**Bits and Pieces**

* Online W3C HTML Validator: <http://validator.w3.org/>
* This *document type declaration*—doctype for short—also points the Web browser to files on the Internet that contain definitions for that type of file. The doctype plays a key role in how a Web browser displays CSS—in fact a missing or incorrect doctype’s enough to make Internet Explorer completely mishandle the presentation of a CSS-heavy Web page. A DTD is an XML document that explains what tags, attributes, and values are valid for a particular type of HTML. And for each version of HTML, there’s a corresponding DTD.
* When you mistype the doctype declaration, you can throw most browsers into an altered state called *quirks mode*. For more technical information on quirks mode, visit <http://www.quirksmode.org/index.html?/css/quirksmode.html> and<http://hsivonen.iki.fi/doctype/>.
* Use HTML to organize your content, and CSS to make that content look great.
* Try to use only one <h1> tag per page and use that to identify the main topic in that page, something like chapter title. The page will get indexed properly for search engine benefit. Use headings to indicate other important topic. If two topics are of same importance, use <h2> on both. If one is main topic and the other is subtopic, use <h2> followed by <h3> for the subtopic.

**HTML – DOCTYPE DECLARATIONS**

The <!DOCTYPE> declaration must be the very first thing in your HTML document, before the <html> tag. The <!DOCTYPE> declaration is not an HTML tag; it is an instruction to the web browser about what version of HTML the page is written in.

In HTML 4.01, the <!DOCTYPE> declaration refers to a DTD, because HTML 4.01 was based on Standard Generalized Markup Language (SGML). The DTD specifies the rules for the markup language, so that the browsers render the content correctly.

HTML5 is not based on SGML, and therefore does not require a reference to a DTD. See: <http://www.w3.org/TR/REC-html40/intro/sgmltut.html>

HTML 5

<!DOCTYPE html>

HTML 4.01 Strict

This DTD contains all HTML elements and attributes, but does NOT INCLUDE presentational or deprecated elements (like font). Framesets are not allowed.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

HTML 4.01 Transitional

This DTD contains all HTML elements and attributes, INCLUDING presentational and deprecated elements (like font). Framesets are not allowed.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

HTML 4.01 Frameset

This DTD is equal to HTML 4.01 Transitional, but allows the use of frameset content.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">

XHTML 1.0 Strict

This DTD contains all HTML elements and attributes, but does NOT INCLUDE presentational or deprecated elements (like font). Framesets are not allowed. The markup must also be written as well-formed XML.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

XHTML 1.0 Transitional

This DTD contains all HTML elements and attributes, INCLUDING presentational and deprecated elements (like font). Framesets are not allowed. The markup must also be written as well-formed XML.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

XHTML 1.0 Frameset

This DTD is equal to XHTML 1.0 Transitional, but allows the use of frameset content.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

XHTML 1.1

This DTD is equal to XHTML 1.0 Strict, but allows you to add modules (for example to provide ruby support for East-Asian languages).

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

**XHTML**

To make an XHTML files comply with XML, however, there are a few strict rules to keep in mind:

* **Begin the page with a document type declaration**. That’s the first few lines in the file, starting with <!DOCTYPE. The document type declaration is the most important part of an XHTML page. The doctype denotes that the page is a type of XML document, in particular, an XHTML document
* **Tags and tag attributes must be lowercase**. Unlike with HTML, typing the tag <BODY> is a no-no; when you’re writing XHTML, capitalized tags aren’t invited to the party.
* **Quotation marks are required for tag attributes**. For example, links written like this: <a <http://www.google.com>> is valid in HTML, but won’t work in XHTML. You have to enclose the value of the Href property in quotes: <a ”<http://www.google.com>“>.
* **All tags (even empty tags) must be closed**. To create a paragraph in XHTML, for example, you must begin with <p> and end with </p>. Trouble is, some tags don’t come in pairs. These tags, called empty tags have no closing tag. The line break tag’s one example. To close an empty tag, include a forward slash at the end of the tag, like this: <br />.

**SIMPLE HTML IS SEARCH ENGINE FRIENDLY**

Once you take the mental leap of picturing HTML as the way to structure a document’s content, and CSS as the tool for making that content look good, you’ll discover additional benefits to writing lean, mean HTML. For one thing, you may boost your search engine ranking as determined by sites like Google, Yahoo, and MSN. That’s because when search engines crawl the Web, indexing the content on Web sites, they must go through *the entire* HTML on a page to discover the actual content. The old HTML-way of using special tags (like <font>) and lots of tables to design a page gets in the way of a search engine’s job. In fact, some search engines stop reading a page’s HTML after a certain number of characters. When you use a lot of HTML just for design, the search engine may miss important content on the page, or even fail to rank it at all.

