**Робота препроцессора sass**

Прежде, чем Вы сможете использовать Sass, Вам необходимо его настроить в вашем проекте. Если Вы хотите просто почитать, то не стесняйтесь — читайте, но мы рекомендуем сначала установить Sass. Установите Sass для того, чтобы разобраться во всех возможностях Sass.

## Приложения

## Есть множество приложений под Windows, Mac и Linux, которые помогут в освоении и работе с Sass. Большинство из них бесплатные, но некоторые из них все-таки платные (но это того стоит).

* [CodeKit](https://incident57.com/codekit/) (Платно) Mac
* [Compass.app](http://compass.kkbox.com/) (Условно-бесплатно) Mac Windows Linux
* [Ghostlab](http://www.vanamco.com/ghostlab/) (Платно) Mac Windows
* [Hammer](http://hammerformac.com/) (Платно) Mac
* [Koala](http://koala-app.com/) (Бесплатно) Mac Windows Linux
* [LiveReload](http://livereload.com/) (Условно-бесплатно) Mac Windows
* [Prepros](https://prepros.io/) (Платно) Mac Windows Linux
* [Scout-App](http://scout-app.io/) (Бесплатно) Windows Linux Mac

## Краткое описание теории

## Препроцессинг

Написание CSS само по себе весело, но когда таблица стилей становится огромной, то становится и сложно её обслуживать. И вот в таком случае нам поможет препроцессор. Sass позволяет использовать функции недоступные в самом CSS, например, переменные, вложенности, миксины, наследование и другие приятные вещи, возвращающие удобство написания CSS.

Как только Вы начинаете пользоваться Sass, препроцессор обрабатывает ваш Sass-файл и сохраняет его как простой CSS-файл, который Вы сможете использовать на любом сайте.

Самый простой способ получить такой результат - использовать терминал. После того, как Sass установлен, вы можете компилировать ваш Sass в CSS, используя команду sass. Вам всего лишь нужно сообщить Sass, где взять файл Sass и в какой файл CSS его скомпилировать. Например, запустив команду sass input.scss output.css в терминале, вы сообщаете Sass взять один Sass файл, input.scss, и скомпилировать в файл output.css.

Также, вы можете следить за изменениями только определенных файлов или папок, используя флаг --watch. Данный флаг сообщает Sass, что необходимо следить за изменениями указанных файлов и при наличии таковых производить перекомпиляцию CSS после сохранения файлов. Если вы хотите отслеживать изменения (вместо ручной перекомпиляции) вашего файла, например, input.scss, то вам необходимо просто добавить флаг в команду:

sass –watch input.scss output.css

Вы также можете указать папки для отслеживания изменений и куда сохранять компилированные CSS файлы, для этого достаточно указать пути и разделить их двоеточием, например:

sass --watch app/sass:public/stylesheets

Sass будет отслеживать все файлы в директории app/sass и компилировать CSS в директорию public/stylesheets.

## Переменные

Думайте о переменных, как о способе хранения информации, которую вы хотите использовать на протяжении написания всех стилей проекта. Вы можете хранить в переменных цвета, стеки шрифтов или любые другие значения CSS, которые вы хотите использовать. Чтобы создать переменную в Sass нужно использовать символ $. Рассмотрим пример:

* [SCSS](https://sass-scss.ru/guide/#example-1-scss)
* [Sass](https://sass-scss.ru/guide/#example-1-sass)
* [CSS](https://sass-scss.ru/guide/#example-1-css)

### **SCSS SYNTAX**

$font-stack: Helvetica**,** sans-serif;

$primary-color: #333;

body {

**font**: 100% $font-stack;

**color**: $primary-color;

}

### **CSS OUTPUT**

body {

**font**: 100% Helvetica, sans-serif;

**color**: #333;

}

Когда Sass обрабатывается, он принимает значения, заданные нами в $font-stack и $primary-color и вставляет их в обычном CSS-файле в тех местах, где мы указывали переменные как значения. Таким образом переменные становятся мощнейшей возможностью, например, при работе с фирменными цветами, используемыми на всем сайте.

