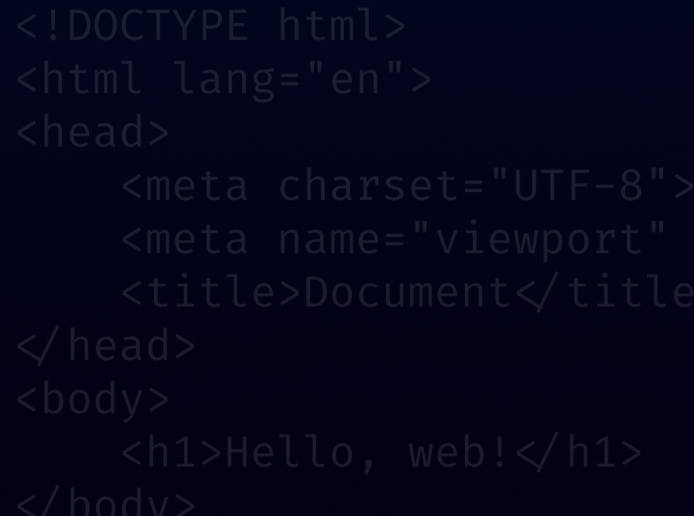
учебное пособие для самостоятельного изучения

Программирование в Web



Автор: whoennrl

Содержание

Введение 2
1. HTML — структура страницы 3
1.1. Что такое HTML — язык разметки, а не программирования 3
1.2. Минимальная структура документа: <!DOCTYPE>, <html>, <head>, <body> 4
1.3. Основные теги: заголовки, параграфы, списки, ссылки, изображения 5
1.4. Формы: <form>, <input>, <textarea>, <button> — только базовые типы 7
1.5. Семантика: <header>, <main>, <section>, <footer> — кратко и по делу 9
2. CSS — оформление 11
2.1. Подключение CSS: внутренние и внешние стили 11
2.2. Селекторы: тег, класс, id 13
2.3. Базовые свойства: цвет, шрифт, отступы, рамки 15
2.4. Блочная модель: margin / padding / border 18
2.5. Flexbox — основной инструмент для макетов 20
2.6. Адаптивность: один пример с медиазапросом (max-width: 768px) 22
3. JavaScript — интерактивность 23
3.1. Подключение скрипта и базовый синтаксис 23
3.2. Переменные, условия (if), циклы (for) 25
3.3. Функции: объявление и вызов 27
3.4. Работа с DOM: querySelector(), textContent, addEventListener() 29
3.5. Пример: скрыть/показать блок, валидация поля формы 31
3.6. fetch() — один пример отправки данных на PHP-скрипт 34
4. Бэкенд: сервер и база данных 36
4.1. Что такое PHP и как он работает на сервере 36
4.2. Синтаксис: переменные, строки, условия, циклы, функции 38
4.3. Работа с формами: $_POST, $_GET 40
4.4. Безопасность: htmlspecialchars() и фильтрация ввода 43
4.5. Пример: приём данных из формы и вывод на странице 45
5. MySQL и работа с базой данных 48
5.1. Что такое база данных и зачем она нужна 48
5.2. Основы SQL: CREATE TABLE, INSERT, SELECT, UPDATE, DELETE 49
5.3. Подключение PHP к MySQL через mysqli 52
5.4. Подготовленные запросы — защита от SQL-инъекций 54
5.5. Пример: сохранение комментария в базу и вывод списка 57
Заключение и что дальше 58

Введение

Современные веб-приложения состоят из двух основных частей: **фронтенда** — той части, которую видит и с которой взаимодействует пользователь в браузере, и **бэкенда** — серверной логики, обрабатывающей запросы, хранящей данные и управляющей приложением «под капотом». Чтобы создавать даже самые простые сайты с интерактивностью и хранением информации (например, гостевую книгу, блог или каталог товаров), необходимо понимать обе стороны.

Это учебное пособие предназначено для начинающих разработчиков, которые хотят пошагово освоить основы **веб-разработки с нуля**, используя только бесплатные и доступные технологии:

- **HTML** для структуры страницы,
- **CSS** для оформления,
- **JavaScript** для клиентской интерактивности,
- **PHP** как серверный язык,
- **MySQL** как систему управления базой данных.

Курс построен так, чтобы каждая тема логично вытекала из предыдущей. Мы начнём с создания статической HTML-страницы, затем добавим стили и поведение, после чего перейдём к серверной части: обработке форм, взаимодействию с базой данных и, наконец, соберём всё вместе в полноценное приложение — **гостевую книгу**.

Все примеры ориентированы на **локальную разработку**: вы сможете работать на своём компьютере без подключения к интернету, используя такие инструменты, как **VS Code**, **XAMPP** и **phpMyAdmin**.

Готовы? Тогда начнём с самого начала — с языка гипертекстовой разметки **HTML**.

1. HTML — структура страницы

1.1. Что такое HTML — язык разметки, а не программирования

HTML *(HyperText Markup Language)* — это не язык программирования, а **язык разметки**. Он предназначен для описания структуры веб-страницы: какие блоки на ней есть, где заголовок, где текст, где изображение или форма ввода. Браузер читает HTML-документ и отображает его содержимое в соответствии с тегами, которые вы указали.

Важно понимать разницу:

- **Программирование** — это описание логики: «что делать, если…», «повторять пока…», «вычислить сумму…».
- **Разметка** — это описание структуры: «это заголовок», «это абзац», «это список».

HTML не выполняет вычислений, не хранит данные и не реагирует на действия пользователя — за всё это отвечают другие технологии (JavaScript, PHP и т.д.). Но без HTML не существует ни одной веб-страницы.

Простейший HTML-документ выглядит так:

Учебное пособие по web программированию

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Моя первая страница</title>
</head>
<body>
  <h1>Привет, мир!</h1>
  <p>Это мой первый HTML-документ.</p>
</body>
</html>
```

Когда вы откроете этот файл в браузере, вы увидите заголовок и абзац текста. Никакой логики — только структура. Именно с неё мы и начнём.

1.2. Минимальная структура документа: <!DOCTYPE>, <html>, <head>, <body>

Любой корректный HTML-документ должен содержать **четыре обязательных элемента**:

1. <!DOCTYPE html> — объявляет, что документ использует HTML5. Это не тег, а инструкция для браузера.
2. <html> — корневой элемент всей страницы. Всё содержимое размещается внутри него.
3. <head> — «голова» документа. Здесь не отображается контент, но задаются метаданные: заголовок, кодировка, подключение стилей и скриптов.
4. <body> — «тело» документа. Всё, что пользователь видит на странице, размещается здесь.

Пример с пояснениями:

Учебное пособие по web программированию

```
<!DOCTYPE html>                    <!-- Версия HTML -->
<html lang="ru">                   <!-- Язык страницы -->
<head>
  <meta charset="utf-8">           <!-- Кодировка символов -->
  <title>Заголовок вкладки</title>  <!-- Заголовок, отображаемый в браузере -->
</head>
<body>
  <!-- Видимый контент -->
</body>
</html>
```

Без <meta charset="utf-8"> русские буквы могут отображаться как «кракозябры», поэтому эту строку стоит добавлять всегда.

1.3. Основные теги: заголовки, параграфы, списки, ссылки, изображения

Теперь, когда у нас есть базовая структура HTML-документа, можно наполнять страницу содержимым. Ниже перечислены самые часто используемые теги для создания базового контента.

Заголовки (<h1> – <h6>)

Заголовки помогают организовать текст по уровням важности. <h1> — самый главный (обычно один на странице), <h6> — самый мелкий.

```
Учебное пособие по web программированию

<h1>Основной заголовок</h1>
<h2>Подзаголовок раздела</h2>
<h3>Подпункт</h3>
```

Параграфы (<p>)

Обычный текст разбивается на абзацы с помощью тега <p>:

```
Учебное пособие по web программированию

<p>Это первый абзац. Он содержит вводную информацию.</p>
<p>А это второй — с дополнительными деталями.</p>
```

Списки

Существуют два типа списков:

- **Нумерованный список** (<ol> — ordered list):

```
Учебное пособие по web программированию

<ol>
  <li>Первый шаг</li>
  <li>Второй шаг</li>
  <li>Третий шаг</li>
</ol>
```

- **Маркированный список** (<ul> — unordered list):

```
Учебное пособие по web программированию

<ul>
  <li>Яблоко</li>
  <li>Банан</li>
  <li>Апельсин</li>
</ul>
```

Каждый элемент списка обязательно оборачивается в <li> (list item).

Ссылки (<a>)

Гиперссылки позволяют переходить на другие страницы или ресурсы:

```
Учебное пособие по web программированию

<a href="https://example.com">Перейти на Example.com</a>
```

Атрибут href указывает адрес назначения. Чтобы ссылка открывалась в новой вкладке, добавьте target="_blank":

```
Учебное пособие по web программированию

<a href="https://example.com" target="_blank">Открыть в новой вкладке</a>
```

Изображения (<img>)

Для вставки изображения используется самозакрывающийся тег <img>:

```
Учебное пособие по web программированию

<img src="images/photo.jpg" alt="Описание изображения">
```

- src — путь к файлу изображения (относительный или абсолютный).
- alt — альтернативный текст. Он отображается, если изображение не загрузилось, и используется для доступности (например, скринридерами).

Важно: файл изображения должен находиться в указанной папке *(в примере — images/photo.jpg)*. Если путь неверный, изображение не отобразится, но alt-текст останется видимым.

1.4. Формы: <form>, <input>, <textarea>, <button> — только базовые типы

Формы — это способ получить **ввод от пользователя**: имя, email, сообщение и т.д. Даже простая гостевая книга или поиск на сайте начинаются с формы.

Основной контейнер: <form>

Все элементы формы размещаются внутри тега <form>:

```
Учебное пособие по web программированию

<form action="process.php" method="POST">
  <!-- Поля формы -->
</form>
```

- action — адрес (чаще всего PHP-скрипт), куда отправятся данные.
- method — способ отправки: GET (в строке URL) или POST (в теле запроса, безопаснее для конфиденциальных данных).

Для учебных целей мы будем использовать method="POST".

Поле ввода: <input>

Самый универсальный элемент. Его поведение зависит от атрибута type.

Основные типы:

- text — обычная строка:

```
Учебное пособие по web программированию

<input type="text" name="username" placeholder="Ваше имя">
```

- email — email с базовой проверкой в браузере:

```
Учебное пособие по web программированию

<input type="email" name="email" placeholder="Ваш email">
```

- submit — кнопка отправки:

```
Учебное пособие по web программированию

<input type="submit" value="Отправить">
```

Атрибут name **обязателен** — именно он определяет, как PHP будет обращаться к значению ($_POST['username']).

Многострочное поле: <textarea>

Для длинных сообщений (комментарии, отзывы):

```
Учебное пособие по web программированию

<textarea name="message" placeholder="Ваше сообщение..."></textarea>
```

В отличие от <input>, текст размещается **между открывающим и закрывающим тегами**.

Кнопка: <button>

Альтернатива <input type="submit">, но с большей гибкостью:

```
Учебное пособие по web программированию

<button type="submit">Отправить</button>
```

Если не указать type="submit", браузер может считать кнопку обычной (и форма не отправится!).

Полный пример формы

```
Учебное пособие по web программированию

<form action="guestbook.php" method="POST">
  <label>
    Имя:<br>
    <input type="text" name="name" required>
  </label><br><br>

  <label>
    Email:<br>
    <input type="email" name="email" required>
  </label><br><br>

  <label>
    Сообщение:<br>
    <textarea name="message" rows="4" required></textarea>
  </label><br><br>

  <button type="submit">Оставить запись</button>
</form>
```

- Атрибут required — встроенная HTML5-валидация: браузер не даст отправить форму, если поле пустое.
- Тег <label> связывает подпись с полем — улучшает доступность и UX (клик по надписи фокусирует поле).

1.5. Семантика: <header>, <main>, <section>, <footer> — кратко и по делу

Современный HTML — это не только про отображение, но и про **смысл** содержимого. Семантические теги помогают браузерам, поисковым системам и вспомогательным технологиям (например, скринридерам) понимать структуру страницы.

Вот ключевые семантические элементы, которые стоит использовать даже в простых проектах:

<header>

Содержит вводную часть страницы или раздела: логотип, навигацию, заголовок.

```
Учебное пособие по web программированию

<header>
  <h1>Мой сайт</h1>
  <nav>
    <a href="/">Главная</a> |
    <a href="/about">О сайте</a>
  </nav>
</header>
```

<main>

Обозначает **основное содержимое** страницы — то, ради чего пользователь сюда пришёл. На одной странице должен быть только один <main>.

```
Учебное пособие по web программированию

<main>
  <h2>Добро пожаловать!</h2>
  <p>Здесь вы найдёте полезные статьи по веб-разработке.</p>
</main>
```

<section>

Группирует тематически связанный контент — например, раздел блога или блок отзывов.

```
Учебное пособие по web программированию

<section>
  <h2>Последние записи</h2>
  <p>Запись 1...</p>
  <p>Запись 2...</p>
</section>
```

> ⚠️ *Не путайте <section> с . — просто контейнер без смысла. Используйте <section>, только если блок имеет собственный заголовок и логически завершён.*

<footer>

Содержит заключительную информацию: авторские права, контакты, ссылки.

Учебное пособие по web программированию

```
<footer>
  <p>&copy; 2025 Мой сайт. Все права защищены.</p>
</footer>
```

Пример полной структуры

Учебное пособие по web программированию

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <title>Семантическая страница</title>
</head>
<body>
  <header>
    <h1>Гостевая книга</h1>
  </header>

  <main>
    <section>
      <h2>Оставьте сообщение</h2>
      <!-- форма из раздела 1.4 -->
    </section>

    <section>
      <h2>Последние записи</h2>
      <p>Иван: Спасибо за сайт!</p>
      <p>Мария: Очень полезно!</p>
    </section>
  </main>

  <footer>
    <p>Создано для обучения. 2025 г.</p>
  </footer>
</body>
</html>
```

Семантическая разметка делает код **понятнее**, **лучше индексируется поисковиками** и **доступнее** для людей с ограниченными возможностями — даже если вы не добавляете CSS, структура остаётся логичной.

2. CSS — оформление

2.1. Подключение CSS: внутренние и внешние стили

HTML отвечает за **структуру**, а **CSS** *(Cascading Style Sheets)* — за **внешний вид**: цвета, шрифты, отступы, расположение элементов. Чтобы применить стили к HTML-документу, их нужно подключить.

Существует три способа:

1. Встроенные стили (inline) — не рекомендуются для обучения

Указываются прямо в атрибуте style тега:

Учебное пособие по web программированию

```
<p style="color: blue; font-size: 18px;">Текст синего цвета</p>
```

Минусы: дублирование кода, сложность поддержки. Используются редко — только для исключений.

2. Внутренние стили (внутри <style>**)**

Размещаются в <head> документа:

Учебное пособие по web программированию

```
<head>
  <meta charset="utf-8">
  <title>Моя страница</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f5f5f5;
    }
    h1 {
      color: #333;
    }
  </style>
</head>
```

Подходит для **одной страницы**, но если сайт состоит из нескольких страниц — стили придётся копировать.

3. Внешние стили — рекомендуемый способ

Создаётся отдельный файл с расширением .css (например, styles.css), и подключается через тег <link>:

```
Учебное пособие по web программированию

<head>
  <meta charset="utf-8">
  <title>Моя страница</title>
  <link rel="stylesheet" href="styles.css">
</head>
```

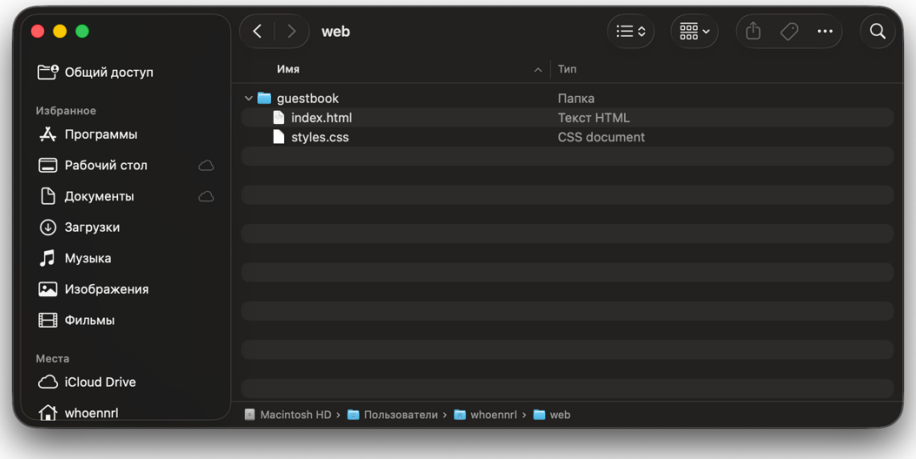
Файл styles.css:

```
Учебное пособие по web программированию

body {
  font-family: Arial, sans-serif;
  background-color: #f5f5f5;
  margin: 0;
  padding: 20px;
}

h1 {
  color: #333;
}
```

Преимущества:

- Разделение структуры и оформления.
- Один файл стилей можно подключить ко **всем страницам** сайта.
- Проще редактировать и поддерживать.

> 💡 *Совет: В учебных проектах всегда используйте внешние стили. Создайте папку проекта, например guestbook/, внутри — index.html и styles.css.*



2.2. Селекторы: тег, класс, id

Чтобы применить стили к элементам HTML, CSS использует **селекторы** — правила, указывающие, к каким элементам относится блок стилей.

1. Селектор по тегу

Применяется ко **всем элементам** с таким тегом:

```
Учебное пособие по web программированию

p {
  color: green;
}
```

Все абзацы (<p>) на странице станут зелёными.

2. Селектор по классу (.имя)

Классы — основной способ стилизации отдельных элементов или групп. Один элемент может иметь несколько классов.

HTML:

```
Учебное пособие по web программированию

<p class="intro">Это вводный абзац.</p>
<p class="note warning">Внимание: важная информация!</p>
```

CSS:

```
Учебное пособие по web программированию

.intro {
  font-size: 1.2em;
  font-weight: bold;
}

.note {
  background-color: #fff3cd;
  padding: 10px;
}

.warning {
  border-left: 4px solid #ffc107;
}
```

- Точка (.) означает, что селектор применяется к **классу**.
- Элемент с class="note warning" получит стили **обоих** классов.

3. Селектор по id (#имя)

Идентификатор (id) должен быть **уникальным** на странице. Используется редко для стилей (чаще — в JavaScript).

HTML:

```
Учебное пособие по web программированию

<header id="main-header">
  <h1>Гостевая книга</h1>
</header>
```

CSS:

```
Учебное пособие по web программированию

#main-header {
  background-color: #007bff;
  color: white;
  padding: 1rem;
}
```

Диез (#) означает, что селектор применяется к **id**.

> ⚠️ *Правило хорошего тона:*
>
> - *Используйте **классы** для стилизации.*
> - *Используйте **id** только когда элемент действительно уникален и нужен для скриптов или якорных ссылок (например, <a href="#contacts">).*

Приоритет селекторов

Если к одному элементу применяются несколько правил, браузер выбирает по **специфичности**:

!important > id > класс / псевдокласс / атрибут > тег

Пример:

```
Учебное пособие по web программированию

<p id="special" class="text">Текст</p>
```

```
Учебное пособие по web программированию

p { color: black; }
.text { color: blue; }
#special { color: red; }
```

2.3. Базовые свойства: цвет, шрифт, отступы, рамки

CSS предлагает сотни свойств, но для начала достаточно освоить несколько ключевых. Они позволяют быстро сделать страницу читаемой и визуально приятной.

