Семинар 1

CSS - Каскадные таблицы стилей

Преподаватель:
Горев Данил Максимович
Разработчик образовательных проектов VK
(В том числе VK Education!)

Для связи - https://ketovx.t.me
Почта - danil.gorev@vk.team

Структура занятия

1. Вспомним что такое HTML и HTTP
2. Узнаем о потребности оформления сайтов
3. Разберёмся, что такое CSS
4. Научимся использовать CSS в связке с HTML
5. Поговорим о пре-процессорах SCSS/LESS
6. Посмотрим на инструменты упрощающие разработку
7. Перейдём к разбору Домашнего задания и практике программирования

Что такое HTML и HTTP?

- **HTML** - Язык текстовой разметки, возникший как необходимость добавить контенту в сети свойство "*гипермедийности*", т.е. возможность изменяться и обновляться в зависимости от потребностей пользователя и определять свойства собственного содержимого;
- **HTTP** - Протокол, описывающий правила передачи контента, включая HTML-страницы, файлы каскадных стилей, скрипты и медиа-файлы (изображения, видео).

Крохотный пример HTML

```
<!DOCTYPE html>
<html>
<body>
        <nav>
                <a href="/">Главная</a>
                <a href="/me">Профиль</a>
        </nav>
        <main>
                <h1>My First Heading</h1>
                <p>My first paragraph.</p>
        </main>
</body>
</html>
```

Почему нам так нужен CSS?

CSS - не столько язык или инструмент, сколько стандарт, определяющий способы оформления HTML документов.

Как вы понимаете, реализовывать стандарт, следовать ли ему в точности или вводить в него собственные альтернативные инструменты - задача, которую выполняют браузеры (встроенные в них движки отрисовки HTML + CSS контента).

CSS возник из двух основных потребностей

1. Создать способы выделения важного контента в HTML вёрстке, например добавить к тексту фоновый цвет или изменить его расположение на странице;
2. Добавить к HTML переиспользуемый инструмент оформления, который был бы обобщённо стандартизирован.

На данный момент мы пользуемся уже третьей версией CSS, в которой нам доступны не только способы оформления, но и логика анимаций, управления Layout'ами (Flex и Grid), и даже переменные.

Основы синтаксиса: селекторы, свойства и значения

1. Селектор - правило, по которому будут применяться CSS-свойства к элементам на странице;
2. Свойства - описательные характеристики, влияющие на внешний вид выбранных селекторами элементов;
3. Значения свойств - набор доступных для каждого свойства входных значений, например, цвет может принимать значения: white, #fff, rgb(255, 255, 255).

Пример синтаксиса CSS

```
h1 {
    color: navy; /* <- цвет текста */
    font-size: 32px; /* <- размер текста */
    margin-bottom: 20px; /* <- отступ сверху */
}
```

На этом примере `h1` - это селектор, `color`, `font-size` и `margin-bottom` - свойства, а `navy`, `32px` и `20px` - значения соответствующих свойств.

Обязательно за каждым селектором следуют открывающие фигурные скобки, обьединяющие набор свойств, который будет применён к элементам.

Сердце CSS — Селекторы

Селектор определяет, к каким элементам HTML будут применены стили. Основные типы селекторов:

```
p { } /* Селектор тега, стилизует все теги `<p>` */
/* <p>...</p> */

.header { } // Селектор класса, частый и универсальный способ
/* <div class="header">...</div> */

#main-content { } // Селектор по уникальному id в HTML
/* <div id="main-content">...</div> */
```

Селекторы также можно комбинировать и усложнять, например, перечислить через запятую или указать, что свойства нужно применить только к дочерним элементам выбранного селектора.

```
h1, h2, h3 {...} /* Несколько селекторов через запятую */

div p {...}
/* Выберет все <p> внутри <div> элементов, даже вложенные в другие теги */

div > p {...}
/* Выберет только те <p>, родительским элементом которых является <div> */

input[type="text"] {...}
/* Выберет все элементы <input type='text'/ > */
```

Свойства и Значения

1. **Свойство** — это аспект внешнего вида, который вы хотите изменить.
2. **Значение** — это конкретная настройка для этого свойства.

