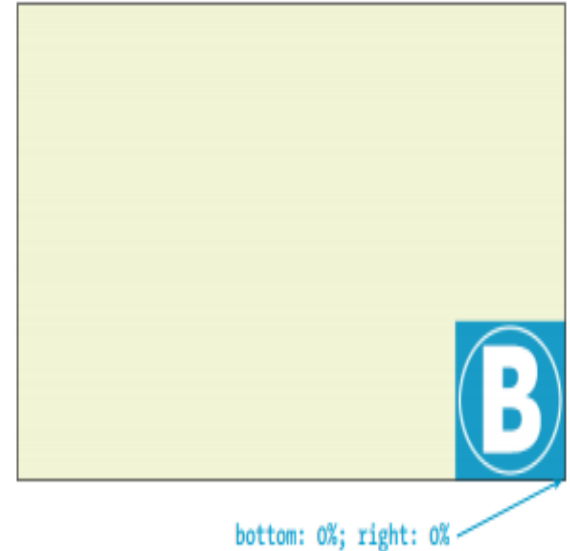
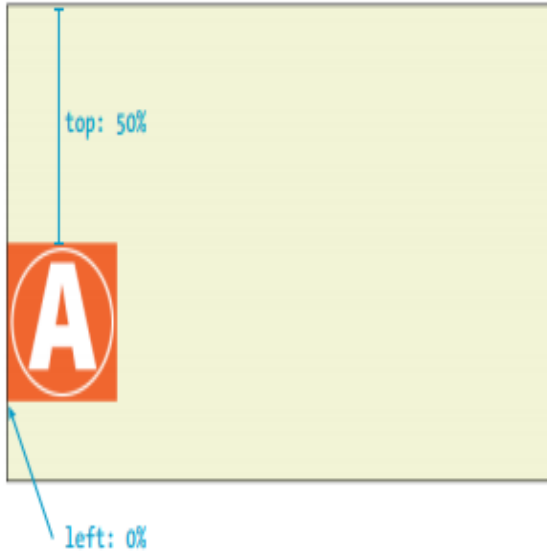
```
div#A {  
  position: relative; /* creates the containing block */  
  width: 600px;  
  height: 300px;  
  background-color: #C6DE89; /* green */  
}  
div#B {  
  position: absolute;  
  top: 25px;  
  right: 50px;  
  bottom: 75px;  
  left: 100px;  
  background-color: steelblue;  
}
```



Percentage values

```
img#A {  
  position: absolute;  
  top: 50%;  
  left: 0%;  
}
```

```
img#B {  
  position: absolute;  
  bottom: 0%;  
  right: 0%;  
}
```



Selectors Advance Pseudo-classes

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

Style an element when a user mouse over it

Style visited and unvisited links differently

Style an element when it gets focus.

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

Hover on <div>

An example of using the :hover pseudo-class on a <div> element:

```
<head>
<style>
p {
  display: none;
  background-color: yellow;
  padding: 20px;
}

div:hover p {
  display: block;
}
</style>
</head>
<body>

<div>Hover over me to show the p element
  <p>Tada! Here I am!</p>
</div>
```

The :first-child Pseudo-class

The :first-child pseudo-class matches a specified element that is the first child of another element.

```
p:first-child {  
  color: blue;  
}
```

I am a *strong* man. I am a *strong* man.

I am a *strong* man. I am a *strong* man.

<p>I am a <i>strong</i> man. I am a <i>strong</i> man.</p>

<p>I am a <i>strong</i> man. I am a <i>strong</i> man.</p>

Match the first <i> element in all <p> elements

```
p i:first-child {  
  color: blue;  
}
```

I am a *strong* man. I am a *strong* man.

I am a *strong* man. I am a *strong* man.

Match all <i> elements in all first child <p> elements

```
p:first-child i {  
  color: yellow;  
}
```

I am a **strong** man. I am a **strong** man.

I am a *strong* man. I am a *strong* man.

Pseudo-element

pseudo-element is used to style specified parts of an element.

For example, it can be used to:

Style the first letter, or line, of an element

Insert content before, or after, the content of an element.

```
p::first-line OR first-letter {
  color: #ff0000;
}
```

<p>You can use the ::first-line pseudo-element to add a
special effect to the first line of a text. Some more text.
And even more, and more, and more, and more, and
more, and more, and more, and more, and more, and
more, and more, and more.</p>

142

The ::before Pseudo-element

The ::before pseudo-element can be used to insert some content before the content of an element.

```
h1::before {  
  content: url(smiley.gif);  
}
```


<h1>This is a heading</h1>

<p>The ::before pseudo-element inserts content before the content of an element.</p>


<h1>This is a heading</h1>

```
h1::after {  
  content: url(smiley.gif);  
}
```

```
h1::after {  
  content: url(smiley.gif);  
}
```

 **This is a heading**

The ::before pseudo-element inserts content before the content of an element.