## Вложенности

При написании HTML, Вы, наверное, заметили, что он имеет четкую вложенную и визуальную иерархию. С CSS это не так.

Sass позволит вам вкладывать CSS селекторы таким же образом, как и в визуальной иерархии HTML. Но помните, что чрезмерное количество вложенностей делает ваш документ менее читабельным и воспринимаемым, что считается плохой практикой.

Чтобы понять что мы имеем ввиду, приведем типичный пример стилей навигации на сайте:

* [SCSS](https://sass-scss.ru/guide/#example-2-scss)
* [Sass](https://sass-scss.ru/guide/#example-2-sass)
* [CSS](https://sass-scss.ru/guide/#example-2-css)

### **SCSS SYNTAX**

nav {

ul {

**margin**: 0;

**padding**: 0;

**list-style**: none;

}

li { **display**: inline-block; }

a {

**display**: block;

**padding**: 6px 12px;

**text-decoration**: none;

}

}

### **CSS OUTPUT**

nav ul {

**margin**: 0;

**padding**: 0;

**list-style**: none;

}

nav li {

**display**: inline-block;

}

nav a {

**display**: block;

**padding**: 6px 12px;

**text-decoration**: none;

}

Вы заметили, что селекторы ul, li, и a являются вложенными в селектор nav? Это отличный способ сделать ваш CSS-файл более читабельным. Когда вы сгенерируете CSS-файл, то на выходе вы получите что-то вроде этого:

## Фрагментирование

Вы можете создавать фрагменты Sass-файла, которые будут содержать в себе небольшие отрывки CSS, которые можно будет использовать в других Sass-файлах. Это отличный способ сделать ваш CSS модульным, а также облегчить его обслуживание. Фрагмент — это простой Sass-файл, имя которого начинается с нижнего подчеркивания, например, \_partial.scss. Нижнее подчеркивание в имени Sass-файла говорит компилятору о том, что это только фрагмент и он не должен компилироваться в CSS. Фрагменты Sass подключаются при помощи директивы @import.

## Импорт

CSS имеет возможность импорта, которая позволяет разделить ваш CSS-файл на более мелкие и облегчить@import, то в CSS создается еще один HTTP-запрос. Sass берет идею импорта файлов через директиву @import, но вместо создания отдельного HTTP-запроса Sass импортирует указанный в директиве файл в тот, где он вызывается, т.е. на выходе получается один CSS-файл, скомпилированный из нескольких фрагментов.

Например, у вас есть несколько фрагментов Sass-файлов — \_reset.scss и base.scss. И мы хотим импортировать \_reset.scss в base.scss.

* [SCSS](https://sass-scss.ru/guide/#example-3-scss)
* [Sass](https://sass-scss.ru/guide/#example-3-sass)
* [CSS](https://sass-scss.ru/guide/#example-3-css)

### **SCSS SYNTAX**

*// \_reset.scss*

html**,**

body**,**

ul**,**

ol {

**margin**: 0;

**padding**: 0;

}

*// base.scss*

**@import** 'reset';

body {

**font**: 100% Helvetica**,** sans-serif;

**background-color**: #efefef;

}

### **CSS OUTPUT**

html**,**

body**,**

ul**,**

ol {

**margin**: 0;

**padding**: 0;

}

body {

**font**: 100% Helvetica, sans-serif;

**background-color**: #efefef;

}

Обратите внимание на то, что мы используем @import 'reset'; в base.scss файле. Когда вы импортируете файл, то не нужно указывать расширение .scss. Sass — умный язык и он сам догадается.

