**AngularJS Overview**

* Views are rendered in AngularJs as a combination of templates and controllers
* Templates are built using directives
* Data filtering using **filters**
* Controllers pull the data in using **factories, services, and providers**
* **Factories, services, and providers** interact with the server to pull in the data into application
* Angular application is organized as a module or as a group of cooperating modules. Each module has a configuration. **Routes** take care moving from one view to other.

**Front-End JS Frameworks**

* Well defined architectures (MVC, MVVM) helps with complexity of managing DOM manipulation and data updates manually
* Software Library:
  + Collection of implementation of behavior with a well defined interface by which the behavior is revoked
  + E.g. jQuery
* Software Framework
  + Abstraction in which software provides a generic functionality and you will writing you our code to add further to that functionality
  + Provides reusable environment that provides particular functionality
  + E.g. Angular, Ember, Backbone
* Library vs Framework
  + **Library:** a collection of functions, which are useful when writing web apps. You code is in charge and it calls into the library when sees fit.
  + **Frameworks:** a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific.
    - Hollywood principle: Don’t call us, we’ll call you! (Framework is in control)
    - Inversion of Control
    - Imperative vs. Declarative Programming
      * Traditionally we use **imperative** programming approach. We specify steps to be done and specify them in sequence. You’re in control to how the steps will be executed.
      * In **declarative** programming, you’re specifying what needs to be done but don’t care how. You leave that on framework to execute what needs to be done.
* Terms we’ll be hearing a lot in this course
  + Single Page Application/Rich Internet Aplication
  + Model-View-Controller
    - Data binding, routing
  + Scalable, Reusable, Maintainable JS code
  + Test driven development
* Popular JS Frameworks
  + Angular, Ember, Backbone, React, Aurelia, Meteor, Polymer (from Google), Knockout, Vue, Mercury
* Ember vs. Angular vs. Backbone
  + ***Backbone*** provides most flexibility and ***Ember*** is most opinionated. As for Ease of use ***Ember*** is hardest and ***Backbone*** is the easiest. Leaning curve ***Backbone*** is easy to start with and ***Ember*** is hardest. ***Angular*** sits in the middle for **Flexibility**, **ease of use**, and **learning curve**, hence it’s the most popularly used out of the three.

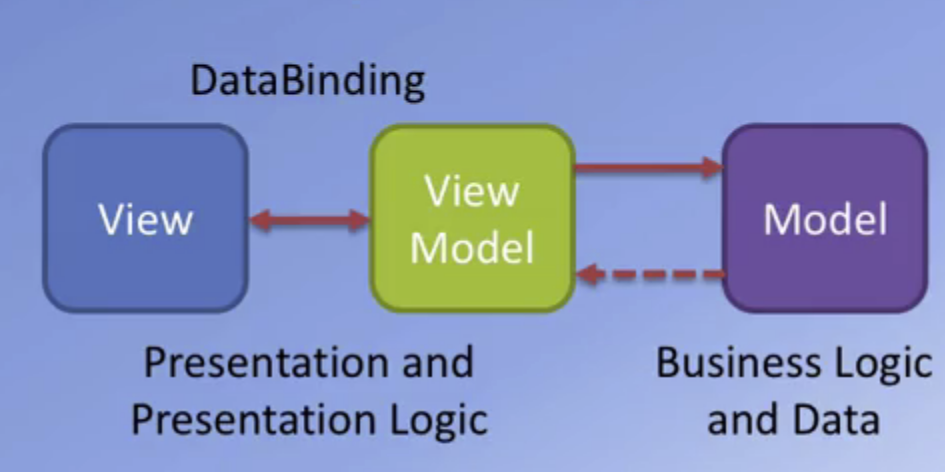
***Introduction to AngularJS***

* First released in 2012 by a Google employee Misko Hevery
* Structural framework for dynamic web applications
  + HTML only display static documents, but Angular fills in the gap to make the HTML dynamic
  + Angular works well with CRUD (date-driven) applications
    - CRUD: Create, read, update, and delete
* Important Angular Vocabulary
  + Two-way Data Binding
  + Scope
  + Directives
  + Templates
  + Routing
  + Testing
  + Modules
  + Controllers
  + Filters
  + Factory
  + Service
  + Provider
* Angular Built-in Directive
  + HTML custom attributes
    - Data-\* attributes (Bootstrap/jQuery)
    - ng-\* attributes/ data-ng-\* (for **Angular**)
  + Examples:
    - ng-app, ng-bind, ng-model, ng-init, ng-repeat, etc
    - ngApp, ngBind, ngModel, ngInit, ngRepeat, etc
      * Camel case converted to “–” (ngApp same as ng-app)
  + Directives are **declarative** programming in action
    - ng-app – Start up the Angular app
    - ng-init – Ask angular to initialize some variable or execute an expression
* The ngApp directive
  + It is applied to a HTML tag to specify the root of the application.
  + Applying ng-app to <html> tag means the entire page is under the control of the Angular application
* The ngInit directive
  + Evaluate an expression
  + Initialize a JS variable
    - <p ng-init= “index=1”></p>
    - <div class=”row” ng-init=“dish= {name:‘Test’, …}”></div>
* The ngModel Direction
  + Binds the input value to a variable within the scope (Very useful in input boxes)
    - Two-way data binding
* The ngRepeat Directive
  + This directive is a looping construct
  + Loops over items in a collection
  + Instantiates a template for each item
  + Example:
    - <ul>