 **This is a heading**

::after

```
::marker {  
  color: red;  
  font-size: 23px;  
}
```

The ::marker Pseudo-element

The ::marker pseudo-element selects the markers of list items.

```
<ul>  
  <li>Coffee</li>  
  <li>Tea</li>  
  <li>Milk</li>  
</ul>  
<ol>  
  <li>First</li>  
  <li>Second</li>  
  <li>Third</li>  
</ol>
```

- Coffee
- Tea
- Milk

1. First
2. Second
3. Third

The ::selection Pseudo-element

The ::selection pseudo-element matches the portion of an element that is selected by a user. The following CSS properties can be applied to ::selection: color, background.

```
::-moz-selection { /* Code for Firefox */  
  color: red;  
  background: yellow;  
}  
::selection {  
  color: red;  
  background: yellow;  
}
```

Select some text on this page:

This is a paragraph.

This is some text in a div element.

Note: Firefox supports an alternative, the ::-moz-selection property.

<h1>Select some text on this page:</h1>

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

<div>This is some text in a div element.</div>

<p>Note: Firefox supports an alternative, the ::-moz-selection property.</p>

:focus Selector

Select and style an input field when it gets focus:

```
input:focus {  
  background-color: yellow;  
}
```

When an `<input type="text">` gets focus, gradually change the width from 100px to 250px:

```
input[type=text] {  
  width: 100px;  
}
```

```
input[type=text]:focus {  
  width: 250px;  
}
```

:read-only Selector

Select and style only if the input element is "readonly":

```
input:read-only {  
  background-color: yellow;  
}  
input:read-only {  
  background-color: yellow;  
}
```

<h3>A demonstration of the :read-only selector.</h3>

<p>A normal input element:
<input value="hello"></p>

<p>A readonly input element:
<input readonly value="hello"></p>

<p>The :read-only selector selects form elements with a "readonly" attribute.</p>

Show and hide a "dropdown" menu on mouse hover:

```
ul {  
  display: inline;  
  margin: 0;  
  padding: 0;  
}  
ul li {display: inline-block;}  
ul li:hover {background: #555;}  
ul li:hover ul {display: block;}  
ul li ul {  
  position: absolute;  
  width: 200px;  
  display: none;  
}  
ul li ul li {  
  background: #555;  
  display: block;  
}  
ul li ul li a {display: block;}  
ul li ul li:hover {background: #666;}
```

```
<div>  
  <a href="#">Useless Link</a>  
  <ul>  
    <li>  
      <a href="#">Dropdown Link</a>  
      <ul>  
        <li><a href="#">Link 1</a></li>  
        <li><a href="#">Link 2</a></li>  
        <li><a href="#">Link 3</a></li>  
      </ul>  
    </li>  
  </ul>  
</div>
```

LAYOUT WITH FLEXBOX

Setting Up a Flexbox Container You've already learned about the block layout mode for stacking elements in the normal flow and the inline mode for displaying content within it horizontally. Flexbox is another layout mode with its own behaviors. To turn on flexbox mode for an element, set its display property to flex or inline flex .It is now a flex container, and all of its direct child elements (whether they are div, list items, paragraphs, etc.) automatically become flex items in that container. The flex items (the beads) are laid out and aligned along flex lines (the string).

Flexbox Layout Module

Before the Flexbox Layout module, there were four layout modes:

Block, for sections in a webpage

Inline, for text

Table, for two-dimensional table data.

Positioned, for explicit position of an element

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

Example

```
<div id="container">  
  <div class="box box1">1</div>  
  <div class="box box2">2</div>  
  <div class="box box3">3</div>  
  <div class="box box4">4</div>  
  <div class="box box5">5</div>  
</div>
```

THE STYLES

```
#container {  
  display: flex;  
}
```


You can see that the items have lined up in a row from left to right, which is the default Flexbox behavior if your page is in English or another language written in rows from left to right

By default, the `divs` display as block elements, stacking up vertically. Turning on flexbox mode makes them line up in a row.