Примеры популярных свойств:

- `color` — цвет текста.
- `background-color` — цвет фона.
- `font-size` — размер шрифта.
- `margin` — внешний отступ.
- `padding` — внутренний отступ.

Практический пример "Вживую"

```html
<button class="my-button">Нажать меня</button>
```

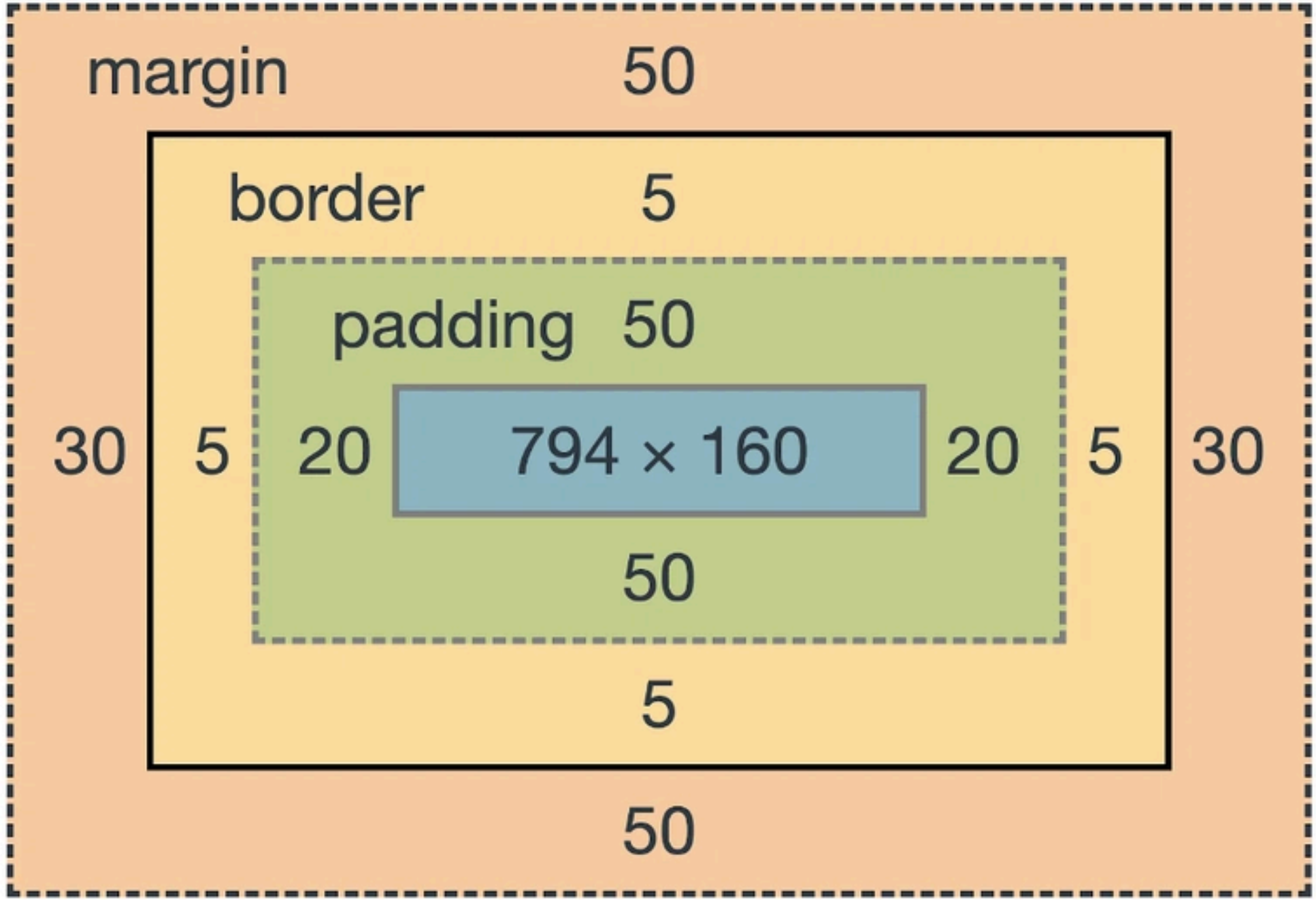
```css
.my-button {
    background-color: #4CAF50; /* Зеленый фон */
    color: white;              /* Белый текст */
    padding: 10px 20px;        /* Внутренние отступы */
    border: none;              /* Убираем рамку */
    border-radius: 5px;        /* Закругляем углы */
    font-size: 16px;           /* Размер текста */
    cursor: pointer;           /* Курсор-указатель при наведении */
}
```

Нажать меня

Что такое Блочная модель (Box Model)?