## Миксины (примеси)

Некоторые вещи в CSS весьма утомительно писать, особенно в CSS3, где плюс ко всему зачастую требуется использовать большое количество вендорных префиксов. Миксины позволяют создавать группы деклараций CSS, которые вам придется использовать по нескольку раз на сайте. Вы даже можете передавать переменные в миксины, чтобы сделать их более гибкими. Так же хорошо использовать миксины для вендорных префиксов. Пример для transform:

* [SCSS](https://sass-scss.ru/guide/#example-4-scss)
* [Sass](https://sass-scss.ru/guide/#example-4-sass)
* [CSS](https://sass-scss.ru/guide/#example-4-css)

### **SCSS SYNTAX**

**@mixin** **transform**($property) {

-webkit-transform: $property;

-ms-transform: $property;

**transform**: $property;

}

**.box** { **@include** **transform**(**rotate**(30deg)); }

### **CSS OUTPUT**

**.box** {

**-webkit-transform**: rotate(30deg);

**-ms-transform**: rotate(30deg);

**transform**: rotate(30deg);

}

To create a mixin you use the @mixin directive and give it a name. We've named our mixin transform. We're also using the variable $property inside the parentheses so we can pass in a transform of whatever we want. After you create your mixin, you can then use it as a CSS declaration starting with @include followed by the name of the mixin.

## Расширение/Наследование

Это одна из самых полезных функций Sass. Используя директиву @extend можно наследовать наборы свойств CSS от одного селектора другому. Это позволяет держать ваш Sass-файл в «чистоте». В нашем примере мы покажем вам как сделать стили оповещений об ошибках, предупреждениях и удачных исходах, используя другие возможности Sass, которые идут рука-об-руку с расширением, классами-шаблонами. Класс-шаблон - особый тип классов, который выводится только при использовании расширения - это позволит сохранить ваш скомпилированный CSS чистым и аккуратным.

* [SCSS](https://sass-scss.ru/guide/#example-5-scss)
* [Sass](https://sass-scss.ru/guide/#example-5-sass)
* [CSS](https://sass-scss.ru/guide/#example-5-css)

### **SCSS SYNTAX**

*/\* This CSS will print because %message-shared is extended. \*/*

%message-shared {

**border**: 1px solid #ccc;

**padding**: 10px;

**color**: #333;

}

*// This CSS won't print because %equal-heights is never extended.*

%equal-heights {

**display**: flex;

**flex-wrap**: wrap;

}

**.message** {

**@extend** %message-shared;

}

**.success** {

**@extend** %message-shared;

**border-color**: green;

}

**.error** {

**@extend** %message-shared;

**border-color**: red;

}

**.warning** {

**@extend** %message-shared;

**border-color**: yellow;

}

### **CSS OUTPUT**

**.message,** **.success,** **.error,** **.warning** {

**border**: 1px solid #ccc;

**padding**: 10px;

**color**: #333;

}

**.success** {

**border-color**: green;

}

**.error** {

**border-color**: red;

}

**.warning** {

**border-color**: yellow;

}

Вышеуказанный код сообщает классам .message, .success, .error и .warning вести себя как %message-shared. Это означает, что где бы не вызывался %message-shared, то и .message, .success, .error и .warning тоже будут вызваны. Магия происходит в сгенерированном CSS, где каждый из этих классов получает css-свойства, как и %message-shared. Это позволит вам избежать написания множества классов в HTML элементах.

Вы можете расширить большинство простых CSS селекторов прибавление к классам-шаблонам в Sass, однако, использование шаблонов - простейший способ быть уверенным, что вы не расширяете класс везде, где он используется в ваших стилях, что могло бы привести к непреднамеренным наборам стилей в вашем CSS.

Когда вы генерируете ваш CSS, то он будет выглядеть как пример ниже. Обратите внимание, %equal-heights не попадает в CSS, так как ни разу не был использован.

## Математические операторы

Использовать математику в CSS очень полезно. Sass имеет несколько стандартных математических операторов, таких как +, -, \*, / и %. В нашем примере мы совершаем простые математические вычисления для расчета ширины aside и article.