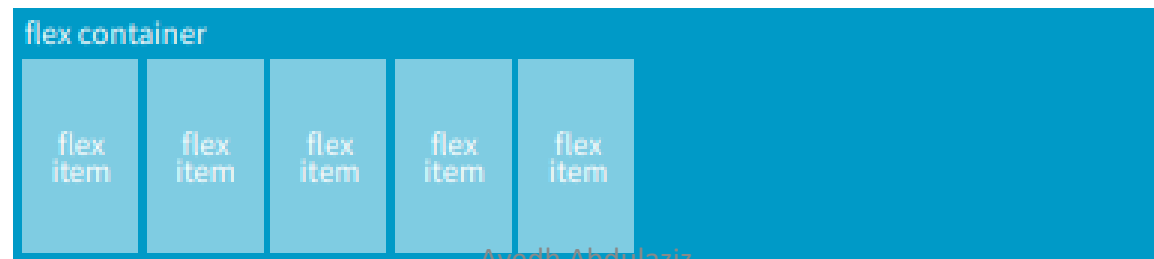


block layout mode

`display: flex;`



flexbox layout mode



Controlling the “Flow” Within the Container

Once you turn an element into a flex container, there are a few properties you can set on that container to control how items flow within it. The flow refers to the direction in which flex items are laid out as well as whether they are permitted to wrap onto additional lines.

- *float, clear, multicolumn layout, and vertical-align do not work with elements in flexbox mode*

flex-direction

Values: row | column | row-reverse | column-reverse Default: row

Applies to: flex containers

Inherits: no

flex-direction

`flex-direction: row;` (default)



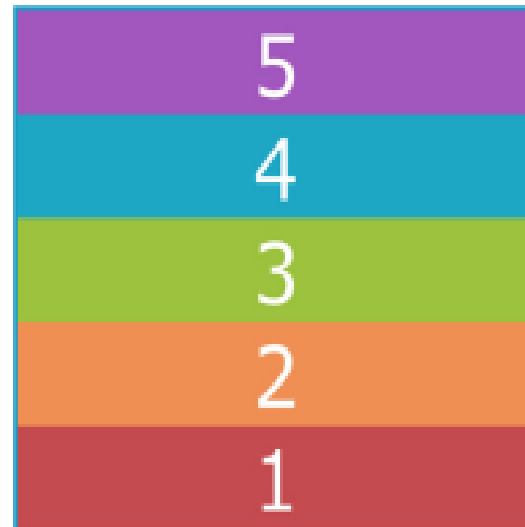
`flex-direction: row-reverse;`



`flex-direction: column;`

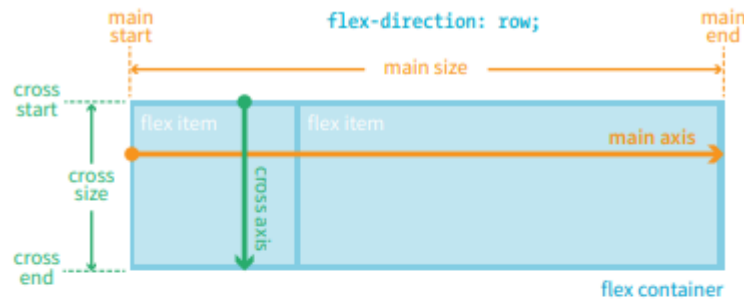


`flex-direction: column-reverse;`

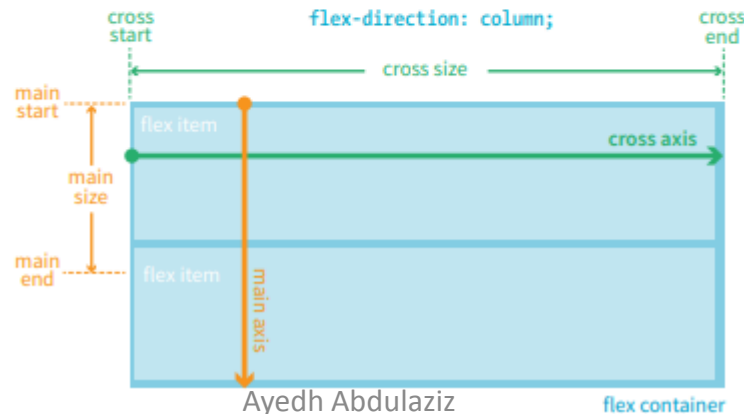


Wrapping onto multiple lines

If you have a large or unknown number of flex items in a container and don't want them to get all squished into the available space, you can allow them to break onto additional lines with the flex-wrap property.



