Managing Packages with Npm - How to Use package.json, the Core of Any Node.js Project or npm Package

The Node Package Manager (npm) is a command-line tool used by developers to share and control modules (or packages) of JavaScript code written for use with Node.js.  
  
When starting a new project, npm generates a package.json file. This file lists the package dependencies for your project. Since npm packages are regularly updated, the package.json file allows you to set specific version numbers for each dependency. This ensures that updates to a package don't break your project.  
  
npm saves packages in a folder named node*modules. These packages can be installed in two ways:*

1. *globally in a root node*modules folder, accessible by all projects.
2. locally within a project's own node\_modules folder, accessible only to that project.

Most developers prefer to install packages local to each project to create a separation between the dependencies of different projects. Working on these challenges will involve you writing your code on Glitch on our starter project. After completing each challenge you can copy your public Glitch url (to the homepage of your app) into the challenge screen to test it! Optionally you may choose to write your project on another platform but it must be publicly visible for our testing.

VERSIONING

Knowing SemVer can be useful when you develop software that use external dependencies (which you almost always do). One day, your understanding of these numbers will save you from accidentally introducing breaking changes to your project without understanding why things “that worked yesterday” suddenly doesn’t.

This is how Semantic Versioning works according to the official website:

Given a version number MAJOR.MINOR.PATCH, increment the:

MAJOR version when you make incompatible API changes,

MINOR version when you add functionality in a backwards-compatible manner, and

PATCH version when you make backwards-compatible bug fixes.

This means that PATCHes are bug fixes and MINORs add new features but neither of them break what worked before. Finally, MAJORs add changes that won’t work with earlier versions.

Example

A semantic version number: 1.3.8

KEEP THE LATEST PATCH VERSION

In the last challenge, we told npm to only include a specific version of a package. That’s a useful way to freeze your dependencies if you need to make sure that different parts of your project stay compatible with each other. But in most use cases you don’t want to miss bug fixes, since they often include important security patches and (hopefully) don’t break things in doing so.

To allow a npm dependency to get updated to the latest PATCH-version, you can prefix the dependency’s version with the tilde-character (~). In package.json, our current rule for how npm may upgrade moment is to use a specific version only (2.10.2), but we want to allow the latest 2.10.x-version.

Example

"some-package-name": "~1.3.8" allows updates to any 1.3.x version.

UPDATE TO LATEST MINOR VERSION

Similar to how the tilde (~) we learned about in the last challenge allow npm to install the latest PATCH for a dependency, the caret (^) allows npm to install future updates as well. The difference is that the caret will allow both MINOR updates and PATCHes.

At the moment, your current version of moment should be ~2.10.2 which allows npm to install to the latest 2.10.x-version. If we instead were to use the caret (^) as our version prefix, npm would instead be allowed to update to any 2.x.x-version.

Example

"some-package-name": "^1.3.8" allows updates to any 1.x.x version.