Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки

Кафедра «Информатика»

ВЕРСТКА ВЕБ-СТРАНИЦ

ПОСОБИЕ
для слушателей специальности переподготовки 9-09-0611-02 «Веб-дизайн и компьютерная графика» заочной формы обучения

Гомель 2025

УДК 004.738.1(075.8)
ББК 32.973.42я73
В26

Рекомендовано Советом института повышения квалификации и переподготовки ГГТУ им. П. О. Сухого
(*протокол № 4 от 26.12.2024 г.*)

Составитель *В. Н. Леонова*

Рецензент: доц. каф «Информатика» ГГТУ им. П. О. Сухого канд., экон. наук, доц. *Н. В. Ермалинская*

Верстка веб-страниц : пособие для слушателей специальности переподготовки
В26 9-09-0611-02 «Веб-дизайн и компьютерная графика» заоч. формы обучения / сост. В. Н. Леонова. – Гомель : ГГТУ им. П. О. Сухого, 2025. – 139 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://elib.gstu.by. – Загл. с титул. экрана.

Пособие дает комплексное понимание процесса создания веб-страниц. Рассмотрены ключевые темы для изучения: создание веб-страниц с помощью HTML и каскадных таблиц стилей (CSS); веб-стандарты, современные технологии верстки, оптимизация кода веб-страниц, а также проблемы и решения кроссбраузерности.

Для слушателей специальности переподготовки 9-09-0611-02 «Веб-дизайн и компьютерная графика» ИПКиП.

УДК 004.738.1(075.8)
ББК 32.973.42я73

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Пособие по дисциплине «Верстка веб-страниц» охватывает теоретические и практические аспекты, необходимые для освоения современных технологий веб-дизайна и верстки. Оно предназначено для предоставления комплексного понимания процесса создания веб-страниц начинающим веб-разработчикам и желающим освоить навыки верстки.

Задачи пособия заключаются в ознакомлении слушателей с основами HTML, CSS, с современными технологиями и инструментами верстки, кроссбраузерностью и оптимизацией веб-страниц.

Материал пособия направлен на формирование профессиональных компетенций и практических навыков, включая:

- создания и редактирования HTML-документов, работу с основными тегами и структурой страницы;
- умение стилизовать веб-страницы с помощью CSS, включая использование селекторов, свойств и единиц измерения;
- понимание применения и умение встраивать в структуру веб-страницы код JavaScript;
- понимание принципов адаптивного и отзывчивого дизайна для обеспечения корректного отображения на различных устройствах;
- осознание проблем кроссбраузерности и методов их решения для обеспечения совместимости веб-страниц;
- навыки работы с фреймворками и библиотеками, такими как Bootstrap и jQuery, для ускорения процесса разработки и улучшения функциональности.

В условиях стремительного развития цифровых медиа и увеличения объема информации, умение эффективно верстать страницы становится важным навыком для веб-дизайнеров. Современные инструменты и программное обеспечение позволяют создавать сложные макеты, которые не только привлекают внимание, но и обеспечивают удобство восприятия информации.

Использование наукоёмких технологий, таких как системы автоматизированной верстки и инструменты для адаптивного дизайна, помогает значительно ускорить процесс создания и редактирования контента. Эти технологии позволяют верстальщикам работать более эффективно, минимизируя ошибки и сокращая время

на доработку. В результате, верстка страниц становится более гибкой и позволяет быстро реагировать на изменения в требованиях к контенту.

Процесс верстки страниц включает в себя не только визуальное оформление, но и оптимизацию пользовательского опыта. Веб-дизайнеры должны учитывать различные аспекты, такие как читабельность, доступность и совместимость с различными устройствами. Правильное использование шрифтов, цветов и графических элементов позволяет создать гармоничное и привлекательное оформление, что, в свою очередь, влияет на восприятие информации пользователями.

С развитием технологий, таких как CSS Grid и Flexbox, верстка страниц становится более интуитивной и доступной для широко круга специалистов, включая тех, кто только начинает свой путь в дизайне. Однако, несмотря на доступность инструментов, умение создавать качественную верстку требует глубокого понимания как визуальных, так и технических аспектов, что делает образование в этой области особенно важным.

В последние десятилетия мир дизайна находится в состоянии постоянного изменения, вызванного появлением новых платформ и форматов для представления информации. Современные тренды в веб-дизайне, такие как минимализм и адаптивность, требуют от верстальщиков постоянного обновления знаний и навыков. Это также связано с необходимостью интеграции новых технологий, таких как анимация и интерактивные элементы, которые делают контент более привлекательным и вовлекающим.

Таким образом, верстка страниц становится не просто технической задачей, а важной частью стратегического подхода к созданию контента. Понимание современных тенденций, технологий и инструментов позволяет верстальщикам создавать уникальные и эффективные решения, которые отвечают требованиям времени и ожиданиям аудитории. В условиях быстро меняющегося цифрового мира, умение адаптироваться и осваивать новые технологии становится ключевым фактором успеха в сфере верстки и дизайна.

Веб-дизайн

Веб-дизайн (от англ. web design) — отрасль веб-разработки и разновидность дизайна, в задачи которой входит проектирование пользовательских веб-интерфейсов для сайтов или веб-приложений.

Веб-дизайнеры: проектируют логическую структуру веб-страниц; продумывают наиболее удобные решения подачи информации; занимаются художественным оформлением веб-проекта.

В настоящее время под термином веб-дизайн понимают именно проектирование структуры веб-ресурса, обеспечение удобства пользования ресурсом для пользователей.

Немаловажной частью проектирования ресурса стало приведение ресурса в соответствие стандартам W3C, что обеспечивает доступность содержания для инвалидов и пользователей портативных устройств, а также кросс-браузерность вёрстки ресурса. Также непосредственно с дизайном сайтов смежен интернет-маркетинг, то есть продвижение и реклама созданного ресурса, поисковая оптимизация.

Основные термины и определения

1. Юзабилити (Usability) - удобство использования сайта, включая простоту навигации и доступность информации.
2. Дизайн пользовательского интерфейса (UI Design) - фокусируется на создании интерфейсов, с которыми взаимодействуют пользователи. Это включает в себя внешние элементы, такие как кнопки, меню и графические элементы.
3. Пользовательский опыт (UX) - оценка общего впечатления пользователя от взаимодействия с сайтом. UX охватывает все аспекты использования продукта, включая использование интерфейса, удовлетворение и взаимодействие с контентом.
4. Адаптивный и отзывчивый дизайн:

- Адаптивный дизайн - создание нескольких версий сайта для различных устройств с фиксированными размерами экрана.
- Отзывчивый дизайн (Responsive Design) - подход, который позволяет сайту динамически изменять свои элементы, чтобы они соответствовали экрану пользователя, без необходимости создания нескольких версий.

5. Прототипирование - создание предварительных моделей веб-сайтов для тестирования и оценки дизайна перед его окончательной разработкой.

6. Типографика - искусство и техника расположения текста. Веб-дизайнеры выбирают шрифты, разметку и размеры текста для улучшения читабельности и визуальной эстетики.

7. Цветовая палитра - набор цветов, используемых в дизайне сайта. Веб-стандарты - набор рекомендаций и норм, разработанных W3C (World Wide Web Consortium) для обеспечения совместимости и доступности веб-сайтов. Это включает в себя семантическое HTML, CSS и принципы доступности.

8. Контент - информация, размещенная на сайте, включая текст, изображения, видео и аудио. Качество и структура контента играют важную роль в привлечении и удержании пользователей.

9. SEO (Search Engine Optimization) - совокупность методов, направленных на улучшение видимости сайта в поисковых системах.

10. HTML (HyperText Markup Language) - язык разметки, используемый для создания структуры веб-страниц. Позволяет описывать содержание и структуру документа с помощью тегов.

11. CSS (Cascading Style Sheets) - каскадные таблицы стилей, язык для описания внешнего вида HTML-документов. Позволяет управлять стилем и компоновкой элементов на веб-странице.

12. DOM (Document Object Model) - объектная модель документа, представляющая структуру HTML-документа в виде объектов, с которыми можно взаимодействовать с помощью JavaScript.

13. Тег - элемент HTML, который определяет структуру и содержание веб-страницы. Каждый тег открывается и закрывается, или является одиночным.

14. Атрибут - дополнительная информация, связанную с тегом, задающая свойства его поведения или отображения. Например, атрибут src в теге `<img>`.

15. Семантические теги - теги, которые имеют чёткое значение в контексте содержимого, например, `<header>`, `<footer>`, `<article>`, что улучшает доступность и SEO.

16. Блочный элемент - элемент, который занимает всю ширину доступного пространства и начинает новую строку (например, `<div>`, `<p>`).

17. Строчный элемент - элемент, который занимает лишь ту ширину, которая необходима для его содержимого, и не начинает новую строку (например, `<span>`, `<a>`).

18. Медиазапросы - условия в CSS, позволяющие применять разные стили в зависимости от характеристик устройства, например, ширины экрана.

19. Адаптивная верстка - подход, при котором веб-страница создается с фиксированным набором дизайнов для разных разрешений экрана.

20. Отзывчивый дизайн - метод, позволяющий веб-странице автоматически адаптироваться к размеру устройства, на котором она просматривается, используя гибкие сетки и медиазапросы.

21. Фреймворк - набор инструментов и библиотек, упрощающих разработку веб-приложений.

22. Иконочный шрифт - специальный шрифт, который предоставляет набор иконок вместо символов.

23. Оптимизация кода - процесс улучшения кода, чтобы уменьшить его размер и время загрузки страницы. Включает минификацию CSS и JavaScript, оптимизацию изображений и кэширование.

24. Кроссбраузерность - способность веб-страницы одинаково корректно отображаться и функционировать в разных браузерах и на различных устройствах.

25. Единицы измерения - спецификаторы, используемые в CSS для задания размеров, например, `px` (пиксели), % (проценты), `em` (относительная единица, основанная на размере шрифта).

26. Логическая структура - последовательное расположение различных элементов на странице для удобства восприятия и навигации.

27. Контент - все элементы, находящиеся на веб-странице: текст, изображения, видео и т.д.

28. Инлайн стили - это CSS-стили, которые применяются непосредственно к HTML-элементу с помощью атрибута `style`. Такие стили определяются внутри тега элемента и имеют приоритет над внешними и внутренними таблицами стилей.

Эти понятия являются основополагающими для понимания и практики веб-дизайна.

1. HTML (HyperText Markup Language)

HTML (HyperText Markup Language) — это стандартный язык разметки для создания веб-страниц. Он используется для описания структуры содержимого на веб-сайте с помощью различных тегов и атрибутов.

Основные функции HTML:

- Структурирование контента — HTML позволяет организовывать текст, изображения, видео и другие элементы на странице.
- Создание ссылок — с помощью тегов <a> можно создавать гиперссылки, которые связывают разные страницы или ресурсы в интернете.
- Форматирование элементов — HTML поддерживает различные стили текста, таблицы, списки и формы для сбора информации от пользователей.
- Интеграция медиа — HTML позволяет вставлять изображения, аудио и видео на страницы.
- Основы для CSS и JavaScript — HTML является основой, на которой строятся стили (CSS) и интерактивные элементы (JavaScript).

Таким образом, HTML является основой в разработке веб-сайтов и создании многогранного пользовательского опыта.

1.1 История появления языков разметки

Языки разметки появились в 1960-х годах для упрощения работы с текстовыми документами. Основные этапы их развития:

1. SGML (Standard Generalized Markup Language):

- Разработан в 1986 году. SGML стал основой для многих языков разметки и позволил структурировать текстовые данные. Он обеспечивал гибкость и расширяемость, что сделало его популярным для создания документации.

2. HTML (HyperText Markup Language):

- В 1991 году Тим Бернерс-Ли создал HTML как язык разметки для документов в сети. HTML позволял связывать документы с помощью гипертекстовых ссылок и стал основой для веб-страниц.

Пример гиперссылки:

```
1  Этот просто текст предложения, а по этой
2  <a href="http://server/123/ABC.TXT">ссылке</a>
3  можно узнать подробнее.
4
```

Этот просто текст предложения, а по этой ссылке можно узнать подробнее.

Рис. 1. Пример кода гиперссылки и её отображения на странице

В HTML для гиперссылок используется тег <a> (anchor / якорь), который окружает текст, являющийся ссылкой. С помощью тега можно связать адрес ссылки, URL, и, непосредственно, текст ссылки.

Существует множество различных тегов HTML, которые позволяют определять помимо ссылок различные структурные элементы документа, такие как: списки, таблицы, заголовки, разделители и многое другое.

Гипертекст сам по себе не определяет внешний вид документа. Он позволяет определить его структурные элементы и взаимосвязи между ними (например: заголовок, подзаголовок, параграф). Такой подход к построению документа называется *семантической версткой*. Семантическая разметка позволяет определить четкую структуру документа.

За внешний вид отвечает интерпретатор гипертекста. Обычно это *веб-браузер*.

Для верстки HTML не нужен сложный инструментарий — достаточно простого текстового редактора.

И хотя язык HTML предназначен для достаточно узкой цели — разметки структуры текстовых документов, однако подобные языки также удобно использовать для структуризации любых данных.

Для этого была придумана модификация, которая называется XML (eXtensible Markup Language / расширяемый язык разметки).

XML (eXtensible Markup Language) — это метаязык, предназначенный для описания структурированных данных. Он позволяет пользователям создавать свои собственные теги и определять структуру данных, что делает его гибким и универсальным.

Основные характеристики XML:

Расширяемость: Пользователи могут создавать собственные теги, что делает язык подходящим для различных типов данных.

Структурированность: XML предоставляет четкую структуру для представления данных, которая может быть легко интерпретирована компьютерами и людьми.

Совместимость: XML поддерживается многими языками программирования и инструментами, что позволяет интегрировать его с различными системами.

Пространства имен: Это часть спецификации XML, которая помогает избегать конфликтов имен и обеспечивает однозначность элементов, определенных в разных контекстах. Спецификация XML, в контексте пространств имен, описывает правила и синтаксис, которые нужно соблюдать при их использовании.

Основные моменты спецификации пространств имен XML:

- Объявление пространства имен: Спецификация определяет, как объявлять пространства имен с помощью атрибута xmlns. Это можно сделать как для элементов, так и для атрибутов.

- Правила проинтерпретации: Спецификация описывает, как обработчики XML должны обрабатывать элементы и атрибуты с префиксами, определенными в пространстве имен. Это необходимо для однозначного интерпретирования документов.

- Иерархия: Пространства имен могут быть вложенными, и спецификация определяет их порядок и правила обработки в таких случаях.

- Кросс-схемная совместимость: Использование пространств имен позволяет создавать документы, которые могут комбинировать элементы из различных схем, что делает их более универсальными в использовании.

Спецификация пространств имен XML была представлена в документе W3C «Namespaces in XML. Эти правила необходимы для обеспечения корректной работы XML-документов и взаимодействия различных приложений.

Таким образом, пространства имен в XML — это один из аспектов этого языка, который способствует его гибкости и универсальности в представлении данных.

Существует множество модификаций XML, самые известные из них:

- форматы файлов для текстовых и табличных процессоров (в частности, продуктов Microsoft и их аналогов): DOCX, XLSX, PPTX;
- формат для описания векторных изображений — SVG;
- определение плейлистов для видео или музыки — XSPF.

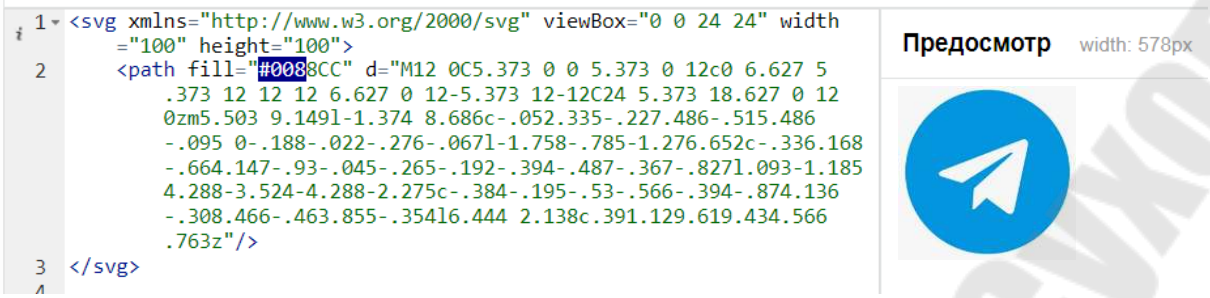


Рис. 2. Иконка Telegram в виде кода SVG

3. XHTML (Extensible HyperText Markup Language):

- В начале 2000-х годов XHTML был разработан как более строгая версия HTML, основанная на XML (Extensible Markup Language). Он обеспечивал более четкие правила разметки и поддержку для различных устройств.

4. HTML5:

- В 2014 году был принят стандарт HTML5, который добавил новые элементы и атрибуты, улучшил поддержку мультимедиа и мобильных устройств, а также обеспечил более семантическую разметку.

1.2. Стандарты HTML

HTML (HyperText Markup Language) – это не язык программирования, а так называемый язык разметки. Он необходим для сообщения браузеру способа отображения посещаемой веб-страницы. Вместо того чтобы показывать сплошной текст, HTML позволяет обернуть его в специальные элементы (теги). Это обеспечивает корректное отображение любой информации.

Теги способны нести как структурный смысл (например, показать, что перед нами таблица некоторой размерности с заголовком), так и семантический (выделить информацию определенным образом для поисковых роботов, чтобы те лучше индексировали сайт).

Изначально HTML был кроссплатформенным, что создавало определенные трудности. Разные браузеры отображали теги по-своему, некоторые элементы могли восприниматься не всеми программами. Причина: отсутствие единого стандарта, поддерживаемого разработчиками. В последние годы данная проблема фактически решена, так как крупные игроки на рынке браузеров пришли к соглашению.

Современные стандарты HTML

Базис современной разработки – стандарт HTML5. До этого главенствующим стандартом являлся HTML 4 (с 1997 года). Он встречается и сегодня на ряде сайтов. Определить его можно на основании так называемого доктайпа.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

Это первая строка любого HTML-документа, позволяющего браузерам понять способ работы с содержимым страницы. Хоть стандарт и является устаревшим, в нем содержится основной набор тегов, которыми пользуются при создании сайтов. Важно учесть, что часть элементов уже не рекомендована к применению и со временем будет убрана.

Несмотря на строгую типизацию в языке HTML, ряд браузеров достаточно лояльно относится к документам. Так, тег абзаца `<p></p>` написанный следующим образом – будет корректным:

- `<p>`Абзац`</p>` (написание в соответствии со стандартом);
- `<P>`Абзац`</P>` (браузеры редко учитывают регистр тегов);
- `<p>`Абзац (если тег не закрыт, то отображение все равно будет корректным).

Для введения единого подхода в верстке, был разработан и внедрен стандарт XHTML. Он сочетает в себе HTML и XML форматы.

Ключевые особенности стандарта XHTML:

– все элементы пишутся строго в нижнем регистре;

– одиночные теги должны быть закрыты (например, `<img />`);

– основная кодировка – UTF-8;

– все атрибуты записываются только в развернутом виде (например, если в HTML5 отключить возможность выбора элемента из выпадающего списка можно указанием простого атрибута `disabled`, то в XHTML обязательно писать `disabled="disabled"`).

Специфика HTML5:

– добавлены семантические элементы (<nav>, <section>, <article> и др.), которые облегчают чтение структуры страницы как разработчикам, так и поисковым машинам;

– внедрена поддержка векторной графики и специальных форматов (svg, canvas, MathML);

– представлены новые элементы управления (dates, email, tel);

– удалены устаревшие теги (big, center, isindex).

Стандарты языков разметки разрабатываются и поддерживаются организациями, такими как:

1. W3C (World Wide Web Consortium) – основная организация, занимающаяся разработкой стандартов для веб-технологий, включая:

- CSS (каскадные таблицы стилей);
- DOM (объектная модель документа);
- PNG (формат хранения растровых изображений);
- HTTP (протокол передачи данных);
- RDF (модель представления метаданных);
- XPath (язык для доступа к частям XML-документов) и др.

2. WHATWG (Web Hypertext Application Technology Working Group) – группа, занимающаяся разработкой спецификаций для HTML и других веб-технологий.

1.3 Возможности и особенности использования HTML

- HTML позволяет организовывать контент с помощью заголовков, абзацев, списков и других элементов;
- HTML предоставляет возможность создавать гиперссылки для перехода между страницами и ресурсами через тег <a>;
- HTML поддерживает создание форм для ввода данных с помощью элементов <form>, <input>, <textarea>, <select> и других;
- HTML позволяет вставлять изображения, видео и аудио с помощью тегов <img>, <video>, <audio>;

- семантические элементы помогают лучше структурировать контент и улучшают доступность и SEO (продвижение поисковыми системами) `<article>`, `<section>`, `<header>`, `<footer>`;
- HTML работает совместно с CSS (Cascading Style Sheets), что позволяет изменять внешний вид элементов и создавать адаптивные дизайны;
- учтена доступность веб-контента для людей с ограниченными возможностями, используя атрибуты для изображений, такие как `alt`, и правильную структуру заголовков.

HTML является основным языком разметки для создания веб-страниц. Понимание его структуры, использования тегов и специальных символов позволяет разработчикам эффективно создавать и организовывать контент, обеспечивая хорошее взаимодействие с пользователями и доступность информации.

2. ЭЛЕМЕНТЫ ЯЗЫКА HTML

HTML (HyperText Markup Language) состоит из различных элементов, которые обозначаются с помощью тегов. Теги определяют структуру и содержание веб-страницы. В HTML различают служебные (или пустые) теги и неслужебные (или парные) теги.

2.1. Теги (служебные и не служебные)

Служебные (пустые) теги:

Служебные теги не имеют закрывающей части и используются для вставки элементов, которые не содержат вложенного контента. Примеры:

- `<br>`: перенос строки;
- `<hr>`: горизонтальная линия;
- `<img>`: вставка изображения;
- `<input>`: элемент ввода данных;
- `<meta>`: метаинформация о документе.

Пример использования служебных тегов `img` и `br`:

```
1  <p>Это текст с переносом строки.<br>Вот следующая строка.</p>
2  <img src="image.jpg" alt="Описание изображения">
3
```

Это текст с переносом строки.
Вот следующая строка.

Описание изображения

Рис. 3. Пример использования служебных тегов

Неслужебные (парные) теги:

Неслужебные теги имеют открывающую и закрывающую часть. Примеры:

- `<p></p>`: абзац текста.
- `<h1></h1>, <h2></h2>, ..., <h6></h6>`: заголовки различных уровней.
- `<div></div>`: блочный элемент для группировки содержимого.
- `<span></span>`: строчный элемент для выделения текста.

Пример использования неслужебных тегов `p, h1, div`:

```
1  <h1>Заголовок первого уровня</h1>
2  <p>Это абзац текста.</p>
3  <div>
4      <p>Это текст внутри блока.</p>
5  </div>
6
```

Предосмотр width: 572px

Заголовок первого уровня

Это абзац текста.

Это текст внутри блока.

Рис. 4. Пример использования неслужебных тегов

2.2. Специальные символы

В HTML существуют специальные символы, которые нельзя просто так вставить в текст, так как они могут быть интерпретированы как теги. Для их отображения используются HTML-сущности. Примеры:

- `<` — меньше чем (<)
- `>` — больше чем (>)
- `&` — амперсанд (&)
- `&quot;` — двойные кавычки (")
- `&apos;` — одинарные кавычки (')

Пример использования специальных символов:

```
1  <p>Используйте &lt;h1&gt; для заголовка первого уровня.</p>
2  <p>Символ амперсанда: &amp;</p>
3
```

Предосмотр width: 572px

Используйте <h1> для заголовка первого уровня.

Символ амперсанда: &

Рис. 5. Специальные символы в коде и их отображение на веб-странице

2.3. Структура тега

Типичная структура веб-элемента:

- Открывающий тег (все теги представляются в угловых скобках, в открывающем, если требуется, перечисляются атрибуты);

- Закрывающий тег (присутствует не во всех тегах, идентифицируется косой чертой);
- Атрибут, свойство (название свойства и его значение предпочтительно в двойных кавычках, возможно использовать и одинарные);
- Содержимое (внутреннее содержимое тега, обычно в виде текста либо других вложенных тегов).

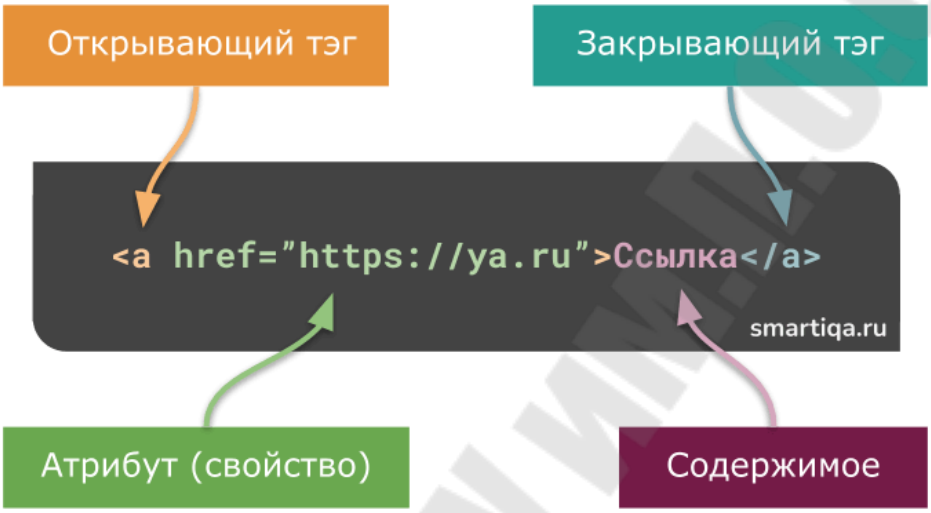


Рис. 6. Структура web-элемента

Подобным образом выглядит любой элемент веб-страницы. Сайт представляет собой совокупность таких страниц, связанных между собой внутренними и внешними гиперссылками.

2.4 Парные и одиночные теги

Теги бывают парными или одиночными.

Одиночные тэги

Одиночных элементов насчитывается 16 штук. Наиболее часто используемые:

- `<!doctype>` – некая инструкция браузерам для определения того, к какому типу относится документ, какую версию HTML использовать;
- `<meta>` – необходим для задания дополнительной информации о странице, указывается в заголовке HTML-документа;
- `<hr>` – задает горизонтальную линию для визуального отделения элементов страницы;

- `<br>` – перенос нижеидущего содержимого на новую строку;
- `<img>` – используется для вставки на страницу изображения;
- `<input>` – применяется в формах и задает тип полей (позволяет выбирать дату, отмечать те или иные элементы списка и т.п.).

Парные тэги

Парные теги обязательно имеют закрывающий тег и содержимое. В HTML их около 100. Пример тегов:

- `<p>Текст</p>` – представляет заключенный внутри текст в виде отдельного абзаца;
- `<a href="#">Текст ссылки</a>` – используется для задания и представления в документе ссылки;
- `<small>Текст</small>` – делает текст меньшего масштаба по сравнению с основным;
- `<mark>Текст</mark>` – выделяет текст по подобию маркера (аналог подчеркивания на бумаге самого важного фломастером).

С помощью CSS имеется возможность менять поведение практически всех тегов.

2.5 Блочные и строчные теги

Блочные теги занимают всю ширину страницы или родительского элемента.

Строчные теги не имеют строго размера. Они зависят от того, сколько символов в них содержится.

Пример блочных тегов:

- `<div>` – применяется максимально широко. Пример:

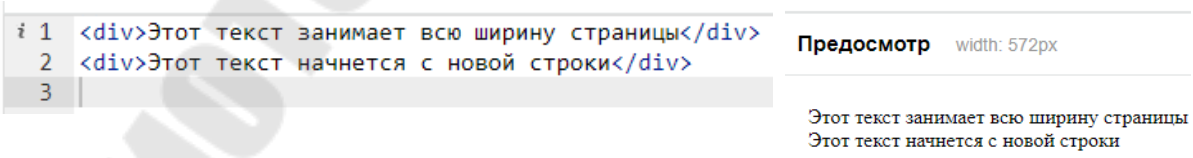


Рис. 7. Блочные теги и их отображение на веб-странице

Если сохранить приведенный код в виде `html`-файла и открыть в браузере, то увидим, что куски текста начинаются с разных строк и стремятся занять всю доступную ширину, которая им предоставлена;

- `<table></table>` – позволяет создавать таблицы;
- `<p>Параграф</p>` – обозначает параграф текста;
- `<h1>Большой заголовок</h1>` – для оформления заголовка самого верхнего уровня (в роли такового обычно выбирается основная статья страницы);
- `<form></form>` – отражает на сайте форму (авторизации, регистрации, опроса и др.);
- `<span>` – часто используемый строчный элемент. При помощи атрибутов его содержимое можно определенным образом выделять.

Наиболее используемые строчные теги:

- `<b>Выделение жирным шрифтом</b>` – выделяет участок содержимого жирным шрифтом без придания особой важности. Тег применяют в целях визуализации, чтобы информация воспринималась лучше;

```
1  •  <b>Выделение жирным шрифтом</b>
```

• **Выделение жирным шрифтом**

Рис. 8. Пример отображения текста в строчном теге

- `<strong>Важный текст</strong>` Обозначение значимого участка текста. Чаще всего выглядит как выделенный жирным шрифтом, но воспринимается поисковыми системами как имеющий особую важность в приведенном контексте;

3. `<a href="#">Ссылка в никуда</a>` Ссылка занимает тот объем на странице, который требуется для демонстрации ее текста

4. `<sub>, <sup>` Необходимы для представления степеней и индексов.