By contrast, simple, structured HTML is easy for a search engine to read and index. Using an <h1> tag to indicate the most important topic of the page (as opposed to just making the text big and bold) is smart strategy: Search engines give greater weight to the contents inside that tag while indexing the page.

To see Google’s suggestions for building search-friendly Web sites, visit <http://www.google.com/webmasters/guidelines.html>.

You can also hear a podcast of an excellent speech on SEO (search engine optimization) from the 2005 Web Visions conference at<http://www.webvisionsevent.com/podcasts/WV05_Alan_Knecht.mp3> and read an article by that speaker covering the same topic at:<http://www.digital-web.com/articles/seo_and_your_web_site/>.

**HTML TAGS TO AVOID – Quick Reference**

* <font> - For controlling the display of text. **STOP** using this tag. Use CSS.
* <b> - For making text bold. **STOP** using this tag. Use CSS. Can use <strong> tag if required. Browser will show the text in bold.
* <i> - For making text italic. **STOP** using this tag. Use CSS. Can use <em> to emphasize a text. Browser will italicize the content.
* <table> - Use this tag only to display the content in tabular format like spreadsheet, schedule or chart. **STOP** using this tag for page layout. Use CSS.
* Attributes for <body> tag: **STOP** using presentation related <body> tag attributes like
* background
* bgcolor
* text
* link
* alink
* vlink

to set colors and images for page, text and links. Use CSS.

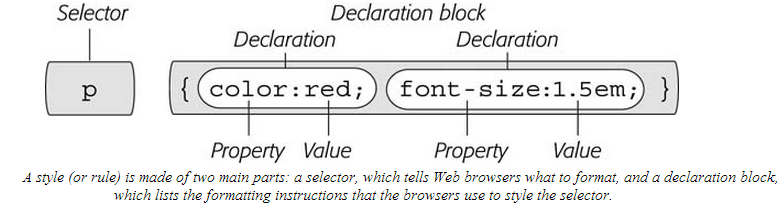
* Attributes to set page margin: **STOP** using browser specific browser attributes to set page margins. Use CSS
* leftmargin
* topmargin
* marginwidth
* marginheight

**HTML TAGS – Quick Reference**

* <div> - Stands for division. It has no inherent visual properties. Used for dividing a page into logical sections like header-section, menu-bar-section, footer-section, etc.
* <span> - Used for inline elements i.e. words or phrases that appear within a large paragraph. Other common inline HTML tags are <a> and <strong>.
* <cite> - Use this when you would want a text to be cited (to list the source from which one took information from i.e. like bibliography). It will make the text italic and also for search engine benefit.
* <br/> - **Don’t abuse this tag**. If your purpose to use this tag is to insert a line break without creating a new paragraph, then the different browsers can show the result differently. They can automatically insert a bit of space between paragraphs, including between headers and <p> tag. Use CSS margin controls to set the amount of space between paragraphs, headers and other block level elements.
* <p> - Use this tag for paragraphs of text.
* <ul> - Use unordered list when we have several related items such as navigation links, headlines.
* <ol> - Use ordered list (also called numbered list) to indicate steps in a process or to define the order of a set of items.
* <dl> - In HTML 4.01 it’s called definition list. In HTML 5 it’s called description list. It’s used in conjunction with <dt> and <dd> tags.
* <dt> - This tag defines terms/names. It’s called definition term. Used within <dl> tag.
* <dd> - This tag describes each term/name. It’s called definition description. Used within <dl> tag along with <dt>.
* <blockquote> - To include a quotation like a snippet of text from another Web site, a movie review. This is used for long passages. Browsers usually indent <blockquote> elements i.e. we might not see quotes around the text like we can see when using <q> tag. In HTML 4.01, this tag denotes a long quotation. In HTML 5, this tag denotes a section that is quoted from another source.
* <q> - Similar to <blockquote> tag i.e. to include a quotation like snippet of text. But this is used for one liner. Browsers usually insert quotation marks around the quoted text.
* <address> - Use this tag identify and supply contact information for the author of a page. Good for copyright notice section.

