CSS - SCSS & БЭМ

Препроцессоры и методологии

Препроцессор

CSS препроцессор (CSS preprocessor) - это программа, которая имеет свой собственный синтаксис (syntax (en-US)), но может генерировать из него CSS код . Существует множество препроцессоров. Большинство из них расширяет возможности чистого CSS, добавляя такие опции как: примеси, вложенные правила, селекторы наследования и др. Эти особенности облегчают работу с CSS: упрощают чтение кода и его дальнейшую поддержку.

SASS

Основные фишки SASS:

- Переменные: SASS позволяет определять переменные для удобного использования в различных частях кода.
- Миксины: Миксины предоставляют возможность создавать наборы стилей, которые могут быть многократно использованы в коде.
- Вложенные стили: SASS поддерживает вложенность стилей, что делает код более организованным и читаемым.
- Операции с цветами: SASS предоставляет функции для работы с цветами, такие как darken(), lighten(), mix() и другие.
- Импорт файлов: SASS позволяет импортировать стили из других файлов для удобного управления кодом.

```
// Определение переменных
$primary-color: #007bff
$font-stack: Arial, sans-serif

// Создание миксина
=button-styles($bg-color, $text-color)
    background-color: $bg-color
    color: $text-color
    padding: 10px 20px
    border-radius: 5px
    cursor: pointer

// Использование миксина
.button
    +button-styles($primary-color, #fff)

    &:hover
        background-color: darken($primary-color, 10%)

// Вложенные стили
.container
    width: 100%

    .header
        background-color: $primary-color
        color: #fff
        font-family: $font-stack
```

SASS (SCSS)

Основные фишки SCSS:

- Переменные: Определение переменных позволяет легко использовать одни и те же значения в различных частях вашего кода.
- Миксины: Миксины позволяют создавать наборы стилей, которые могут быть многократно использованы в вашем коде, что уменьшает дублирование и повышает читаемость.
- Вложенные стили: SCSS позволяет использовать вложенные стили, что отражает структуру HTML и делает код более организованным и понятным.
- Операции с цветами: SCSS предоставляет множество встроенных функций для работы с цветами, таких как darken(), lighten(), mix() и другие, что делает работу с цветами более удобной и гибкой.
- Импорт файлов: SCSS позволяет импортировать стили из других файлов, что упрощает организацию и структурирование вашего кода.

```scss
// Определение переменных
$primary-color: #007bff;
$font-stack: Arial, sans-serif;

// Создание миксина
@mixin button-styles($bg-color, $text-color) {
    background-color: $bg-color;
    color: $text-color;
    padding: 10px 20px;
    border-radius: 5px;
    cursor: pointer;
}

// Использование миксина
.button {
    @include button-styles($primary-color, #fff);

    &:hover {
        background-color: darken($primary-color, 10%);
    }
}

// Вложенные стили
.container {
    width: 100%;

    .header {
        background-color: $primary-color;
        color: #fff;
        font-family: $font-stack;
    }
}
```

LESS

Основные фишки LESS:

- Переменные: LESS позволяет определять переменные для удобного использования в различных частях кода.
- Миксины: Миксины в LESS позволяют создавать наборы стилей, которые могут быть многократно использованы в коде.
- Вложенные стили: LESS поддерживает вложенность стилей, что делает код более организованным и читаемым.
- Операции с цветами: LESS предоставляет функции для работы с цветами, такие как darken(), lighten(), mix() и другие.
- Импорт файлов: LESS позволяет импортировать стили из других файлов для удобного управления кодом.

```less
// Определение переменных
@primary-color: #007bff;
@font-stack: Arial, sans-serif;

// Создание миксина
.button-styles(@bg-color, @text-color) {
    background-color: @bg-color;
    color: @text-color;
    padding: 10px 20px;
    border-radius: 5px;
    cursor: pointer;
}

// Использование миксина
.button {
    .button-styles(@primary-color, #fff);

    &:hover {
        background-color: darken(@primary-color, 10%);
    }
}

// Вложенные стили
.container {
    width: 100%;

    .header {
        background-color: @primary-color;
        color: #fff;
        font-family: @font-stack;
    }
}
```

Stylus

Основные фишки Stylus:

- Без скобок и точек с запятой: Stylus позволяет писать код без использования фигурных скобок и точек с запятой, что делает его более кратким и читаемым.
- Переменные: Stylus поддерживает использование переменных для удобного управления стилями.
- Миксины: Миксины в Stylus позволяют создавать наборы стилей, которые могут быть многократно использованы в коде.
- Вложенные стили: Stylus поддерживает вложенность стилей, что делает код более организованным и читаемым.
- Операции с цветами: Stylus предоставляет функции для работы с цветами, такие как darken(), lighten(), mix() и другие.

```
// Определение переменных
primary-color = #007bff
font-stack = Arial, sans-serif

// Создание миксина
button-styles(bg-color, text-color)
    background-color: bg-color
    color: text-color
    padding: 10px 20px
    border-radius: 5px
    cursor: pointer

// Использование миксина
.button
    button-styles(primary-color, #fff)

    &:hover
        background-color: darken(primary-color, 10%)

// Вложенные стили
.container
    width: 100%

    .header
        background-color: primary-color
        color: #fff
        font-family: font-stack
```

Наиболее популярный - SASS (SCSS)

SCSS (Sassy CSS) является одним из самых популярных препроцессоров CSS по нескольким причинам:

- Близость к CSS: SCSS является расширением синтаксиса CSS3, что делает его более привычным для разработчиков, знакомых с обычным CSS. Это упрощает процесс изучения и принятия SCSS для многих разработчиков.
- Обратная совместимость: SCSS синтаксически обратно совместим с обычным CSS, что означает, что вы можете использовать существующий CSS-код в ваших SCSS-файлах без необходимости его изменения. Это облегчает интеграцию SCSS в существующие проекты.
- Мощные функциональные возможности: SCSS предоставляет богатый набор функциональных возможностей, таких как переменные, миксины, вложенные селекторы, операторы и многое другое. Это позволяет писать более модульный, гибкий и управляемый код стилей.
- Широкая поддержка и сообщество: SCSS имеет широкую поддержку в сообществе разработчиков. Существует множество инструментов, редакторов и фреймворков, которые поддерживают SCSS, что делает его привлекательным выбором для разработчиков в различных экосистемах.
- Интеграция с инструментами разработки: SCSS легко интегрируется с различными инструментами разработки, такими как сборщики, фреймворки и системы управления проектами. Это облегчает процесс разработки и поддержки проектов.

плюсы и минусы SASS (SCSS)

Плюсы SASS:

- Переменные: Вы можете определить переменные для использования во всем вашем CSS, что позволяет легко изменять значения и стили в разных частях проекта.
- Вложенность: SASS позволяет использовать вложенные правила CSS, что делает код более организованным и удобным для чтения, особенно при работе с БЭМ-структурой.
- Миксины: Миксины позволяют создавать наборы стилей, которые могут быть многократно использованы в вашем проекте, что уменьшает дублирование кода и упрощает его поддержку.
- Вложенные медиа-запросы: SASS позволяет создавать вложенные медиа-запросы, что делает адаптивный дизайн более удобным и организованным.
- Множественные выражения: SASS поддерживает использование операций, как в математике, так и в строках, что упрощает создание сложных стилей.

Минусы SASS:

- Изучение: Новым пользователям может потребоваться время, чтобы освоить синтаксис SASS и его особенности.
- Компиляция: Проекты на SASS требуют компиляции в обычный CSS перед использованием на веб-сайте, что может потребовать настройки дополнительных инструментов или процессов сборки.
- Интеграция: В некоторые проекты может потребоваться дополнительная интеграция с системами сборки или средами разработки.

SASS и SCSS разница

SASS:

- Синтаксис: SASS использует упрощенный синтаксис с отступами в качестве разделителей блоков кода, без использования фигурных скобок и точек с запятой.
- Читаемость: Благодаря своему минималистичному синтаксису, SASS-файлы обычно выглядят более чисто и читаемо.
- Привычность: Для некоторых разработчиков SASS может быть более привычным, если они ранее использовали подобные языки, такие как Python или Ruby.
- Сокращенные записи: SASS предлагает более краткие записи для некоторых стилей, что может уменьшить объем кода.

Плюсы SASS:

- Читаемость кода благодаря минималистичному синтаксису.
- Возможность использования более кратких записей.
- Привычный синтаксис для тех, кто знаком с Python или Ruby.

Минусы SASS:

- Необходимость в обучении новому синтаксису, если вы прежде не работали с ним.