Каждый элемент в HTML — это прямоугольный бокс. Блочная модель описывает, как вычисляются его размеры.

1. **Контент (Content)** - Ядро блока, его внутреннее содержимое. Основные свойства CSS - `width` и `height`;
2. **Внутренний отступ (Padding)** - прозрачная область вокруг контента. Свойство - `padding`;
3. **Граница (Border)** - Линия, окружающая паддинг. Свойство - `border`;
4. **Внешний отступ (Margin)** - Прозрачная область снаружи границы. Свойство - `margin`.


margin
50
border
5
padding
50
30
5
20
794 × 160
20
5
30
50
5
50

Волшебное свойство `box-sizing`

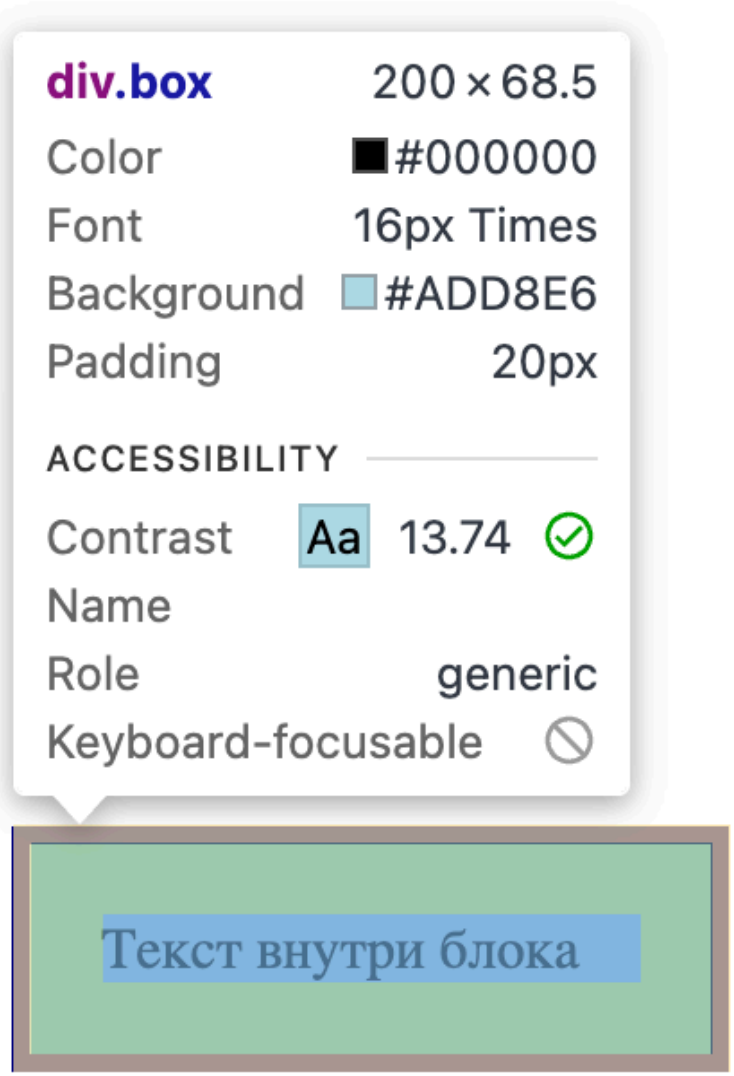
Проблема: Сложно предсказать итоговый размер блока, если постоянно учитывать паддинги и границы.
Решение: Воспользоваться особым значение свойства box-sizing!

