Curso de PostCSS

# Bienvenida al Curso de PostCSS

**Qué es postCSS?**

PostCSS es una herramienta para transformar CSS con JavaScript.  
Es una herramienta construida en JavaScript que sirve para manipular los archivos de CSS y convertirlos en “un mejor CSS”.

Existen nuevos features de CSS que no son todavía soportados por todos los navegadores.  
PostCSS trabaja con features y sintaxis de CSS4 transpilando un archivo de código en CSS3 que todos los navegadores hasta la fecha puedan interpretar.

# Instalación y uso del cliente de PostCSS

* Puedes usar PostCSS diferentes formas, por ejemplo con webpack, gulp, etc. En esta clase vamos a instalar PostCSS usando el cliente de PostCSS.
* Para instalar PostCSS primero debes tener instalado node.js en tu equipo porque vamos a estar trabajando con NPM

**(\*) Repositorio de curso PostCSS:** <https://github.com/LeonidasEsteban/platzi-video-postcss>

**1.** Requisitos:

* Tener instalado [NodeJS](https://nodejs.org/en/" \t "_blank)

**2.** Para empezar debemos crear un archivo package.json inicial rápido:

npm init -y

**3.** Ahora tenemos que instalar el postcss-cli en el entorno de trabajo y no global.

npm install postcss-cli --save-dev

**4.** Para ejecutar el comando en el entorno de trabajo ejecutamos:

npx postcss-cli --version

**(\*) Documentación PostCSS CLI:** <https://github.com/postcss/postcss-cli>

## Transformar Estilos

Los comandos del PostCSS se encuentran en el [Repositorio Oficial](https://github.com/postcss/postcss-cli)

**1.** Para transformar un archivo en ser ejecuta el siguiente comando:

npx postcss src/css/home.css -o dist/css/home.css

* La primera ruta indica el archivo base en PostCSS
* -o indica la salida
* La segunda ruta es donde se guardara el archivo transformado

**2.** Transformar un ves que se realice un cambio:

npx postcss src/css/home.css -o dist/css/home.css -w

* [ -w | --watch] revisa los cambios

**3.** Uso de Plugins de PostCSS:

npx postcss src/css/home.css -o dist/css/home.css -w -u

* [ -u | --use] uso de plugins de PostCSS

**4.** Cambiar la ruta en nuestro index.html al archivo transformado.

<link rel="stylesheet" href="dist/css/home.css" />

# Instalando y usando plugins en PostCSS

**npx postcss src/css/home.css -o dist/css/home.css -w -u autoprefixer**

**npx postcss**: linea de comando para llamar la funcionalidad de postcss  
**src/css/home.css:** Archivo de entrada de postcss (entry point)  
**-o (–output):** Flag usado para decirles postcss a donde enviar el archivo transformado  
**dist/css/home.css:** Archivo final que entrega postcss(output). si no están creadas las carpetas, las crea.  
**-w (–watch):** Flag usado para decirle a postcss que se quede escuchando los cambios en los archivos  
**-u(–use):** Flag usado para decirle a postcss que plugin vamos usar.  
**autoprefixer:** Plugin mas usado en postcss. Usado para agregar prefijos para algunas reglas de css que soportan los navegadores antiguos pero con prefijos.

PostCSS es una herramienta para transformar CSS con JS y todo lo que hará es transpilar un archivo legible para multiples browsers a partir de los parámetros de los plugins que instalemos.

[Referencia plugins](https://www.postcss.parts/).

Podemos también generar nuestros plugins personales.

El primero que usaremos es **autoprefixer**. Este es el plugin más popular y existe desde antes de postCSS.  
Lo que este plugin hace es agregar prefijos como -webkit- o -moz- por ejemplo a las propiedades de CSS siempre que sea necesario y los navegadores no soporten esa propiedad.

Para instalar:

npm i --save autoprefixer

Se agrega entonces la dependencia. Podemos ver la versión en el package.json file.

"dependencies": {

"autoprefixer": "^9.4.9",

"postcss-cli": "^6.1.2"

}

Para utilizarlo hay dos formas.