When `flex-direction` is set to `column`, the main axis is vertical and the cross axis is horizontal.



flex-wrap

flex-wrap

Values: nowrap | wrap | wrap-reverse

Default: nowrap

Applies to: flex containers

Inherits: no

By default, items do the squish thing and do not wrap onto additional lines (nowrap). The wrap keyword turns on the ability to wrap onto multiple lines in the direction from cross start to cross end. For example, if the direction is row, then lines are positioned from the top down.

flex-wrap with flex-direction: row; (Example)

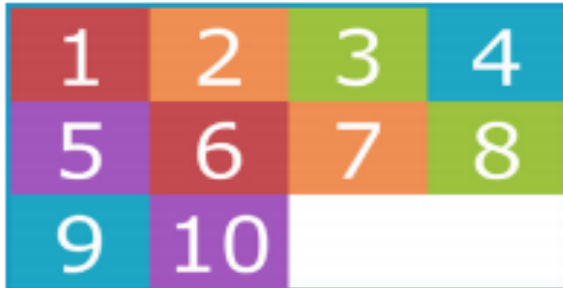
THE MARKUP

```
<div id="container">
  <div class="box box1">1</div>
  <!-- more boxes here -->
  <div class="box box10">10</div>
</div>
```

THE STYLES

```
#container {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
}
.box {
  width: 25%;
}
```

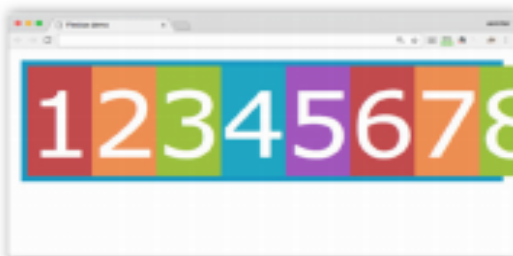
flex-wrap: wrap;



flex-wrap: wrap-reverse;



flex-wrap: nowrap; (default)

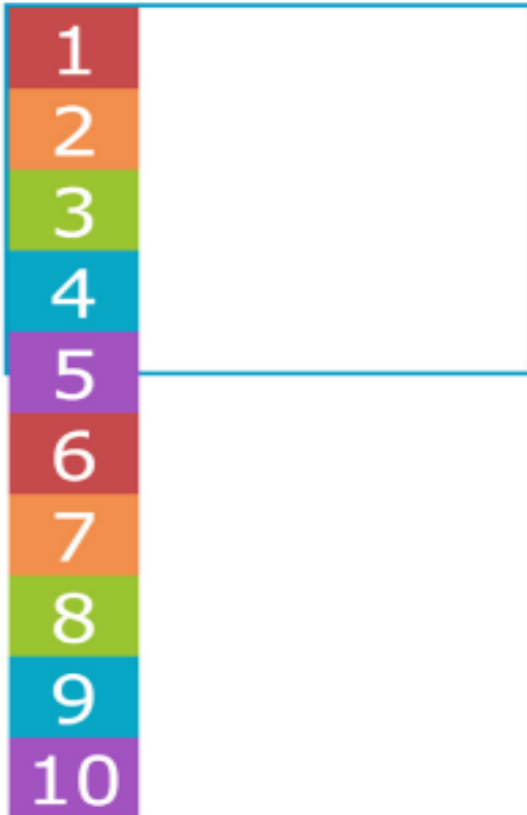


When wrapping is disabled, flex items squish if there is not enough room, and if they can't squish any further, may get cut off if there is not enough room in the viewport.

flex-wrap with flex-direction: column; (Example)

```
#container {  
  display: flex;  
  height: 350px;  
  flex-direction: column;  
  flex-wrap: wrap;  
}  
.box {  
  width: 25%;  
}
```

flex-wrap: nowrap; (default)



flex-wrap: wrap;



flex-wrap: wrap-reverse;



Putting it together with flex-flow .

The shorthand property flex-flow makes specifying flex-direction and flex-wrap short and sweet.

```
#container {  
  display: flex;  
  height: 350px;  
  flex-flow: column wrap;  
}
```

The flex-basis Property

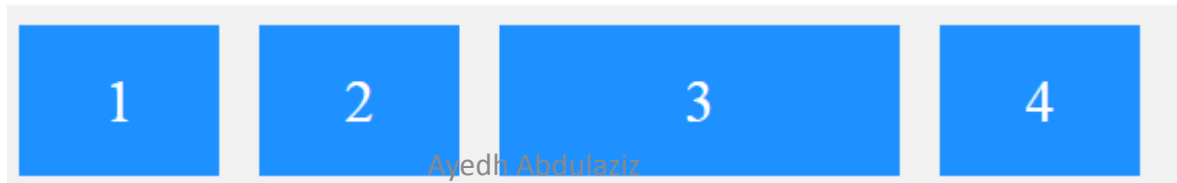
The flex-basis property specifies the initial length of a flex item.