- `content-box` **(значение по умолчанию)**
  - width и height = только контент.
  - paddign и border добавляются сверху
- `border-box` **(рекомендуемое значение!)**
  - width и height = контент + паддинг + бордер.

```
<...>
<div class="box">Текст внутри блока</div>
<...>
```

```
/* Лучше задавать сразу для всех элементов */
* {
  box-sizing: border-box;
}

.box {
  background: red;
  width: 200px; /* Теперь это ОБЩАЯ ширина блока */
  padding: 20px; /* Паддинг "внутри" width */
  border: 5px solid navy; /* Бордер тоже "внутри" width */
}
```


div.box
200 × 68.5
Color
#000000
Font
16px Times
Background
#ADD8E6
Padding
20px
ACCESSIBILITY
Contrast
Aa
13.74
Name
Role
generic
Keyboard-focusable
Текст внутри блока

Позиционирование элементов

Элементы на странице можно сдвигать - относительно границ родительского элемента или относительно границ окна.

Для этого существуют CSS-свойства `top`, `left`, `right` и `bottom`, задающие отступы в пикселях, например `20px` или даже `-20px`.

Но помимо этого можно управлять наложением элементов друг на друга, задавая свойство `z-index` - чем он выше, тем выше будет элемент относительно других на странице.

Правда всё это не будет работать без свойства `position`...

CSS-Свойство `position`

- `static` (по умолчанию) - запрещает влиять на положение элемента
- `relative` - позволяет смещать элемент относительно границ родительского блока
- `absolute` - позволяет остальным элементам игнорировать наличие элемента, позиционируя его в зависимости от свойства `position` родителя
- `fixed` - позиционирует обьект относительно окна браузера
- `sticky` - Гибрид `relative` и `fixed`, меняющий поведение в зависимости от прокрутки страницы

Практический пример позиционирования

Возьмём следующий HTML. Наша задача сделать так, чтобы навигация - всегда была вверху страницы, а кнопка "Чат" всегда оставалась на странице справа снизу.

```
<body>
        <header class="header">Навигация...</header>
        <main class="container">... основной контент ...</main>
        <button class="chat-button">Чат</button>
</body>
```

Мы намеренно не будем изменять свойства блока с контентом, он будет отображаться по умолчанию.

```
.header {
  position: sticky; /* "Прилипнет" при прокрутке */
  top: 0; /* Начнет прилипать, достигнув верха окна */
  background: lightblue;
  padding: 10px;
}

.chat-button {
  position: fixed; /* Всегда на одном месте экрана */
  bottom: 20px;   /* 20px от низа */
  right: 20px;    /* 20px от правого края */
}
```

Навигация...

... основной контент ...

Особенная кнопка

Свойство `display` - основа потоков

`display` — говорит браузеру, *как отображать элемент*.
Ключевые значения:

- `display: block` (блочное) - Элемент займёт всю доступную ширину начиная с новой строки. Примеры: `<div>, <p>, <h1>, <section>`;
- `display: inline` (строчное) - Элемент занимает только необходимую ширину без переноса на новую строку. Примеры: `<span>, <a>, <strong>`;
- `display: inline-block` (строчно-блочное) - не переносит строку, но позволяет задать `width/height/padding/margin`;