Пример - HTML

```
1  <span>sin<sup>2</sup>x + cos<sup>2</sup>x = 1</span>
2  <br>
3  <span>CO<sub>2</sub></span>
4
```

Предосмотр

$\sin^2 x + \cos^2 x = 1$
CO_2

Рис. 9. Степени и индексы в коде

В HTML5 прямого разделения тегов на блочные и строчные фактически нет. Сейчас говорят о категориях контента, который содержится внутри элемента.

Контентная модель HTML5 позволяет более точно описать типы содержимого. Она включает такие типы:

- поток (`flow`) – отображает основное содержимое страницы (большая часть тегов включается именно сюда. Например: `a, button, header, nav, ol, section`);
- метаданные (`metadata`) – применяются в заголовке документа и определяют поведение остального содержимого и связь с иными данными (Пример тегов: `title, meta, script, style`);
- текст (`phrasing`) – включает непосредственно текст страницы и используемые для его форматирования теги. Фактически, это строчные элементы: `a, button, i, img, span, textarea`;
- встроенный контент (`embedded`) – часть текстового, строчного содержимого, включающая импортированный контент (Пример тегов: `audio, video, img, svg, canvas`);
- интерактивные элементы (`interactive`) – позволяют посетителю сайта взаимодействовать с ресурсом (Включает теги: `button, select, video, a, input, textarea`);
- заголовки (`heading`) – для определения заголовков сайта, статей, подразделов (теги от `h1` до `h6`);
- секции (`sectioning`) – выделение изолированного контента в блоки. Включает теги: `article, aside, nav, section`.

2.6 Семантические теги

Появление HTML5 позволило подойти к вопросу размещения контента на сайтах по-новому, так как огромное количество сайтов затрудняет демонстрацию пользователям максимально релевантных. Немаловажной стала задача обеспечения доступности ресурсов посетителям с ограниченными возможностями.

Обычного набора тегов стало не хватать. Потребовался особый семантический набор для поисковых систем, чтобы они стали лучше понимать интернет-ресурсы.

К основным семантическим тегам такого типа относят:

- `<article>`Будущая статья`</article>` – используется для выделения независимого, самоопределяемого контента. Тег

подразумевает содержимое, которое несет ценность само по себе, обосабливается в общем контексте сайта. Хорошим примером такого вида служит пост на форуме или в блоге, отдельная статья сайта, новость;

- `<aside>`Побочная информация`</aside>` – некие дополнительные данные, косвенно связанные с основным содержанием страницы;
- `<details>`Дополнительные сведения`</details>` – справочная, дополняющая информация, которую пользователь по своему желанию может не смотреть. Сюда могут включать сведения об авторе, некие уточнения;
- `<summary></summary>` – дает краткую характеристику содержимого блока `details`. Посмотрим на примере совместную работу элементов. Пример:

```
<details>
    <summary>W3-Консорциум</summary>
    Возглавляется Тимом Бернесом-Ли, разработчиком языка разметки
        HTML.
</details>
```

Предосмотр widt

▶ W3-Консорциум

Рис. 10. Использование семантических тегов `details` и `summary`

Вместо кнопки «Подробнее» пользователь увидит надпись «W3-Консорциум», нажав на которую получит дополнительные сведения;

- `<figcaption>`Описание`</figcaption>` – комментарий к картинке, пояснения, название;
- `<figure></figure>` – выделяет рисунок и его описание в единый блок. Пример:

```
<figure>
    <img src="cat.jpg">
    <figcaption>Мой кот сегодня доволен</figcaption>
</figure>
```

Предосмотр width: 572px

Мой кот сегодня доволен

Рис. 11. Использование семантического тега `figure`

- `<footer>`Заключение`</footer>` – позволяет выделить нижний колонтитул статьи или раздела сайта;
- `<header>`Введение`</header>` – служит для вступительного содержания страницы сайта, указывают заголовок, некие вводные данные;

- `<nav>Разделы сайта</nav>` – основное меню сайта, содержит ссылки на разделы сервиса;
- `<section>Некий блок сайта</section>` – отдельный раздел документа, который требуется выделить в изолированный блок;

`<time></time>` – позволяет браузеру и поисковым системам понять, что заключенные в тег данные представляют собой дату. Визуально этот элемент себя никак не проявляет, тем не менее подразумевает семантический контекст. Пример:

```
1  Мы работаем до <time>22:00</time> ежедневно
```

Мы работаем до 22:00 ежедневно

Рис. 12. Семантический тег `<time>`

- `<mark>Выделенные данные</mark>` – содержит информацию, требующую выделения или подчеркивания.

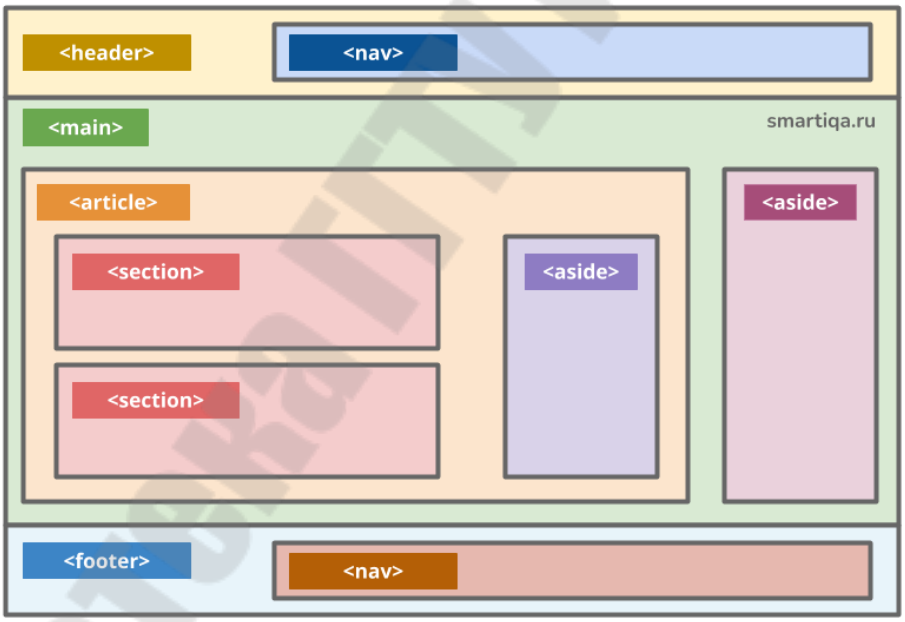


Рис. 13. Структура веб-страницы, построенной на семантических тегах

Представленные семантические теги являются новыми, введенными в HTML5. Многие старые элементы также относятся к семантическим (`<h1>`, `<strong>`, `<table>`, `<form>` и др.).

Использование семантической разметки элементов положительно влияет на эффективность и «рейтинг» сайта.

2.7 Классификация атрибутов

Атрибуты или свойства тегов позволяют расширить функционал HTML-элементов либо предоставить дополнительные сведения о них.

Атрибуты бывают: универсальные, уникальные, событийные или специфические.

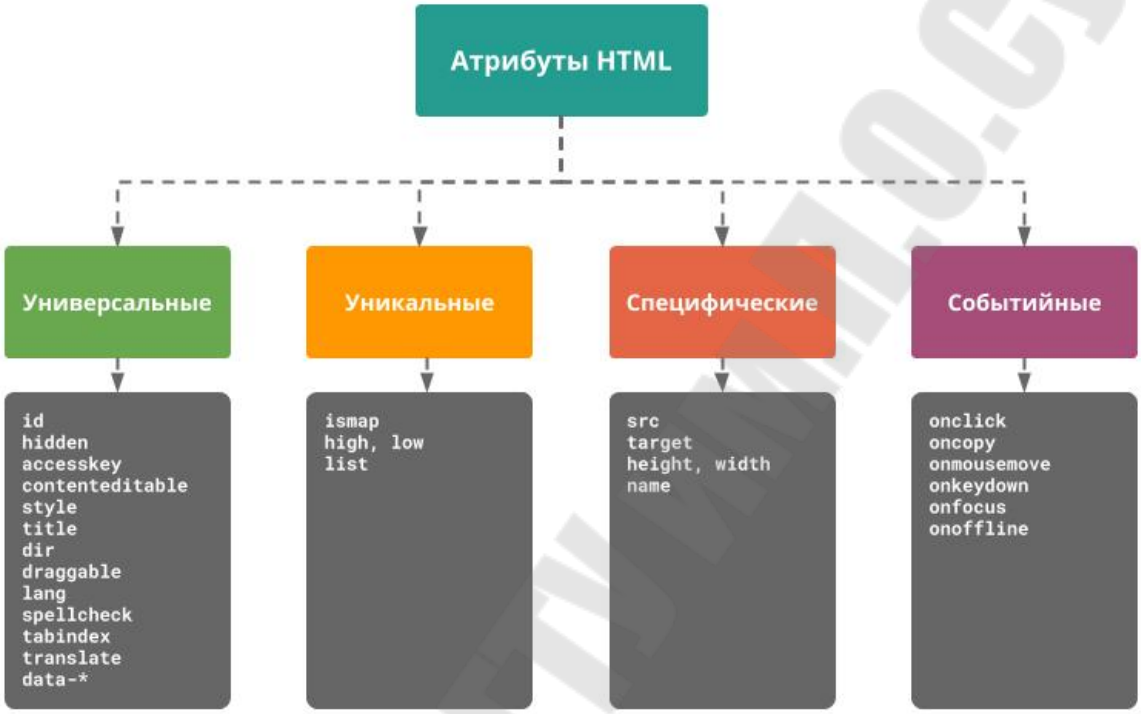


Рис. 14. Классификация атрибутов HTML тэгов

Универсальные атрибуты

Пример перечня свойств для универсальных атрибутов:

- `class` – указывает на имя класса элемента, по которому можно обращаться через каскадные таблицы стилей (или через js-скрипт). Классовое свойство с тем же именем может присутствовать у нескольких тегов;
- `id` – позволяет уникализировать элемент и получить возможность воздействовать на него скриптами, не затрагивая другие теги;
- `hidden` – если указать такое свойство, то элемент не будет отображаться на странице (в противном случае он видим);
- `style` – применяется для задания CSS-свойств прямо в теге без использования CSS-файла;

• title – дополнительные сведения об элементе. Если навести на него мышкой, то выводится всплывающая подсказка с содержимым атрибута title.

Уникальные атрибуты

Уникальные атрибуты характерны только для одного элемента:

• ismap – имеется только у тега <img> (позволяет получать координаты клика по картинке);

• high, low – уникальны для тега <meter> – для задания верхней и нижней границы предела определенного диапазона в рамках элемента (например, мы показываем пользователю процент прочтения статьи. Он может варьироваться от 0 до 100 %);

• list – принадлежит элементу <input> – указывает на id списка из элементов которого можно выбирать значения для выбранного поля формы.

Специфические атрибуты

Существенная часть атрибутов относится не к одному тегу, а к нескольким. Пример атрибутов, актуальных в HTML5:

• src – ссылка на источник медиа-файла (применим к тегам: <audio>, <embed>, <iframe>, <img>, <input>, <script>, <source>, <track>, <video>).

• target – задает источник целевого открытия ссылки или валидации формы. Актуально для элементов: <a>, <area>, <base>, <form>. Может принимать следующие значения: _blank (откроется новая вкладка в браузере), _self (останемся на текущей вкладке), _parent (загрузка в родительский фрейм, если страница на них разделена), _top (несмотря на наличие фреймов они все закроются, а ссылка откроется на всё окно браузера);

• height, width – определяют высоту и ширину элемента. Применимы для тегов <canvas>, <embed>, <iframe>, <img>, <input>, <object>, <video>. Значения – положительные числа (трактуются как пиксели). Можно использовать и проценты - считаются относительно родительского элемента;

• name – для задания имени элемента. Применяется для следующих тегов: <button>, <fieldset>, <form>, <iframe>, <input>, <map>, <meta>, <object>,

<output>, <param>, <select>, <textarea>. По этому названию к объекту можно обращаться через JavaScript, например.

Событийные атрибуты

Событийные атрибуты связаны с JavaScript-событиями. При взаимодействии пользователя с элементами эти события могут срабатывать и изменять поведение объектов. Пример событийных атрибутов:

- onclick – скрипт срабатывает при клике мышкой по элементу (можно вызвать всплывающее окно, изменить размер или цвет, перенаправить на другую страницу и т.п.);
- oncopy – вызывает скрипт при попытке скопировать содержимое элемента (таким способом часто запрещают копировать содержимое либо добавляют ссылку на первоисточник);
- onmousemove – эффект при движении мыши (позволяет создавать интересные эффекты на странице: смещение, увеличение, тряску и т.п.);
- onkeydown – обработка нажатия определенной клавиши;
- onfocus – изменение поведения объекта при фокусе на нем (например, когда поставили курсор в поле для ввода логина, само поле может изменить цвет, стать больше, дать какую-то подсказку);
- onoffline – поведение страницы в момент отсутствия активного подключения к сети.

Атрибуты тегов существенно повышают возможности HTML. Они позволяют не просто добавлять новые характеристики и менять свойства элементов, но и дают возможность использовать фактически неограниченный потенциал языка JavaScript.

2.8 Основные теги элементов HTML-разметки

Структура HTML-документа включает в себя основные элементы, которые определяют его содержание и функциональность.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Название страницы</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>

    <header>
        <h1>Заголовок сайта</h1>
        <nav>
            <ul>
                <li><a href="#home">Главная</a></li>
                <li><a href="#about">О нас</a></li>
                <li><a href="#services">Услуги</a></li>
                <li><a href="#contact">Контакты</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <section id="home">
            <h2>Добро пожаловать на наш сайт</h2>
            <p>Краткое описание услуги или продукта.</p>
        </section>

        <section id="about">
            <h2>О нас</h2>
            <p>Информация о компании.</p>
        </section>

        <section id="services">
            <h2>Наши услуги</h2>
            <p>Описание предоставляемых услуг.</p>
        </section>

        <section id="contact">
            <h2>Контакты</h2>
            <p>Информация для связи.</p>
        </section>
    </main>

    <footer>
        <p>© 2025 Ваша компания. Все права защищены.</p>
    </footer>

</body>
</html>
```

Предосмотр width: 578px

Заголовок сайта

- Главная
- О нас
- Услуги
- Контакты

Добро пожаловать на наш сайт

Краткое описание услуги или продукта.

О нас

Информация о компании.

Наши услуги

Описание предоставляемых услуг.

Контакты

Информация для связи.

© 2025 Ваша компания. Все права защищены.

Рис. 15. Структура простого HTML-документа

Теги для задания основных элементов HTML-разметки:

1. `<!DOCTYPE html>` – определяет тип документа и версию HTML.
2. `<html lang="ru">` – открывающий тег HTML-документа с атрибутом `lang`, указывающим язык содержимого.
3. `<head>` – секция документа, содержащая метаданные, ссылки на стили, скрипты и заголовок:

- <meta charset="UTF-8"> – указывает кодировку документа (UTF-8);
- <meta name="viewport" content="width=device-width, initial-scale=1.0"> – устанавливает настройки для адаптивного дизайна;
- <title> – заголовок страницы, отображаемый в заголовке браузера;
- <link rel="stylesheet" href="styles.css">: Подключение CSS-стилей;

4. <body> – основная область контента HTML-документа, содержащая видимые элементы:

- <header> – заголовок страницы, обычно включает название сайта и навигацию;
- <nav> – навигационное меню;
- <main> – основной контент страницы;
- <section> – логически разделяет контент на разные секции;
- <footer> – нижний колонтитул, обычно содержащий информацию о конфиденциальности и контактные данные.

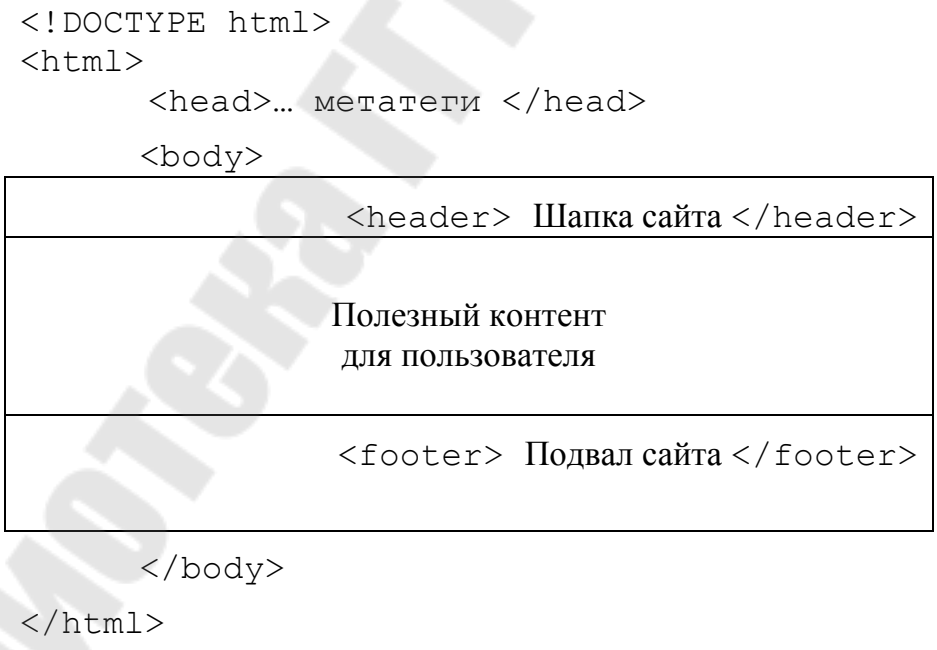


Рис. 16. Схема нахождения на веб-странице основных тегов HTML-разметки

5. Заголовочные теги: <h1>,<h2>,<h3>,...<h6> – заголовки, где <h1> представляет самый важный заголовок, а <h6> — наименее важный.

6. Параграф: <p> – определяет абзац текста.

Полезное для пользователя содержимое веб-ресурса располагается внутри тега <body>.

Большая часть элементов – его потомки. Они выполняют разные функции: работают с изображениями или видео, форматируют контент, представляют ссылки, изображают таблицы, отрисовывают формы.

Эта структура и набор тегов формируют основу любого HTML-документа и помогают организовать контент, делая его доступным и удобным для восприятия.

2.9 Теги верхнего уровня

Помимо информации, демонстрируемой непосредственно посетителю сайта, на нем обязана присутствовать и служебная. Она включает объявление доктайпа, заголовка, перечисление метасведений, ссылок на CSS- и JS-файлы и т.д.

<!DOCTYPE>

Доктайп задает стандарт, в котором создана веб-страница, чтобы браузерам было проще построить DOM-дерево (см. раздел DOM). На текущее время DOCTYPE можно встретить трех типов:

1. HTML5: <!DOCTYPE html>
2. HTML4: <!DOCTYPE ... "http://www.w3.org/TR/html4/loose.dtd">
3. XHTML: <!DOCTYPE ... "http://www.w3.org/TR/.../xhtml11.dtd">

Самым популярным и функциональным является первый вид (HTML5), но встречаются и остальные.

Корневые элементы

Помимо объявления стандарта документа, типичная веб-страница обычно включает четыре парных тега: <html>,<head>,<body>,<futer>

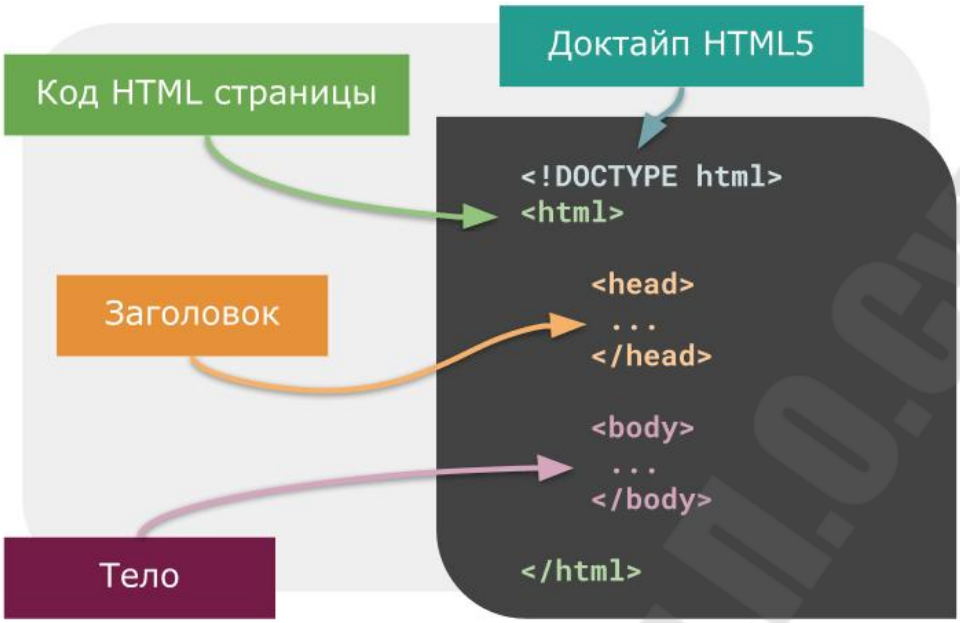


Рис. 17. Структура веб-страницы HTML

1. Тэг `<html>` - начинает и закрывает веб-страницу. В качестве языка документа задан русский язык и `xml` пространство имен. Пример:

```
1 <html lang="ru" xmlns="http://www.w3.org/1999/xhtml">
2     …
3 </html>
```

Рис. Обозначение языка документа и пространство имен

2. Тэг `<head>` – блок кода, в который оборачивается заголовочная часть веб-страниц. Это контейнер с метаданными, т.е. информацией о самом документе, которые напрямую не отображаются в окне браузера. Внутри обязательно наличие тега `<title>`. Может также включать элементы: `<style>`, `<base>`, `<link>`, `<meta>`, `<script>`.

Тэг `<title>` – необходим для отображения заголовка страницы. Его видно во вкладке окна. Контент – только текст. Содержимое заголовка важно для SEO, оптимизации поиска роботами вашего сайта. Для каждой страницы возможен только в единственном экземпляре.

Пример заполнения `title`:

```
1 <head>
2     <title>Заголовок страницы</title>
3 </head>
```

Рис. 18. Пример простого кода с тегом `<title>`

Пример отображения `title` страницы в браузере.

```
<!DOCTYPE html>
<html lang="ru-ru" dir="ltr">
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta charset="utf-8" />
    <base href="https://ipk.gstu.by/programmy-obucheniya/perepodgotovka-kadrov
    <meta name="keywords" content="ИПК ГГТУ им.П.О.Сухого, институт повышения
    <meta name="description" content="Институт повышения квалификации и перепо
Дополнительное образование (или переквалификация) расширяет возможности трудоу
По окончании обучения выдается диплом государственного образца о переподготовк
    <meta name="generator" content="Joomla! - Open Source Content Management"
    <title>Переподготовка кадров в ИПК ГГТУ им.П.О.Сухого</title>
    <link href="/programmy-obucheniya/perepodgotovka-kadrov.feed?type=rss" rel
    <link href="/programmy-obucheniya/perepodgotovka-kadrov.feed?type=atom" re
    <link href="/templates/protostar-2/favicon.ico" rel="shortcut icon" type="
```

Рис. 19. Html- код `<title>`

Отображение заголовка страницы в браузере «Переподготовка кадров в ИПК ГГТУ им. П.О.Сухого».

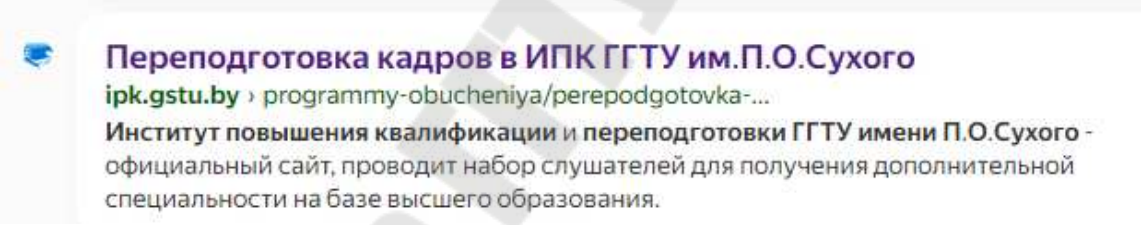


Рис. 20. Отображение `<title>` страницы в поиске Яндекса

Тэг `<style>` – применяется для указания информации о каскадных таблицах стилей.

Тэг `<base>` – тег является одиночным и необходим для указания основного адреса сайта. Все относительные ссылки внутри портала будут отталкиваться от этого корня. Пример:

```
<head>
    <meta name="viewport" content="width=device-width, init
    <meta charset="utf-8" />
    <base href="https://ipk.gstu.by/" />
    <meta name="keywords" content="ИПК ГГТУ им.П.О.Сухого,
```

Рис. 21. Пример тега `<base>` в `html`-коде страницы

Тэг <link> – является одиночным элементом, но может присутствовать во многих экземплярах в документе. Необходим для указания ссылок на CSS-документы, шрифты, фавикон.

Тэг <meta> – одиночный тег, используемый для указания метаинформации о странице. Эти сведения используются браузером и поисковыми машинами.

Тэг <script> – тег применим в любом другом месте (в зависимости от задачи). Содержит JavaScript код или ссылки на js-скрипты. Пример кода JavaScript:

```
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
    ></script>
<script>
    console.log('Все работает!')
</script>
```

Рис. 22. Пример кода JavaScript

Первый тег <script> говорит о том, что сайт использует фреймворк jQuery, а во втором случае в теги обернут просто код на языке JavaScript, который выводит в консоли инструментов разработчика в браузере обозначенный текст.

2.10 Тело веб-страницы

Между парными тэгами <body> – основная часть кода HTML-страницы, так называемое тело документа. Все оставшиеся теги будут присутствовать здесь. В документе тег может встречаться только один раз. Допустимо применение глобальных и событийных свойств.

Комментарии - особый тип данных, который встречается как в заголовочной части документа, так и в его теле. Код никак не обрабатывается браузером напрямую, но достаточно важен для разработчиков, так как позволяет включать пояснения, комментарии, уточнения. Пример:

```
<!--Это комментарий. Он не отражается браузером-->
```

Рис. 23. Комментарий в html-коде

2.10.1 Работа с текстом

Так как огромная часть наполнения сайта – текст, то для работы с ним предусмотрено солидное количество тегов. Они могут носить как стилистический, так и семантический контекст.

Пример тегов, входящих в `body`:

1. Тэг `<address>` – тег семантический, внутри тега приводится контактная информация. По умолчанию текст внутри тега выделяется курсивом.

Пример:

```
<address>
    Написано <a href="mailto:webmaster@example.com">Ивановым К. Т
        .</a><br>
    Адрес:<br>
    www.site.ru<br>
    400300, Новогородск, а/я 100<br>
    РФ
</address>
```

Предосмотр width: 578px

Написано Ивановым К. Т.
Адрес:
www.site.ru
400300, Новогородск, а/я 100
РФ

Рис. 24. Использование семантического тега `<address>`

2. Тэг `<b>` – делает текст жирным. Используется для визуального эффекта. Тег является строчным, поэтому текст внутри не переносится на новую строку. Пример:

```
<body>
    Обычный текст
    <b>Выделенный текст</b>
    Обычный текст
</body>
```

Предосмотр width: 880px

Обычный текст **Выделенный текст** Обычный текст

Рис. 25. Использование тега `<b>` для выделения текста жирным

3. Тэг `<q>` – для обозначения коротких цитат:

```
<p>Закон Мерфи:
    <q>Когда опаздываешь,
        маршрутка едет очень
        медленно и
        останавливается у
        каждого светофора.</q>
</p>
```

Закон Мерфи: “Когда опаздываешь, маршрутка едет очень медленно и останавливается у каждого светофора.”

Рис. 26. Использование тега `<q>` для ковычек

В данном примере приведены 2 тега: <p> (абзац, блочный) и <q> (строчный). Текст обрамляется кавычками.

4. Тэг <abbr> используется для выделения аббревиатуры. Можно применять совместно с элементом <dfn>, когда требуется сообщить, что термин будет расшифрован или ему будет дано определение.

Пример:

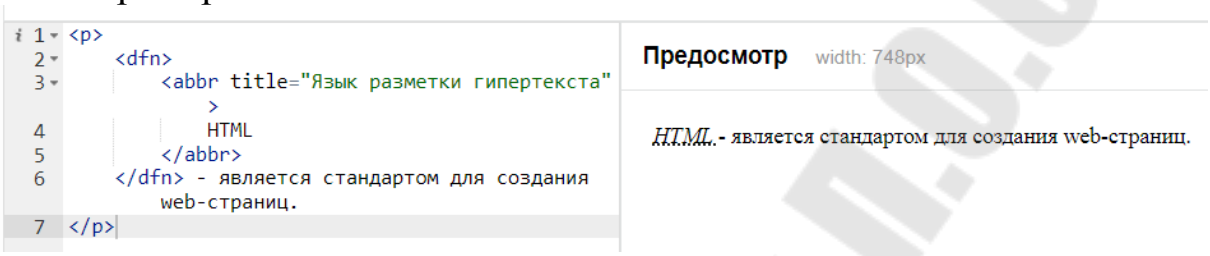


Рис. 27. Отображение текста на странице в теге <abbr>

Имеются и другие теги для взаимодействия с текстовыми данными: <blockquote>, <cite>, , <i>, <mark>, <s>, , <sup>, <u> и т.д.

2.10.2 Ссылки

Гипертекст – это текст со ссылками (гиперссылками) на другой текст, к которому читатель может немедленно получить доступ.

Гипертекстовые документы связаны между собой гиперссылками, которые обычно активируются щелчком мыши, набором нажатий клавиш или касанием экрана.

Для обозначения ссылок в HTML используется ряд тегов.

1. Тэг <a> – определяет гиперссылку. Имеет несколько атрибутов (главный из которых – href), поддерживает глобальные и событийные свойства. Поведение тега по умолчанию в современных браузерах следующее:

- не посещённая ссылка выделяется синим цветом и подчеркиванием;
- посещенная ссылка – голубого цвета и подчеркнута;
- активная ссылка – красная с подчеркиванием.

Пример:

```
1  <p>Текст</p>
2  <a href="https://yandex.ru/" target="_blank"
       >Яндекс</a><br>
3  <a href="mailto:mymail@mail.ru">Моя почта</a
       ><br>
4  <a href="tel:+79009000000" target="_blank">Мой
       телефон</a><br>
5  <a href="#part2" target="_blank">Якорная ссылка
       </a><br>
6  <span id="part2">Текст</span>
```

Предосмотр width: 748px

Текст

Яндекс
Моя почта
Мой телефон
Якорная ссылка
Текст

Рис. 28. Свойства ссылок в браузере

Ссылки могут вести на другие страницы сайта или внешние ресурсы, почтовые ящики, телефонные номера или другие части внутри документа.

2. Тэг `<nav>` – служит оберткой для главного меню сайта (элементы которого уже являются ссылками). Обычно применяется в единственном экземпляре на странице. Относится к блочным тегам.

Пример:

```
1  <nav>
2      <a href="#">HTML</a> |
3      <a href="#">CSS</a> |
4      <a href="#">JavaScript</a> |
5      <a href="#">Python</a>
6  </nav>
```

Предосмотр width: 748px

HTML | CSS | JavaScript | Python

Рис. 29. Пример навигационного меню в теге `<nav>`

Как видно, внутри элемента перечисляются ссылки, относящиеся к навигационному меню сайта.

2.10.3 Рисунки и мультимедиа

Для вставки мультимедиа используются специальные теги.

1. Тэг `<img>` – любое изображение растрового формата на сайте обозначается тегом `<img>`. Для его вставки применяются атрибуты `src` (указывается адрес) и `alt` (выведется текст на случай невозможности отобразить изображение; `width` и `height` – ширина и высота в `px`).

Пример:

```
<img src="images/photo.jpg" alt="Прекрасный пейзаж" width="600"
    height="400">
```

Предосмотр width: 57

Прекрасный пейзаж

Рис. 30. Вставка изображения с помощью тега <img>

Так как рисунка py.jpg не существует, отобразилось содержимое свойства alt. Для изображений также можно указывать размеры по ширине (width) и высоте (height) в пикселях или процентах.

2. Тэг <figure> – более семантически грамотный способ обозначать иллюстрации на странице. Задается адрес изображения и подпись к нему. Внутри себя содержит теги <img> и <figcaption>.

Пример:

```
<figure>
    <img src="https://yastatic.net/s3/home
        -static/_/_/i/tLI86LIriNsIJwhEqZS2o1Ckk
        .webp" alt="Яндекс">
    <figcaption>Логотип Яндекса на светлом фоне
        </figcaption>
</figure>
```

Предосмотр width: 748px

Яндекс

Логотип Яндекса на светлом фоне

Рис. 31. Использование семантического тега <figure>

Особых свойств не имеет, но может поддерживать глобальные и событийные атрибуты.

3. Тэг <video> – новый тег, появившийся в HTML5. Используется для внедрения на сайт видео-содержимого. Внутри включает элементы <source>, используемые для роликов разных форматов: MP4, WebM, и OGG.

Пример – HTML:

```
<video width="1300" height="800" controls>
    <source src="film.mp4" type="video/mp4">
    <source src="film.ogg" type="video/ogg">
    Ваш браузер не поддерживает тег VIDEO
</video>
```

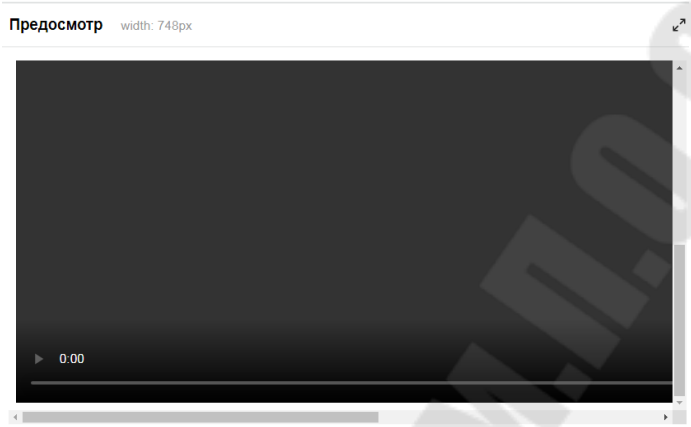


Рис. 32. Код для вставки видео-содержимого.

4. Тэг `<picture>` – более гибкая настройка отображения картинок. Позволяет не просто шкалировать рисунки в зависимости от используемого девайса или ширины экрана, но и отображать разные фото (по качеству, содержанию). Для этого применяют дополнительно теги `<source>, <img>`.

Пример – HTML:

```
<picture>
    <source media="(min-width:900px)" srcset
        ="http://ipic.su/img/img7/fs
        /operator_python-1024x575.1611716628
        .jpg">
    <source media="(min-width:600px)" srcset
        ="http://ipic.su/img/img7/fs/python
        .1611716845.jpg">
    <img src="http://ipic.su/img/img7/fs/v20642
        .1611716739.jpg" alt="Python">
</picture>
```

Рис. 33. Гибкая настройка отображения картинок для девайсов

По мере изменения размера окна браузера (или девайса) пользователю будут демонстрироваться разные иллюстрации.

2.10.4 Списки

Для более наглядного восприятия информации ее принято делить на списки, которые могут быть нумерованными или ненумерованными. HTML позволяет их создавать при помощи набора тегов: `<ul>`, `<ol>`, `<li>`, `<dl>`, `<dt>`, `<dd>`.

1. Нумерованные списки

Представляют собой перечисление элементов порядковыми значениями (числами, буквами). Пример:

```
<ol start="26" reversed type="a">
    <li>Буква Z</li>
    <li>Буква Y</li>
    <li>Буква X</li>
</ol>
```

Предосмотр width: 748px

z. Буква Z
y. Буква Y
x. Буква X

Рис. 34. Пример вывода нумерованного списка

Тег `<li>` отображает отдельные элементы списка. `<ol>` показывает, что список нумерованный. К нему применено 3 атрибута:

- `reversed` (обратный порядок нумерации);
- `start` (начали с последней буквы английского алфавита);
- `type` – указали, что в качестве номеров будут использоваться буквы в нижнем регистре.

2. Ненумерованные списки

В качестве маркеров используются специальные значки. Пример:

```
<ul type="disc">
    <li>Молоко</li>
    <li>Чай</li>
    <li>Квас</li>
</ul>
```

Предосмотр width: 748px

- Молоко
- Чай
- Квас

Рис. 35. Пример вывода ненумерованного списка

По умолчанию в роли маркеров используются закрашенные диски. Изменить тип маркера можно через CSS (вплоть до своих собственных рисунков) либо через атрибут `type` (`type="circle"`

- выведет окружности, `type="square"` – делает квадратные маркеры).

3. Списки с определениями

Особый тип списков, в которых содержатся термины и их трактовка. Удобно применять при формировании толкового словаря терминов.

Пример:

```
<dl>
    <dt>Гипертекст</dt>
    <dd>Группа документов, связанных воедино
        взаимными ссылками</dd>
    <dt>HTML</dt>
    <dd>Язык разметки гипертекста</dd>
</dl>
```

Предосмотр width: 748px

Гипертекст
    Группа документов, связанных воедино взаимными ссылками
HTML
    Язык разметки гипертекста

Рис. 36. Списки с определениями

2.10.5 Таблицы

Таблицы используются для представления данных в виде строк и столбцов. Они позволяют организовать информацию в удобном для восприятия формате.

Для построения таблиц применяются теги:

- `<table>` (содержимое таблицы);
- `<caption>` (заголовок), `<th>` (заглавная ячейка), `<thead>` (группирование заголовочной части таблицы);

Пример.

```
<table>
    <tr>
        <th>Название</th>
        <th>Автор</th>
        <th>Год</th>
    </tr>
    <tr>
        <td>1984</td>
        <td>Джордж Оруэлл</td>
        <td>1949</td>
    </tr>
    <tr>
        <td>Убить пересмешника</td>
        <td>Харпер Ли</td>
        <td>1960</td>
    </tr>
</table>
```

Предосмотр width: 578px

Название	Автор	Год
1984	Джордж Оруэлл	1949
Убить пересмешника	Харпер Ли	1960

Рис. 37. Простая таблица

- <tbody> (основная часть);
- <tr> (строка таблицы);
- <col> (колонка таблицы), <colgroup> (группировка колонок);
- <td> (ячейка);
- <tfoot> (итоговое содержимое).

При помощи свойств rowspan и colspan у тегов <th>, <td> ячейки таблиц можно объединять.

Для задания ширины и высоты элементов в CSS предназначены свойства width и height. Для ширины бордюра таблицы используется свойство border.

Пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Таблица</title>
    <style>
        td {
            border:indigo solid 3px;
            width: 100px;
            height: 100px;
        }
    </style>
</head>
<body>
    <table>
        <caption>Пример таблицы</caption>
        <tbody>
            <tr>
                <td rowspan="2">1</td>
                <td colspan="2">2</td>
                <td colspan="2">3</td>
            </tr>
            <tr>
                <td>4</td>
                <td>5</td>
                <td>6</td>
                <td>7</td>
            </tr>
            <tr>
                <td>8</td>
                <td colspan="4">9</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

Предосмотр width: 578px

Пример таблицы

1	2	3		
	4	5	6	7
8	9			

Рис. 38. Пример объединения ячеек в таблице

2.10.6 Формы и поля ввода

Форма является интерактивным элементом, так как позволяет пользователю вводить некоторые данные и отправлять их на сервер.

Формы оборачиваются тегом <form>, внутри которого возможно наличие элементов:

- <input> (после ввода);
- <textarea> (текстовое поле для ввода длинного текста);

- <button> (кнопка);
- <select> (меню опций);
- <option> (определенная опция);
- <optgroup> (группирование опций);
- <fieldset> (группирование части формы);
- <label> (подпись элемента);
- <output> (поле для результатов вычисления в ответ на действия пользователя);
- <legend> (заголовок для содержимого тега <fieldset>);
- <datalist> (набор опций для выбора в поле ввода со свойством list).

В теге <form> указывают адрес скрипта, который обрабатывает результат заполнения формы, а также метод отправки данных (GET или POST).

Пример:

```
<form action="/action.py" method="get">
        <label for="name">Имя:</label>
        <input type="text" id="fname" name
            ="fname"><br>
        <label for="lname">Фамилия:</label>
        <input type="text" id="lname" name
            ="lname"><br>
        <label for="age">Возраст:</label>
        <input type="number" id="age" name
            ="age" max="120", min="10"><br>
        <label for="browser">Какой у вас
            браузер?</label>
        <input list="browsers" name="browser"
            id="browser"><br>
        <datalist id="browsers">
          <option value="Edge">
          <option value="Firefox">
          <option value="Chrome">
          <option value="Opera">
          <option value="Safari">
        </datalist>
        <input type="submit" value="Submit">
    </form>
```

Предосмотр width: 748px

Имя:
Фамилия:
Возраст:
Какой у вас браузер?
Submit

Рис. 39. Пример вывода формы

2.10.7 Фреймы

Фреймы представляют собой множественные веб-страницы в рамках одной. Каждый из них загружается в своем пространстве и может отдельно обновляться. На текущий момент представлен единственным тегом: <iframe>.

Пример:

```
1  <p>Первый фрейм:</p>
2  <iframe src="" width="50%" height="200">
3  </iframe>
4
5  <p>Второй фрейм:</p>
6  <iframe src="" width="100%" height="300">
7  </iframe>
```

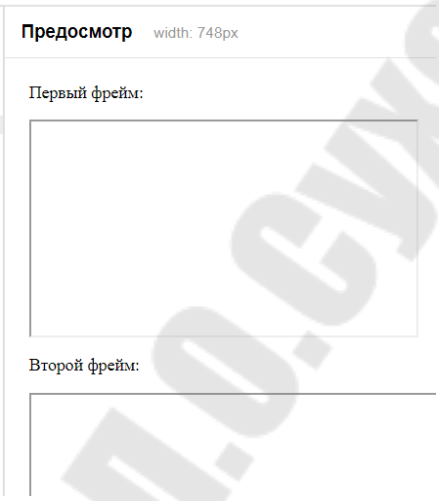


Рис. 40. Пример вывода фреймов

Приведенный пример достаточно условен, так как в качестве ссылки на фрейм требуется указать конкретный адрес или документ. Сторонние сайты в большинстве своем не позволяют установить соединение с ними через фрейм. Для элементов можно задать ширину и высоту.

2.10.8 Стили и семантика

Теги данной категории несут в себе некий смысл, понятный разработчикам, роботам, браузерам, но не посетителям сайта. Они даже не заметят, что на сайте имеется логическая верстка.

Чтобы ресурс индексировался лучше, использование семантики обязательно.

1. Тэги `<div>` и `<span>` – нейтральные теги, не имеющие смысловой нагрузки. Основная задача – соединить определенный контент в единое целое (как строку или блок). Когда нельзя по смыслу отнести содержимое к какому-то семантическому элементу, но требуется его стилизация, оно обрамляется в один из этих тегов.

Элементы не имеют уникальных атрибутов, но могут использовать любые событийные или глобальные.

Пример:

```
<body>
    <div>Блок текста</div>
    <span>Произвольный текст в строке</span>
</body>


<КАРТИНКА 25>Пример - Браузер

Элементы не имеют уникальных атрибутов, но
    могут использовать любые событийные или
    глобальные.

2. Тэг <details>
Предназначается для данных, которые
    пользователь может раскрывать или прятать.
    Это удобно в случае дополнительных
    пояснений, наличия необязательной
    информации, которую можно пропустить.
    Применяется в совокупности с тегом
    <summary>.

Пример – HTML
----
<details open>
    <summary>Дополнительная информация
        </summary>
    <p>Можно не читать</p>
</details>
```

Предосмотр width: 748px

Блок текста
Произвольный текст в строке <КАРТИНКА 25>Пример - Браузер Элементы не имеют уникальных атрибутов, но могут использовать любые событийные или глобальные. 2. Тэг
▶ . Пример – HTML ----
▼ Дополнительная информация

Можно не читать

Рис. 41. Использование тегов `<div>` и `<span>`

2. Тэг `<details>` – предназначается для данных, которые пользователь может раскрывать или прятать. Это удобно в случае дополнительных пояснений, наличия необязательной информации, которую можно пропустить. Применяется в совокупности с тегом `<summary>`.

Пример:

```
<details open>
    <summary>Дополнительная информация</summary
        >
    <p>Можно не читать</p>
</details>
```

Предосмотр width: 748px

▼ Дополнительная информация

Можно не читать

Рис. 42. Пример использования тега `<details>`

Использовано свойство `open`, которое принудительно раскрывает содержимое элемента.

3. Тэг `<section>` – данным элементом выделяют обособленную часть страницы, имеющую законченную мысль. В отличие от `<article>` (который является самостоятельным и независимым) подразумевает логический раздел документа. Пример:

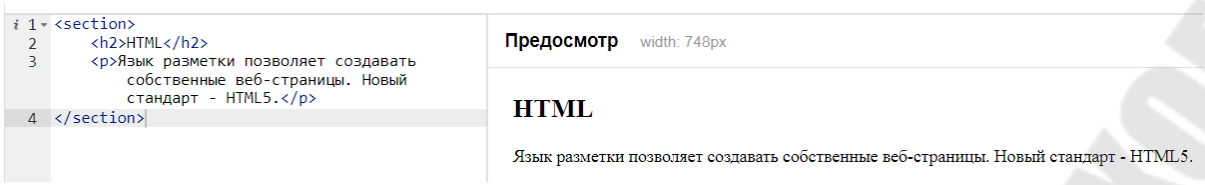


Рис. 43. Пример использования тега `<section>`

4. Тэги `<h1>...<h6>` – в качестве заголовков разделов и подразделов используются теги группы `h`. `H1` – самый крупный заголовок, а `h6` – самый маленький. Здесь важен не размер текста, а именно семантическое значение.

Так, на странице предполагается наличие одного заголовка `h1`, нескольких `h2` и т.п. Располагаются сверху вниз, от большего к меньшему `h1-h6`. Заключать в эти теги необходимо не любое содержание, а важное, так как поисковики ориентируются на них в выдаче результатов поиска.

Пример:

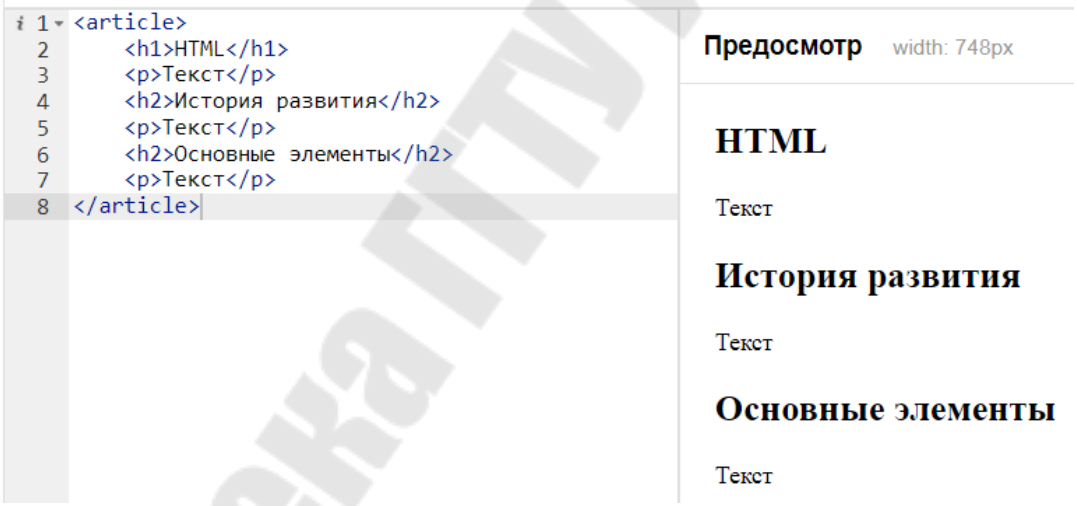


Рис. 44. Оформление текста в теги `h1 и h2`

Страница содержит информацию по HTML, о чем сказано в заголовке статьи. Подзаголовками служат разделы.

2.11 Подвал веб-страницы

Тег нижнего уровня `<footer>` используется для определения нижнего колонтитула в HTML-документе (подвала веб-страницы).

Обычно он содержит информацию о копирайте, ссылках на политику конфиденциальности или контактные данные.

Футер также может содержать другие разнообразные элементы:

- авторские права;
- навигационные ссылки - дополнительные ссылки на важные разделы сайта, такие как «О нас», «Услуги», «Контакты»;
- социальные сети - иконки с ссылками на страницы в социальных сетях (Facebook, Twitter, Instagram и т.д.);
- подписка на рассылку - форма для ввода адреса электронной почты для подписки на новости;
- карта сайта - ссылки на различные разделы сайта для удобной навигации;
- логотип компании или название.

Пример футера с различными элементами:

```
<footer>
    <div>
        <p>&copy; 2025 Мой Сайт. Все права защищены.</p>
    </div>
    <nav>
        <ul>
            <li><a href="#">О нас</a></li>
            <li><a href="#">Услуги</a></li>
            <li><a href="#">Контакты</a></li>
        </ul>
    </nav>
    <div>
        <p>Следите за нами в социальных сетях:</p>
        <a href="#">Facebook</a>
        <a href="#">Twitter</a>
        <a href="#">Instagram</a>
    </div>
    <div>
        <form action="#" method="post">
            <label for="email">Подпишитесь на нашу рассылку
                :</label>
            <input type="email" id="email" name="email"
                placeholder="Ваша электронная почта">
            <button type="submit">Подписаться</button>
        </form>
    </div>
</footer>
```

Предосмотр width: 578px

© 2025 Мой Сайт. Все права защищены.

- О нас
- Услуги
- Контакты

Следите за нами в социальных сетях:

Facebook Twitter Instagram

Подпишитесь на нашу рассылку: Ваша электронная почта Подписаться

Рис. 45. Вывод дополнительных элементов в футере

Элемент `<footer>` не является обязательным в HTML, но его рекомендуется использовать для улучшения структуры документа и семантики.

2.12 Инструменты для верстки

Верстка веб-страниц требует использования различных инструментов, которые помогают разработчикам создавать, тестировать и оптимизировать свои проекты.

Наиболее популярные инструменты и редакторы для верстки:

1. Редакторы кода:

- Visual Studio Code – один из самых популярных текстовых редакторов с поддержкой множества расширений для HTML, CSS и JavaScript. Имеет встроенный терминал и поддержку Git.
- Sublime Text – легкий и быстрый редактор с множеством плагинов. Поддерживает многоязычную подсветку синтаксиса;
- Atom – редактор от GitHub, который предоставляет возможность кастомизации и имеет встроенную поддержку Git.

2. Фреймворки и библиотеки:

- Bootstrap – популярный CSS-фреймворк, который упрощает создание адаптивных и мобильных веб-страниц с помощью готовых компонентов.
- Foundation – мощный CSS-фреймворк, предлагающий гибкие сетки и множество инструментов для разработки.
- Tailwind CSS – CSS-фреймворк, который позволяет быстро создавать пользовательские дизайны с помощью классов.

3. Инструменты для работы с графикой:

- Adobe Photoshop – используется для создания и редактирования графики, макетов и изображений для веба.
- Figma – онлайн-инструмент для дизайна интерфейсов, который позволяет создавать прототипы и взаимодействовать с командой.
- Sketch – популярный инструмент для дизайна интерфейсов, доступный только для операционной системы macOS.

4. Инструменты для тестирования и отладки:

- Google Chrome DevTools – встроенные инструменты для разработчиков в браузере Chrome, которые позволяют отлаживать HTML, CSS и JavaScript, а также анализировать производительность;
- Responsive Design Mode – инструменты в браузерах, позволяющие тестировать адаптивность страниц на разных устройствах и разрешениях.

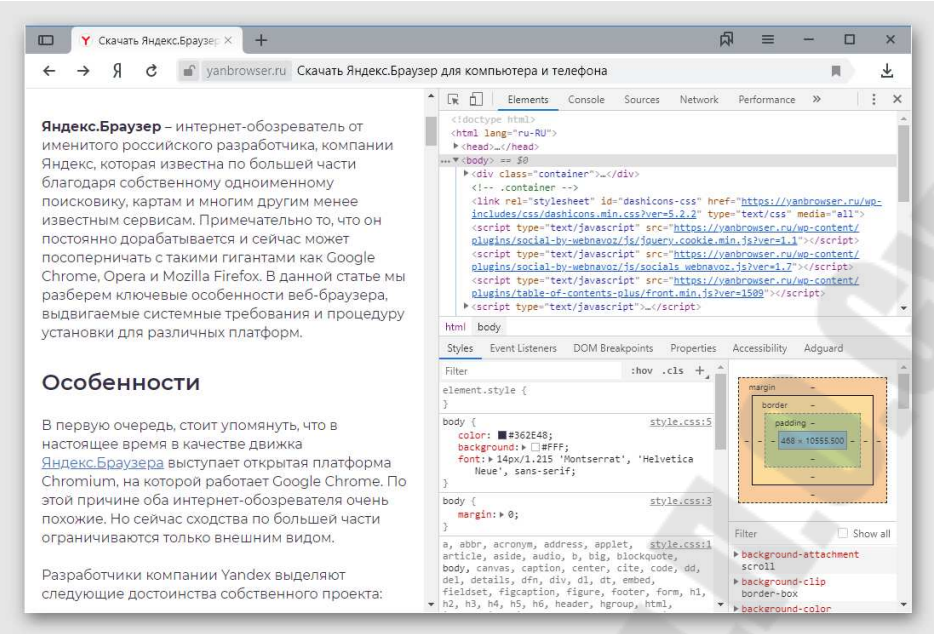


Рис. 46. Режим разработчика в браузере

5. Системы контроля версий:

- Git – система контроля версий, позволяющая отслеживать изменения в коде и работать в команде. Основной инструмент для совместной работы над проектами.

6. Инструменты для проверки кода:

- W3C Validator – онлайн-инструмент для проверки валидности HTML и XHTML-кода.
- CSS Lint – инструмент для проверки CSS-кода на наличие ошибок и потенциальных проблем.

Выбор инструментов для верстки может зависеть от предпочтений разработчика, стиля работы и требований проекта. Использование правильных инструментов поможет улучшить процесс разработки, повысить производительность и качество конечного продукта.

2.13. Типичные ошибки при использовании HTML

1. Неправильное закрытие тегов:

Ошибка: не закрываются парные теги или используется неправильная последовательность.

Решение: Убедитесь, что все открытые теги имеют соответствующее закрытие, и следуйте правильному порядку вложенности.

2. Использование устаревших тегов:

Ошибка: Применение устаревших тегов, таких как `<font>` и `<center>`.

Решение: Используйте CSS для стилизации страниц и работы с макетом.

3. Отсутствие метатегов:

Ошибка: Пропуск важных метатегов, таких как `<meta charset="UTF-8">` и `<meta name="viewport">`.

Решение: Всегда добавляйте метатеги для правильного отображения контента и адаптивности на мобильных устройствах.

4. Не семантическая разметка:

Ошибка: Использование `<div>` и `<span>` вместо семантических тегов, таких как `<header>`, `<footer>`, `<article>`.

Решение: Используйте семантические теги для улучшения структуры и доступности страницы.

5. Неправильное использование атрибутов:

Ошибка: Применение атрибутов в неверных элементах или без значений (например, `src` в тегах, где это не применяется).

Решение: Убедитесь, что атрибуты применяются только к соответствующим тегам и имеют правильные значения.

6. Пропущенные атрибуты `alt` в изображениях:

Ошибка: Отсутствие атрибута `alt` в теге `<img>`.

Решение: Всегда добавляйте атрибут `alt` для улучшения доступности и SEO.

7. Неправильный порядок тегов:

Ошибка: Использование тегов, которые должны быть дочерними к другим (например, блок `<h1>` внутри `<p>`).

Решение: Следите за правильным порядком иерархии тегов в соответствии с спецификацией HTML.

8. Инлайн стили и скрипты:

Ошибка: Частое использование встроенных стилей или скриптов вместо внешних файлов.

Решение: Старайтесь использовать внешние CSS и JavaScript файлы для улучшения организации кода и его повторного использования.

9. Отсутствие `doctype`:

Ошибка: Пропуск объявления типа документа в начале HTML-файла.

Решение: Добавляйте <!DOCTYPE html> в начало вашего документа, чтобы указать браузеру, какую версию HTML использовать.

10. Избыточные теги:

Ошибка: Использование лишних тегов или дублирование элементов может приводить к избыточности и усложнению кода.

Решение: Минимизируйте количество тегов, применяя CSS для стилизации.

11. Упрощение по значению атрибутов:

Ошибка: Применение стандартных значений для атрибутов без учета контекста (например, `style` для повторяющихся стилей).

Решение: Используйте классы или `ID` для повторяющихся стилей.

Избегая этих распространённых ошибок, можно создавать более качественные и стабильные веб-страницы, улучшая их функциональность и доступность.

3. КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ (CSS)

CSS (Cascading Style Sheets - каскадные таблицы стилей) — это язык стилей, используемый для описания внешнего вида и форматирования веб-страниц, написанных на HTML или XML.

3.1. Основы и стандарты каскадных таблиц стилей (CSS)

Стандарты CSS:

CSS разрабатывается и поддерживается W3C (World Wide Web Consortium). Стандарты CSS постоянно обновляются, и на данный момент актуальной версией является CSS3, которая включает множество новых возможностей, таких как анимации, гибкие сетки и т.д.

Основные концепции CSS:

1. Селекторы: CSS использует селекторы для выбора элементов HTML, к которым будут применяться стили. Селекторы могут быть простыми (например, по тегам) или сложными (по классам, идентификаторам и атрибутам).

Примеры селекторов:

- `h1`: выбирает все заголовки первого уровня.
- `.class-name`: выбирает все элементы с указанным классом.
- `#id-name`: выбирает элемент с указанным идентификатором.

2. Свойства и значения: Каждое правило CSS состоит из селектора и набора свойств с их значениями. Свойства определяют, каким образом будет стилизован выбранный элемент.

Пример правила CSS для заголовка первого уровня `h1`:

```
1  h1 {
2      color: blue;          /* Цвет текста */
3      font-size: 24px;      /* Размер шрифта */
4      margin: 10px;         /* Отступы */
5  }
```

Рис. 47. Пример правила CSS для заголовка первого уровня `h1`

3. Каскадность: CSS позволяет определять несколько правил для одного элемента. При этом применяются правила каскадности,

которые определяют, какое правило будет иметь приоритет. Основные принципы каскадности:

- приоритет селекторов – более специфичные селекторы имеют больший приоритет;
- последовательность – если два правила имеют одинаковую специфичность, применяется последнее объявление.

4. Медиа-запросы: CSS поддерживает адаптивный дизайн через медиазапросы, которые позволяют применять разные стили в зависимости от характеристик устройства (например, ширины экрана).

Пример медиазапроса:

```
1  @media (max-width: 600px) {
2      body {
3          background-color: lightblue;
4      }
5  }
6
```

Рис. 48. Пример медиазапроса

3.2. Способы подключения CSS к HTML-документу

Существует три основных способа подключения CSS к HTML-документу:

1. Встроенные стили (`inline styles`):

- Стили применяются непосредственно к элементам с помощью атрибута `style`. Этот метод используется для быстрого применения стилей к отдельным элементам, но не рекомендуется для общего использования, так как затрудняет поддержку и масштабируемость.

Пример:

```
1  <h1 style="color: blue; font-size: 24px;">Заголовок</h1>
2
```

Предосмотр wi

Заголовок

Рис. 49. Пример использования встроенных стилей CSS

2. Внутренние стили (`internal styles`):

- Стили определяются внутри тега `<style>` в разделе `<head>` HTML-документа. Этот метод удобен для стилизации страницы, но стили не будут применимы к другим страницам, если они не будут дублироваться.

Пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Пример внутреннего стиля</title>
    <style>
        body {
            background-color: lightgrey;
        }
        h1 {
            color: blue;
        }
    </style>
</head>
<body>
    <h1>Заголовок</h1>
</body>
</html>
```

Предосмотр width:

Заголовок

Рис. 50. Пример использования внутренних стилей CSS внутри парного тега `<head>`

3. Внешние стили (`external styles`):

- Стили хранятся в отдельном CSS-файле, который подключается к HTML-документу с помощью тега `<link>` в разделе `<head>`. Этот метод позволяет легко использовать одни и те же стили на нескольких страницах, что делает его наиболее предпочтительным.