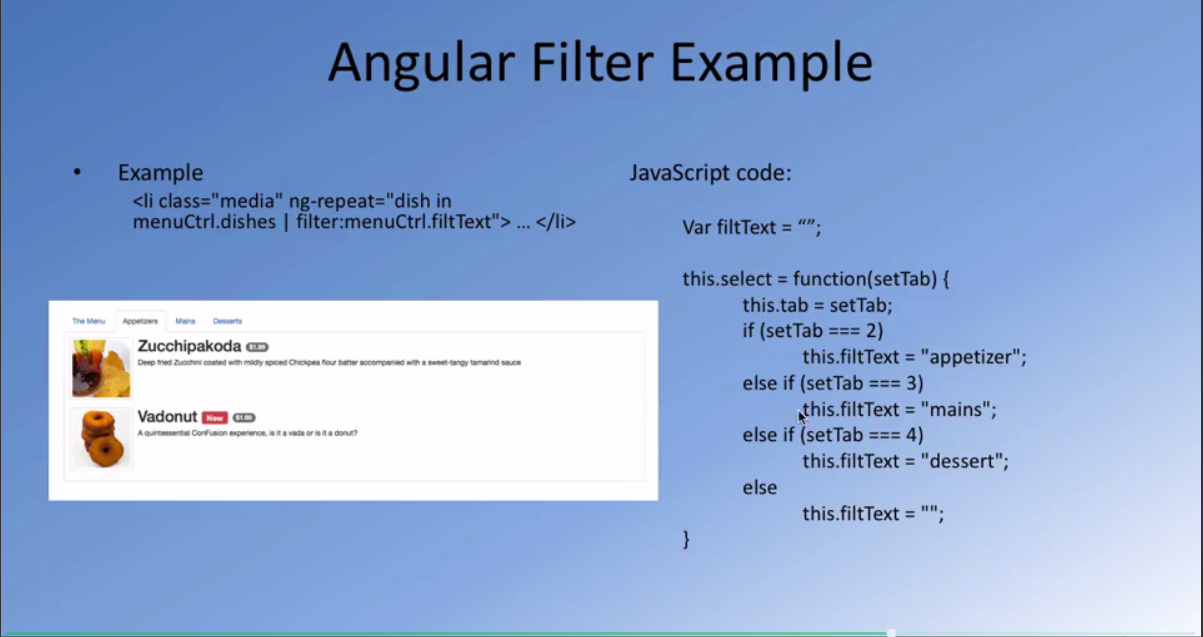
<li ng-repeat=“dish in dishes”>

* Two-way data binding
  + Bind a HTML or CSS property to a JS variable
  + When the value of the variable is updated the HTML/CSS property is also updated and vise versa
* Angular Expressions
  + Simple JS Expressions
    - Evaluated against against an Angular **scope** object
    - Expressions enclosed in {{ }}
  + Example: Expression with **scope** object
    - {{dish.name}}, {{dish.description}}

***Models, Views, and Controllers (MVC)***

* Software design pattern
  + Reusable solution to commonly occurring problems
* MVC is a software engineering architecture pattern
  + It isolates the domain logic from UI
  + Separation of Concerns
    - Independent, testing, and maintenance of the different parts of the app
* View: Presenting the info to user
  + Renders the model into a form suitable for interaction
  + Multiple views can exist for a single model for different purposes
  + A viewport typically has a one to one correspondence with the display surface
* Model: Stores the domain state/logic of the app
  + Manages the behavior and data of the app
  + Responds of the requests for info about it’s state (typically when view wants to update itself)
  + Responds to instructions to the change state (usually issued by the controller
  + Notifies all the observers (Views) for the model. When the model is updated, view is automatically notified about the changes in the model.
* Controller: Mediates between view and model
  + Receives information from users (through user input etc) and initiates a change in the state of the model
  + Accepts input from the user and instructs the model. Simultaneously, instruct the view to change the display of the info on the view.
* Model View-View Model (MVVM)
  + It is the descendent of MVC
  + 
* Angular Modules
  + Structure you code for readability of a big Angular project. **Angular module** comes in handy for such cases
  + Angular module is a collection of:
    - Controllers
    - Directives
    - Filters
    - Services
    - Other Configuration Information
  + It help keep code organized, maintainable, and easily testable
  + Angular Module Example
    - <html app=“angularAPP”>
    - In JS
      * var app=angular.module(‘angularAPP’, []);
* Angular Controller
  + JS object containing attributes/properties and functions to be used in angular expressions and directives
  + Controller example
    - <div ng-controller=“menuController as menuCtrl”>
    - In JS
      * app.controller(‘menuController’, function() {});
      * app is coming from defined in angular module