SCSS:

- Синтаксис: SCSS является расширением синтаксиса CSS3 и использует фигурные скобки и точки с запятой, что делает его более похожим на обычный CSS.
- Привычность: Для многих разработчиков SCSS может быть более привычным, так как его синтаксис ближе к обычному CSS.
- Интеграция: Поскольку SCSS ближе к CSS, его легче интегрировать в существующие проекты.
- Инструменты: Есть больше инструментов и редакторов, которые поддерживают SCSS.

Плюсы SCSS:

- Привычный синтаксис, близкий к CSS.
- Легкая интеграция в существующие проекты.
- Больше инструментов и редакторов, поддерживающих SCSS.

Минусы SCSS:

- Возможно, менее читаемый код из-за использования фигурных скобок и точек с запятой.

Вложенность и &

Вложенность в SCSS позволяет вам организовывать стили для вложенных элементов, делая код более читаемым и поддерживаемым. Когда вы используете вложенность в SCSS, вы можете обращаться к вложенным селекторам, определяя их в контексте родительского элемента.

Кроме того, символ & в SCSS используется для ссылки на родительский селектор. Это позволяет вам создавать более специфичные правила стилей, используя родительский селектор в комбинации с другими селекторами.

```scss
// SCSS
.container {
  width: 100%;

  .box {
    padding: 20px;
    background-color: #007bff;

    &:hover {
      background-color: #0056b3;
    }
  }
}

/* CSS */
.container {
  width: 100%;
}

.container .box {
  padding: 20px;
  background-color: #007bff;
}

.container .box:hover {
  background-color: #0056b3;
}
```

Пример с переменными

Задача 1: Создание переменной для основного цвета сайта и использование ее в нескольких местах стилевого файла.

```
// Определение переменной для основного цвета сайта
$primary-color: #007bff;

// Использование переменной в различных стилях
.header {
    background-color: $primary-color;
}

.button {
    background-color: $primary-color;
}

.footer {
    color: $primary-color;
}
```

Пример с миксинами

Задача 2: Создание миксина для стилизации кнопок с различными параметрами (цвет, размер, скругленные углы и т. д.) и применение его к нескольким кнопкам на сайте.

```
// Определение миксина для стилизации кнопок
@mixin button-styles($bg-color, $text-color, $padding, $border-radius) {
    background-color: $bg-color;
    color: $text-color;
    padding: $padding;
    border-radius: $border-radius;
    cursor: pointer;
}

// Применение миксина к кнопкам с различными параметрами
.button-primary {
    @include button-styles(#007bff, #fff, 10px 20px, 5px);
}

.button-secondary {
    @include button-styles(#6c757d, #fff, 15px 25px, 8px);
}
```

Переменные в SCSS

Переменные в SCSS предоставляют возможность определить и использовать именованные значения, которые могут быть повторно использованы в различных частях вашего стилевого файла. Вот основные различия между переменными в SCSS и переменными в обычном CSS:

Область видимости:

- В CSS переменные имеют глобальную область видимости. Они доступны в любом месте после их объявления.
- В SCSS переменные могут иметь локальную или глобальную область видимости в зависимости от того, где они были объявлены. Переменные, объявленные в пределах блока, будут доступны только в этом блоке и его вложенных блоках.

Динамическое обновление:

- Переменные в CSS являются статическими. Они не могут быть изменены во время выполнения.
- Переменные в SCSS могут быть динамически изменены во время выполнения, так как SCSS компилируется в CSS перед тем, как отправляться на сервер. Это позволяет использовать переменные для управления различными стилями и их изменением в зависимости от условий.

Удобство использования:

- Использование переменных в CSS требует повторного написания значения для каждого свойства, где они используются.
- Использование переменных в SCSS позволяет определить значение один раз и использовать его многократно по всему файлу стилей. Это упрощает поддержку и изменение стилей, так как достаточно изменить значение переменной, чтобы оно автоматически применилось ко всем местам, где эта переменная используется.
- Применение переменных в SCSS существенно улучшает организацию и поддерживаемость вашего кода стилей, делая его более гибким и легким для внесения изменений.

```scss
// Определение переменных
$primary-color: #007bff;
$secondary-color: #6c757d;

// Использование переменных
.header {
  background-color: $primary-color;
  color: white;
}

.footer {
  background-color: $secondary-color;
  color: black;
}
```

```css
.header {
  background-color: #007bff;
  color: white;
}

.footer {
  background-color: #6c757d;
  color: black;
}
```

Миксины в SCSS

Миксины в SCSS позволяют определить набор стилей, который можно повторно использовать в различных местах ваших стилей.