**CSS– Bits and Pieces**

* Online W3C CSS Validator: <http://jigsaw.w3.org/css-validator/>
* A CSS style/rule is made up of two elements: *selector* and *declaration block*. A selector is a web page element that the browser will format. The declaration block will contain the actual formatting instructions.
* Elements that make up a CSS style are
* Selector: it tells a Web browser, which element or elements on a page to style
* Declaration Block: the code following the selector which includes all the formatting options that should be applied to element(s) identified via the selector. The block begins with an opening brace ({) and ends with a closing brace (}).
* Declaration: Between the opening and closing braces of the declaration block, we add one or more declaration or formatting instructions. Every declaration has two parts: a property and a value, and ends with a semicolon (;).
* Property: They are the formatting options to indicate certain style effect like color, background-color, etc. It’s a work or few hyphenated words (we cannot have space between words for a single property. They have to be a single word).
* Value: the value assigned for the property like red for the property color.



* Internal Style Sheets: Present within the web page file, always between opening and closing HTML’s <style> tag within the <head> element. Web designer’s prefer to place the <style> tag just before closing the <head> element i.e. <style> should be the last child node within <head> tag.
* External Style Sheets: Attach the style sheet to a HTML page using <link rel=”stylesheet” type=”text/css” href=”path/to/stylesheet.css” />. Can also use ‘@import’ directive to connect the style sheet to the HTML page.
* @import directive has one advantage over <link> tag: attach an external style sheet to another external style sheet. Also @import directive is part of CSS language and not HTML.
* To import multiple style sheets using <link>, add multiple <link> tags each pointing to a unique stylesheet.
* To import style sheet using @import directive, have the following under <script> tag. Have a semi-colon in the end. Also the quotes around URL are optional, personally I prefer we have as it becomes easier to copy to <link> attribute if we want to change it. To include multiple style sheets, have multiple @import statements.

<style type=”text/css”>

@import url(“path/to/stylesheet.css”);

@import url(“path/to/another-stylesheet.css”);

</style>

* We can add regular CSS styles after the @import statement within the same <style> tag. It’s advisable to have @import statements at the start of <style> tag if they are present. Browsers can ignore @import directive after a regular CSS styles in the <style> tag.

**CSS– SELECTORS**

* Type of selectors: tag selectors, class selectors, id selectors
* Tag Selectors
* Page-Wide styling.
* Apply the declaration block to every occurrence of an HTML tag on the page.
* Selectors will bear the same name as the tag they style like p, h1, table, span, etc.
* Disadvantage: If we want some paragraphs to look red and some green, this will not help.
* Class Selectors
* Selective group styling like applying a style to some group of elements like apply the same styling for paragraph and heading or a certain group of paragraphse.
* Apply the declaration block to all the HTML tags that have an attribute named ‘class’. Like

<p class=”some-class”>This is some text</p>

* Selectors will start with a period followed by the class name (as defined in the HTML tag). For the above example, the class selector will be .some-class
* Allowed characters in class name are: letters, numbers, hyphens and underscores. Class name should always start with a letter.
* Class names are case-sensitive. So .some-class and .Some-class are two different selectors.
* ID Selectors
* ldanvl

**WORKAROUND WORKSHOP: DON’T GET CAUGHT IN THE CACHE**

A browser’s cache is a great speed-boost for Web surfers. Whenever the cache downloads and stores a frequently used file—like an external CSS file or an image—it saves precious moments traveling the relatively sluggish highways of the Internet. Instead of re-downloading the next time it needs the same file, the browser can go straight to the new stuff—like a yet-to-be-viewed page or graphic.

But what’s good for your visitors isn’t always good for you. Because the Web browser caches and recalls downloaded external CSS files, you can often get tripped up as you work on a site design. Say you’re working on a page that uses an external style sheet, and you preview the page in a browser. Something doesn’t look quite right, so you return to your Web editor and change the external CSS file. When you return to the Web browser and reload the page, the change you just made doesn’t appear! You’ve just been caught by the cache. When you reload a Web page, browsers don’t always reload the external style sheet, so you may not be seeing the latest and greatest version of your styles.

You have two ways around this snafu: turn off the cache or force the browser to reload everything.

In most browsers you can force reload a page (which also reloads all linked files) by pressing the Ctrl (⌘) key and pressing the browser’s Reload button; Ctrl+F5 also works on Windows for Internet Explorer; and Ctrl+Shift+R (⌘-Shift-R) is Firefox’s keyboard shortcut.