Set the initial length of the third flex item to 200 pixels:

```
<div class="flex-container">  
  <div>1</div>  
  <div>2</div>  
  <div style="flex-basis:  
200px">3</div>  
  <div>4</div>  
</div>
```

```
.flex-container {  
  display: flex;  
  background-color: #f1f1f1;  
}
```

```
.flex-container > div {  
  background-color: DodgerBlue;  
  color: white;  
  width: 100px;  
  margin: 10px;  
  text-align: center;  
  font-size: 30px;  
}
```



Responsive Flexbox

if you want to create a two-column layout for most screen sizes, and a one-column layout for small screen sizes (such as phones and tablets), you can change the flex-direction from row to column at a specific breakpoint (800px in the example below):

```
.flex-container {  
  display: flex;  
  flex-direction: row;  
}  
  
/* Responsive layout - makes a one column layout instead of a  
two-column layout */  
  
@media (max-width: 800px) {  
  .flex-container {  
    flex-direction: column;  
  }  
}
```

Example

```
* {  
  box-sizing: border-box;  
}
```

```
.flex-container {  
  display: flex;  
  flex-direction: row;  
  font-size: 30px;  
  text-align: center;  
}
```

```
.flex-item-left {  
  background-color: #f1f1f1;  
  padding: 10px;  
  flex: 50%;  
}
```

```
.flex-item-right {  
  background-color: dodgerblue;  
  padding: 10px;  
  flex: 50%;  
}
```

```
/* Responsive layout - makes a one  
column-layout instead of two-  
column layout */
```

```
@media (max-width: 800px) {  
  .flex-container {  
    flex-direction: column;  
  }  
}
```

Responsive Image Gallery using Flexbox

Use flexbox to create a responsive image gallery that varies between four, two or full-width images, depending on screen size:

```
* {
  box-sizing: border-box;
}

body {
  margin: 0;
  font-family: Arial;
}

.header {
  text-align: center;
  padding: 32px;
}

.row {
  display: flex;
  flex-wrap: wrap;
  padding: 0 4px;
}

.column {
  flex: 25%;
  max-width: 25%;
  padding: 0 4px;
}

.column img {
  margin-top: 8px;
  vertical-align: middle;
}

@media (max-width: 800px) {
  .column {
    flex: 50%;
    max-width: 50%;
  }
}

@media (max-width: 600px) {
  .column {
    flex: 100%;
    max-width: 100%;
  }
}
```

Responsive Web Design - Media Queries

Now we get to the real meat of responsive design—media queries! Media queries apply different styles based on characteristics of the browser: its width, whether it is vertically or horizontally oriented, its resolution, and more.

```
@media type and (feature: value) {  
  /* styles for browsers that meet this criteria */  
}
```

```
@media print { /* print-specific styles here */ }  
////////////////////
```

```
h1 { font-family: Georgia, serif; }
```

```
@media screen and (min-width: 40em) {  
h1  
{ font-family: 'Lobster', cursive; } }
```

Media

```
@media screen and (min-width: 600px) {  
  header p {  
    display: block;  
    margin-top: -1.5em;  
    font-family: Georgia, serif;  
    font-style: italic;  
    font-size: 1.2em;  
  }  
  main, h2, h3 {  
    font-family: 'Stint Ultra Expanded', Georgia, serif;  
  }  
  h2, h3 {  
    font-weight: bold;  
  }  
  main {  
    line-height: 1.8em;  
    padding: 1em;  
    border: double 4px #EADDC4;  
    border-radius: 25px;  
    margin: 2.5%;  
  }  
}
```

Responsive- Media

Media queries can help with that. We can add a breakpoint where certain parts of the design will behave differently on each side of the breakpoint.

