Lesson - 11

Introduction to NPM.
CSS Pre-processors

SkillUp by Dobrea Vladislav

NPM

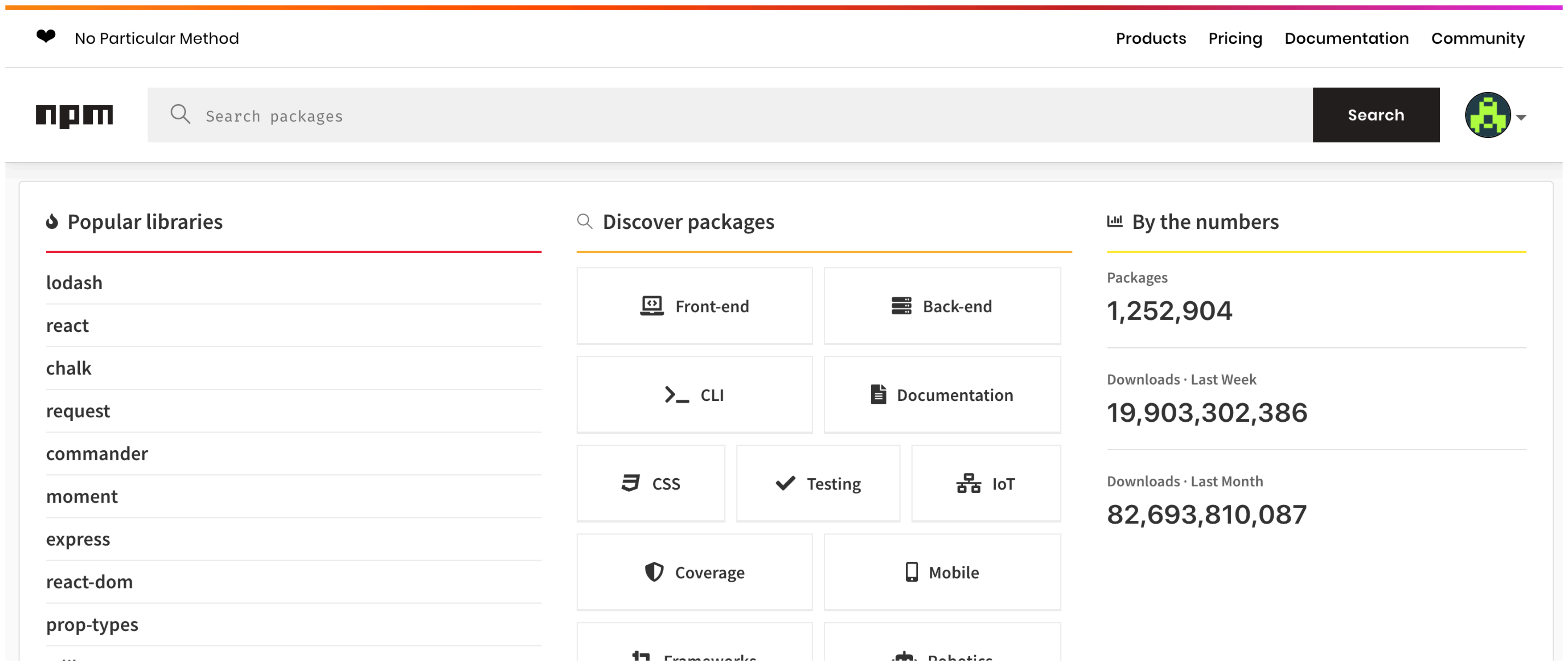
npm stands for node package manager. It allows for seamless node.js package management. You can install, share and manage node.js packages.

npm consists of three components:

- Website
- Registry
- CLI

NPM Website

npm official website is https://www.npmjs.com/. Using this website you can find packages, view documentation, share and publish packages.



