جامعة فلسطين التقنية
خضوري
Palestine Technical University
Kadoorei

برمجة وتطبيقات الأعمال الالكترونية

قسم إدارة الأعمال والتجارة الإلكترونية
كلية الأعمال والاقتصاد

أ. وليد أحمد

waleed.ahmad@ptuk.edu.ps

What is HTML?

Welcome to HTML!

- HTML stands for **H**yper**T**ext **M**arkup **L**anguage.

- Unlike a scripting or programming language that uses scripts to perform functions, a markup language uses tags to identify content.

- Here is an example of an HTML tag:
   I'm a paragraph

The Web Structure

- The ability to code using HTML is essential for any web professional. Acquiring this skill should be the starting point for anyone who is learning how to create content for the web.

- **Modern Web Design:**
  HTML: Structure
  CSS: Presentation
  JavaScript: Behavior

  PHP or similar: Backend
  CMS: Content Management

- HTML is easy to learn. So don't wait! Dive right in!

Basic HTML Document Structure

The <html> Tag

- Although various versions have been released over the years, HTML basics remain the same.

- The structure of an HTML document has been compared with that of a sandwich. As a sandwich has two slices of bread, the HTML document has opening and closing HTML tags.

- These tags, like the bread in a sandwich, surround everything else:
  <html>
  …
  </html>

- Everything in an HTML document is surrounded by the <html> tag.

The <head> Tag

- Immediately following the opening HTML tag, you'll find the **head** of the document, which is identified by opening and closing head tags.

- The head of an HTML file contains all of the **non-visual elements** that help make the page work.

```
<html>
    <head>…</head>
</html>
```

- The head section elements will be discussed later.

The <body> Tag

- The **body** tag follows the head tag.
- All visual-structural elements are contained within the body tag.

- Headings, paragraphs, lists, quotes, images, and links are just a few of the elements that can be contained within the body tag.

- **Basic HTML Structure:**

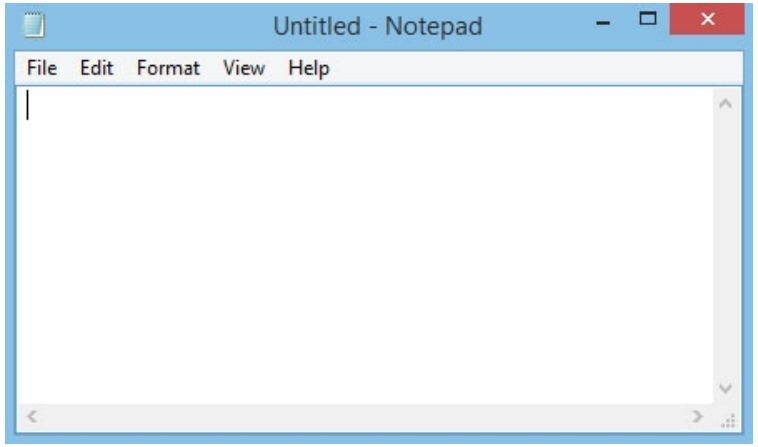
```
<html>
    <head>
    </head>
    <body>
    </body>
</html>
```

- The <body> tag defines the main content of the HTML document.

Creating Your First HTML Page

The HTML File

- HTML files are text files, so you can use any **text editor** to create your first webpage.
- There are some very nice HTML editors available; you can choose the one that works for you. For now let's write our examples in **Notepad**.



- You can run, save, and share your HTML codes on our **Code Playground**, without installing any additional software.

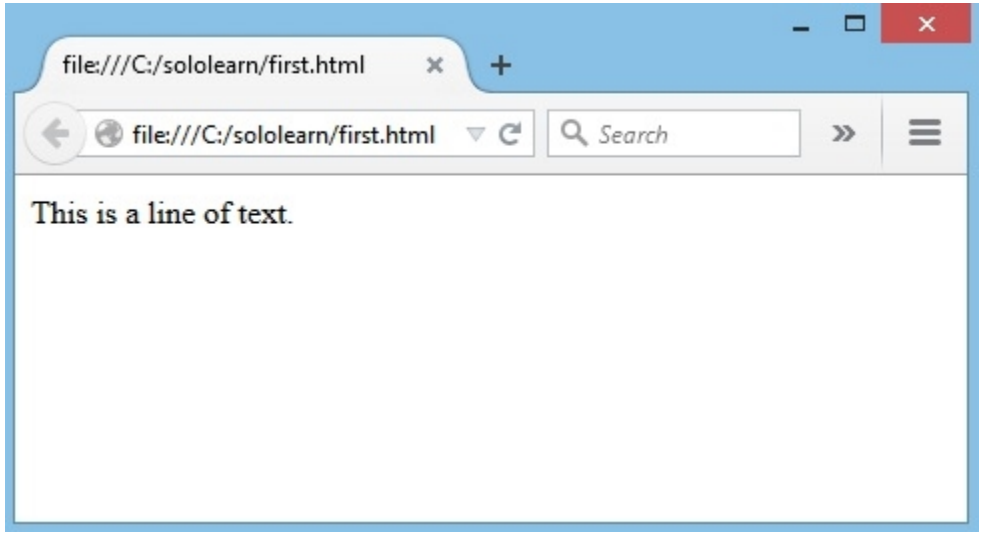
The HTML File

- Add the basic HTML structure to the text editor with "This is a line of text" in the body section.

```
<html>
    <head>
    </head>
    <body>
        This is a line of text.
    </body>
</html>
```

- In our example, the file is saved as **first.html**

- When the file is opened, the following result is displayed in the web browser:



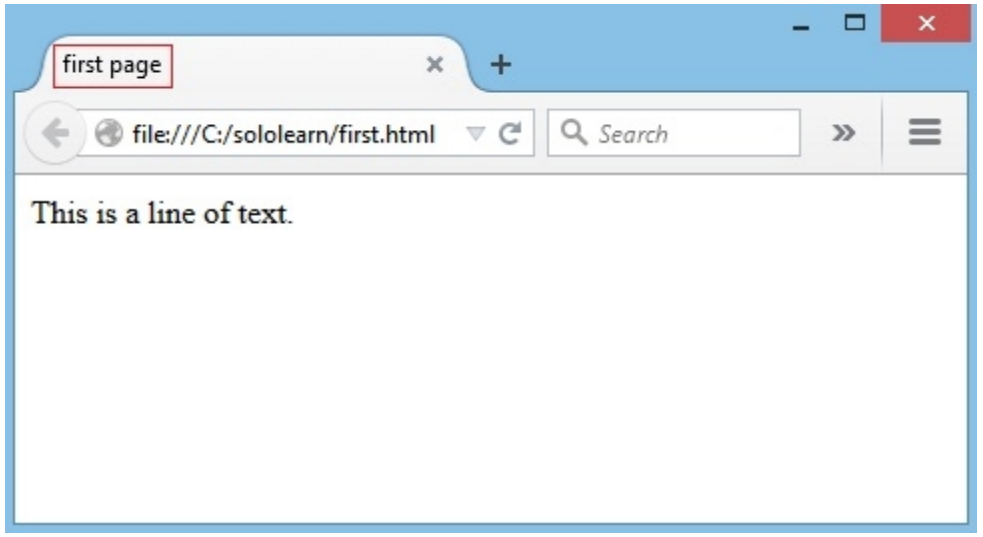
- Don’t forget to save the file. HTML file names should end in either **.html** or **.htm**

The <title> Tag

- To place a title on the tab describing the web page, add a <title> element to your head section:

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        This is a line of text.
    </body>
</html>
```

- This will produce the following result:



- The title element is important because it describes the page and is used by search engines.

Favicon

- Adding a Favicon to your Website:

  <link rel="icon" href="demo_icon.gif" type="image/gif" sizes="16x16">

  or:

  <link rel="icon" href=http://example.com/favicon.ico type="image/gif" sizes="16x16" />

- Apple iOS Home Screen Icons:

- iOS will add rounded corners and a reflective shine to your iOS home screen icon.
  <link rel="apple-touch-icon" href="http://example.com/images/apple-touch-icon.png" />

- If you prefer it without the reflective shine, use the following code:
  <link rel="apple-touch-icon-precomposed" href="http://example.com/images/apple-touch-icon.png" />

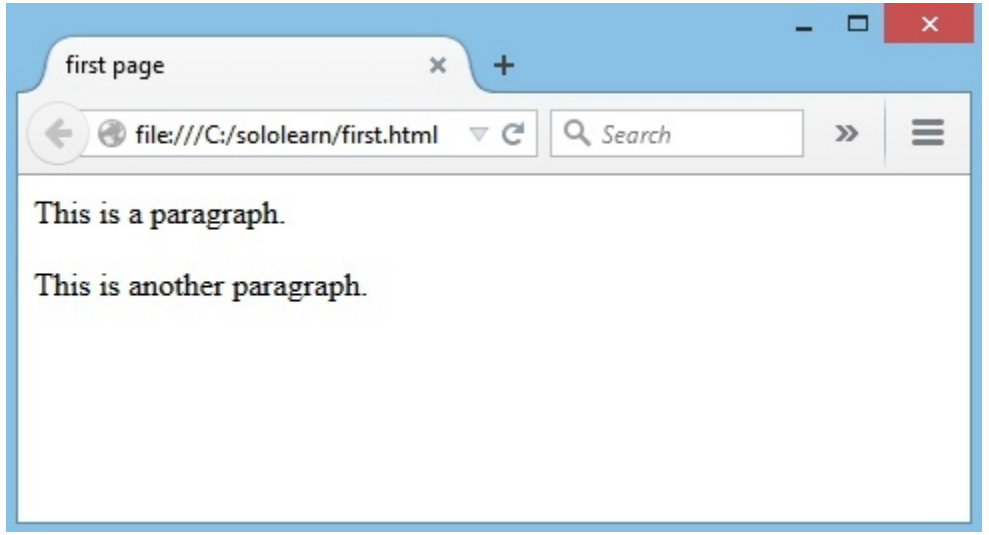
Paragraphs

The <p> Element

- To create a paragraph, simply type in the **<p>** element with its opening and closing tags:

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <p>This is a paragraph. </p>
        <p>This is another paragraph.
</p>
    </body>
</html>
```

- The result:



- Browsers automatically add an empty line before and after a paragraph.

Single Line Break

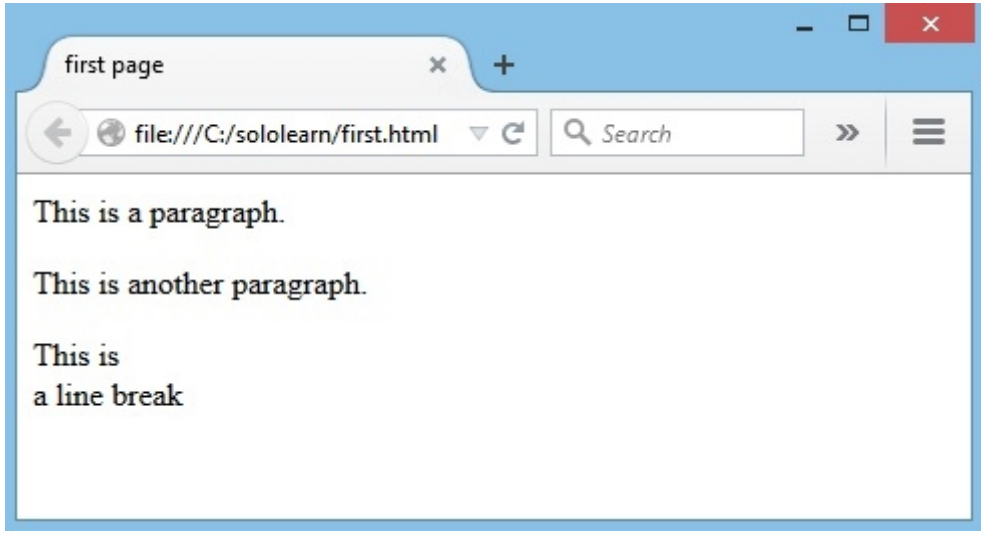
- Use the **
** tag to add a single line of text without starting a new paragraph:

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <p>This is a paragraph.</p>
        <p>This is another paragraph. </p>
        <p>This is <br /> a line break </p>
    </body>
</html>
```

- The
 element is an empty HTML element. It has no end tag.

Single Line Break

- Opening the HTML file in the browser shows that a single line break has been added to the paragraph:



- The
 element has no end tag.

Text Formatting

Formatting Elements

- In HTML, there is a list of elements that specify text style.

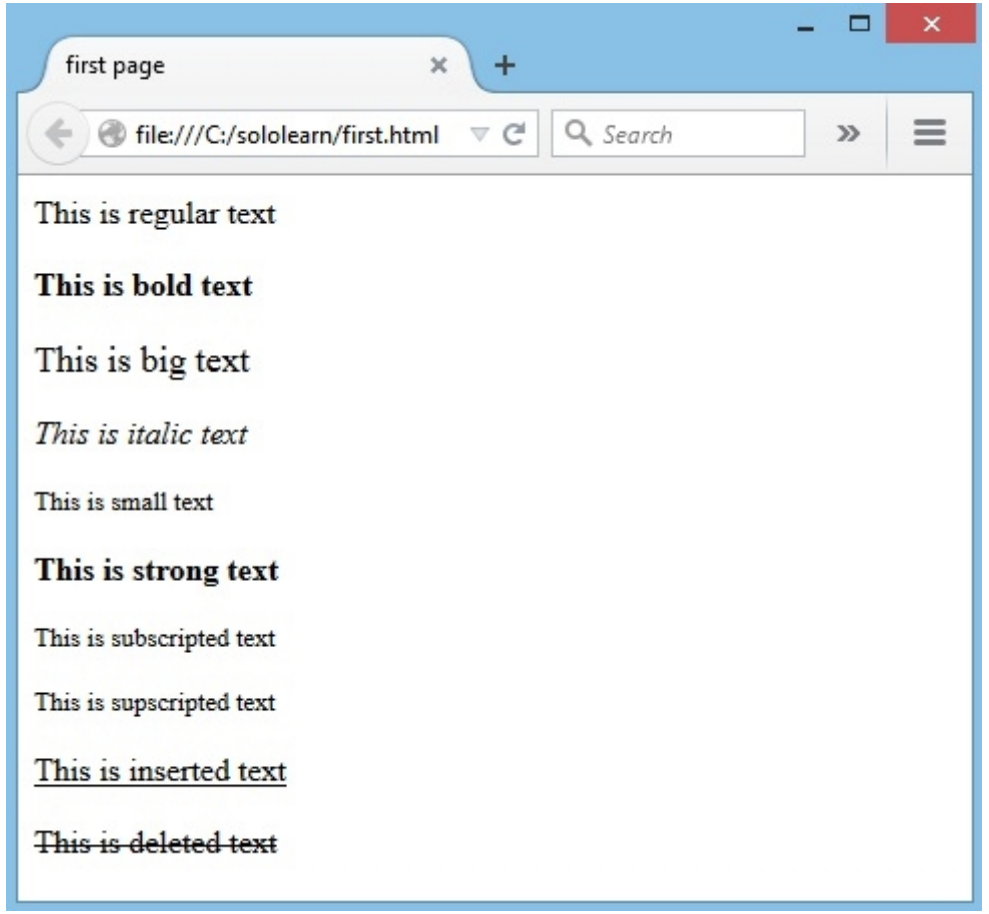
- Formatting elements were designed to display special types of text:

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <p>This is regular text </p>
        <p><b> bold text </b></p>
        <p><big> big text </big></p>
        <p><i> italic text </i></p>
        <p><small> small text </small></p>
        <p><strong> strong text </strong></p>
        <p><sub> subscripted text </sub></p>
        <p><sup> superscripted text </sup></p>
        <p><ins> inserted text </ins></p>
        <p><del> deleted text </del></p>
    </body>
</html>
```

Formatting Elements

- Each paragraph in the example is formatted differently to demonstrate what each tag does:

- Browsers display as <b>, and as <i>.
- However, the meanings of these tags differ: <b> and <i> define bold and italic text, respectively, while and indicate that the text is "**important**".



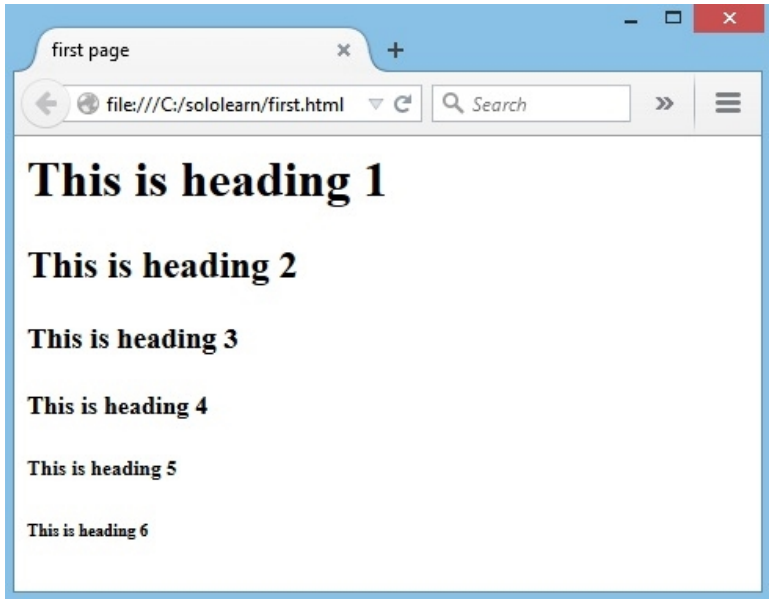
Headings, Lines, Comments

HTML Headings

- HTML includes six levels of headings, which are ranked according to importance. These are **<h1>**, **<h2>**, **<h3>**, **<h4>**, **<h5>**, and **<h6>**.
- The following code defines all of the headings:

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <h1>This is heading 1</h1>
        <h2>This is heading 2</h2>
        <h3>This is heading 3</h3>
        <h4>This is heading 4</h4>
        <h5>This is heading 5</h5>
        <h6>This is heading 6</h6>
    </body>
</html>
```

- Result:



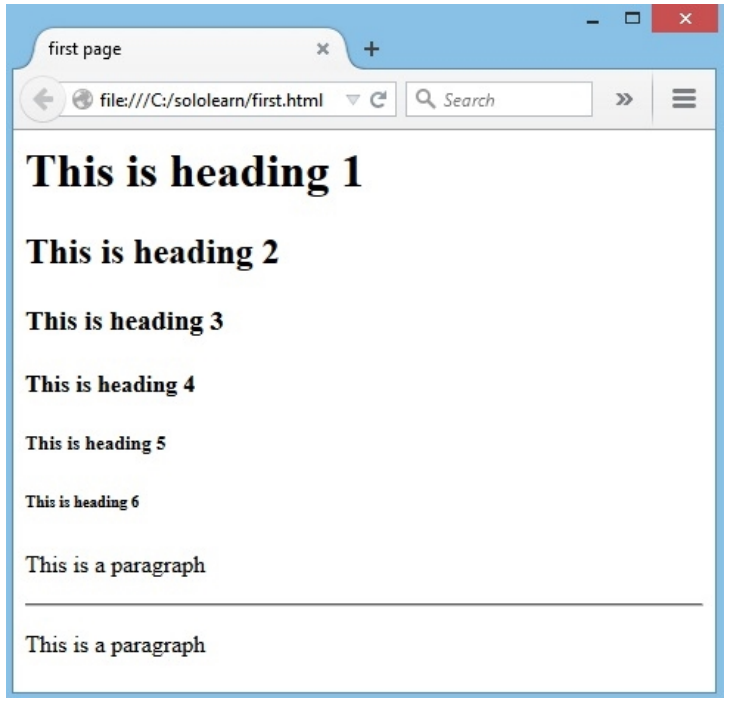
- It is not recommended that you use headings just to make the text big or bold, because search engines use headings to index the web page structure and content.

Horizontal Lines

- To create a horizontal line, use the **<hr />** tag.

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <h1>This is heading 1</h1>
        <h2>This is heading 2</h2>
        <h3>This is heading 3</h3>
        <h4>This is heading 4</h4>
        <h5>This is heading 5</h5>
        <h6>This is heading 6</h6>
        <p>This is a paragraph </p>
        <hr />
        <p>This is a paragraph </p>
    </body>
</html>
```

- Result:



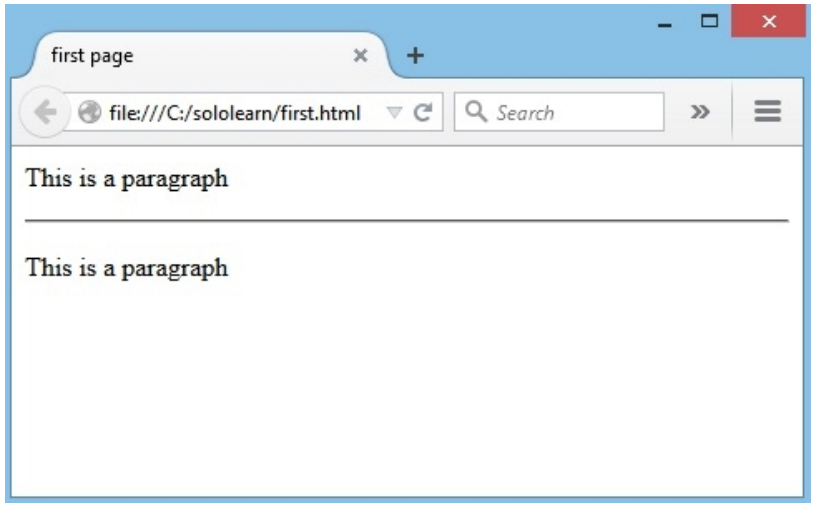
- In HTML5, the <hr> tag defines a thematic break.

Comments

- The browser does not display comments, but they help document the HTML and add descriptions, reminders, and other notes.
  <!-- Your comment goes here **-->**
- **Example:**

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <p>This is a paragraph </p>
        <hr />
        <p>This is a paragraph </p>
        <!-- This is a comment -->
    </body>
</html>
```

- Result:



- As you can see, the comment is not displayed in the browser.
- There is an exclamation point (!) in the opening tag, but not in the closing tag.

Elements

HTML Elements

- HTML documents are made up of HTML elements.
- An HTML element is written using a **start tag** and an **end tag**, and with the **content** in between.

- HTML documents consist of nested HTML elements. In the example below, the body element includes the <p> tags, the
 tag and the content, "This is a paragraph".

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <p>This is a paragraph <br /></p>
    </body>
</html>
```

- Some HTML elements (like the **
** tag) do not have end tags.

HTML Elements

- Some elements are quite small. Since you can't put contents within a break tag, and you don't have an opening and closing break tag, it’s a separate, single element.

- So HTML is really scripting with elements within elements.<html>

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <p>This is a paragraph</p>
        <p>This is a <br /> line break</p>
    </body>
</html>
```

- Some HTML elements (like the **
** tag) do not have end tags.

Attributes

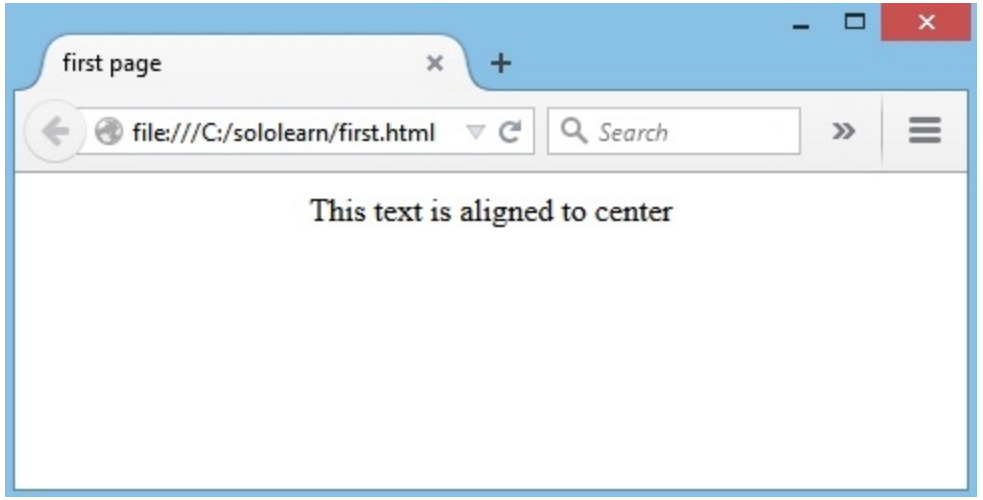
HTML Attributes

- Attributes provide **additional information** about an element or a tag, while also **modifying** them. Most attributes have a value; the value modifies the attribute.

```
<p align="center">
    This text is aligned to center
</p>
```

- In this example, the value of "center" indicates that the content within the p element should be aligned to the center:

- Result:



- Attributes are always specified in the start tag, and they appear in name="value" pairs.

Attribute Measurements

- As an example, we can modify the horizontal line so it has a width of 50 pixels.

- This can be done by using the width attribute:
  <hr **width**="50**px**" />

- An element's width can also be defined using percentages:
  <hr **width**="50**%**" />

- An element's width can be defined using pixels or percentages.

The Align Attribute

- The **align** attribute is used to specify how the text is aligned.

- In the example below, we have a paragraph that is aligned to the center, and a line that is aligned to the right.

```
<html>
    <head>
        <title>Attributes</title>
    </head>
    <body>
        <p align="center">This is a text <br />
            <hr width="10%" align="right" /> This is also a text.
        </p>
    </body>
</html>
```

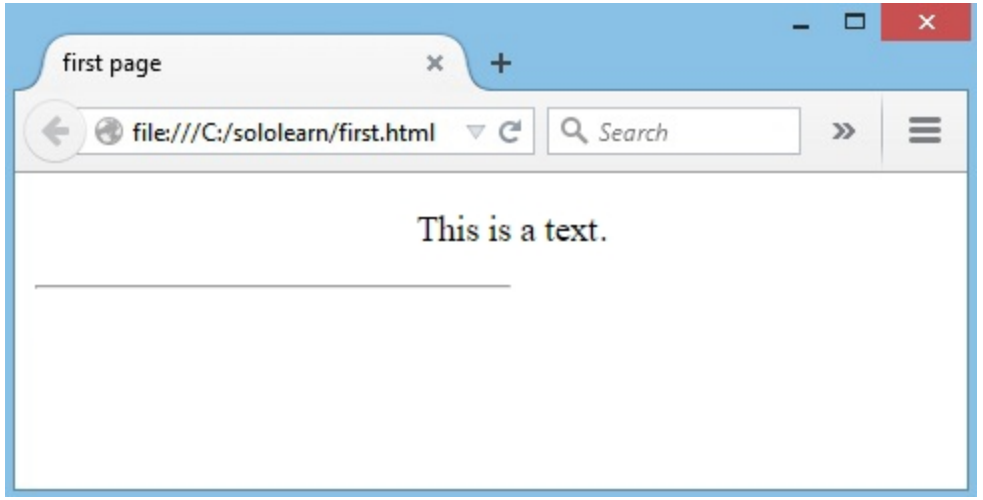
- The align attribute of <p> is not supported in HTML5.

Attributes

- You may be wondering what happens if you try to apply contradictory attributes within the same element.