Use a media query to add a breakpoint at 768px:

```
/* For desktop: */
```

```
.col-1 {width: 8.33%;}  
.col-2 {width: 16.66%;}  
.col-3 {width: 25%;}  
.col-4 {width: 33.33%;}  
.col-5 {width: 41.66%;}  
.col-6 {width: 50%;}  
.col-7 {width: 58.33%;}  
.col-8 {width: 66.66%;}  
.col-9 {width: 75%;}  
.col-10 {width: 83.33%;}  
.col-11 {width: 91.66%;}  
.col-12 {width: 100%;}
```

```
@media only screen and (max-width: 768px) {
```

```
  /* For mobile phones: */  
  [class*="col-"] {  
    width: 100%;  
  }  
}
```

```
/* For mobile phones: */
```

```
[class*="col-"] {  
  width: 100%;  
}
```

```
@media only screen and (min-width: 768px) {
```

```
  /* For desktop: */  
  .col-1 {width: 8.33%;}  
  .col-2 {width: 16.66%;}  
  .col-3 {width: 25%;}  
  .col-4 {width: 33.33%;}  
  .col-5 {width: 41.66%;}  
  .col-6 {width: 50%;}  
  .col-7 {width: 58.33%;}  
  .col-8 {width: 66.66%;}  
  .col-9 {width: 75%;}  
  .col-10 {width: 83.33%;}  
  .col-11 {width: 91.66%;}  
  .col-12 {width: 100%;}  
}
```

Training to Responsive Website Layout

Resize the browser window to see the responsive effect.

My Website

With a **flexible** layout.

[Link](#) [Link](#) [Link](#) [Link](#)

About Me

Photo of me:

Image

Some text about me in culpa qui officia deserunt mollit anim..

More Text

Lorem ipsum dolor sit ame.

Image

Image

Image

TITLE HEADING

Title description, Dec 7, 2017

Image

Some text..

Sunt in culpa qui officia deserunt mollit anim id est laborum consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco.

TITLE HEADING

Title description, Sep 2, 2017

Image

Some text..

Sunt in culpa qui officia deserunt mollit anim id est laborum consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco.

TRANSITIONS, TRANSFORMS, AND ANIMATION

We've seen CSS used for visual effects like rounded corners, color gradients, and drop shadows that previously had to be created with graphics. In this chapter, we'll look at some CSS3 properties for producing animated interactive effects that were previously possible only with JavaScript or Flash.

Transition Basics.

Transitions are a lot of fun, so let's give them a whirl. When applying a transition, you have a few decisions to make, each of which is set with a CSS property:

- Which CSS property to change (transition-property) (Required)
- How long it should take (transition-duration) (Required)
- The manner in which the transition accelerates (transition-timing-function)
- Whether there should be a pause before it starts (transition-delay)

transition-property

Transitions require a beginning state and an end state. The element as it appears when it first loads is the beginning state. The end state needs to be triggered by a state change such as :hover, :focus, or :active, which is what we'll be using for the examples in this chapter. You could use JavaScript to change the element (such as adding a class attribute) and use that as a transition trigger as well.

transition-property

Values: property-name | all | none

Default: all

Applies to: all elements, :before and :after pseudo-elements

Inherits: no

transition-duration

transition-duration sets the amount of time it takes for the animation to complete in seconds (s) or milliseconds (ms). I've chosen .3 seconds, which is just enough to notice something happened but not so long that the transition feels sluggish or slows the user down. There is no correct duration, of course, but I've found that .2s seems to be a popular transition time for UI elements. Experiment to find the duration that makes sense for your application.

transition-duration

Values: time

Default: 0s

Applies to: all elements, :before and :after pseudo-elements

Inherits: no

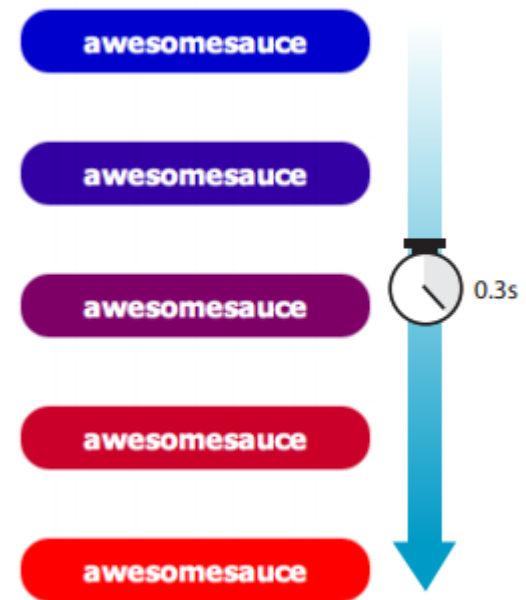
Example

THE MARKUP

```
<a href="..." class="smooth">awesomesauce</a>
```

THE STYLES

```
.smooth {  
  display: block;  
  text-decoration: none;  
  text-align: center;  
  padding: 1em 2em;  
  width: 10em;  
  border-radius: 1.5em;  
  color: #fff;  
  background-color: mediumblue;  
  transition-property: background-color;  
  transition-duration: 0.3s;  
}  
.smooth:hover, .smooth:focus {  
  background-color: red;  
}
```