• De forma básica o por defecto a través del terminal con el comando -u después de -w:

npx postcss src/css/home.css -o dist/css/home.css -w -u autoprefixer

• De forma custom generando un archivo postcss.config.js en el root de nuestro proyecto.

* carga por defecto

module.exports = {

plugins: [

require('autoprefixer')

]

}

* carga avanzada agregando prefijos para propiedades específicas de CSS:

module.exports = {

plugins: [

require('autoprefixer')({

grid: **true**

})

]

}

Como algo todavía más específico otra posibilidad es la de encender e interrumpir el mecanismo de generar prefijos por cada clase o id de css es nuestro style.css de origen.  
[Ver documentación](https://github.com/postcss/autoprefixer) por propiedades ajustables.

Ej:

.a {

transition: 1s; /\* será prefixeada \*/

}

.b {

/\* autoprefixer: off \*/

transition: 1s; /\* no será prefixeada \*/

}

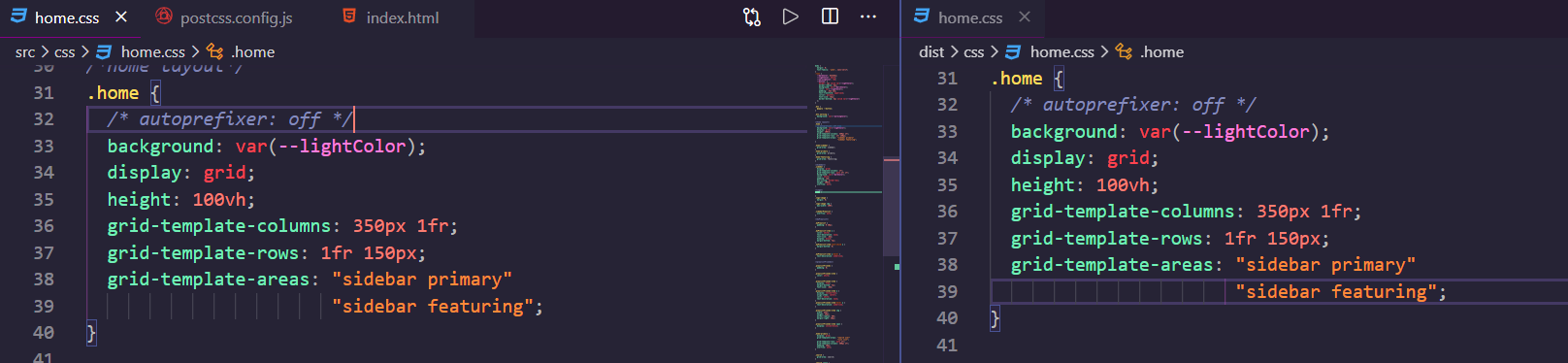
.c {

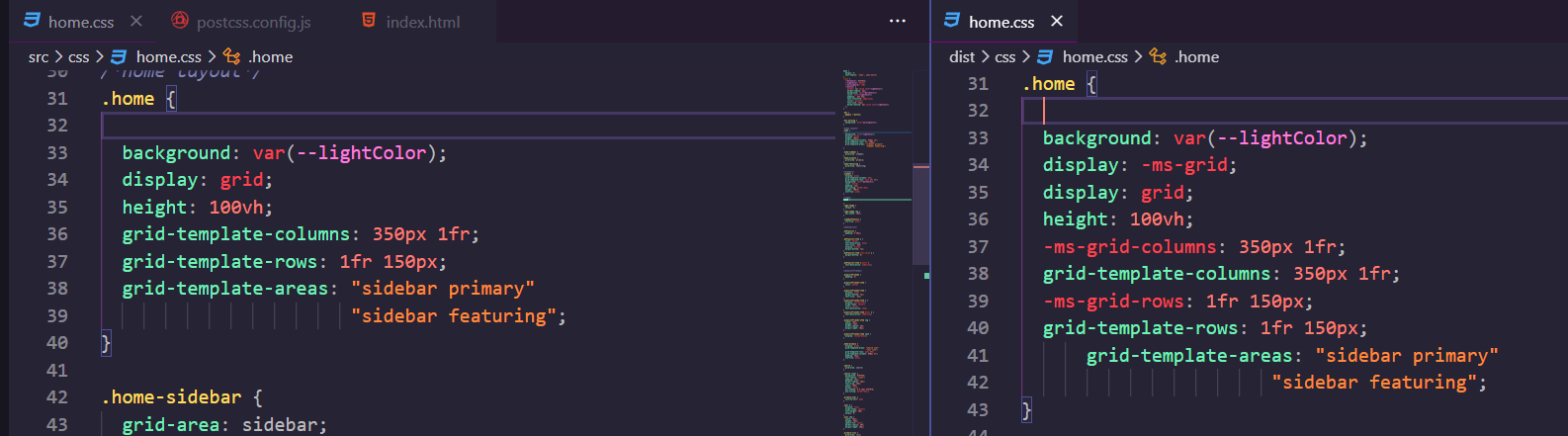
/\* autoprefixer: ignore next \*/

transition: 1s; /\* no será prefixeada \*/

mask: url(image.png); /\* será prefixeada \*/

}





NextCSS - El futuro de CSS

# Instalando postcss-preset-env

El creador de CSSNext ha anunciado que ya no va a seguir con el desarrollo del plugin y ahora se encuentra obsoleto.

<https://moox.io/blog/deprecating-cssnext/>

Recomienda usar [postcss-preset-env.](https://preset-env.cssdb.org/) en su lugar.

**npm install postcss-preset-env**

Entiendo que para usar este nuevo plugin la única diferencia para el resto del curso sería instalar el plugin y agregarlo al postcss.config.js?

module.exports = {

plugins: [

require("postcss-preset-env")

]

}

Para que funcione el autoprefixer con este nuevo plugin solo eliminen la parte de features.

**Antes**

module.exports = {

plugins: [

require('postcss-preset-env')({

features: {

autoprefixer: {

grid: **true**,

}

}

})

]

}

**Después**

module.exports = {

plugins: [

require('postcss-preset-env')({

autoprefixer: {