* [SCSS](https://sass-scss.ru/guide/#example-6-scss)
* [Sass](https://sass-scss.ru/guide/#example-6-sass)
* [CSS](https://sass-scss.ru/guide/#example-6-css)

### **SCSS SYNTAX**

**.container** {

**width**: 100%;

}

article**[**role**=**"main"**]** {

**float**: left;

**width**: 600px **/** 960px **\*** 100%;

}

aside**[**role**=**"complementary"**]** {

**float**: right;

**width**: 300px **/** 960px **\*** 100%;

}

### **CSS OUTPUT**

**.container** {

**width**: 100%;

}

article**[**role**=**"main"**]** {

**float**: left;

**width**: 62.5%;

}

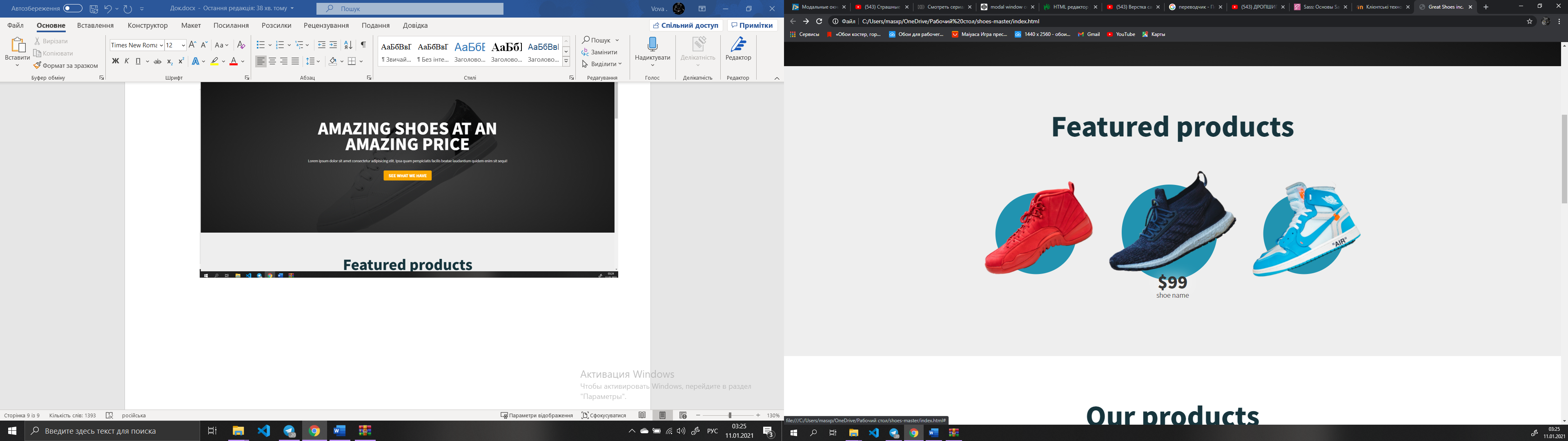
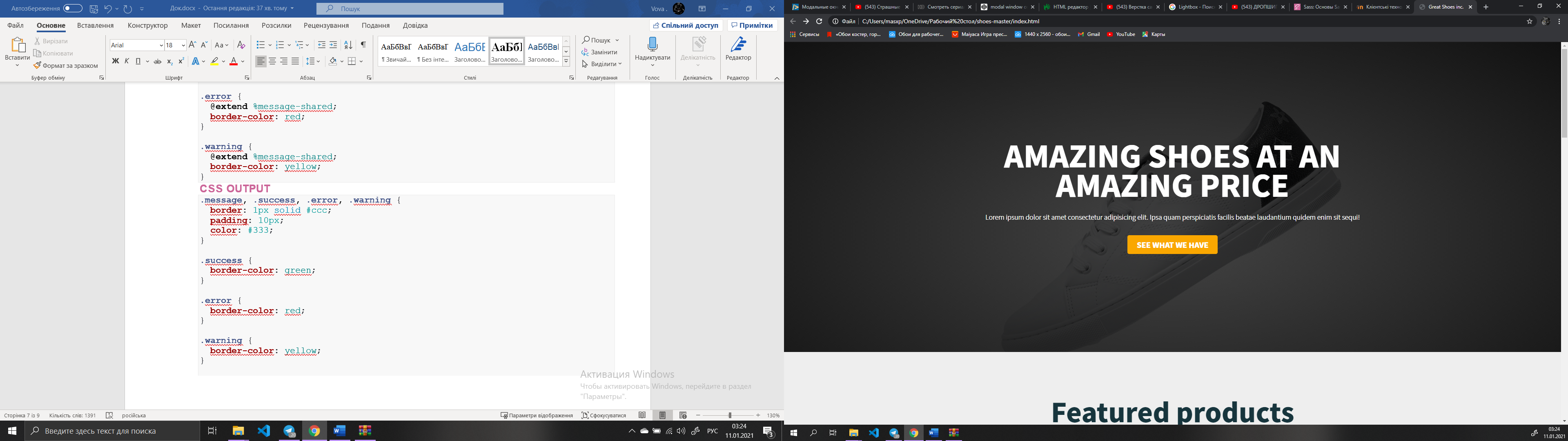
aside**[**role**=**"complementary"**]** {