Popular libraries

- lodash
- react
- chalk
- request
- commander
- moment
- express
- react-dom
- prop-types

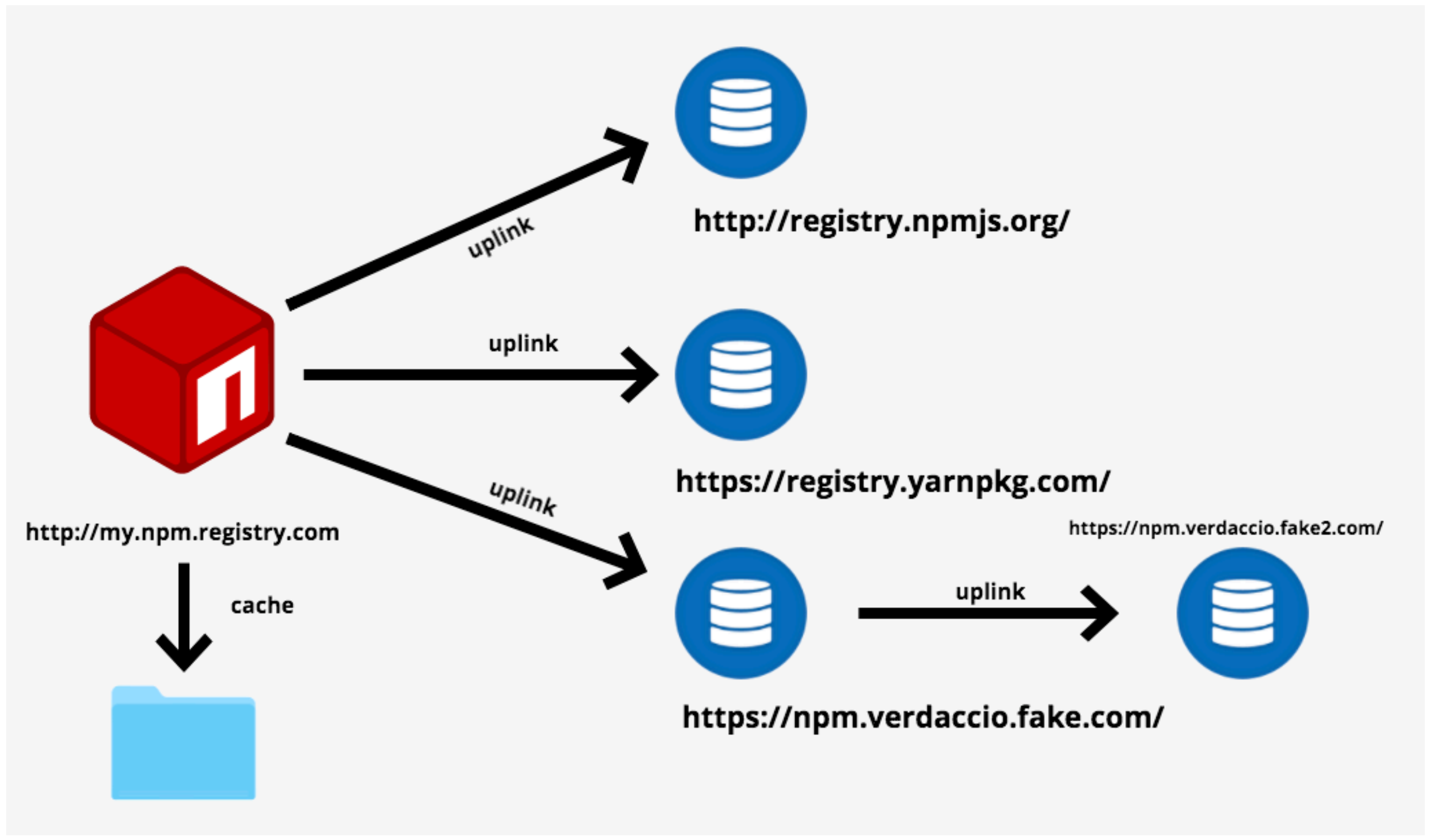
Discover packages

By the numbers

Packages	1,252,904
Downloads · Last Week	19,903,302,386
Downloads · Last Month	82,693,810,087