Миксины позволяют сократить дублирование кода и упростить поддержку стилей, позволяя использовать один и тот же набор правил в различных местах вашего проекта.

```scss
// Определение миксина
@mixin button-styles($bg-color, $text-color) {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  background-color: $bg-color;
  color: $text-color;
  cursor: pointer;

  &:hover {
    background-color: darken($bg-color, 10%);
  }
}

// Использование миксина
.button1 {
  @include button-styles(#007bff, #fff);
}

.button2 {
  @include button-styles(#6c757d, #000);
}
```

```css
.button1 {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  background-color: #007bff;
  color: #fff;
  cursor: pointer;
}

.button1:hover {
  background-color: #0056b3;
}

.button2 {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  background-color: #6c757d;
  color: #000;
  cursor: pointer;
}

.button2:hover {
  background-color: #5a6268;
}
```

Встроенные функции

`lighten($color, $amount)`: Увеличивает яркость цвета на определенный процент.

scss Copy code

```scss
$primary-color: #007bff;
.button {
  background-color: lighten($primary-color, 10%);
}
```

`darken($color, $amount)`: Уменьшает яркость цвета на определенный процент.

scss Copy code

```scss
$primary-color: #007bff;
.button {
  background-color: darken($primary-color, 10%);
}
```

`mix($color1, $color2, $weight)`: Смешивает два цвета в определенной пропорции.

scss Copy code

```scss
$color1: #007bff;
$color2: #6c757d;
.button {
  background-color: mix($color1, $color2, 50%);
}
```

`saturate($color, $amount)`: Увеличивает насыщенность цвета на определенный процент.

scss Copy code

```scss
$primary-color: #007bff;
.button {
  background-color: saturate($primary-color, 10%);
}
```

`desaturate($color, $amount)`: Уменьшает насыщенность цвета на определенный процент.

scss Copy code

```scss
$primary-color: #007bff;
.button {
  background-color: desaturate($primary-color, 10%);
}
```

Наследование @extend

Директива @extend в SCSS позволяет одному селектору наследовать стили другого селектора. Это очень полезная функция, которая помогает избежать дублирования кода и сделать стили более модульными и поддерживаемыми.

1. **Определение базового селектора**:
   Сначала определяется базовый селектор с определенными стилями:

```scss
.button {
  padding: 10px 20px;
  border: 1px solid #ccc;
  background-color: #eee;
}
```

2. **Использование `@extend`**:
   Затем вы можете создать другой селектор и использовать `@extend`, чтобы наследовать стили из базового селектора:

```scss
.special-button {
  @extend .button;
  color: #fff;
  background-color: #007bff;
}
```

   Это приведет к созданию нового селектора `.special-button`, который наследует все стили из `.button`, а также добавляет свои собственные стили.

3. **Результат**:
   После компиляции SCSS в CSS вы получите следующий результат:

```css
.button, .special-button {
  padding: 10px 20px;
  border: 1px solid #ccc;
  background-color: #eee;
}

.special-button {
  color: #fff;
  background-color: #007bff;
}
```

   Обратите внимание, что в результате получается один селектор с объединенными стилями обоих селекторов.

Импорты в SCSS @import

SCSS (Sassy CSS) - это метаязык каскадных таблиц стилей (CSS), который добавляет множество дополнительных возможностей к обычному CSS, таких как переменные, вложенные правила, миксины и многое другое.

- Контроль порядка: Порядок импорта важен, так как правила, объявленные позже, переопределяют правила, объявленные ранее. Поэтому важно убедиться, что файлы импортируются в нужном порядке.
- Использование переменных и миксинов из других файлов: После импорта переменные и миксины из других файлов становятся доступными в текущем файле.

@import директива: В SCSS для импорта других SCSS или CSS файлов используется директива @import. Она имеет два основных синтаксиса:

```
@import 'file'; // импорт файла file.scss
@import 'file.css'; // импорт файла file.css
```

Методологии в CSS

Популярные методологии в написании CSS

- БЭМ
- SMACSS
- OOCSS
- Atomic CSS

Методологии в CSS представляют собой наборы правил и подходов к организации и написанию кода стилей для веб-проектов. Они помогают разработчикам создавать более чистый, структурированный и поддерживаемый CSS-код.