Пример подключения внешнего файла `styles.css`:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Пример внешнего стиля</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>Заголовок</h1>
</body>
</html>
```

Предосмотр width:

Заголовок

Рис. 51. Пример подключения внешнего файла `styles.css` Заголовок по умолчанию черный.

Пример содержимого `styles.css`:

```
body {
    background-color: lightgrey;
}
h1 {
    color: blue;
}
```

Рис. 52. Пример содержимого `styles.css`

CSS является важным инструментом для веб-разработчиков, позволяющим управлять стилем и внешним видом веб-страниц. Понимание основ CSS, его стандартов и способов подключения помогает создавать красивые и адаптивные веб-дизайны, улучшая пользовательский опыт.

3.3. Селекторы CSS. Виды селекторов

Селекторы — это ключевые элементы в CSS, которые определяют, к каким HTML-элементам будут применяться заданные стили. Существует несколько видов селекторов, и их правильное использование позволяет эффективно управлять стилями на веб-страницах.

1. Селекторы по элементам (тегам) – выбирают все элементы определенного типа.

Пример: все абзацы, заключенные в тег `p` будут окрашены в цвет `blue`.

```
<p>Мороз и солнце; день чудесный!</p>

<p>Еще ты дремлешь, друг прелестный –
Пора, красавица, проснись:</p>

<p>
Пора, красавица, проснись:
Открой сомкнуты негой взоры
Навстречу северной Авроры,
Звездою севера явись!</p>
```

```
p {
    color: blue;
}
```

Мороз и солнце; день чудесный!

Еще ты дремлешь, друг прелестный — Пора, красавица, проснись:

Пора, красавица, проснись: Открой сомкнуты негой взоры Навстречу северной Авроры, Звездою севера явись!

Рис. 53. Задаем в CSS цвет текста для тега `p`

2. Селекторы по классам – выбирают элементы с указанным классом. Класс обозначается точкой (.) перед именем класса.

Пример: `.highlight` — выбирает все элементы с классом «`highlight`».

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Пример</title>
    <style>
        .highlight {
            background-color: yellow;
        }
    </style>
</head>
<body>
    <div class="highlight">Этот блок имеет желтый фон</div>
</body>
</html>
```

Предосмотр width: 578px

Этот блок имеет желтый фон

Рис. 54. Выбор элементов с классом `.highlight`

3. Селекторы по идентификаторам – выбирают элемент с указанным идентификатором. Идентификатор в CSS перед именем обозначается знаком #.

Пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Пример ID-селекторов</title>
    <style>#header {
    background-color: lightblue;
    padding: 10px;
    font-size: 24px;
}

#content {
    background-color: lightgreen;
    padding: 10px;
    font-size: 20px;
}

#footer {
    background-color: lightcoral;
    padding: 10px;
    font-size: 18px;
}
</style>
</head>
<body>
    <div id="header">Заголовок</div>
    <div id="content">Содержимое</div>
    <div id="footer">Подвал</div>
</body>
</html>
```

Предосмотр width: 578px

Заголовок

Содержимое

Подвал

Рис. 55. Выбор элемента по идентификатору

В данном примере:

- Селектор `#header` (в CSS) выбирает элемент `<div>` с идентификатором `header` и применяет к нему стили (заливка контейнера голубым цветом).

- Селектор `#content` выбирает элемент с идентификатором `content` (заливка контейнера зеленым цветом).
- Селектор `#footer` выбирает элемент с дентификатором `footer` (заливка контейнера красным цветом).

Использование селекторов по идентификаторам позволяет легко стилизовать и управлять конкретными элементами на странице.

4. Селекторы потомков – выбирают элементы, которые являются потомками другого элемента.

Пример: `div p` – выбирает все абзацы внутри элементов `<div>` и окрашивает их в зеленый цвет.

```html
<div><p>Мороз и солнце; день чудесный!</p>

<p>Еще ты дремлешь, друг прелестный –
Пора, красавица, проснись:</p></div>

<p>
Пора, красавица, проснись:
Открой сомкнуты негой взоры
Навстречу северной Авроры,
Звездою севера явись!</p>
```

```css
div p {
    color: green;
}
```

Мороз и солнце; день чудесный!

Еще ты дремлешь, друг прелестный — Пора, красавица, проснись:

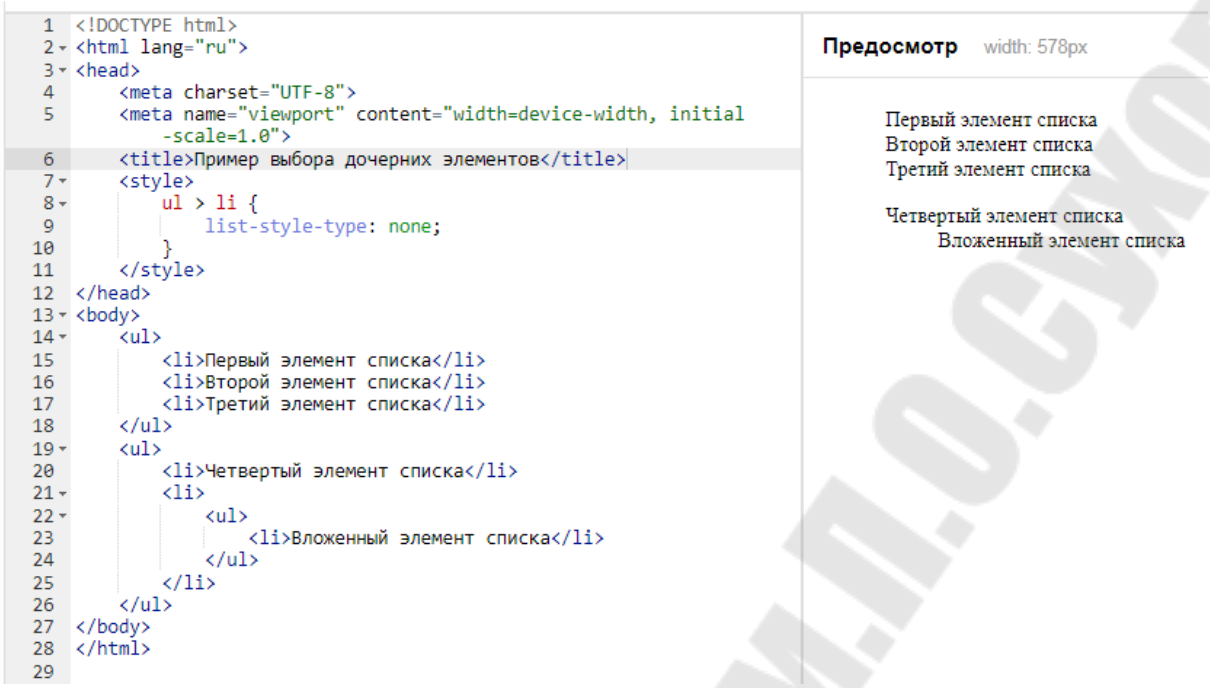
Пора, красавица, проснись: Открой сомкнуты негой взоры Навстречу северной Авроры, Звездою севера явись!

Рис. 56. Все элементы внутри тегов `p`, находящиеся в блоке `<div>` будут зеленого цвета

5. Селекторы родительских элементов:

Выбирают элементы, которые являются непосредственными дочерними элементами другого элемента.

Пример: `ul > li` — выбирает все элементы списка `<li>`, которые являются непосредственными дочерними элементами `<ul>`.



```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Пример выбора дочерних элементов</title>
    <style>
        ul > li {
            list-style-type: none;
        }
    </style>
</head>
<body>
    <ul>
        <li>Первый элемент списка</li>
        <li>Второй элемент списка</li>
        <li>Третий элемент списка</li>
    </ul>
    <ul>
        <li>Четвертый элемент списка</li>
        <li>
            <ul>
                <li>Вложенный элемент списка</li>
            </ul>
        </li>
    </ul>
</body>
</html>
```

Предосмотр width: 578px

Первый элемент списка
Второй элемент списка
Третий элемент списка

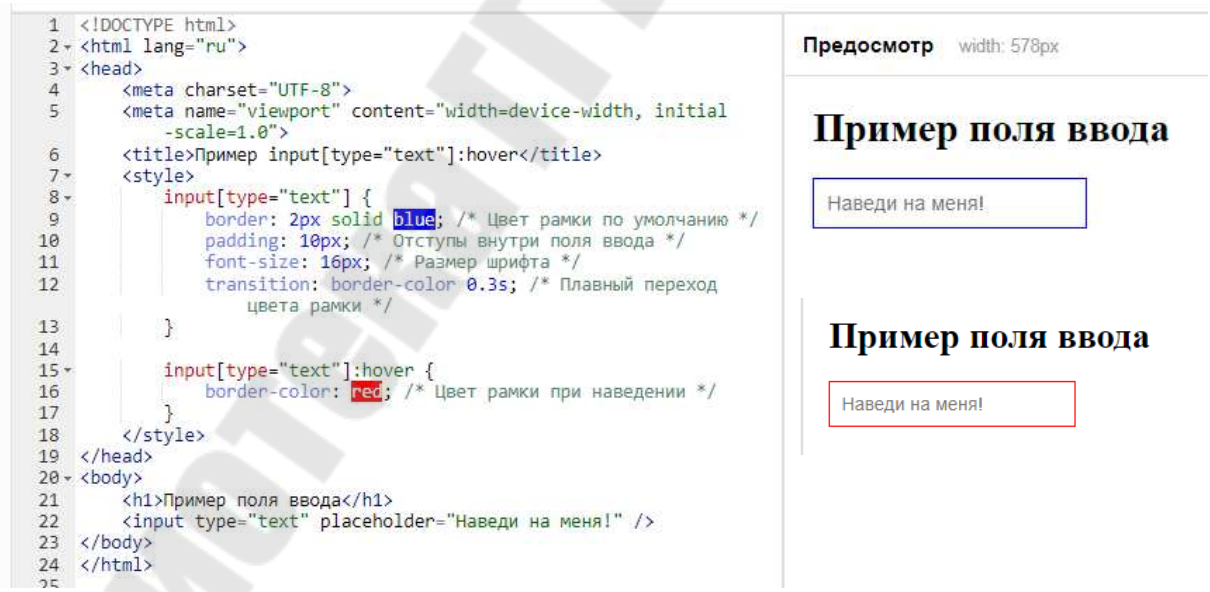
Четвертый элемент списка
Вложенный элемент списка

Рис. 57. Выбор дочерних элементов

6. Селекторы по атрибутам:

Выбирают элементы на основе значений их атрибутов.

Пример: `input[type="text"]` — выбирает все текстовые поля ввода.



```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Пример input[type="text"]:hover</title>
    <style>
        input[type="text"] {
            border: 2px solid blue; /* Цвет рамки по умолчанию */
            padding: 10px; /* Отступы внутри поля ввода */
            font-size: 16px; /* Размер шрифта */
            transition: border-color 0.3s; /* Плавный переход
                цвета рамки */
        }

        input[type="text"]:hover {
            border-color: red; /* Цвет рамки при наведении */
        }
    </style>
</head>
<body>
    <h1>Пример поля ввода</h1>
    <input type="text" placeholder="Наведи на меня!" />
</body>
</html>
```

Предосмотр width: 578px

Пример поля ввода

Наведи на меня!

Пример поля ввода

Наведи на меня!

Рис. 58. Выбор элементов на основе значений их атрибутов

При наведении курсора цвет ссылки меняется на красный. В исходном состоянии ободка блока синяя.

7. Псевдоклассы:

Выбирают элементы в зависимости от их состояния.

Пример: `a:hover` — выбирает ссылки при наведении курсора.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Пример a:hover</title>
    <style>
        a {
            color: blue; /* Цвет ссылки по умолчанию */
            text-decoration: none; /* Убираем подчеркивание */
        }

        a:hover {
            color: red; /* Цвет ссылки при наведении */
        }
    </style>
</head>
<body>
    <h1>Пример ссылки</h1>
    <p>Наведи на <a href="#">эту ссылку</a>, чтобы увидеть
        изменение цвета.</p>
</body>
</html>
```

Предосмотр width: 578px

Пример ссылки

Наведи на эту ссылку, чтобы увидеть изменение цвета.

Пример ссылки

Наведи на эту ссылку, чтобы увидеть изменение цвета.

Рис. 59. Изменение цвета ссылки после изменения цвета курсора

В данном примере при наведении курсора цвет ссылки меняется на красный. В исходном состоянии цвет ссылки синий.

8. Псевдоэлементы:

Позволяют стилизовать определенные части элементов.

Пример: `p::first-line` — выбирает первую строку абзаца.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Пример a:hover</title>
    <style>
      p::first-line  {
            font-weight: bold;
        }
    </style>
</head>
<body>
    <h1>Стихотворение</h1>
    <p>Мороз и солнце; день чудесный!</p>
Еще ты дремлешь, друг прелестный<br> –
Пора, красавица, проснись:<br>
Открой сомкнуты негой взоры<br>
Навстречу северной Авроры,<br>
Звездою севера явись!</p>
</html>
```

Предосмотр width: 578px

Стихотворение

Мороз и солнце; день чудесный!

Еще ты дремлешь, друг прелестный
— Пора, красавица, проснись:
Открой сомкнуты негой взоры
Навстречу северной Авроры,
Звездою севера явись!

Рис. 60. Стилизация определенных частей элементов

3.4. Правила формирования и чтения селекторов

Чтение селекторов: Селекторы читаются справа налево, начиная с самого глубокого уровня. Например, в селекторе `div p.highlight`, сначала выбираются все элементы `<p>` с классом «`highlight`», а затем только те, которые находятся внутри элементов `<div>`.

Комбинирование селекторов: Селекторы могут комбинироваться для более точного выбора элементов. Например, `div.highlight` выберет только элементы `<div>` с классом «`highlight`».

3.5. Наследование и каскадность

1. Наследование:

Некоторые CSS-свойства наследуются от родительских элементов к дочерним. Это означает, что если свойство задано для родительского элемента, дочерние элементы могут автоматически использовать его значение.

Примеры наследуемых свойств: `color`, `font-family`, `line-height`. Примеры не наследуемых свойств: `margin`, `padding`, `border`.

Пример наследования:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Пример наследования</title>
    <style>
.parent {
    color: blue; /* Наследуемое свойство */
    font-family: Arial, sans-serif; /* Наследуемое свойство */
    margin: 20px; /* Не наследуемое свойство */
}
.child {
    font-size: 18px;
    color: green;
}
</style>
</head>
<body>
    <div class="parent">
        Родительский элемент
        <p class="child">Дочерний элемент</p>
    </div>
</body>
</html>
```

Предосмотр width: 578px

Родительский элемент

Дочерний элемент

Рис. 61. Пример наследования

Данный пример демонстрирует, как некоторые свойства в CSS наследуются от родительских элементов к дочерним:

1. Родительский элемент (`.parent`) имеет заданные свойства `color` и `font-family` (шрифт `Arial`). Свойства шрифта и цвета будет автоматически наследоваться дочерним элементом (`<p class="child">`).

2. Дочерний элемент (`.child`) будет отображаться цветом текста зеленым, а не синим и иметь размер шрифта 18`px` (`font-size`) – потому что он имеет свои собственные значения.

3. Свойство `margin` в родительском элементе не будет наследоваться дочерним элементом, т.к. это свойство не наследуемое; поэтому дочерний элемент будет иметь свои собственные значения для отступов.

2. Каскадность:

Каскадность относится к тому, как браузер решает, какие стили применять к элементу, когда есть стили, определенные в разных источниках. Основные источники стилей могут включать:

- встроенные стили (`inline styles`);
- внешние таблицы стилей (`external stylesheets`);
- встроенные таблицы стилей (`internal styles`).

Каскадное правило определяет, какое правило CSS будет применяться, если несколько правил применимы к одному элементу:

- Поздние определения приоритетнее ранних: Если стили определены в разных местах, стили, указанные позже, будут приоритетнее.

- Важно место определения в коде: Например, если одно и то же свойство определено в нескольких стилях, будет применяться последнее объявление в каскадной системе.

Селекторы в CSS — это мощный инструмент, позволяющий точно управлять стилями веб-страниц. Понимание принципов наследования и каскадности помогает разработчикам эффективно организовывать и управлять стилями, обеспечивая гибкость и устойчивость к изменениям в дизайне.

3.6. Специфичность селектора

Специфичность селектора в CSS определяет, какой стиль применяется к элементу, если к нему применяются несколько правил. Специфичность рассчитывается на основе типа селектора и имеет следующую иерархию:

1. Инлайн-стили (например, `style="..."`) имеют наивысшую специфичность.
2. ID-селекторы (например, `#id`) имеют более высокую специфичность, чем классы и теги.
3. Классы, атрибуты и псевдоклассы (например, `.class`, `[type="text"]`, `:hover`) имеют среднюю специфичность.
4. Теговые селекторы (например, `div`, `p`) имеют наименьшую специфичность.
5. Универсальный селектор (`*`) и селекторы потомков имеют самую низкую специфичность.

Пример специфичности селектора 1:

```
1  /* Наименьшая специфичность */
2  p {
3      color: blue;
4  }
5
6  /* Средняя специфичность */
7  .class {
8      color: red;
9  }
10
11 /* Высокая специфичность */
12 #id {
13     color: green;
14 }
15
16 /* Наивысшая специфичность */
17 <div style="color: yellow;">Text</div>
18
```

Рис. 62. Порядок специфичности

Если к элементу применяются все эти стили, будет применен стиль с наивысшей специфичностью, то есть в данном случае текст будет желтым.

Пример специфичности 2:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
    <title>Пример специфичности</title>
    <style>
/* Селектор по классу (специфичность = 0-1-0) */
.box {
    color: green;
}

/* Селектор по идентификатору (специфичность = 1-0-0) */
#text {
    color: red;
}

/* Селектор по тегу и классу (специфичность = 0-1-1) */
.box p {
    font-weight: bold;
}
    </style>
</head>
<body>
    <div class="box">
        <p id="text">Пример текста</p>
    </div>
</body>
</html>
```

Предосмотр width:

Пример текста

Рис. 63. Пример применения стилей к элементам с учетом специфичности

Результат:

- Текст «Пример текста» будет красным из-за идентификатора `#text`.

- Текст будет жирным, так как применяется стиль `.box p`.

Объяснение специфичности:

1. `p` — селектор по тегу, его специфичность 0-0-1. Применяет синий цвет к любому `<p>`.

2. `.box` — селектор по классу, его специфичность 0-1-0. Применяет зеленый цвет к тексту внутри элемента с классом box.

3. `#text` — селектор по идентификатору, его специфичность 1-0-0. Применяет красный цвет к элементу с идентификатором text. Этот стиль будет иметь приоритет над стилем для p и box, поскольку у него более высокая специфичность.

4. `.box p` — селектор по классу и тегу, его специфичность 0-1-1. Применяет жирное начертание к тексту внутри `<p>`, который находится в элементе с классом `box`.

Таким образом, при использовании нескольких селекторов, специфичность помогает определить, какие стили будут применены к элементам. В данном примере, несмотря на то, что класс `.box` задает зеленый цвет, идентификатор `#text` переопределяет его, так как имеет более высокую специфичность.

Информация о специфичности селекторов и каскадировании доступна в спецификации www.w3.org.

Итак:

- Каскадность определяет, как применяются стили из разных источников.
- Специфичность определяет приоритет стилей, когда стили от различных селекторов конфликтуют.

Оба концепта работают совместно для определения конечного стиля для элементов на странице.

4. ФРЕЙМВОРКИ CSS

Фреймворки CSS представляют собой наборы инструментов и библиотек, которые упрощают процесс стилизации веб-страниц. Они предоставляют разработчикам готовые решения для создания адаптивных и кроссбраузерных интерфейсов, что позволяет ускорить процесс разработки и улучшить качество конечного продукта.

4.1. Введение в фреймворки

Фреймворк – это готовый набор инструментов, который помогает разработчику быстро создать продукт. Фреймворки помогают: ускорить разработку и предотвратить ошибки. Благодаря фреймворкам разработчик может избежать распространённых ошибок — как архитектурных, так и функциональных.

Пример наиболее популярных CSS-фреймворков в веб-разработке:

- Bootstrap - популярный CSS-фреймворк для создания отзывчивого дизайна;
- Tailwind CSS - утилитарный CSS-фреймворк для кастомизации;
- Bulma - современный CSS-фреймворк с гибкой сеткой.

Фреймворки CSS предлагают ряд преимуществ, которые делают их популярными среди веб-разработчиков:

- стандартизация – фреймворки следуют определенным стандартам и лучшим практикам, что способствует созданию чистого и понятного кода;
- готовые компоненты – большинство фреймворков предлагают набор готовых компонентов (кнопки, формы, навигационные элементы и т.д.), что значительно ускоряет процесс разработки;
- адаптивный дизайн – фреймворки часто включают в себя системы сеток, которые позволяют легко создавать адаптивные макеты, подходящие для различных устройств и экранов;
- кроссбраузерная совместимость – фреймворки обеспечивают совместимость с различными браузерами, что упрощает разработку и тестирование;

- сообщество и поддержка – популярные фреймворки имеют большое сообщество разработчиков, что обеспечивает доступ к документации, учебным материалам и поддержке;
- кастомизация – многие фреймворки позволяют настраивать стили и компоненты, чтобы соответствовать требованиям конкретного проекта.

4.2. Фреймворк Bootstrap

Bootstrap — это один из самых популярных фреймворков CSS, разработанный компанией Twitter. Он предоставляет мощные инструменты для создания адаптивных и мобильных веб-приложений, а также упрощает процесс стилизации интерфейсов.

Bootstrap часто называют «фреймворком CSS», так как он предоставляет готовый набор CSS-стилей и компонентов для упрощения разработки веб-интерфейсов.

Bootstrap включает:

1. Стилевые правила CSS (`bootstrap.css`): для оформления элементов, таких как кнопки, таблицы, формы и др.

В файле CSS, предоставляемом Bootstrap, содержатся стили для оформления различных компонентов, включая классы для кнопок, форм, навигационных панелей и многих других элементов. Например, классы `.btn` для создания кнопок, `.alert` для уведомлений и `.card` для карточек с контентом.

Примеры:

Заголовок	Пример
`<h1></h1>`	h1. Заголовок Bootstrap
`<h2></h2>`	h2. Заголовок Bootstrap
`<h3></h3>`	h3. Заголовок Bootstrap
`<h4></h4>`	h4. Заголовок Bootstrap
`<h5></h5>`	h5. Заголовок Bootstrap
`<h6></h6>`	h6. Заголовок Bootstrap

Рис. 64. Стилевые правила CSS для заголовков

Чтобы узнать о доступных классах и их использовании, можно обратиться к официальной документации Bootstrap на сайте getbootstrap.com. Документация содержит примеры кода, описания классов и варианты настройки компонентов, что позволяет легко ориентироваться и эффективно применять стили в ваших проектах.

2. Компоненты Bootstrap

Перечень популярных компонентов Bootstrap:

- Система сеток (`grid system`): для создания адаптивной и отзывчивой верстки;

В приведенной таблице показано, как меняется ширина контейнера в зависимости от девайса.

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	X-Large ≥1200px	XX-Large ≥1400px
.container	100%	540px	720px	960px	1140px	1320px
.container-sm	100%	540px	720px	960px	1140px	1320px
.container-md	100%	100%	720px	960px	1140px	1320px
.container-lg	100%	100%	100%	960px	1140px	1320px
.container-xl	100%	100%	100%	100%	1140px	1320px
.container-xxl	100%	100%	100%	100%	100%	1320px
.container-fluid	100%	100%	100%	100%	100%	100%

Рис. 65. Система сеток

Система сеток в Bootstrap помогает создавать гибкие и адаптивные макеты, которые корректно отображаются на различных устройствах.

- `Navigation Bar` –верхняя панель навигации.

Готовый код и визуализация Navigation Bar (меню):

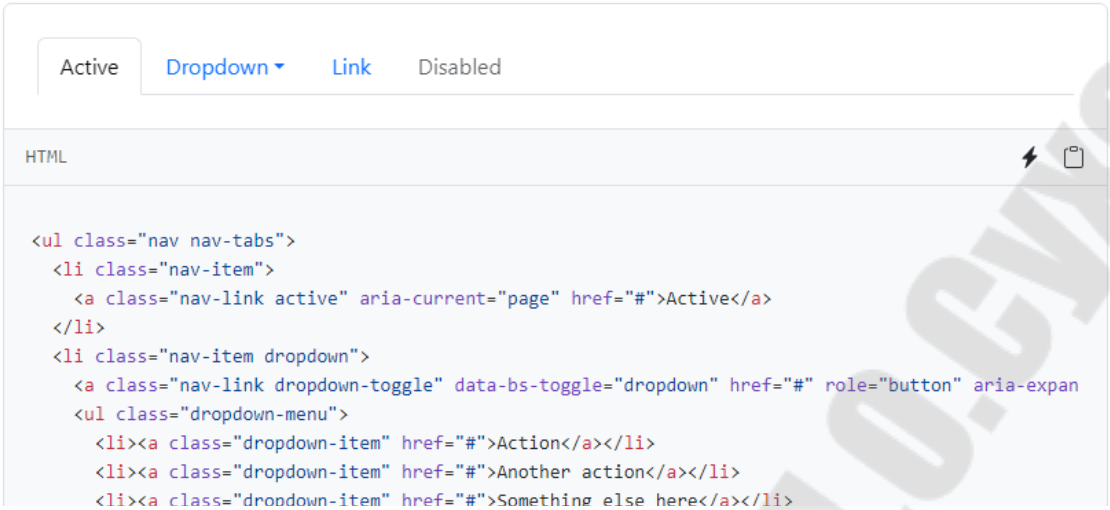


Рис. 66. Пример готового кода и визуализация Navigation Bar (меню)

- `Buttons` – стилизованные кнопки.

В этом примере для стилизации кнопок (готовые компоненты) используется классы `btn-...`

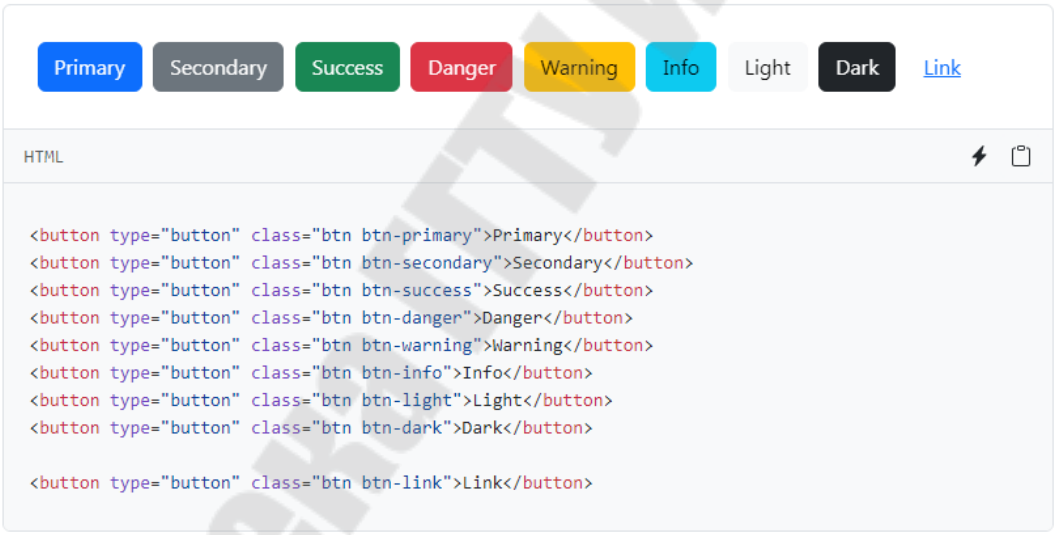


Рис. 67. Код кнопок и их стилизация классами Bootstrap

А также:

- `Forms` –элементы форм (ввод, выбор, текстовые области и т.д.).
- `Modals` – всплывающие окна для отображения информации.
- `Dropdowns` –выпадающие меню.
- `Cards` –карточки для отображения контента.
- `Carousel` – слайдер изображений или контента.
- `Tooltips` – подсказки при наведении на элементы.

- `Pagination` – для навигации по страницам.
- `Breadcrumbs` – навигационные цепочки.
- `Spinners` – индикаторы загрузки.

Эти компоненты позволяют быстро создавать адаптивные и стильные интерфейсы с помощью Bootstrap.

- **JavaScript-компоненты**: для добавления интерактивности (например, модальные окна, карусели).

Таким образом, он упрощает процесс разработки и делает его более эффективным, что и делает Bootstrap популярным среди веб-разработчиков.

Для начала работы с Bootstrap необходимо подключить его к проекту с официального сайта. Можно использовать предлагаемый способ (через CDN) или скачать файлы `Bootstrap` на ПК.

Подключение через CDN:

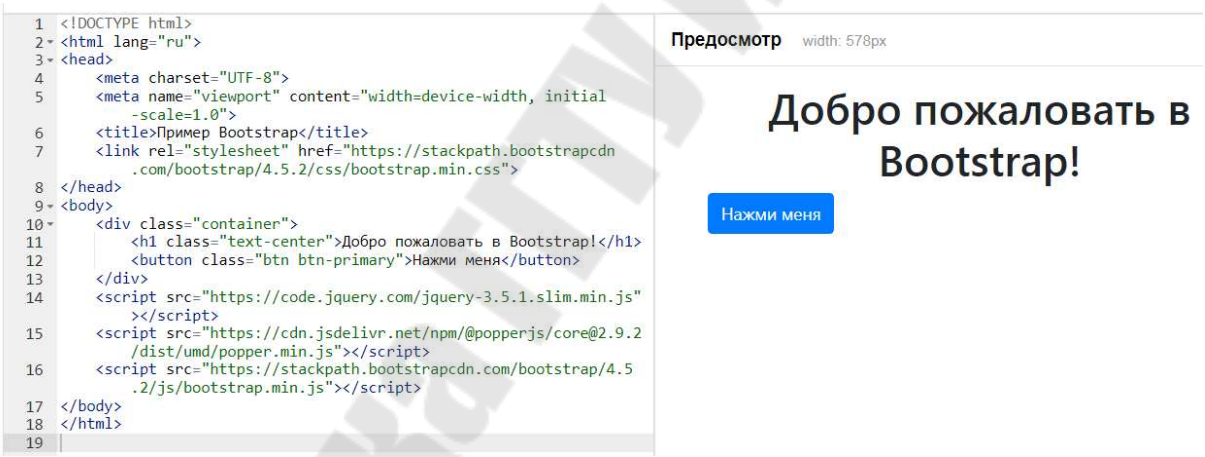


Рис. 68. Подключение Bootstrap.min.css через CDN

В этом примере мы подключаем Bootstrap через CDN и используем класс `container` для создания обертки, а также класс `btn` для стилизации кнопки.

Контейнеры являются самым основным элементом макета в Bootstrap и необходимы при использовании сеточной системы по умолчанию. Контейнеры используются для содержания, заполнения и (иногда) центрирования содержимого внутри них.

Преимущества Bootstrap:

- адаптивность – легко создавать адаптивные дизайны, которые хорошо выглядят на любых устройствах;
- быстрая разработка – готовые компоненты и стили позволяют быстро прототипировать и разрабатывать интерфейсы;
- совместимость – широкая поддержка браузеров и устройств.

Фреймворки CSS, такие как Bootstrap, значительно упрощают процесс разработки веб-приложений, предоставляя разработчикам готовые решения для создания адаптивных и красивых интерфейсов. Знание основ работы с Bootstrap позволяет быстро и эффективно разрабатывать современные веб-сайты и приложения.

5. СТИЛИ И ЕДИНИЦЫ ИЗМЕРЕНИЯ CSS

5.1. Сброс стилей браузера

Сброс стилей (reset CSS) — это метод, используемый для удаления или нормализации стандартных стилей, которые браузеры применяют к HTML-элементам по умолчанию. Сброс помогает:

- избежать конфликтов и разброса значений, основанных на начальных браузерных стилях;
- обеспечить единый старт для всех браузеров, что делает дальнейшую работу со стилями и единицами измерения более эффективной и контролируемой.

Сброс создает основу для дальнейшей стилизации, позволяя разработчикам устанавливать свои собственные значения для отступов, шрифтов, размеров и других атрибутов, заботясь о единообразии.

Пример сброса стилей:

```
1  * {
2      margin: 0;
3      padding: 0;
4      box-sizing: border-box;
5  }
6
```

Рис. 69. Код в CSS - сброс стилей

Существуют также более полные библиотеки сброса стилей, такие как Normalize.css, которые не просто обнуляют стили, а обеспечивают более согласованное поведение элементов в разных браузерах.

5.2. Единицы измерения в CSS

CSS поддерживает несколько единиц измерения для задания размеров, отступов и других свойств. Основные единицы измерения:

1. Абсолютные единицы:

- `px` (пиксели) — фиксированная единица, которая не зависит от других факторов;

- pt (пункты) — используется в печати, 1 пункт примерно равен 1/72 дюйма;

- in (дюймы), cm (сантиметры), mm (миллиметры) — также абсолютные единицы.

2. Относительные единицы:

- % (проценты) — относительная единица к родительскому элементу;

- em — размер шрифта родительского элемента. Например, если родительский элемент имеет размер шрифта 16px, то 1em будет равен 16px;

- rem — размер шрифта корневого элемента (обычно <html>);

- vw (viewport width) — 1% от ширины окна просмотра;

- vh (viewport height) — 1% от высоты окна просмотра.

Использование относительных единиц позволяет создавать более адаптивные и гибкие дизайны, которые лучше реагируют на изменения размеров экранов и устройств.

5.3. Способы кодировки цвета в CSS

Способы кодировки цвета в CSS являются неотъемлемой частью работы со стилями и единицами измерения:

- Согласование цвета: Понимание различных способов кодировки цвета позволяет разработчикам выбирать наиболее подходящие для конкретных случаев, что является частью общей стилизации.

- Взаимодействие с единицами измерения: Цвета могут влиять на восприятие размеров и расстояний. Например, использование цвета для фона и текста может влиять на читаемость и визуальное восприятие элементов, основанных на единицах измерения (например, px, em, %).

В CSS существует несколько способов задания цвета:

1. Именованные цвета:

CSS поддерживает более 140 предопределённых именованных цветов, таких как red, blue, green, black, white и т.д.

```
1    color: red;
2
```

Рис. 70. Пример простого цвета

Используется для простых и понятных значений, когда достаточно базовых цветов.

2. HEX-коды:

Цвета можно задавать в шестнадцатеричном формате. HEX-код состоит из знака решётки (#) и шести цифр (две для красного, две для зелёного и две для синего).

```
1    color: #ff0000; /* Красный */
2    color: #00ff00; /* Зеленый */
3    color: #0000ff; /* Синий */
4
```

Рис. 71. Пример цвета в HEX-коде

Используется для точного указания цвета. HEX-коды дают возможность выбрать любой цвет в цифровом формате. Часто используются в веб-дизайне.

3. RGB и RGBA:

Цвета можно задавать с помощью RGB (красный, зелёный, синий), где каждая компонента принимает значение от 0 до 255. RGBA добавляет альфа-канал для прозрачности.

```
1    color: rgb(255, 0, 0);     /* Красный */
2    color: rgba(255, 0, 0, 0.5); /* Полупрозрачный красный */
3
```

Рис. 72. Пример цвета в RGB и RGBA -кодах

Используется, когда требуется именно RGB-значение. Удобно для настройки цветовой модели через три цветовых канала.

4. HSL и HSLA:

Цвета можно задавать с помощью HSL (оттенок, насыщенность, светлота). HSLA также добавляет альфа-канал для прозрачности.

```
1    color: hsl(0, 100%, 50%);     /* Красный */
2    color: hsla(0, 100%, 50%, 0.5); /* Полупрозрачный красный */
3
```

Рис. 73. Пример красного цвета в HSL и HSLA -кодах

Цветовая модель HSL удобна при работе с оттенками.

Выбор метода зависит от контекста: простые имена цветов подойдут для быстрого использования, HEX-коды и RGB/RGBA — для точных значений, а HSL/HSLA — для высококонтрастной работы с цветовыми тонами.

6. ШРИФТЫ И ИКОНКИ

6.1. Подключение дополнительных шрифтов

Подключение дополнительных шрифтов позволяет разработчикам и дизайнерам значительно расширять возможности стилизации и повышать качество пользовательского опыта на сайте:

1. Улучшение визуальной эстетики: Дополнительные шрифты дают возможность создавать уникальный и привлекательный дизайн, отражая стиль и индивидуальность бренда.
2. Повышение читабельности: Использование специализированных шрифтов может улучшить читаемость текста, особенно для заголовков и важных разделов, что способствует лучшему восприятию информации.
3. Создание иерархии: Разные шрифты и их веса могут быть использованы для создания визуальной иерархии, что помогает пользователю быстрее ориентироваться на странице.
4. Согласованность с брендом: Если компания уже использует определённые шрифты в своих материалах (логотипах, визитках и т.д.), их подключение на сайте помогает сохранить консистентный стиль.
5. Доступность: Некоторые шрифты могут включать специальные символы или акценты, которые могут быть полезны для целевой аудитории (например, шрифты для разных языков).
6. Адаптивный дизайн: Дополнительные шрифты, такие как Google Fonts, упрощают адаптацию к различным устройствам, так как они могут автоматически подстраиваться под размеры экрана.

Для подключения дополнительных шрифтов в CSS существует несколько методов:

1. Google Fonts:

Один из самых популярных способов подключения веб-шрифтов являются шрифты от Google - Google Fonts. Веб-шрифты — это шрифты, которые загружаются с удаленного сервера и отображаются на веб-странице, даже если они не установлены на устройстве пользователя.

На сервисе Google Fonts необходимо выбрать шрифт, получить на него ссылку и вставить её в `<head>` в HTML-документ.

Рис. 74. Веб-шрифты Google Fonts

```
<link href="https://fonts.googleapis.com/css2?family=Roboto
    :wght@400;700&display=swap" rel="stylesheet">
```

Рис. 75. Подключение выбранного веб-шрифта на сайт

Затем в CSS установленный шрифт можно использовать, указав его в CSS:

```
body {
    font-family: 'Roboto', sans-serif;
}
```

Рис. 76. Указание в CSS установленного шрифта

2. `@font-face`:

С помощью правила `@font-face` можно подключить собственные шрифты, которые уже загружены в папку `fonts` сайта:

```
@font-face {
    font-family: 'MyCustomFont';
    src: url('fonts/MyCustomFont.woff2') format('woff2'),
         url('fonts/MyCustomFont.woff') format('woff');
    font-weight: normal;
    font-style: normal;
}
```

Рис. 77. Подключение собственного шрифта

Использование установленного шрифта с помощью CSS:

```
body {
    font-family: 'MyCustomFont', sans-serif;
}

```

Рис. 78. Указание в CSS установленного шрифта

6.2. Использование иконочных шрифтов

Иконочные шрифты — это шрифты, которые содержат символы и иконки вместо букв. Один из самых популярных наборов иконочных шрифтов – Font Awesome.

Font Awesome – это библиотека и сервис иконок, который предоставляет большой выбор векторных иконок и шрифтов. Он используется для добавления иконок в веб-проекты без необходимости загружать изображения или графику.

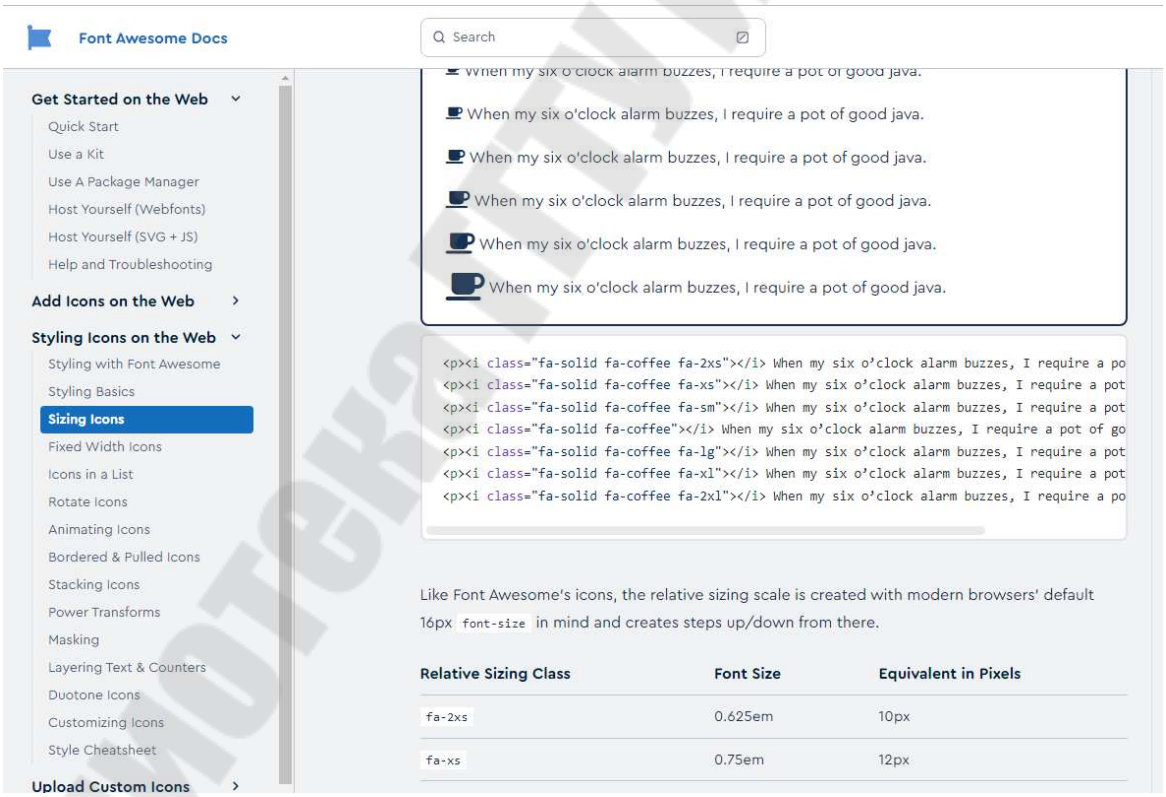


Рис. 79. Сайт Font Awesome

1. Подключение шрифтов Font Awesome:

Подключить шрифты можно с официального сайта Font Awesome с помощью предлагаемого способа CDN (копирование

специальной ссылки), добавив следующий код в <head> HTML-документа:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax
    /libs/font-awesome/6.0.0-beta3/css/all.min.css">
```

Рис. 80. Подключение на сайт шрифтов Font Awesome

2. Использование иконок:

После подключения иконочных шрифтов можно использовать иконки, добавляя соответствующий класс (class) к элементу <i> или .

```
<i class="fas fa-camera"></i> <!-- Иконка камеры -->
<i class="fab fa-github"></i> <!-- Иконка GitHub -->
```

Рис. 81. Классы иконок для элементов

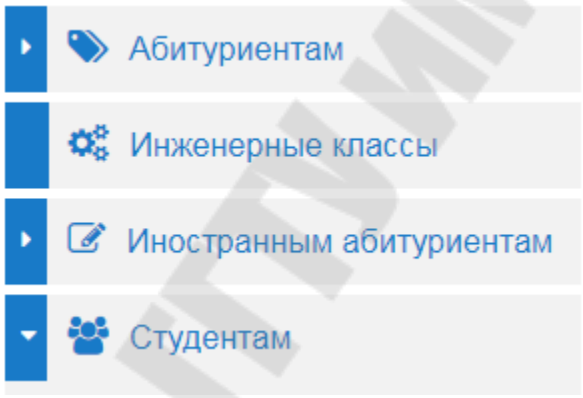


Рис. 82. Пример использования на сайте иконочных шрифтов

Иконочные шрифты позволяют легко добавлять иконки на веб-страницы, используя CSS для стилизации и настройки их внешнего вида.

6.3. Лучшие практики работы с шрифтами и иконками

Лучшие практики работы с шрифтами:

1. Чтобы обеспечить хорошую совместимость и легкую загрузку веб-шрифтов применяют доверенные сервисы, такие как Google Fonts или Adobe Fonts.

2. Чтобы сохранить целостность дизайна и улучшить читаемость на странице используют не более 2-3 различных шрифтов.

3. Выбирают шрифты, которые соответствуют стилю проекта (оформление, тональность и аудитория).

4. Для улучшения визуального восприятия, через `font-weight` и `font-style` указывают начертание шрифта (обычный, жирный, курсив).

5. Чтобы обеспечить иерархию информации применяют различные размеры шрифтов для семантических заголовков (`h1`, `h2`, и т.д.).

6. Для уменьшения размера файла шрифта и ускорения загрузки страниц используют только необходимые наборы символов и начертания.

7. Не применяют тонкие шрифты на сложном фоне, чтобы обеспечить хорошую читаемость, особенно для пользователей с нарушениями зрения.

8. Для обеспечения адаптивности и лучшего взаимодействия с настройками браузера пользователя для шрифтов применяют относительные единицы измерения (например, `em`, `rem`).

Лучшие практики работы с иконками:

1. Интегрирование иконочных шрифтов в проект позволяют сохранять четкое качество отображения (например, `Font Awesome`, `Material Icons`).

2. Чтобы получить масштабируемость без потери качества и меньший размер (вес), по сравнению с растровыми изображениями, для иконок используют SVG-файлы.

3. Для улучшения доступности контента используют семантические теги.

5. Для обеспечения согласованности в дизайне выбирают иконки с единым стилем.

7. Для обеспечения доступности иконок для пользователей с нарушениями слуха или зрения используют атрибуты `alt` или `aria-label`.

8. Необходимо избегать избыточного количества иконок, чтобы не перегружать интерфейс.

Применение этих лучших практик поможет создать качественный, профессиональный и доступный веб-дизайн, где шрифты и иконки будут служить как вспомогательными элементами в визуальном восприятии информации.

9. УПРАВЛЕНИЕ ВНЕШНИМ ВИДОМ ЭЛЕМЕНТОВ

Управление внешним видом элементов и основы модели визуального форматирования — это ключевые аспекты веб-дизайна и верстки в CSS.

9.1 Управление внешним видом элементов

CSS предоставляет множество свойств для управления внешним видом HTML-элементов. Вот некоторые из наиболее распространенных свойств:

1. Цвет и фон:

- `color`: задает цвет текста;
- `background-color`: задает цвет фона элемента;
- `background-image`: позволяет установить изображение в качестве фона.

Пример:

```
p {
    color: blue;
    background-color: lightgray;
    background-image: url('background.jpg');
}
```

Рис. 83. Представление в CSS цвета шрифта и фона

2. Шрифт:

- `font-family`: задает шрифт текста;
- `font-size`: задает размер шрифта;
- `font-weight`: задает толщину шрифта (например, `normal`, `bold`).

Пример:

```
h1 {
    font-family: 'Arial', sans-serif;
    font-size: 24px;
    font-weight: bold;
}
```

Рис. 84. Стилизация шрифта в CSS

3. Отступы и поля:

- `margin`: задает внешние отступы элемента;
- `padding`: задает внутренние отступы элемента.

Пример:

```
div {
    margin: 20px;   /* Внешние отступы */
    padding: 10px;  /* Внутренние отступы */
}
```

Рис. 85. Внешние и внутренние отступы для блока `div`

4. Границы:

- `border`: задает границу вокруг элемента, включая цвет, стиль и ширину. Пример:

```
.box {
    border: 2px solid black;  /* Черная сплошная граница */
}
```

Рис. 86. Элементам класса `.box` задана черная сплошная граница

5. Размеры и позиционирование:

- `width` (ширина) и `height` (высота): задают размеры элемента.
- `position`: определяет способ позиционирования элемента (`static, relative, absolute, fixed, sticky`).
- `display`: определяет способ отображения элемента (`block, inline, inline-block, flex, grid` и т.д.).

Пример:

```
.container {
    width: 80%;
    height: 400px;
    position: relative;
    display: flex;
}
```

Рис. 87. Контейнеру задана ширина, высота и способ отображения

9.2 Основы модели визуального форматирования

Модель визуального форматирования в CSS описывает, как браузер отображает элементы на странице. Основные концепции модели визуального форматирования включают:

1. Блочная и строчная модель:

- блочные элементы (например, `<div>`, `<h1>`, `<p>`) – занимают всю ширину доступного пространства и начинаются с новой строки;
- строчные элементы (например, `<span>`, `<a>`, `<strong>`) – занимают только необходимую ширину и не начинают новую строку.

2. Контекст форматирования – каждый элемент может создавать свой контекст форматирования, который влияет на то, как его дочерние элементы располагаются и отображаются.

3. Модель коробки (Box Model) – каждый элемент на веб-странице представлен как прямоугольная коробка, состоящая из следующих частей:

- Содержимое (`content`) – фактический контент элемента (текст, изображения и т.д.).
- Отступы (`padding`) – пространство между содержимым и границей элемента.
- Границы (`border`) – линия, окружающая элемент.
- Внешние отступы (`margin`) – пространство между элементами.

Модель областей (`box`) CSS:

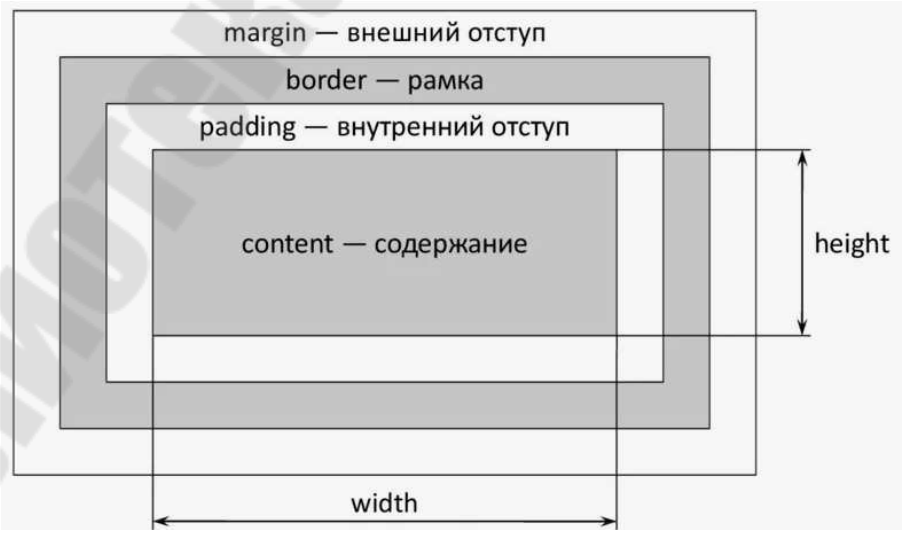


Рис. 88. Модель областей CSS

Пример:

```
.box {
    width: 200px;         /* Ширина содержимого */
    padding: 10px;       /* Отступы */
    border: 5px solid black; /* Граница */
    margin: 20px;        /* Внешние отступы */
}

```

Рис. 89. Свойства CSS для модели `.box`

4. Поток документа – элементы располагаются в потоке документа, что означает, что их расположение зависит от того, как они организованы в HTML и как применяются стили CSS.

Эти концепции позволяют разработчикам управлять внешним видом и расположением элементов на веб-странице, создавая привлекательные и удобные интерфейсы.

10. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ВЕРСТКИ

CSS Grid и Flexbox — это две мощные технологии для создания макетов на веб-страницах. Каждая из них предназначена для решения различных задач, и они могут использоваться как отдельно, так и в комбинации друг с другом.

10.1. Технология Grid

CSS Grid — это более мощная и гибкая технология для создания двумерных макетов (по строкам и колонкам). Она предоставляет возможность управлять размещением элементов в сетке.

Преимущества гибких сеток:

- Адаптивность: Гибкие сетки позволяют элементам автоматически изменять размеры и положение в зависимости от размеров экрана, что делает сайт удобным для пользователей на различных устройствах.

- Упрощение макета: Использование Flexbox и Grid упрощает создание сложных макетов без необходимости использования множества вспомогательных классов и позиционирования.

- Управление пространством: Гибкие сетки позволяют легко управлять расстоянием между элементами, что улучшает визуальное восприятие.

Примеры применения гибких сеток

1. Галерея изображений:

Создание адаптивной галереи изображений, где изображения автоматически располагаются в сетке.

```
.gallery {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr
        ));
    gap: 10px; /* Расстояние между изображениями */
}
```

Рис. 90. Грид-сетка галереи изображений

2. Карточки товаров:

Создание сетки для отображения карточек товаров в интернет-магазине.

```css
.product-grid {
    display: flex;
    flex-wrap: wrap;
}
.product-card {
    flex: 1 1 300px; /* Минимальная ширина карточки 300px */
    margin: 10px;
}
```

Рис. 91. Грид-сетка карточек товаров

Гибкие сетки являются важным инструментом в современном веб-дизайне, позволяя создавать адаптивные и отзывчивые макеты, которые хорошо смотрятся на любых устройствах. Использование Flexbox и CSS Grid значительно упрощает процесс создания сложных интерфейсов, обеспечивая при этом высокое качество пользовательского опыта.

Ключевые моменты работы с грид в CSS

`Grid Layout` в CSS — это гибкая разметка, при которой элементы располагаются в виде таблицы. Таблица состоит из ячеек, которые можно собирать в целые области.

Чтобы создать грид, нужно в файле с CSS-разметкой добавить строчку `display: grid`.

Грид состоит из родительского контейнера и вложенных в него элементов. Элементы могут занимать несколько ячеек, которые складываются в столбцы, колонки или области.

Фракция — единица измерения в CSS, созданная для удобства — чтобы можно было задавать соотношения между элементами и не подгонять вручную проценты.

Грид может содержать в себе другой грид — такая конструкция называется вложенным `subgrid`.

Основными свойствами грид-контейнера являются: `grid-template`, `grid-template-columns`, `grid-template-rows`, `grid-template-areas`. Они позволяют задавать размеры колонок, строк или целых областей.

За отступы между колонками грида отвечает свойство `column-gap`, а между рядами — `row-gap`.

Горизонтально выровнять элементы контейнера можно с помощью `justify-items` и `justify-content`, а вертикально — с `align-items` и `align-content`.

Настроить расположение элементов от определённой линии грид-сетки можно с помощью свойств:

по горизонтальной оси: `grid-column-start`, `grid-column-end`, `grid-column`;

по вертикальной оси: `grid-row-start`, `grid-row-end`, `grid-row`; или общим свойством `grid-area`.

Выровнять конкретный элемент грида можно с помощью `justify-self` (по горизонтали) и `align-self` (по вертикали).

Части грид-разметки:

- `grid`-контейнер — самый главный элемент во всей разметке, в нём хранится всё содержимое сетки;
- `grid`-ячейка — единица грид-сетки, сюда можно положить один или несколько блоков кода;
- `grid`-линия — горизонтальная или вертикальная линия, разделяющая столбцы и колонки;
- `grid`-строка (`row`) — ряд ячеек;
- `grid`-столбец (`column`) — колонка ячеек;
- `grid`-элемент — какой-либо элемент сайта;
- `grid`-область (`area`) — пространство из ячеек, в CSS можно объединить несколько ячеек в одну и работать с ними как с единым целым.

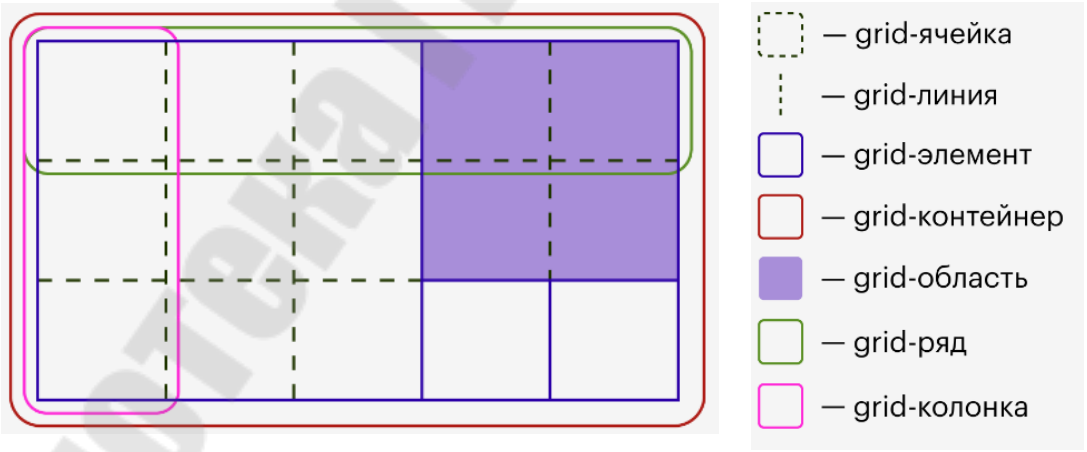


Рис. 92. Грид-разметка

Пример технологии Grid

Создадим файл HTML с простыми объектами – буквами A, B, C, D и E – и расположим их на отдельных строчках:

```
<head>
    <title>CSS Grid Layout</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <!-- Это будет грид-контейнер с пятью элементами внутри -->
    <div class="container">
        <div class="A">A</div>
        <div class="B">B</div>
        <div class="C">C</div>
        <div class="D">D</div>
        <div class="E">E</div>
    </div>
</body>
```

Рис. 93. Html-код веб-страницы с простыми объектами

В CSS-файле (`style.css`) контейнеру присвоим свойства `grid`-контейнера:

```
/* Объявляем grid-контейнер */
.container { /* Задаём свойства элементов HTML-класса «контейнер» */
    display: grid; /* Режим отображения меняем на «сетку» */
}
```

Рис. 94. Присвоение контейнеру свойства сетки

Стилизуем `grid`-контейнер – добавим в CSS дополнительные стили.

```
/* Добавляем оформление grid-контейнеру: */
.container {
    border: 2px solid rgb(80, 145, 27); /* Обводим зелёной линией */
    border-radius: 5px; /* Скругляем углы на 5 пикселей */
}

/* Добавляем оформление элементам grid-контейнера: */
.container * {
    border: 1px dashed; /* Обводим чёрным пунктиром */
    border-radius: 10px; /* Скругляем углы на 10 пикселей */
    background-color: rgba(82, 29, 179, 0.5); /* Фон красим фиолетовым */
    text-align: center; /* Контент размещаем в центре по горизонтали */
}

/*
    Для разделения действий мы несколько раз объявили один и тот же элемент
    .container. На самом деле все эти блоки – единое целое, и относятся к одному классу – container.
*/
```

Рис. 95. Стилизация grid-контейнера

Итог – вывод строк таблицы.