```
<p align="center">
  This is a text.
  <hr width="50%" align="left" />
</p>
```

- Result:



- The align attribute of <p> is not supported in HTML5.

Images

The <img> Tag

- The **<img>** tag is used to insert an image. It contains only attributes, and does not have a closing tag.

- The image's URL (address) can be defined using the **src** attribute.

- The HTML image syntax looks like this:
  ****

- The alt attribute specifies an alternate text for an image.

Image Location

- You need to put in the **image location** for the src attribute that is between the quotation marks.

- For example, if you have a photo named "tree.jpg" in the same folder as the HTML file, your code should look like this:

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <img src="tree.jpg" alt="" />
    </body>
</html>
```

- In case the image cannot be displayed, the alt attribute specifies an alternate text that describes the image in words. The alt attribute is **required**.

Image Resizing

- To define the image size, use the width and height attributes.
- The value can be specified in **pixels** or as a **percentage:**

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <img src="tree.jpg" height="150px" width="150px" alt="" />
        <!-- or -->
        <img src="tree.jpg" height="50%" width="50%" alt="" />
    </body>
</html>
```

- Loading images takes time. Using large images can slow down your page, so use them with care.

Image Border

- By default, an image has no borders. Use the border attribute within the image tag to create a border around the image.

- By default, Internet Explorer 9, and its earlier versions, display a border around an image unless a border attribute is defined.

Links

The <a> Tag

- Links are also an integral part of every web page. You can add links to text or images that will enable the user to click on them in order to be directed to another file or webpage.
- In HTML, links are defined using the **<a>** tag.

- Use the **href** attribute to define the link's destination address:
  <a **href**="">

- To link an image to another document, simply nest the <img> tag inside <a> tags.

Creating Your First Link

- In the example below, a link to SoloLearn's website is defined:
   Visit PTUK

- Once the code has been saved, "Visit PTUK" will display as a link:
  Visit PTUK

- Clicking on "**Visit PTUK**" redirects you to www.ptuk.edu.ps

- Links can be either absolute or relative.

The target Attribute

- The **target** attribute specifies where to open the linked document.

- Giving a **_blank** value to your attribute will have the link open in a new window or new tab:

```
<a href="https://www.ptuk.edu.ps" target="_blank">
    Visit PTUK
</a>
```

- A visited link is underlined and purple.

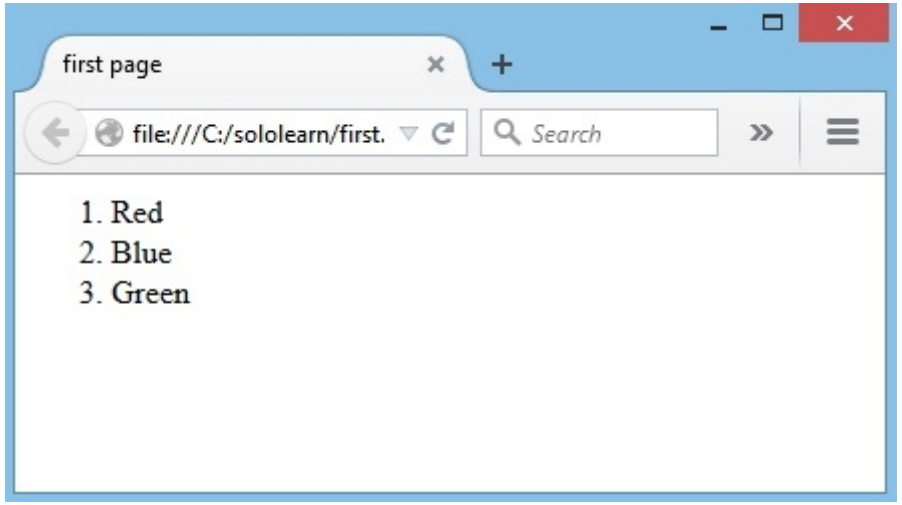
Lists

HTML Ordered Lists

- An ordered list starts with the **<ol>** tag, and each list item is defined by the **<li>** tag.
- Here is an example of an **ordered list**:

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <ol>
            <li>Red</li>
            <li>Blue</li>
            <li>Green</li>
        </ol>
    </body>
</html>
```

- Result:



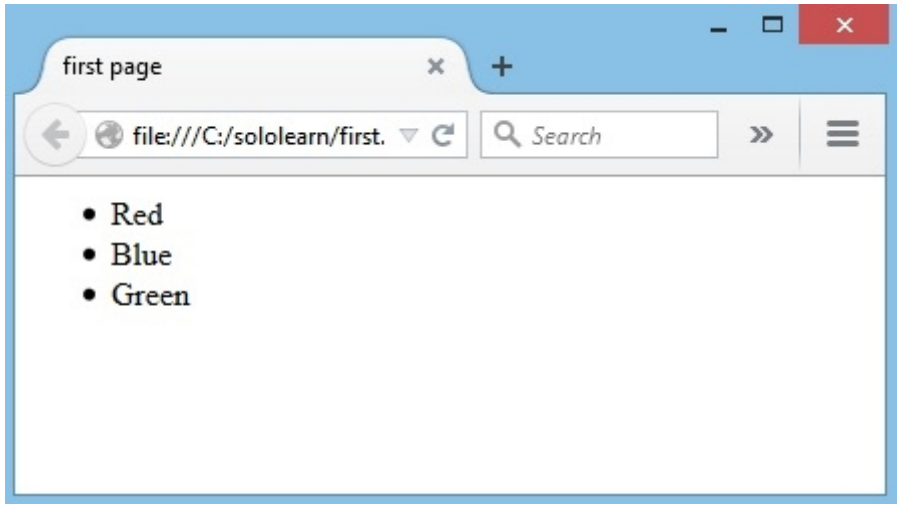
- The list items will be automatically marked with numbers.

HTML Unordered Lists

- An unordered list starts with the **<ul>** tag.

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body>
        <ul>
            <li>Red</li>
            <li>Blue</li>
            <li>Green</li>
        </ul>
    </body>
</html>
```

- Result:



- The list items will be marked with bullets.

Tables

Creating a Table

- Tables are defined by using the **<table>** tag.
- Tables are divided into table rows with the **<tr>** tag.
- Table rows are divided into table columns (table data) with the <**td**> tag.

- Here is an example of a table with **one row** and **three columns:**

```
<table>
    <tr>
        <td></td>
        <td></td>
        <td></td>
    </tr>
</table>
```

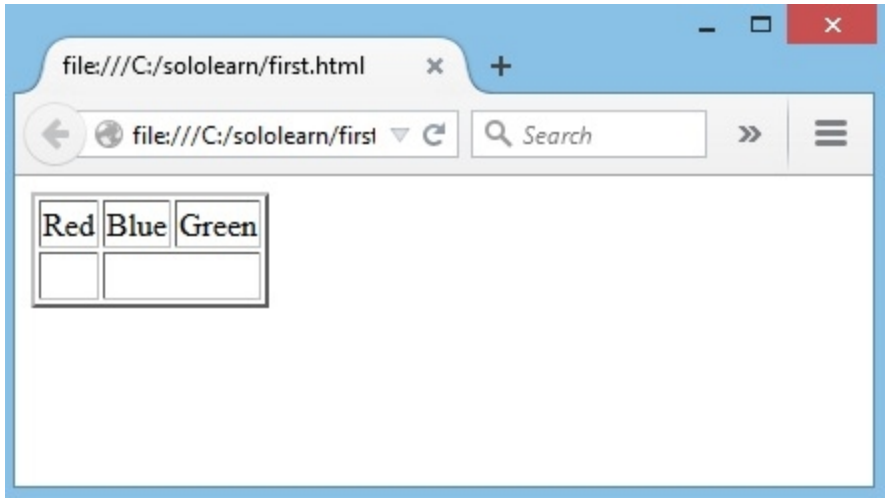
- Table data tags <td> act as data containers within the table.
- They can contain all sorts of HTML elements, such as text, images, lists, other tables, and so on.

The border and colspan Attributes

- A border can be added using the **border** attribute:
  <table **border**="2">
- A table **cell** can span two or more columns:

```
<table border="2">
    <tr>
        <td>Red</td>
        <td>Blue</td>
        <td>Green</td>
    </tr>
    <tr>
        <td><br /></td>
        <td colspan="2"><br /></td>
    </tr>
</table>
```

- Result:



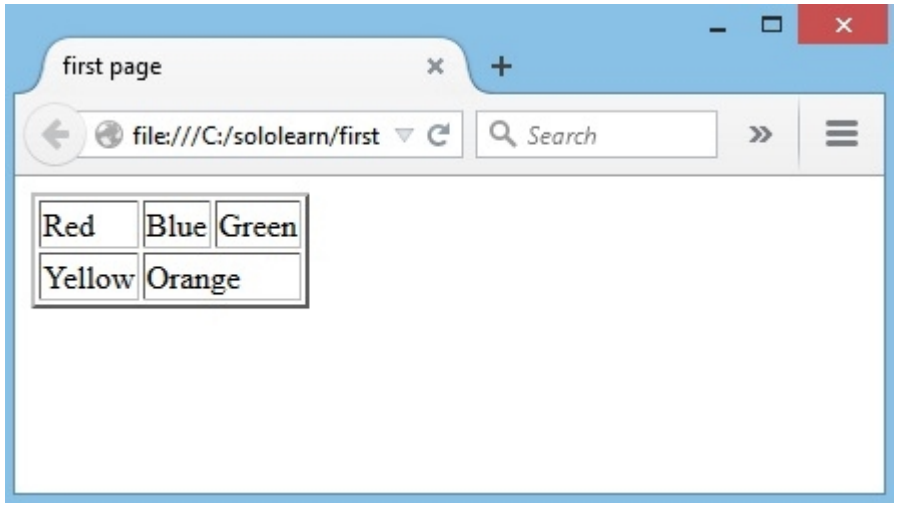
- The border attribute is not supported in HTML5.

Colspan Color

- The example below demonstrates the **colspan** attribute in action:

```
<table border="2">
    <tr>
        <td>Red</td>
        <td>Blue</td>
        <td>Green</td>
    </tr>
    <tr>
        <td>Yellow</td>
        <td colspan="2">Orange</td>
    </tr>
</table>
```

- Result:



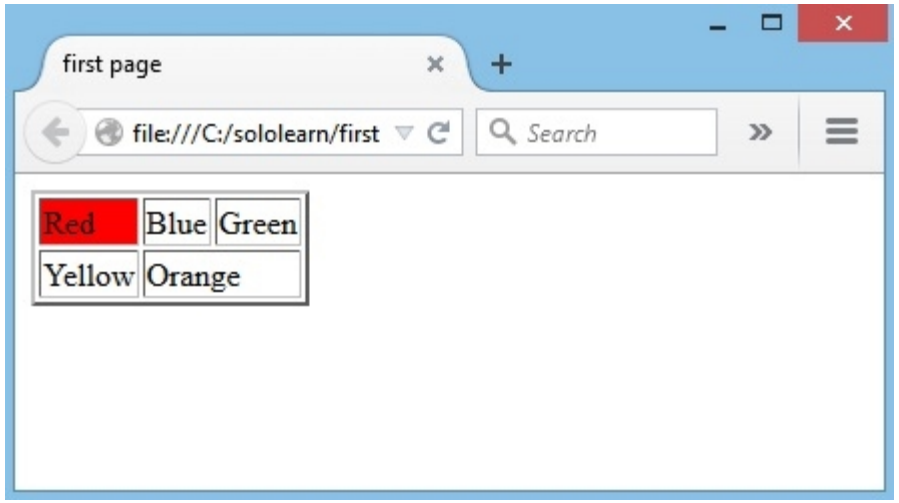
- You can see that the cell containing "Orange" spans two cells.
- To make a cell span more than one row, use the **rowspan** attribute.

The align and bgcolor Attributes

- To change your table's position, use the **align** attribute inside your table tag: <table **align="center"**>
- Now let's specify a background color of red for a table cell. To do that, just use the bgcolor attribute.

```
<table border="2">
    <tr>
        <td bgcolor=”red">Red</td>
        <td>Blue</td>
        <td>Green</td>
    </tr>
    <tr>
        <td>Yellow</td>
        <td colspan="2">Orange</td>
    </tr>
</table>
```

- Result:



- In the case of styling elements, CSS is more effective than HTML. Try our free "**Learn CSS**" course to learn more about CSS and styles.

Inline and Block Elements

Types of Elements

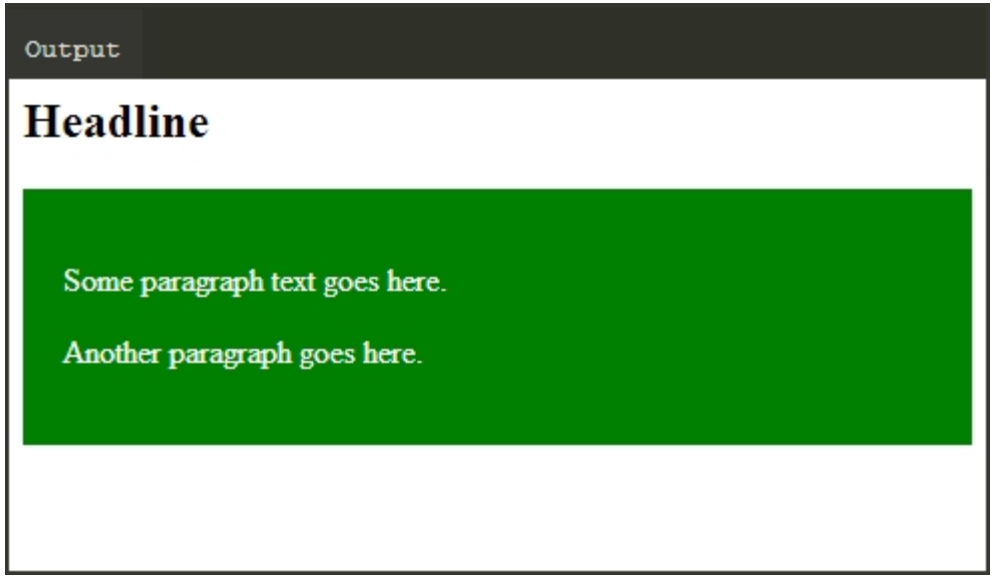
- In HTML, most elements are defined as **block level** or **inline** elements. Block level elements start from a new line.
- **For example**: <h1>, <form>, <li>, <ol>, <ul>, <p>, <pre>, <table>, , etc.

- Inline elements are normally displayed without line breaks.
- **For example**: <b>, <a>, , <img>, <input>, , , etc.

- The ** ** element is a block-level element that is often used as a **container for other HTML elements**.
- When used together with some CSS styling, the element can be used to style blocks of content:

Types of Elements

```
<html>
    <body>
        <h1>Headline</h1>
        <div style="background-color:green; color:white; padding:20px;">
            <p>Some paragraph text goes here.</p>
            <p>Another paragraph goes here.</p>
        </div>
    </body>
</html>
```



Types of Elements

- Similarly, the **** element is an inline element that is often used as a container for some text.
- When used together with CSS, the element can be used to style **parts of the text**:

```
<html>
    <body>
        <h2>Some
        <span
style="color:red">Important</span>
Message</h2>
    </body>
</html>
```

- **Summary**
  The element defines a **block-level** section in a document.
  The element defines an **inline** section in a document.

Types of Elements

- Other elements can be used either as block level elements or inline elements. This includes the following elements:
  APPLET - embedded Java applet
  IFRAME - Inline frame
  INS - inserted text
  MAP - image map
  OBJECT - embedded object
  SCRIPT - script within an HTML document

- You can insert inline elements inside block elements. For example, you can have multiple **** elements inside a ** ** element.

- Inline elements **cannot** contain any block level elements.

Forms

The <form> Element

- HTML forms are used to collect information from the user.
- Forms are defined using the **<form>** element, with its opening and closing tags:

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

- Use the **action** attribute to point to a webpage that will load after the user submits the form.<**form action**="http://www.sololearn.com"> </**form**>

- Usually the form is submitted to a web page on a web server.

The method and name Attributes

- The **method attribute** specifies the HTTP method (**GET** or **POST**) to be used when forms are submitted (see below for description):
  <form action="url" **method="GET"**>
  <form action="url" **method="POST"**>

- When you use **GET**, the form data will be visible in the page address.

- Use **POST** if the form is updating data, or includes sensitive information (passwords).
- POST offers better security because the submitted data is not visible in the page address.

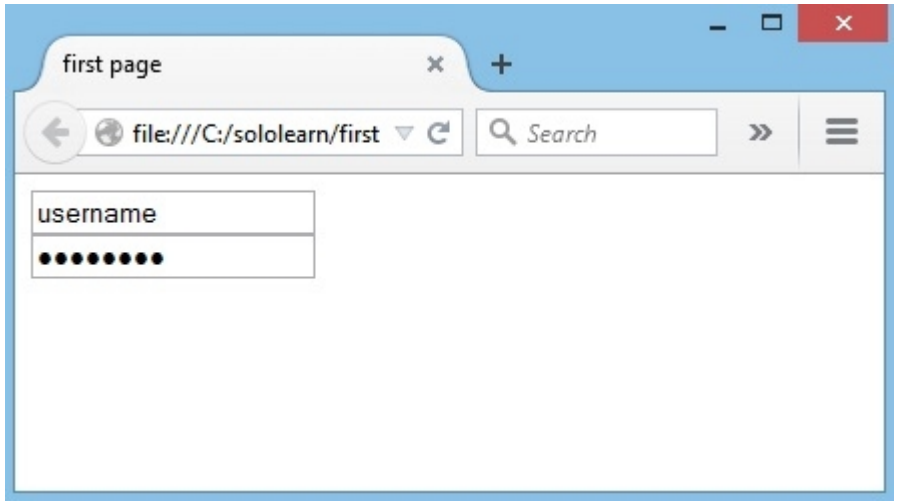
- To take in user input, you need the corresponding form elements, such as text fields. The **<input>** element has many variations, depending on the type attribute. It can be a text, password, radio, URL, submit, etc.

The method and name Attributes

- The example below shows a form requesting a username and password:

```
<form>
    <input type="text"
name="username" /><br />
    <input type="password"
name="password" />
</form>
```

- Result:



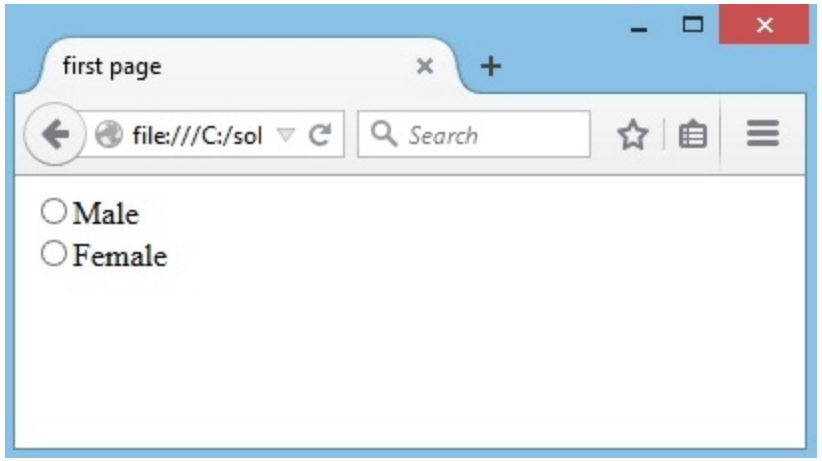
- The name attribute specifies a name for a form.

Form Elements

- If we change the input type to **radio**, it allows the user select only one of a number of choices:

```
<input type="radio"
name="gender" value="male" />
Male <br />
<input type="radio"
name="gender" value="female"
/> Female <br />
```

- Result:

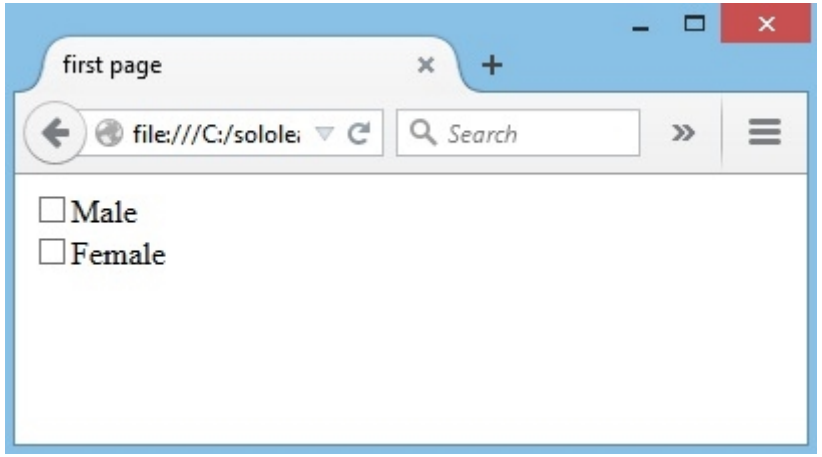


Form Elements

- The type "checkbox" allows the user to select more than one option:

```
<input type="checkbox"
name="gender" value="1" />
Male <br />
<input type="checkbox"
name="gender" value="2" />
Female <br />
```

- Result:



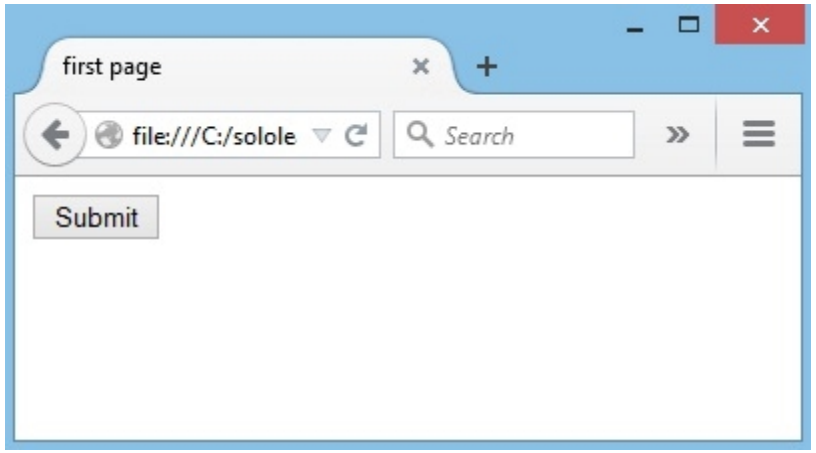
- The <input> tag has no end tag.

Form Elements

- The submit button **submits a form** to its action attribute:

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

- Result:



- After the form is submitted, the data should be processed on the server using a programming language, such as PHP.

HTML Colors

HTML Colors!

- HTML colors are expressed as hexadecimal values.

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

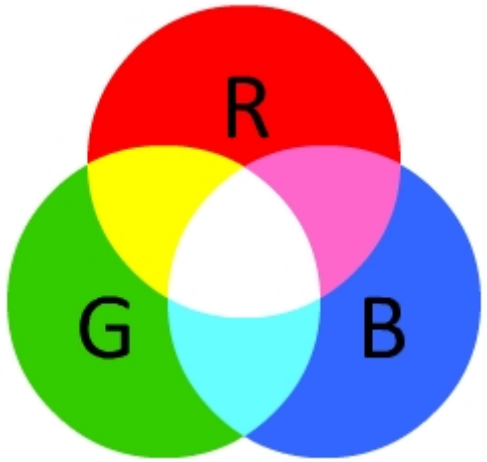
- As you can see, there are 16 values there, 0 through F. Zero represents the lowest value, and F represents the highest.

HTML Color Model

- Colors are displayed in combinations of **red**, **green**, and **blue** light (**RGB**).

- Hex values are written using the hashtag symbol (**#**), followed by either three or six hex characters.

- As shown in the picture below, the circles overlap, forming new colors:



- RGB color values are supported in all browsers.

Color Values

- Hexadecimal color values are supported in all browsers.
- All of the possible **red**, **green**, and **blue** combinations potentially number over 16 million.
- Here are only a few of them:

- We can mix the colors to form additional colors:
  Orange and red mix:

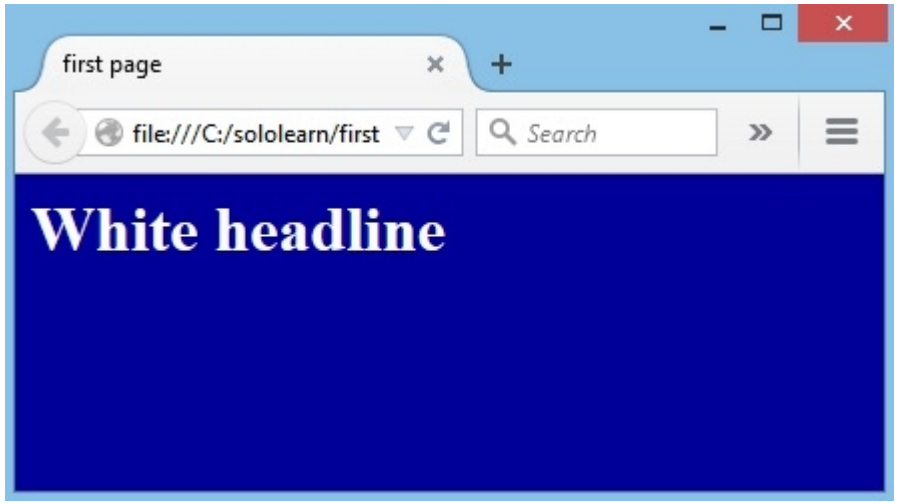
Background and Font Colors

- The **bgcolor** attribute can be used to change the web page's background color.

- This example would produce a dark blue background with a white headline:

```
<html>
    <head>
        <title>first page</title>
    </head>
    <body bgcolor="#000099">
        <h1>
            <font color="#FFFFFF"> White headline </font>
        </h1>
    </body>
</html>
```

- Result:



- The color attribute specifies the color of the text inside a <font> element.

Frames

The <frame> Tag

- A page can be divided into frames using a special frame document.

- The **<frame>** tag defines one specific window (frame) within a **<frameset>**. Each <frame> in a <frameset> can have different attributes, such as border, scrolling, the ability to resize, etc.

- The <frameset> element specifies the number of columns or rows in the frameset, as well as what percentage or number of pixels of space each of them occupies.
  <**frameset cols**="100, 25%, *"></**frameset**>
  <**frameset rows**="100, 25%, *"></**frameset**>

- The <frameset> tag is not supported in HTML5.

Working with Frames

- Use the **<noresize>** attribute to specify that a user cannot resize a <frame> element:
  <frame **noresize="noresize"**>
- Frame content should be defined using the **src** attribute.
- The <frame> tag is not supported in HTML5.
- Lastly, the **<noframes>** element provides a way for browsers that do not support frames to view the page. The element can contain an alternative page, complete with a body tag and any other elements.
-
```
<frameset cols="25%,50%,25%">
    <frame src="a.htm" />
    <frame src="b.htm" />
    <frame src="c.htm" />
    <noframes>Frames not supported!</noframes>
</frameset>
```

YouTube Videos Frames?

Introduction to HTML5

HTML5

- When writing HTML5 documents, one of the first new features that you'll notice is the doc type declaration: **<!DOCTYPE HTML>**

- The character encoding (charset) declaration is also simplified: **<meta charset="UTF-8">**

- **New Elements in HTML5**
  <article>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <footer>, <header>, <nav>, <output>, <progress>, <section>, <video>, and even more!

- The default character encoding in HTML5 is UTF-8.

New in HTML5

- **Forms**
  - The Web Forms 2.0 specification allows for creation of more powerful forms and more compelling user experiences.
  - Date pickers, color pickers, and numeric stepper controls have been added.
  - Input field types now include email, search, and URL.
  - PUT and DELETE form methods are now supported.

- **Integrated API** (Application Programming Interfaces)
  - Drag and Drop
  - Audio and Video
  - Offline Web Applications
  - History
  - Local Storage
  - Geolocation
  - Web Messaging

- You will learn more about these new features in the upcoming lessons.

Content Models

The List of Content Models

- In HTML, elements typically belonged in either the block level or inline content model. HTML5 introduces **seven** main content models.
  - Metadata
  - Embedded
  - Interactive
  - Heading
  - Phrasing
  - Flow
  - Sectioning

- The HTML5 content models are designed to make the markup structure more meaningful for both the browser and the web designer.

Content Models

- **Metadata**: Content that sets up the presentation or behavior of the rest of the content. These elements are found in the **head** of the document.
  Elements: <base>, <link>, <meta>, <noscript>, <script>, <style>, <title>

- **Embedded**: Content that imports other resources into the document.
  Elements: <audio>, <video>, <canvas>, <iframe>, <img>, <math>, <object>, <svg>

- **Interactive**: Content specifically intended for user interaction.
  Elements: <a>, <audio>, <video>, <button>, <details>, <embed>, <iframe>, <img>, <input>, <label>, <object>, <select>, <textarea>

- **Heading**: Defines a section header.
  Elements: <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hgroup>

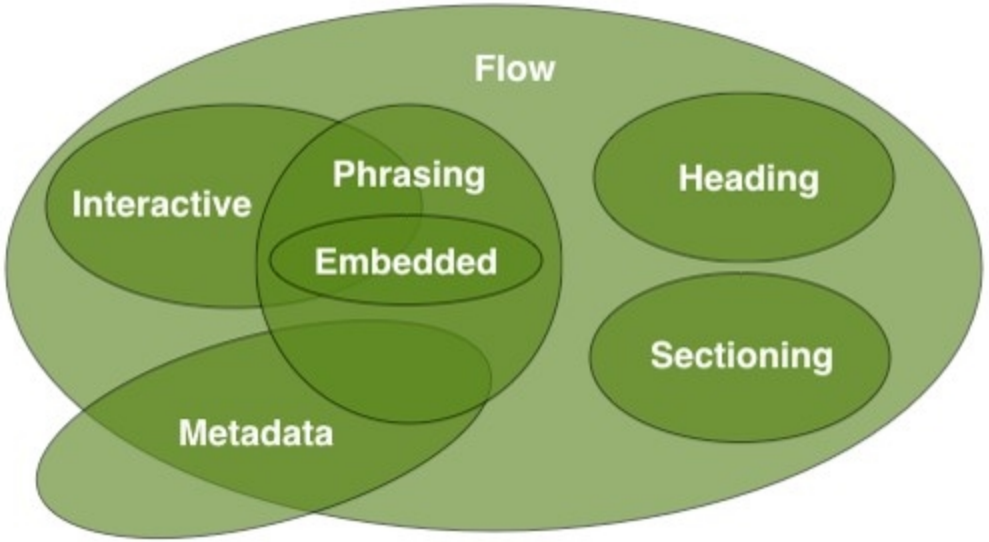
- **Phrasing**: This model has a number of inline level elements in common with HTML4.
  Elements: <img>, , , <label>,
, <small>, <sub>, and more.

- The same element can belong to more than one content model.

Content Models

- **Flow content**: Contains the majority of HTML5 elements that would be included in the normal flow of the document.

- **Sectioning content:** Defines the scope of headings, content, navigation, and footers.
  Elements: <article>, <aside>, <nav>, <section>

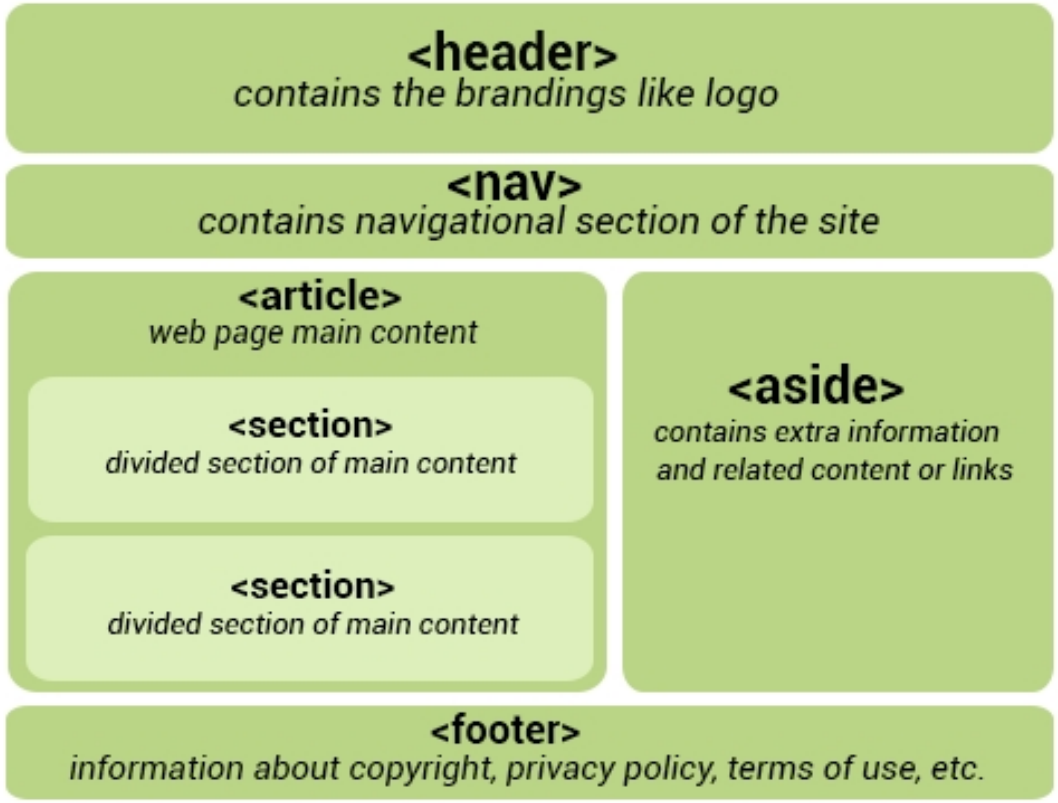


- The various content models overlap in certain areas, depending on how they are being used.

HTML5 Page Structure

Page Structure in HTML5

- A generic HTML5 page structure looks like this:



- You may not need some of these elements, depending on your page structure.

header, nav & footer

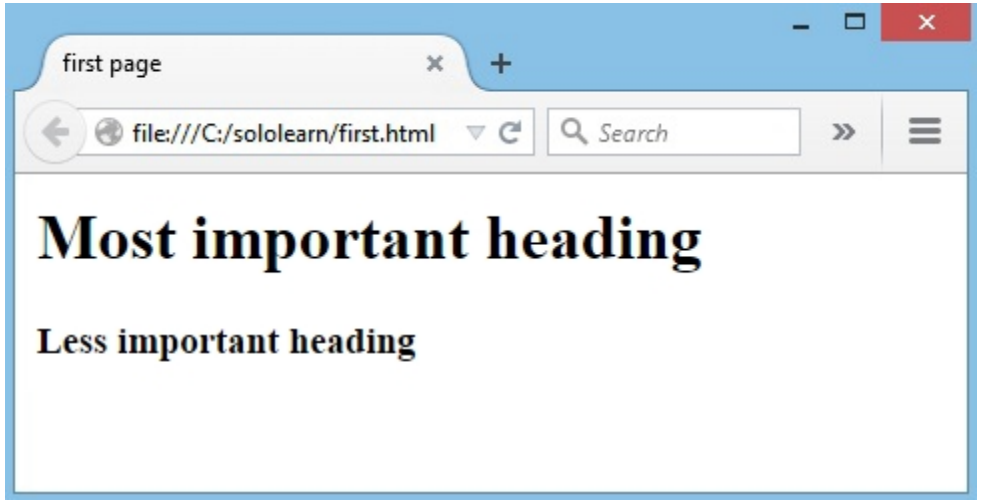
The <header> Element

- In HTML4, we would define a header like this:
  In HTML5, a simple **<header>** tag is used, instead.

- The <header> element is appropriate for use inside the body tag.

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
     <header>
       <h1> Most important heading </h1>
       <h3> Less important heading </h3>
     </header>
   </body>
</html>
```

- Result:



- Note that the <header> is completely different from the <head> tag.

The <footer> Element

- The footer element is also widely used. Generally we refer to a section located at the very bottom of the web page as the footer.

  <footer>…</footer>

- The following information is usually provided between these tags:
  - Contact Information
  - Privacy Policy
  - Social Media Icons
  - Terms of Service
  - Copyright Information
  - Sitemap and Related Documents

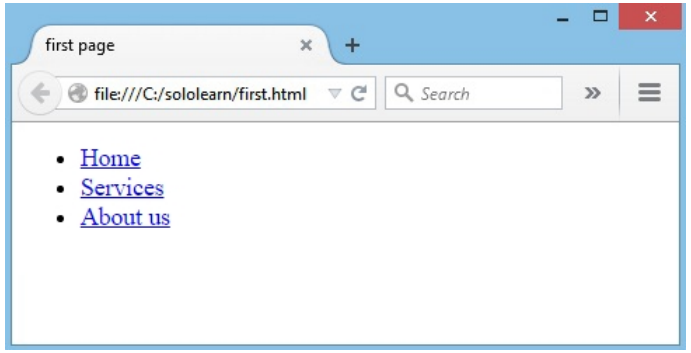
The <nav> Element

- This tag represents a section of a page that links to other pages or to certain sections within the page. This would be a section with navigation links.

- Here is an example of a major block of navigation links:

```
<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">Services</a></li>
    <li><a href="#">About us</a></li>
  </ul>
</nav>
```

- Result:



- Not all of the links in a document should be inside a <nav> element. The <nav> element is intended only for major blocks of navigation links. Typically, the <footer> element often has a list of links that don't need to be in a <nav> element.

article, section & aside

The <article> Element

- **Article** is a self-contained, independent piece of content that can be used and distributed separately from the rest of the page or site. This could be a forum post, a magazine or newspaper article, a blog entry, a comment, an interactive widget or gadget, or any other independent piece of content.

- The <article> element replaces the element that was widely used in HTML4, along with an id or class.

- **<article>**
  <h1>The article title</h1>
  Contents of the article element
  </article>

- When an <article> element is nested, the inner element represents an article related to the outer element. For example, blog post comments can be <article> elements nested in the <article> representing the blog post.

The <section> Element

- **<section>** is a logical container of the page or article.
  Sections can be used to divide up content within an article.
  For example, a homepage could have a section for introducing the company, another for news items, and still another for contact information.

- Each **<section>** should be identified, typically by including a heading (h1-h6 element) as a child of the <section> element.

- <article>
  <h1>Welcome</h1>
  <section>
  <h1>Heading</h1>
  content or image
  </section>
  </article>

- If it makes sense to separately syndicate the content of a <section> element, use an <article> element instead.

The <aside> Element

- **<aside>** is secondary or tangential content which could be considered separate from but indirectly related to the main content.
- This type of content is often represented in sidebars.
- When an <aside> tag is used within an <article> tag, the content of the <aside> should be specifically related to that article.

```
<article>
    <h1> Gifts for everyone </h1>
    <p> This website will be the best place for choosing gifts </p>
    <aside>
        <p> Gifts will be delivered to you within 24 hours </p>
    </aside>
</article>
```

- When an <aside> tag is used outside of an <article> tag, its content should be related to the surrounding content.

The audio Element

Audio on the Web

- Before HTML5, there was no standard for playing audio files on a web page.
  The HTML5 <audio> element specifies a standard for embedding audio in a web page.

- There are two different ways to specify the audio source file's URL. The first uses the source attribute:
  <**audio** src="audio.mp3" controls>
  Audio element not supported by your browser
  </**audio**>

- The second way uses the <source> element inside the <audio> element:

```
<audio controls>
    <source src="audio.mp3" type="audio/mpeg">
    <source src="audio.ogg" type="audio/ogg">
</audio>
```

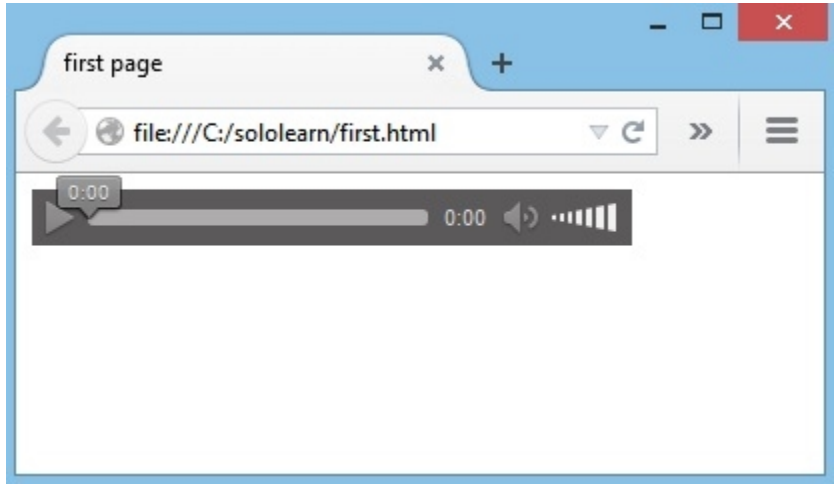
- Multiple <source> elements can be linked to different audio files. The browser will use the first recognized format.

Audio on the Web

- The <audio> element creates an audio player inside the browser.

```
<audio controls>
    <source src="audio.mp3"
type="audio/mpeg">
    <source src="audio.ogg"
type="audio/ogg">
    Audio element not supported by
your browser.
</audio>
```

- Result:



- The text between the <audio> and </audio> tags will display in browsers that do not support the <audio> element.

Attributes of <audio>

- **controls**
  Specifies that audio controls should be displayed (such as a play/pause button, etc.)

- **autoplay**
  When this attribute is defined, audio starts playing as soon as it is ready, without asking for the visitor's permission.
  <audio controls **autoplay**>

- **loop**
  This attribute is used to have the audio replay every time it is finished.
  <audio controls autoplay **loop**>

- Currently, there are three supported file formats for the <audio> element: MP3, WAV, and OGG.

The video Element

Videos in HTML

- The video element is similar to the audio element.
  You can specify the video source URL using an attribute in a video element, or using source elements inside the video element:

```
<video controls>
    <source src="video.mp4" type="video/mp4">
    <source src="video.ogg" type="video/ogg">
    Video is not supported by your browser
</video>
```

- Another aspect that the audio and video elements have in common is that the major browsers do not all support the same file types. If the browser does not support the first video type, it will try the next one.

Attributes of <video>

- Another aspect shared by both the audio and the video elements is that each has **controls**, **autoplay** and **loop** attributes.

- In this example, the video will replay after it finishes playing:

```
<video controls autoplay loop>
    <source src="video.mp4" type="video/mp4">
    <source src="video.ogg" type="video/ogg">
    Video is not supported by your browser
</video>
```

- Currently, there are three supported video formats for the <video> element: MP4, WebM, and OGG.

The progress Element

Progress Bar

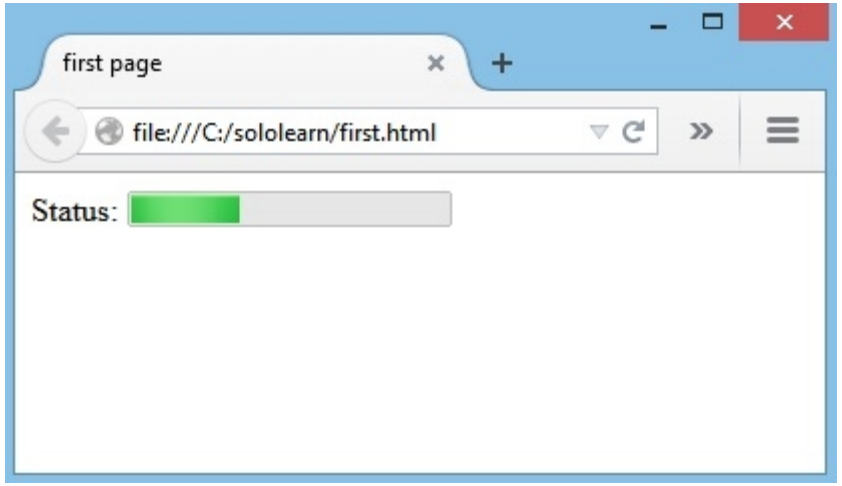
- The **<progress>** element provides the ability to create progress bars on the web.
- The progress element can be used within headings, paragraphs, or anywhere else in the body.

- **Progress Element Attributes**
  Value: Specifies how much of the task has been completed.
  Max: Specifies how much work the task requires in total.

- **Example:**

```
Status: <progress min="0" max="100"
value="35">
</progress>
```

- Result:



- Use the <progress> tag in conjunction with JavaScript to dynamically display a task's progress.

Web Storage API

HTML5 Web Storage

- With HTML5 web storage, websites can store data on a user's local computer.
- Before HTML5, we had to use **JavaScript cookies** to achieve this functionality.

- **The Advantages of Web Storage**
  - More secure
  - Faster
  - Stores a larger amount of data
  - Stored data is not sent with every server request

- Local storage is per domain. All pages from one domain can store and access the same data.

Types of Web Storage Objects

- There are two types of web storage objects:
  - sessionStorage()
  - localStorage()

- **Local vs. Session**
  - Session Storage is destroyed once the user closes the browser
  - Local Storage stores data with no expiration date

- You need to be familiar with basic JavaScript in order to understand and use the API.

Working with Values

- The syntax for web storage for both local and session storage is very simple and similar.
  The data is stored as key/value pairs.

- **Storing a Value:**
  localStorage.**setItem**("key1", "value1");

- **Getting a Value:**
  //this will print the value
  alert(localStorage.**getItem**("key1"));

- **Removing a Value:**
  localStorage.**removeItem**("key1");

- **Removing All Values:**
  localStorage.**clear**();

- The same syntax applies to the session storage, with one difference: Instead of localStorage, sessionStorage is used.

Geolocation API

What is the Geolocation API?

- In HTML5, the Geolocation API is used to obtain the user's geographical location.

- Since this can compromise user privacy, the option is not available unless the user approves it.

- Geolocation is much more accurate for devices with GPS, like smartphones and the like.

Using HTML Geolocation

- The Geolocation API's main method is getCurrentPosition, which retrieves the current geographic location of the user's device.
  navigator.geolocation.getCurrentPosition();

- Parameters:
  showLocation (mandatory): Defines the callback method that retrieves location information.
  ErrorHandler(optional): Defines the callback method that is invoked when an error occurs in processing the asynchronous call.
  Options (optional): Defines a set of options for retrieving the location information.

- You need to be familiar with basic JavaScript in order to understand and use the API.

Presenting Data

- User location can be presented in two ways: **Geodetic** and **Civic**.

  1. The geodetic way to describe position refers directly to latitude and longitude.
  2. The civic representation of location data is presented in a format that is more easily read and understood by the average person.

  Each parameter has both a geodetic and a civic representation:

Attribute	Geodetic	Civic
Position	59.3, 18.6	Stockholm
Elevation	10 meters	4 th floor
Heading	234 degrees	City center
Speed	5km / h	Walking
Orientation	45 degrees	North-East

- The getCurrentPosition() method returns an object if it is successful. The latitude, longitude, and accuracy properties are always returned.

Drag&Drop API

Making Elements Draggable

- The drag and drop feature lets you "grab" an object and drag it to a different location.
  To make an element draggable, just set the **draggable** attribute to true:

- Any HTML element can be draggable.

- The API for HTML5 drag and drop is event-based.

- Example:

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev) {
ev.preventDefault();
}

function drag(ev) {
ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev) {
ev.preventDefault();
var data = ev.dataTransfer.getData("text");
ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<div id="box" ondrop="drop(event)"
ondragover="allowDrop(event)"
style="border:1px solid black;
width:200px; height:200px"></div>

<img id="image" src="sample.jpg" draggable="true"
ondragstart="drag(event)" width="150" height="50" alt="" />

</body>
</html>
```

Making Elements Draggable

- **What to Drag**
  When the element is dragged, the **ondragstart** attribute calls a function, drag(event), which specifies what data is to be dragged.

- The **dataTransfer.setData()** method sets the data type and the value of the dragged data:
  ```
  function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
  }
  ```

- In our example, the data type is "text" and the value is the ID of the draggable element ("image").

Making Elements Draggable

- **Where to Drop**
  The **ondragover** event specifies where the dragged data can be dropped. By default, data and elements cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element.
- This is done by calling the event.**preventDefault**() method for the **ondragover** event.

- **Do the Drop**
  When the dragged data is dropped, a drop event occurs.
- In the example above, the **ondrop** attribute calls a function, drop(event):

```
function drop(ev) {
ev.preventDefault();
var data = ev.dataTransfer.getData("text");
ev.target.appendChild(document.getElementById(data));
}
```

Making Elements Draggable

- The **preventDefault()** method prevents the browser's default handling of the data (default is open as link on drop).
- The dragged data can be accessed with the **dataTransfer.getData()** method. This method will return any data that was set to the same type in the setData() method.
- The dragged data is the ID of the dragged element ("image").

- At the end, the dragged element is appended into the drop element, using the appendChild() function.

- Basic knowledge of JavaScript is required to understand and use the API.

SVG

Drawing Shapes

- **SVG** stands for **S**calable **V**ector **G**raphics, and is used to draw shapes with HTML-style markup.

- It offers several methods for drawing paths, boxes, circles, text, and graphic images.

- SVG is not pixel-based, so it can be magnified infinitely with no loss of quality.

Inserting SVG Images

- An SVG image can be added to HTML code with just a basic image tag that includes a source attribute pointing to the image:

- SVG defines vector-based graphics in XML format.

Drawing a Circle

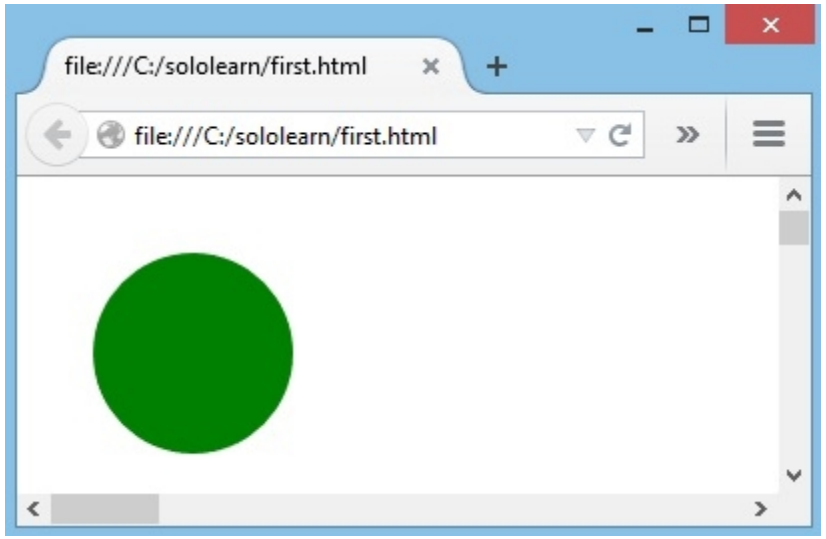
- To draw shapes with SVG, you first need to create an **SVG** element tag with two attributes: width and height.
  <**svg width**="1000" **height**="1000"></**svg**>
- To create a circle, add a **<circle>** tag:

```
<svg width="2000" height="2000">
    <circle cx="80" cy="80" r="50" fill="green" />
</svg>
```

- **cx** pushes the center of the circle further to the right of the screen
- **cy** pushes the center of the circle further down from the top of the screen
- **r** defines the radius
- **fill** determines the color of our circle
- **stroke** adds an outline to the circle

Drawing a Circle

- Result:



- Every element and every attribute in SVG files can be animated.

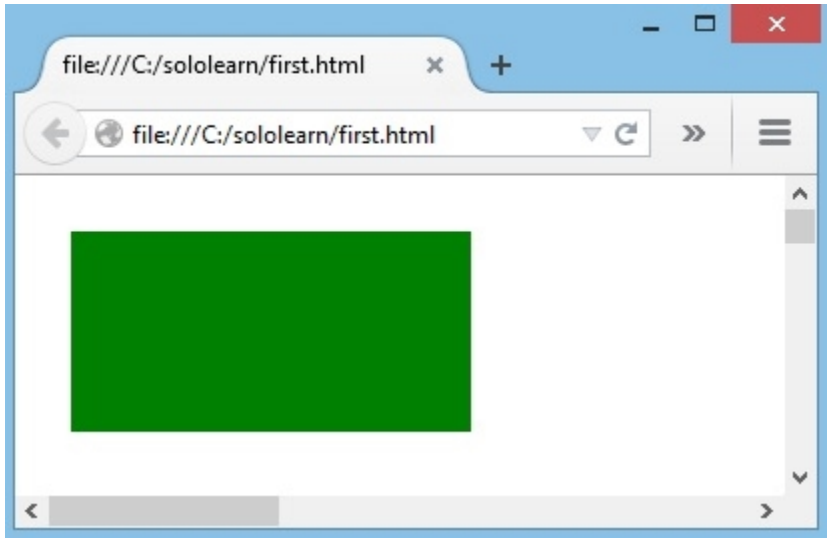
Other Shape Elements

- **<rect>** defines a rectangle:

```
<svg width="2000"
height="2000">
    <rect width="300"
height="100" x="20" y="20"
fill="green" />
</svg>
```

- The following code will draw a green-filled rectangle.

- Result:

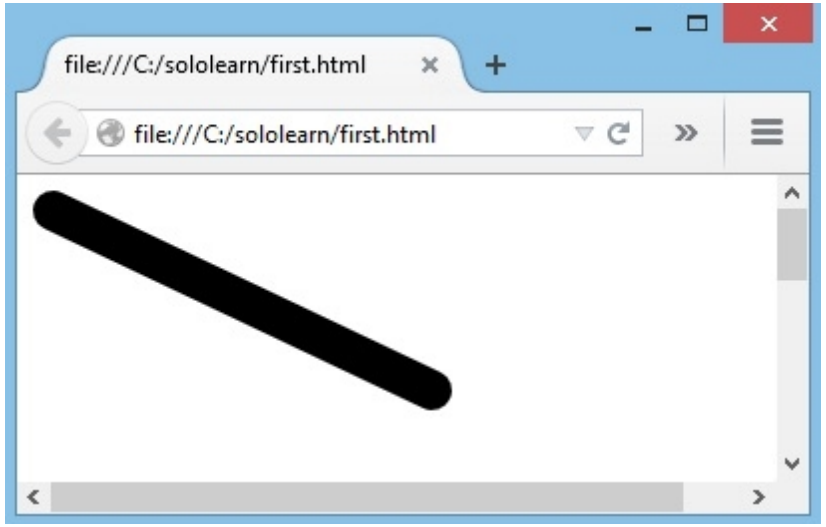


Other Shape Elements

- **<line>** defines a line segment:

```
<svg  width="400"
height="410">
    <line x1="10" y1="10"
x2="200" y2="100"
style="stroke:#000000;  stroke-
linecap:round;
stroke-width:20"  />
</svg>
```

- Result:



- (x1, y1) define the start coordinates(x2, y2) define the end coordinates.

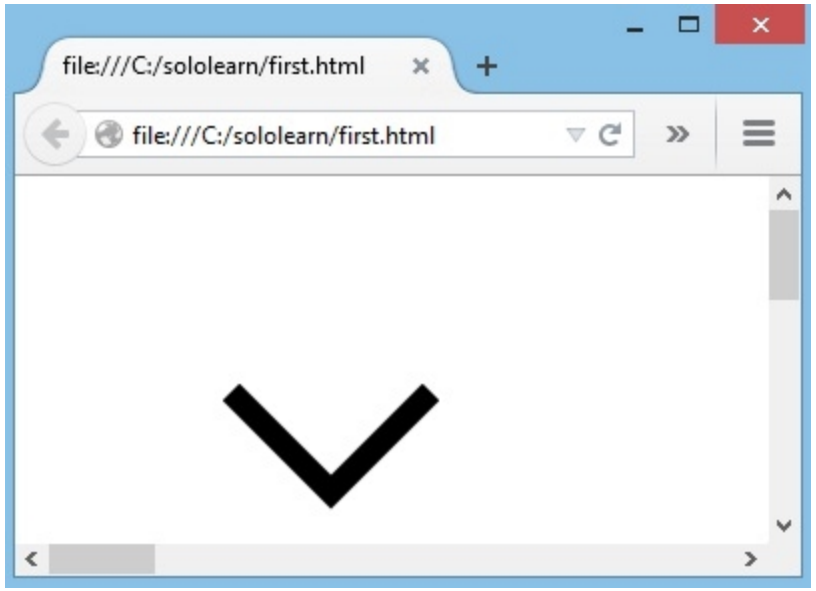
Other Shape Elements

- **<polyline>** defines shapes built from multiple line definitions:

```
<svg width="2000" height="500">
    <polyline style="stroke-
linejoin:miter; stroke:black;
stroke-width:12; fill: none;"
points="100 100, 150 150, 200 100"
/>
</svg>
```

- Points are the polyline's coordinates.
- The code below will draw a black check sign:

- Result:



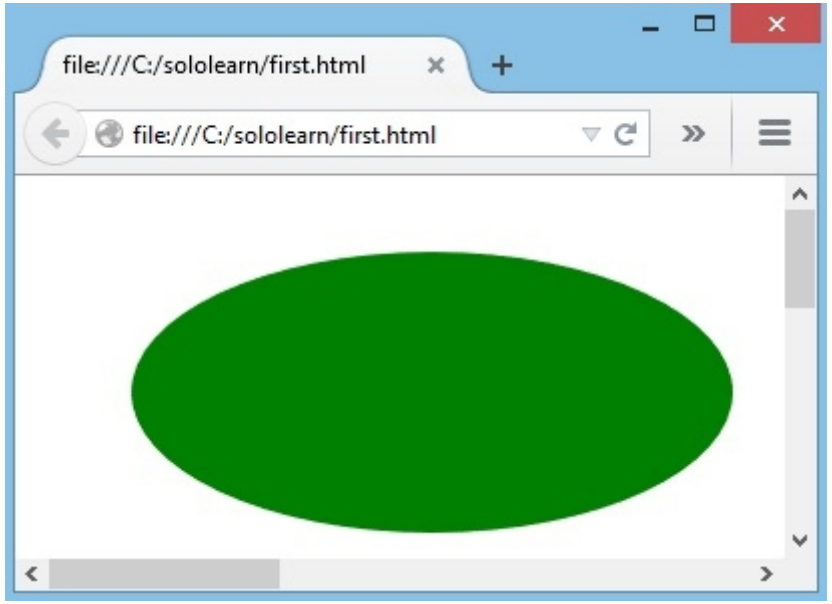
- The width and height attributes of the <rect> element define the height and the width of the rectangle.

<ellipse> and <polygon>

- **Ellipse**
  The <**ellipse**> is similar to the <**circle**>, with one exception:
- You can independently change the horizontal and vertical axes of its radius, using
  the **rx** and **ry** attributes.

```
<svg height="500" width="1000">
    <ellipse cx="200" cy="100"
rx="150" ry="70" style="fill:green"
/>
</svg>
```

- Result:

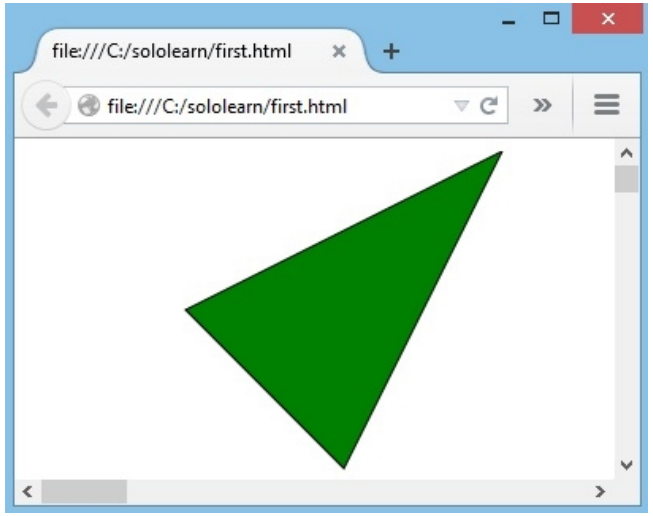


<ellipse> and <polygon>

- **Polygon**
  The **<polygon>** element is used to create a graphic with at least three sides. The polygon element is unique because it automatically closes off the shape for you.

```
<svg width="2000" height="2000">
    <polygon points="100 100, 200 200, 300 0"
style="fill: green; stroke:black;" />
</svg>
```

- Result:



- Polygon comes from Greek. "Poly" means "many" and "gon" means "angle."

HTML5 Forms

HTML5 Forms

- HTML5 brings many features and improvements to web form creation. There are new attributes and input types that were introduced to help create better experiences for web users.

- Form creation is done in HTML5 the same way as it was in HTML4:

```
<form>
    <label>Your name:</label>
    <input id="user" name="username" type="text" />
</form>
```

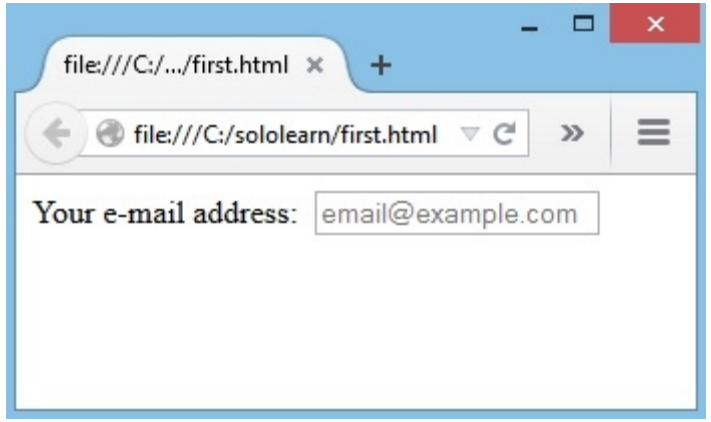
- Use the novalidate attribute to avoid form validation on submissions.

New Attributes

- HTML5 has introduced a new attribute called **placeholder**. On <input> and <textarea> elements, this attribute provides a hint to the user of what information can be entered into the field.

```
<form>
    <label for="email">Your e-mail address: </label>
    <input type="text" name="email" placeholder="email@example.com" />
</form>
```

- Result:

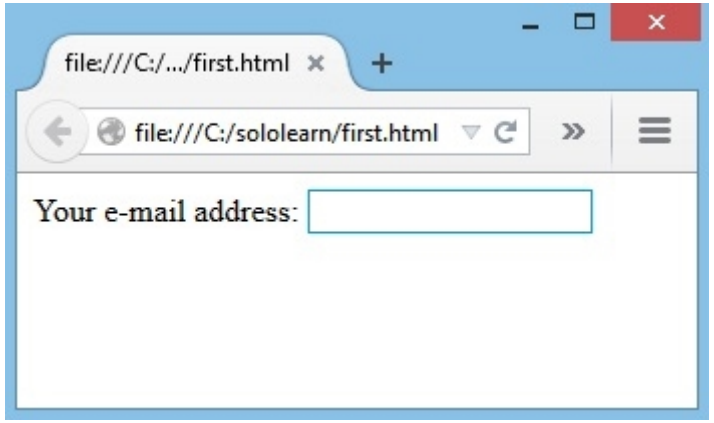


New Attributes

- The **autofocus** attribute makes the desired input focus when the form loads:

```
<form>
    <label for="email">Your e-mail address: </label>
    <input type="text" name="email" autofocus/>
</form>
```

- Result:



- The required attribute tells the browser that the input is required.

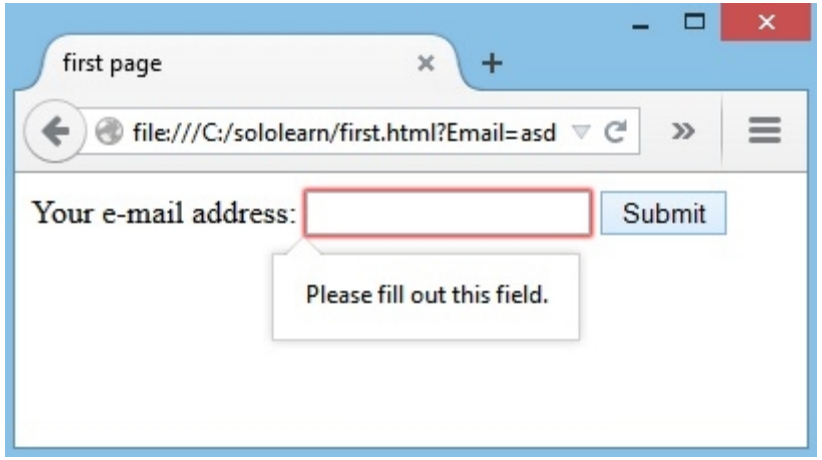
Forms with Required Fields

- The "**required**" attribute is used to make the input elements required.

```
<form autocomplete="off">
    <label for="e-mail">Your e-mail address:
</label>
    <input name="Email" type="text" required
/>
    <input type="submit" value="Submit"/>
</form>
```

- The form will not be submitted without filling in the required fields.

- Result:



- The **autocomplete** attribute specifies whether a form or input field should have autocomplete turned on or off.
- When autocomplete is on, the browser automatically complete values based on values that the user has entered before.

Forms with Required Fields

- **HTML5 added several new input types:**
  - color
  - date
  - datetime
  - datetime-local
  - email
  - month
  - number
  - range
  - search
  - tel
  - time
  - url
  - week

- **New input attributes in HTML5:**
  - autofocus
  - form
  - formaction
  - formenctype
  - formmethod
  - formnovalidate
  - formtarget
  - height and width
  - list
  - min and max
  - multiple
  - pattern (regexp)
  - placeholder
  - required
  - step

- Input types that are not supported by old web browsers, will behave as input type text.

Creating a Search Box

- The new **search** input type can be used to create a search box:

```
<input id="mysearch"
name="searchitem"
type="search" />
```

- Result:

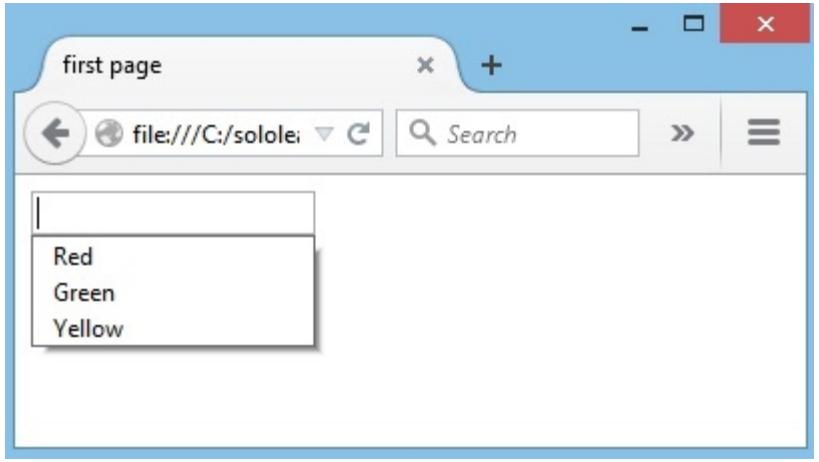
- You must remember to set a name for your input; otherwise, nothing will be submitted.

Search Options

- The **<datalist>** tag can be used to define a list of pre-defined options for the search field:

```
<input id="car" type="text"
list="colors" />
<datalist id="colors">
    <option value="Red">
    <option value="Green">
    <option value="Yellow">
</datalist>
```

- Result:



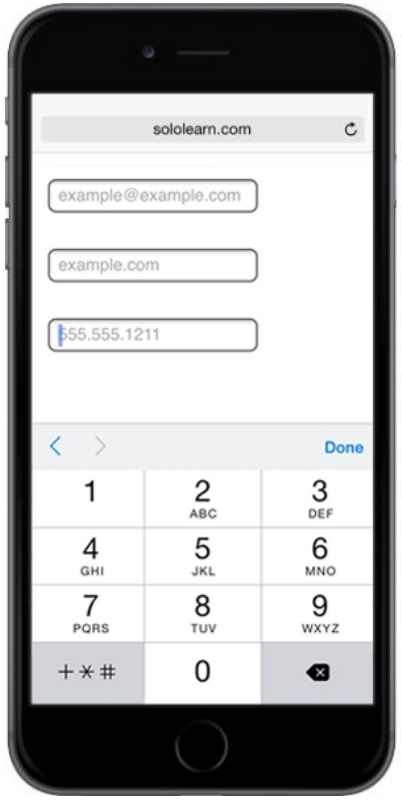
- **<option>** defines the options in a drop-down list for the user to select.
- The ID of the datalist element must match with the list attribute of the input box.

Creating More Fields

- Some other new input types include **email**, **url**, and **tel**:

```
<input id="email" name="email"
type="email"
placeholder="example@example.com" />
<br />
<input id="url" name="url" type="url"
placeholder="example.com" />
<br />
<input id="tel" name="tel" type="tel"
placeholder="555.555.1211" />
```

- These are especially useful when opening a page on a modern mobile device, which recognizes the input types and opens a corresponding keyboard matching the field's type:



- These new types make it easier to structure and validate HTML forms.

What is CSS?

Welcome to CSS!

- CSS stands for **C**ascading **S**tyle **S**heets.

  - **Cascading** refers to the way CSS applies one style on top of another.
  - **Style Sheets** control the look and feel of web documents.

- **CSS** and **HTML** work hand in hand:
  - HTML sorts out the page structure.
  - CSS defines how HTML elements are displayed.
- To understand CSS, you should already have a basic knowledge of HTML.

Why Use CSS?

- CSS allows you to apply specific styles to specific HTML elements.
- The main benefit of CSS is that it allows you to separate **style** from **content**.
- Using just HTML, all the styles and formatting are in the same place, which becomes rather difficult to maintain as the page grows
- All formatting can (and **should**) be removed from the HTML document and stored in a separate CSS file.

Inline, Embedded, External CSS

Inline CSS

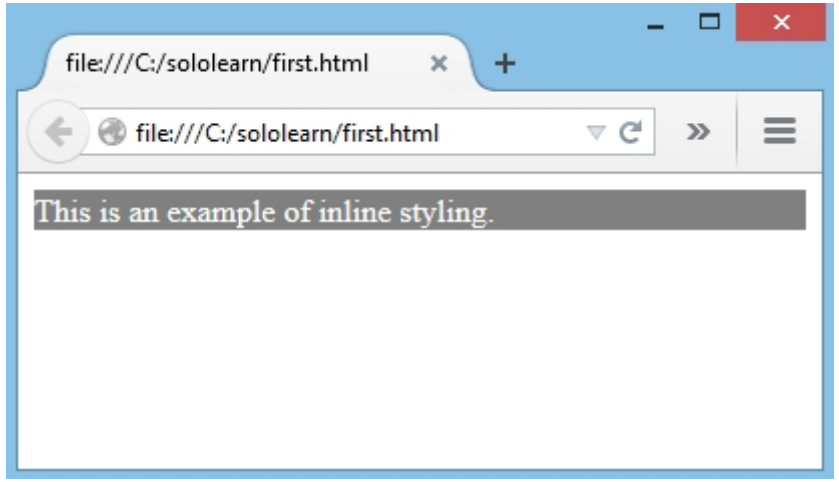
- Using an inline style is one of the ways to insert a style sheet. With an inline style, a unique style is applied to a single element.

- In order to use an inline style, add the **style attribute** to the **relevant tag**.

- The example below shows how to create a paragraph with a gray background and white text:

```
<p style="color:white; background-color:gray;">
    This is an example of inline styling.
</p>
```

- Result:



- The style attribute can contain any CSS property.

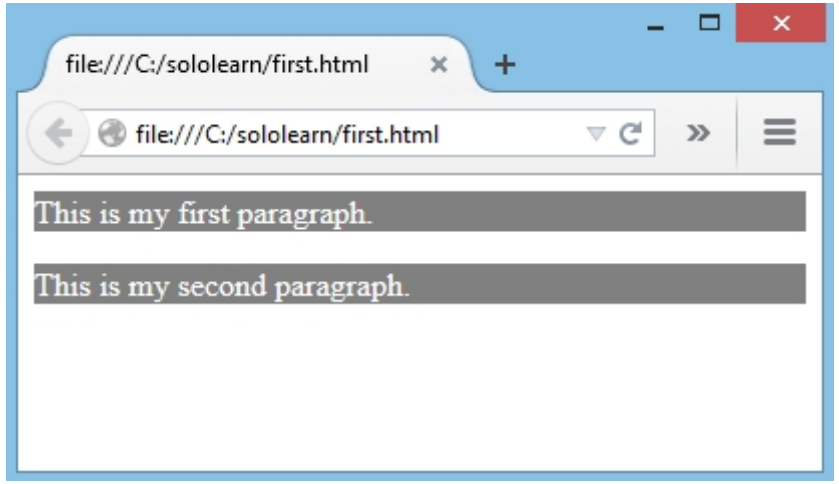
Embedded/Internal CSS

- Internal styles are defined within the **<style>** element, inside the **head** section of an HTML page.

- For example, the following code styles **all** paragraphs:

```
<html>
    <head>
        <style>
            p {
                color:white;
                background-color:gray;
               }
        </style>
    </head>
    <body>
        <p>This is my first paragraph. </p>
        <p>This is my second paragraph. </p>
    </body>
</html>
```

- All paragraphs have a white font and a gray background:



- An internal style sheet may be used if one single page has a unique style.

External CSS

- With this method, all styling rules are contained in a single text file, which is saved with the **.css** extension.

- This CSS file is then referenced in the HTML using the **<link>** tag. The <link> element goes inside the head section.

- Here is an example:
  The CSS:

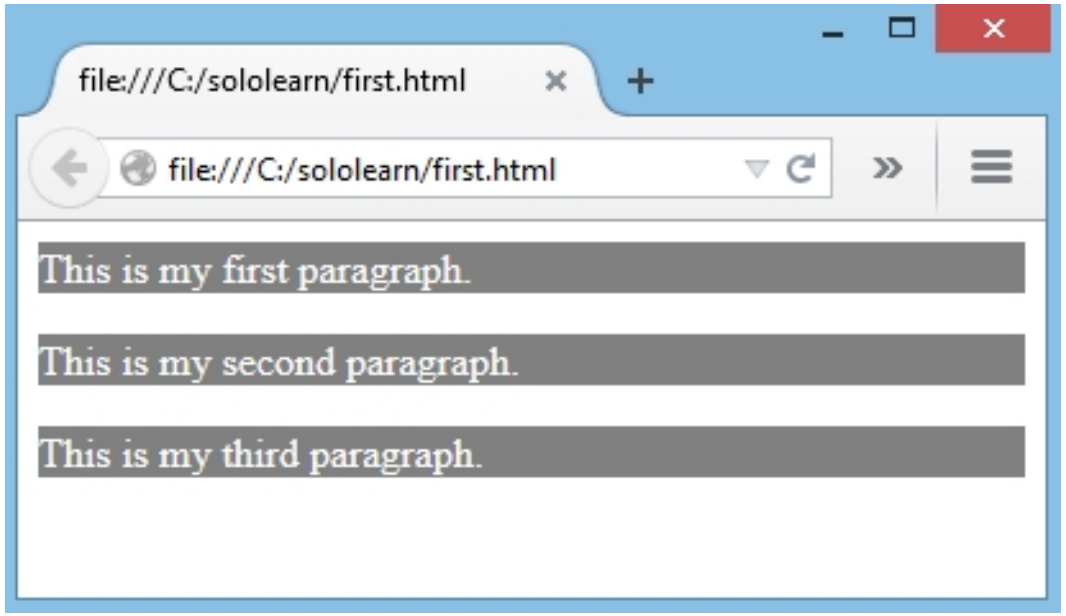
```
p {
    color:white;
    background-color:gray;
}
```

- **The HTML:**

```
<head>
    <link rel="stylesheet"
href="example.css">
</head>
<body>
    <p>This is my first paragraph.</p>
    <p>This is my second paragraph.
</p>
    <p>This is my third paragraph. </p>
</body>
```

External CSS

- Result:



- Both relative and absolute paths can be used to define the **href** for the CSS file. In our example, the path is relative, as the CSS file is in the same directory as the HTML file.

CSS Rules and Selectors

CSS Syntax

- CSS is composed of style rules that the browser interprets and then applies to the corresponding elements in your document.
  A style rule has three parts: **selector**, **property**, and **value**.

- For example, the headline color can be defined as:
  h1 { color: orange; }

- Where:

- The selector points to the HTML element you want to style.
  The declaration block contains one or more declarations, separated by semicolons.
- Each declaration includes a property name and a value, separated by a colon.

Type Selectors

- The most common and easy to understand selectors are **type selectors**. This selector targets element types on the page.

- For example, to target all the paragraphs on the page:

```
p {
    color: red;
    font-size:130%;
}
```

- A CSS declaration always ends with a semicolon, and declaration groups are surrounded by curly braces.

id and class Selectors

- **id selectors** allow you to style an HTML element that has an **id** attribute, regardless of their position in the document tree. Here is an example of an id selector:

- **The HTML:**

```
<div id="intro">
    <p> This paragraph is in the intro section.</p>
</div>
<p> This paragraph is not in the intro section.</p>
```

- **The CSS:**

```
#intro {
    color: white;
    background-color: gray;
}
```

- Result:

id and class Selectors

- To select an element with a specific id, use a hash character, and then follow it with the id of the element.
- **Class selectors** work in a similar way. The major difference is that IDs can only be applied once per page, while classes can be used as many times on a page as needed.
- In the example below, both paragraphs having the class "first" will be affected by the CSS:

- **The HTML:**

```
<div>
    <p class="first">This is a paragraph</p>
    <p> This is the second paragraph. </p>
</div>
<p class="first"> This is not in the intro section</p>
<p> The second paragraph is not in the intro section. </p>
```

- **The CSS:**

```
.first {font-size: 200%;}
```

id and class Selectors

- To select elements with a specific class, use a period character, followed by the name of the class.

- Do **NOT** start a class or id name with a number!

Descendant Selectors

- These selectors are used to select elements that are descendants of another element. When selecting levels, you can select as many levels deep as you need to.

- For example, to target only elements in the first paragraph of the "intro" section:

- **The HTML:**