- Структурированность: Методологии предоставляют определенные правила организации CSS-кода, что делает его более понятным и легко читаемым как для вас, так и для других разработчиков.
- Поддерживаемость: Чем более структурированным и чистым является CSS-код, тем проще поддерживать и вносить изменения в проект. Методологии помогают избежать спагетти-кода и создают основу для более эффективной поддержки.
- Масштабируемость: При разработке крупных веб-проектов важно иметь масштабируемый подход к написанию CSS-кода. Методологии предоставляют набор правил, которые помогают эффективно масштабировать стили и управлять ими даже при росте проекта.
- Повторное использование: Методологии способствуют созданию компонентного подхода к стилям, что позволяет повторно использовать стилизацию на различных частях сайта или в разных проектах.
- Коллективная работа: Если вы работаете в команде, методологии помогают обеспечить единообразие в написании CSS-кода и упрощают совместную работу над проектом.

БЭМ

BEM (Block Element Modifier) - это методология разработки веб-интерфейсов, предложенная компанией Яндекс. Её основная идея заключается в создании модульных и повторно используемых компонентов интерфейса, которые легко читаемы и поддерживаемы.

Блоки (Block): Определяют независимые компоненты интерфейса, которые могут быть повторно использованы. Каждый блок имеет уникальное имя.

Элементы (Element): Части блоков, которые не могут быть использованы вне своего контекста. Имена элементов привязаны к блокам и разделяются двумя подчеркиваниями __.

Модификаторы (Modifier): Изменения внешнего вида или поведения блока или элемента. Модификаторы применяются с помощью двойного дефиса – (иногда с помощью одного подчеркивания _).

БЭМ плюсы и минусы + пример

Преимущества BEM:

- Читаемость кода: Благодаря ясным именам классов код становится легко читаемым и понятным.
- Масштабируемость: Компонентный подход позволяет создавать модули, которые легко масштабируются и поддерживаются.
- Повторное использование: Компоненты BEM можно использовать многократно на разных частях сайта без изменений.
- Уменьшение конфликтов стилей: Изоляция стилей блоков и элементов снижает вероятность конфликтов и улучшает управление CSS-кодом.

Недостатки BEM:

- Длинные имена классов: Полное имя класса BEM может быть длинным, особенно когда используются модификаторы, что может привести к увеличению объема CSS-кода.
- Накладные расходы: Добавление классов BEM может потребовать больше времени и усилий при разработке, особенно на начальных этапах проекта.

```css
.block {
    background-color: #f0f0f0;
    padding: 10px;
}

.block__element {
    font-weight: bold;
}

.block__element--modifier {
    color: red;
}
```

```html
<div class="block">
    <div class="block__element">Элемент</div>
    <div class="block__element block__element--modifier">Модификатор элемента</div>
</div>
```

SMACSS

SMACSS (Scalable and Modular Architecture for CSS) - это методология организации CSS, разработанная Джоном Элесси (Jonathan Snook). Основная идея SMACSS заключается в разделении стилей на пять категорий, каждая из которых имеет свою роль в организации и поддержке кода.

Вот эти пять категорий SMACSS:

- Base (Базовые стили): Эта категория включает в себя базовые стили, такие как шрифты, цвета, стандартные стили для элементов HTML (например, body, a, p и т.д.).
- Layout (Разметка): Здесь определяются стили для структуры страницы, такие как сетка, расположение блоков и общая компоновка.
- Modules (Модули): Этот раздел включает стили для независимых компонентов, которые можно повторно использовать на разных частях сайта.
- State (Состояния): Здесь определяются стили, которые изменяются в зависимости от состояния элемента (например, :hover, :active, :focus).
- Theme (Тема): Этот раздел содержит стили, которые определяют внешний вид, специфичный для конкретной темы или стилизации.

SMACSS плюсы и минусы + пример

Плюсы SMACSS:

- Структурированность: Помогает организовать CSS-код в логические блоки, что облегчает его понимание и поддержку.
- Повторное использование: Модульный подход позволяет повторно использовать компоненты стилей, что экономит время и уменьшает объем кода.
- Масштабируемость: Позволяет легко добавлять новые функциональности и изменять стили без нарушения целостности проекта.