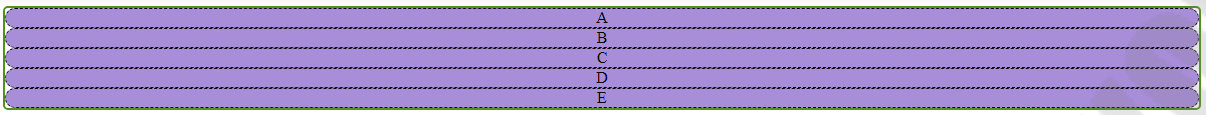


Рис. 96. Вывод таблицы

Чтобы изменить размеры строки, необходимо воспользоваться свойствами грид-контейнера.

```
1  .container {
2      display: grid;
3      grid-template-rows: 3fr 2fr 2fr 1fr;
4  }
```

Рис. 97. Изменение высоты строк

Т.к. размеры двух строчек повторяются — они занимают ровно `2fr`. Чтобы не писать дважды одно и то же, поместим их в функцию `repeat`. Она принимает на вход два параметра — размер строки во фракциях и количество его повторений:

```
1  grid-template-rows: 3fr repeat(2, 2fr) 1fr;
2
```

Рис. 98. Изменение высоты строк с использованием функции `repeat`

Итог вывода строк.

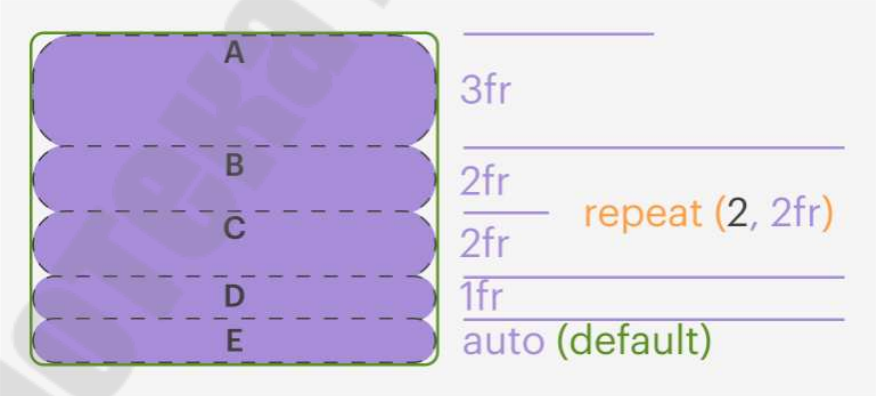


Рис. 99. Наглядная работа функции `repeat`

Свойства грид-контейнера задают параметры таблицы — количество строк и столбцов, их размеры, расположение в сетке и другие.

Колонки (`grid-template-columns`):
Чтобы создать колонки, нужно просто записать их размеры через пробел. Размеры можно указывать в разных единицах измерения: пикселях, процентах, сантиметрах и так далее. Причём можно смешивать несколько единиц в пределах одного свойства.
Например:

```
1  .container {
2      display: grid;
3      grid-template-columns: 30% 100px 70%;
4  }
5
```

Рис. 100. Установка размеров колонок

Для удобства отображения, гридам придумали новую единицу измерения — фракцию (`fr`). Она позволяет разделить свободное пространство экрана на несколько частей и не возиться с ручной подгонкой процентов.

```
1  .container {
2      display: grid;
3      grid-template-columns: 1fr 100px 2fr;
4  }
```

Рис. 101. Оптимизация кода

Итог:

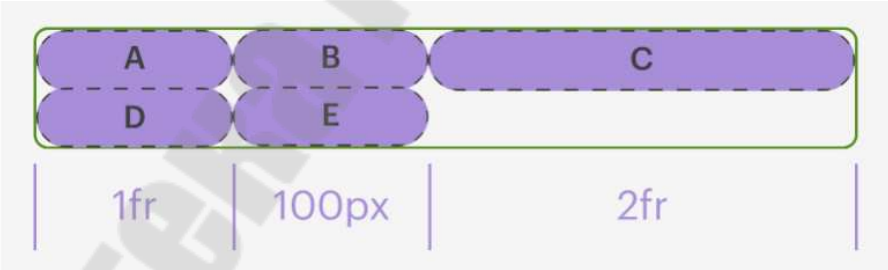


Рис. 102. Внешний вид ячеек в грид-сетке после оптимизации кода

Области (`grid-template-areas`)

Одна из особенностей грид-разметки — возможность создавать области и гибко регулировать их размеры. Пример:

Чтобы создать область, необходимо для каждого элемента в CSS-файле добавить свойство `grid-area` — его параметром будет любое имя. А затем добавить этот код в CSS-файл:

```
.A { grid-area: f; }
.B { grid-area: i; }
.C { grid-area: b; }
.D { grid-area: o; }
.E { grid-area: n; }
```

Рис. 103. Добавление свойств для элементов областей в файле CSS

Затем в грид-контейнере необходимо создать «матрицу» из этих имён:

```
.container {
    display: grid;
    grid-template-areas: "f f f f f i i i"
                         "f f f f f i i i"
                         "f f f f f i i i"
                         "f f f f f o b b"
                         "f f f f f n b b";
}
```

Рис. 104. Создание матрицы, т.е. шаблона

Тогда простые элементы выстроятся по шаблону.

Рис. 105. Внешний вид грид-сетки

Каждое имя в `grid-template-areas` соответствует определённому объекту. Одинаковые имена, стоящие рядом, собирают несколько ячеек в единое целое — то есть в грид-область:

- элемент A с «псевдонимом» f занимает область в 25 ячеек;
- элемент B (i) — в 9 ячеек;
- элемент C (b) — в 4 ячейки;
- элемент D (o) — в 1 ячейку;
- элемент E (n) — тоже в 1 ячейку.

Пример: Короткая форма для грид-областей.

HTML:

```html
<head>
    <title>CSS Grid Layout</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <!-- Это будет грид-контейнер с пятью элементами внутри -->
    <div class="container">
        <div class="A">A</div>
        <div class="B">B</div>
        <div class="C">C</div>
        <div class="D">D</div>
        <div class="E">E</div>
    </div>
</body>
```

Рис. 106. Html-код веб-страницы с простыми объектами

CSS:

```css
.container {
    display: grid;
    grid-template: "f f f i i"
                   "f f f i i"
                   "f f f b o"
                   ". . n n n";
}

.A { grid-area: f; }
.B { grid-area: i; }
.C { grid-area: b; }
.D { grid-area: o; }
.E { grid-area: n; }
```

Рис. 107. Параметры грид-сетки в CSS

Результат:

Рис. 108. Вывод результата грид-сетки

Пример: Грид-сетка установочного шаблона CMS Joomla «Cassiopeia».

```
.site-grid {
    grid-template-areas: ". banner banner banner banner ."
                         ". top-a top-a top-a top-a ."
                         ". top-b top-b top-b top-b ."
                         ". side-l comp comp side-r ."
                         ". bot-a bot-a bot-a bot-a ."
                         ". bot-b bot-b bot-b bot-b .";
}
```

Рис. 109. Грид-сетка шаблона CMS Joomla «Cassiopeia»

Позиции установочного шаблона CMS Joomla «Cassiopeia».

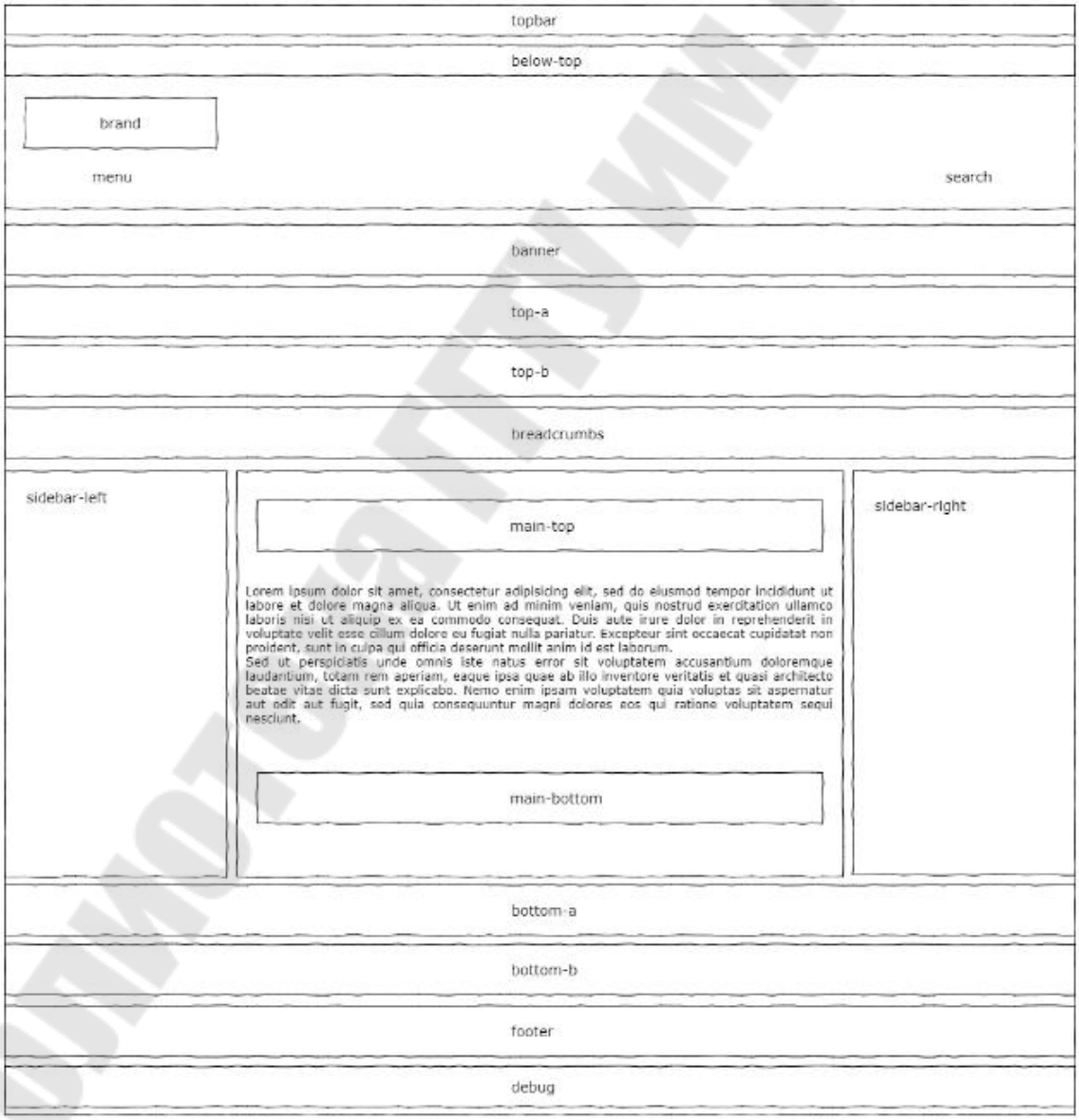


Рис. 110. Позиции шаблона CMS Joomla «Cassiopeia»

11.2. Технология FlexBox

Flexbox (Flexible Box Layout) — это технология, позволяющая легко управлять расположением и размером элементов в одномерном пространстве (по горизонтали или вертикали).

Flexbox лучше подходит для одноосевых (линейных) макетов, где нужно управлять расположением элементов в одной строке или колонне.

Во Flexbox есть два вида свойств: одни применяются к `flex`-контейнеру, другие — к элементам, которые в нём расположены.

Flex-контейнер — это «коробка», в которой хранятся `flex`-элементы (`flex item`). Чтобы превратить элемент во `flex`-контейнер, нужно установить ему свойство `display: flex` или `display: inline-flex`.

Разница между `flex` и `inline-flex` в том, что в первом случае контейнер будет занимать всю ширину экрана (как блочный элемент), а во втором — лишь пространство, занимаемое его содержимым.

`Flex`-элементы (`flex items`) — это дочерние элементы `flex`-контейнера. Мы можем управлять их расположением, размерами и расстоянием между ними.

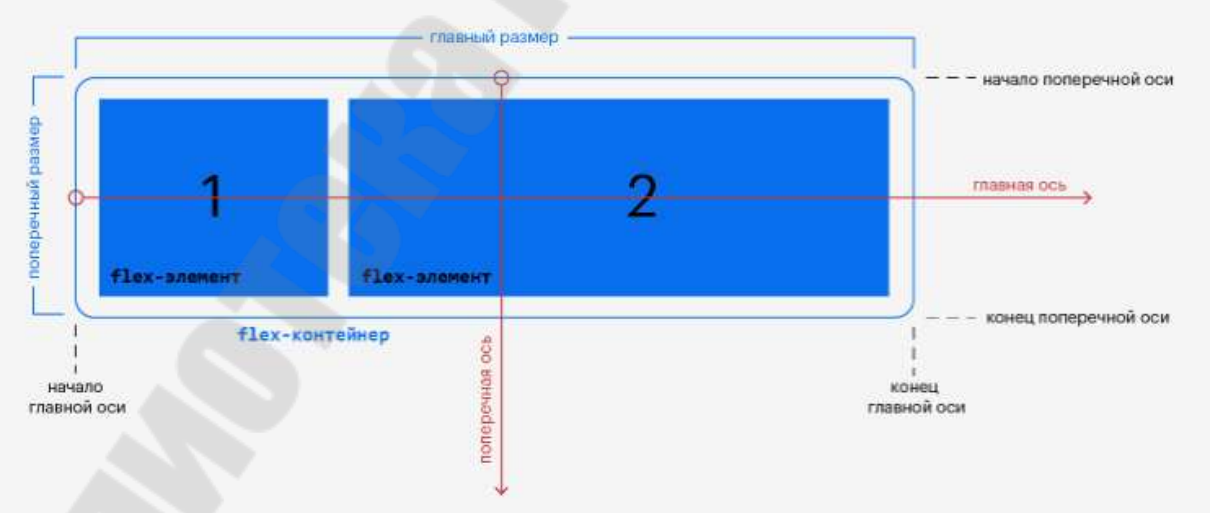


Рис. 111. Flex-контейнер

Главная ось (`main axis`) — направление, в котором располагаются flex-элементы.

Поперечная ось (cross axis) — ось, перпендикулярная главной оси.

Главный размер (main size) — размер, соответствующий направлению главной оси.

Начало главной оси (main start) — точка, в которой начинается последовательность flex-элементов, расположенных по главной оси.

Конец главной оси (main end) — точка, в которой заканчивается последовательность flex-элементов, расположенных по главной оси.

Поперечный размер (cross size) — размер, соответствующий поперечной оси.

1. Контейнер Flex:

Чтобы сделать элемент flex-контейнером, нужно установить свойство display: flex; или display: inline-flex;.

```
1  .flex-container {
2          display: flex;
3      }
```

Рис. 112. Свойство flex-контейнера в CSS

2. Основные свойства контейнера:

- flex-direction – определяет направление расположения элементов (по умолчанию row);
- row – элементы располагаются в строку (по умолчанию):
  - column: элементы располагаются в столбец;
  - row-reverse: элементы располагаются в строку в обратном порядке;
  - column-reverse: элементы располагаются в столбец в обратном порядке.
- justify-content – управляет выравниванием элементов по основной оси:
  - flex-start: выравнивание по началу;
  - flex-end: выравнивание по концу;
  - center: выравнивание по центру;
  - space-between: равномерное распределение с отступами между элементами;

- space-around: равномерное распределение с отступами вокруг элементов.

• align-items – управляет выравниванием элементов по поперечной оси:

- flex-start: выравнивание по началу;
- flex-end: выравнивание по концу;
- center: выравнивание по центру;
- baseline: выравнивание по базовой линии текста;
- stretch: растягивание элементов, чтобы заполнить контейнер.

3. Свойства элементов:

• flex-grow – определяет, как элемент будет расти, чтобы заполнить доступное пространство;

• flex-shrink – определяет, как элемент будет сжиматься, если не хватает места;

• flex-basis – задает начальный размер элемента до распределения пространства.

Пример использования Flexbox:

```html
<div class="container">
        <div class="inner-div">
            <p>1</p>
        </div>
        <div class="inner-div">
            <p>2</p>
        </div>
        <div class="inner-div">
            <p>3</p>
        </div>
    </div>
```

```css
p {
    padding-top: 40px;
    margin: 0;
    font-size: 100px;
    font-family: sans-serif;
    text-align: center;
    vertical-align: middle;
  }


.inner-div {
    height: 200px;
    width: 400px;
    margin: 10px;
    background-color: #096eeb;
  }
```

Рис. 113. Простой HTML и стилизация элементов с помощью CSS (справа)

Так как блочные элементы занимают всю ширину, то блоки `.inner-div` располагаются один под другим.

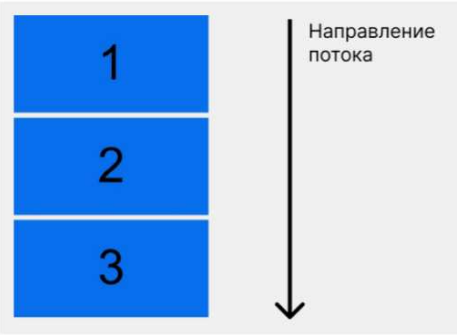


Рис. 114. Расположение элементов и направление основного потока

При присвоении родительскому элементу свойства `display: flex` – элементы выстраиваются горизонтально

```
<div class="container">
        <div class="inner-div">
            <p>1</p>
        </div>
        <div class="inner-div">
            <p>2</p>
        </div>
        <div class="inner-div">
            <p>3</p>
        </div>
    </div>
```

```
.container {
    display: flex;
}
p {
    padding-top: 40px;
    margin: 0;
    font-size: 100px;
    font-family: sans-serif;
    text-align: center;
    vertical-align: middle;
  }


.inner-div {
    height: 200px;
    width: 400px;
```

Рис. 115. Html и CSS-код контейнера- Flexbox

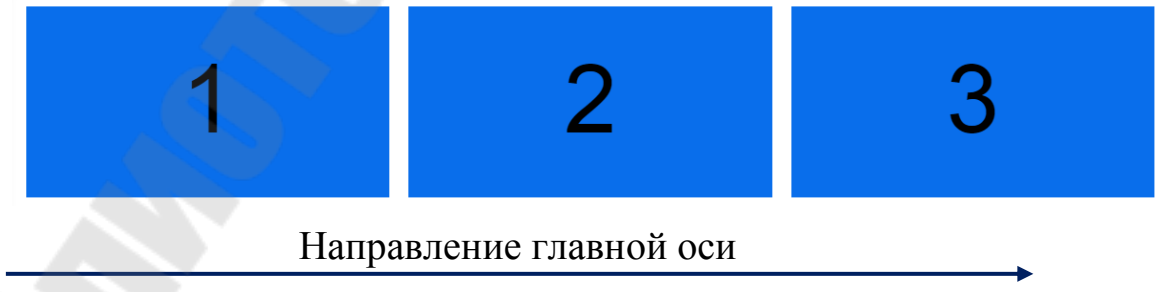


Рис. 116. Выравнивание элементов по технологи Flexbox

11. АДАПТИВНЫЙ И ОТЗЫВЧИВЫЙ ДИЗАЙН

Адаптивный и отзывчивый дизайн – это два подхода к созданию веб-сайтов, которые позволяют им корректно отображаться на различных устройствах и экранах.

11.1. Адаптивная верстка

Адаптивная верстка (или адаптивный дизайн - Adaptive Design) — это подход к веб-разработке, который позволяет веб-сайту автоматически изменять свой внешний вид и структуру в зависимости от размера экрана устройства, на котором он отображается. Это достигается с помощью медиа-запросов и гибких макетов.

Основные принципы адаптивной верстки:

1. **Фиксированные макеты**

Адаптивная верстка подразумевает создание нескольких фиксированных макетов для различных разрешений экрана (например, для мобильных телефонов, планшетов и настольных компьютеров). Каждый макет имеет свои стили и структуру, которые применяются в зависимости от ширины экрана.

Ранее шаблоны сайтов не обладали адаптивностью. Отображение сайтов было одинаково и на компьютере, и на телефоне.

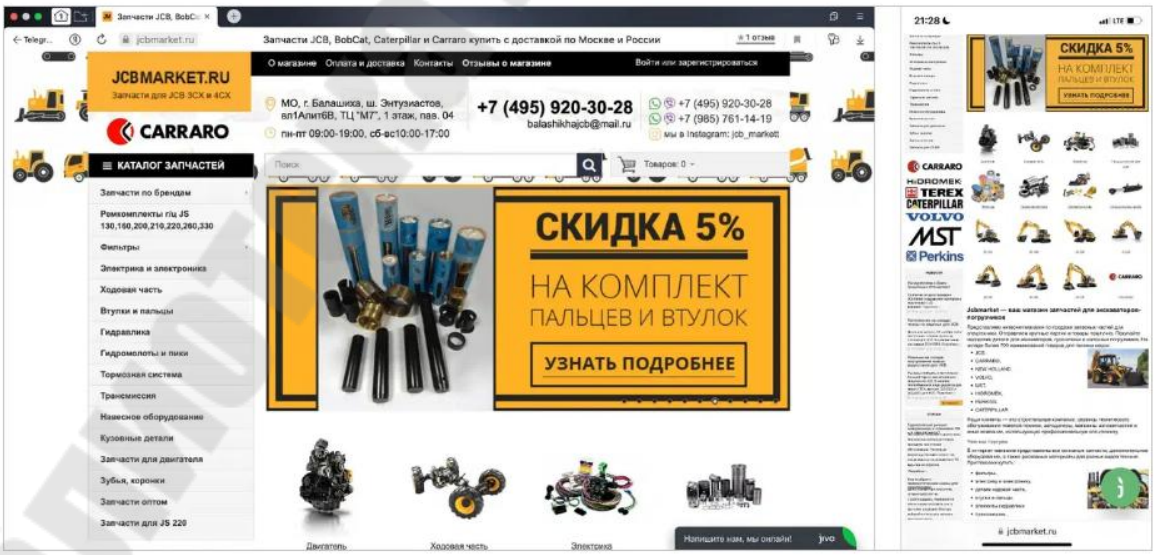


Рис. 117. Неадаптивный шаблон сайта

Первые адаптивные шаблоны создавались с использованием технологии «резиновой верстки». В мобильном браузере сайт отображался в исходном состоянии, но масштабировался под размеры дисплея. Отдельные мелкие детали становились видимыми при увеличении масштаба экрана.

Рис. 118. Поведение элементов сайта при резиновой верстке

При адаптивной верстке шаблон сайта в мобильном браузере перестраивается определенным образом, так, чтобы им было удобнее пользоваться именно с этого устройства.

Рис. 119. Поведение элементов сайта при адаптивной верстке

В адаптивной вёрстке выделяют следующие принципы:

1. Контент десктопной и мобильной версии сайта дублируется. Всё, что пользователь может увидеть с компьютера, должно быть доступно и с телефона.

2. Не меняется дизайн. Шрифты, цвета, логотипы должны быть одинаковыми в десктопной и мобильной версиях.

3. Сохраняется иерархия элементов. Иерархия заголовков, подзаголовков, текстовых блоков, иллюстраций и кнопок выстраивается на основе цели сайта.

4. Высота и ширина кликабельных элементов — не менее 44 пикселей. По десктопной версии сайта пользователь перемещается с помощью мыши. По мобильной версии можно перемещаться только с помощью пальца. Область нажатия — 44–48 пикселей. Если кнопка или знак подсказки меньше этих значений, пользователю будет сложно попасть по ним.

5. Функционал десктопного сайта повторяется. Если с компьютера доступны фильтры и разные режимы отображения товаров — например, списком и карточками, — в мобильной версии этот функционал должен быть повторён.

6. Число колонок уменьшается. При вёрстке десктопных сайтов обычно используют сетку из 12 колонок. Для узкого мобильного экрана столько колонок не нужно, обычно хватает одной, максимум четырёх. Для экрана планшета подойдёт сетка из шести или восьми колонок.

7. Компактность — это вертикальное расположение в мобильной версии карточек и других элементов, которые на десктопной идут в ряд. Это обосновано тем, что на компьютере просматривают контент слева направо, а на телефоне — сверху вниз.

Основных разрешений для адаптивной вёрстки три:

- 1600 пикселей — для компьютеров;
- 960 пикселей — для планшетов;
- 375 пикселей — для телефонов.

Это усреднённые значения. Ширина может меняться в зависимости от сайта, идеи и технических особенностей. Ещё эти значения называют брейкпоинтами (от англ. `breakpoint`) — то есть разрешениями, при которых сайт меняет отображение на экране. По брейкпоинтам алгоритм на сайте определяет, какой шаблон выбрать для отображения.

В популярном фреймворке разработчиков Bootstrap сейчас представлено шесть брейкпоинтов: 1920, 1200, 992, 768, 576 и 376 пикселей.

Пример адаптивного дизайна с учётом шести брейкпоинтов. Элементы одни и те же, но их расположение подстраивается под ширину экрана.

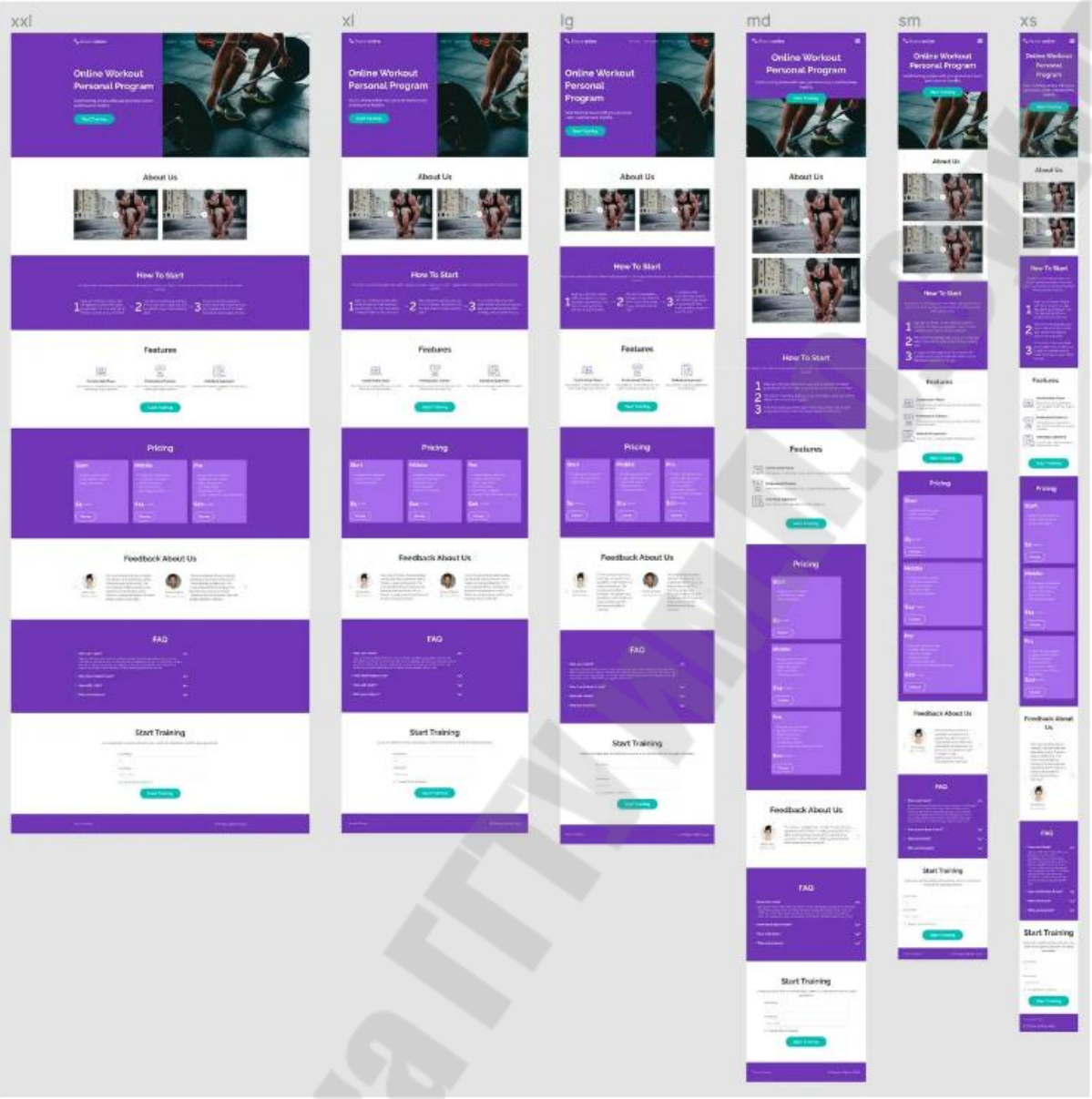


Рис. 120. Адаптивный дизайн сайта с учётом шести брейкпоинтов

2. **CSS медиазапросы**

CSS медиазапросы используются для:

- адаптивного веб-дизайна – CSS позволяют изменять стиль страницы в зависимости от размера экрана устройства (например, смартфоны, планшеты, десктопы);
- оптимизации представления контента – позволяют скрывать, изменять размеры или перераспределять элементы на странице для улучшения пользовательского опыта на разных устройствах;
- изменения стилей в зависимости от характеристик устройства: Например, для изменения шрифтов, отступов, изображений и других

элементов в зависимости от разрешения экрана, ориентации (портретная или альбомная) или плотности пикселей;

- различной навигации и раскладки – позволяют адаптировать меню и другие навигационные элементы для более удобного взаимодействия на мобильных устройствах.