grid: **true**,

}

})

]

}

# Los nuevos módulos de CSS

Vamos a entrar a mi parte favorita de este curso y la razón más importante para incluir PostCSS + CSSNext a tu Stack.

CSS ya no es un conjunto de propiedades para crear nuestros estilos, ya no es más un paquete que recibe un único nombre y se optó por algo mucho mejor para que estas nuevas características sean adoptadas por los navegadores más rápidamente, a estos los llamamos módulos.

Una ventaja de los módulos es que no necesariamente tienen que estar completos para ser implementados en el browser, pueden ir por niveles de la especificación y así garantizar constantes mejoras y nuevas características.

Te haré un resumen de los módulos a tratar en este curso y que gracias a CSSNext podemos hacerlo **compatibles hoy mismo**.

### CSS Custom Properties for Cascading Variables Module Level 1

Esta es una característica que nos permitirá traer a CSS algo que extraños mucho de los lenguaje de programación, las variables. Así podemos guardar por ejemplo el color hexadecimal preciso que necesitamos y darle un nombre que recordemos como –elVerdePerfecto.  
<https://www.w3.org/TR/css-variables>

### Media Queries Level 4

Los media queries son nuestros mejores amigos para cambiar el CSS de algunos elementos dependiendo de las condiciones del navegador, es decir, en Responsive Design. Ahora podremos nombrar un media query como si fuera una variable para ser más fácil de reutilizar y rangos de media queries para una mejor sintaxis.  
<https://drafts.csswg.org/mediaqueries/>

### CSS Images Module Level 3

Con image-set() vamos a poder elegir una determinada image de background dependiendo de la densidad de pixel que tenga el monitor  
<https://drafts.csswg.org/css-images-3/#image-set-notation>

### CSS Color Module Level 4

Ya conocemos los hexadecimales, rgb() rgba() y ahora con la función color() vamos a poder seguir creando variaciones a la forma de asignar colores.  
<https://drafts.csswg.org/css-color/#modifying-colors>