***Filters***

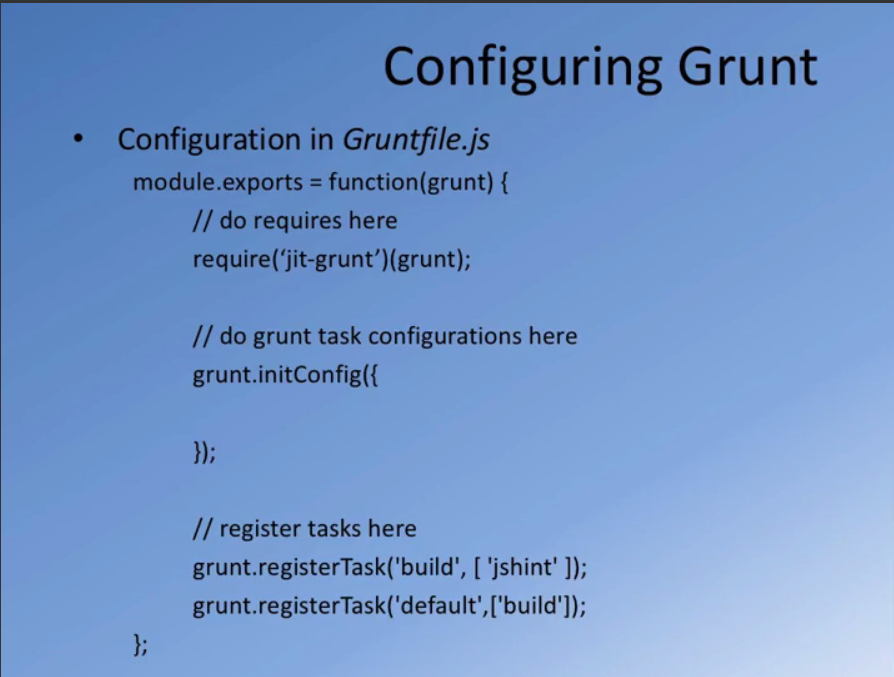
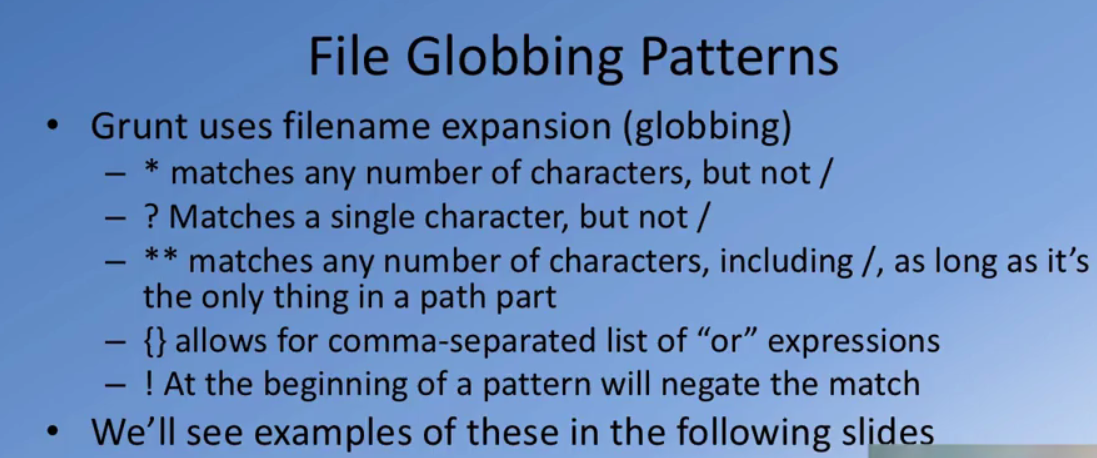
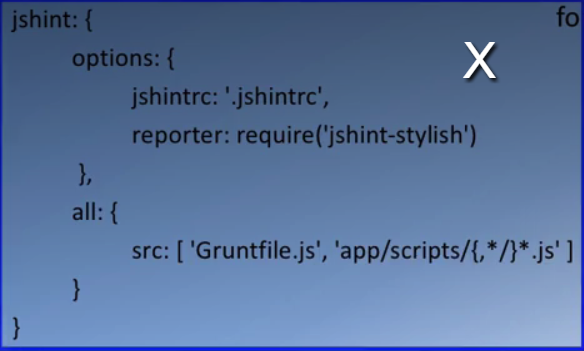
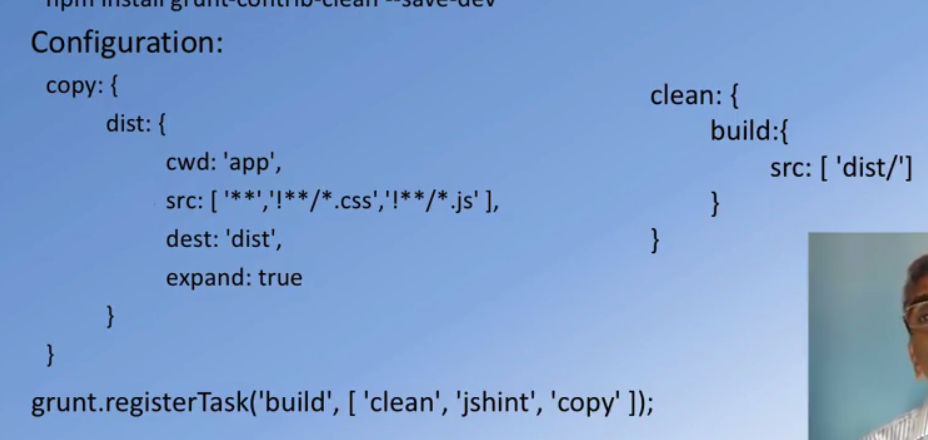
* It allow us to format the value of an expression before it is displayed to the end user. It can be used in view ***templates***, ***controllers***, or ***services***
* Use of filter in template
  + In an expression
    - {{ dish.price | currency }}
    - | tells the user to apply currency filter
  + With js code
    - 
* Built in Angular Filters
  + uppercase/lowercase
  + currency
  + date
  + filter
    - selects a subset of an array based on criteria specified and returns a new array
  + orderBy
  + json
  + limitTo