1. Цвет (color и background-color)

- color — цвет **текста**.
- background-color — цвет **фона** элемента.

```
Учебное пособие по web программированию

body {
  color: #333;                /* тёмно-серый текст */
  background-color: #f9f9f9;  /* почти белый фон */
}

.highlight {
  background-color: #ffeb3b;  /* жёлтая подсветка */
  color: #000;                /* чёрный текст на жёлтом */
}
```

Цвета можно задавать:

- в **шестнадцатеричном формате**: #ff0000
- в **именованном**: red, blue, lightgray
- в **RGB/RGBA**: rgb(255, 0, 0) или rgba(255, 0, 0, 0.5) (последнее — с прозрачностью)

2. Шрифт (font-family, font-size, font-weight)

- font-family — список шрифтов (с запасными вариантами):

```
Учебное пособие по web программированию

body {
  font-family: "Helvetica Neue", Arial, sans-serif;
}
```

Браузер возьмёт первый доступный шрифт. sans-serif — универсальный запасной.

- font-size — размер шрифта:

```
Учебное пособие по web программированию

h1 { font-size: 2rem; }     /* 1rem = размер шрифта body (обычно 16px) */
p  { font-size: 16px; }
```

- font-weight — жирность:

```
Учебное пособие по web программированию

.bold { font-weight: bold; }   /* или числом: 700 */
.normal { font-weight: normal; } /* или 400 */
```

3. Отступы: margin и padding

Эти свойства управляют **пространством вокруг и внутри** элемента — подробнее в разделе 2.4, но базовое использование:

- padding — **внутренний отступ** (между содержимым и рамкой):

```
Учебное пособие по web программированию

.box {
  padding: 10px;        /* со всех сторон */
  padding: 5px 10px;    /* вертикаль | горизонталь */
}
```

- margin — **внешний отступ** (между элементом и другими блоками):

```
Учебное пособие по web программированию

.box {
  margin: 20px 0;       /* сверху и снизу 20px, слева и справа 0 */
}
```

4. Рамки (border)

Добавляют видимую границу вокруг элемента:

```
Учебное пособие по web программированию

.notice {
  border: 2px solid #28a745;  /* толщина стиль цвет */
  padding: 12px;
}
```

Возможные значения стиля:

- solid — сплошная
- dashed — пунктир
- dotted — точечная

Можно задавать стороны отдельно:

```
Учебное пособие по web программированию

.top-border {
  border-top: 1px dashed #ccc;
}
```

Пример: стиль для формы из раздела 1.4

```
Учебное пособие по web программированию

form {
  background-color: white;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 8px;       /* скругление углов */
  max-width: 500px;
  margin: 20px 0;
}

input[type="text"],
input[type="email"],
textarea {
  width: 100%;
  padding: 8px;
  margin: 6px 0;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;   /* ширина включает padding и border */
}

button {
  background-color: #007bff;
  color: white;
  padding: 10px 16px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

button:hover {
  background-color: #0056b3; /* затемнение при наведении */
}
```

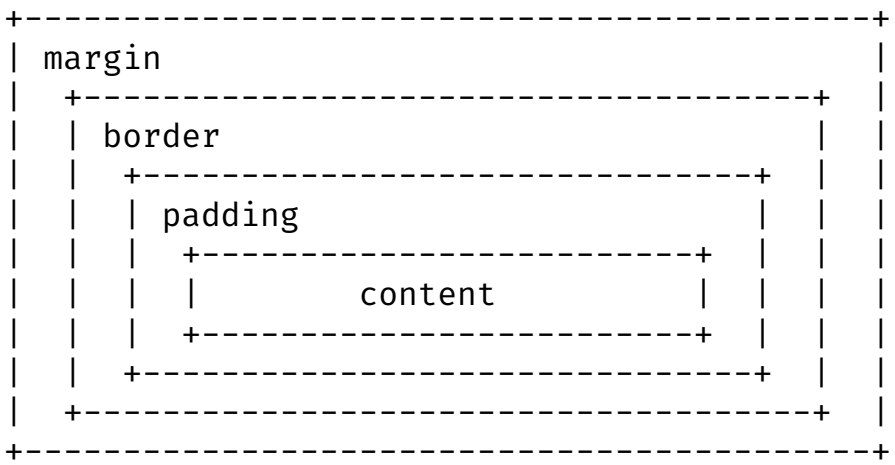
> 💡 *Совет: box-sizing: border-box — одно из самых полезных правил. Оно заставляет width: 100% включать padding и border, чтобы элемент не «вылазил» за пределы контейнера.*

2.4. Блочная модель: margin / padding / border

Всё в HTML — **прямоугольник**. Даже круглое изображение с border-radius на самом деле занимает прямоугольную область. Чтобы правильно размещать элементы и управлять их размерами, нужно понимать **блочную (box) модель**.

Каждый элемент состоит из четырёх слоёв (изнутри наружу):

1. **Контент (content)** — текст, изображение и др.
2. **Внутренний отступ (padding)** — пространство между контентом и рамкой.
3. **Рамка (border)** — видимая граница (может быть невидимой, если не задана).
4. **Внешний отступ (margin)** — пространство между этим элементом и соседними.



Как рассчитывается ширина и высота?

По умолчанию (box-sizing: content-box, стандартное поведение):

Учебное пособие по web программированию

```css
.box {
  width: 200px;
  padding: 10px;
  border: 5px solid black;
}
```

Фактическая ширина = 200px (контент) + 2×10px (padding) + 2×5px (border) = **230px**.

Это часто приводит к неожиданностям: элемент «вылезает» за пределы колонки.

Решение: box-sizing: border-box

Учебное пособие по web программированию

```css
* {
  box-sizing: border-box;
}
```

Это правило применяется ко **всем элементам** (* — универсальный селектор). Теперь:

- width: 200px означает **общую ширину** элемента, включая padding и border.
- Контент сжимается, чтобы уместиться.

Рекомендуется добавлять этот сброс в начало каждого CSS-файла.

Margin collapse (схлопывание отступов)

Особенность блочной модели: **вертикальные margin у соседних блоков «схлопываются»** — берётся больший из них, а не сумма.

Учебное пособие по web программированию

```
<p>Абзац 1</p>
<p>Абзац 2</p>
```

Учебное пособие по web программированию

```
p {
  margin-top: 20px;
  margin-bottom: 20px;
}
```

Расстояние между абзацами — **20px**, а не 40px.

> *Это поведение нормальное, но может сбивать с толку. Чтобы избежать неожиданностей, старайтесь задавать отступы только с одной стороны (например, только margin-bottom).*

Блочная модель — фундамент компоновки. Поняв её, вы сможете предсказуемо управлять размерами и отступами.

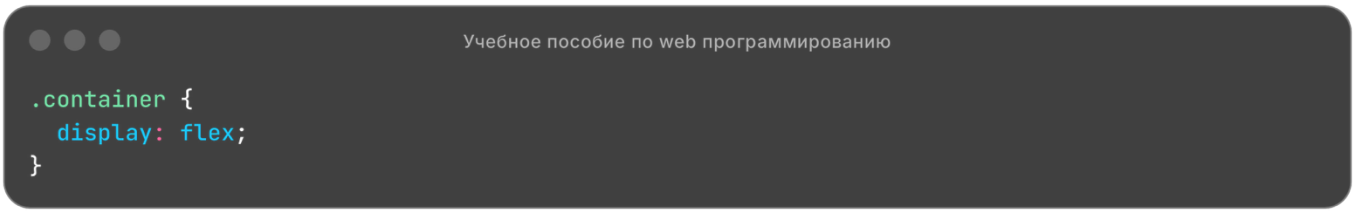
2.5. Flexbox — основной инструмент для макетов

Когда нужно разместить элементы **в строку или столбец**, выровнять их по центру или распределить пространство между ними, на помощь приходит **Flexbox** *(Flexible Box Layout)*.

Flexbox работает с **одним измерением**: либо по горизонтали (строка), либо по вертикали (столбец). Это делает его идеальным для навигации, карточек, форм и других компонентов.

Как включить Flexbox?

Нужно задать контейнеру свойство display: flex:



Все **прямые потомки** этого контейнера становятся **flex-элементами** и подчиняются правилам Flexbox.

Основные свойства контейнера

1. flex-direction — направление главной оси:
    - row (по умолчанию) — слева направо
    - row-reverse — справа налево
    - column — сверху вниз
    - column-reverse — снизу вверх
2. justify-content — выравнивание по **главной оси**:
    - flex-start — в начало (по умолчанию)
    - center — по центру
    - flex-end — в конец
    - space-between — равномерно, первый — у начала, последний — у конца
    - space-around — равномерно, с равными отступами вокруг
3. align-items — выравнивание по **поперечной оси**:
    - stretch (по умолчанию) — растягивает по высоте контейнера
    - center — по центру
    - flex-start / flex-end — в начало / конец поперечной оси

Свойство gap работает как margin, но только между flex-элементами — без риска схлопывания.

Поддержка: gap в Flexbox поддерживается во всех современных браузерах (включая последние версии на 2025 г.).

Когда использовать Flexbox?

- Навигация
- Кнопки в строку
- Карточки товаров
- Формы с полями в ряд
- Центрирование любого блока

Не стоит использовать Flexbox для сложных двумерных сеток (например, таблицы с ячейками в несколько строк и столбцов) — для этого существует **CSS Grid**, но в рамках этого пособия мы ограничимся Flexbox, так как он проще и покрывает 90 % повседневных задач.

2.6. Адаптивность: один пример с медиазапросом (max-width: 768px)

Сайт должен хорошо выглядеть **и на компьютере, и на смартфоне**. Для этого используется **адаптивный дизайн** — изменение стилей в зависимости от ширины экрана.

Основной инструмент — **медиазапросы** (@media).

Как работает медиазапрос?

Учебное пособие по web программированию

```
/* Стили по умолчанию — для больших экранов */
.container {
  display: flex;
  width: 100%;
}

.item {
  flex: 1;
  padding: 1rem;
}

/* Адаптация для экранов ≤ 768px (планшеты и телефоны) */
@media (max-width: 768px) {
  .container {
    flex-direction: column;
  }
}
```

- max-width: 768px означает: «применить эти стили, если ширина экрана **768 пикселей или меньше**».
- Внутри блока @media переопределяются только нужные правила.

