**Введение**

CSS-препроцессоры — это «программистский» подход к CSS. Они позволяют при написании стилей использовать свойственные языкам программирования приёмы и конструкции: переменные, вложенность, наследуемость, циклы, функции и математические операции. Синтаксис препроцессоров похож на обычный CSS. Код, написанный на языке препроцессора, не используется прямо в браузере, а преобразуется в чистый CSS-код с помощью специальных библиотек.

Три самых известных препроцессора — это [Less](http://lesscss.org/" \t "_blank), [SASS](http://sass-lang.com/) и [Stylus](http://learnboost.github.io/stylus/). Они во многом похожи между собой, но имеют и ключевые различия. В этом и последующих частях цикла мы рассмотрим препроцессор Less.

в тренажёре по Less мы будем шаг за шагом строить небольшой  фреймворк компонентов. Когда он будет готов, мы сможем собирать из компонентов цельные интерфейсы. Компонентный подход позволяет структурировать большие объёмы кода и легко масштабировать проекты. Препроцессор в этом деле — хорошее подспорье.

В этом тренажёре редактор CSS заменён на редактор Less. Чтобы увидеть скомпилированный из Less кода CSS код, можете использовать кнопку CSS в правом верхнем углу редактора.

**Переменные, шаг 1**

Цветовая схема — основа любого дизайна в вебе. Применяя цветовое кодирование, можно сделать интерфейс более понятным. Поэтому первым делом при создании нашего мини-фреймворка давайте займёмся цветовой схемой основных элементов интерфейса. В этой задаче Less нам очень поможет.

В прошлом задании в коде встречалась подобная запись:

@navy: #1d365d;

Так в Less описываются переменные. Синтаксис переменных такой:

@название\_переменной: значение\_переменной;

Создав переменную один раз, можно использовать её в любом месте кода. Например:

background-color: @navy;

color: @navy;

border-color: @navy;

Во всех местах, где указана переменная, Less заменит строку @navy на #1d365d. Теперь, если понадобится изменить цвет, не нужно искать все его объявления в файле, а достаточно просто изменить значение переменной в одном месте.

# Переменные, шаг 2

Переменные можно объявлять как «снаружи» правил, так и «внутри». В случае «внутреннего» объявления переменная будет доступна только внутри правила, в котором она объявлена:

.rule-1 {

@align: right;

text-align: @align; // text-align задаётся значение right

}

.rule-2 {

text-align: @align; // в этом месте произойдёт ошибка

}

Если переменная объявлена и «внутри» правила, и «снаружи» — Less применит «внутреннее» значение.

@align: left;

.rule-1 {

@align: right;

text-align: @align; // text-align задаётся значение right

}

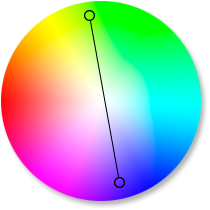
.rule-2 {

text-align: @align; // text-align задаётся значение left

}

Таким образом можно «переопределять» глобальные переменные в локальном контексте.

**Цветовые функции, шаг 1**



Итак, мы задали базовый цвет для нашей схемы, от него мы будем отталкиваться при выборе других цветов. Давайте узнаем, как это сделать.

Все цвета модели RGB можно расположить на цветовом колесе, где они плавно переходят друг в друга.

С помощью Less-функции spin можно повернуть цветовое колесо на определённый угол относительно заданного цвета и получить новый цвет. Функция принимает два параметра, синтаксис её такой:

spin(цвет, угол\_поворота)

Цвет можно задавать в любом цветовом формате. Значение угла может быть как положительным, так и отрицательным. При положительном угле функция повернёт колесо по часовой стрелке, при отрицательном — против. Примеры:

color: spin(red, 90); // цвет повернётся от красного на 90° по часовой

border-color: spin(#f0f, -45); // цвет на 45° от #f0f против часовой

Противоположный цвет на колесе называется *комплементарным*. Он находится под углом 180° к заданному цвету. Комплементарные цвета используют для создания контраста.