**Week 2:**

***Task Runners: Grunt and Gulp***

* CSS Tasks
  + Compiling SASS or LESS into CSS
  + Running Autoprefixer to add any vendor prefixes
  + Minification
    - Remove unnecessary characters from source code without compromising functionality
  + Concatenation
* JS Tasks
  + JSHint
    - Checking JS code for errors and/or potential problems (static code analysis)
  + Concatenation
  + Uglification
    - Minification + mangling (reduce local varables to single letters)
  + Rechecking for errors
* Other Tasks
  + ***Images***: optimizing files to reduce file size
  + ***Watch***: watching for changes in files and automatically rerunning tasks
  + ***Server*** and ***Livereload***
  + ***Testing***
* Grunt
  + Configuration based task runner
* Gulp
  + More code based task runner.
  + Write code to configure the task

***Grunt***

* Task runner based on configuration of tasks
* Installing Grunt
  + npm install –g grunt-cli
    - -g allows you to install it globally
  + Install grunt locally
    - npm install grunt –save-dev
      * need to setup package.json file
* 
* 
* JSHint
  + Allows us to find JS errors or obvious mistakes in our code or any sort of syntax errors.
  + Install
    - npm install grunt-contrib-jshint –save-dev
    - npm install jshint-styling –save-dev
  + Configuration
    - 
    - .jshintrc
      * json file contains configuration for JSHint
      * This is where we specify what JSHint should be checking
    - jshint-stylish
      * we want JSHint to style the output in more readable manner
    - object “all”
      * contains the source property that specify all the js files that we’re going to ask JSHint to check
      * /{,\*/}\*
        + It will take all js files in the subdirectory of app.
* Distribution Folder
  + We might want to create a distribution folder that contains only files essential to serve up the website
  + It contains:
    - Combined JS files
    - Combined CSS files
    - Images
    - Fonts
    - All essential files
  + Copy and Clean up
    - Creating a distribution folder and clean it up
      * npm install grunt-contrib-copy --save-dev
      * npm install grunt-contrib-clean --save-dev
    - Configuration
      * 
      * Copy task will copy over all the from app folder into the dist folder
        + ***cwd***: from the ***current work directory*** “app”
        + ***src***: source files in app folder except for css and js files
        + ***dest***: destination folder “dist”
      * Clean task will clean the dist folder and will rebuild it
        + ***src***: it will clean the dist folder since it will be recreated again
      * We want to run these tasks in a sequence using grunt configuration “***grunt.registerTask***” and specify it as “***build***”
        + Run clean ***first***, then ***jshint***, finally ***copy***
  + Completely preparing the ***distribution folder*** requires prepare all css code and js code that will be used for our HTML pages
    - Grunt modules
      * npm install grunt-contrib-concat --save-dev
      * npm install grunt-contrib-cssmin --save-dev
      * npm install grunt-contrib-uglify --save-dev
      * npm install grunt-filerev --save-dev
      * npm install grunt-usemin --save-dev
        + Umbrella task that configures and completes most of the CSS and JS minification and uglification tasks
        + It uses the help of all other grunt modules (concat, cssmin, uglify, filerev) in order to accomplish it’s task
        + Flow of Tasks:

useminPrepare 🡪 concat 🡪 cssmin 🡪 filerev 🡪 usemin

***useminPrepare*** prepares set of files to be used for remaining tasks

***concat*** with concatenate all the css files or js files depending on whatever we’re working on

***cssmin*** will minify ass css files/ JS task will uglification and minification task.

***filerev***: file revision

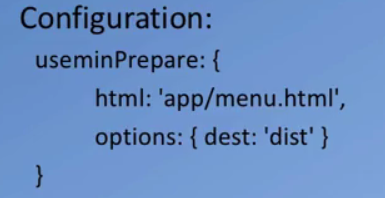
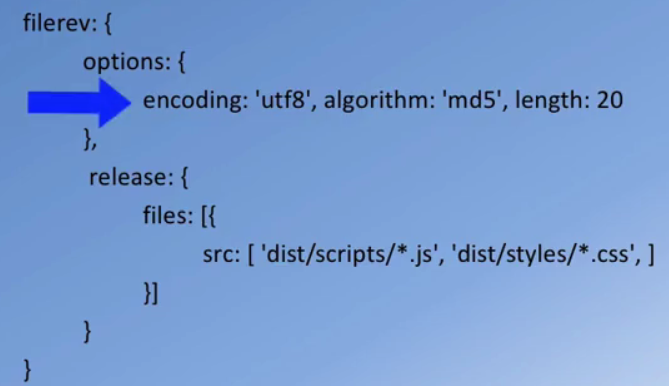
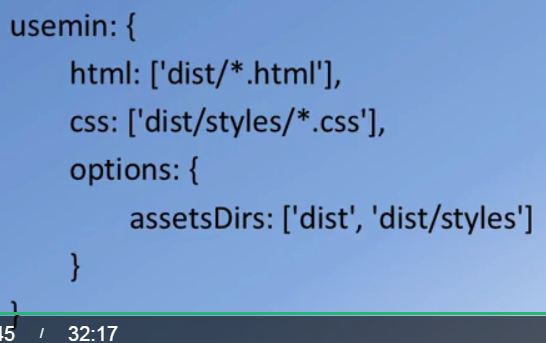
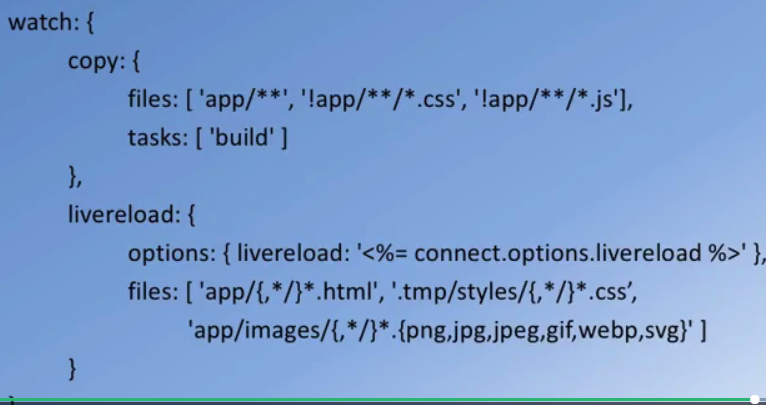
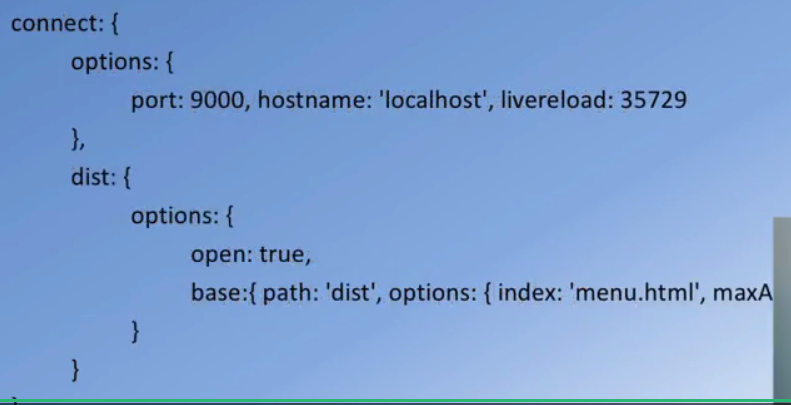
It attaches an additional division number to the combined CSS and JS files

It will ensure that user will served up with newer CSS and JS files in case user has cached the previous CSS and JS files