You can also completely turn off the cache. In Internet Explorer, choose Tools → Internet Options → General tab; in the Temporary Internet Files section, click Settings. Make sure the “Check for newer versions of stored pages” option’s set to “Every visit to the page.” Click OK twice to close the Internet Options window. In Firefox, choose Tools → Options (on Windows), or Firefox → Preferences (Mac) to open Firefox’s Preferences window; click the Privacy button, then the Cache tab, and then set the cache disk space to 0.

For Safari on the Mac, download the free Safari Enhancer at <http://www.versiontracker.com/dyn/moreinfo/macosx/17776>.

Turning off the cache can drastically slow your regular Web surfing activity, so make sure you turn it back on once you’re done editing your CSS files.

**AngularJS – Bits and Pieces**

ng-app

ng-controller

ng-hide

ng-model

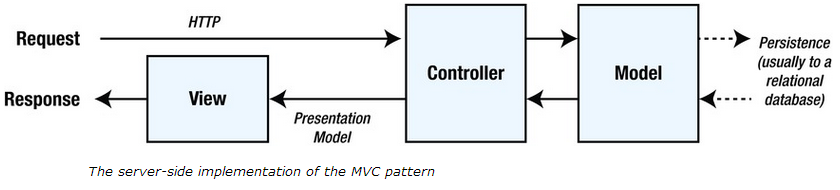
angular.module( String, Array)

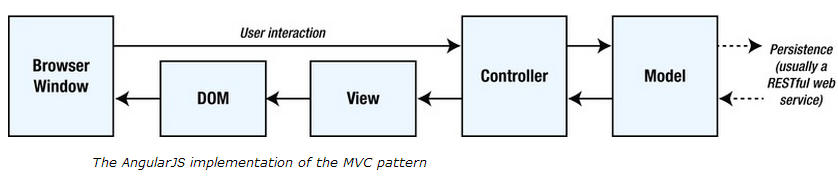
angular.controller (String, function)

**AngularJS - Notes**

🡺**MVC: Model-View-Controller**

The goal of the MVC framework is to divide an application into three functional areas i.e. model, view and controller; each area to contain both logic and data.





🡪Models

(a) Two types of models: view models and domain models.

(b) View Models: Represents data passed from Controller to View.

© View Models data are not persisted. It’s created by synthesizing some aspects of domain model data or in response to a user interaction. In the HTML, we could have ng-model attribute to filter out the entries. See filter example.

(d) Domain Models: Represents data in business domain. It should encapsulate operations, transformations and rules for creating, storing and manipulating that data.

The model in an application built using the MVC pattern should

* Contain the domain data
* Contain the logic for creating, managing, and modifying the domain data (even if that means executing remote logic via web services)
* Provide a clean API that exposes the model data and operations on it

The model should not

* Expose details of how the model data is obtained or managed (in other words, details of the data storage mechanism or the remote web service should not be exposed to controllers and views)
* Contain logic that transforms the model based on user interaction (because this is the controller’s job)
* Contain logic for displaying data to the user (this is the view’s job)

🡪Controllers

(a) Connective tissue in an AngularJS app. Connects Models and Views.

(b) They add business domain logic (behavior) to scopes, which are subset of models

A controller built using the MVC should

* Contain the logic required to initialize the scope
* Contain the logic/behaviors required by the view to present data from the scope
* Contain the logic/behaviors required to update the scope based on user interaction

The controller should not

* Contain logic that manipulates the DOM (that is the job of the view)
* Contain logic that manages the persistence of data (that is the job of the model)
* Manipulate data outside of the scope

🡪Views

* 1. Plain HTML. They are enhanced.
  2. Data binding and directives of AngularJS are used to make it dynamic.

Views should

* Contain the logic and markup required to present data to the user

Views should not

* Contain complex logic (this is better placed in a controller)
* Contain logic that creates, stores, or manipulates the domain model

Views can contain logic, but it should be simple and used sparingly. Putting anything but the simplest method calls or expressions in a view makes the overall application harder to test and maintain.

Knowing where to put logic becomes second nature as you get more experience in AngularJS development, but here are the three rules:

1. View logic should prepare data only for display and never modify the model.
2. Controller logic should never directly create, update, or delete data from the model.
3. The client should never directly access the data store.