transition-timing-function

The timing function you choose can have a big impact on the feel and believability of the animation, so if you plan on using transitions and CSS animations, it is a good idea to get familiar with the options. If I set the transition-timing-function to ease-in-out, the transition will start out slow, then speed up, then slow down again as it comes to the end state.

Values: ease | linear | ease-in | ease-out | ease-in-out |

Default: ease

Applies to: all elements, :before and :after pseudo-elements

Inherits: no.

transition-timing-function

```
.smooth {  
  ...  
  transition-property: background-color;  
  transition-duration: 0.3s;  
  transition-timing-function: ease-in-out;  
}
```

The transition-timing-function property takes one of the following keyword values:

ease

Starts slowly, accelerates quickly, and then slows down at the end. This is the default value and works just fine for most short transitions.

linear

Stays consistent from the transition's beginning to end. Because it is so consistent, some say it has a mechanical feeling.

ease-in

Starts slowly, then speeds up.

ease-out

Starts out fast, then slows down.

ease-in-out

Starts slowly, speeds up, and then slows down again at the very end. It is similar to ease, but with less pronounced acceleration in the middle.

transition-delay

Values: time

Default: 0s

Applies to: all elements, :before and :after pseudo-elements

Inherits: no

The transition-delay property, as you might guess, delays the start of the animation by a specified amount of time. In the following example, the background color transition starts .2 seconds after the pointer moves over the link.

```
.smooth {
```

```
...
```

```
  transition-property: background-color;
```

```
  transition-duration: 0.3s;
```

```
  transition-timing-function: ease-in-out;
```

```
  transition-delay: 0.2s;
```

```
}
```

The Shorthand transition Property

Thankfully, the authors of the CSS3 spec had the good sense to give us the

shorthand transition property to combine all of these properties into one

declaration. You've seen this sort of thing with the shorthand border property.

Here is the syntax:

`transition: property duration timing-function delay;`

Using the blue-to-red link example, we could combine the four transition

properties we've applied so far into this one line:

```
.smooth {
```

```
...
```

```
transition: background-color 0.3s ease-in-out 0.2s;
```

```
}
```