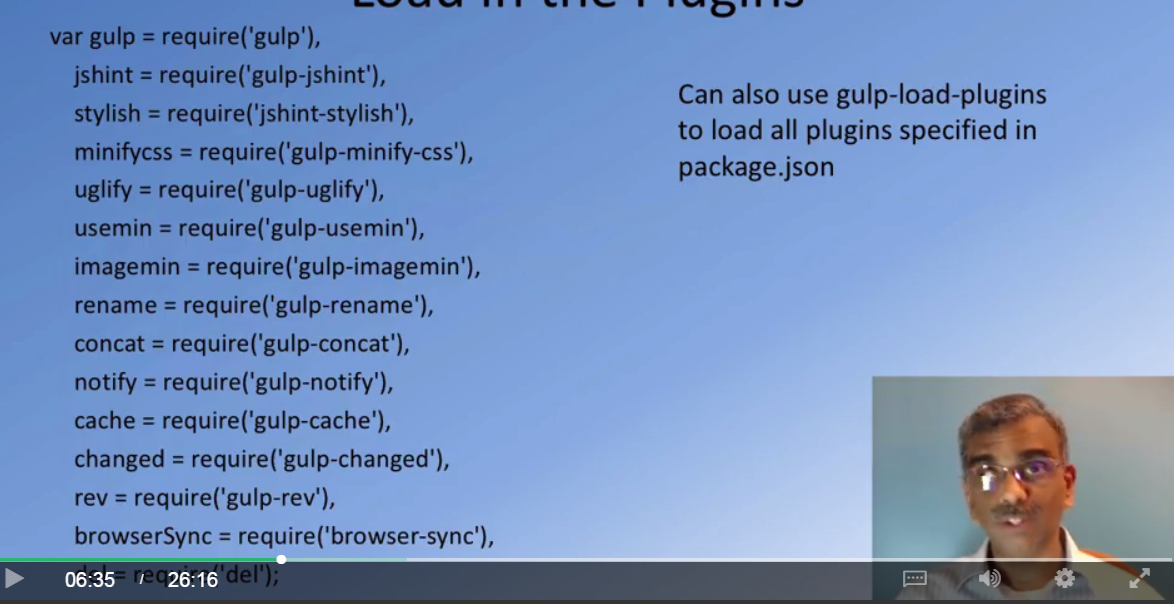
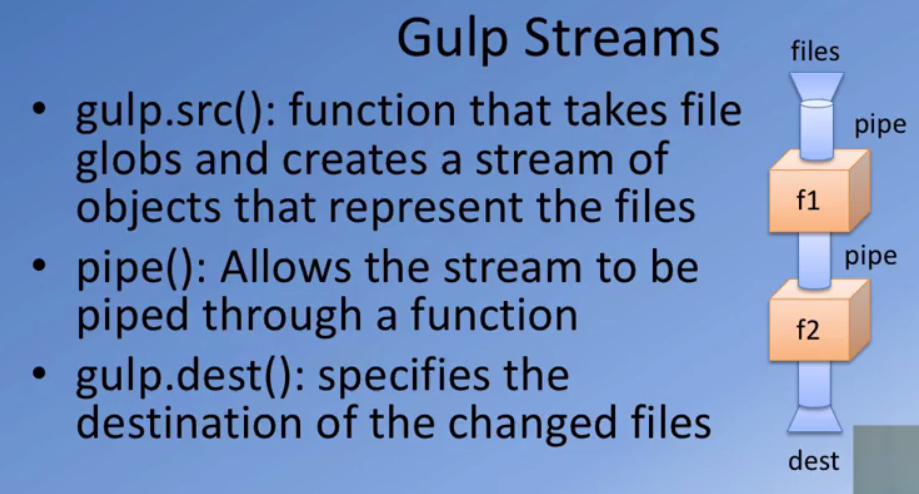
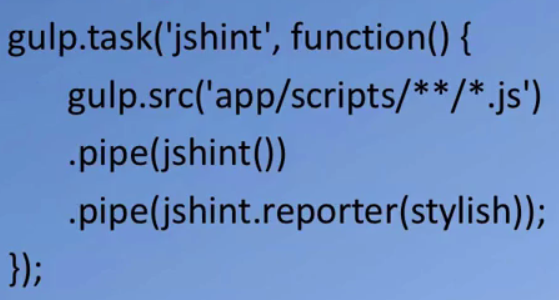
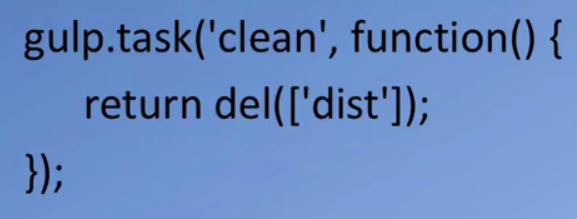
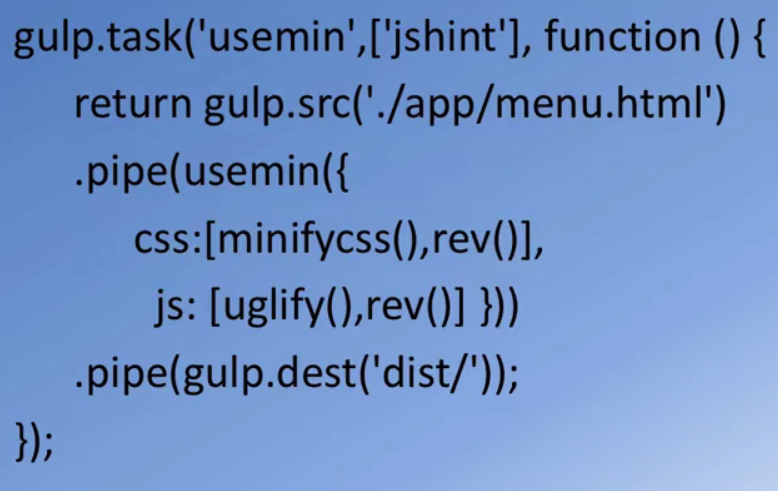