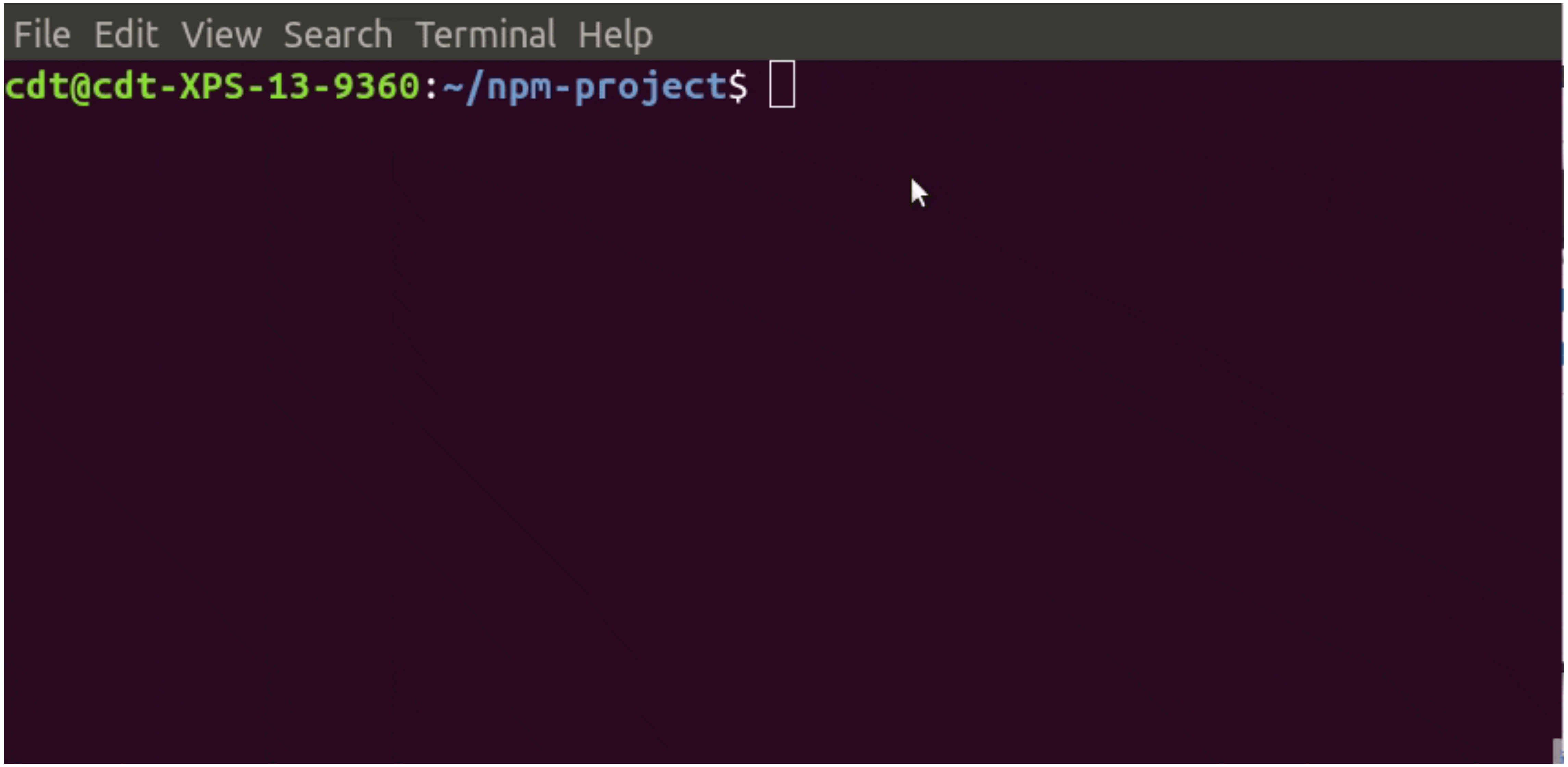
NPM Registry

npm registry is a large database consisting of more than half a million packages. Developers download packages from the npm registry and publish their packages to the registry.


http://registry.npmjs.org/
uplink
uplink
https://registry.yarnpkg.com/
uplink
http://my.npm.registry.com
cache
https://npm.verdaccio.fake2.com/
uplink
https://npm.verdaccio.fake.com/

NPM CLI(Command Line Interface)

The is the command line that helps in interacting with the npm for installing, updating and uninstalling packages and managing dependencies.



```
cdt@cdt-XPS-13-9360:~/npm-project$
```

Installing NPM

npm comes with the node.js. So, you don’t have to install it separately. You can install node.js from their official website https://nodejs.org/en/download/.