Наш основной цвет находится в сине-фиолетовой области цветового круга. Давайте для него вычислим комплементарный цвет с помощью функции spin.

# Вложенные правила, шаг 1

Отвлечёмся ненадолго от цвета и рассмотрим ещё одну замечательную особенность Less — вложенные правила. Они позволяют избавиться от дублирования одинаковых названий в коде и делают его более структурированным. Например, вот такой код:

.super-class-name {

color: #ffffff;

}

.super-class-name a {

text-decoration: none;

}

.super-class-name a span {

font-size: 1em;

}

можно более кратко и без повторов написать, используя вложенность:

.super-class-name {

color: #ffffff;

a {

text-decoration: none;

span {

font-size: 1em;

}

}

}

То есть вложенные правила просто пишутся внутри других правил. Из цепочек вложенных правил Less сам составляет итоговые селекторы.

**Вложенные правила, шаг 2**

С помощью вложенных правил можно не только обращаться к дочерним элементам, но и составлять по частям комплексные названия классов. Например, следующий код:

.super-button-red {

color: red;

}

.super-button-blue {

color: blue;

}

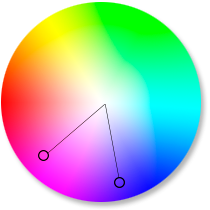
можно записать проще c помощью вложенных правил:

.super-button {

&-red { color: red; }

&-blue { color: blue; }

}

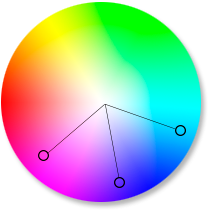


То есть, если перед вложенным правилом поставить амперсанд &, то оно станет *родственным* родительскому, и Less подставит родительский селектор вместо амперсанда.

Воспользуемся этой особенностью для структурирования кода нашей цветовой схемы. Создадим второй цвет палитры — яркий акцентный цвет ошибки, повернув цветовое колесо на 60° по часовой стрелке относительно базового, чтобы цвет сместился в красную область.

Сохраним полученный цвет в переменной и зададим с её помощью фон второго контейнера.

**Математические операции, шаг 1**



Теперь давайте создадим третий цвет. Его «сдвинем» по кругу на 60° в другую сторону от основного. Он светлее и будет использоваться для выделения второстепенной информации и элементов.

Чтобы «сдвинуть» цвет против часовой стрелки, нужно в функцию spin передать отрицательное значение переменной @distance.

Над любыми численными значениями в Less-коде можно произвести математические операции сложения, вычитания, умножения или деления:

padding-top: 10px + 20; // = 30px

padding-bottom: 100px - 50; // = 50px

font-size: 2em \* 2; // = 4em

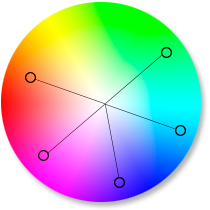
left: 50% / 2; // 25%

Less выполнит математическую операцию и вернёт в CSS уже вычисленное значение. Единицы измерения всегда берутся от первого параметра в выражении.

Чтобы вычислить отрицательное значение @distance просто умножим переменную на -1.

В выражениях Less с обеих сторон операторов + и - необходимы пробелы. Так мы не допустим синтаксической двусмысленности. Без пробелов символ - одновременно может быть интерпретирован и как оператор вычитания, и как часть ключевого слова (например, в inline-block). Операторы \* и / можно использовать без пробелов вокруг них, однако обычно их тоже выделяют пробелами для единообразия в коде.  
Корнями это требование уходит в CSS и его синтаксис, который полностью поддерживается в Less. Подробнее о синтаксисе выражений в CSS читайте в [спецификации](https://drafts.csswg.org/css-values/#calc-syntax).

**Математические операции, шаг 2**



Осталось добавить ещё два цвета. Чтобы их вычислить, мы «повернём» круг на двойное расстояние @distance от базового цвета в обе стороны.

Получившиеся цвета будут комплементарными по отношению к инфо-цвету и цвету ошибки. Это именно то, что нужно: к примеру, по смыслу цвет ошибки полностью противоположен цвету успеха, поэтому они должны быть максимально контрастны.