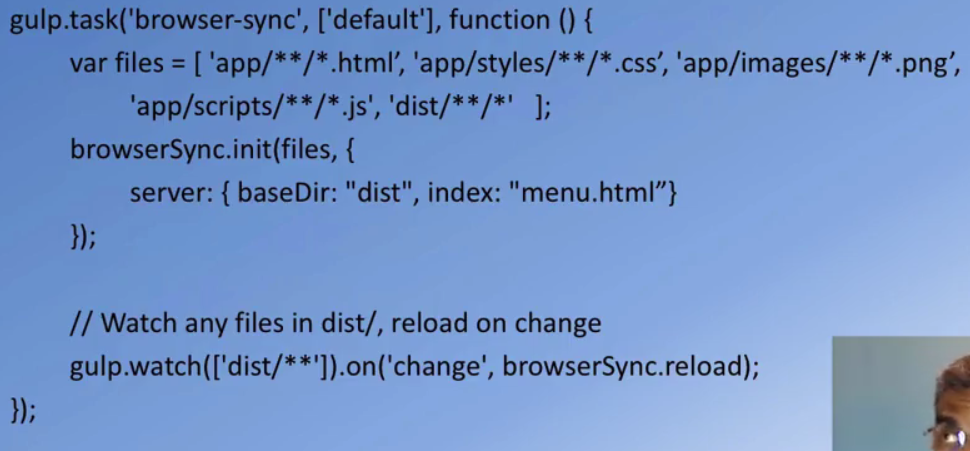
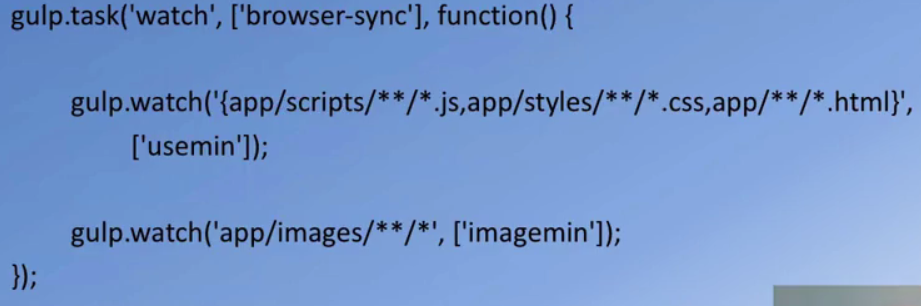
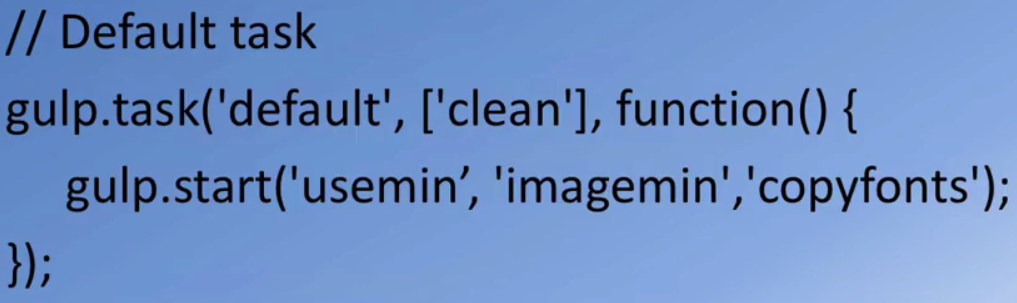
***usemin*** will configure the HTML files so that they will use the concatenated CSS and JS files

