Packages and npm

The Node Package Manager

- We've seen ***npm*** (the Node Package Manager) briefly before, when we used it to install a third-party package.

- More generally, npm provides a way to work with ***packages***, which are bundled-up pieces of shared, reusable code.
    - Really, a package is just a directory with one or more files in it, one of which is called `package.json`, which contains some metadata about the package.

- npm itself consists of:
    - A package registry, which is a big database with info about all publicly-shared packages.
    - A command-line tool that allows developers to work with and share packages.

Installing packages locally with npm

- One of the main uses of npm is to install publicly-shared packages to use in your own code. For example, if we wanted to install the package `lodash`, we could do this:

```
npm install lodash
```

    - This would create a directory named `node_modules` within the current directory and install the `lodash` package into that directory.

    - We could then use `lodash` in our code by `require()`'ing it:

```
var lodash = require('lodash');
console.log(lodash.without([1, 2, 3], 1));
```

Building your own package with `package.json`

- It can be useful to treat your own piece of code as a package, even if you don't intend to share it with others. Doing so provides many benefits:
    - It allows you to specify your dependencies (with versions) so that they're very easy to install again.
    - It allows you to specify commands to perform common tasks associated with your code, e.g.:
        - Build code
        - Run setup steps
        - Run code (production or development mode)
        - Run tests
        - Etc.
    - Make note of important info about your code, e.g. info about you, the author, info about the code's version control repository, info about a website associated with the code, etc.

- In general, though, packaging your code makes it more reusable and easier to share with other developers.

- To make your code into a package, you'd just need to put a `package.json` file in place.
    - As the extension implies, the file is written in JSON syntax.
        - https://en.wikipedia.org/wiki/JSON

- The `package.json` file specifies metadata about your package. At a minimum, it contains a name and a version for your package:

```
{
      "name": "my-package",
      "version": "1.0.0"
}
```

- You can create a `package.json` file using npm's `init` command:

```
npm init
```

    - This will walk you through several questions about your package and use your answers to construct a `package.json`.

Specifying dependencies in `package.json`

- There are two fields you can use in `package.json` to specify dependencies:
  - `dependencies` – packages specified here are ones that are required by your application to run in production.
  - `devDependencies` – packages specified here are ones that are only needed for development and testing.

- You can manually specify packages in either `dependencies` or `devDependencies`. Each entry in either of these fields is a `"name": "version"` pair:

```
"dependency-name": "^4.0"
```

  - The version is specified as a ***semver*** expression.
    - https://docs.npmjs.com/getting-started/semantic-versioning

- Here's an example of a package with dependencies:

```
{
     "name": "my-package",
     "version": "1.0.0",
     "dependencies": {
          "my-dependency": "^2.0.0"
     },
     "devDependencies": {
          "my-testing-dependency": "^3.2.0"
     }
}
```

- You can use the `--save` or `--save-dev` option in an `npm install` command to automatically save the installed package to one of `package.json`'s dependency fields:

```
npm install my-dependency --save
npm install my-testing-dependency --save-dev
```

- If you have a `package.json` file with dependencies specified, you can install all of them by using npm's `install` command without arguments from the directory where `package.json` lives:

```
npm install
```

Installing global packages

- Sometimes a package contains a binary executable that you want to be available globally on your system. You can install a package globally using the `-g` option to `npm install`:

  ```
  npm install -g eslint
  ```