```
<div id="intro">
    <p class="first">This is a <em>
paragraph.</em></p>
    <p> This is the second paragraph. </p>
</div>
<p class="first"> This is not in the intro
section.</p>
<p> The second paragraph is not in the
intro section. </p>
```

- **The CSS:**

```
#intro .first em {
    color: pink;
    background-color:gray;
}
```

Descendant Selectors

- As a result, only the elements selected will be affected:

- The descendant selector matches all elements that are descendants of a specified element.

CSS Comments

Comments

- Comments are used to explain your code, and may help you when you edit the source code later. Comments are ignored by browsers.

- A CSS comment look like this:
  /* Comment goes here */

- **Example:**

```
p {
    color: green;
    /* This is a comment */
    font-size: 150%;
}
```

- The comment does not appear in the browser:

- Comments can also span multiple lines.

Style Cascade and Inheritance

Cascade

- The final appearance of a web page is a result of different styling rules.

- The three main sources of style information that form a cascade are:
  - The stylesheet created by the **author of the page**
  - The **browser's default styles**
  - Styles specified **by the user**

- CSS is an acronym for Cascading Style Sheets.

Inheritance

- Inheritance refers to the way properties flow through the page. A child element will usually take on the characteristics of the parent element unless otherwise defined.

- **For example:**

```
<html>
    <head>
        <style>
            body {
                color: green;
                font-family: Arial;
            }
        </style>
    </head>
    <body>
        <p>
            Text inside the paragraph.
        </p>
    </body>
</html>
```

Inheritance

- Result:

- Since the paragraph tag (child element) is inside the body tag (parent element), it takes on any styles assigned to the body tag.

Text font-family

The font-family Property

- The font-family property specifies the font for an element.
  There are two types of font family names:
  - **font family**: a specific font family (like Times New Roman or Arial)
  - **generic family**: a group of font families with a similar look (like Serif or Monospace)

Generic family	Font family
Serif	Times New Roman Georgia
Sans - serif	Arial Verdana
Monospace	Courier New Lucida Console

- Here is an example of different font styles:

The font-family Property

- **The HTML:**

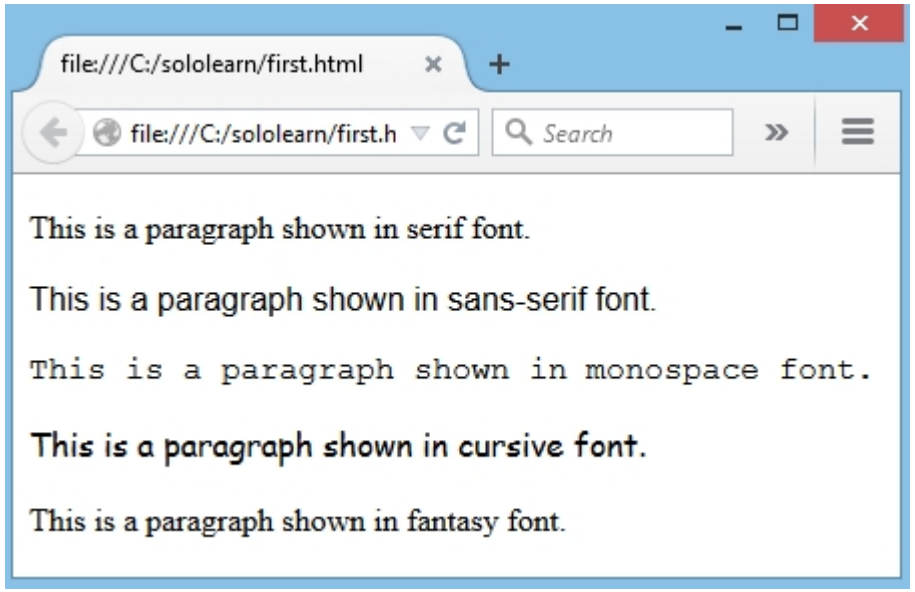
```
<p class="serif">
    This is a paragraph shown in serif font.
</p>
<p class="sansserif">
    This is a paragraph shown in sans-serif
font.
</p>
<p class="monospace">
    This is a paragraph shown in
monospace font.
</p>
<p class="cursive">
    This is a paragraph shown in cursive
font.
</p>
<p class="fantasy">
    This is a paragraph shown in fantasy
font.
</p>
```

- **The CSS:**

```
p.serif {
    font-family: "Times New Roman",
Times, serif;
}
p.sansserif {
    font-family: Helvetica, Arial, sans-serif;
}
p.monospace {
    font-family: "Courier New", Courier,
monospace;
}
p.cursive {
    font-family: Florence, cursive;
}
p.fantasy {
    font-family: Blippo, fantasy;
}
```

The font-family Property

- Result:



- Separate each value with a **comma** to indicate that they are alternatives.
- If the name of a font family is more than one word, it must be in quotation marks: **"Times New Roman"**.

The font-family Property

- The font-family property should hold several font names as a "fallback" system. When specifying a web font in a CSS style, add more than one font name, in order to avoid unexpected behaviors. If the client computer for some reason doesn't have the one you choose, it will try the next one.

- It is a good practice to specify a generic font family, to let the browser pick a similar font in the generic family, if no other fonts are available.

- body {
  font-family: Arial, "Helvetica Neue", Helvetica, sans-serif;
  }

- If the browser does not support the font **Arial**, it tries the next fonts (**Helvetica Neue**, then **Helvetica**). If the browser doesn't have any of them, it will try the generic **sans-serif**.

- Remember to use quotation marks if the font name consists of more than one word.

Arabic Text font-family

Text font-size

The font-size Property

- The font-size property sets the size of a font. One way to set the size of fonts on the web is to use **keywords.** For example **xx-small**, **small**, **medium**, **large**, **larger**, etc.

The font-size Property

- **The HTML:**

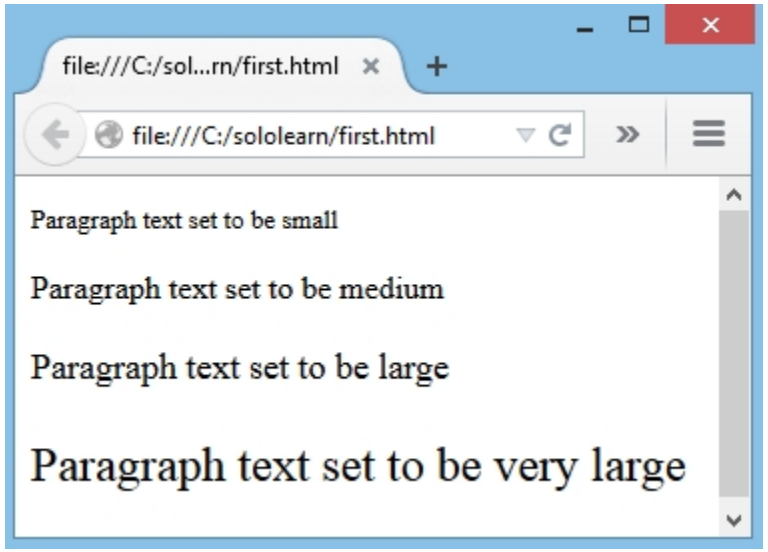
```
<p class="small">
    Paragraph text set to be small
</p>
<p class="medium">
    Paragraph text set to be medium
</p>
<p class="large">
    Paragraph text set to be large
</p>
<p class="xlarge">
    Paragraph text set to be very
large
</p>
```

- **The CSS:**

```
p.small {
    font-size: small;
}
p.medium {
    font-size: medium;
}
p.large {
    font-size: large;
}
p.xlarge {
    font-size: x-large;
}
```

The font-size Property

- Result:



- Keywords are useful if you do not want the user to be able to increase the size of the font because it will adversely affect your site's appearance.

The font-size Property

- You can also use numerical values in **pixels** or **ems** to manipulate font size.

- Setting the font size in pixel values (**px**) is a good choice when you need pixel accuracy, and it gives you full control over the text size.

- The **em** size unit is another way to set the font size (**em** is a relative size unit). It allows all major browsers to resize the text. If you haven't set the font size anywhere on the page, then it is the browser default size, which is **16px**.

- To calculate the em size, just use the following formula: **em = pixels / 16**

The font-size Property

- **For example:**

```
h1 {
        font-size: 20px;
}

h1 {
        font-size: 1.25em;
}
```

- Both of the examples will produce the same result in the browser,
because **20/16=1.25em**.

- Try different combinations of text size and page zooming in a variety of browsers to ensure that the text remains readable.

Text font-style

The font-style Property

- he font-style property is typically used to specify italic text.

- **The HTML:**
```
<p class="italic">This is a paragraph in italic style.</p>
```

- **The CSS:**
```
p.italic {
    font-style: italic;
}
```

- Result:

The font-style Property

- The font-style property has three values: **normal**, **italic**, and **oblique**. Oblique is very similar to italic, but less supported.

- **The HTML:**

```
<p class="normal">This paragraph is normal.</p>
<p class="italic">This paragraph is italic.</p>
<p class="oblique">This paragraph is oblique.</p>
```

- **The CSS:**

```
p.normal {
    font-style: normal;
}
p.italic {
    font-style: italic;
}
p.oblique {
    font-style: oblique;
}
```

- The HTML **<i>** tag will produce exactly the same result as the **italic font style**.

Text font-weight

The font-weight Property

- The font-weight controls the boldness or thickness of the text. The values can be set as **normal** (default size), **bold**, **bolder**, and **lighter**.

- **The HTML:**

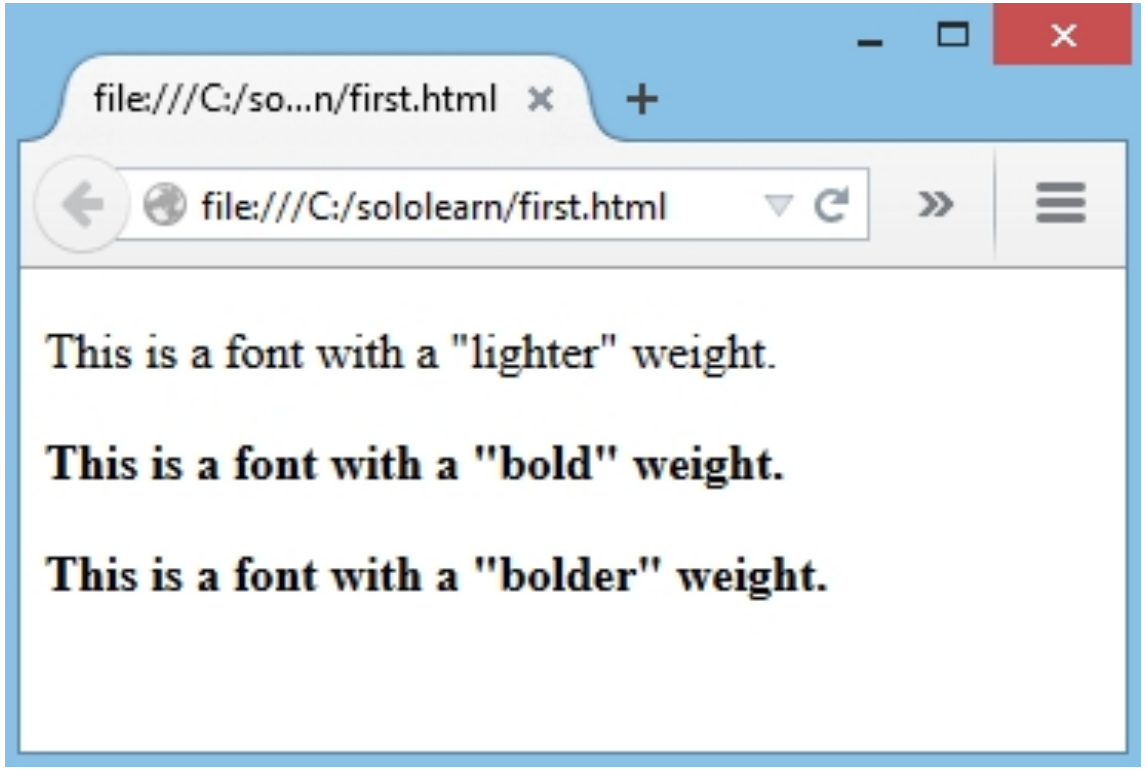
```
<p class="light">This is a font with a "lighter" weight.</p>
<p class="bold">This is a font with a "bold" weight.</p>
<p class="bolder">This is a font with a "bolder" weight.</p>
```

- **The CSS:**

```
p.light {
    font-weight: lighter;
}
p.bold {
    font-weight: bold;
}
p.bolder {
    font-weight: bolder;
}
```

The font-weight Property

- Result:



The font-weight Property

- You can also define the font weight with a number from **100** (thin) to **900** (thick), according to how thick you want the text to be. 400 is the same as normal, and 700 is the same as bold.

- **The HTML:**

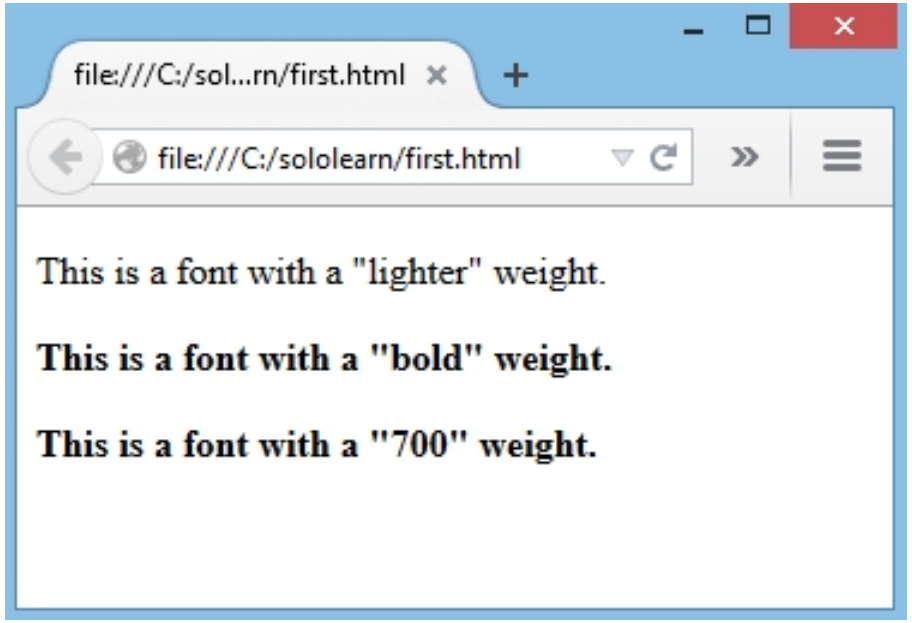
```
<p class="light">This is a font with a "lighter" weight.</p>
<p class="thick">This is a font with a "bold" weight.</p>
<p class="thicker">This is a font with a "700" weight.</p>
```

- **The CSS:**

```
p.light {
    font-weight: lighter;
}
p.thick {
    font-weight: bold;
}
p.thicker {
    font-weight: 700;
}
```

The font-weight Property

- Result:



- The HTML **** tag also makes the text **bold**.

Text color

The color Property

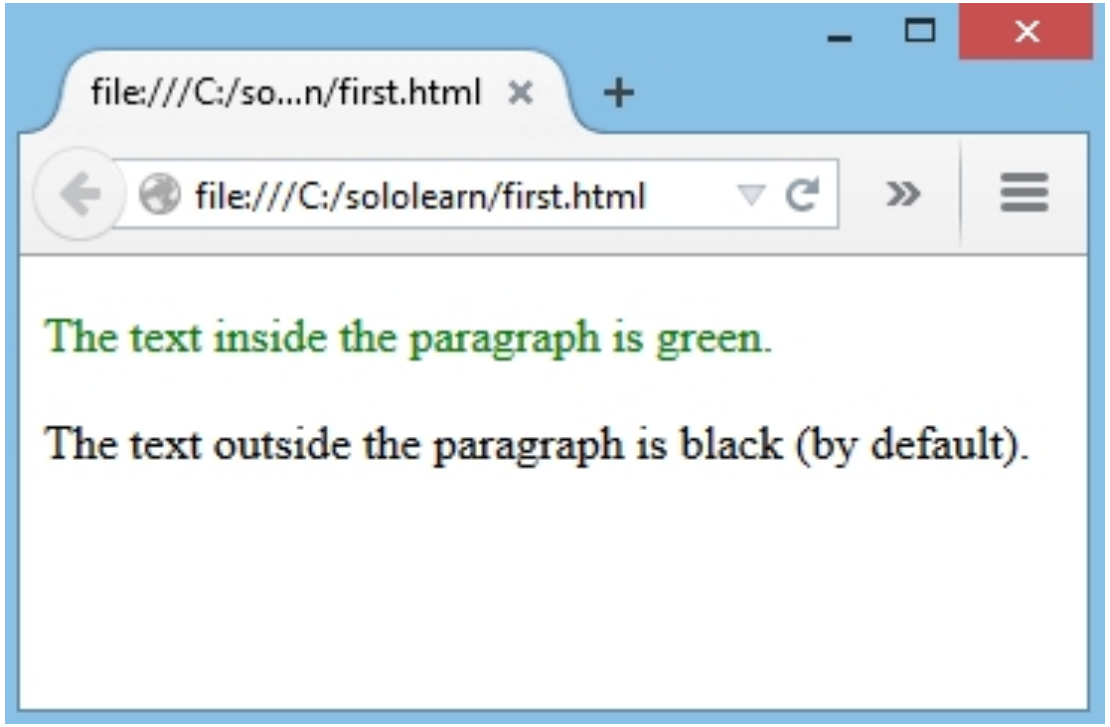
- The CSS **color** property specifies the color of the text.
  One method of specifying the color of the text is using a **color name**: like red, green, blue, etc.
  Here's an example of changing the color of your font.

- **The HTML:**
  ```
  <p class="example">The text inside the paragraph is green.</p>
  The text outside the paragraph is black (by default).
  ```

- **The CSS:**
  ```
  p.example {
      color: green;
  }
  ```

The color Property

- Result:



The color Property

- Another way of defining colors is using **hexadecimal values** and **RGB**.
- Hexadecimal form is a pound sign (**#**) followed by at most, **6 hex values** (0-F).
- RGB defines the individual values for **Red**, **Green**, and **Blue**.

- In the example below, we use hexadecimal value to set the heading color to blue, and RGB form to make the paragraph red.

- **The HTML:**

```
<h1>This is a heading</h1>
<p class="example">This is a paragraph</p>
```

- **The CSS:**

```
h1 {
    color: #0000FF;
}
p.example {
    color: rgb(255,0,0);
}
```

The color Property

- Result:

Aligning Text Horizontally

The text-align Property

- The text-align property specifies the horizontal alignment of text in an element. By default, text on your website is aligned to the left. However, at times you may require a different alignment.

- text-align property values are as follows: **left**, **right**, **center**, and **justify**.

- **The HTML:**

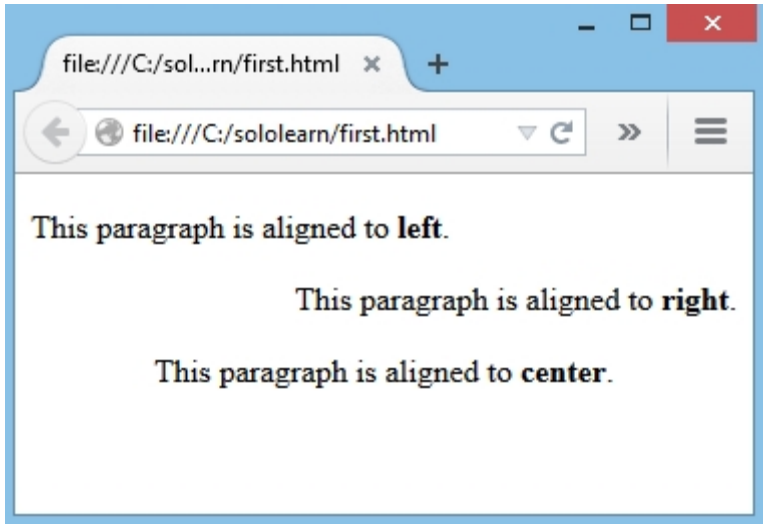
```
<p class="left">This paragraph is aligned to <strong>left.</strong></p>
<p class="right">This paragraph is aligned to <strong>right.</strong></p>
<p class="center">This paragraph is aligned to <strong>center.</strong></p>
```

The text-align Property

- **The CSS:**

```
p.left {
    text-align: left;
}
p.right {
    text-align: right;
}
p.center {
    text-align: center;
}
```

- Result:



- When text-align is set to "**justify**", each line is stretched so that every line has equal width, and the left and right margins are straight (as in magazines and newspapers).

Aligning Text Vertically

The vertical-align Property

- The vertical-align property sets an element's vertical alignment. Commonly used values are **top**, **middle**, and **bottom**.

- The example below shows how to vertically align the text between the table.

- **The HTML:**

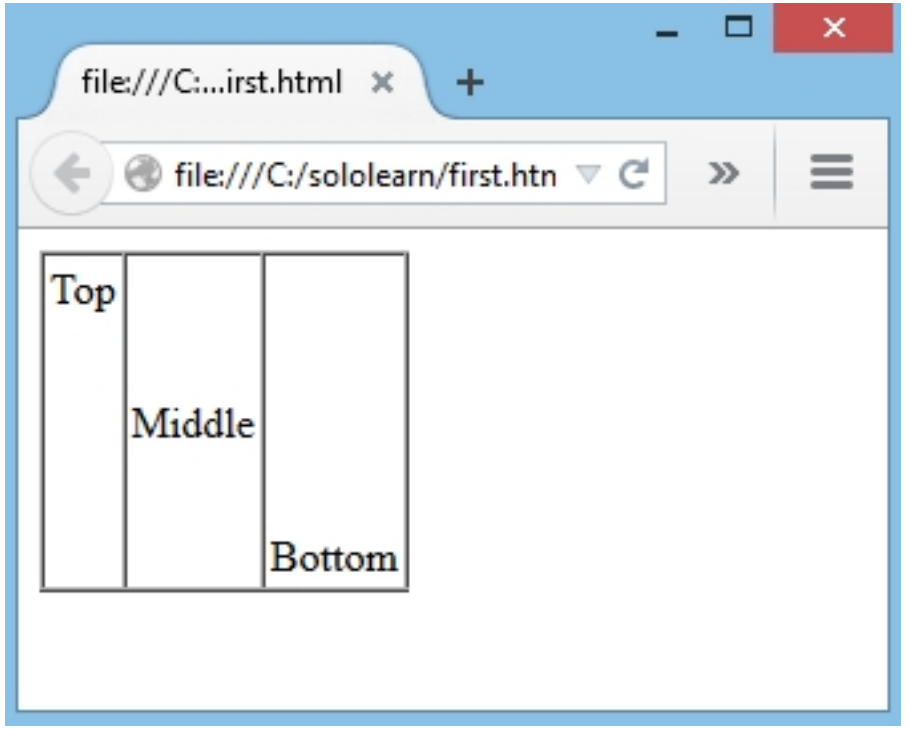
```
<table border="1" cellpadding="2" cellspacing="0" style="height: 150px;">
    <tr>
        <td class="top">Top</td>
        <td class="middle">Middle</td>
        <td class="bottom">Bottom</td>
    </tr>
</table>
```

The vertical-align Property

- **The CSS:**

```
td.top {
    vertical-align: top;
}
td.middle {
    vertical-align: middle;
}
td.bottom {
    vertical-align: bottom;
}
```

- Result:



The vertical-align Property

- The vertical-align property also takes the following values: **baseline**, **sub**, **super**, **%** and **px** (or pt, cm).
- The example below shows the difference between them.

- **The HTML:**

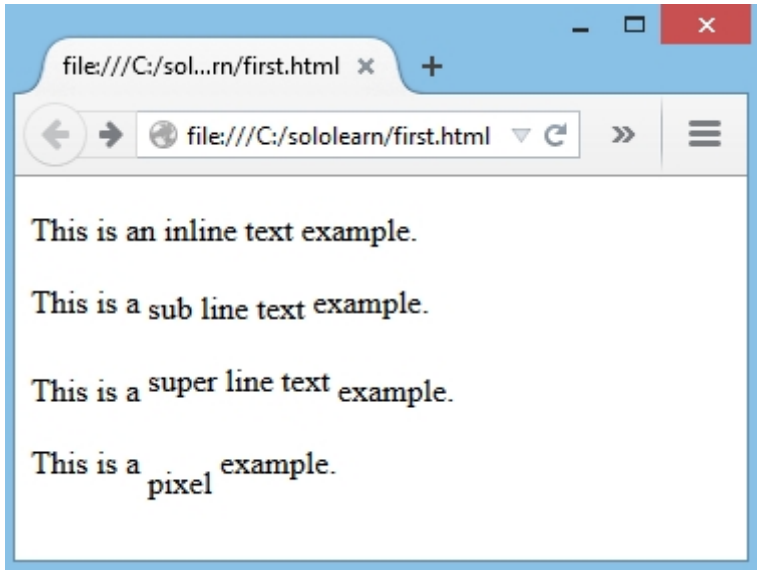
```
<p>This is an <span class="baseline">inline text</span> example.</p>
<p>This is a <span class="sub">sub line text</span> example.</p>
<p> This is a <span class="super">super line text</span>
example.</p>
<p> This is a <span class="pixel">pixel</span> example.</p>
```

The vertical-align Property

- **The CSS:**

```
span.baseline {
    vertical-align: baseline;
}
span.sub {
    vertical-align: sub;
}
span.super {
    vertical-align: super;
}
span.pixel {
    vertical-align: -10px;
}
```

- Result:



- Instead of **px** values, you can use **pt** (points), **cm** (centimeters) and **%** (percentage) values.

The vertical-align Property

- Vertical align property does not act the same way for all elements.
- For example, some additional CSS styling is needed for div elements.

- **The HTML:**

```
<div class="main">
    <div class="paragraph">
        This text is aligned to the middle
    </div>
</div>
```

The vertical-align Property

- **The CSS:**

