HTML

Структура и семантика

Занятие 2

Frontend Course 2025

Что такое HTML?

> **Вопрос:** HTML это язык программирования?

Нет! HTML — язык разметки. Он говорит браузеру: "Это заголовок, это абзац, это ссылка". Как разметка в текстовом редакторе, только для веба.

HTML5 vs старые версии

> **Вопрос:** Чем HTML5 лучше предыдущих версий?

- Семантические теги (header, nav, main)
- Новые типы input (email, date, range)
- Встроенная валидация форм
- Поддержка мультимедиа (video, audio)
- Лучшая доступность

Структура HTML документа

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Моя страница</title>
  </head>
  <body>
    <!-- Контент здесь -->
  </body>
</html>
```

DOCTYPE

> **Вопрос:** Что будет, если забыть DOCTYPE?

Браузер включит "режим совместимости" и будет вести себя непредсказуемо. Как старый компьютер — работает, но не так как надо.

`<!DOCTYPE html>` говорит: "Привет, я современный HTML5!"

Атрибут lang

> **Вопрос:** Зачем браузеру знать язык страницы?

- Программы чтения с экрана произносят правильно
- Переводчики работают корректно
- Поисковики понимают аудиторию
- Автокоррекция в формах работает лучше

```
<html lang="ru">
  <!-- Русский -->
  <html lang="en">
    <!-- Английский -->
  </html>
</html>
```

Meta viewport

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

> **Вопрос:** Что происходит без этого тега?

На мобильном ваш сайт будет выглядеть как уменьшенная копия десктопной версии. Пользователи будут зумить и ругаться.

Open Graph

```
<meta property="og:title" content="Крутая статья" />
<meta property="og:description" content="Описание" />
<meta property="og:image" content="image.jpg" />
```

> **Вопрос:** Что это дает?

Красивые превью при репосте в Facebook, ВК, Telegram. Без этого — грустная ссылка без картинки.

Семантика vs div-суп

> **Вопрос:** В чем разница между div и section?

- **div** = "просто контейнер, без смысла"
- **section** = "тематический раздел контента"
- Браузеры, поисковики и скринридеры понимают разницу
- SEO любит семантику

Структура сайта

```
<header>    <!-- Шапка -->
<nav>       <!-- Навигация -->
<main>      <!-- Основной контент -->
  <section> <!-- Тематические разделы -->
  <article> <!-- Самостоятельные статьи -->
  <aside>   <!-- Боковая панель -->
</main>
<footer>    <!-- Подвал -->
```

Header

> **Вопрос:** Что должно быть в header?

- Логотип или название сайта
- Главное меню навигации
- Поиск (если есть)
- Контактная информация

НЕ нужно: футер, основной контент

Navigation

> **Вопрос:** Сколько nav можно на странице?

Сколько угодно! Главное меню, меню в футере, хлебные крошки, пагинация — всё это nav.

```
<nav aria-label="Главное меню">
  <nav aria-label="Хлебные крошки">
    <nav aria-label="Пагинация"></nav>
  </nav>
</nav>
```

Main

> **Вопрос:** Можно ли несколько main на странице?

НЕТ! Главный контент один. Это как главная роль в фильме — она одна.

```
<main>
  <h1>Заголовок страницы</h1>
  <section><!-- разделы --></section>
</main>
```

Article vs Section

> **Вопрос:** Когда использовать article, а когда section?

Article — самостоятельный контент:
• Статья в блоге
• Новость
• Комментарий

Section — тематический раздел:
• Глава в статье
• Группа похожего контента

Иерархия заголовков

> **Вопрос:** Можно ли пропускать уровни заголовков?

НЕЛЬЗЯ! H1 → H2 → H3. Как в книге: глава → параграф → подпараграф.

Почему нельзя пропускать:

- Скринридеры используют заголовки для навигации по странице
- SEO-роботы анализируют структуру для понимания важности контента
- Нарушается логическая иерархия — h1 важнее h2, h2 важнее h3
- Автоматическое оглавление работает неправильно
- Accessibility API браузера строит "дерево заголовков" для вспомогательных технологий

Правильная иерархия заголовков

❌ Неправильно:

```
<h1>Сайт</h1>
<h4>О нас</h4>  <!-- Пропущен h2,h3 -->
<h2>Контакты</h2>
<h1>Еще один h1</h1> <!-- Два h1! -->
```

✅ Правильно:

```
<h1>Главная страница</h1>
  <h2>О компании</h2>
    <h3>Наша история</h3>
    <h3>Команда</h3>
  <h2>Контакты</h2>
```

Только один h1 на странице, остальные по порядку!

Формы

> **Вопрос:** Зачем так много атрибутов в формах?

Каждый атрибут улучшает UX:

- `required` — показывает обязательность
- `placeholder` — подсказывает формат
- `type="email"` — включает валидацию и мобильную клавиатуру
- `label` — связывает подпись с полем

Типы input

Старый подход:

```
<input type="text" placeholder="Email" />
<input type="text" placeholder="Телефон" />
<input type="text" placeholder="Дата" />
```

HTML5 подход:

```
<input type="email" />
<input type="tel" />
<input type="date" />
<input type="color" />
<input type="range" />
```

Браузер сам добавит валидацию и удобства!

Доступность

> **Вопрос:** Кому нужна доступность?

- Людям с нарушениями зрения (скринридеры)
- Людям с ограниченной подвижностью (только клавиатура)
- Всем нам в определенных ситуациях (сломанная мышка)
- Поисковым роботам (они тоже "слепые")

Label и input

> **Вопрос:** Нельзя просто подписать поле текстом рядом?

Можно, но плохо! Label связывает подпись с полем. Клик по label фокусирует поле. Скринридер читает подпись при фокусе.

```
<label for="email">Email</label> <input type="email" id="email" />
```

ARIA

> **Вопрос:** Что такое ARIA и зачем оно нужно?

ARIA добавляет смысл там, где HTML не справляется. Как субтитры в фильме — помогают понять происходящее.

Важно: ARIA атрибуты обрабатываются браузером автоматически. Разработчик только добавляет атрибуты, а браузер сам передает информацию скринридерам.

ARIA: примеры работы

> **Вопрос:** Как именно ARIA помогает пользователям?

Примеры работы:

- `aria-expanded` — браузер автоматически сообщает скринридеру "кнопка меню, свернуто"
- `aria-controls` — браузер связывает кнопку с контролируемым элементом
- `role="alert"` — браузер автоматически озвучивает через скринридер
- `aria-live="polite"` — браузер читает изменения в подходящий момент

```
<button aria-expanded="false" aria-controls="menu">Меню</button>
<div role="alert" aria-live="polite">Форма отправлена!</div>
```

Альтернативы HTML

> **Вопрос:** Неужели есть что-то лучше HTML?

Не лучше, а удобнее для разных задач:

- **Pug** — без закрывающих тегов
- **Markdown** — для документации
- **JSX** (JavaScript XML) — HTML + JavaScript
- **Haml** — из мира Ruby

Pug

HTML:

```
<div class="card">
  <h3 class="title">Заголовок</h3>
  <p class="text">Текст</p>
</div>
```

Pug:

```
.card
  h3.title Заголовок
  p.text Текст
```

Меньше кода, больше читаемости!

Markdown

> **Вопрос:** Где используется Markdown?

- GitHub README файлы
- Документация проектов
- Блоги (Jekyll, Hugo)
- Заметки (Obsidian, Notion)
- Сообщения в Discord, Telegram

```
# Заголовок

**Жирный** и _курсив_
[Ссылка](https://example.com)
```

JSX

> **Вопрос:** JSX это будущее HTML?

JSX = JavaScript XML. Популярен в React мире. Позволяет смешивать HTML и JavaScript, создавать динамичные компоненты. Компилируется в обычный JavaScript.

```
function Header({ title, items }) {
  return (
    <header>
      <h1>{title}</h1>
      <nav>
        {items.map((item) => (
          <a href={item.url}>{item.name}</a>
        ))}
      </nav>
    </header>
  );
}
```

Инструменты проверки

> **Вопрос:** Как проверить качество HTML?

- **W3C Validator** — проверка на ошибки
- **WAVE** — анализ доступности
- **Lighthouse** — общая оценка качества
- **DevTools** — инспектор элементов

Чек-лист качественного HTML

- Валидный HTML5
- Семантические теги
- Правильная иерархия заголовков
- Доступные формы с label
- Alt для изображений
- Lang атрибут
- Meta viewport
- Осмысленные имена классов

Домашнее задание

- Создать семантическую HTML страницу-портфолио
- Использовать все изученные элементы
- Проверить через W3C Validator
- Попробовать Pug (дополнительно)

Следующее занятие: CSS

Вопросы?

HTML — это фундамент.
Семантика важнее красоты.