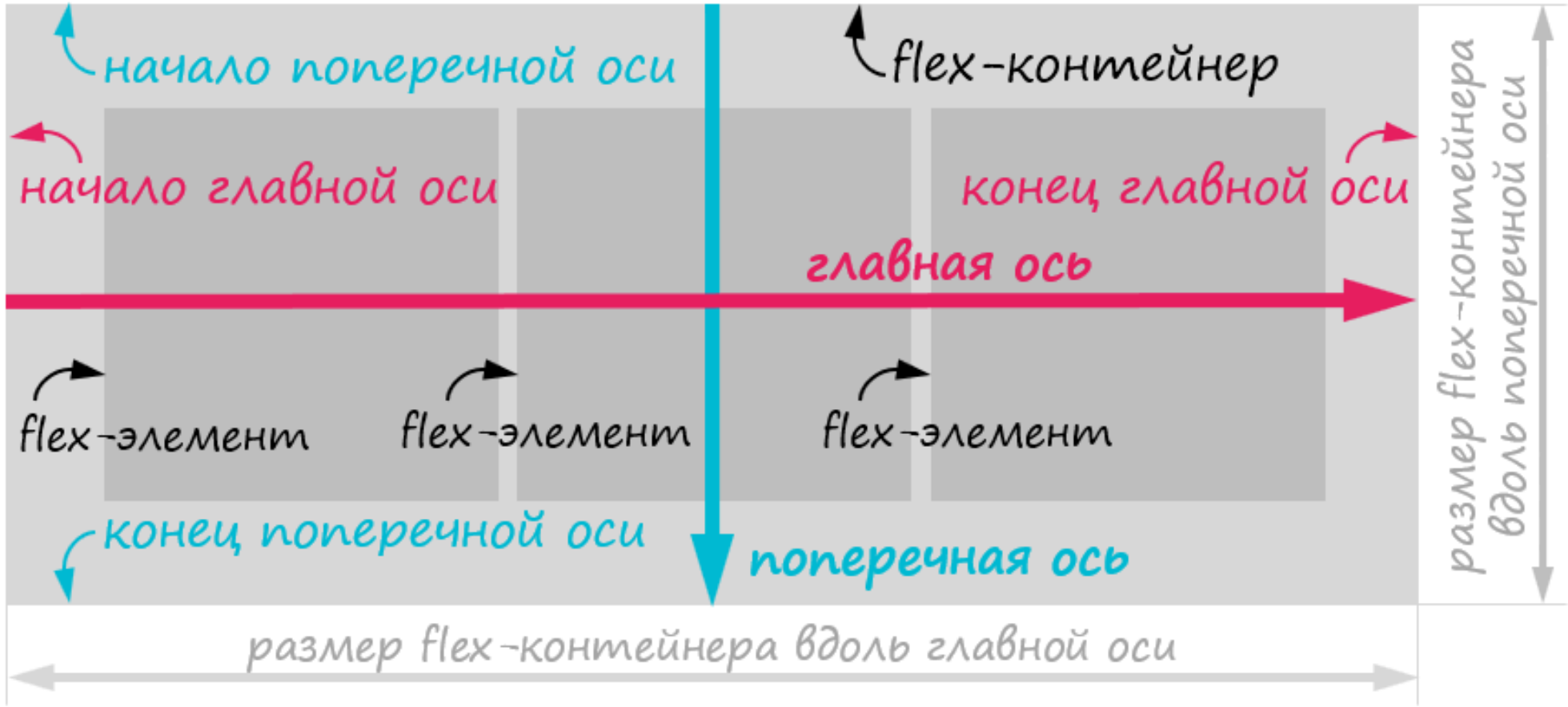
Новая эра: `display: flex`

Flexbox предназначен для одномерного расположения элементов — в строку или колонку.

Основные концепции:

- Flex-контейнер — элемент с `display: flex`
- Flex-элементы — прямые потомки контейнера
- Основная ось (main-axis) — направление расположения элементов
- Поперечная ось (cross-axis) — перпендикулярное направление

Пример схемы *main-axis/cross-axis*



Практика Flexbox

Задача: разработать панель навигации для сайта.

```
<div class="navbar">
  <div class="logo">Лого</div>
  <nav class="menu">
    <a href="#">Главная</a>
    <a href="#">О нас</a>
    <a href="#">Контакты</a>
  </nav>
  <button class="login">Войти</button>
</div>
```

```css
.navbar {
  display: flex;
  justify-content: space-between; /* Лого слева, кнопка справа */
  align-items: center; /* Все по центру по вертикали */
  padding: 1rem;
  background: #f8f9fa;
  gap: 8px; /* Промежутки между Flex-элементами */
}

.menu {
  display: flex;
  gap: 2rem; /* Простой способ сделать отступы между ссылками */
}

.login {
  margin-left: auto; /* "Прижимает" кнопку к правому краю */
}
```

Лого Главная О нас Контакты Войти

Снова новая эра: `display: grid`

Grid создает сетку для точного позиционирования по горизонтали и вертикали одновременно.