After installing node, You can check the version of node and npm by

```
node -v
npm -v
```

package.json

This file can contain a lot of meta-data about your project. But mostly it will be used for two things:

- Managing dependencies of your project
- Scripts, that helps in generating builds, running tests and other stuff in regards to your project

Creating package.json

Run the command npm init and it will ask you some information about your project and will create a basic package.json in that directory.
The initial content of that file would look something like this:

```
{
  "name": "npm-example",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

To quickly create a package.json file. You can use command

```
npm init -y
```

What's inside package.json?

Identifier Metadata

These metadata describes the project: what it's called, what is it's purpose, version and other things.

A package.json file must have a **name** and a **version** property. These two properties uniquely identifies your project in npm repository.

name

The name of the project/module that this package.json file is describing

version

Denotes the current version of your module.

Some other optional meta properties are **license**, **description** and **keywords**.

Functional Metadata

These properties help in coordinating with `npm` to perform certain tasks related to execution, build process and other stuff.

main
It describes the entry point to the application. When you run this application, or require it as a module in another application, the file described by this property will be included.

repository
It defines where the source for this application/module is located. Mostly it will be github in case of open source projects. Specify the type of repository it is: `git` or `svn` or anything else, followed by the url to the repository.

scripts
The scripts properties takes a list of key-value pairs. Each `key` specifies the name of the tasks the script will perform and the `value` will contain the command to invoke the task. The scripts can be invoked by `npm run <task-name>`. Scripts are mostly used for testing, building and defining some of the commands to work with your module/application.

dependencies
A list of modules that this project depends upon are defined here.

devDependencies
The modules that assist you in creating a build, testing your code, deploying your code and other development level modules.

Basic NPM Commands

Installing packages:

- **Locally:** A locally installed package can be accessed only on the folder you've downloaded it.

```
npm install <package_name>
```

The *node_modules* is the folder in which our local packages are installed.
There will be a new file named *package-lock.json*. This file contains the exact version of the package, unlike *package.json* which contains the semantic version(which we will be learning later).

You can also install packages as a developer dependency i.e., these packages are only needed for development. For example, they can be any package for testing the project. To install packages as a developer dependency use the command

```
npm install <package_name> --save-dev
```

Basic NPM Commands

Installing packages:

Globally: A globally installed packages works anywhere on the machine. To install global packages you've to use -g flag.

Generally, any packages you use in your project have to be installed locally. And packages you use in the command line are to be installed globally.
The command for the local and global packages are same except that you have to use -*g* flag for global packages.

Updating packages:
Since we have installed packages sometimes we need to update our packages to get new features. To do that, you've to use

```
npm update <package_name>
```

for a specific package (or) just

```
npm update
```

to update all packages.

Basic NPM Commands

Uninstalling packages:
Sometimes you don’t need a particular package and you want to remove it. It’s not a good idea to manually remove the package from the *node_modules* folder as it can be a dependency for the other packages. So, to safely remove a package you’ve to use the command