Почему именно 768px?

- 768px — классическая граница между **планшетами в портретной ориентации** и **десктопами**.

Другие популярные точки:

- 480px — маленькие телефоны
- 1024px — большие планшеты

Но для простого сайта достаточно **одного медиазапроса**.

Советы по адаптивности

1. **Сначала проектируйте для мобильных** (mobile-first), затем улучшайте для больших экранов — так проще.
2. Используйте **относительные единицы**: rem, %, vw, а не только px.
3. Всегда проверяйте отображение в **инструментах разработчика браузера** *(вкладка «Device Toolbar» или Ctrl+Shift+M в Firefox)*.

3. JavaScript — интерактивность

3.1. Подключение скрипта и базовый синтаксис

JavaScript (JS) — это язык программирования, который выполняется в браузере и позволяет делать веб-страницы **интерактивными**: реагировать на действия пользователя, изменять содержимое без перезагрузки, отправлять данные на сервер и многое другое.

Подключение скрипта

Есть два способа добавить JavaScript на страницу:

1. Внешний файл *(рекомендуется)*:

```
Учебное пособие по web программированию

<script src="script.js"></script>
```

Размещайте этот тег перед закрывающим </body>, чтобы HTML успел загрузиться:

```
Учебное пособие по web программированию

<body>
  <!-- весь HTML -->
  <script src="script.js"></script>
</body>
```

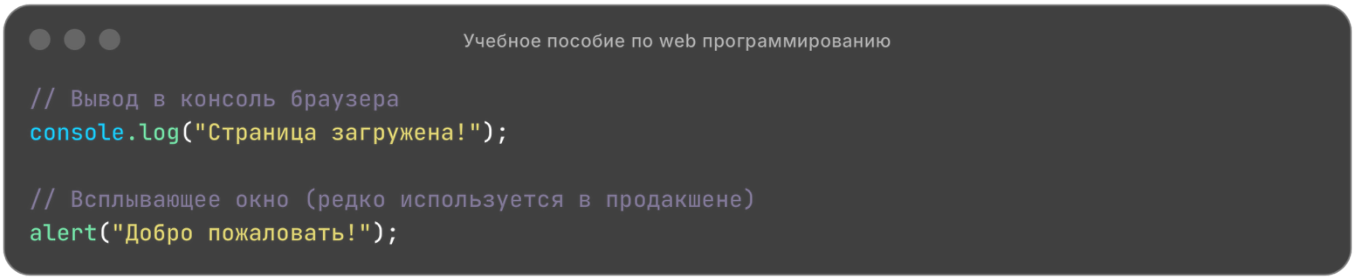
2. Встроенный скрипт (для коротких примеров):

```
Учебное пособие по web программированию

<script>
  console.log("Привет из JavaScript!");
</script>
```

Базовый синтаксис

- Каждая инструкция обычно заканчивается точкой с запятой (;), хотя не всегда обязательно.
- Комментарии:
    - Однострочные: // это комментарий
    - Многострочные: /* это тоже комментарий */

Пример простого скрипта:



Откройте **инструменты разработчика** в браузере *(F12 → вкладка Console)*, чтобы увидеть результат console.log.

3.2. Переменные, условия (if), циклы (for)

Переменные

Переменная — это именованное хранилище для данных. В современном JavaScript используются ключевые слова let и const:

- let — для значений, которые **могут меняться**.
- const — для значений, которые **не меняются** после присваивания (но объекты внутри могут изменяться).

```
Учебное пособие по web программированию

let userName = "Анна";
const PI = 3.14159;

userName = "Иван"; // допустимо
// PI = 3.14;      // ошибка: нельзя перезаписать const
```

Типы данных, с которыми вы будете работать чаще всего:

- string — строка: "Привет"
- number — число: 42, 3.14
- boolean — логическое значение: true / false

Условия (if)

Позволяют выполнять код только при определённом условии:

```
Учебное пособие по web программированию

let age = 18;

if (age >= 18) {
  console.log("Совершеннолетний");
} else {
  console.log("Несовершеннолетний");
}
```

Можно добавлять дополнительные проверки с else if:

```
Учебное пособие по web программированию

let score = 85;

if (score >= 90) {
  console.log("Отлично");
} else if (score >= 70) {
  console.log("Хорошо");
} else {
  console.log("Нужно подтянуть");
}
```

Циклы (for**)**

Используются для повторения действий. Цикл for состоит из трёх частей: начальное значение, условие продолжения, изменение счётчика.

```
Учебное пособие по web программированию

// Вывести числа от 1 до 5
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
```

Пример: проверка каждого элемента массива (массивы — список значений в квадратных скобках):

```
Учебное пособие по web программированию

let colors = ["красный", "зелёный", "синий"];

for (let i = 0; i < colors.length; i++) {
  console.log("Цвет: " + colors[i]);
}
```

> *colors.length — свойство, возвращающее количество элементов в массиве.*

Эти три конструкции — основа почти любой логики в JavaScript. Далее вы увидите, как они применяются при работе с элементами страницы.

3.3. Функции: объявление и вызов

Функция — это **повторно используемый блок кода**, который выполняет определённую задачу. Вместо того чтобы писать одну и ту же логику много раз, вы создаёте функцию и вызываете её при необходимости.

Объявление функции

Самый простой способ — **function declaration**:

```
Учебное пособие по web программированию

function greet(name) {
  console.log("Привет, " + name + "!");
}
```

- greet — имя функции.
- name — **параметр** (входное значение).
- Внутри фигурных скобок — тело функции.

Вызов функции

Чтобы выполнить код внутри функции, её нужно **вызвать**:

```
Учебное пособие по web программированию

greet("Мария");   // Выведет: Привет, Мария!
greet("Андрей");  // Выведет: Привет, Андрей!
```

Возврат значения

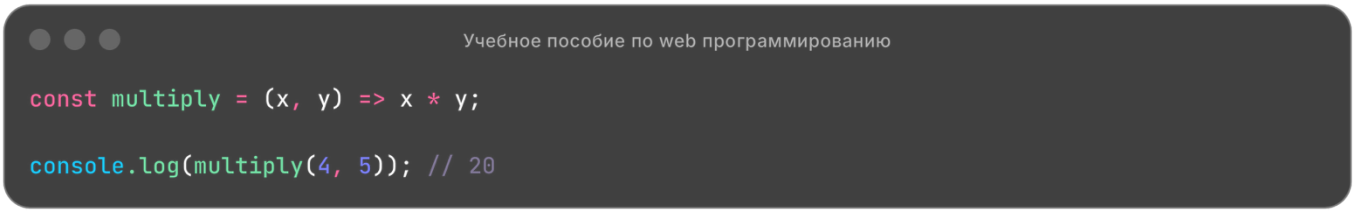
Функция может **возвращать результат** с помощью return:

```
Учебное пособие по web программированию

function add(a, b) {
  return a + b;
}

let sum = add(5, 3); // sum = 8
console.log("Сумма: " + sum);
```

После return выполнение функции **останавливается**.

Функции без имени (стрелочные функции)

Для коротких действий часто используют **стрелочную запись**:

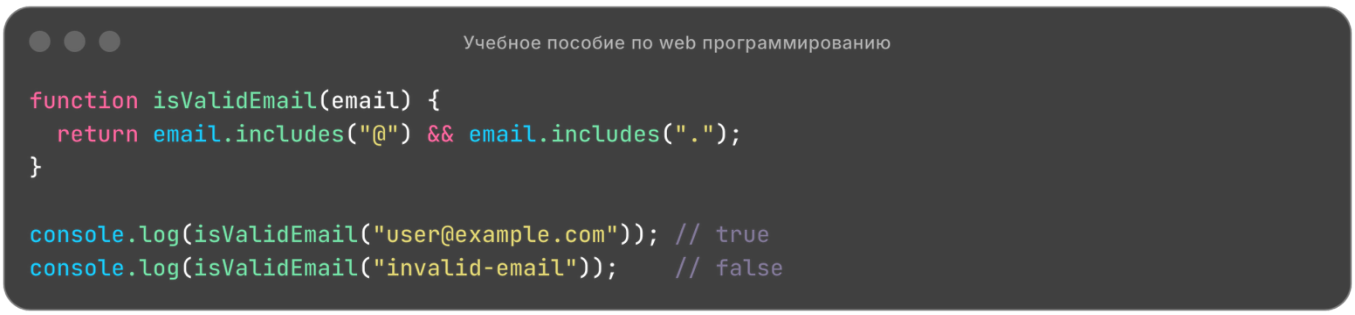


Это особенно удобно при работе с событиями и асинхронным кодом (например, fetch), о чём позже.

Почему функции важны?

- Делают код **понятнее** и **структурированнее**.
- Позволяют **избежать дублирования**.
- Облегчают **тестирование и отладку**.

Пример: валидация email



Теперь эту проверку можно использовать в любом месте, не повторяя логику.

3.4. Работа с DOM: querySelector(), textContent, addEventListener()

DOM *(Document Object Model)* — это программное представление HTML-документа. С помощью JavaScript вы можете **читать**, **изменять** и **реагировать** на элементы страницы через DOM.

Получение элемента: querySelector()

Метод document.querySelector() находит **первый элемент**, соответствующий CSS-селектору:

```
Учебное пособие по web программированию

// Получить заголовок h1
let header = document.querySelector("h1");

// Получить элемент по классу
let button = document.querySelector(".submit-btn");

// Получить элемент по id (через #)
let form = document.querySelector("#guest-form");
```

Если элемент не найден, вернётся null.