Из математических операций используем умножение дистанции на 2.

# Математические операции, шаг 3

И теперь, когда палитра цветов построена, можно немного «поиграть»  с базовым цветом и посмотреть, как вместе с ним будут меняться остальные.

Для этого давайте просто поменяем параметры в RGB-записи цвета в переменной @base-color математическими операциями. Кстати, к значению цвета «целиком» тоже можно применять операции. В случае суммы числа и RGB-записи цвета слагаемое будет прибавляться к каждому цветовому каналу одновременно:

rgb(10, 10, 10) + 10

// то же самое, что

rgb(20, 20, 20)

# Цветовые функции, шаг 2

Продолжим создавать библиотеку компонентов: с помощью цветов из разработанной палитры зададим состояния кнопок. И по ходу дела разберём другие функции Less для работы с цветом.

Сначала зададим базовый цвет для фона кнопок и добавим его вариации по наведению и нажатию на кнопку. По наведению будем делать кнопку чуть светлее, а в момент нажатия — темнее. Для этого воспользуемся Less-функциями lighten и darken. Их синтаксис одинаковый:

color: lighten(red, 50%); // светлее red на 50%

color: darken(blue, 25%); // темнее blue на 25%

Второе значение задаётся в процентах от 0% до 100%. При задании 100% в lighten функция возвращает полностью белый цвет, а 100% для darken — полностью чёрный. То есть эти функции «смешивают» заданный цвет с белым или чёрным.

Итак, давайте зададим кнопкам нужные цвета и снова воспользуемся вложенными правилами. Чтобы задать смену цвета и для демонстрационного класса btn-hover, и для псевдокласса btn:hover, нужно прописать вложенные правила в Less так:

.btn {

&-hover,

&:hover {

color: red;

}

}

Этот Less преобразуется в следующий CSS:

.btn-hover,

.btn:hover {

color: red;

}

# Цветовые функции, шаг 3

Ещё две Less-функции для работы с цветом: saturate и desaturate. Они увеличивают и уменьшают насыщенность заданного цвета. Их синтаксис такой же как и у lighten/darken:

color: saturate(green, 20%); // green насыщеннее на 20%

color: desaturate(blue, 50%); // blue бледнее на 50%

Функцию desaturate мы используем для создания стиля «отключённой» кнопки. Совместно с desaturate воспользуемся lighten, чтобы сделать кнопку бледной и светлой. Одновременно функции цвета можно использовать так:

// цвет светлее красного на 50% и насыщеннее на 20%

color: saturate(lighten(red, 50%), 20%);

// цвет темнее синего на 20% и бледнее на 50%

color: desaturate(darken(blue, 20%), 50%);

То есть функции можно «вкладывать» друг в друга, используя их в качестве аргументов.

# Примеси

Ещё одна интересная возможность Less — примеси. Мы можем «примешивать» содержимое одного CSS-правила в другое. Для этого нужно написать имя «примешиваемого» правила внутри другого правила. Рассмотрим пример:

Less.white { color: white; } /\* объявление примеси \*/

.text { .white; } /\* применение примеси \*/

Этот Less-код скомпилируется в такой CSS:

CSS.white { color: white; }

.text { color: white; }

Как мы видим, в правиле, где была «вызвана» примесь .white, появилось её содержимое.

Чтобы не выводить саму примесь в CSS, нужно после объявления примеси поставить скобки:

Less.white() { color: white; } /\* объявление невыводимой примеси \*/

.text { .white; } /\* применение примеси \*/

CSS.text { color: white; }

При применении примеси скобки писать необязательно.

Less/\* Эти выражения дают один и тот же результат: \*/

.mixin();

.mixin;

# Несколько примесей

К одному правилу можно применить несколько примесей одновременно. В таком случае примеси просто «вызываются» по очереди.