### CSS Fonts Module Level 4

La forma de agregar tipografias también viene con mejoras y mi favorita se llama “System UI”. Con System UI podemos asignar un fallback que tomará la fuente predefinida de tu sistema operativo, en el caso de mac "San Francisco"  
<https://drafts.csswg.org/css-fonts-4/#valdef-font-family-system-ui>

### CSS Extensions - Custom Selectors

CSS quiere permitirte agrupar un selector o conjunto de selectores en algo más fácil de recordar como es el caso de las variables, estas se podrán anidar

### Selectors Level 4

Aprenderemos a usar psedu clases como :any-link() :not() :matches()  
<https://drafts.csswg.org/selectors-4/#matches>

### Nesting

Este es mi favorito, es una propuesta que nos permitirá usar mi cualidad favorita de los preprocesadores, evitar repetir un selector previamente escrito, es dificil de explicar así que dejaré un pequeño ejemplo

CSS actual

.mi .selector .css {

color: orange;

}

CSS Nesting

.mi {

color: red;

& .selector {

color: blue;

& .css {

color: orange;

}

}

}

Mientras que con nuestros selectores tradicionales tenemos que escribir en profundidad tanto como sea necesario con nesting (indentado en español) vamos a poder escribir los selectores una sola vez e ir agregándoles estilos independientemente

<http://tabatkins.github.io/specs/css-nesting/>

El conjunto de todo esto nos da un mejor CSS, uno que día a día se irá agregando en los navegadores y hará que no sea necesario hacerlo compatible con cssnext, cuando ese día llegue solo tienes que desactivarlo y como verás en las próximas clases podrás hacerlo independientemente por cada característica.

Ahora que sabes que es una buena idea escribir código que en un futuro será el estándar continúa con el curso que espero te deje impresionado clase a clase.

# Variables

npm install postcss-apply --save-de

Es necesario instalar ‘postcss-apply’, en caso instalaron ‘postcss-preset-env’. El archivo postcss.config.js quedaría así:

module.exports = {

plugins: [

require('postcss-apply'),

require('postcss-preset-env')({

autoprefixer: {

grid: **true**,

flexbox: **false**,

},

preserve: **true**, //Valor por defecto de "true".Se asemeja a customProperties

//cambiar a false para notar el cambio

}),

],

};

[postcss-preset-env](https://preset-env.cssdb.org/features#custom-properties) por defecto ya no elimina el root(lo eliminaba para hacer compatible nuestro código) y agrega las variables de esta forma:

.btn.warning {

background: red;//Se elimina por cascada, aunque a mi parecer ya no debería añadirse

background: var(--warningColor);

}

El [repo](https://github.com/postcss/postcss-custom-properties) es aún muy joven y tiene algunas deficiencias pero lo bueno es que tiene una comunidad muy activa dándole soporte.  
P.D. el último enlace pertenece a ‘postcss-custom-properties’ el cual se instala por defecto cuando se instala ‘postcss-preset-env’.

Me parece bastante interesante la posibilidad de crear **mixins** con CSS, que recuerdo que ya están **disponibles** las variables en CSS en **la mayoría de los navegadores** estables.

## CSS Variables

Las variables en CSS se pueden definir con 2 guiones medios (–) --variable:"contenido de la variable" y usarse con la palabra reservada var, de la siguiente manera propiedad:var(--variable);

Así se definen variables

:root {

/\* Variables \*/

--darkColor: #15192a;

--lightColor: white;

--warningColor: red;

}

Así se llaman las variables

.btn.warning {

background: var(--warningColor);

}

## Mixins

Para hacer un mixin en CSS necesitas crear una variable CSS con todas las propiedades de CSS que quieras agregar y aplicarlo con @apply, de la siguiente forma:

:root {

/\* Mixins \*/

--button: {

border: 1px solid var(--lightColor);

border-radius: 5px;

background: var(--darckColor);

color: var(--lightColor);

padding: .5em 2em;

text-transform: uppercase;

cursor: pointer;

font-size: 14px;

border-bottom: 5px solid var(--lightColor)

};

}

Aplicación de estilos

.btn {

@apply --button;

}