> *Для получения всех подходящих элементов используйте querySelectorAll(), но для базовых задач чаще достаточно querySelector().*

Изменение содержимого: textContent **и** innerHTML

- textContent — работает только с **текстом**, безопасен от XSS:

```
Учебное пособие по web программированию

let message = document.querySelector("#message");
message.textContent = "Новое сообщение!";
```

- innerHTML — позволяет вставлять **HTML**, но требует осторожности (опасно, если данные от пользователя):

```
Учебное пособие по web программированию

message.innerHTML = "<strong>Важно!</strong> Прочтите внимательно.";
```

В учебных целях и при отсутствии пользовательского ввода можно использовать innerHTML, но **в реальных проектах** с данными от пользователя — только textContent или экранирование.

Реакция на действия: addEventListener()

Чтобы выполнить код при клике, отправке формы и других событиях, используйте addEventListener:

```
Учебное пособие по web программированию

let btn = document.querySelector("#toggle-btn");

btn.addEventListener("click", function() {
  console.log("Кнопка нажата!");
});
```

Второй аргумент — функция-обработчик (callback). Её можно объявить отдельно:

```
Учебное пособие по web программированию

function handleClick() {
  alert("Привет из обработчика!");
}

document.querySelector("#my-button").addEventListener("click", handleClick);
```

3.5. Пример: скрыть/показать блок, валидация поля формы

Теперь объединим знания из предыдущих разделов, чтобы создать два практических примера:

1. Переключение видимости блока.
2. Простая валидация формы перед отправкой.

Пример 1: Скрыть/показать блок с помощью класса

Использование CSS-классов вместо прямого изменения style.display делает код чище и гибче.

HTML:

Учебное пособие по web программированию

```html
<button id="toggle-info">Подробнее</button>
<div id="extra-info" class="hidden">
  <p>Это дополнительная информация, скрытая по умолчанию.</p>
</div>
```

CSS:

Учебное пособие по web программированию

```css
.hidden {
  display: none;
}
```

JavaScript:

Учебное пособие по web программированию

```javascript
document.querySelector("#toggle-info").addEventListener("click", function() {
  document.querySelector("#extra-info").classList.toggle("hidden");
});
```

> *Метод classList.toggle("hidden") автоматически **добавляет** класс, если его нет, и **удаляет**, если он есть.*

Пример 2: Валидация формы

Предположим, у нас есть форма с полем имени и email. Мы хотим:

- Проверить, что оба поля заполнены.
- Убедиться, что email содержит @.

HTML:

```
Учебное пособие по web программированию

<form id="contact-form">
  <input type="text" id="name" placeholder="Имя" required>
  <input type="email" id="email" placeholder="Email" required>
  <button type="submit">Отправить</button>
  <p id="error-message" class="error hidden"></p>
</form>
```

CSS (для ошибок):

```
Учебное пособие по web программированию

.error {
  color: red;
  margin-top: 8px;
}
.hidden {
  display: none;
}
```

JavaScript:

```
Учебное пособие по web программированию

document.querySelector("#contact-form").addEventListener("submit", function(event) {
  const name = document.querySelector("#name").value.trim();
  const email = document.querySelector("#email").value.trim();
  const errorEl = document.querySelector("#error-message");
  // Сброс предыдущих ошибок
  errorEl.textContent = "";
  errorEl.classList.add("hidden");
  let isValid = true;
  if (name === "") {
    errorEl.textContent = "Пожалуйста, введите имя.";
    isValid = false;
  } else if (email === "") {
    errorEl.textContent = "Пожалуйста, введите email.";
    isValid = false;
  } else if (!email.includes("@")) {
    errorEl.textContent = "Email должен содержать символ @.";
    isValid = false;
  }
  if (!isValid) {
    errorEl.classList.remove("hidden");
    event.preventDefault(); // отменить отправку формы
  }
  // Если всё в порядке — форма отправится обычным способом
});
```

- event.preventDefault() — останавливает стандартное поведение браузера (в данном случае — отправку формы).
- trim() удаляет пробелы по краям, чтобы избежать «ложно заполненных» полей.

Этот подход работает **без перезагрузки страницы** и даёт мгновенную обратную связь пользователю.

Эти два примера демонстрируют основные сценарии взаимодействия с пользователем: динамическое изменение интерфейса и проверка ввода.

3.6. fetch() — один пример отправки данных на PHP-скрипт

До сих пор форма отправлялась стандартным способом — с перезагрузкой страницы. Но с помощью JavaScript можно отправить данные **асинхронно**, без перезагрузки. Для этого используется встроенный метод fetch().

Как работает fetch()**?**

fetch() позволяет делать HTTP-запросы к серверу. Чтобы отправить данные формы на PHP-скрипт:

1. Собрать данные из полей.
2. Преобразовать их в формат, понятный серверу (обычно FormData или JSON).
3. Вызвать fetch() с нужным URL и параметрами.
4. Обработать ответ.

Пример: отправка комментария в гостевую книгу без перезагрузки

HTML:

Учебное пособие по web программированию

```html
<form id="guest-form">
  <input type="text" id="author" placeholder="Ваше имя" required>
  <textarea id="text" placeholder="Ваше сообщение" required></textarea>
  <button type="submit">Отправить</button>
</form>
<div id="messages"></div>
```

PHP-скрипт save_comment.php **(предварительный вариант — подробнее в 4 части)**

Учебное пособие по web программированию

```php
<?php
header("Content-Type: text/plain; charset=utf-8");

$author = $_POST['author'] ?? '';
$text = $_POST['text'] ?? '';

if ($author && $text) {
  // Здесь позже будет запись в базу
  echo "OK";
} else {
  http_response_code(400);
  echo "Ошибка: пустые данные";
}
?>
```

JavaScript:

Учебное пособие по web программированию

```
document.querySelector("#guest-form").addEventListener("submit", function(event) {
  event.preventDefault(); // отменить стандартную отправку

  const author = document.querySelector("#author").value.trim();
  const text = document.querySelector("#text").value.trim();

  if (!author || !text) {
    alert("Заполните все поля");
    return;
  }

  // Подготовка данных
  const formData = new FormData();
  formData.append("author", author);
  formData.append("text", text);

  // Отправка на сервер
  fetch("save_comment.php", {
    method: "POST",
    body: formData
  })
  .then(response => response.text())
  .then(result => {
    // Очистить форму
    document.querySelector("#guest-form").reset();

    // Показать сообщение (можно добавить в список)
    const messagesDiv = document.querySelector("#messages");
    messagesDiv.innerHTML += `<p><strong>${author}:</strong> ${text}</p>`;
  })
  .catch(error => {
    console.error("Ошибка:", error);
    alert("Не удалось отправить сообщение.");
  });
});
```

На данном этапе скрипт только принимает данные и отвечает — без сохранения. В части 2 мы подключим MySQL и сделаем это по-настоящему.

Почему fetch() **лучше стандартной отправки?**

- Нет перезагрузки → лучше пользовательский опыт.
- Можно обновить только нужную часть страницы.
- Легко обрабатывать ошибки и показывать уведомления.

> 💡 *Важно: fetch() не отправляет куки по умолчанию в кросс-доменных запросах, но в рамках одного домена (ваш локальный сервер) это не проблема.*

4. Бэкенд: сервер и база данных

4.1. Что такое PHP и как он работает на сервере

PHP (Hypertext Preprocessor) — это **серверный язык программирования**, предназначенный специально для веб-разработки. В отличие от JavaScript, который выполняется в браузере пользователя, **PHP работает на сервере**.

Как это происходит?

1. Пользователь открывает страницу, например, http://localhost/guestbook/index.php.
2. Браузер отправляет **запрос** на сервер.
3. Сервер (например, Apache в составе XAMPP) видит, что запрашивается файл с расширением .php.
4. **PHP-интерпретатор** читает файл, **выполняет весь PHP-код** внутри него и **формирует чистый HTML**.
5. Сервер отправляет **результат (только HTML)** обратно в браузер.
6. Браузер отображает готовую страницу — он **никогда не видит исходный PHP-код**.

Файл hello.php**:**

Учебное пособие по web программированию

```
<!DOCTYPE html>
<html>
<body>
  <h1>Привет, <?php echo "Мир!"; ?></h1>
</body>
</html>
```

Что получает браузер:

Учебное пособие по web программированию

```
<!DOCTYPE html>
<html>
<body>
  <h1>Привет, Мир!</h1>
</body>
</html>
```

🔒 *Безопасность: поскольку PHP выполняется на сервере, вы можете хранить в нём конфиденциальные данные (пароли, ключи), и они никогда не попадут к пользователю.*

Почему PHP популярен для обучения?

- Простой синтаксис, похожий на C/JavaScript.
- Встроенная поддержка работы с HTML.
- Широкая поддержка хостингов и локальных серверов (XAMPP, MAMP).
- Отлично сочетается с MySQL.

Где писать PHP?

- Установите **XAMPP** (бесплатный локальный серверный пакет).
- Поместите файлы в папку htdocs внутри XAMPP.
- Запустите Apache и MySQL через панель управления XAMPP.
- Откройте в браузере: http://localhost/имя_вашего_файла.php.

> 💡 *Напоминание: PHP не работает при открытии файла через file:// (двойной клик по файлу). Он требует HTTP-сервера.*

4.2. Синтаксис: переменные, строки, условия, циклы, функции

PHP имеет синтаксис, похожий на C и JavaScript, но с собственными особенностями. Ниже — краткий обзор базовых конструкций, необходимых для начала.

Переменные

Все переменные в PHP начинаются со знака $:

```
Учебное пособие по web программированию

$name = "Анна";
$age = 25;
$isActive = true;
```

- Тип переменной определяется автоматически.
- Имена чувствительны к регистру: $Name ≠ $name.

Строки

Строки можно задавать в **одинарных** или **двойных** кавычках:

```
Учебное пособие по web программированию

$greeting = 'Привет';
$message = "Привет, $name!"; // внутри двойных кавычек можно вставлять переменные
$alt = 'Привет, ' . $name . '!'; // конкатенация через точку
```

> ⚠ *В одинарных кавычках переменные не интерполируются: '$name' выведет буквально $name.*

Условия (if, else, elseif)

Синтаксис почти идентичен JavaScript:

```
Учебное пособие по web программированию

$score = 85;

if ($score >= 90) {
    echo "Отлично";
} elseif ($score >= 70) {
    echo "Хорошо";
} else {
    echo "Нужно подтянуть";
}
```

- Блоки кода ограничиваются фигурными скобками {}.
- Для сравнения используются операторы: ==, ===, !=, <, >, <=, >=.