Например:

Less.big() {

width: 100500px;

}

.white() {

color: white;

}

.block {

.big();

.white();

}

CSS.block {

width: 100500px;

color: white;

}

# Примесь с параметром

В примесь можно передавать параметры. Они указываются внутри скобок объявления примесей. Названия параметров начинаются с символа @. Рассмотрим пример:

Less.margin(@value) {

margin-top: @value;

margin-bottom: @value;

}

.block {

.margin(10px);

}

CSS.block {

margin-top: 10px;

margin-bottom: 10px;

}

# Значение параметра примеси по умолчанию

Параметризованные примеси можно сделать ещё универсальнее.

Параметру примеси можно задать значение по умолчанию. Оно указывается через двоеточие после названия параметра. Если в примесь при применении параметр не передаётся, то используется значение по умолчанию.

Рассмотрим пример, в котором значение параметра по умолчанию указано, но не задействуется, так как в примесь передаётся явный параметр.

Less.big(@size: 100500px) {

width: @size;

}

.block {

.big(10px);

}

CSS.block {

width: 10px;

}

В следующем примере примесь применяется без параметров, поэтому используется значение по умолчанию:

Less.big(@size: 100500px) {

width: @size;

}

.block {

.big();

}

CSS.block {

width: 100500px;

}

# Примесь с несколькими параметрами

В примесь можно передавать несколько параметров. Параметры перечисляются через запятую , или через точку с запятой ;. Рекомендуется использовать точку с запятой. Пример:

Less.offset(@padding; @margin) {

padding: @padding;

margin: @margin;

}

.block {

.offset(5px; 10px);

}

CSS.block {

padding: 5px;

margin: 10px;

}

# Шаблоны примесей

Иногда бывает полезным изменить поведение примеси в зависимости от передаваемых параметров. Например, у нас есть примесь, задающая размер шрифта:

Less.set-font-size(@size) {

font-size: @size;

}

Мы можем создать ещё одну примесь с таким же названием, но передать дополнительно первым параметром название шаблона этой примеси. Имя шаблона указывается первым перед параметрами самой примеси. Добавим примеси дополнительный первый параметр smaller и немного изменим принцип её работы:

Less.set-font-size(smaller; @size) {

font-size: @size / 2;

}

Теперь можно вызывать эту примесь с названием шаблона и без него и, в зависимости от этого, получать соответствующие результаты:

Less.text {

.set-font-size(100px);

}

.small-text {

.set-font-size(smaller; 100px);

}

CSS.text {

font-size: 100px;

}

.small-text {

font-size: 50px;

}

Таким образом можно для схожих действий не создавать несколько примесей с разными названиями. Лучше делать шаблоны одной примеси и просто вызывать её с дополнительным параметром.

Давайте попробуем создать шаблон примеси и применить её для окраски монстров.

В задании применяется встроенная в Less функция lighten для работы с цветом. Она рассматривается в [задании](https://htmlacademy.ru/courses/85/run/10) первой части тренажёра по Less.

# Шаблоны примесей, часть 3

Если нужно задать общие свойства для нескольких шаблонов одной и той же примеси, можно создать универсальный шаблон:

Less.font-size(bigger; @size) {

font-size: @size \* 2;

}

.font-size(smaller; @size) {

font-size: @size / 2;

}

.font-size(@\_; @size) {

color: #000000;

} // универсальный шаблон

.content-bigger {

.font-size(bigger; 20px);

}

.content-smaller {

.font-size(smaller; 20px);

}

Универсальный шаблон применяется вместе с остальными шаблонами:

CSS.content-bigger {

font-size: 40px;

color: #000000;

}

.content-smaller {

font-size: 10px;

color: #000000;

}

В качестве имени в универсальный шаблон передаётся специальная переменная @\_, за ней следуют параметры. Важно, чтобы универсальный шаблон принимал те же параметры, что и все остальные шаблоны.

# Примесь с условием

В примесях можно использовать полноценные условия, которые могут изменять поведение примеси в зависимости от значений входных параметров.

Чтобы создать условие, нужно после названия примеси поставить ключевое слово when, за которым в скобках написать условную конструкцию. Пример:

Less.mixin(@variable) when (@variable = 1) {

// сделать что-то

}

Такая примесь применится, если «вызвать» её с параметром 1. В противном случае примесь не применится.