```
.main {
height: 150px; width: 400px;
background-color: LightSkyBlue;
display: inline-table;
}
.paragraph {
display: table-cell;
vertical-align: middle;
}
```

- Result:

- **display: inline-table;** and **display: table-cell;** styling rules are applied to make the vertical-align property work with divs.

text-decoration

The text-decoration Property

- The text-decoration property specifies how the text will be decorated.

- Commonly used values are:
  none - The default value, this defines a normal text
  inherit - Inherits this property from its parent element
  overline - Draws a horizontal line above the text
  underline - Draws a horizontal line below the text
  line-through - draws a horizontal line through the text (substitutes the HTML **<s>** tag)

- The example below demonstrates the difference between each value.

The text-decoration Property

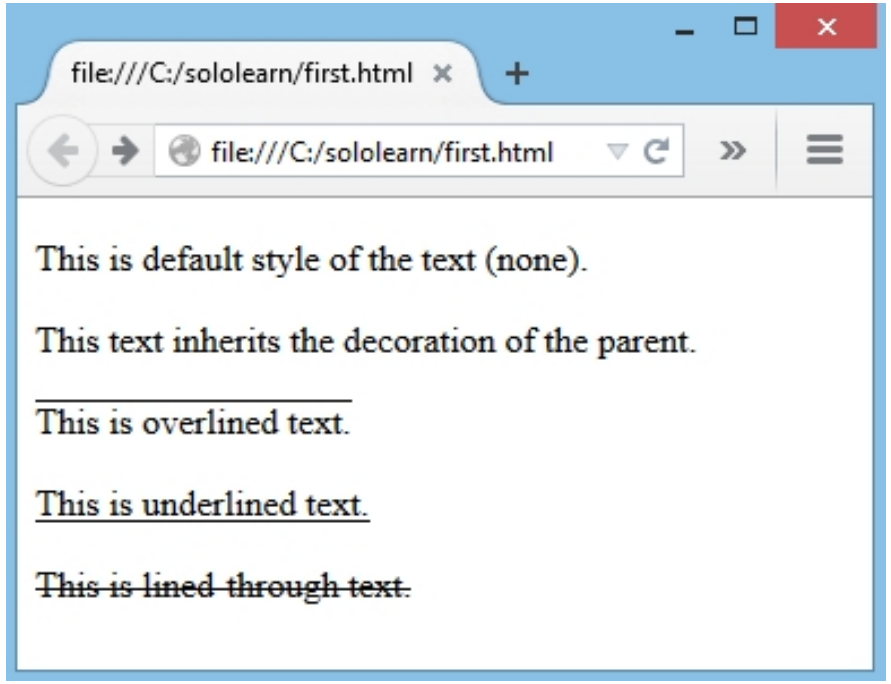
- **The HTML:**

```
<p class="none">This is default
style of the text (none).</p>
<p class="inherit">This text inherits
the decoration of the parent.</p>
<p class="overline">This is
overlined text.</p>
<p class="underline">This is
underlined text.</p>
<p class="line-through">This is
lined-through text.</p>
```

- **The CSS:**

```
p.none {
    text-decoration: none;
}
p.inherit {
    text-decoration: inherit;
}
p.overline {
    text-decoration: overline;
}
p.underline {
    text-decoration: underline;
}
p.line-through {
    text-decoration: line-through;
}
```

The text-decoration Property



- You can combine the **underline**, **overline**, or **line-through** values in a space-separated list to add multiple decoration lines.

The text-decoration Property

- Another value of text-decoration property is **blink** which makes the text blink.

- CSS syntax looks like this:text-decoration: **blink;**

- This value is valid but is deprecated and most browsers ignore it.

text-transform

The text-transform Property

- The text-transform CSS property specifies how to capitalize an element's text. For example, it can be used to make text appear with each word capitalized.

- **The HTML:**

```
<p class="capitalize">
    The value capitalize transforms the first
    character in each word to uppercase;
    all other characters remain unaffected.
</p>
```

The text-transform Property

- **The CSS:**

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

- Result:

text-transform Values

- Using text-transform property you can make text appear in all-uppercase or all-lowercase. Here is an example:

- **The HTML:**

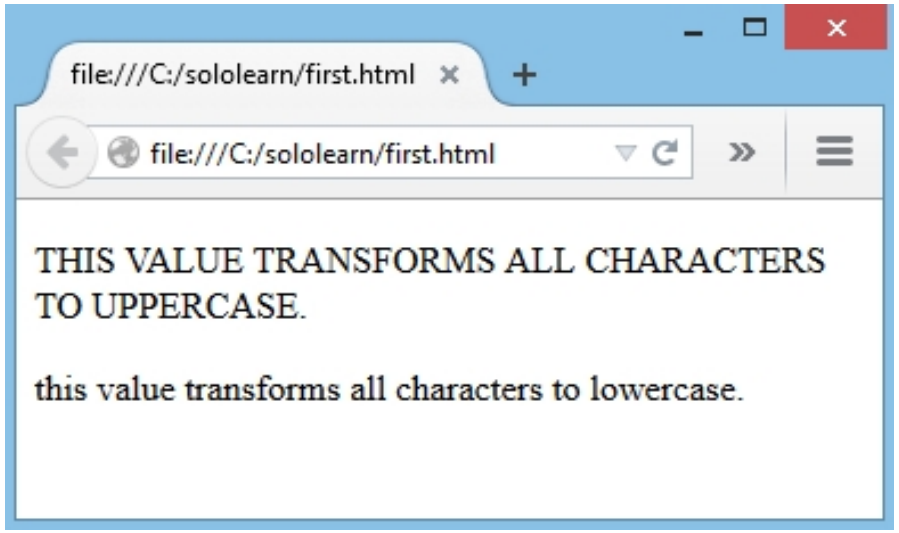
```
<p class="uppercase">This value transforms all characters to uppercase.</p>
<p class="lowercase">This value transforms all characters to lowercase.</p>
```

text-transform Values

- **The CSS:**

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

- Result:



- The value **none** will produce no capitalization effect at all.

letter-spacing

The letter-spacing Property

- The letter-spacing property specifies the **space between characters** in a text. The values can be set as:
  - **normal** defines the default style with no extra space between characters
  - **length** defines an extra space between characters using measurement units like px, pt, cm, mm, etc.;
  - **inherit** inherits the property from its parent element;

- **The HTML:**

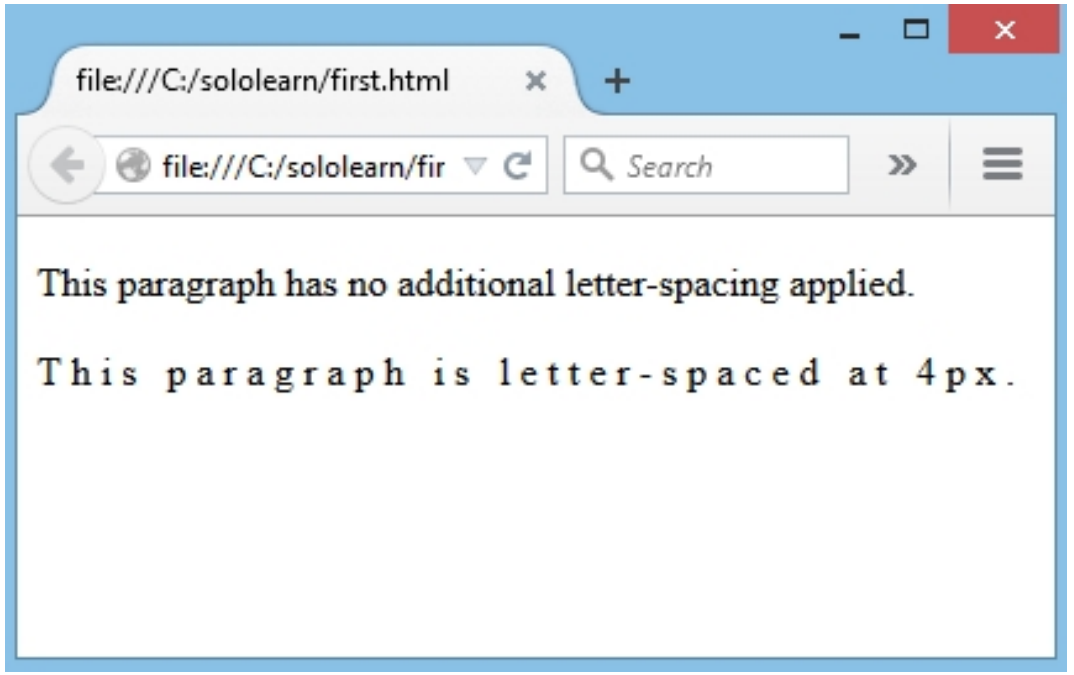
```
<p class="normal">This paragraph has no additional letter-spacing applied.</p>
<p class="positive ">This paragraph is letter-spaced at 4px.</p>
```

The letter-spacing Property

- **The CSS:**

```
p.normal {
    letter-spacing: normal;
}
p.positive {
    letter-spacing: 4px;
}
```

- Result:



Using Negative Values

- For defining an extra space between characters, negative values are also permitted.
  Here is an example demonstrating the difference between **positive** and **negative** values:

- **The HTML:**

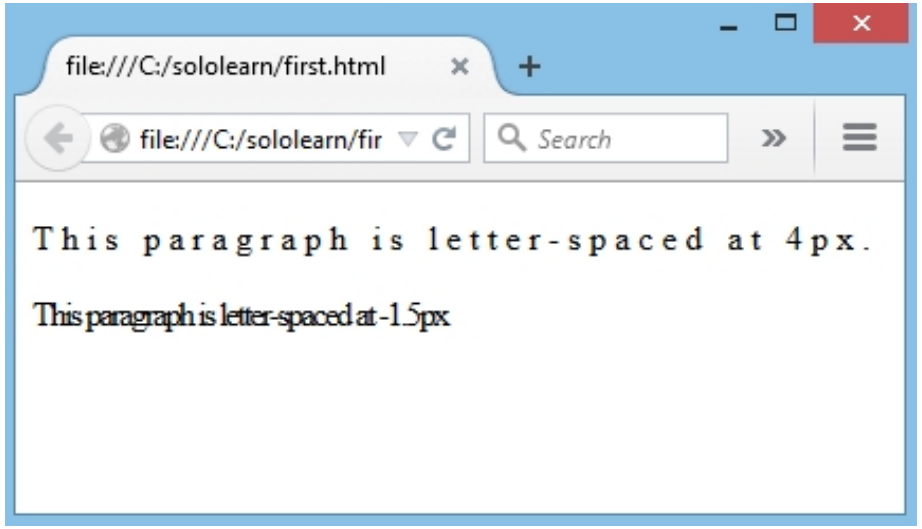
```
<p class="positive">This paragraph is letter-spaced at 4px.</p>
<p class="negative">This paragraph is letter-spaced at -1.5px</p>
```

Using Negative Values

- **The CSS:**

```
p.positive {
    letter-spacing: 4px;
}
p.negative {
    letter-spacing: -1.5px;
}
```

- Result:



- Always test your result, to ensure the text is readable.

Width and Height

CSS Width and Height

- To style a ** ** element to have a total width and height of **100px**:

- **The HTML:**

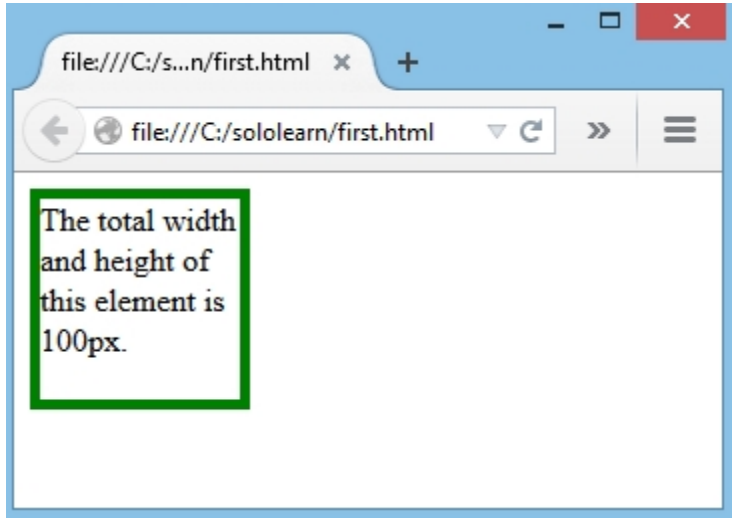
```
<div>The total width and height
of this element is 100px.</div>
```

- **The CSS:**

```
div {
    border: 5px solid green;
    width: 90px;
     height: 90px;
}
```

CSS Width and Height

- Result:



- The total width and height of the box will be the 90px+5px (border)+5px(border) = 100px;

Width and Height Measurement

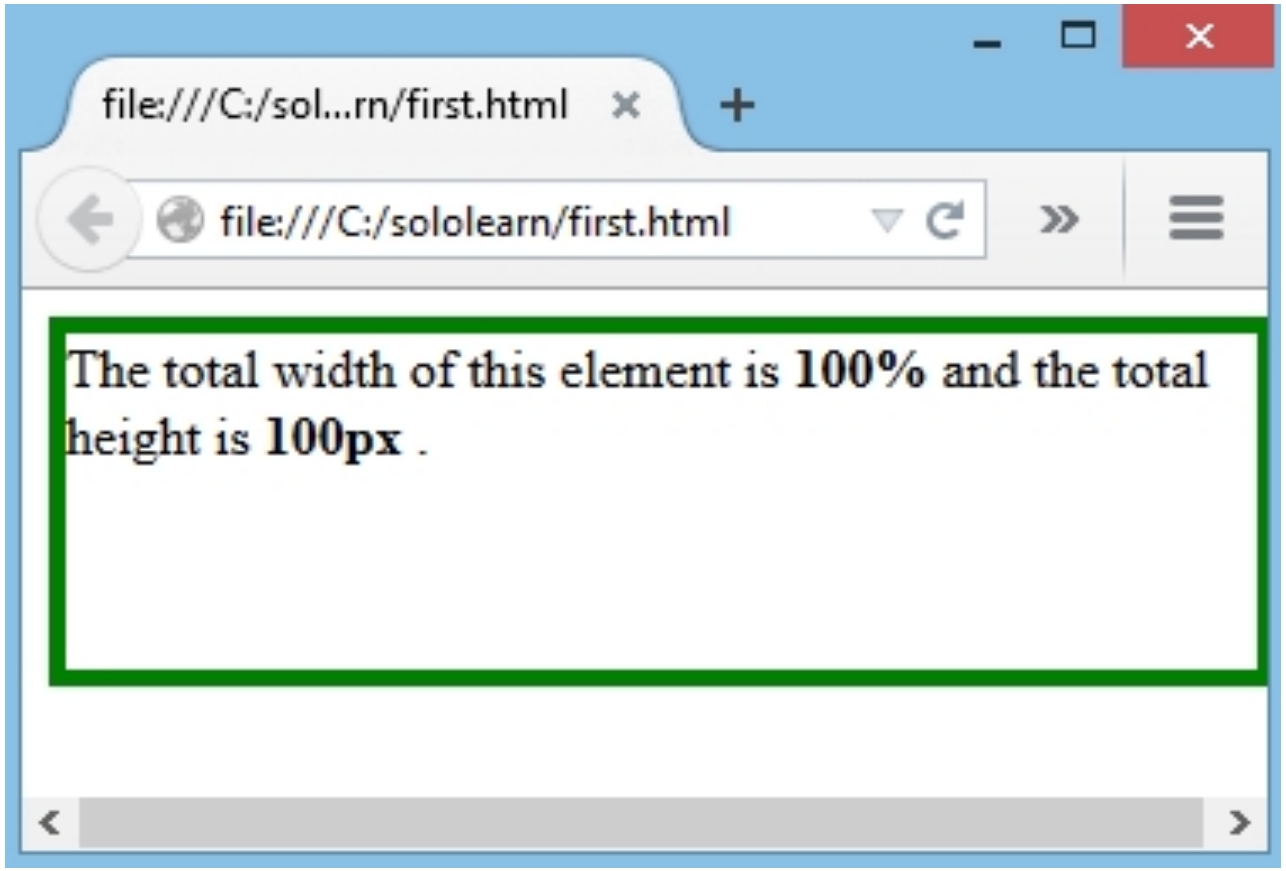
- The width and height of an element can be also assigned using **percents.**
  In the example below the width of an element is assigned in percentages, the height is in pixels.

- **The HTML:**
```
<div>The total width of this element is <strong>100%</strong> and the total height is <strong>100px</strong> .</div>
```

- **The CSS:**
```
div {
    border: 5px solid green;
    width: 100%;
    height: 90px;
}
```

Width and Height Measurement

- Result:



The Minimum and Maximum Sizes

- To set the minimum and maximum height and width of an element, you can use the following properties:

  min-width - the minimum width of an element
  min-height - the minimum height of an element
  max-width - the maximum width of an element
  max-height - the maximum height of an element

- In the example below, we set the minimum height and maximum width of different paragraphs to 100px.

- **The HTML:**

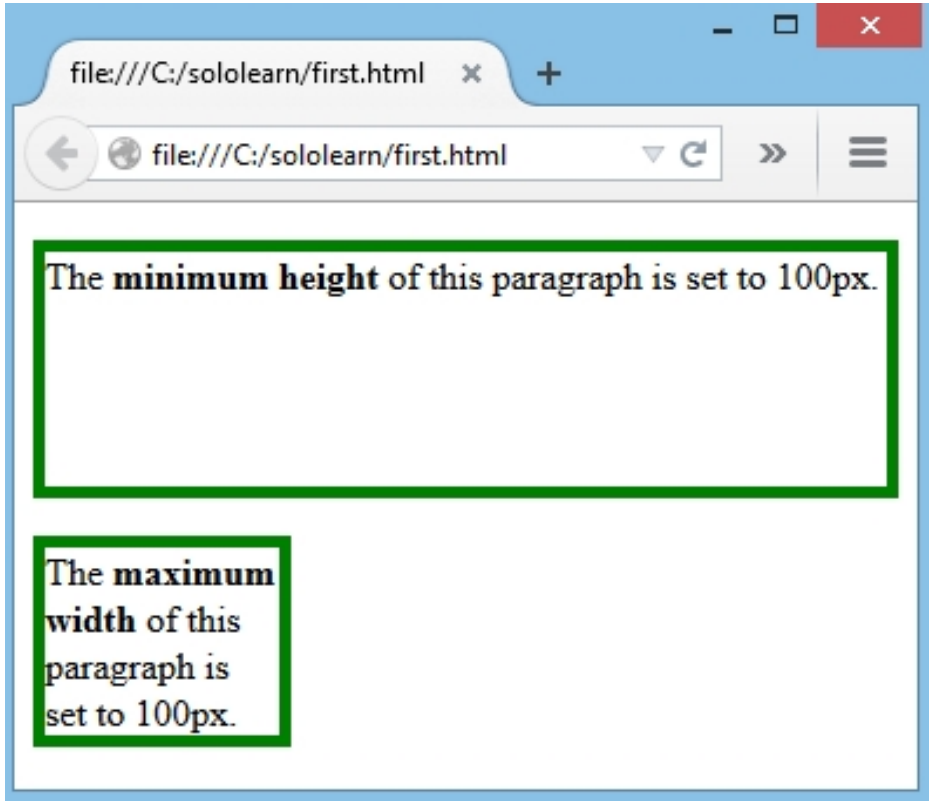
```
<p class="first">The <strong>minimum height </strong> of this paragraph is set to 100px.</p>
<p class="second">The<strong> maximum width </strong> of this paragraph is set to 100px.</p>
```

The Minimum and Maximum Sizes

- **The CSS:**

```
p.first {
    border: 5px solid green;
    min-height: 100px;
}
p.second {
    border: 5px solid green;
    max-width: 100px;
}
```

- Result:



background-color

The background-color Property

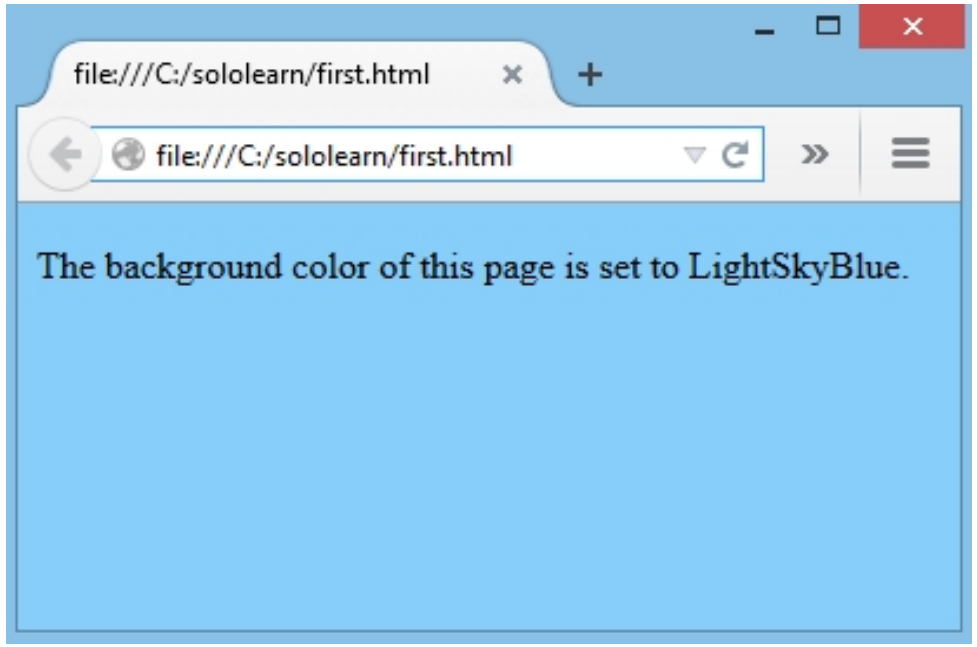
- The **background-color** property is used to specify the background color of an element.
- **The HTML:**

```
<p>The background color of this page is set to LightSkyBlue.</p>
```

- **The CSS:**

```
body {
    background-color: #87CEFA;
}
```

- Result:



The Background Color Values

- The color of the background can be defined using three different formats: a **color name**, **hexadecimal values**, and **RGB**.

- In the example below, the body, h1, and p elements are assigned different background colors:

- **The HTML:**
  <h1>This is a heading</h1>
  This is a paragraph

- **The CSS:**

```
body {
    background-color: #C0C0C0;
}
h1 {
    background-color:
    rgb(135,206,235);
}
p {
    background-color: LightGreen;
}
```

The Background Color Values

- Result:

background-image

The background-image Property

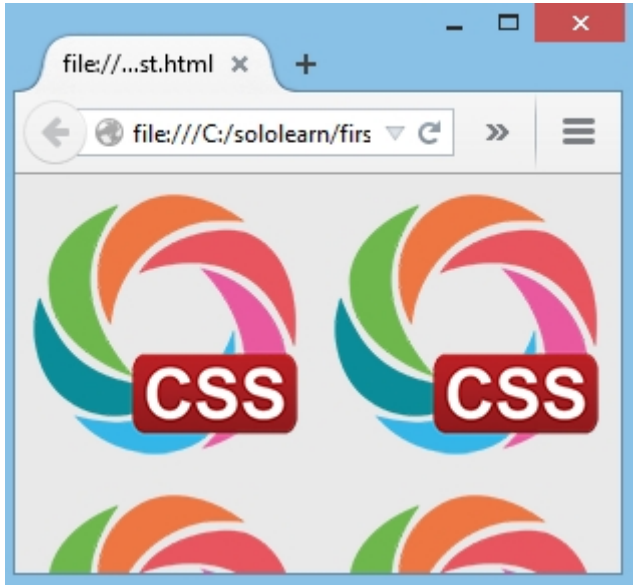
- The **background-image** property sets one or several background images in an element. Let's set a background-image for the <body> element.

- **The CSS:**

```
body {
    background-image:
    url("css_logo.png");
    background-color: #e9e9e9;
}
```

- The **url** specifies the path to the image file. Both relative and absolute paths are supported.

- Result:



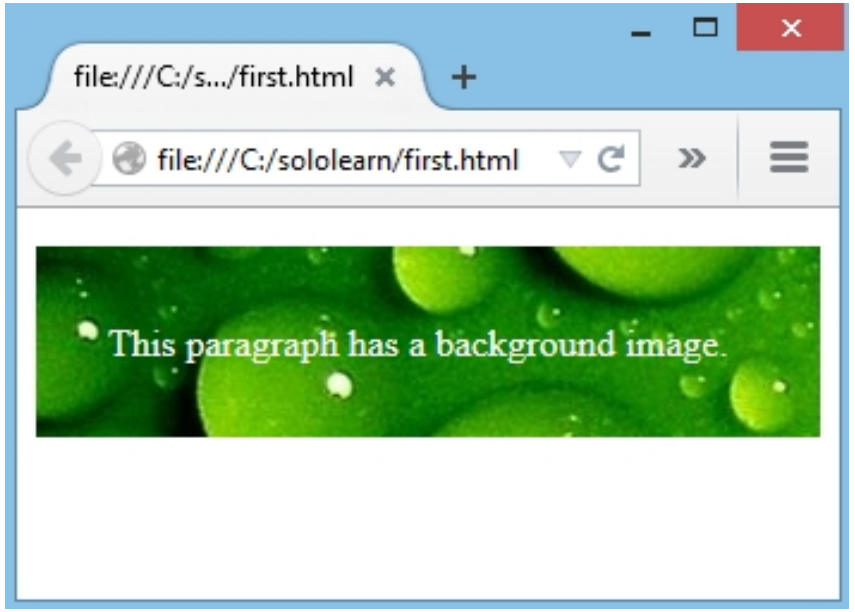
- By default, a background-image is placed at the top-left corner of an element, and is repeated both vertically and horizontally to cover the entire element.

The background-image Property

- Background-image can be set not only for the whole page, but for individual elements, as well.
  Below we set a background image for the element.
- **The HTML:**
  This paragraph has a background image.
- **The CSS:**

