The first tool we use in JavaScript is Babel to convert cutting edge JavaScript to ES6/ESNext back to ES5 so all browser will be able to run our JavaScript code. In a couple of years this may not be necessary, but for now it's an important step in our development process.  
Next we want to use webpack in order to make our ES6 code more modular and therefore easier to maintain by separating our app into different files. Most browsers out there, do not support this functionality, so we must bundle these files together and the most popular tool is webpack. Webpack can also do much more like code splitting, loading many types of assets like sap and images, decreasing our JavaScript bundle size using an  
algorithm like three-shaking and much more. The easiest way to run these packages is by using NPM scripts.

The following are all command line entries.

# **Node.js**

* First install the latest release of node.js to be found on the internet
* This will also install the NPM package.
* These are run from your terminal they can also be run from the terminal within Intellij Idea.
* Check its version of node.js
* node -v

# **NPM commands**

* These are run from your terminal they can also be run from the terminal within Intellij Idea.
  + Check the version of npm:
    - npm -v
* **Now ensure you are in your working directory.**
* **We now to create a package.json file for our project in our working directory.**
  + npm init
    - It will start to ask us some question during the package.json creation.
      * Reply to the questions or accept the default.
      * Finally, it will ask if it is ok
        + Is This ok? (yes), enter yes, if you say no it will not build the file.
* You can modify the file package.json within Intellij Idea or any other json editor such as VSCode, Brackets or Sublime Text.
* **Local installation of packages for the current project:**
  + To install webpack and webpack-cli development tools

Npm install webpack --save-dev & npm install webpack-cli --save-dev

* + - -dev on the end put the package in the development dependency list
    - To uninstall it and remove it from the devtools dependency list.
      * Npm uninstall webpack --save-dev
  + To install jquery dependency, this is not a dev dependency

* + npm install jquery --save (Notice the save part is different.)
    - We will not be using this product so uninstall it.
    - This will be listed in the dependencies within our package.json file.
      * To uninstall it:
        + npm uninstall jquery –save
  + Our package contains a folder called : ‘node\_modules library root’.
  + This folder contains all the packages from the dev-dependencies and dependencies list within our package.json file.
  + This file is very large, and we do not need to share it with others when we generate our project. All that you need to do to recreate the folder:
  + npm install
    - This will re-create the folder from the dependency lists in the package.json file.

* **Global installation of packages to make them accessible from anywhere on your PC:**
  + To install live-server globally (Create a local server for testing)
    - npm install live-server --global
      * These modules are stored within the folder:
        + c:/users/Melvyn/appdata/roaming/npm/node\_modules

To run live-server you just type live-server on the command line and Ctrl+c to quit.

# **COURSE - TOOLS UPDATE BEFORE PROGRESSING**

* + - * + Modern tools like Webpack and Babel change all the time, which is great for the development community, but difficult for course creators.
        + So please understand that I cannot re-record these videos each time a new change is introduced to one of the tools. Instead, **I will keep this lecture updated with instructions to make your tooling setup work as it should.**
        + There are two ways of approaching this: you either just want tools to work like in the videos, or you want to use the latest versions of the tools. Choose the one that works best 😉
        + **If you just want it to work, follow this (RECOMMENDED) 👇**
        + Tools do not add that much functionality in new versions, so you are just fine using this method.
        + You will have to install the same package versions that I install in my videos. So instead of the npm install commands I use in the videos, use the following (we use @ to specify the version number)
        + **For webpack:**
        + npm install --save-dev webpack@4 webpack-cli@2 webpack-dev-server@3
        + **For babel:**
        + npm install --save-dev babel-core@6 babel-preset-env@1 babel-loader@7
        + npm install --save babel-polyfill@6
        + **If you want the latest versions, follow this 👇**
        + This method requires a little more work, as babel has recently changed how the configuration works.
        + For webpack, just follow the videos.
        + For babel, **instead** of installing babel-core, babel-preset-env and babel-polyfill in the babel lecture, please install @babel/core, @babel/preset-env, core-js@3, and regenerator-runtime like this:
        + npm install --save-dev @babel/core @babel/preset-env babel-loader npm install --save core-js@3 regenerator-runtime
        + You will also need to change the entry in webpack.config.js (a file we create during the video) from this:
        + entry: ['babel-polyfill', './src/js/index.js'],
        + to this:
        + entry: ['./src/js/index.js'],
        + and the code in .babelrc (also created during the video) from this:
        + {
        + "presets": [
        + ["env", {
        + "targets": {
        + "browsers": [
        + "last 5 versions",
        + "ie >= 8"
        + ]
        + }
        + }]
        + ]
        + }
        + to this:
        + {
        + "presets": [
        + ["@babel/env", {
        + "useBuiltIns": "usage",
        + "corejs": "3",
        + "targets": {
        + "browsers": [
        + "last 5 versions",
        + "ie >= 8"
        + ]
        + }
        + }]
        + ]
        + }

# **Configure Webpack**

* Within your folder create a file called ‘webpack.config.js’
  + const path = require(’path’);
  + modules.exports = {
    - Entry Point, this is where webpack will start the bundling, here we can specify one or more entry files.
      * Entry: ‘./src/js/index.js,
    - Output property tells web app where to put the bundled file. Into here we pass an object.
      * output: {
        + path: path.resolve(\_\_dirname,’dist/js’),
        + filename: ‘bundle.js’
      * },

* + - We now have something called the Production and Development mode, the development mode simply creates our bundle without minifying our code whereas the production mode compresses and minifies the code and other types of optimization.
      * mode: ‘development’

Development mode is faster because it does not do any of the compressions etc required for production.

* Loaders and plugins will be discussed later in the course

}

# **Test Setup**

To test what we have done so far:

Created a test script file ‘test.js’:

console.log(Imported module);

export default 23;

Entered the following in our main script file ‘ index.js’:

import num from ‘./test’;

console.log(`I imported ${num} from another module!`);

If we did not use webpack the above would not work.

Next we need to add an ‘npm’ script to the ‘package.json’ file:

Remove the script the script; section with;

“scripts”: {

“dev”: “webpack”

},

Webpack reads the ‘webpack.config.js’ file and does its work as per the script in the file, first reading the entry file ‘./src/js/index.js’ and create a bundle file called ‘bundle.js’.

We also need to ensure the webpack command line interface dependency is installed. This was done previously:

npm install webpack-cli --save-dev

To launch our script from the terminal command line:

npm run dev

To open an html file from the command prompt (mac = **open**, Windows 10 = **start)**

**start** dist/index.html