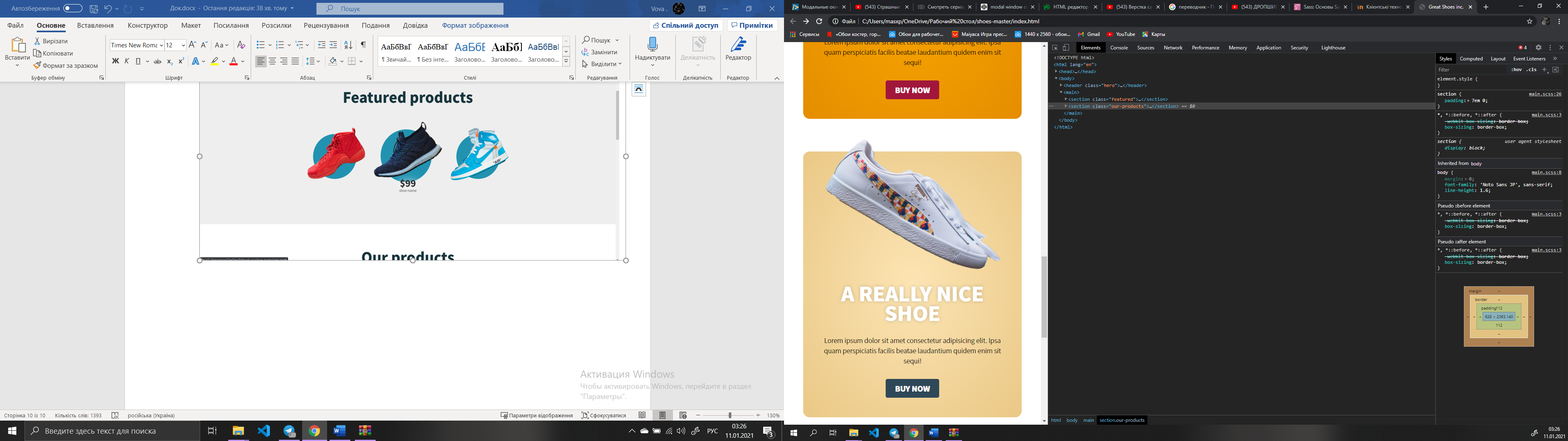
**float**: right;

**width**: 31.25%;

}

Мы создали простую адаптивную модульную сетку, с шириной в 960 пикселей. Используя математические операторы, мы использовали полученные данные с пиксельными значениями и конвертировали их в процентные, причем без особых усилий. Скомпилированный CSS выглядит так:

Скриншоты сайта



Код и картинки будут в папке