Циклы

Часто используется цикл for или foreach (особенно с массивами):

```php
// Цикл for
for ($i = 1; $i <= 5; $i++) {
    echo "Число: $i<br>";
}

// Массив
$colors = ["красный", "зелёный", "синий"];

// Цикл foreach
foreach ($colors as $color) {
    echo "Цвет: $color<br>";
}
```

Функции

Объявление функции:

```php
function add($a, $b) {
    return $a + $b;
}

$result = add(3, 4); // $result = 7
echo $result;
```

- Аргументы передаются в скобках.
- Возврат значения — через return.

PHP также предоставляет множество **встроенных функций**, например:

- strlen($str) — длина строки
- htmlspecialchars($str) — защита от XSS (см. раздел 4.4)
- date("Y-m-d") — текущая дата

Комментарии

- Однострочные: // комментарий или # комментарий
- Многострочные: /* комментарий */

Эти базовые конструкции позволят вам писать логику обработки форм, генерации страниц и взаимодействия с базой данных.

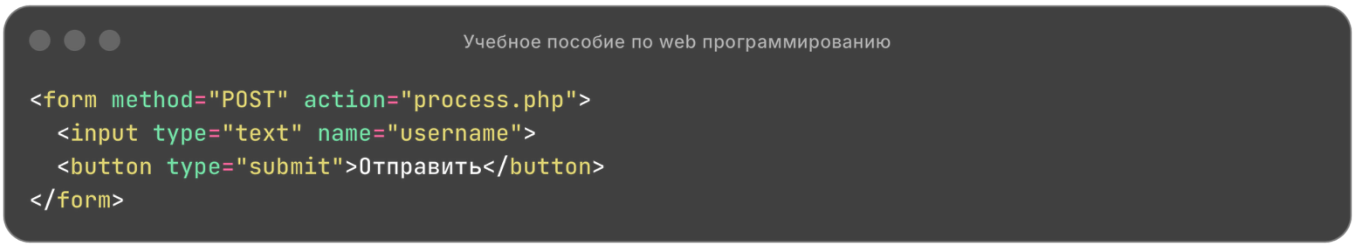
4.3. Работа с формами: $_POST, $_GET

Когда пользователь отправляет HTML-форму, данные попадают в PHP через **суперглобальные массивы**: $_POSTили $_GET, в зависимости от атрибута method формы.

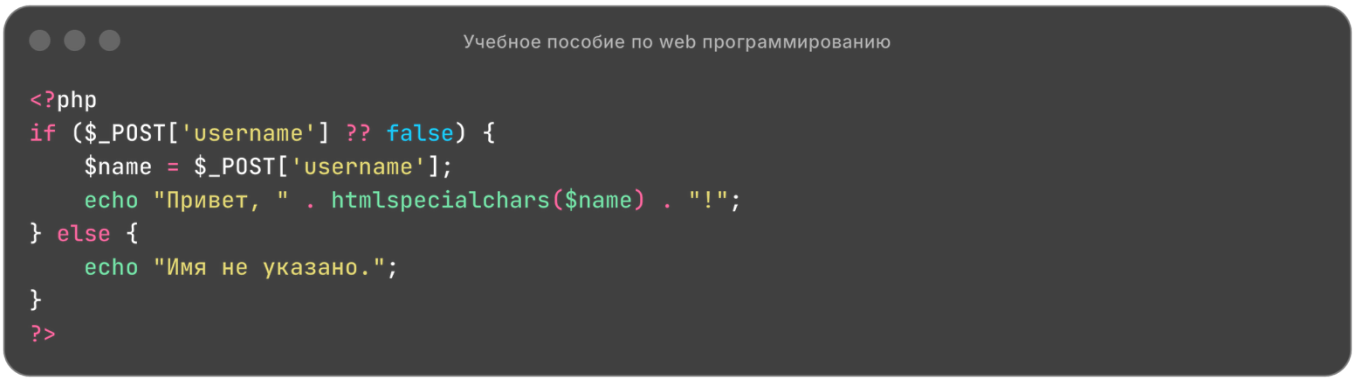
$_POST **— для конфиденциальных или больших данных**

Используется, когда method="POST" в форме. Данные **не видны в URL** и имеют больший лимит размера.

HTML:



PHP (process.php**):**



- $_POST['username'] — значение поля с name="username".
- Оператор ?? (null coalescing) возвращает false, если ключ отсутствует — это предотвращает ошибку «Undefined index».

$_GET **— для фильтров, поиска, навигации**

Используется, когда method="GET" (или не указан, так как GET — значение по умолчанию). Данные **добавляются в URL**.

Пример URL после отправки:

http://localhost/search.php?query=книги

HTML:

```
Учебное пособие по web программированию

<form method="GET" action="search.php">
  <input type="text" name="query">
  <button type="submit">Найти</button>
</form>
```

PHP (search.php**):**

```
Учебное пособие по web программированию

<?php
$query = $_GET['query'] ?? '';
if ($query) {
    echo "Вы ищете: " . htmlspecialchars($query);
} else {
    echo "Введите запрос для поиска.";
}
?>
```

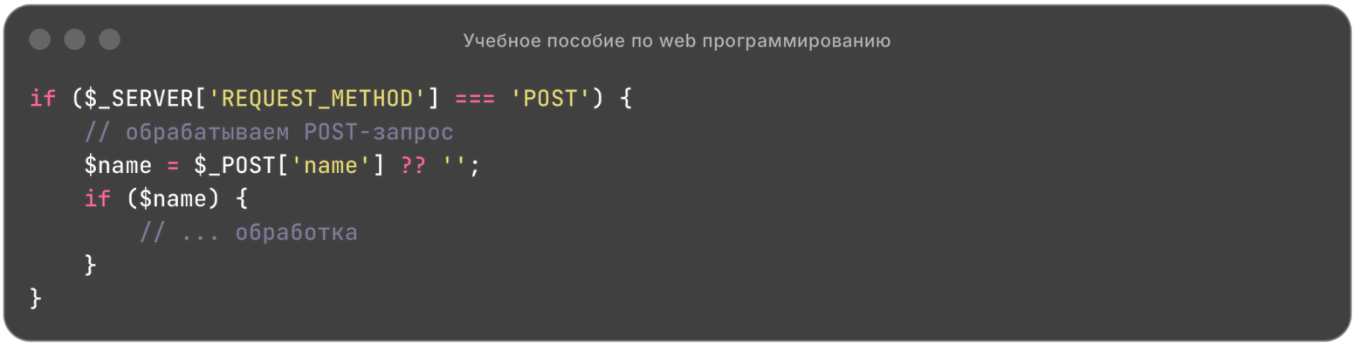
Когда использовать POST**, а когда** GET**?**

Критерий	GET	POST
Данные в URL	Да	Нет
Безопасность	Не для паролей/личных данных	Подходит для конфиденциальных данных
История браузера	Можно перезагрузить/поделиться ссылкой	Нельзя (предупреждение при повторной отправке)
Ограничение длины	Около 2000 символов	Практически нет (настраивается на сервере)
Использование	Поиск, фильтрация, пагинация	Отправка форм, загрузка файлов, изменения данных

> 💡 *Правило: если действие меняет данные (добавляет комментарий, удаляет запись), используйте POST.*

Проверка наличия данных

Всегда проверяйте, были ли отправлены данные, чтобы избежать ошибок:



$_SERVER['REQUEST_METHOD'] показывает, каким методом пришёл запрос — это надёжнее, чем проверять только наличие ключей.

Это основа взаимодействия между фронтендом и бэкендом. В следующем разделе мы научимся **защищать эти данные** от вредоносного ввода.

4.4. Безопасность: htmlspecialchars() и фильтрация ввода

Когда вы принимаете данные от пользователя (через $_POST или $_GET), **никогда не доверяйте им напрямую**. Злоумышленник может ввести вредоносный HTML или JavaScript, который выполнится у других пользователей — это называется **XSS (Cross-Site Scripting)**.

Пример уязвимости

Предположим, вы выводите имя без обработки:

```
Учебное пособие по web программированию

// ОПАСНО!
echo "Привет, " . $_POST['name'];
```

Если пользователь введёт: <script>alert(“Взлом”);</script>

То браузер других пользователей выполнит этот скрипт при открытии страницы.

Защита: htmlspecialchars()

Функция htmlspecialchars() преобразует специальные HTML-символы в «безопасные» сущности:

```
Учебное пособие по web программированию

$name = $_POST['name'] ?? '';
$safe_name = htmlspecialchars($name, ENT_QUOTES, 'UTF-8');
echo "Привет, $safe_name!";
```

Преобразования:

- < → <
- > → >
- " → "
- ' → ' (при ENT_QUOTES)

Теперь даже вредоносный ввод отобразится как **обычный текст**, а не как исполняемый код.

> ✅ *Всегда используйте htmlspecialchars() при выводе пользовательских данных в HTML.*

Дополнительная фильтрация

Помимо экранирования при выводе, можно **фильтровать и валидировать** данные при приёме.

PHP предоставляет встроенные функции:

- filter_var() — для проверки email, URL и др.
- trim() — удаление пробелов по краям.