Основные концепции:

- Grid-контейнер — элемент с `display: grid`
- Ячейки (Grid Cells) — минимальные единицы сетки
- Линии (Grid Lines) — линии, разделяющие сетку
- Области (Grid Areas) — именованные области сетки

Практика Grid

Задача: Сверстать страницу с шапкой, основным контентом, сайдбаром и футером.

```
<div class="layout">
  <header>Шапка</header>
  <aside>Сайдбар</aside>
  <main>Основной контент</main>
  <footer>Подвал</footer>
</div>
```

```css
.layout {
  display: grid;
  grid-template-areas:
    "header header"
    "sidebar content"
    "footer footer";
  grid-template-columns: 200px 1fr;
  grid-template-rows: auto 1fr auto;
  height: 100vh;
  gap: 1rem;
}

header { grid-area: header; }
aside { grid-area: sidebar; }
main { grid-area: content; }
footer { grid-area: footer; }
```

Когда использовать Flexbox?

Используйте Flexbox для:

- Одномерного размещения (строка ИЛИ колонка)
- Выравнивания элементов внутри контейнера
- Навигационных панелей, карточек в ряду
- Простых центрирований

Или когда вам это удобно...

Когда использовать Grid?

Используйте Grid для:

- Сложных двумерных макетов (и строки, и колонки)
- Макетов всего сайта (шапка, сайдбар, контент, подвал)
- Когда нужен точный контроль над обеими осями
- Сеток карточек, галерей

Или когда вам это удобно...

Разработка под разные устройства

Проблема: Один и тот же сайт просматривают на экранах с разным разрешением:

- Десктопы: 1920px и больше
- Ноутбуки: 1366px - 1440px
- Планшеты: 768px
- Смартфоны: 320px - 425px

Вопрос: Как сделать так, чтобы сайт хорошо выглядел на ВСЕХ этих устройствах?

Отзывчивая верстка (Responsive Web Design)

Ключевая идея: Макет плавно подстраивается под любую ширину экрана.

Основные инструменты:

- Flexbox и Grid в сочетании с `max-width` и `max-height`;
- Использование процентного масштабирования вместо пикселей: `height: 100%` а не `1080px`;
- Медиа-запросы, которые применяют CSS-правила только при определённом разрешении экрана;

```css
.container {
  width: 80%; /* Контейнер займет 80% от ширины родителя */
  margin: 0 auto;
}

.sidebar {
  float: left;
  width: 25%; /* Сайдбар всегда 25%, независимо от экрана */
}

/* При ширине экрана меньше 768px изменим layout */
@media (max-width: 768px) {
  .sidebar {
    float: none;
    width: 100%;
  }
}
```

Практика

Задача: разработать макет карточек товаров, которые отображались бы списком на компьютере (desktop) и столбцом на мобильном устройстве (mobile).

```
<...>
        <div class="container">
                <div class="card">...</div>
                <div class="card">...</div>
                <div class="card">...</div>
        </div>
<...>
```