```
p {
    padding: 30px;
    background-image:
    url("green_photo.jpg");
    color: white;
}
```

- Result:



- To specify more than one image, just separate the URLs with **commas**.

Styling the Lists

The list-style-type Property

- The CSS list properties allow us to set different list item markers. In HTML, there are two types of lists:
  - **unordered lists** (<ul>) - the list items are marked with bullets
  - **ordered lists** (<ol>) - the list items are marked with numbers or letters
- With CSS, lists can be styled further, and images can be used as the list item marker.
- One of the ways is to use the **list-style-type** property, which can be set to **circle**, **square**, **decimal**, **disc**, **lower-alpha**, etc.

- **The HTML:**

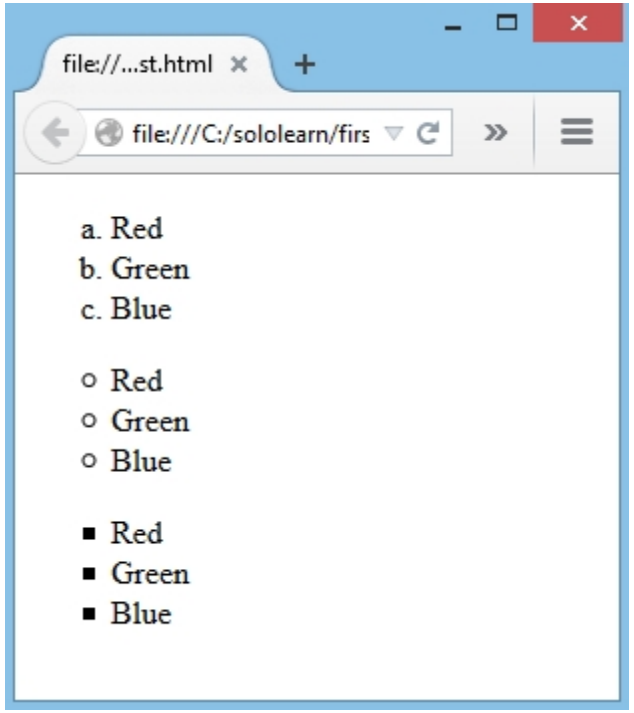
```
<ol class="lower-alpha">
    <li>Red</li>
    <li>Green</li>
    <li>Blue</li>
</ol>
<ul class="circle">
    <li>Red</li>
    <li>Green</li>
    <li>Blue</li>
</ul>
<ul class="square">
    <li>Red</li>
    <li>Green</li>
    <li>Blue</li>
</ul>
```

The list-style-type Property

- **The CSS:**

```
ol.lower-alpha {
    list-style-type: lower-alpha;
}
ul.circle {
    list-style-type: circle;
}
ul.square {
    list-style-type: square;
}
```

- Result:



- Some of the values are for unordered lists, and some for ordered lists.

The List Image and Position

- There are also other list properties, such as:
  - **list-style-image** - specifies an image to be used as the list item marker.
  - **list-style-position** - specifies the position of the marker box (inside, outside).
- In the example below, we use an image as the list item marker, and specify the position to be inside of the content flow.

- **The HTML:**

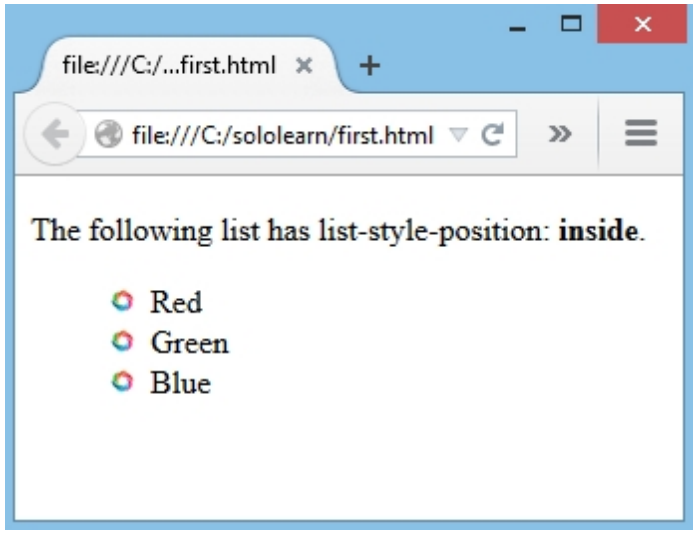
```
<p>The following list has list-style-position:
<strong>inside</strong>.</p>
<ul>
    <li>Red</li>
    <li>Green</li>
    <li>Blue</li>
</ul>
```

The List Image and Position

- **The CSS:**

```
ul {
  list-style-image: url("logo.jpg");
  list-style-position: inside;
}
```

- Result:



- "list-style-position: **outside**" is the default value.

The list-style Property

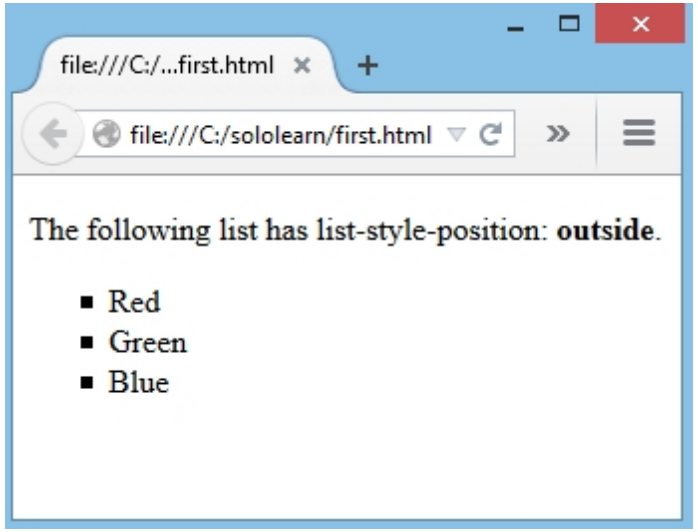
- The **list-style** property is a shorthand property for setting list-style-type, list-style-image and list-style-position. It is used to set all of the list properties in one declaration:

```
ul {
    list-style: square outside none;
}
```

- This would be the same as the longhand version.

```
ul {
    list-style-type: square;
    list-style-position: outside;
    list-style-image: none;
}
```

- Result:



- If one of the property values are missing, the default value for the missing property will be inserted, if any.

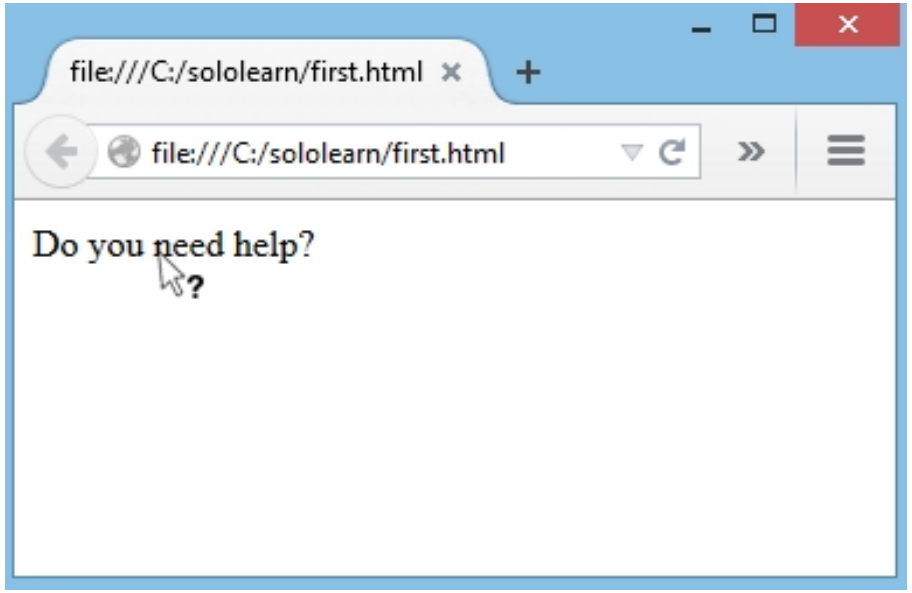
Customizing the Mouse Cursor

Setting the Mouse Cursor Style

- CSS allows you to set your desired cursor style when you mouse over an element. For example, you can change your cursor into a hand icon, help icon, and much more, rather than using the default pointer.
- In the example below, the mouse pointer is set to a help icon when we mouse over the span element:

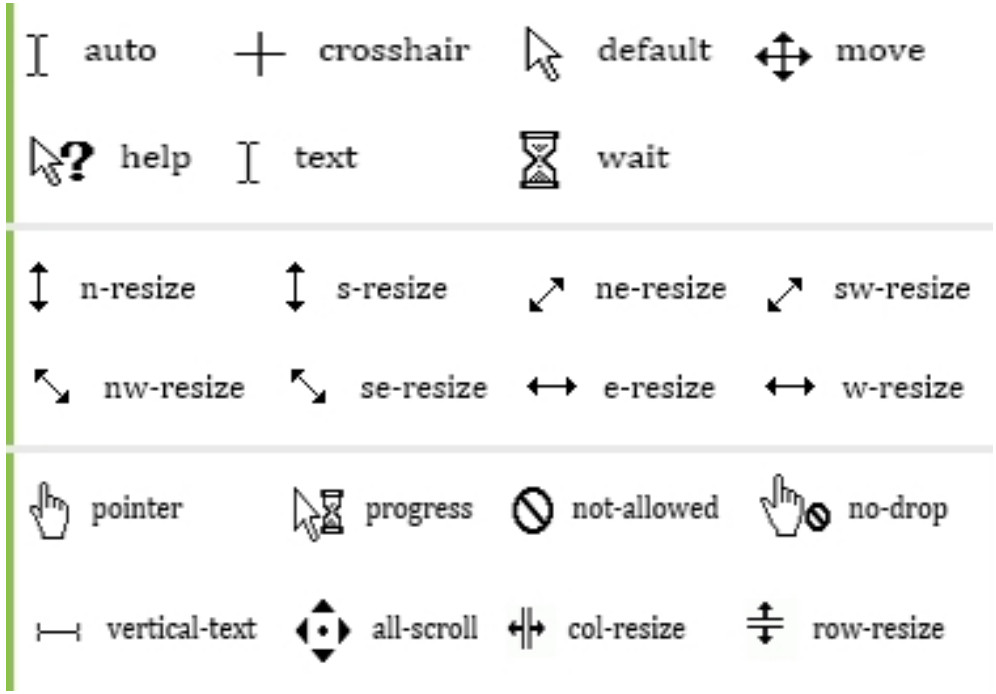
```
<span style="cursor:help;">
    Do you need help?
</span>
```

- Result:



The cursor Property Values

- There are numerous other possible values for the **cursor** property, such as:
  - **default** - default cursor
  - **crosshair** - cursor displays as crosshair
  - **pointer** - cursor displays hand icon
- The list of possible values is quite long. The image below demonstrates the various available styles:



- CSS allows you to set your desired cursor style when you mouse over an element.

Introduction to JavaScript

Welcome to JavaScript

- **JavaScript** is one of the most popular programming languages on earth and is used to add interactivity to webpages, process data, as well as create various applications (mobile apps, desktop apps, games, and more)

- Learning the fundamentals of a language will enable you to create the program you desire, whether client-side or server-side.

Creating Your First JavaScript

Your First JavaScript

- Let's start with adding JavaScript to a webpage.
- JavaScript on the web lives inside the **HTML** document.
- In HTML, JavaScript code must be inserted between **<script>** and **</script>** tags:
  <script>
  ...
  </script>
- JavaScript can be placed in the HTML page's **<body>** and **<head>** sections.
  In the example below, we placed it within the **<body>** tag.
- Remember that the script, which is placed in the head section, will be executed before the <body> is rendered. If you want to get elements in the body, it's a good idea to place your script at the end of the body tag.

Output

- Let's use JavaScript to print "Hello World" to the browser.

```
<html>
    <head> </head>
    <body>
        <script>
            document.write("Hello
            World!");
        </script>
    </body>
</html>
```

- The **document.write()** function writes a string into our HTML document. This function can be used to write text, HTML, or both.

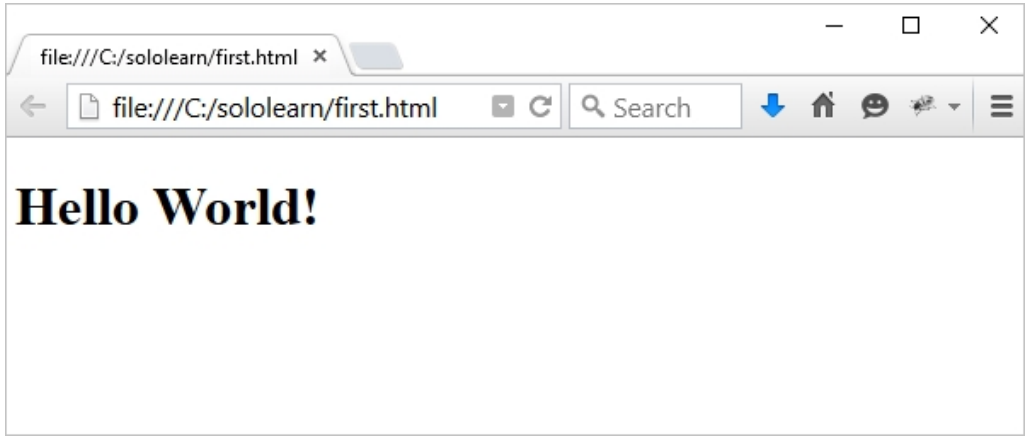
- The **document.write()** method should be used only for testing. Other output mechanisms appear in the upcoming lessons.

Formatting Text

- Just like in HTML, we can use HTML tags to format text in JavaScript. For example, we can output the text as a heading.

```
<html>
    <head> </head>
    <body>
        <script>
            document.write("<h1>
            Hello World!</h1>");
        </script>
    </body>
</html>
```

- Result:



- You can output almost everything in the webpage using JavaScript. Many JavaScript frameworks use this to create HTML pages.

Adding JavaScript to a Web Page

JavaScript in <head>

- You can place any number of scripts in an HTML document.
- Typically, the script tag is placed in the head of the HTML document:

```
<html>
    <head>
        <script>
        </script>
    </head>
    <body>
    </body>
</html>
```

- There is also a <noscript> tag. Its content will be shown if the client's browser doesn't support JS scripts.

JavaScript in <body>

- Alternatively, include the JavaScript in the <body> tag.

```
<html>
    <head> </head>
    <body>
        <script>
        </script>
    </body>
</html>
```

- It's a good idea to place scripts at the bottom of the <body> element.
- This can improve page load, because HTML display is not blocked by scripts loading.

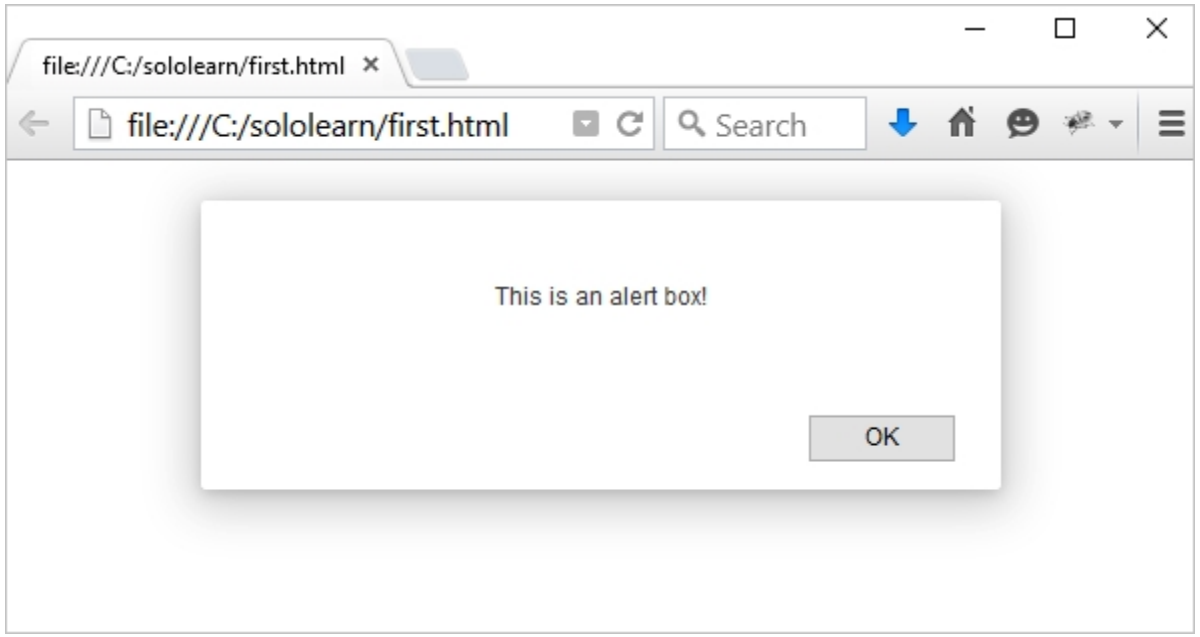
The <script> Tag

- The <script> tag can take two attributes, **language** and **type**, which specify the script's type:
  <script **language**="javascript" **type**="text/javascript">
  </script>
- The **language** attribute is deprecated, and should not be used.
- In the example below, we created an alert box inside the script tag, using the **alert()** function.

```
<html>
    <head>
        <title></title>
        <script
        type="text/javascript">
        alert("This is an alert box!");
        </script>
    </head>
    <body>
    </body>
</html>
```

Adding JavaScript to a Web Page

- Result:

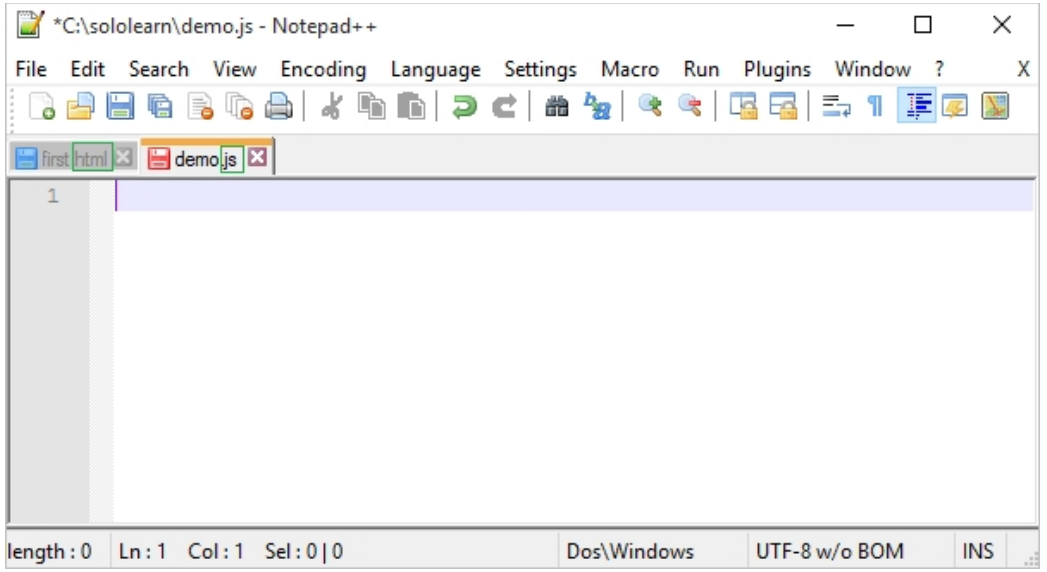


- The **type** attribute: <script type="text/javascript"> is also no longer required, as JavaScript is the default HTML scripting language.

External JavaScript

External JavaScript

- Scripts can also be placed in **external files**.
- External scripts are useful and practical when the same code is used in a number of different web pages.
- JavaScript files have the **file extension .js**.
- Below, we've created a new **text file**, and called it **demo.js**.



- Having JS scripts in separate files makes your code much readable and clearer.

External JavaScript

- To use an external script, put the name of the script file in the **src** (source) attribute of the <script> tag.

- Here is an example:

```
<html>
  <head>
  <title> </title>
    <script src="demo.js"></script>
  </head>
  <body>
  </body>
</html>
```

- Your **demo.js** file includes the following JavaScript:

```
alert("This is an alert box!");
```

- External scripts cannot contain <script> tags.

External JavaScript

- You can place an external script reference in <head> or <body>, whichever you prefer.

- The script will behave as if it were located exactly where the <script> tag is located.

- Placing a JavaScript in an external file has the following advantages:

  - It separates HTML and code.
  - It makes HTML and JavaScript easier to read and maintain.
  - Cached JavaScript files can speed up page loads.

Comments in JavaScript

- Not all JavaScript statements are "executed".
- Code after a double slash //, or between /* and */, is treated as a **comment**.
- Comments are ignored, and are not executed.
- **Single line** comments use double slashes.

```
<script>
    // This is a single line comment
    alert("This is an alert box!");
</script>
```

- It's a good idea to make a comment about the logic of large functions to make your code more readable for others.

Multiple-Line Comments

- Everything you write between /*and */ will be considered as a multi-line comment.

- Here is an example.

```
<script>
    /* This code
    creates an
    alert box */
    alert("This is an alert box!");
</script>
```

- Comments are used to describe and explain what the code is doing.

Variables

Variables

- **Variables** are containers for storing data values. The value of a variable can change throughout the program.

- Use the **var** keyword to declare a variable:
  var x = 10;

- In the example above, the value **10** is assigned to the variable **x**.

- JavaScript is case sensitive. For example, the variables *lastName* and *lastname*, are two different variables.

The Equal Sign

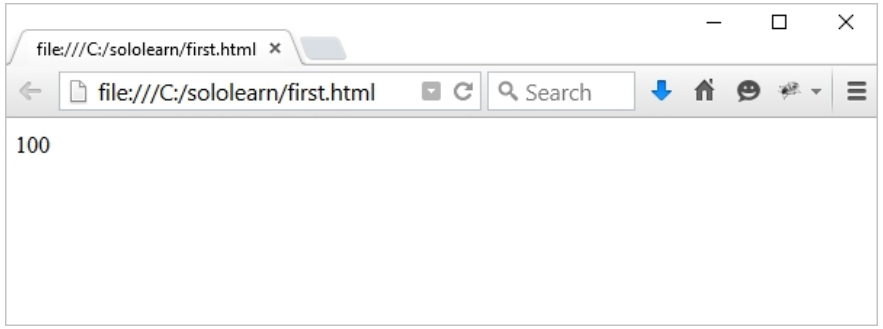
- In JavaScript, the equal sign (=) is called the "**assignment**" operator, rather than an "equal to" operator.
- For example, **x = y** will assign the value of **y** to **x**.
- A variable can be declared without a value. The value might require some calculation, something that will be provided later, like user input.
- A variable declared without a value will have the value **undefined**.

Using Variables

- Let's assign a value to a variable and output it to the browser.

```
var x = 100;
document.write(x);
```

- **Result:**



- Using variables is useful in many ways. You might have a thousand lines of code that may include the variable x. When you change the value of x **one time**, it will automatically be changed in **all places** where you used it.
- Every written "instruction" is called a **statement**. JavaScript statements are separated by **semicolons**.

Naming Variables

- JavaScript variable names are case-sensitive.
  In the example below we changed x to uppercase:
  var **x** = 100;
  document.write(**X**);

- This code will not result in any output, as x and X are two different variables.

- Hyphens are not allowed in JavaScript. It is reserved for subtractions.

- Naming rules:
  - The first character **must be** a letter, an underscore (_), or a dollar sign ($). Subsequent characters may be letters, digits, underscores, or dollar signs.
  - Numbers are **not allowed** as the first character.
  - Variable names **cannot** include a **mathematical or logical operator** in the name. For instance, *2*something* or *this+that*;
  - JavaScript names **must not contain spaces**.

Naming Variables

- There are some other rules to follow when naming your JavaScript variables:

  - You **must not** use any **special symbols**, like *my#num, num%*, etc.
  - Be sure that you do not use any of the following JavaScript reserved words.

- When you get more familiar with JavaScript, remembering these keywords will be much easier.

Reserved Words in JavaScript			
abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

Data Types

Data Types

- The term **data type** refers to the types of values with which a program can work. JavaScript variables can hold many data types, such as **numbers**, **strings**, **arrays**, and more.

- Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point, etc.

- JavaScript numbers can be written with or without decimals.
  var num = 42; // A number without decimals

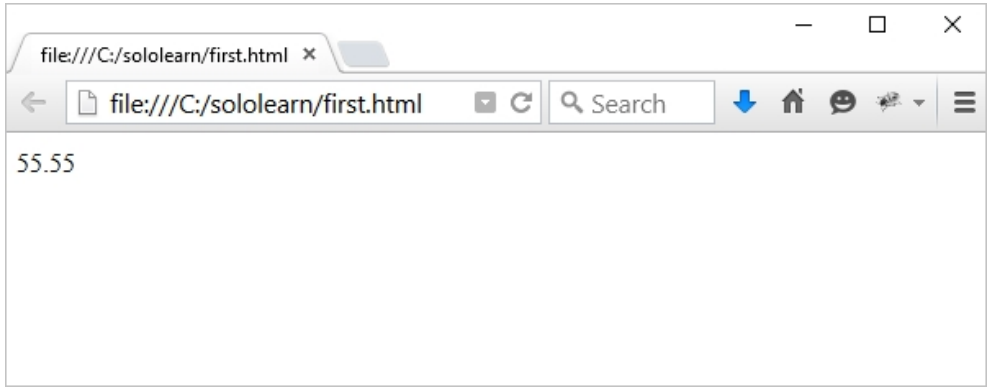
- This variable can be easily changed to other types by assigning to it any other data type value, like num = 'some random string'.

Floating-Point Numbers

- JavaScript numbers can also have decimals.

```
<script>
    var price = 55.55;
    document.write(price);
</script>
```

- Result:



- JavaScript numbers are always stored as **double precision floating point numbers**.

Strings

- JavaScript **strings** are used for storing and manipulating text.

- A string can be any text that appears within **quotes**. You can use single or double quotes.
  var name = 'John';
  var text = "My name is John Smith";

- You can use quotes inside a string, as long as they don't match the quotes surrounding the string.
  var text = "My name is 'John' ";

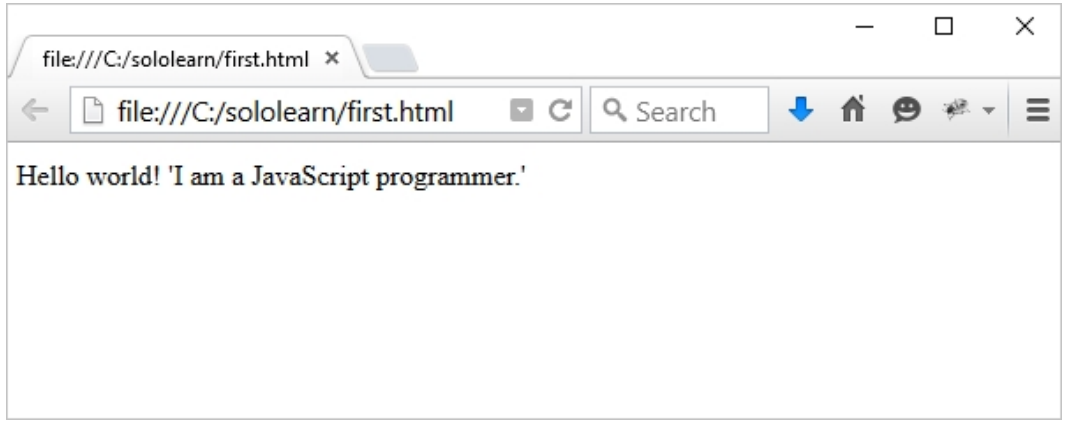
- You can get double quotes inside of double quotes using the escape character like this: \" or \' inside of single quotes.

Strings

- As strings must be written within quotes, quotes inside the string must be handled. The **backslash (\) escape character** turns special characters into string characters.