Минусы SMACSS:

- Излишняя сложность для небольших проектов: Для небольших проектов использование всех пяти категорий может показаться излишним.
- Необходимость обучения команды: Внедрение новой методологии требует времени и усилий для обучения команды.

```css
/* Base styles */
body {
    font-family: Arial, sans-serif;
    font-size: 16px;
    color: #333;
}

/* Layout styles */
.container {
    width: 960px;
    margin: 0 auto;
}

/* Module styles */
.button {
    display: inline-block;
    padding: 10px 20px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 5px;
}

/* State styles */
.button:hover {
    background-color: #0056b3;
}

/* Theme styles */
.dark-theme {
    background-color: #222;
    color: #fff;
}
```

OOCSS

Object-Oriented CSS (OOCSS) - это методология стилей, разработанная Николасом Галлиасом (Nicolas Gallagher) и Джеймсом Бенеттом (Jonathan Neal). Она базируется на принципах объектно-ориентированного программирования и призвана увеличить переиспользуемость и гибкость CSS-кода.

Основные принципы OOCSS:

- Разделение структуры и внешнего вида: Основная идея OOCSS заключается в разделении структуры HTML и её визуального оформления. Это достигается путем создания независимых объектов (objects) и их состояний (states), которые могут быть применены к различным элементам HTML.
- Повторное использование кода: OOCSS поощряет создание маленьких, независимых компонентов стилей, которые могут быть многократно использованы на разных частях сайта.
- Принцип сепарации структуры и скинов (поведения): Стили разделяются на две категории: скины (skins) и поведение (behavior). Скины отвечают за внешний вид элемента, а поведение - за его интерактивное поведение.

OOCSS плюсы и минусы + пример

Преимущества OOCSS:

- Переиспользуемость: Позволяет создавать независимые компоненты стилей, которые легко переносить между различными проектами.
- Гибкость: Позволяет легко изменять внешний вид элементов, не затрагивая их структуру, и наоборот.
- Увеличение производительности: Меньший объем CSS-кода и повторное использование стилей уменьшают время загрузки страницы и упрощают её поддержку.

Недостатки OOCSS:

- Сложность внедрения: Внедрение OOCSS может потребовать времени и усилий для обучения команды и перехода от привычных методов написания стилей.
- Возможность конфликтов стилей: Если не следовать правильному разделению стилей, это может привести к конфликтам и непредсказуемому поведению элементов.

```
/* Object styles */
.button {
    display: inline-block;
    padding: 10px 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

/* Skin styles */
.button-primary {
    background-color: #007bff;
    color: #fff;
}

.button-secondary {
    background-color: #6c757d;
    color: #fff;
}

/* Behavior styles */
.button:hover {
    background-color: #0056b3;
}
```

Atomic CSS

Atomic CSS - это методология написания CSS, при которой каждое свойство стиля представлено отдельным классом. В отличие от традиционного подхода, где создаются классы для определенных компонентов или стилей, Atomic CSS разбивает стили на мельчайшие кирпичики (атомы), которые можно комбинировать для быстрой стилизации элементов.

Основные принципы Atomic CSS:

- Мелкие классы: Каждый класс отвечает за одно конкретное свойство стиля, например, mt-4 для отступа сверху, bg-blue для цвета фона и т.д.
- Комбинируемость: Классы можно комбинировать, чтобы создавать нужные стили. Например, mt-4 + bg-blue применяет отступ сверху и цвет фона к элементу.
- Быстрая разработка: Atomic CSS позволяет быстро создавать стили без необходимости писать новые классы или править существующие. Это особенно удобно при работе с большим количеством повторяющихся элементов.

Atomic CSS плюсы и минусы + пример

Преимущества Atomic CSS:

- Большая гибкость: Можно легко создавать различные комбинации стилей без необходимости создания новых классов.
- Быстрота разработки: Позволяет быстро стилизовать элементы без написания дополнительного CSS-кода.
- Меньший объем CSS: Поскольку Atomic CSS использует только необходимые свойства стилей, он обычно имеет более компактный размер.

