What’s this npm thing anyways?

npm, commonly referred to as “node package manager” is just that. It’s a package manager for node. Although, npm does *not* stand for “node package manager”. I won’t go into the details of that, but if you’d like to, you can read all about it here.

npm’s purpose is to install and maintain specific versions of JavaScript packages without the old copy and paste that we used to have to do back in the “old days”.

Dependencies

Application requirements are defined in a file in the root directory of your application, called `package.json`.

```
{
  "name": "My Cool App",
  "version": "1.0.0",
  "description": "The most awesome nebulous app ever made",
  "author": "Tony Stark <foo@bar.com> (https://github.com/i-am-fictitious)",
  "license": "ISC",
  "devDependencies": {
    "react-hot-loader": "1.3.0",
    "webpack": "1.12.6",
    "webpack-dev-server": "1.12.1"
  },
  "dependencies": {
    "babel-cli": "6.2.0",
    "babel-core": "6.1.21",
    "babel-loader": "6.2.0",
    "babel-polyfill": "^6.2.0",
    "babel-preset-es2015": "6.1.18",
    "babel-preset-react": "6.1.18",
    "babel-preset-stage-0": "6.1.18",
    "babel-runtime": "6.1.18",
    "css-loader": "^0.23.0",
    "express": "4.13.3",
    "extract-text-webpack-plugin": "^0.9.1",
    "history": "~1.13.1",
    "node-sass": "^3.4.2",
    "normalize.css": "^3.0.3",
    "react": ">=0.14.2",
    "react-dom": ">=0.14.2",
    "sass-loader": "^3.1.2",
    "style-loader": "^0.13.0"
  }
}
```

There are two major sections where we define our dependencies: `dependencies` and `devDependencies`. The `dependencies` section defines everything that our app needs during *run time*, while the `devDependencies` section defines all of the modules we need during *development*. It also includes some ancillary information that is needed if we are going to push our package up to npm.

Semantic Versioning

npm dictates the use of semantic versioning in all modules to allow module authors to define how much their module has changed since its last release. We can declare what module updates, if any, we want to allow while we're building our app. We do this by using `comparators` – an `operator` and a `version` number – in `package.json`.

Ranges

We'll start with the primitive operators, eg: >=. It means "install the newest version of this package that has at least this version number". Other primitive operators include >, <, =, and <=. They all do exactly what you think they do, but do not take advantage of the granularity provided by semantic versioning.

Tilde Ranges

Let's take a look at the `history` entry in our `package.json` file:

```
"history": "~1.13.1"
```

This is called a tilde range comparator, and is equivalent to "at least `1.13.1`, but less than `1.14.0`". If we had omitted the `PATCH` version and only asked for `~1.13`, npm would have installed the latest version beneath `1.14`, but would not update `1.13.0` if it is already installed locally.

By extension, if we had merely declared `"history": "~1"`, npm would install any version beneath `2.0.0`, but would not fetch a newer version if `1.0.0` or newer was already installed.

Caret Ranges

Next we'll take a quick peek at the caret ranges that are seen in our `package.json` file:

```
"sass-loader": "^3.1.2"
```

Allows changes that do not modify the left-most non-zero digit in the [major, minor, patch] tuple. In other words, this allows patch and minor updates for versions 1.0.0 and above, patch updates for versions 0.X >=0.1.0, and no updates for versions 0.0.X.

Essentially, this means "give me the newest version of this package that doesn't have a breaking change". Typically when a package author using semver with a `0.x.x` `MAJOR` version makes a relatively minor, but breaking change to their module, they will bump the minor version. So a `0.1.x` package will become `0.2.0`. Let's say that we had a package that was
at version `0.1.3` and we were using caret ranges. Our `package.json` might look like this:

```
"sass-loader": "^0.1"
```

During installation npm finds two newer version of this package: `0.1.4` and `0.2.0`. Because we specified a carat range, npm will select the `0.1.4` version.

That's all we are going to cover in this section in regards to npm and semantic version, but there is a lot more information about this on the npm semver page if you'd like to dive into it a little deeper.