* + - useminPrepare
      * Looks for block configuration in an HTML file
        + <!--build:css styles/main.css --> … <!--endbuild -->

Whatever is in these in comment useminPrepare will treat those files as CSS files that needs to be concatenated together to prepare one CSS file

* + - * Configuration
        + 
    - fileRev
      * Revision your file: adds revision tag to the name of your file:
        + Ex: main.css 🡪 main.23758735.css
      * Configuration
        + 
        + Pointed arrow above will generate 20 random characters
        + ***release*** ensures that files that are in dist folder will be tagged with revision numbers
    - usemin
      * After concat, cssmin, uglify, and filerev are done. Usemin will replace the css and JS links with the single concatenated files from the dest folder
      * 
      * Configuration specifies when html, css, and js files are
      * Usemin will prepare the final dist folder and replace the information for all css/js files with single css/js file
* **Watch** task
  + Keeps a watch on files and reruns task whenever the changes occur
  + Install
    - npm install grunt-contrib-watch --save-dev
  + Configuration
    - 
    - It watches all files in app folder except css and js files (For css files we need to rerun the usemin task.
    - Then it runs the ***build*** task
    - ***livereload*** module
      * It will reload the app in web browser
* **Connect** module
  + Start a server to serve the website
    - npm install grunt-contrib-connect --save-dev
  + 
  + menu.html will be served on **port: 9000** and **hostname: ‘localhost’**
  + **path** is “dist” folder and webpage will serve **menu.html** page
* Register multiple grunt tasks
  + grunt.registertask(‘build’, [‘clean’, ‘jshint’, ‘useminPrepare’, ‘concat’, ‘cssmin’, ‘uglify’, ‘copy’, ‘filerev’, ‘usemin’]);
  + grunt.registertask(‘serve’, [‘build’, ‘connect:dist’, ‘watch’]);
    - This will start the webserver using the connect module.
    - Watch module will ensure that connect module doesn’t exit after serving up the page. It will make sure that server is continuously running.
  + grunt.registertask(‘default’, [‘build’]);
* time-grunt module
  + Keep track of each of the task and then generate status ticks at the end of grunt line.
* git-grunt
  + It will look inside node\_modules folder and grant modules as they are required by ***Gruntfile.js***
  + It will take care of loading all the grunt module as and when required.

***Gulp***

* Uses node.js streams to build complex pipelines, avoiding creation of intermediate files
* Plugins are simple and do one thing well
* Tasks are executed with maximum concurrency possible
* Install Gulp
  + Globally
    - npm install –g gulp
  + Locally
    - npm install gulp --save-dev
    - need to setup package.json file
* Gulp Plugins
  + npm install
    - gulp-jshint jshint-stylish
    - gulp-imagemin
    - gulp-concat
    - gulp-uglify
    - gulp-minify-css
    - gulp-usemin
    - gulp-cache
    - gulp-rev
    - gulp-rename
    - gulp-notify
    - gulp-browser-snyc del
  + --save-dev
* Load in the Plugins
  + 
* Gulp Streams
  + 
* Some of Gulp Tasks
  + JSHint
    - 
    - gulp.src will grab all the js files which then it will all be piped through jshint function and the reported with styled version
  + Imagemin
    - 
    - It takes all images in image folder and optimize them to be used in webpage
    - **.src** will grab all the image files then pipe through
      * cache
        + Cache can hold all the files and then passed through **imagemin** function which does the image transformation
      * dest
        + It will send the images to dist folder
      * notify
        + It will give the message to the user that this task has been completed (usually in terminal screen)
  + Clean UP
    - 
    - **del** is a node module
      * It is used to delete the dist folder
  + usemin
    - 
    - **usemin** task always run “**jshint**” before running itself
    - **rev():** it is revision plugin for gulp
    - when src is pipe through usemin function
      * It minify and revision the css
      * It uglify and revision the js
    - Then it pipe through **dest** to copy files to dist folder
  + browserSync
    - 
    - Bowser-sync is a node module not gulp plugin so it can also be used with Grunt (instead of connect), vice versa gulp also has connect package
    - Bowser sync is more flexible/powerful than connect.
    - In ***browser-sync*** task “***default***” (as specified above before function) task will run first (which will build the dist folder)
    - ***var files***: files is a variable that will specify an array of files that browser-sync will keep its watch on
    - ***browserSync***.init
      * first parameter is an ***array of files*** that watch will occur on
      * Second parameter provides information of server configuration
        + ***dist*** is the base directory
        + ***menu.html*** is the file that it will be serving since we don’t have ***index.html*** file
    - **gulp.watch()**
      * watch is a built-in task in gulp
        + It keeps a watch on files specified there
      * .on(‘change’, browserSync.reload);
        + If there is any change in specified files then the browserSyns will make the browser reload
  + Watch
    - **watch** task keeps a watch on the files
    - 
  + Default Task
    - 
    - Before the default task is executed, clean task will be executed
    - gulp.start()
      * we execute all the task that needs to executed in this function