Использование медиазапросов делает веб-сайты более универсальными и удобными для пользователей на различных платформах.

Пример медиа-запросов:

```
/* Стиль для мобильных устройств */
@media (max-width: 600px) {
    body {
        background-color: lightblue;
    }
}

/* Стиль для планшетов */
@media (min-width: 601px) and (max-width: 1200px) {
    body {
        background-color: lightgreen;
    }
}

/* Стиль для настольных ПК */
@media (min-width: 1201px) {
    body {
        background-color: lightcoral;
    }
}
```

Рис. 121. Медиа-запросы для мобильных устройств

3. **Гибкие изображения**

Изображения и мультимедиа должны адаптироваться к размерам экрана так же, как и текстовые блоки. Для этого используются относительные значения ширины, такие как `max-width: 100%`. Это гарантирует, что изображение не выйдет за пределы родительского контейнера, независимо от размеров экрана. Таким образом, контент остается аккуратным и легко читаемым.

Гибкие изображения — это изображения, которые адаптируются к размеру контейнера, в котором они находятся, и позволяют создавать отзывчивые веб-дизайны. Это особенно важно для обеспечения хорошего пользовательского опыта на различных устройствах с разными размерами экранов.

Как сделать изображения гибкими

1. Использование CSS:

Чтобы сделать изображения гибкими, обычно применяют CSS-свойство `max-width` вместе с `height`:

```
img {
    max-width: 100%;
    height: auto;
}
```

Рис. 122. Параметры в CSS для гибкого изображения

- `max-width: 100%;`: Это свойство позволяет изображению занимать не более 100% ширины родительского контейнера. Таким образом, изображение будет уменьшаться в размере, если контейнер меньше его естественной ширины.

- `height: auto;`: Это свойство сохраняет пропорции изображения при изменении его ширины. Без него изображение может искажаться.

2. Использование `object-fit`:

Если изображение помещается в контейнер с определенными размерами, то используют свойство `object-fit`, чтобы контролировать его поведение:

```
.image-container {
    width: 100%;
    height: 300px; /* фиксированная высота */
    overflow: hidden; /* скрыть часть изображения, выходящую за пределы контейнера */
}

img {
    width: 100%;
    height: 100%;
    object-fit: cover; /* или contain, в зависимости от желаемого эффекта */
}
```

Рис. 123. Использование свойства `object-fit`, если изображение в контейнере

- `object-fit: cover;`: Изображение будет масштабироваться, чтобы заполнить контейнер, сохраняя при этом свои пропорции. Часть изображения может быть обрезана.

- `object-fit: contain;`: Изображение будет масштабироваться так, чтобы полностью поместиться в контейнер,

сохраняя пропорции. В этом случае могут остаться пустые пространства.

3. Использование адаптивных изображений:

Для достижения еще большего контроля над изображениями на разных устройствах можно использовать адаптивные изображения с помощью HTML5:

```
<picture>
    <source media="(max-width: 600px)" srcset="small-image.jpg">
    <source media="(max-width: 1200px)" srcset="medium-image
        .jpg">
    <img src="large-image.jpg" alt="Описание изображения">
</picture>
```

Рис. 124. Использование адаптивных изображений с помощью HTML5

В этом примере браузер выбирает наиболее подходящее изображение в зависимости от ширины экрана.

Преимущества гибких изображений:

- Улучшение производительности: Гибкие изображения позволяют загружать меньшие файлы на мобильных устройствах, что сокращает время загрузки и потребление данных.
- Оптимизация пользовательского опыта: Гибкие изображения обеспечивают корректное отображение на различных устройствах, что улучшает взаимодействие пользователей с сайтом.
- Сохранение пропорций: Гибкие изображения сохраняют свои пропорции, что предотвращает искажение и ухудшение качества.

Гибкие изображения являются важным аспектом адаптивного веб-дизайна и помогают создавать интерфейсы, которые правильно отображаются на любых устройствах.

4. **Отступы и размеры**:

Для задания размеров и отступов, чтобы элементы могли адаптироваться к размеру экрана используют относительные единицы измерения (например, `em`, `rem`, `%`).

Пример простой адаптивной верстки с использованием медиа-запросов. Отображение элементов сайта будет меняться в зависимости от устройства:

```html
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device
        -width, initial-scale=1.0">
    <title>Пример адаптивной вёрстки</title>
     <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Адаптивная вёрстка</h1>
    </header>
    <div class="container">
        <div class="content">
            <div class="box">Блок 1</div>
            <div class="box">Блок 2</div>
            <div class="box">Блок 3</div>
        </div>
    </div>
</body>
</html>

```

```css
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}
.container {
    width: 90%;
    max-width: 1200px;
    margin: auto;
    padding: 20px;
}
header {
    background: lightgrey;
    padding: 10px;
    text-align: center;
}
.content {
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    gap: 20px;
}
.box {
    flex: 1 1 300px; /* Гибкая ширина с минимальным
        значением 300px */
    background: lightblue;
    padding: 20px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
}

/* Медиа-запрос для мобильных устройств */
@media (max-width: 600px) {
    .content {
        flex-direction: column; /* Столбцы на мобильных
            устройствах */
    }
}
```

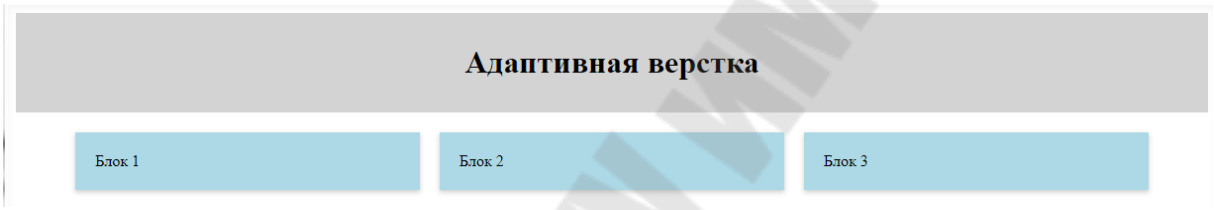


Рис. 125. Адаптивная верстка с помощью медиа-запросов

Разработка адаптивного сайта — более трудоемкая и более дорогостоящая услуга.

Проверка на разных устройствах

Важно тестировать адаптивную верстку на различных устройствах и браузерах, чтобы убедиться, что сайт отображается корректно на всех экранах. Это позволяет выявить возможные проблемы и убедиться, что все элементы правильно отображаются на каждом экране. Инструменты, такие как Google Chrome DevTools, помогают разработчикам тестировать адаптивность, имитируя различные размеры экранов и устройства.

11.2 Отзывчивый дизайн (Responsive Design)

Отзывчивый дизайн предполагает создание единого макета, который автоматически адаптируется к различным размерам экранов.

Вместо создания нескольких фиксированных версий, отзывчивый дизайн использует гибкие сетки и медиа-запросы для изменения стилей в зависимости от размера экрана. Пример отзывчивых дизайнов – установочные шаблоны для современных CMS.

Основные характеристики отзывчивого дизайна:

1. Гибкие сетки – используются относительные единицы измерения (например, проценты) для задания размеров элементов, что позволяет им изменять размеры в зависимости от размера экрана.

2. Медиа-запросы – как и в адаптивном дизайне, медиа-запросы позволяют изменять стили на основе характеристик устройства, но вместо фиксированных макетов используются относительные размеры и пропорции.

3. Гибкие изображения – изображения и другие медиа-элементы обычно имеют свойство `max-width: 100%;`, что позволяет им масштабироваться в зависимости от размера контейнера.

4. Универсальность – отзывчивый дизайн обеспечивает более универсальный подход, так как он работает на всех устройствах без необходимости создания отдельных макетов.

11.3 Сравнение адаптивного и отзывчивого дизайна

Таблица 1: Сравнение адаптивного и отзывчивого дизайна

Характеристика	Адаптивный дизайн	Отзывчивый дизайн
Макеты	Несколько фиксированных макетов	Один гибкий макет
Подход	Определенные точки перелома	Гибкое изменение в зависимости от размера экрана
Использование медиа-запросов	Да, для каждого размера устройства	Да, для динамического изменения стилей
Гибкость	Ограниченная	Высокая
Оптимизация	Оптимизированы для конкретных устройств	Подходит для всех устройств

Оба подхода имеют свои преимущества и недостатки. Выбор между адаптивным и отзывчивым дизайном зависит от требований проекта, целевой аудитории и специфики контента. В современных веб-приложениях часто используются элементы обоих подходов для достижения максимальной гибкости и удобства использования.

11.4. Сравнение различных подходов к верстке

Краткий обзор различных подходов к верстке веб-страниц с их преимуществами и недостатками:

1. Традиционная верстка

Традиционная верстка основывается на использовании фиксированных размеров и стилей, созданных с помощью HTML и CSS. Часто используется фиксированная ширина для контейнеров и элементов.

Преимущества:

- более простой подход для быстрого создания макетов;
- легко управлять, когда контент имеет предсказуемые размеры.

Недостатки:

- плохо адаптируется к различным устройствам (мобильным, планшетам);
- может потребовать дополнительных медиа-запросов для разных размеров экранов.

2. Адаптивная верстка

Адаптивная верстка использует фиксированные размеры и точки останова (`breakpoints`) для создания различных версий страницы для разных устройств. Каждый из них имеет свое собственное планирование верстки.

Преимущества:

- оптимизированные макеты для конкретных устройств;
- позволяет создавать более «идеальные» версии страниц под каждую платформу.

Недостатки:

- требует больше времени и усилий на проектирование и разработку;
- может вызвать трудности при добавлении новых устройств в будущем.

3. Отзывчивый дизайн (Responsive Design)

Отзывчивый дизайн предполагает использование гибких сеток, медиазапросов и относительных единиц измерения (например, `%`, `em`, `vw`) для создания макета, который автоматически подстраивается под размеры экрана.

Преимущества:

- автоматическая адаптация к любому устройству и разрешению экрана;

- более эффективное управление и упрощает поддержание кода.

Недостатки:

- иногда сложнее в реализации, особенно для старых браузеров.;

- может потребовать тестирования в большом количестве разрешений.

4. Фреймворки CSS (например, Bootstrap)

Использование CSS-фреймворков, таких как Bootstrap, которые предлагают готовые стили и компоненты для облегчения процесса верстки.

Преимущества:

- ускоряет процесс разработки благодаря предопределённым стилям и компонентам;

- обеспечивает кросс-браузерную совместимость и адаптивность по умолчанию.

Недостатки:

- возможно наличие лишнего кода, который не используется (но можно настроить);

- ограниченная индивидуальность, если не настроить стиль под проект.

5. Технологии CSS Grid и Flexbox

Современные технологии CSS Grid и Flexbox, которые позволяют создавать сложные и адаптивные макеты с использованием сеток и гибких контейнеров.

Преимущества:

- отлично подходит для создания отзывчивых макетов без необходимости в дополнительных медиазапросах;

- высокая степень контроля над расположением элементов.

Недостатки:

- может быть сложным для понимания новичками;

- потребует внимательного использования для обеспечения кросс-браузерной совместимости.

Каждый из этих подходов имеет свои особенности, преимущества и недостатки. Выбор подхода зависит от требований проекта, целевой аудитории и времени на разработку. Отзывчивый дизайн, как правило, является наиболее предпочтительным вариантом для современных веб-приложений из-за своей гибкости и универсальности.

11.5. Контрольные точки

Контрольные точки (или брейкпойнты) в контексте адаптивного и отзывчивого дизайна – это определенные размеры экрана, на которых разработчик изменяет стили или структуру страницы для обеспечения оптимального отображения контента. Они позволяют адаптировать макет и элементы интерфейса в зависимости от размеров устройства пользователя.

Контрольные точки являются строительными блоками адаптивного дизайна.

Брейкпойнты обычно устанавливаются в пикселях и соответствуют распространенным разрешениям экранов (например, 320px, 768px, 1024px).

Bootstrap включает шесть контрольных точек по умолчанию, иногда называемых уровнями сетки, для быстрого реагирования.

	Контрольные точки	Инфикс класса	Размеры
	X-Small	*Нет*	<576px
	Small	`sm`	≥576px
	Medium	`md`	≥768px
	Large	`lg`	≥992px
	Extra large	`xl`	≥1200px
	Extra extra large	`xxl`	≥1400px

Рис. 126. Доступные контрольные точки CSS Bootstrap

В адаптивном дизайне часто используются фиксированные брейкпойнты, при которых для каждой группы устройств создаются отдельные макеты. Например, сайт может иметь отдельные версии для мобильных, планшетов и десктопов.

В отзывчивом дизайне контрольные точки более гибкие. CSS-стили применяются на основе диапазонов размером экрана, что позволяет элементам масштабироваться и адаптироваться к любому разрешению.

12. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ CSS

12.1. Стили пользовательского интерфейса

CSS (Cascading Style Sheets) предоставляет множество возможностей для стилизации пользовательского интерфейса, что позволяет создавать привлекательные и функциональные веб-приложения. Вот некоторые из дополнительных возможностей CSS, а также стили, которые могут улучшить пользовательский интерфейс.

1. Псевдоклассы и псевдоэлементы

- Псевдоклассы: Позволяют применять стили к элементам в зависимости от их состояния:

- `:hover`: применяется, когда пользователь наводит курсор на элемент;

- `:focus`: применяется, когда элемент находится в фокусе;

- `:active`: применяется, когда элемент активен (например, при нажатии кнопки).

```
button:hover {
    background-color: lightblue;
}

input:focus {
    border: 2px solid blue;
}
```

Рис. 127. Псевдоклассы в CSS

- Псевдоэлементы: Позволяют стилизовать определенные части элемента.

- `::before`: добавляет содержимое перед содержимым элемента.

- `::after`: добавляет содержимое после содержимого элемента.

```
h1::before {
    content: "★ ";
    color: gold;
}
```

Рис. 128. Псевдоэлемены в CSS

2. Анимации и переходы:

- Переходы: Позволяют плавно изменять свойства элементов при взаимодействии.

```
button {
    transition: background-color 0.3s ease;
}

button:hover {
    background-color: lightgreen;
}
```

Рис. 129. Изменение свойств элементов при взаимодействии

- Анимации: Позволяют создавать более сложные эффекты, задавая ключевые кадры.

```
@keyframes fadeIn {
    from { opacity: 0; }
    to { opacity: 1; }
}

.fade-in {
    animation: fadeIn 1s ease-in;
}
```

Рис. 130. Анимация в CSS

4. Стилизация форм:

- Стилизация полей ввода и кнопок: Использование CSS для улучшения внешнего вида форм.

```
input[type="text"], input[type="email"] {
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

button {
    background-color: blue;
    color: white;
    padding: 10px 15px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
```

Рис. 131. Стилизация кнопок в CSS

5. Темы и переменные CSS

- Переменные CSS: Позволяют определять значения, которые можно использовать повторно в различных частях стилей.

```css
:root {
    --main-color: blue;
    --secondary-color: lightblue;
}

body {
    background-color: var(--main-color);
}

button {
    background-color: var(--secondary-color);
}
```

Рис. 132. Переменные CSS

6. Мультимедийные запросы

- Адаптивный дизайн: Используйте медиазапросы для изменения стилей в зависимости от размера экрана.

```css
@media (max-width: 600px) {
    body {
        font-size: 14px;
    }
}
```

Рис. 133. Использование медиазапросов для адаптивности

7. Стилизация навигации:

- Стилизация меню: Используйте Flexbox или Grid для создания адаптивных навигационных меню.

```css
nav {
    display: flex;
    justify-content: space-around;
    background-color: #333;
}

nav a {
    color: white;
    padding: 14px 20px;
    text-decoration: none;
}

nav a:hover {
    background-color: lightblue;
}
```

Рис. 134. Стилизация навигации

CSS предлагает множество возможностей для стилизации пользовательского интерфейса, позволяя создавать как простые, так и сложные макеты. Используя псевдоклассы, Flexbox, Grid, анимации,

переменные и медиазапросы, можно значительно улучшить пользовательский опыт и сделать ваш сайт более привлекательным и функциональным.

CSS является неотъемлемой частью веб-разработки и дизайна. Его применимость охватывает все аспекты стилизации и форматирования веб-страниц, что позволяет создавать удобные, привлекательные и функциональные интерфейсы. Правильное использование CSS помогает улучшить пользовательский опыт и делает веб-приложения более доступными и эстетически привлекательными.

12.2 Ошибки и решения при работе с CSS

Список распространённых ошибок при работе с CSS и возможные решения для их устранения:

1. Неиспользование нормализации стилей (CSS Reset):

Ошибка – разные браузеры по умолчанию применяют разные стили к элементам, что может привести к непредсказуемому отображению.

Решение – для установки единых стандартов к элементам используют CSS `Reset` (например, Normalize.css).

2. Ошибки в написании селекторов:

Ошибка – неправильный синтаксис селекторов (например, лишние пробелы или забытые запятые) могут привести к тому, что стили не применяются.

Решение – необходимо проверить синтаксис селекторов и убедитесь, что они корректны и соответствуют документу.

3. Конфликт стилей:

Ошибка – применение нескольких стилей к одному элементу с одинаковой специфичностью может приводить к конфликтам.

Решение – чтобы убедиться, что нужный стиль будет применен используют более конкретные селекторы или приоритеты.

4. Применение инлайн-стилей:

Ошибка – чрезмерное использование инлайн-стилей затрудняет управление стилями и снижает читаемость кода.

Решение – использование внешних или внутренних таблиц стилей. Инлайн-стили следует применять только при необходимости.

5. Отсутствие медиа-запросов

Ошибка – пропуск медиа-запросов может привести к проблемам с адаптивностью и отображением на мобильных устройствах.

Решение – для стилизации элементов под разные размеры экранов добавляют медиа-запросы.

6. Неоптимизированные изображения

Ошибка – использование изображений больших размеров может негативно сказаться на времени загрузки веб-страницы.

Решение – оптимизирование изображений (минификация, выбор формата) и использование атрибутов `width` и `height`, чтобы указать размеры.

7. Неиспользование комментариев:

Ошибка – отсутствие комментариев делает код сложнее для понимания и поддержки.

Решение – Для облегчения понимания кода для других разработчиков, в коде CSS оставляют комментарии (рис. 23, рис. 155).

8. Игнорирование специфики

Ошибка – неосознанное использование селекторов с одинаковой специфичностью может приводить к ошибкам в применении стилей.

Решение – изучение как работает специфичность в CSS.

9. Поддержка кроссбраузерности:

Ошибка – пропуск тестирования и стилизации для разных браузеров и устройств.

Решение – проведение тестирования в различных браузерах и использование специальных инструментов.

10. Отсутствие структурирования кода

Ошибка – неорганизованные стили приводят к проблемам с поддержкой.

Решение – следование лучшим практикам структурирования кода, разбивание стилей на логические секции (например, оформление заголовков, параграфов, кнопок и т.д.).

Избегая этих распространённых ошибок и применяя предложенные решения, можно улучшить качество CSS-кода и обеспечить лучшее отображение веб-страниц.

12.3. Оптимизация CSS-кода

Оптимизация CSS-кода — это процесс улучшения производительности и читаемости стилей, а также снижения времени загрузки страницы.

Ключевые стратегии и практики для оптимизации CSS-кода:

1. Удаление неиспользуемого CSS

Необходимо проанализировать и удалить неиспользуемые стили на странице.

Необходимо использовать инструменты, такие как PurifyCSS или UnCSS, чтобы автоматически выявить неиспользуемый CSS.

2. Минификация CSS

Сжать CSS-код, убрав все пробелы, комментарии и ненужные символы.

Для минификации можно использовать инструменты, такие как CSSNano, CleanCSS или встроенные функции сборщиков модулей, например, Webpack.

3. Консолидирование файлов CSS

Необходимо свести к минимуму количество CSS-файлов. Чем меньше HTTP-запросов, тем быстрее загружается страница. Рекомендуется объединить несколько файлов стилей в один.

4. Использование отмечания (Naming Convention)

Необходимо использовать систему именования, например, BEM (Block Element Modifier), чтобы улучшить читаемость и поддержку кода.

5. Применение Flexbox и Grid

Необходимо использовать современные CSS-модели, такие как Flexbox и CSS Grid, для более упрощенного и гибкого макета, что может убрать необходимость в дополнительных классах и правилах.

6. Оптимизация селекторов

Следует избегать слишком специфических или вложенных селекторов. Это не только улучшает производительность, но и упрощает поддержку кода:

```
/* Сложный селектор */
div#header > ul > li a {
    color: red;
}

/* Оптимизированный селектор */
#header a {
    color: red;
}
```

Рис. 135. Сравнение сложного и оптимизированного селектора

7. Использование переменных CSS

Рекомендуется использовать CSS-переменные для хранения и повторного использования значений, что повышает гибкость и уменьшает повторяемость:

```
:root {
    --main-color: #4CAF50;
}

.header {
    background-color: var(--main-color);
}
```

Рис. 136. Пример переменных в CSS

8. Мультимедийные запросы

Необходимо убедиться, что медиа-запросы упорядочены корректно и не дублируются. Это приведет к уменьшению объема кода и улучшит производительность.

9. Настройка кэширования

Стоит настроить кэширование для ваших CSS-файлов на сервере, чтобы браузеры могли хранить их и не загружать снова при повторных посещениях.

10. Использование инструмента анализа

Рекомендуется применять инструменты для анализа производительности, такие как Google PageSpeed Insights или Lighthouse, чтобы выявлять узкие места и области для улучшения.

Пример оптимизации:

Исходный код CSS и оптимизированный:

```css
body {
    margin: 0;
}

header {
    background-color: #4CAF50;
    color: white;
    padding: 20px;
}

.header-title {
    font-size: 24px;
}
```

```css
:root {
    --main-color: #4CAF50;
    --header-padding: 20px;
}

* {
    margin: 0;
}

header {
    background-color: var(--main-color);
    color: white;
    padding: var(--header-padding);
}

h1 {
    font-size: 24px; /* Изменение селектора для общности */
}
```

Рис. 137. Оптимизация CSS-кода

Оптимизация CSS-кода применяется в различных случаях и сценариях, когда есть необходимость улучшить производительность веб-сайта или приложения. Вот основные ситуации, в которых оптимизация CSS становится особенно актуальной:

1. Улучшение времени загрузки страницы

- Большие CSS-файлы: Если CSS-код становится слишком объемным, это может значительно увеличить время загрузки страницы. Оптимизация позволяет уменьшить размер файла.
- Много HTTP-запросов: Когда есть множество CSS-файлов, оптимизация с их объединением может снизить количество запросов к серверу.

2. Безопасность производительности на мобильных устройствах

- Медленные сети: На мобильных устройствах с низким качеством соединения пользователи могут сталкиваться с долгими периодами загрузки. Оптимизация CSS помогает улучшить пользовательский опыт в таких условиях.

3. Поддержка больших проектов

- Сложные стили: В комплексных проектах с большим объемом CSS может стать сложно отследить и поддерживать код. Оптимизация (например, использование систем именования и переменных) делает код более понятным и управляемым.

4. Повышение производительности взаимодействий

- Анимации и переходы: Оптимизация стилей может сделать анимации более плавными, снижая время рендеринга и сглаживая поведение при взаимодействии.

5. Улучшение SEO

- Улучшение пользовательского опыта: Быстрые страницы, которые загружаются и работают хорошо, могут повлиять на рейтинг в поисковых системах, поскольку быстрота загрузки является фактором ранжирования.

6. При работе с системами управления контентом (CMS)

- Темы и шаблоны: При разработке тем для CMS, таких как WordPress, оптимизация CSS важна, чтобы кейс был легковесным и удобным для использования.

7. Поддержка разных устройств

- Адаптивный и отзывчивый дизайн: На сайте с адаптивным дизайном необходимо оптимизировать медиазапросы и уменьшить количество задействованных стилей для разных разрешений экрана.

8. Частые изменения и обновления

- Итеративная разработка: Если разрабатывается проект с частыми изменениями, оптимизированный код будет проще обновлять и менять, что экономит время разработчиков.

9. Использование фреймворков и библиотек

- Интеграция фреймворков: При использовании сторонних библиотек и фреймворков (например, Bootstrap) иногда требуется оптимизация для сокращения избыточного кода и стилей.

10. Когда возникают проблемы с производительностью

- Замедление работы сайта: Если веб-сайт начинает работать медленно, а пользователи жалуются на долгую загрузку, стоит проанализировать CSS и провести его оптимизацию.

Оптимизация CSS — это не одноразовая задача, а постоянный процесс, который необходимо учитывать на всех этапах разработки веб-приложения или сайта для достижения наилучших результатов.

13. JAVASCRIPT В ВЕБ-ДИЗАЙНЕ

3.1. Структура DOM

Современная разработка сайтов – процесс сложный и многоуровневый. Программисты работают не только с HTML, CSS или JavaScript в рамках нескольких отдельных файлов. Перед окончательной публикации сайта в сеть, все файлы объединяются воедино.

Таким образом, открываемая в браузере конкретным пользователем HTML-страница является результатом:

- соединения разных документов (HTML, CSS и скриптов) в единую структуру;
- генерации тегов языками программирования и их фреймворками (например: Python и Django, JavaScript и Angular);
- обращения к данным из некоторой базы (реляционной или NoSQL);
- воздействия Javascript и AJAX-технологии (позволяет динамически подгружать элементы сайта);
- работы браузера по построению дерева документа.

Последний пункт связан с так называемой моделью DOM.

DOM (Document Object Model) — это объектная модель документа, которая представляет структуру HTML-документа в виде дерева объектов. Каждый элемент, атрибут и текстовый узел в HTML-документе становятся объектами в DOM, что позволяет программам (например, на JavaScript) взаимодействовать с документом.

Другими словами, непосредственно сам документ в виде HTML браузер не показывает в исходном виде. Сначала идет запрос к серверу, а полученный ответ разбирается для построения дерева страницы или DOM-дерева. Схема такова:

1. Сервер создает HTML-код страницы или отдает его (если это простой статический сайт);
2. Браузер получает код и разбирает на элементы дерева;
3. При необходимости подключается JavaScript, если он используется, чтобы изменить поведение тегов и их содержимого в зависимости от воздействий пользователя;

4. DOM-дерево отображается во вкладке браузера в том виде, который задуман разработчиками.

Основные понятия модели DOM

Основные структуры, используемые в объектной модели документа:

1. Дерево – веб-страница может быть представлена как иерархическое перевернутое дерево, начинающееся с главного элемента и расширяющееся к низу. У каждого объекта дерева есть родитель, который может быть пустым, т.е. null, и дети, если он находится не в самом низу;

2. Наследник – любой дочерний элемент узла дерева. Наследник может быть не прямым. Пример: у тега `<html>` есть дочерний элемент `<body>`, внутри которого содержится элемент `<article>`, тогда тег `<article>` это не прямой наследник `<html>`;

3. Предок – если у объекта DOM-дерева есть потомки, то для них он является предком. Здесь также возможна инклюзивность. Из примера выше у тега `<article>` есть прямой предок – тег `<body>`, а также инклюзивный родитель – тег `<html>`;

4. Братья / сестры – так объект подразумевает наличие у него «соседей», стоящих на там же уровне дерева и единого предка, не являющегося пустым;

5. События – к объектам дерева можно применять разные события при помощи JavaScript. Они могут срабатывать на действия пользователя или сетевую активность;

6. Узел – все элементы DOM-дерева являются узлами, нодами, nodes. Их можно создавать, изменять, вызывать свойства и методы с помощью JS.

Пример простого DOM-дерева.

```
1  <!DOCTYPE html>
2  <html lang="ru">
3  <head>
4      <title>HTML</title>
5  </head>
6  <body>
7      <span>Изучаем HTML</span>
8  </body>
9  </html>
```

Рис. 138. Структура DOM-дерева

В данном примере представлена веб-страница, состоящая из:

- объявления стандарта HTML (доктайп свидетельствует об использовании HTML5);
- тега `<html>` (все заключенное между ним будет воспринято как язык разметки);
- тега заголовков (`<head>`, включающего `<title>`, внутри которого текст);
- тела страницы, отображаемого пользователю (обозначен тегом `<body>`). Здесь находится только один HTML-элемент – тег `<span>` с некоторым текстовым содержимым.

Такая структура веб-документа является стандартной. Этого минимума достаточно, чтобы пройти валидацию кода.

Означенное выше наследование элементов представляется браузером как дерево.