Недостатки Atomic CSS:

- Читаемость кода: Из-за большого количества классов и их комбинаций код может стать менее читаемым и поддерживаемым.
- Нарушение семантики: Использование Atomic CSS может привести к нарушению семантики HTML, так как классы могут быть очень специфичными и не понятными для других разработчиков.

```css
/* Atomic CSS classes */
.mt-4 {
    margin-top: 1rem;
}

.bg-blue {
    background-color: blue;
}

.text-white {
    color: white;
}

.p-2 {
    padding: 0.5rem;
}
```

```html
<div class="mt-4 bg-blue text-white p-2">
    Этот блок имеет отступ сверху, синий фон, белый текст и внутренние отступы.
</div>
```

БЭМ наиболее популярный

БЭМ (Block-Element-Modifier) является одной из самых популярных методологий веб-разработки по нескольким причинам:

- Четкая структура и нейминг: БЭМ предлагает понятную и последовательную систему именования классов для элементов веб-страницы, что делает код более понятным и легко читаемым как для разработчиков, так и для других участников команды.
- Масштабируемость: БЭМ позволяет создавать масштабируемые и поддерживаемые проекты. Благодаря своей модульной структуре, БЭМ упрощает добавление нового функционала и изменение существующего без значительного воздействия на другие части кода.
- Повторное использование кода: БЭМ способствует повторному использованию кода блоков и элементов, что снижает вероятность ошибок и ускоряет разработку.
- Компонентный подход: БЭМ подходит для создания компонентов, которые могут быть легко переносимы и встраиваемы в другие проекты.
- Активное сообщество: БЭМ имеет активное сообщество разработчиков, которые создают инструменты и решения для улучшения разработки на основе этой методологии.
- Поддержка инструментов: Существует множество инструментов, таких как препроцессоры CSS и сборщики проектов, которые интегрируют поддержку БЭМ, что облегчает разработку.
- Гибкость: БЭМ можно адаптировать под различные требования проекта, изменяя некоторые аспекты методологии в зависимости от конкретной ситуации.
- Эти факторы делают БЭМ привлекательным выбором для многих веб-разработчиков и компаний, которые ценят чистый, организованный и масштабируемый код.

БЭМ + SCSS

Связка БЭМ и SCSS имеет несколько преимуществ:

- Читаемость кода: Использование БЭМ упрощает структурирование CSS классов, делая код более понятным и легко читаемым. SCSS добавляет еще большую читаемость и удобство с помощью возможности использования вложенности и переменных.
- Масштабируемость и поддерживаемость: Методология БЭМ и SCSS способствуют созданию кода, который легко масштабируется и поддерживается. Правильно организованный код благоприятно влияет на дальнейшую разработку и сопровождение проекта.
- Повторное использование стилей: Благодаря БЭМ и SCSS можно легко создавать модульные компоненты стилей, которые могут быть повторно использованы в различных частях проекта. Это уменьшает дублирование кода и облегчает его поддержку.
- Удобство разработки: SCSS предоставляет удобные инструменты для разработчиков, такие как вложенные правила, миксины, переменные и т.д., что упрощает написание и организацию CSS кода.
- Гибкость и расширяемость: SCSS позволяет создавать дополнительные инструменты и шаблоны стилей, которые могут быть переиспользованы в разных проектах, что способствует увеличению производительности и эффективности разработки.
- В целом, связка БЭМ и SCSS обеспечивает эффективный и гибкий подход к разработке веб-интерфейсов, позволяя создавать чистый, структурированный и легко поддерживаемый код стилей.

```scss
// Общие стили для кнопок
.button {
  border: none;
  border-radius: 4px;
  cursor: pointer;
  padding: 10px 20px;
  font-size: 16px;
  transition: background-color 0.3s ease;

  &--primary {
    background-color: #007bff;
    color: #fff;

    &:hover {
      background-color: darken(#007bff, 10%);
    }
  }

  &--secondary {
    background-color: #6c757d;
    color: #fff;

    &:hover {
      background-color: darken(#6c757d, 10%);
    }
  }

  &--disabled {
    background-color: #ccc;
    color: #888;
    cursor: not-allowed;
  }
}
```

Ресурсы

- SASS (SCSS) документация - ТЫК
- LESS документация - ТЫК
- Stylus документация - ТЫК
- SASS (SCSS) на русском - ТЫК
- БЭМ документация - ТЫК
- SMACSS (методология) документация - ТЫК