```
npm uninstall <package_name>
```

CSS Pre-processors

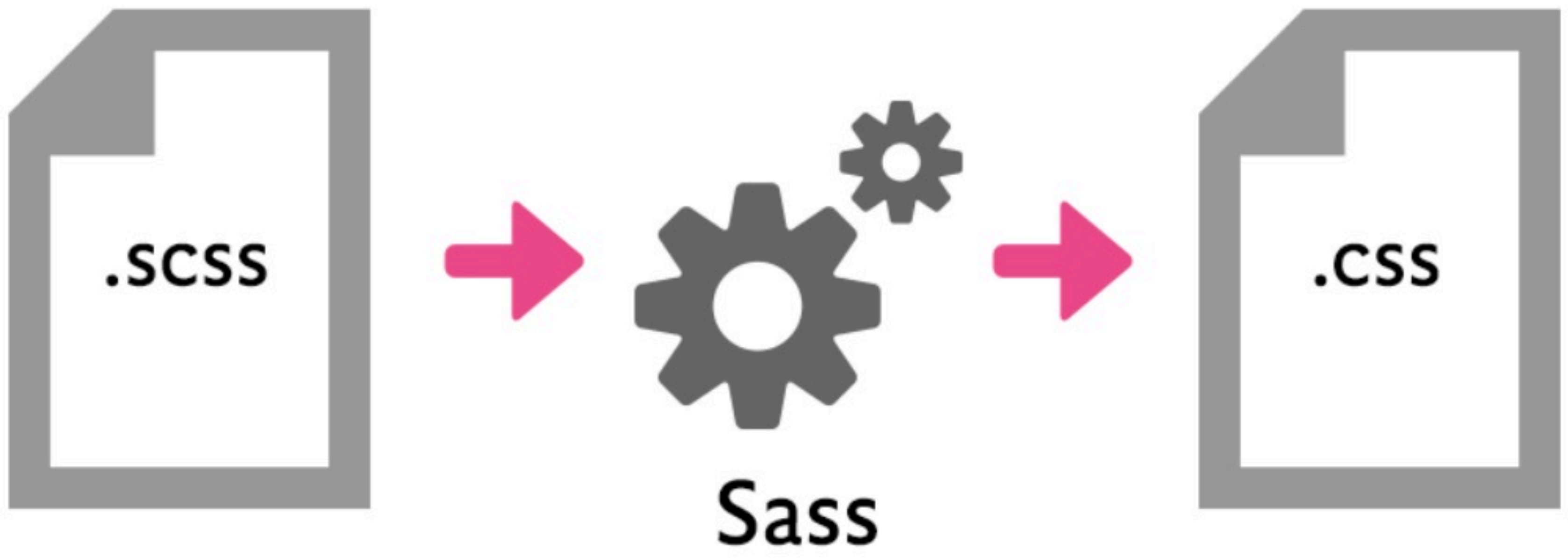
Sass

{less}

What is CSS pre-processor?

A **CSS preprocessor** is a program that lets you generate CSS from the preprocessor's own unique syntax. There are many CSS preprocessors to choose from, however most CSS preprocessors will add some features that don't exist in pure CSS, such as mixin, nesting selector, inheritance selector, and so on. These features make the CSS structure more readable and easier to maintain.

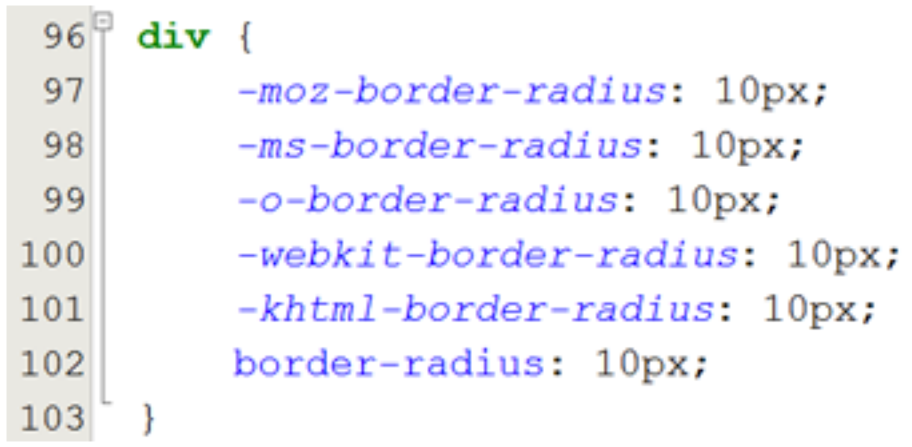
To use a CSS preprocessor, you must install a CSS compiler on your web server.



Why pre-processing CSS?

CSS is primitive and incomplete. Building a function, reusing a definition or inheritance are hard to achieve. For bigger projects, or complex systems, maintenance is a very big problem. On the other hand, web is evolving, new specs are being introduced to HTML as well as CSS. Browsers apply these specs while they are in proposal state with their special vendor prefixes. In some cases (as in background gradient), coding with vendor specific properties become a burden. You have to add all different vendor versions for a single result.

Pre-processors, with their advanced features, helped to achieve writing reusable, maintainable and extensible codes in CSS. By using a pre-processor, you can easily increase your productivity, and decrease the amount of code you are writing in a project.



```
div {
    -moz-border-radius: 10px;
    -ms-border-radius: 10px;
    -o-border-radius: 10px;
    -webkit-border-radius: 10px;
    -khtml-border-radius: 10px;
    border-radius: 10px;
}
```

Digging the features

Like every programming language, pre-processors have different syntax, but hopefully, not too separated. All of them support classic CSS coding and their syntax are like classic CSS. SASS and Stylus have additional styles. In SASS, you can omit curly brackets and semicolon, whereas in Stylus, you can also omit colons. These are optional in both SASS and Stylus.

Variables

Variables were all time wanted feature for CSS. Every developer, wanted to define a base color and use it all over the CSS file, in stead of writing the hex or named color in a property each time. Same as "color", variables needed for "width", "font-size", "font-family", "borders" etc.

Sass