Пример: проверка email

```
Учебное пособие по web программированию

$email = trim($_POST['email'] ?? '');

if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    // email валиден
    $safe_email = htmlspecialchars($email, ENT_QUOTES, 'UTF-8');
    echo "Email: $safe_email";
} else {
    echo "Некорректный email.";
}
```

Пример: ограничение длины имени

```
Учебное пособие по web программированию

$name = trim($_POST['name'] ?? '');
if (strlen($name) < 2) {
    echo "Имя должно быть не короче 2 символов.";
} elseif (strlen($name) > 50) {
    echo "Имя слишком длинное.";
} else {
    // OK
    echo "Привет, " . htmlspecialchars($name) . "!";
}
```

Общие правила безопасности для форм

1. **Экранируйте при выводе**: всегда htmlspecialchars() при вставке данных в HTML.
2. **Валидируйте при приёме**: проверяйте тип, длину, формат.
3. **Не храните и не выводите «как есть»** то, что пришло от пользователя.
4. **Используйте** ENT_QUOTES **и явно указывайте кодировку** (UTF-8) в htmlspecialchars().

Эти меры защитят ваше приложение от базовых атак. В следующем разделе мы покажем, как применить их на практике — в примере приёма и вывода данных из формы.

4.5. Пример: приём данных из формы и вывод на странице

Соберём всё вместе: создадим простую страницу с формой, которая принимает имя и сообщение, проверяет их и отображает на той же странице. Пока без базы данных — данные будут «жить» только до перезагрузки, но логика обработки уже готова к интеграции с MySQL.

Шаг 1: HTML-форма и контейнер для сообщений

Файл: index.php

Учебное пособие по web программированию

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="utf-8">
  <title>Гостевая книга (без БД)</title>
  <style>
    .error { color: red; }
    .message { margin: 10px 0; padding: 8px; background: #f0f0f0; }
  </style>
</head>
<body>
  <h1>Гостевая книга</h1>

  <form method="POST">
    <div>
      <label>Имя:</label><br>
      <input type="text" name="author" value="<?php echo isset($_POST['author']) ?
htmlspecialchars($_POST['author']) : ''; ?>">
    </div>
    <div style="margin: 10px 0;">
      <label>Сообщение:</label><br>
      <textarea name="text" rows="4"><?php echo isset($_POST['text']) ?
htmlspecialchars($_POST['text']) : ''; ?></textarea>
    </div>
    <button type="submit">Оставить запись</button>
  </form>

  <?php if (!empty($error)): ?>
    <p class="error"><?php echo $error; ?></p>
  <?php endif; ?>

  <?php if (!empty($success_message)): ?>
    <div class="message">
      <strong><?php echo $safe_author; ?>:</strong>
      <?php echo $safe_text; ?>
    </div>
  <?php endif; ?>
</body>
</html>
```

Обратите внимание: значение полей формы сохраняется после отправки (благодаря value и textarea содержимому). Это улучшает UX — пользователь не теряет ввод при ошибке.

Шаг 2: PHP-логика обработки (в том же файле, в начале)

Добавьте **в самое начало** файла index.php, **до любого HTML**:

Как это работает?

1. При открытии страницы — ничего не отправлено → форма пустая.
2. При отправке:
    - Проверяются поля.
    - Если ошибка — она сохраняется в $error и выводится.
    - Если всё в порядке — данные экранируются и сохраняются в $safe_author / $safe_text.
3. Ни в коем случае **не выводится** $_POST напрямую — только через htmlspecialchars().

Безопасность на месте?

✅ Да:

- Все пользовательские данные экранируются перед выводом.
- Проверяется длина и наличие содержимого.
- Форма устойчива к XSS (попробуйте ввести <script> — он отобразится как текст).

Этот пример — **готовая основа** для будущей гостевой книги с базой данных. В следующих разделах мы заменим временное хранение на запись в MySQL.

5. MySQL и работа с базой данных

5.1. Что такое база данных и зачем она нужна

огда вы работаете с веб-приложением, часто требуется **хранить данные между запросами**: комментарии пользователей, товары в каталоге, профили и т.д. Обычные PHP-переменные **не подходят** — они живут только во время выполнения одного скрипта и исчезают после завершения.

База данных (БД) — это организованное хранилище данных, которое:

- Сохраняет информацию **постоянно**.
- Позволяет **быстро искать, добавлять, изменять и удалять** записи.
- Обеспечивает **целостность и безопасность** данных.

Почему именно MySQL?

- Бесплатная и открытая (в редакции Community).
- Широко используется в веб-разработке.
- Отлично интегрируется с PHP.
- Поддерживается большинством хостингов и локальных серверов (XAMPP, MAMP).

В нашем учебном проекте MySQL будет хранить **записи из гостевой книги**: автора, сообщение и дату.

Структура базы данных

База данных состоит из:

- **Базы (database)** — контейнер, например, guestbook_db.
- **Таблиц (tables)** — внутри базы, например, таблица comments.
- **Строк (records/rows)** — отдельные записи (один комментарий).
- **Столбцов (fields/columns)** — атрибуты записи: id, author, text, created_at.

Пример таблицы comments:

ID	AUTHOR	TEXT	CREATED_AT
1	Иван	Отличный Сайт!	2025-11-17 10:30:00
2	Мария	Спасибо за книгу!	2025-11-17 11:15:00

Теперь данные будут сохраняться даже после перезапуска сервера.

В следующем разделе мы изучим язык управления базой данных — **SQL**.

5.2. Основы SQL: CREATE TABLE, INSERT, SELECT, UPDATE, DELETE

SQL (Structured Query Language) — это язык для управления реляционными базами данных, включая MySQL. Ниже — пять ключевых команд, которые покрывают 95 % повседневных задач.

1. CREATE TABLE — создание таблицы

Создадим таблицу для гостевой книги:

```
Учебное пособие по web программированию

CREATE TABLE comments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    author VARCHAR(50) NOT NULL,
    text TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Пояснения:

- id — уникальный идентификатор записи. AUTO_INCREMENT — автоматически увеличивается при каждой новой записи. PRIMARY KEY — делает поле уникальным и ускоряет поиск.
- author — строка до 50 символов, обязательная (NOT NULL).
- text — текст произвольной длины (до 65 КБ в TEXT).
- created_at — дата и время создания. DEFAULT CURRENT_TIMESTAMP — ставится автоматически.

> 💡 *Выполните этот запрос в phpMyAdmin (входит в XAMPP): откройте базу → вкладка «SQL» → вставьте код → нажмите «Выполнить».*

2. INSERT — добавление записи

Добавим первый комментарий:

```
Учебное пособие по web программированию

INSERT INTO comments (author, text)
VALUES ('Анна', 'Спасибо за полезный сайт!');
```

- id и created_at заполнятся автоматически.
- Порядок полей в скобках должен соответствовать порядку в VALUES.

3. SELECT — выборка данных

Получить все комментарии:

```
Учебное пособие по web программированию

SELECT * FROM comments;
```

Получить только последние 5, отсортированные по дате (новые сверху):

```
Учебное пособие по web программированию

SELECT author, text, created_at
FROM comments
ORDER BY created_at DESC
LIMIT 5;
```

- * означает «все столбцы».
- ORDER BY ... DESC — сортировка по убыванию.
- LIMIT — ограничение количества записей.

4. UPDATE — изменение записи

Изменим текст комментария с id = 1:

```
Учебное пособие по web программированию

UPDATE comments
SET text = 'Обновлённое сообщение!'
WHERE id = 1;
```

> ⚠️ *Всегда используйте WHERE, иначе обновятся все записи!*

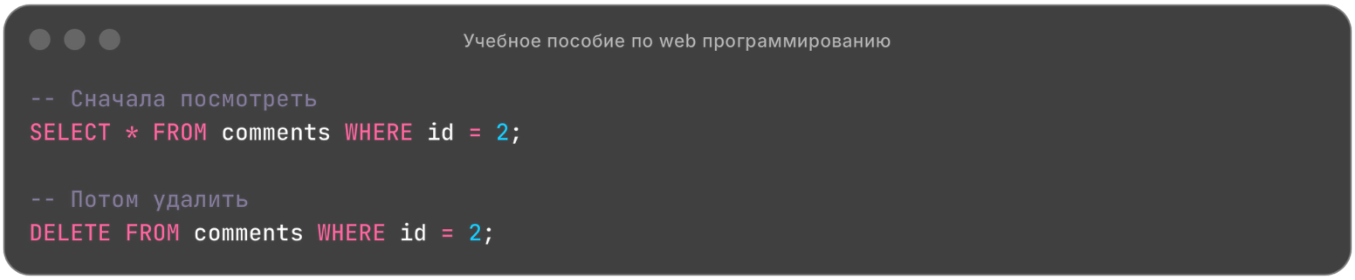
5. DELETE — удаление записи

```
Учебное пособие по web программированию

DELETE FROM comments
WHERE id = 2;
```

> ⚠️ *Без WHERE будет удалена вся таблица!*

Практический совет: проверяйте запросы

Перед выполнением UPDATE или DELETE сначала проверьте, какие строки затронуты:



Эти пять команд — основа любого взаимодействия с базой данных. В следующем разделе мы подключим PHP к MySQL и выполним эти запросы **из кода**.

5.3. Подключение PHP к MySQL через mysqli

Чтобы PHP мог взаимодействовать с MySQL, нужно установить **соединение** с базой данных. В PHP для этого есть несколько расширений; мы будем использовать mysqli (MySQL Improved) — современное, встроенное и подходящее для обучения.