Карточка товара 1

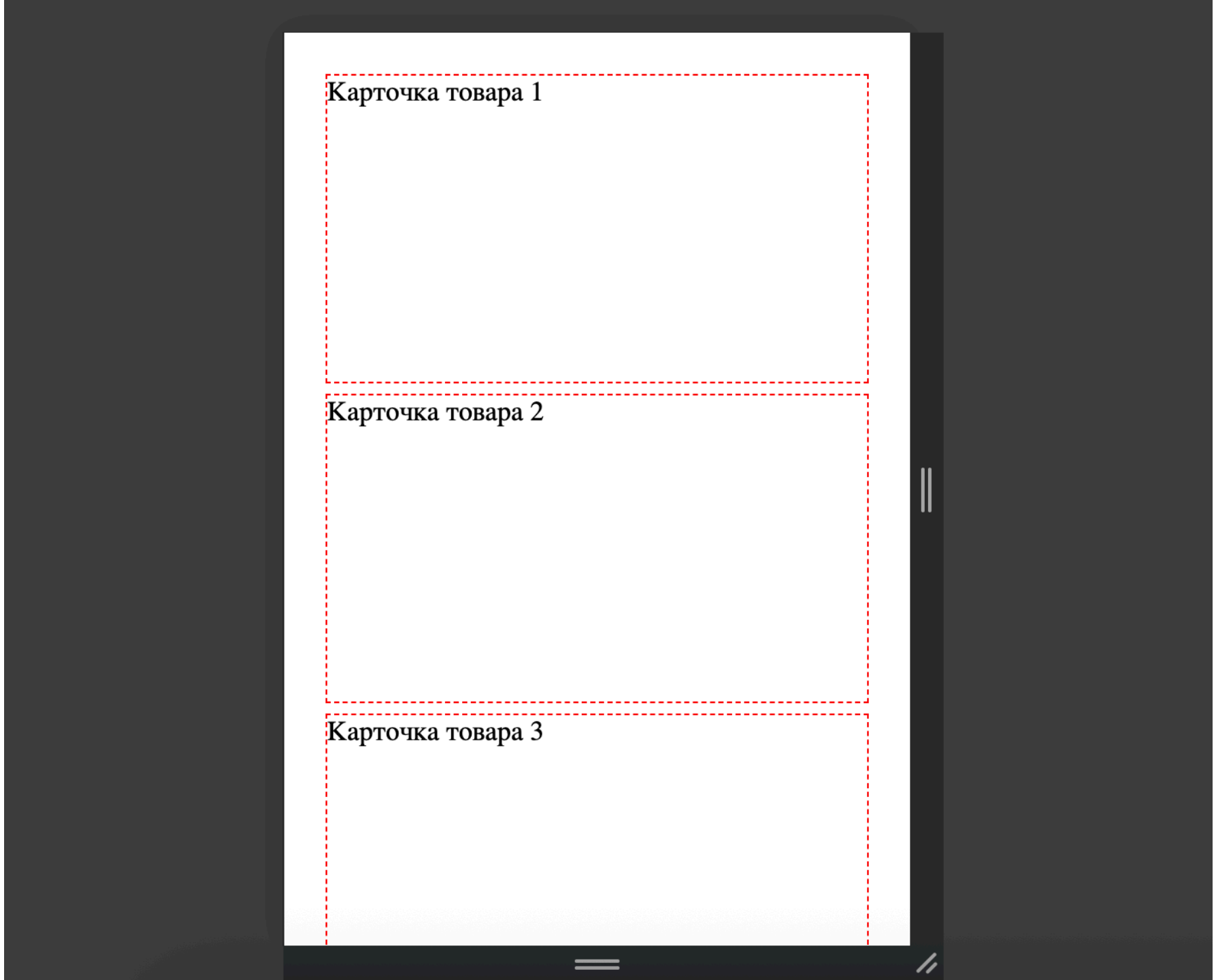
Карточка товара 2

Карточка товара 3

Карточка товара 4

Карточка товара 5

Карточка товара 6


Карточка товара 1
Карточка товара 2
Карточка товара 3

Ограничения CSS

- **Повторение кода:** Одинаковые цвета, шрифты, значения приходится копировать;
- **Нет вложенности:** Приходится постоянно повторять селекторы;
- **Отсутствие логики:** Неудобные переменные, нет условий, циклов, функций;
- **Сложность организации:** Большие CSS-файлы трудно поддерживать;

Решение: Препроцессоры!

Что такое препроцессор?

Препроцессор — это надстройка над CSS, которая добавляет возможности, отсутствующие в чистом CSS.
Как это работает: Код написанный на синтаксисе препроцессора транспилируется в сжатый CSS (что ускоряет его загрузку) и вставляется в ваш HTML.

Популярные препроцессоры:

- SASS/SCSS (самый популярный)
- Less
- Stylus

Основные возможности SASS/SCSS

Переменные (Variables)

```scss
$primary-color: #2ecc71;
$font-stack: 'Arial', sans-serif;
$spacing: 20px;

.header {
  color: $primary-color;
  font-family: $font-stack;
  margin-bottom: $spacing;
}
```

Вложенность (Nesting)

```
.nav {
  background: #333;

  ul {
    list-style: none;
    padding: 0;
  }

  li {
    display: inline-block;
  }

  a {
    color: white;
    text-decoration: none;

    &:hover {
      text-decoration: underline;
    }
  }
}
```

Миксины (Mixins) - переиспользуемые блоки стилей

```scss
@mixin flex-center {
  display: flex;
  justify-content: center;
  align-items: center;
}

@mixin button-style($bg-color) {
  background: $bg-color;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;

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

.container {
  @include flex-center;
}

.btn-primary {
  @include button-style(#2ecc71);
}
```

Импорт файлов (Import)

```scss
// main.scss
@import 'variables';
@import 'mixins';
@import 'components/button';
@import 'components/header';
```

Как подключить SASS/SCSS в проект?