```scss
$font-size: 16px;

div {
    font-size: $font-size;
}
```

Css

```css
div {
    font-size: 16px;
}
```

Nesting

CSS lacks visual hierarchy while working with child selectors. You have to write selectors and their combinations in separate lines. Nesting provides a visual hierarchy as in the HTML and increases the readability. In some cases, nesting causes oversizing the selectors and something like a "selector train", so use it wisely.

Sass

```scss
$link-color: #999;
$link-hover: #229ed3;

ul {
    margin: 0;

    li {
        float: left;
    }

    a {
        color: $link-color;

        &:hover {
            color: $link-hover;
        }
    }
}
```

Css

```css
ul { margin: 0; }
ul li { float: left; }
ul a { color: #999; }
ul a:hover { color: #229ed3; }
```

CSS Nesting Properties

Many CSS properties have the same prefix, like font-family, font-size and font-weight or text-align, text-transform and text-overflow.

With Sass you can write them as nested properties:

SASS

```
font: {
  family: Helvetica, sans-serif;
  size: 18px;
  weight: bold;
}

text: {
  align: center;
  transform: lowercase;
  overflow: hidden;
}
```

CSS

```
font-family: Helvetica, sans-serif;
font-size: 18px;
font-weight: bold;

text-align: center;
text-transform: lowercase;
text-overflow: hidden;
```

SASS Imports

Sass keeps the CSS code DRY (Don't Repeat Yourself). One way to write DRY code is to keep related code in separate files.

You can create small files with CSS snippets to include in other Sass files. Examples of such files can be: reset file, variables, colors, fonts, font-sizes, etc.

Just like CSS, Sass also supports the `@import` directive.

The `@import` directive allows you to include the content of one file in another.

The CSS `@import` directive has a major drawback due to performance issues; it creates an extra HTTP request each time you call it. However, the Sass `@import` directive includes the file in the CSS; so no extra HTTP call is required at runtime!

```
@import "variables";
@import "colors";
@import "reset";
```

SASS Mixins

The `@mixin` directive lets you create CSS code that is to be reused throughout the website.

The `@include` directive is created to let you use (include) the mixin.

Defining a Mixin

```
@mixin name {
  property: value;
  property: value;
  ...
}

@mixin important-text {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
}
```

Using a Mixin

```
.danger {
  @include important-text;
  background-color: green;
}
```

CSS Result

```
.danger {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
  background-color: green;
}
```

SASS Extend

The `@extend` directive lets you share a set of CSS properties from one selector to another.

The `@extend` directive is useful if you have almost identically styled elements that only differ in some small details.

The following Sass example first creates a basic style for buttons (this style will be used for most buttons). Then, we create one style for a "Report" button and one style for a "Submit" button. Both "Report" and "Submit" button inherit all the CSS properties from the .button-basic class, through the `@extend` directive. In addition, they have their own colors defined

SASS

```
.button-basic  {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report  {
  @extend .button-basic;
  background-color: red;
}

.button-submit  {
  @extend .button-basic;
  background-color: green;
  color: white;
}
```

CSS

```
.button-basic, .button-report, .button-submit {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report  {
  background-color: red;
}

.button-submit  {
  background-color: green;
  color: white;
}
```

Learning Resources

1. **NPM**
    1. **https://thecodebarbarian.com/an-introduction-to-npm.html**
    2. **https://medium.com/beginners-guide-to-mobile-web-development/introduction-to-npm-and-basic-npm-commands-18aa16f69f6b**
2. **Pre-Processor**
    1. **https://www.w3schools.com/sass/sass_extend.asp**
    2. **https://htmlmag.com/article/an-introduction-to-css-preprocessors-sass-less-stylus**
    3. **https://medium.com/stories-from-the-keen/when-to-use-extends-vs-mixins-in-sass-b09d55abd53**
    4. **Install node-sass: https://webdesign.tutsplus.com/tutorials/watch-and-compile-sass-in-five-quick-steps--cms-28275**

Home Work

1. Go to your lesson 9 HW folder
2. Initiate npm there
3. Install sass
4. Create a build and watch command
5. Refactor your css with sass
6. Upload this to GitHub
7. You will need to change build step in Netlify config as we did in class