Less.some-class {

.mixin(1);

}

В условной конструкции допускаются следующие операторы: >, >=, =, =<, <. Также допустимо использовать встроенные функции Less, которые возвращают конкретные значения.

К примеру, в Less есть встроенная функция lightness, которая принимает в качестве параметра значение цвета и возвращает степень его светлоты. Чёрный цвет обладает 0% светлоты, а белый — 100%. Вот пример её использования:

Less.mixin(@color) when (lightness(@color) > 50%) {

// сделать что-то, когда цвет светлее серого

}

.mixin(@color) when (lightness(@color) = 100%) {

// сделать что-то, когда цвет полностью белый

}

# Условия и внешние переменные

Условия в примесях могут работать не только с параметрами, с которыми «вызвана» примесь, но и с переменными, объявленными вне примесей. Например, если создать примесь с условием, но без параметров:

Less.text-color() when (@theme = light) {

color: white;

}

А потом создать внешнюю переменную и вызвать где-то примесь:

Less@theme: light;

.content {

.text-color();

}

То условие выполнится, созданная примесь отработает:

CSS.content {

color: white;

}

То есть можно управлять условиями примесей с помощью внешних переменных.

# Условия и типы параметров

В Less есть встроенные функции для проверки типа значения. Их можно применять в условиях примесей для проверки типа переданного параметра. Пример:

Less// проверка: значение — цвет

.mixin(@param) when (iscolor(@param)) { … }

// проверка: значение — число

.mixin(@param) when (isnumber(@param)) { … }

// проверка: значение — строка

.mixin(@param) when (isstring(@param)) { … }

// проверка: значение — ключевое слово

.mixin(@param) when (iskeyword(@param)) { … }

// проверка: значение — url

.mixin(@param) when (isurl(@param)) { … }

Все эти функции возвращают true, если переданный в них параметр соответствует проверяемому типу.

Таким образом можно создать универсальную примесь, которая в зависимости от типа переданных параметров будет работать по-разному.

# Переменные-вставки

Переменные можно использовать не только в значениях CSS-свойств, но и как составные части селекторов, названий свойств или как «кусочки» значений свойств. С помощью такой *подстановки переменных*, или *Variable Interpolation*, можно динамически формировать разные части CSS-правил.

Чтобы сделать подстановку значения переменной, нужно использовать фигурные скобки вокруг её имени:

Less@state: success;

@property: color;

@icon: "question";

@pixels: 2;

.btn-@{state} {

background-color: green;

}

.btn-error {

background-@{property}: red;

}

.btn-help {

background-image: url("/img/icons/@{icon}.png");

}

.btn-info {

border: ~"@{pixels}px" solid blue;

}

Из примеров выше скомпилируется такой CSS:

CSS.btn-success {

background-color: green;

}

.btn-error {

background-color: red;

}

.btn-help {

background-image: url("/img/icons/question.png");

}

.btn-info {

border: 2px solid blue;

}

Кстати, тильда ~ в примере выше нужна для хитрого механизма экранирования Less. Без неё «склеивание» переменной и единицы измерения не сработает.

С помощью «переменных-вставок» можно формировать имена селекторов динамически в зависимости от определённых условий или в цикле. Эти приёмы мы рассмотрим далее в тренажёре.

# Цикл

В Less нет специального синтаксиса для циклов. Но есть возможность вызывать примеси внутри самих себя. Так с помощью рекурсии и реализуются циклы. Рассмотрим пример:

Less.mixin(@n) {

.mixin(@n + 1);

}

.mixin(1);

В примере создаётся «бесконечный» цикл с увеличивающейся переменной-счётчиком, который никогда не закончится. Чтобы рекурсия всё-таки когда-нибудь прекращалась, к примеси добавляется условие выполнения:

Less.mixin(@n) when (@n > 0) {

.mixin(@n - 1);

}

.mixin(2);

Теперь цикл выполнится два раза, сработает условие выполнения примеси и произойдёт выход из рекурсии.