Есть несколько вариантов использования препроцессоров:

- Через Node.js: `sass input.scss output.css`;
- Через плагин VSCode Live Sass Compiler (VS Code);
- Через сборщик: Webpack, Gulp, Parcel.

Лучший вариант - через Node.js, что позволит вам не волноваться о дальнейших конфликтах между SCSS, другими расширениями CSS и сборщиками пакетов.

CSS-фреймворки: Bootstrap, Tailwind

Каждый раз писать CSS для визуально схожих элементов бывает проблематично - кодовая база растёт, время кончается, заказчики требуют!
Решение: воспользоваться CSS-фреймворком.

Существует два ключевых подхода:

- **Компонентный** (Bootstrap) — готовые "блоки"
- **Утилитный** (Tailwind) — готовые "кирпичики"

Bootstrap — компонентный подход

Что предлагает:

- 🎨 Готовые UI-компоненты (кнопки, карточки, навигация)
- 📐 Мощная сеточная система (12 колонок)
- 📱 Встроенная адаптивность
- 🛠️ JavaScript-виджеты (модальные окна, карусели)

Плюсы: Быстрый старт, единый дизайн, много готового
Минусы: Похожие сайты, сложная кастомизация

Пример

```
<!-- Подключение через CDN -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

<!-- Готовая кнопка -->
<button class="btn btn-primary btn-lg">Кнопка</button>

<!-- Готовая карточка -->
<div class="card" style="width: 18rem;">
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">Заголовок карточки</h5>
    <p class="card-text">Текст карточки</p>
    <a href="#" class="btn btn-primary">Перейти</a>
  </div>
</div>
```

Tailwind CSS — утилитный подход

Что предлагает:

- 🎯 Тысячи утилитарных классов
- 🎨 Полная кастомизация через конфиг
- 🚀 Минимальный CSS на выходе
- 🔧 Полный контроль над дизайном

Плюсы: Полная свобода дизайна, легко кастомизировать
Минусы: Долгий набор классов, крутая кривая обучения

Пример

```
<!-- Подключение через CDN -->
<script src="https://cdn.tailwindcss.com"></script>

<!-- Собираем кнопку из утилит -->
<button class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded">
  Кнопка
</button>

<!-- Собираем карточку -->
<div class="max-w-sm rounded overflow-hidden shadow-lg">
  <img class="w-full" src="..." alt="...">
  <div class="px-6 py-4">
    <div class="font-bold text-xl mb-2">Заголовок карточки</div>
    <p class="text-gray-700 text-base">Текст карточки</p>
  </div>
</div>
```

Сравнение Bootstrap vs Tailwind

Когда использовать Bootstrap:

- Быстрый прототип или MVP
- Корпоративные/админ-панели
- Если не нужен уникальный дизайн

Когда использовать Tailwind:

- Уникальный, кастомный дизайн
- Компонентный подход (React/Vue)
- Долгосрочные проекты

Материалы для самоподготовки

Карта изучения CSS - https://roadmap.sh/r/css-6nqag
Сборник инструкций по HTML+CSS - https://htmlbook.ru/

Тренажёр изучения Flexbox - https://flexboxfroggy.com/#ru
Такой же тренажёр для Grid - https://cssgridgarden.com/#ru
Тренажёр по CSS-селекторам - https://flukeout.github.io/

CSS Инструменты:
Foundation CSS - https://get.foundation/
Bulma CSS - https://bulma.io/

Итог

CSS - не один инструмент, а стандарт, который постоянно пополняется, что требует от разработчика постоянного изучения его нововведений.

Чтобы упростить и ускорить разработку мы пользуемся множеством инструментов, от сборщиков (Webpack/Gulp) и препроцессоров (SCSS/LESS), до полноценных UI-фреймворков (Bootstrap, Foundation, Tailwind).

Knowledge deploy completed!

Преподаватель:
Горев Данил Максимович
Разработчик образовательных проектов VK

Для связи - https://ketovx.t.me
Почта - danil.gorev@vk.team

Домашнее задание - https://github.com/ziontab/tp-tasks/blob/master/files/markdown/task-1.md