```
var sayHello = 'Hello world! \'I am a JavaScript programmer.\' ';
document.write(sayHello);
```

- Result:



Strings

- The escape character (\) can also be used to insert other special characters into a string.
  These special characters can be added to a text string using the backslash sign.

- If you begin a string with a single quote, then you should also end it with a single quote. The same rule applies to double quotes.
  Otherwise, JavaScript will become confused.

Code	Outputs
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

Booleans

- In JavaScript Boolean, you can have one of two values, either **true** or **false**.

- These are useful when you need a data type that can only have one of two values, such as Yes/No, On/Off, True/False.

- **Example:**
  var isActive = true;
  var isHoliday = false;

- The Boolean value of 0 (zero), null, undefined, empty string is **false**.
- Everything with a "real" value is **true**.

Math Operators

Arithmetic Operators

- Arithmetic operators perform arithmetic functions on numbers (literals or variables).

Operator	Description	Example
+	Addition	25 + 5 = 30
-	Subtraction	25 - 5 = 20
*	Multiplication	10 * 20 = 200
/	Division	20 / 2 = 10
%	Modulus	56 % 3 = 2
++	Increment	var a = 10; a++; Now a = 11
--	Decrement	var a = 10; a--; Now a = 9

Arithmetic Operators

- In the example below, the addition operator is used to determine the sum of two numbers.

```
var x = 10 + 5;
document.write(x);
// Outputs 15
```

- You can add as many numbers or variables together as you want or need to.

```
var x = 10;
var y = x + 5 + 22 + 45 + 6548;
document.write(y);
//Outputs 6630
```

- You can get the result of a string expression using the eval() function, which takes a string expression argument like eval("10 * 20 + 8") and returns the result. If the argument is empty, it returns undefined.

Multiplication

- The multiplication operator (*) multiplies one number by the other.

```
var x = 10 * 5;
document.write(x);

// Outputs 50
```

- 10 * '5' or '10' * '5' gives the same result. Multiplying a number with string values like 'sololearn' * 5 returns NaN (Not a Number).

Division

- The / operator is used to perform division operations:

```
var x = 100 / 5;
document.write(x);

// Outputs 20
```

- Remember to handle cases where there could be a division by 0.

The Modulus

- Modulus (%) operator returns the division remainder (what is left over).

```
var myVariable = 26 % 6;

//myVariable equals 2
```

- In JavaScript, the modulus operator is used not only on integers, but also on floating point numbers.

Increment & Decrement

- **Increment ++**
  The increment operator increments the numeric value of its operand by one. If placed before the operand, it returns the incremented value. If placed after the operand, it returns the original value and then increments the operand.

- **Decrement --**
  The decrement operator decrements the numeric value of its operand by one. If placed before the operand, it returns the decremented value. If placed after the operand, it returns the original value and then decrements the operand.

Increment & Decrement

- Some examples:

Operator	Description	Example	Result
var++	Post Increment	var a = 0, b = 10; var a = **b++**;	a = 10 and b = 11
++var	Pre Increment	var a = 0, b = 10; var a = **++b**;	a = 11 and b = 11
var--	Post Decrement	var a = 0, b = 10; var a = **b--**;	a = 10 and b = 9
--var	Pre Decrement	var a = 0, b = 10; var a = **--b**;	a = 9 and b = 9

- As in school mathematics, you can change the order of the arithmetic operations by using parentheses.
- Example: var x = (100 + 50) * 3;

Assignment Operators

Assignment Operators

- Assignment operators assign values to JavaScript variables.

Operator	Example	Is equivalent to
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x -y
*=	x *= y	x= x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

- You can use multiple assignment operators in one line, such as x -= y += 9.

Assignment Operators

- Assignment operators assign values to JavaScript variables.

Operator	Example	Is equivalent to
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x -y
*=	x *= y	x= x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

- You can use multiple assignment operators in one line, such as x -= y += 9.

Comparison Operators

Comparison Operators

- Comparison operators are used in logical statements to determine equality or difference between variables or values. They return **true** or **false**.

- The **equal to (==)** operator checks whether the operands' values are equal.

```
var num = 10;
// num == 8 will return false
```

- You can check all types of data; comparison operators always return true or false.

Comparison Operators

- The table below explains the comparison operators.

Operator	Description	Example
==	Equal to	5 == 10 false
===	Identical (equal and of same type)	5 === 10 false
!=	Not equal to	5 != 10 true
!==	Not Identical	10 !== 10 false
>	Greater than	10 > 5 true
>=	Greater than or equal to	10 >= 5 true
<	Less than	10 < 5 false
<=	Less than or equal to	10 <= 5 false

- When using operators, be sure that the arguments are of the same data type; numbers should be compared with numbers, strings with strings, and so on.

Logical or Boolean Operators

Logical Operators

- **Logical** Operators, also known as **Boolean** Operators, evaluate the expression and return **true** or **false**.
- The table below explains the logical operators (**AND**, **OR**, **NOT**).

Logical Operators	
&&	Returns true, if both operands are true
||	Returns true, if one of the operands is true
!	Returns true, if the operand is false, and false, if the operand is true

- You can check all types of data; comparison operators always return true or false.

Logical Operators

- In the following example, we have connected two Boolean expressions with the **AND** operator.
  (4 > 2) && (10 < 15)

- For this expression to be **true**, both conditions must be **true**.
  - The first condition determines whether 4 is greater than 2, which is **true**.
  - The second condition determines whether 10 is less than 15, which is also **true**.
- Based on these results, the whole expression is found to be **true**.

- **Conditional (Ternary) Operator**
- Another JavaScript conditional operator assigns a value to a variable, based on some condition.
  Syntax:
  variable = (condition) ? value1: value2
  For example:
  var isAdult = (age < 18) ? "Too young": "Old enough";
- If the variable *age* is a value below 18, the value of the variable *isAdult* will be "Too young". Otherwise the value of *isAdult* will be "Old enough".
- Logical operators allow you to connect as many expressions as you wish.

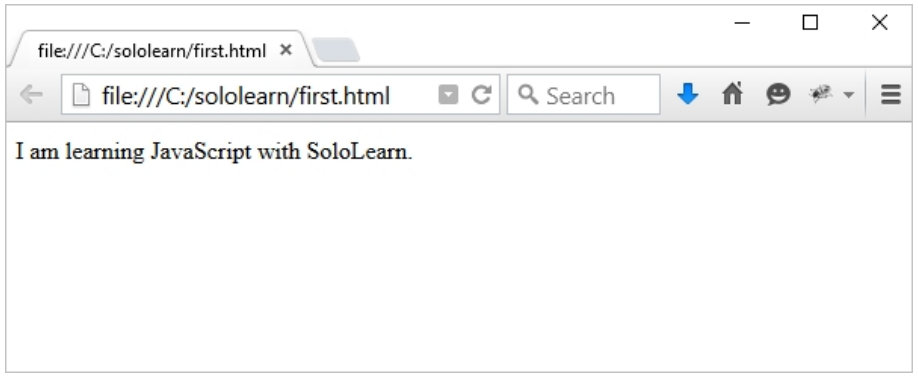
String Operators

String Operators

- The most useful operator for strings is *concatenation*, represented by the + sign.
- Concatenation can be used to build strings by joining together multiple strings, or by joining strings with other types:

```
var mystring1 = "I am learning ";
var mystring2 = "JavaScript with SoloLearn.";
document.write(mystring1 + mystring2);
```

- The above example declares and initializes two string variables, and then concatenates them.



- Numbers in quotes are treated as strings: "42" is not the number 42, it is a string that includes two characters, 4 and 2.

The if Statement

The if Statement

- Very often when you write code, you want to perform different actions based on different conditions.
- You can do this by using **conditional statements** in your code.

- Use **if** to specify a block of code that will be executed if a specified condition is true.

```
if (condition) {
    statements
}
```

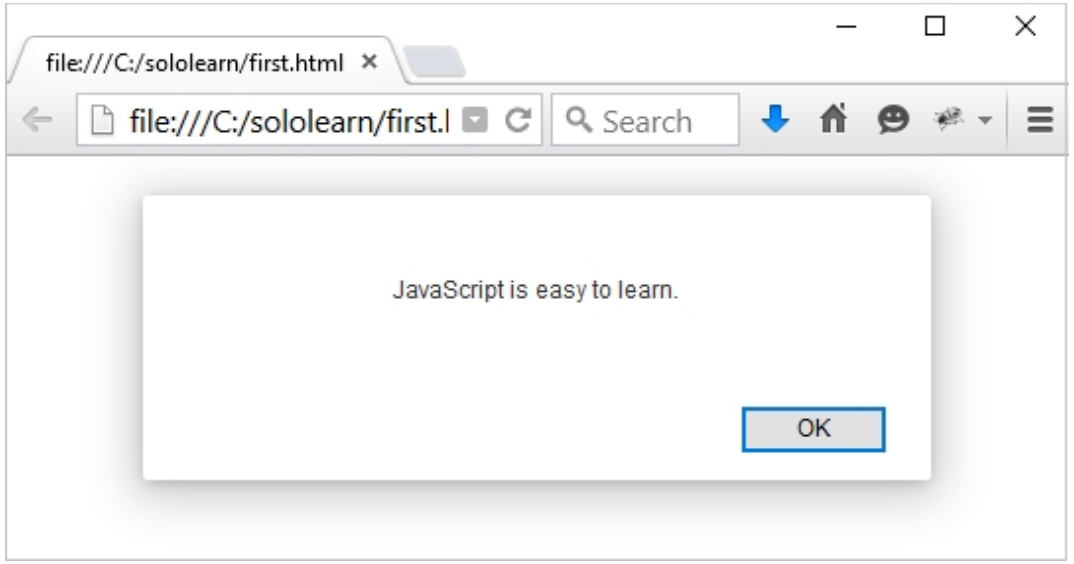
- The statements will be executed only if the specified condition is **true**.

- **Example:**

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 < myNum2) {
 alert("JavaScript is easy to learn.");
}
```

The if Statement

- Result:



- As seen in the picture above, the JavaScript **alert()** method is used to generate a popup alert box that contains the information provided in parentheses.

The if Statement

- This is another example of a **false** conditional statement.

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 > myNum2) {
    alert("JavaScript is easy to
    learn.");
}
```

- As the condition evaluates to false, the alert statement is skipped and the program continues with the line after the if statement's closing curly brace.

- Note that **if** is in lowercase letters. Uppercase letters (If or IF) will generate an error.

The if else Statement

The else Statement

- Use the **else** statement to specify a block of code that will execute if the condition is **false**.

```
if (expression) {
   // executed if condition is true
}
else {
   // executed if condition is false
}
```

- You can skip curly braces if your code under the condition contains only one command.

The else Statement

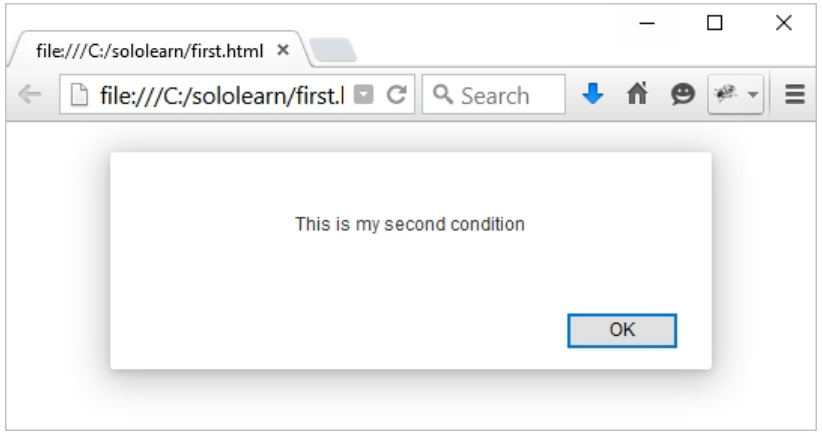
- The example below demonstrates the use of an **if...else** statement.

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 > myNum2) {
     alert("This is my first condition");
}
else {
    alert("This is my second condition");
}
```

- The above example says:
  - **If** *myNum1* is greater than *myNum2,* alert "This is my first condition";
  - **Else**, alert "This is my second condition".

- The browser will print out the second condition, as 7 is not greater than 10.

- Result:



- There is also another way to do this check using the ? operator: a > b ? alert(a) : alert(b).

The if else if else Statement

else if

- You can use the **else if statement** to specify a new condition if the first condition is false.

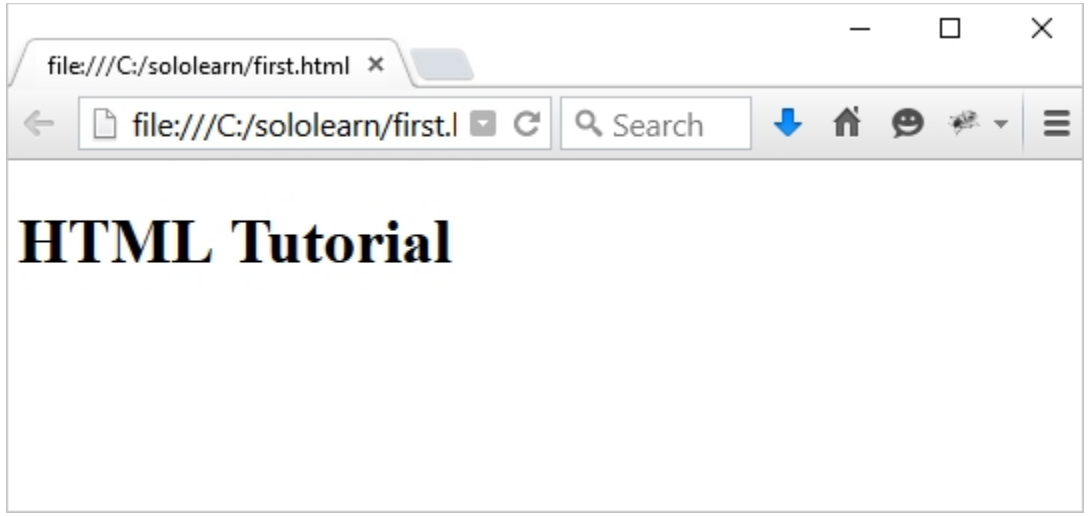
- **Example:**

```
var course = 1;
if (course == 1) {
    document.write("<h1>HTML Tutorial</h1>");
} else if (course == 2) {
    document.write("<h1>CSS Tutorial</h1>");
} else {
    document.write("<h1>JavaScript Tutorial</h1>");
}
```

else if

- The above code says:
  - **if** course is equal to 1, output "HTML Tutorial";
  - **else, if** course is equal to 2, output "CSS Tutorial";
  - if none of the above condition is true, then output "JavaScript Tutorial";

- **course** is equal to 1, so we get the following result:



- The final **else** statement "ends" the else if statement and should be always written after the **if** and **else if** statements.

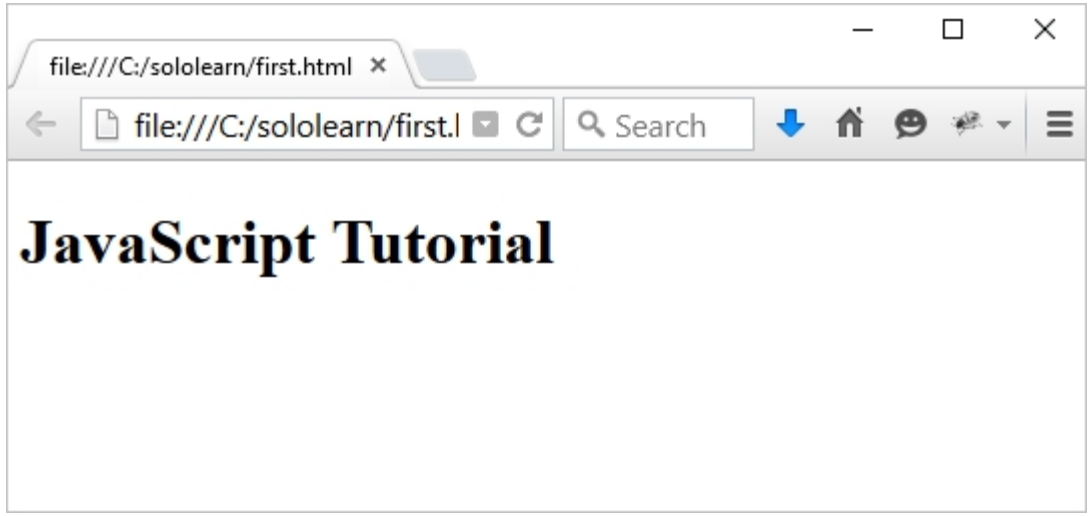
else if

- The final **else** block will be executed when **none** of the conditions is true.

- Let's change the value of the **course** variable in our previous example.

```
var course = 3;
if (course == 1) {
    document.write("<h1>HTML
    Tutorial</h1>");
} else if (course == 2) {
    document.write("<h1>CSS
    Tutorial</h1>");
} else {
    document.write("<h1>JavaScript
    Tutorial</h1>");
}
```

- Result:



- You can write as many **else if** statements as you need.

The For Loop

Loops

- Loops can execute a block of code a number of times. They are handy in cases in which you want to run the same code repeatedly, adding a different value each time.

- JavaScript has three types of loops: **for**, **while**, and **do while**.

- The **for** loop is commonly used when creating a loop.

Loops

- The syntax:

```
for (statement 1; statement 2; statement 3) {
code block to be executed
}
```

- As you can see, the **classic for loop** has **three** components, or statements.

- **Statement 1** is executed before the loop (the code block) starts.
- **Statement 2** defines the condition for running the loop (the code block).
- **Statement 3** is executed each time after the loop (the code block) has been executed.
  As you can see, the **classic for loop** has **three** components, or statements.

The For Loop

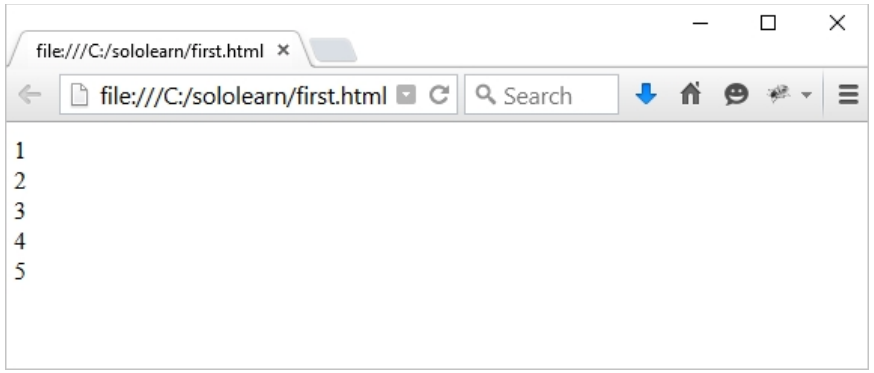
- The example below creates a **for** loop that prints numbers 1 through 5.

```
for (i=1; i<=5; i++) {
  document.write(i + "<br />");
}
```

- In this example, **Statement 1** sets a variable before the loop starts (var i = 1).
- **Statement 2** defines the condition for the loop to run (i must be less than or equal to 5).
- **Statement 3** increases a value (i++) each time the code block in the loop has been executed.

The For Loop

- Result:



- **Statement 1** is optional, and can be omitted, if your values are set before the loop starts.

```
var i = 1;
for (; i<=5; i++) {
    document.write(i + "<br />");
}
```

- Also, you can initiate more than one value in **statement 1**, using **commas** to separate them.

```
for (i=1, text=""; i<=5; i++) {
    text = i;
    document.write(i + "<br />");
}
```

The For Loop

- If **statement 2** returns true, the loop will start over again, if it returns false, the loop will end. Statement 2 is also optional.
- If you omit statement 2, you must provide a **break** inside the loop. Otherwise, the loop will never end.
- You can have multiple nested for loops.
- **Statement 3** is used to change the initial variable. It can do anything, including negative increment (i--), positive increment (i = i + 15), or anything else.
- Statement 3 is also optional, and it can be omitted if you increment your values inside the loop.

```
var i = 0;
for (; i < 10; ) {
    document.write(i);
    i++;
}
```

The While Loop

The While Loop

- The **while** loop repeats through a block of code, as long as a specified condition is **true**.

- **Syntax:**

```
while (condition) {
    code block
}
```

- The **condition** can be any conditional statement that returns true or false.

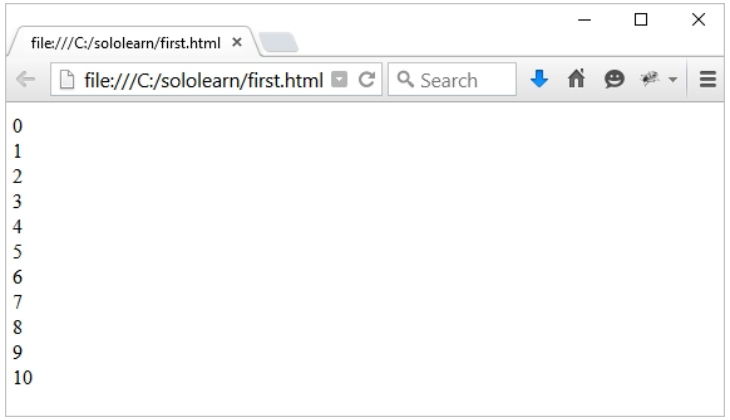
The While Loop

- Consider the following example.

```
var i=0;
while (i<=10) {
    document.write(i + "<br />");
    i++;
}
```

- The loop will continue to run as long as i is less than, or equal to, 10. Each time the loop runs, it will increase by 1.

- This will output the values from 0 to 10.



- Be careful writing conditions. If a condition is always true, the loop will run forever.

The While Loop

- If you forget to increase the variable used in the condition, the loop will never end.

- Make sure that the condition in a while loop eventually becomes **false**.

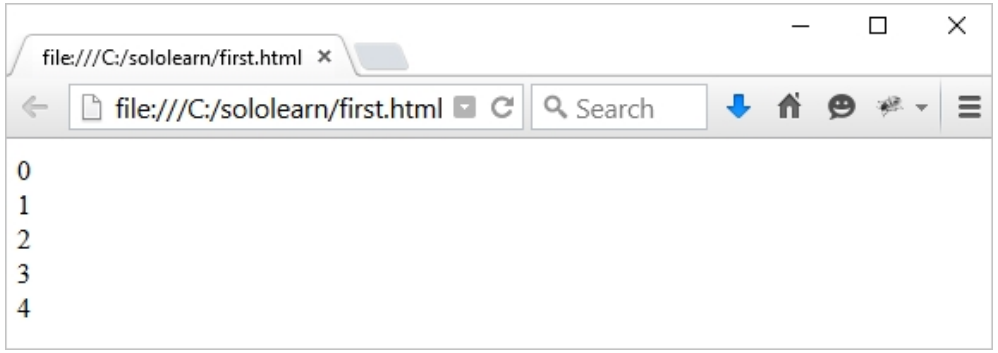
Break and Continue

Break

- The **break** statement "jumps out" of a loop and continues executing the code after the loop.

```
for (i = 0; i <= 10; i++) {
    if (i == 5) {
        break;
    }
    document.write(i + "<br />");
}
```

- Once i reaches 5, it will break out of the loop.



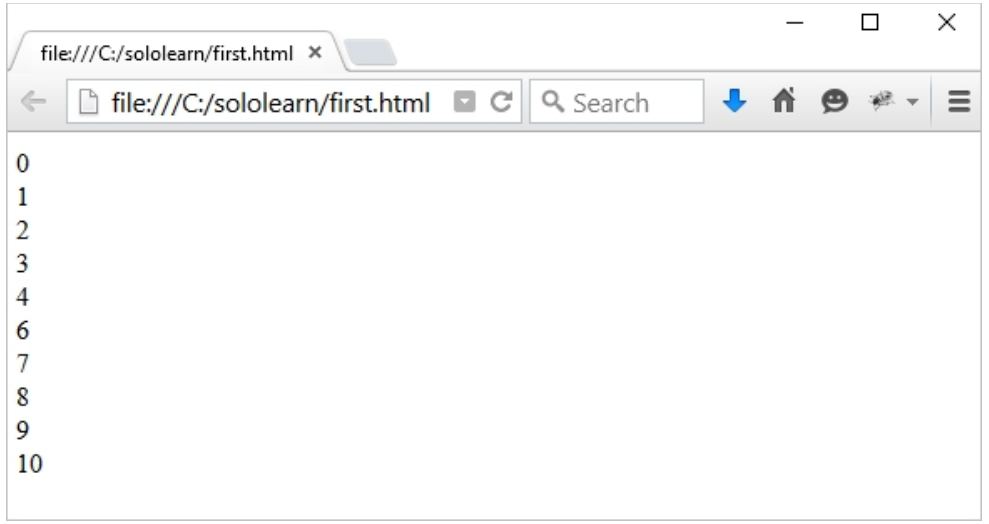
- You can use the return keyword to return some value immediately from the loop inside of a function. This will also break the loop.

Continue

- The **continue** statement breaks only one iteration in the loop, and continues with the next iteration.

```
for (i = 0; i <= 10; i++) {
    if (i == 5) {
        continue;
    }
    document.write(i + "<br />");
}
```

- Result:



- The value 5 is not printed, because **continue** skips that iteration of the loop.