TRANSFORMS

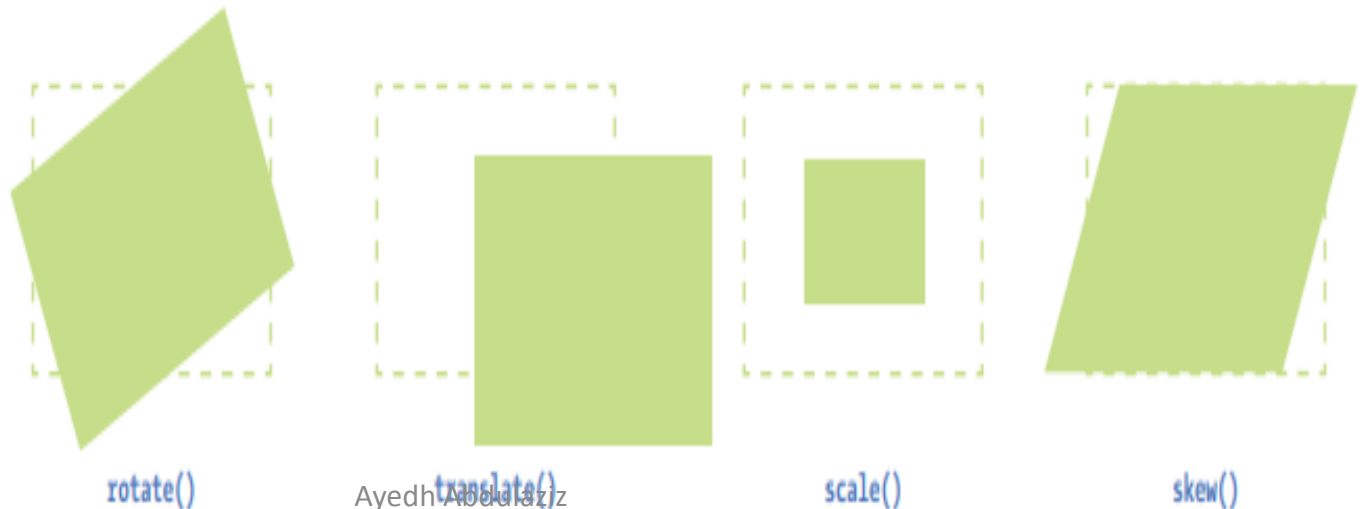
transform

Values: rotate() | rotateX() | rotateY() | rotateZ() | rotate3d() | translate() | translateX() | translateY() | scale() | scaleX() | scaleY() | skew() | skewX() | skewY() | none

Default: none

Applies to: transformable elements (img, canvas, form inputs, display set to block, inline-block, inline flex)

Inherits: no



Transforming the Angle (rotate)

```
img {  
  width: 400px;  
  height: 300px;  
  transform: rotate(-10deg);  
}
```



`transform: rotate(-10deg);`

Transforming the Position (translate)

Another thing you can do with the transform property is give the element's rendering a new location on the page by using one of three translate() functions.



`transform: translate(90px, 60px);`



`transform: translate(-5%, -25%);`

Transforming the Size (scale)

Make an element appear larger or smaller by using one of three scale functions: `scaleX()` (horizontal), `scaleY()` (vertical), and the shorthand `scale()`.

This example makes an image 150% its original width: a

```
img {
```

```
  transform: scaleX(1.5);
```

```
}
```

The `scale()` shorthand lists a value for `scaleX` and a value for `scaleY`. This example makes an element twice as wide but half as tall as the original:

```
a img {
```

```
  transform: scale(2, .5);
```

```
}
```

Unlike translate(),

Unlike translate(), however, if you provide only one value for scale(), it will be used as the scaling factor in both directions. So specifying scale(2) is the same as applying scaleX(2) and scaleY(2), which is intuitively the way you'd want it to be.



```
transform: scale(1.25);
```



```
transform: scale(.75);
```



```
transform: scale(1.5, .5);
```

Making It Slanty (skew)

The quirky collection of skew properties—`skewX()`, `skewY()`, and the shorthand `skew()`—changes the angle of either the horizontal or vertical axis (or both axes) by a specified number of degrees. As for `translate()`, if you provide only one value, it is used for `skewX()`, and `skewY()` will be set to zero.

```
a img {  
  transform: skewX(15deg);  
}  
  
a img {  
  transform: skewY(30deg);  
}  
  
a img {  
  transform: skew(15deg, 30deg);  
}
```



```
transform: skewX(15deg);
```



```
transform: skewY(30deg);
```



```
transform: skew(15deg, 30deg);
```

Applying Multiple Transforms

It is possible to apply more than one transform to a single element by listing out the functions and their values, separated by spaces, like this:

transform: function(value) function(value);

Ex: `img:hover, img:focus {`

`transform: scale(1.5) rotate(-5deg) translate(50px,30px);`

`}`

Normal state



`:hover, :focus`
`rotate(), translate(), and scale()` applied



KEYFRAME ANIMATION

Creating keyframe animations is complex, and more than I can cover here. But I would like for you to have some idea of how it works, so I'll sketch out the minimal details.

Establishing the Keyframes .

The animation process has two parts:

1. Establish the keyframes with a `@keyframes` rule.
2. Add the animation properties to the elements that will be animated.

Here is a very simple set of keyframes that changes the background color of an element over time.

@keyframes example

```
div {  
  width: 100px;  
  height: 100px;  
  background-color: red;  
  position: relative;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-iteration-count: 3;  
  animation-delay: 2s;  
}
```

```
@keyframes example {  
  0% {background-color:red; left:0px; top:0px;}  
  25% {background-color:yellow; left:200px; top:0px;}  
  50% {background-color:blue; left:200px; top:200px;}  
  75% {background-color:green; left:0px; top:200px;}  
  100% {background-color:red; left:0px; top:0px;}  
}
```

<p>Note: This example does not work in Internet Explorer 9 and earlier versions.</p>

Run Animation in Reverse Direction or Alternate Cycles

The animation-direction property specifies whether an animation should be played forwards, backwards or in alternate cycles.

The animation-direction property can have the following values:

normal - The animation is played as normal (forwards). This is default

reverse - The animation is played in reverse direction (backwards)

alternate - The animation is played forwards first, then backwards

alternate-reverse - The animation is played backwards first, then forwards