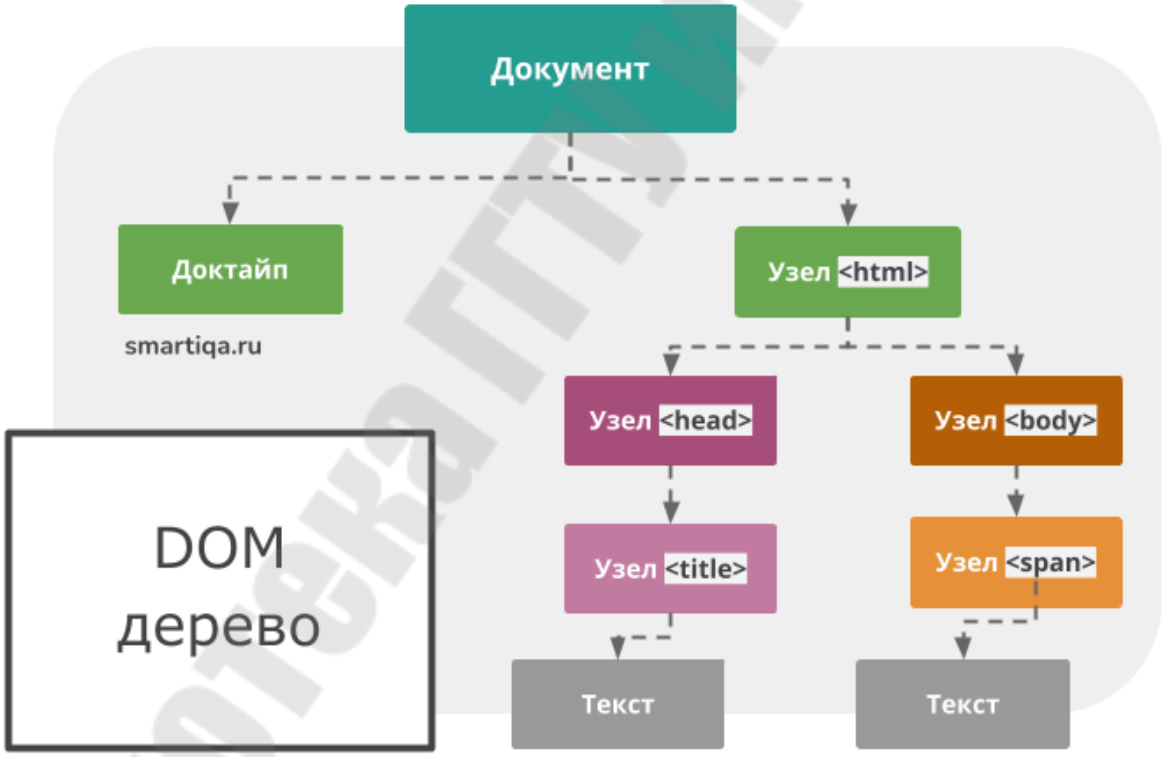


Рис. 139. Структура DOM-дерева

Анализ дерева позволяет выделить следующие объекты:

Узел `<html>` является родителем для `<body>` и `<head>`, а также инклюзивным родителем для, например, `<span>` и текста;

Узел <title> имеет наследника, т.е. текст HTML, а также прямого родителя в виде <head>;

Узлы <body> и <head> относятся к классу родственников (братья или сестры), так как расположены на одном уровне DOM-дерева и имеют общего предка <html>.

13.2. Основы JavaScript для верстки

JavaScript в веб-дизайне играет ключевую роль в создании динамичных и интерактивных веб-страниц.

Подключение JavaScript

JavaScript можно встроить на веб-страницу несколькими способами:

1. Встраивание JavaScript непосредственно в HTML используя тег <script>. Этот подход подходит для небольших скриптов.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Пример JavaScript</title>
</head>
<body>
    <h1>Привет, мир!</h1>
    <script>
        alert("Это сообщение из JavaScript");
    </script>
</body>
</html>

```

Рис. 140. Встраивание JavaScript в код HTML-страницы

2. Подключение внешнего JavaScript файла `script.js` между тегами <head>:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Пример JavaScript</title>
    <script src="script.js"></script> <!-- Подключение файла -->
</head>
```

Рис. 141. Подключение внешнего JavaScript файла `script.js`

Это более предпочтительный способ, особенно для больших проектов, так как он помогает организовать код.

Чтобы избежать ошибок с элементами, загружаемыми до выполнения скрипта, лучше помещать теги `<script>` перед закрывающим тегом `</body>` или использовать атрибут `defer`.

```
<body>
    <h1>Привет, мир!</h1>
<script src="script.js"></script> <!-- Подключение файла перед
    закрывающим тегом body -->
</body>
```

Рис. 142. Размещение тегов `<script>` перед закрывающим тегом `</body>`

```
<head>
    <script src="script.js" defer></script>
</head>
```

Рис. 143. Использование атрибута `defer` при подключении внешнего JavaScript файла `script.js`

Основные аспекты применения JavaScript:

1. Интерактивность:

- JavaScript позволяет добавлять интерактивные элементы, такие как кнопки, слайдеры, выпадающие меню и модальные окна, что улучшает взаимодействие пользователей с сайтом.

Пример1. Кода слайдера с использованием JavaScript:

```
<script>
    let currentIndex = 0;

    function showSlide(index) {
        const slides = document.querySelectorAll('#slider img'
            );
        slides.forEach((slide, i) => {
            slide.classList.toggle('active', i === index);
        });
    }

    function nextSlide() {
        currentIndex = (currentIndex + 1) % document
            .querySelectorAll('#slider img').length;
        showSlide(currentIndex);
    }

    setInterval(nextSlide, 2000); // Меняет слайд каждые 2
        секунды
</script>
```

Рис. 144. Пример кода слайдера с использованием JavaScript

Манипуляции с DOM:

- С помощью JavaScript можно динамически изменять содержимое HTML-документа, добавляя, изменяя или удаляя элементы DOM (Document Object Model).

Пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Изменение содержимого</title>
    <script>
        function changeContent() {
            document.getElementById("myText").innerHTML = "Привет, мир!";
        }
    </script>
</head>
<body>
    <h1 id="myText">Это старый текст</h1>
    <button onclick="changeContent()">Изменить текст</button>
</body>
</html>
```

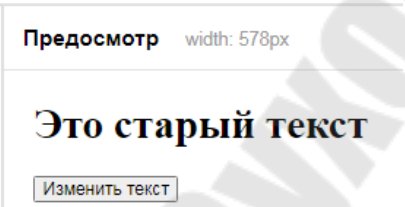


Рис. 145. Пример изменения содержимого с помощью JavaScript

2. Валидация форм:

• JavaScript используется для проверки данных введенных пользователем, перед отправкой формы на сервер.

Пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Валидация формы</title>
    <script>
        function validateForm() {
            var name = document.forms["myForm"]["name"].value;
            if (name == "") {
                alert("Имя должно быть заполнено");
                return false;
            }
        }
    </script>
</head>
<body>
    <form name="myForm" onsubmit="return validateForm()">
        Имя: <input type="text" name="name">
        <input type="submit" value="Отправить">
    </form>
</body>
</html>
```

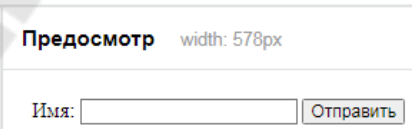


Рис. 146. Проверка данных с помощью JavaScript

3. Асинхронные запросы:

• При помощи AJAX (Asynchronous JavaScript and XML) JavaScript может отправлять и получать данные с сервера без перезагрузки страницы, что делает интерфейс более отзывчивым и быстрым.

4. Анимации и эффекты:

• С помощью JavaScript возможно создание анимаций, таких как плавное появление элементов, прокрутка страницы и другие визуальные эффекты, которые делают сайт более привлекательным.

JavaScript может быть использован для создания анимаций на странице:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Анимация</title>
    <style>
        #box {
            width: 100px;
            height: 100px;
            background-color: red;
            position: relative;
        }
    </style>
    <script>
        function moveBox() {
            let box = document.getElementById('box');
            let position = 0;

            const interval = setInterval(() => {
                if (position >= 350) {
                    clearInterval(interval);
                } else {
                    position++;
                    box.style.left = position + 'px';
                }
            }, 5);
        }
    </script>
</head>
<body>
    <div id="box"></div>
    <button onclick="moveBox()">Двигать коробку</button>
</body>
</html>
```

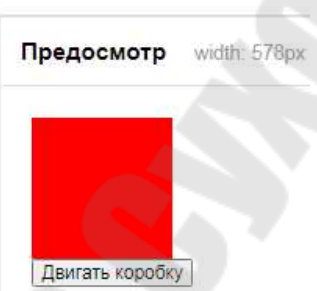


Рис. 147. Пример анимации с передвижением блока по клике на кнопку

5. Работа с API:

- JavaScript позволяет взаимодействовать с различными API (например, для карт, социальных сетей или данных), что расширяет функционал сайта.

Пример использования: вставка карты объектов на сайт с использованием специального кода; получение погоды через онлайн-сервис; код для вставки на сайт информера валют.

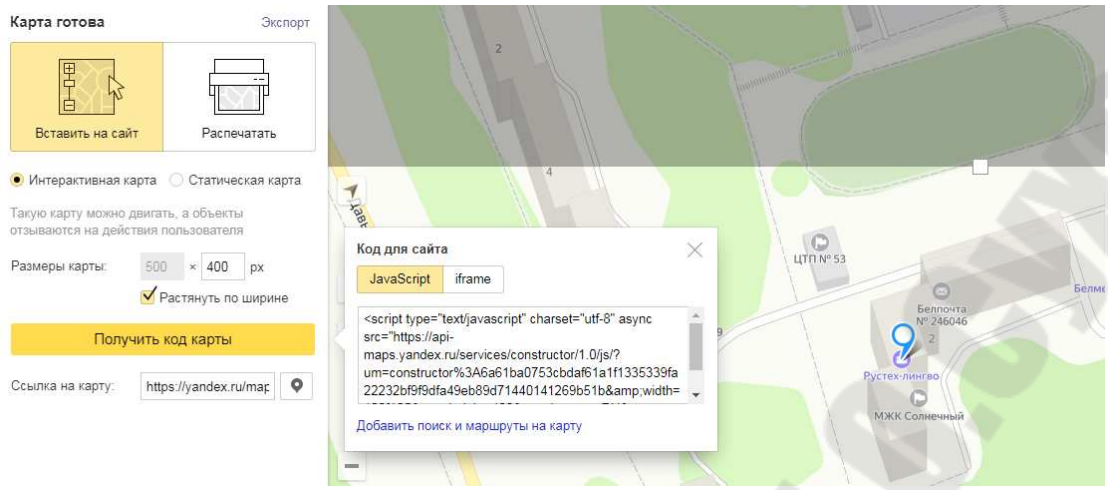


Рис. 148. Скрипт карты для сайта от сервиса Яндекс Карты

Рис. 149. Скрипт погоды

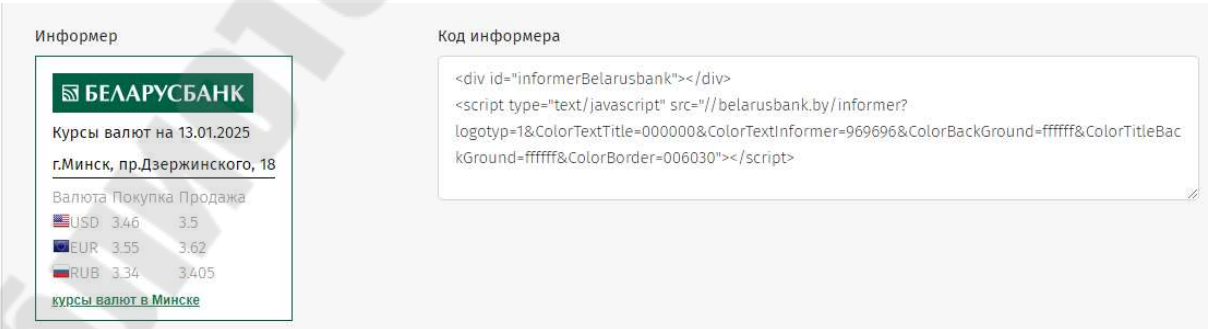


Рис. 150. Скрипт информера валют для вставки на сайт

6. Управление событиями:

- Позволяет обрабатывать события, такие как клики мыши, нажатия клавиш и другие действия пользователя, что улучшает интерактивность и отзывчивость интерфейса.

```
document.addEventListener("keydown", function(event) {
    console.log("Нажата клавиша:", event.key);
});

```

Рис. 151. Реакция на действия пользователя, такие как нажатие клавиш или клики мыши

7. Создание одностраничных приложений:

- JavaScript является основой для разработки SPA, где страницы загружаются динамически, что создает более плавный пользовательский опыт.

```
function loadPage(page) {
    fetch(page)
        .then(response => response.text())
        .then(html => {
            document.getElementById("content").innerHTML = html;
        });
}
```

Рис. 152. Загрузка новых страниц или разделов без обновления всей страницы

JavaScript является незаменимым инструментом для веб-дизайнеров и разработчиков, позволяя создавать современные, интерактивные и пользовательские веб-приложения.

14. ОРГАНИЗАЦИЯ И ОПТИМИЗАЦИЯ КОДА

Организация и оптимизация кода веб-страницы — это важные аспекты веб-разработки, которые помогают улучшить производительность, читаемость и поддерживаемость кода.

14.1. Организация кода

Основные рекомендации по организации кода:

1. Структура файлов и папок:

- Разделение кода по папкам: `html`, `css`, `javascript`, `img` и т.д.

- Использование семантических имен для файлов и папок.

Семантические имена — это названия, которые четко и понятно описывают назначение или содержание элемента в коде, делая его более понятным как для разработчиков, так и для чтения программного обеспечения. В веб-дизайне и разработке это относится как к HTML-тегам, так и к именам переменных или функций в JavaScript. Примеры семантических имен для папок: css, img (изображения), js (JavaScript) и т.д.

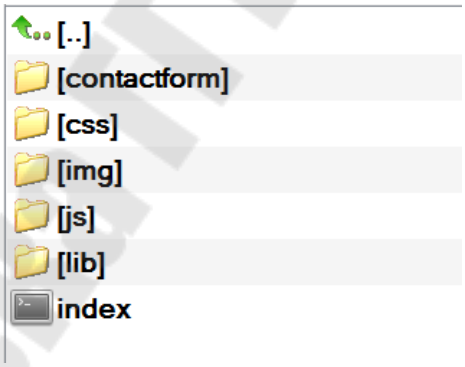


Рис. 153. Семантические имена папок

2. Чистота кода:

- Необходимо следить за отступами и выравниванием – использовать единообразные отступы.

```
index.html                                    style.css
1  <!DOCTYPE html>
2  <html lang="ru">
3  <head>
4      <meta charset="UTF-8">
5      <title>Анимация</title>
6      <style>
7          #box {
8              width: 100px;
9              height: 100px;
10             background-color: red;
11             position: relative;
12         }
13     </style>
14     <script>
15         function moveBox() {
16             let box = document.getElementById('box');
17             let position = 0;
18
19             const interval = setInterval(() => {
20                 if (position >= 350) {
21                     clearInterval(interval);
22                 } else {
23                     position++;
24                     box.style.left = position + 'px';
25                 }
26             }, 5);
27         }
28     </script>
29 </head>
30 <body>
31     <div id="box"></div>
32     <button onclick="moveBox()">Двигать коробку</button>
33 </body>
34 </html>
```

Рис. 154. Чистота кода

3. Комментарии:

- Использование комментариев для пояснения сложных участков кода. Также необходимо избегать избыточных комментариев:

```
134
135     <div id="logo" class="pull-left">
136
137       <h1><a href="#body" class="scrollto">U<span>N</span>K</a></h1>
138
139       <!-- Uncomment below if you prefer to use an image logo -->
140
141       <!-- <a href="#body"><img src="img/logo.png" alt="" title="" /></a>-->
142
143     </div>
144
```

Рис. 155. Пример использования комментариев

4. Семантическая разметка:

- Использование семантических тегов HTML5 (например, `<header>`, `<nav>`, `<article>`, `<footer>`) для улучшения структуры документа и доступности.

14.2. Оптимизация кода

Оптимизация кода — это процесс улучшения качества и производительности программного кода. Основная цель — сделать код более эффективным, понятным и легким для поддержки.

Пример ключевых аспектов оптимизации кода:

1. Минификация CSS и JavaScript, для уменьшения размера файлов и ускорения загрузки страниц.

2. Объединение файлов CSS и JavaScript в один файл, чтобы сократить количество HTTP-запросов.

3. Устранение дублирующегося кода. Используйте функции или классы для повторяющихся задач.

```
// Вместо
let total1 = price * quantity;
let total2 = price * quantity;

// Используйте
function calculateTotal(price, quantity) {
    return price * quantity;
}
```

Рис. 156. Пример использования функции для повторяющихся задач

4. Удаление неиспользуемого кода.

5. Применение библиотек и фреймворков, т.к. они оптимизированы для производительности.

6. Использование сети доставки контента (CDN) для подключения библиотек и фреймворков (например, jQuery, Bootstrap), чтобы улучшить скорость загрузки (см. рис.).

CDN links

As reference, here are our primary CDN links.

Description	URL
CSS	https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css
JS	https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js

Рис. 157. Подключение файлов bootstrap css и js с помощью метода CDN

7. Оптимизация изображений:

- Сжатие изображений перед загрузкой на сайт, использование современных форматов (например, `WebP`).

Пример: До конвертации файл jpg весил 522 кБ. После перевода в формат `webp` вес уменьшился почти в 2 раза.

Рис. 158. «Вес» файла `jpg` после перевода в формат `webp`

8. Кэширование:

- Настройка кэширования для статических ресурсов, чтобы ускорить загрузку страниц для повторных посетителей.

Оптимизация кода — это непрерывный процесс. Регулярный анализ и улучшение кода помогают поддерживать его эффективность и упрощать дальнейшую работу с проектом.

14.3. Проверка кода на валидность

Проверка кода на валидность — это процесс, который позволяет определить, соответствует ли код установленным стандартам и требованиям. Для различных языков программирования и технологий существуют свои методы и инструменты проверки. Проверка валидности кода в различных контекстах:

1. HTML и CSS:

- Для проверки HTML на валидность используют сервис W3C Markup Validation Service https://validator.w3.org/.

- Для проверки CSS используется W3C CSS Validation Service https://jigsaw.w3.org/css-validator/.

2. JavaScript:

- Для проверки JavaScript используют линтеры, такие как ESLint https://eslint.org/, который помогает выявлять ошибки и недочеты в коде, включая проблемы с синтаксисом, непригодные для использования конструкции и даже несоответствия стилю кода.

3. Анализ производительности:

- Для анализа производительности страницы используют инструменты, такие как Google PageSpeed Insights https://developers.google.com/speed/pagespeed/insights/, а также для получения рекомендаций по улучшению.

4. Доступность:

- Доступность сайта проверяют с помощью специальных инструментов, таких как https://www.deque.com/axe/ или https://wave.webaim.org/.

Регулярная проверка кода на валидность помогает обеспечить его качество, улучшает читаемость, уменьшает количество ошибок и упрощает сопровождение.

15. ПРОБЛЕМЫ КРОССБРАУЗЕРНОСТИ

Кроссбраузерность — это способность веб-страницы корректно отображаться и функционировать в различных браузерах. Проблемы кроссбраузерности могут возникать по нескольким причинам:

15.1. Различия между браузерами

1. Разные реализации стандартов:

- Разные браузеры могут по-разному интерпретировать и отображать одни и те же HTML и CSS. Например, некоторые свойства CSS могут поддерживаться в одном браузере, но не поддерживаться в другом.

2. Проблемы с вёрсткой:

- Элементы могут отображаться по-разному в различных браузерах из-за различий в рендеринге. Например, отступы, границы и размеры могут отличаться.

3. Поддержка новых свойств и функций:

- Новые свойства CSS (например, CSS Grid, Flexbox) могут не поддерживаться старыми версиями браузеров. Это может привести к неправильному отображению элементов.

4. JavaScript:

- Некоторые функции JavaScript могут работать в одном браузере и не работать в другом. Например, некоторые методы DOM могут иметь разное поведение.

5. Ошибки в вёрстке:

- Неправильное использование HTML, например, незакрытые теги или неправильная вложенность, может привести к проблемам в рендеринге.

6. Проблемы с шрифтами и графикой:

- Разные браузеры могут по-разному обрабатывать шрифты и изображения. Например, использование нестандартных шрифтов может привести к различиям в отображении текста.

15.2. Решения для обеспечения кроссбраузерной совместимости

Стратегии для обеспечения кроссбраузерности

1. Использование CSS Reset или Normalize.css.

CSS Reset и Normalize.css — это два подхода, используемые для устранения различий в базовых стилях браузеров и создания одинакового отображения веб-страниц на различных устройствах и браузерах.

CSS Reset — это набор стилей, который сбрасывает (или обнуляет) стили по умолчанию, применяемые браузерами. Основная цель — удалить ненужные отступы, поля и другие стили, которые могут различаться в разных браузерах.

Normalize.css — это альтернативный подход к CSS Reset, который не просто сбрасывает стили, но и нормализует их. Он поддерживает стандартные стили для HTML-элементов, что позволяет сохранить консистентность при отображении, сохраняя при этом разумные значения по умолчанию.

2. Проверка совместимости технологий:

- Используйте ресурсы, такие как Can I use https://caniuse.com/, чтобы проверить поддержку различных свойств CSS и JavaScript в разных браузерах.

3. Тестирование в разных браузерах:

- Регулярно тестируйте сайт в разных браузерах (Chrome, Firefox, Safari, Edge, Internet Explorer) и на разных устройствах.

4. Использование полифиллов:

- Для поддержки новых функций в старых браузерах используйте полифиллы, которые добавляют недостающий функционал.

Полифилл (`polyfill`) — это скрипт или библиотека, которая добавляет поддержку современных возможностей JavaScript (или других технологий) в устаревшие или ограниченные браузеры, которые не поддерживают эти функции по умолчанию. Это позволяет разработчикам использовать новые API и методы, не беспокоясь о совместимости с разными браузерами.

Первичное функциональное тестирование кода веб-страницы

Первичное функциональное тестирование — это процесс проверки работы веб-страницы на наличие ошибок и корректности исполнения функционала.

Основные этапы для проведения такого тестирования:

1. Проверка навигации:

- Необходимо убедиться, что все ссылки работают и ведут на правильные страницы. Проверить наличие и корректность навигационного меню.

2. Формы:

- Необходимо проверить все формы на наличие ошибок. Убедится, что валидация работает корректно. Проверить отправку форм и обработку данных.

3. Кнопки и интерактивные элементы:

- необходимо убедиться, что все кнопки и интерактивные элементы реагируют на клики и выполняют свои функции.

4. Проверка отображения:

- Проверить страницу на отображение в разных браузерах и на разных устройствах (мобильные, планшеты, десктопы).

5. JavaScript:

- Необходимо убедиться, что все скрипты работают корректно и нет ошибок в консоли браузера.

6. Кроссбраузерное тестирование:

- Необходимо провести тестирование в разных браузерах и на разных устройствах, чтобы выявить возможные проблемы кроссбраузерности.

7. Скорость загрузки:

- Необходимо проверить время загрузки страницы, чтобы убедиться, что она загружается быстро и без задержек.

8. Доступность:

- Необходимо проверить доступность страницы для пользователей с ограниченными возможностями, используя инструменты для анализа доступности.

Проведение этих тестов поможет выявить и исправить проблемы, обеспечивая тем самым более качественный пользовательский опыт.

ПРИЛОЖЕНИЯ

Документация и полезные ссылки

1. Основные веб-стандарты:
HTML5: https://www.w3.org/TR/html52/
CSS3: https://www.w3.org/Style/CSS/specs.en.html
DOM-стандарт: https://dom.spec.whatwg.org
JavaScript: https://dom.spec.whatwg.org/

2. Иконочные шрифты:
Font Awesome https://fontawesome.com
Google Fonts: https://fonts.google.com

3. Онлайн-редакторы с предварительным просмотром:
HTML/CSS https://webtoolsner.ru/tool/html-editor/
HTML/CSS/JS https://codly.ru/editor/

4. Редакторы кода:
Visual Studio Code: https://code.visualstudio.com/
Sublime Text: https://www.sublimetext.com/

5. Фреймворки и библиотеки:
Bootstrap: https://getbootstrap.com/
React https://reactjs.org/

6. Инструменты для работы с графикой:
Adobe Photoshop: https://www.adobe.com/products/photoshop.html
Figma: https://www.figma.com/
Sketch: https://www.sketch.com/

7. Инструменты для тестирования и отладки:
Google Chrome DevTools:
https://developers.google.com/web/tools/chrome-devtools
https://www.deque.com/axe/ или https://wave.webaim.org/ - проверка доступности сайта

8. Системы контроля версий:
Git: https://git-scm.com/

GitHub: https://github.com/
GitLab: https://gitlab.com/

9. Инструменты для проверки кода:
W3C Validator: https://validator.w3.org/
CSS Lint: http://csslint.net/
Can I use https://caniuse.com (проверка совместимости технологий).
Normalize.css на GitHub https://github.com/necolas/normalize.css

Список использованных интернет-ресурсов

1. Веб-дизайн — Википедия [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/Веб-дизайн (дата обращения: 1.08.2024).
2. Зачем нужен нормализатор CSS. [Электронный ресурс] – URL: https://thecode.media/normalize/ (дата обращения: 7.10.2024).
3. Изучаем CSS Grid на примерах. [Электронный ресурс] – URL: https://medium.com/nuances-of-programming/изучаем-css-grid-на-примерах-c54286e6c462 (дата обращения: 25.06.2024).
4. История веб-дизайна — это история интернета. [Электронный ресурс] – URL: https://skillbox.ru/media/design/veb_dizayn_s_nulya/ (дата обращения: 10.03.2024).
5. История развития HTML. [Электронный ресурс] – URL: https://vertex-academy.com/tutorials/ru/html_history/ (дата обращения: 25.03.2024).
6. История создания HTML и CSS. [Электронный ресурс] – URL: https://sky.pro/wiki/html/istoriya-sozdaniya-html-i-css/ (дата обращения: 15.03.2024).
7. Как адаптировать сайт для мобильных устройств: методы и примеры. [Электронный ресурс] – URL: https://practicum.yandex.ru/blog/kak-adaptirovat-sayt-pod-mobilnye-ustroystva/ (дата обращения: 5.09.2024).
8. Начало работы с API Google Fonts. [Электронный ресурс] – URL: https://developers.google.com/fonts/docs/getting_started?hl=ru (дата обращения: 27.08.2024).
9. Технология FlexBox. [Электронный ресурс] – URL: https://skillbox.ru/media/code/shpargalka-po-flexbox-svoystva-primery-ispolzovaniya-i-besplatnye-trenazhyery/ (дата обращения: 16.07.2024).
10. Урок 2. HTML: Понятие, стандарты, тэги и атрибуты. [Электронный ресурс] – URL: https://smartiqa.ru/courses/web/lesson-2-html (дата обращения: 5.05.2024).
11. Урок 4. Современный CSS: Основные понятия, селекторы, блочная модель. [Электронный ресурс] – URL: https://smartiqa.ru/courses/web/lesson-4-css (дата обращения: 5.06.2024).
12. Уроки. [Электронный ресурс] – URL: https://www.schoolsw3.com/css/css_intro.php (дата обращения: 23.04.2024).

13. Установочный шаблон CMS Joomla «Кассиопея» [Электронный ресурс] – URL: https://cassiopeia.joomla.com (дата обращения: 7.12.2024).

14. Учимся верстать в сетке: большой гайд по CSS Grid. [Электронный ресурс] – URL: https://skillbox.ru/media/code/uchimsya-verstat-v-setke-bolshoy-gayd-po-css-grid/ c54286e6c462 (дата обращения: 1.07.2024).

15. Шпаргалка по Flexbox: свойства, примеры использования, тренажёры [Электронный ресурс] – URL: https://skillbox.ru/media/code/shpargalka-po-flexbox-svoystva-primery-ispolzovaniya-i-besplatnye-trenazhyery/ (дата обращения: 26.08.2024).

16. Шрифты. [Электронный ресурс] – URL: https://m.site-do.ru/css/css6.php (дата обращения: 5.04.2024).

ЛИТЕРАТУРА

1. Нагаева, И. А. Основы web-дизайна. Методика проектирования : учебное пособие : [12+] / И. А. Нагаева, И. А. Кузнецов, А. Б. Фролов. – Москва ; Берлин : Директ-Медиа, 2021. – 236 с. : ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=602208 (дата обращения: 17.01.2025). – Библиогр. в кн. – ISBN 978-5-4499-1957-1. – Текст : электронный.

2. Беликова, С. А. Основы HTML и CSS: проектирование и дизайн веб-сайтов : учебное пособие по курсу «Web-разработка» : [16+] / С. А. Беликова, А. Н. Беликов ; Южный федеральный университет. – Ростов-на-Дону ; Таганрог : Южный федеральный университет, 2020. – 176 с. : ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=598663 (дата обращения: 17.01.2025). – Библиогр. в кн. – ISBN 978-5-9275-3435-7. – Текст : электронный.

3. Солодушкин, С. И. Web и DHTML : учебное пособие / С. И. Солодушкин, И. Ф. Юманова ; науч. ред. В. Г. Пименов ; Уральский федеральный университет им. первого Президента России Б. Н. Ельцина. – Екатеринбург : Издательство Уральского университета, 2018. – 131 с. : схем., табл., ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=696213 (дата обращения: 17.01.2025). – ISBN 978-5-7996-2410-1. – Текст : электронный.

Учебно-методическая литература

1. Верстка WEB-страниц [Электронный ресурс] : курс лекций по одноименной дисциплине для слушателей специальности 1-40 01 74 " WEB-дизайн и компьютерная графика" заочной формы обучения / Т. В. Тихоненко. - Гомель : ГГТУ им. П. О. Сухого, 2013. - 172 с.

Тихоненко, Т. В. Верстка web-страниц [Электронный ресурс] : лабораторный практикум по одноименной дисциплине для слушателей специальности 1-40 01 74 "Web-дизайн и компьютерная графика" заочной формы обучения / Т. В. Тихоненко, А. И. Рябченко ; Министерство образования Республики Беларусь. - Гомель : ГГТУ им. П. О. Сухого, 2013. - 59 с.

ВЕРСТКА ВЕБ-СТРАНИЦ

Пособие
для слушателей специальности переподготовки
9-09-0611-02 «Веб-дизайн и компьютерная графика»
заочной формы обучения

Составитель **Леонова** Вероника Николаевна

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 13.02.25.
Рег. № 58Е.
http://www.gstu.by