Для чего можно применять циклы? Например, для генерирования целых CSS-правил. Если в цикле в имени селектора использовать переменную-вставку из [прошлого задания](https://htmlacademy.ru/courses/125/run/15), то можно на выходе получить набор правил с разными селекторами. В примере ниже цикл исполняется три раза, в каждой итерации создастся правило с переменной-счётчиком @n в качестве суффикса селектора:

Less.mixin(@n) when (@n > 0) {

.text-@{n} {

}

.mixin(@n - 1);

}

.mixin(3);

CSS.text-3 {}

.text-2 {}

.text-1 {}

Внутри «цикла» переменную-счётчик можно использовать не только в условиях или в селекторах, но и в значениях свойств.

@line: 16px;

.set-font(@font-size, @line-height: @line) {

font-size: @font-size;

line-height: @line-height;

}

.set-margin(@margin-top: @line, @margin-bottom: @line) {

margin-top: @margin-top;

margin-bottom: @margin-bottom;

}

@base-color: rgb(110, 27, 255);

.generate-colors(@color) {

@bg-color:lighten(@color, 35%);

background-color:@bg-color;

color:darken(@bg-color, 50%);

border-color:darken(spin(@bg-color, -10), 5%);

}

**Примесь для центровки блока**

Дальше мы рассмотрим несколько полезных примесей, которые могут часто использоваться в повседневной практике.

Одна из самых распространённых задач — горизонтальное и вертикальное выравнивание блока относительно родительского контейнера.

В этом задании давайте создадим примесь для быстрого выравнивания. Особенности этого метода:

* блок позиционируется абсолютно внутри относительно спозиционированного контейнера,
* у блока должны быть фиксированные ширина и высота,
* центровка происходит за счёт нулевых координат top, right, bottom, left и автоматических внешних отступов.

.zero-centered() {

position:absolute;

top:0;

right:0;

bottom:0;

left:0;

margin: auto;

}

# Примесь для треугольных форм

Ещё одна распространённая задача — реализовать блок треугольной формы с помощью CSS. Для её решения подходит управление шириной и цветом рамок, который мы уже рассматривали в [серии заданий](https://htmlacademy.ru/courses/88/run/25) части «Рамки и фоны. Погружение».

Универсальную примесь для создания «треугольников» на CSS удобно держать в своём арсенале наготове, чтобы при необходимости просто написать одну строчку кода, не вспоминая подробности реализации.

Итак, нам нужна примесь, которая будет принимать в качестве параметров: направление, в которое «смотрит» треугольник, его размер и цвет.

Так как в примеси нужно учесть четыре направления (вверх, вправо, вниз и влево), уместно будет воспользоваться шаблонами примесей. В универсальном шаблоне будут заданы общие для всех сторон свойства, а в специфичных шаблонах — разные.

В задании в Less-код уже добавлена универсальная примесь .triangle, в которой заданы правила для всех «треугольников»: нулевые ширина и высота (так как мы будем работать с рамками, сами блоки будут невидимы), а также сплошной стиль рамки.

.triangle(@\_; @size; @color) {

width: 0;

height: 0;

border-width: @size;

border-style: solid;

border-color: @color;

}

.triangle(top; @size; @color) {

border-left-color: transparent;

border-right-color: transparent;

border-top-style: none;

}

.triangle(right; @size; @color) {

border-top-color:transparent;

border-bottom-color:transparent;

border-right-style:none;

}

.triangle(bottom; @size; @color) {

border-left-color:transparent;

border-right-color:transparent;

border-bottom-style:none;

}

.triangle(left; @size; @color) {

border-top-color:transparent;

border-bottom-color:transparent;

border-left-style:none;

}

.triangle-box-top {

.triangle(top; 50px; #ff4136);

}

.triangle-box-right {

.triangle(right; 50px; #0074d9);

}

.triangle-box-bottom {

.triangle(bottom; 50px; #2ecc40);

}

.triangle-box-left {

.triangle(left; 50px; #b10dc9);

}