Шаг 1: Создайте базу данных

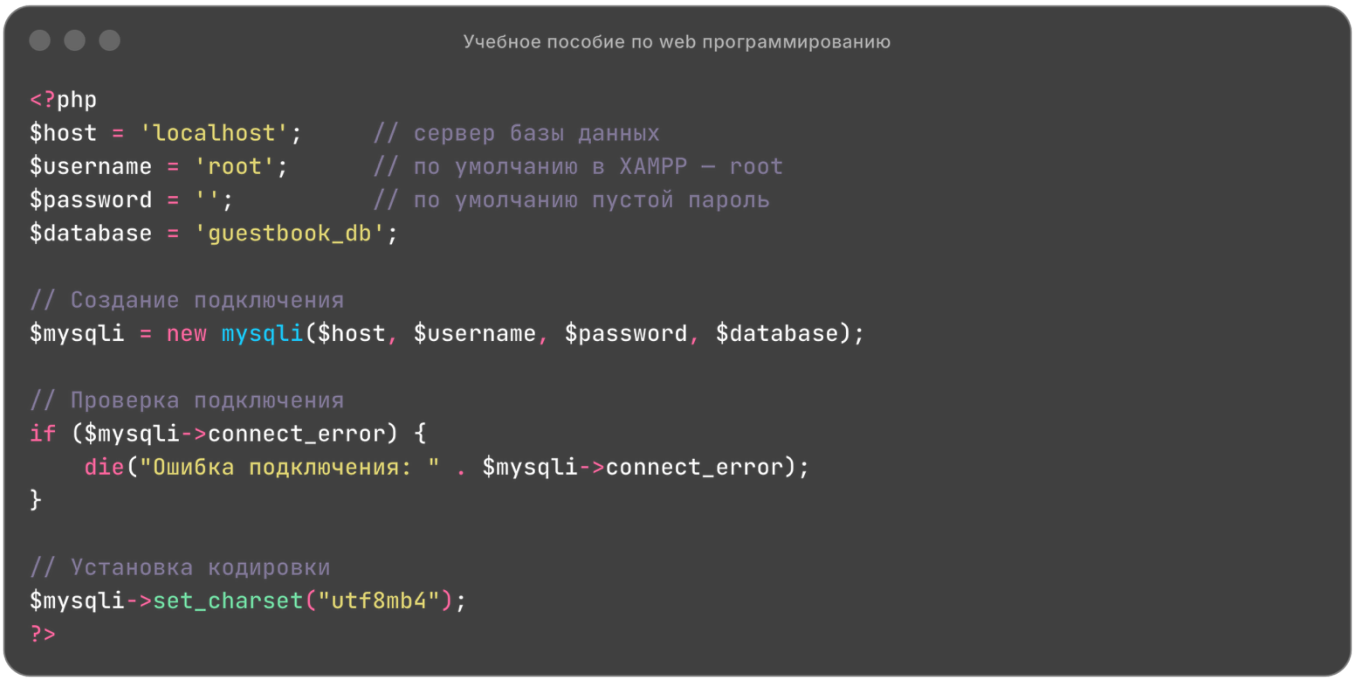
Если ещё не сделали:

1. Откройте **phpMyAdmin** (http://localhost/phpmyadmin).
2. Нажмите «Новая база данных».
3. Введите имя, например: guestbook_db.
4. Выберите кодировку **utf8mb4_unicode_ci** (поддерживает эмодзи и все языки).
5. Нажмите «Создать».

Затем выполните SQL-запрос из раздела 5.2, чтобы создать таблицу comments.

Шаг 2: Подключение из PHP

Создадим файл конфигурации db.php:



> 💡 *В XAMPP учётная запись по умолчанию: логин root, пароль пустой. На боевом хостинге данные будут другими.*

Шаг 3: Пример — вывод всех комментариев

Создадим файл show_comments.php:

Учебное пособие по web программированию

```php
<?php
require_once 'db.php';

// SQL-запрос
$sql = "SELECT author, text, created_at FROM comments ORDER BY created_at DESC";
$result = $mysqli->query($sql);

if ($result && $result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "<div>";
        echo "<strong>" . htmlspecialchars($row['author']) . ":</strong> ";
        echo htmlspecialchars($row['text']) . "<br>";
        echo "<small>" . $row['created_at'] . "</small>";
        echo "</div><hr>";
    }
} else {
    echo "Нет комментариев.";
}

$mysqli->close();
?>
```

- fetch_assoc() — возвращает строку как ассоциативный массив ($row['author']).
- Обязательно используем htmlspecialchars() при выводе данных из БД — они могли быть добавлены кем угодно.

Важно: закрывать соединение

Хотя PHP автоматически закрывает соединение в конце скрипта, явный вызов

$mysqli->close() делает код чище и предсказуемее.

Теперь ваш PHP-скрипт может читать данные из MySQL. В следующем разделе мы научимся **безопасно записывать** данные с помощью подготовленных запросов.

5.4. Подготовленные запросы — защита от SQL-инъекций

Когда вы вставляете пользовательские данные напрямую в SQL-запрос, возникает уязвимость **SQL-инъекция**. Злоумышленник может ввести специальный код, который изменит логику запроса — например, удалить таблицу или получить доступ к чужим данным.

Пример уязвимого кода (НЕ ДЕЛАЙТЕ ТАК!)

Учебное пособие по web программированию

```php
// ОПАСНО!
$author = $_POST['author'];
$text = $_POST['text'];
$sql = "INSERT INTO comments (author, text) VALUES ('$author', '$text')";
$mysqli->query($sql);
```

Если пользователь введёт в поле author:

Учебное пособие по web программированию

```
'); DROP TABLE comments; --
```

Получится запрос:

Учебное пособие по web программированию

```
INSERT INTO comments (author, text) VALUES (''); DROP TABLE comments; --', '...');
```

Это приведёт к **удалению всей таблицы**.

Решение: подготовленные запросы (prepared statements)

Подготовленные запросы **разделяют SQL-логику и данные**. Сначала вы отправляете шаблон запроса с **параметрами-заполнителями** (?), а затем передаёте данные отдельно. MySQL обрабатывает их как **значения, а не как часть кода**.

Пример безопасной вставки:

Учебное пособие по web программированию

```php
$author = $_POST['author'] ?? '';
$text = $_POST['text'] ?? '';

// Шаг 1: подготавливаем запрос
$stmt = $mysqli->prepare("INSERT INTO comments (author, text) VALUES (?, ?)");

// Шаг 2: привязываем параметры
// 'ss' означает: две строки (string, string)
$stmt->bind_param('ss', $author, $text);

// Шаг 3: выполняем
if ($stmt->execute()) {
    echo "Комментарий сохранён.";
} else {
    echo "Ошибка: " . $stmt->error;
}

// Шаг 4: закрываем
$stmt->close();
```

✅ *Даже если $author содержит вредоносный код, он будет сохранён как текст, а не выполнен.*

Типы параметров в bind_param()

Первый аргумент — строка с типами:

- s — string (строка)
- i — integer (целое число)
- d — double (дробное число)
- b — blob (бинарные данные)

Пример с целым числом:

```
Учебное пособие по web программированию

$user_id = 123;
$stmt = $mysqli->prepare("SELECT * FROM comments WHERE id = ?");
$stmt->bind_param('i', $user_id);
$stmt->execute();
```

Выборка с подготовленным запросом

Даже при SELECT с пользовательским вводом (например, поиск по имени) используйте подготовленные запросы:

```
Учебное пособие по web программированию

$search = $_GET['q'] ?? '';
$stmt = $mysqli->prepare("SELECT author, text FROM comments WHERE author LIKE ?");
$search = "%$search%"; // для частичного совпадения
$stmt->bind_param('s', $search);
$stmt->execute();
$result = $stmt->get_result();

while ($row = $result->fetch_assoc()) {
    echo htmlspecialchars($row['author']) . ": " . htmlspecialchars($row['text']) . "<br>";
}
```

Почему это важно?

- **Полная защита** от SQL-инъекций.
- Код становится **чище и читаемее**.
- Подготовленные запросы могут быть **повторно использованы** с разными данными (повышает производительность).

> 🔒 *Правило: всегда используйте подготовленные запросы при работе с данными от пользователя — даже если вы «уверены», что данные безопасны.*

Теперь вы можете безопасно читать и записывать данные в базу. В следующем разделе соберём всё в единый пример: сохранение комментария и вывод списка.

5.5. Пример: сохранение комментария в базу и вывод списка

Соберём все полученные знания в один рабочий пример:

- Форма для отправки комментария.
- Безопасное сохранение в MySQL с помощью подготовленного запроса.
- Вывод всех комментариев на той же странице.

Файл: index.php

Полный код можно посмотреть по ссылке:

https://github.com/ennrl/mybooks/blob/e8d4045b8b4419073a893dd0ad433ff87f9a0aaa/preview-1.php

Как это работает?

1. **Подключение**: через db.php.
2. **Обработка POST**:
    - Проверка полей.
    - Использование подготовленного запроса для безопасной вставки.
    - Обновление $author и $text на пустые строки после успешной отправки, чтобы форма очистилась.
3. **Выборка комментариев**: простой SELECT (без пользовательского ввода → подготовленный запрос не обязателен, но допустим).
4. **Вывод**: все данные экранируются через htmlspecialchars().

Требования к окружению

- Установлен **XAMPP** (или аналог).
- Запущены **Apache** и **MySQL**.
- Создана база guestbook_db с таблицей comments (см. раздел 5.2).
- Файлы index.php и db.php находятся в папке htdocs/guestbook/.

Откройте в браузере: http://localhost/guestbook/

Теперь у вас есть полностью рабочая гостевая книга с безопасным взаимодействием с базой данных.

Заключение и что дальше

Поздравляем! Вы прошли путь от простой HTML-страницы до полноценного веб-приложения с серверной логикой и базой данных. Теперь вы умеете:

- Создавать структуру страницы с помощью **HTML**.
- Оформлять её с помощью **CSS**, включая адаптивность и современные макеты на **Flexbox**.
- Добавлять интерактивность через **JavaScript**: обрабатывать события, манипулировать DOM, отправлять данные без перезагрузки.
- Писать серверный код на **PHP**, обрабатывать формы и работать с пользовательским вводом.
- Хранить данные надёжно в **MySQL**, используя безопасные **подготовленные запросы**.

Это фундамент, на котором строятся миллионы сайтов. Но развитие на этом не заканчивается.

Как развиваться дальше?

- **Git и GitHub** — научитесь контролировать версии кода и делиться проектами.
- **MVC-архитектура** — поймёте, как разделять логику, представление и данные.
- **Фреймворки** — попробуйте Laravel (PHP), Express (Node.js) или другие инструменты, ускоряющие разработку.
- **REST API** — создавайте интерфейсы для обмена данными между клиентом и сервером (полезно для мобильных приложений и SPA).
- **Безопасность** — изучите сессии, авторизацию, защиту от CSRF, хеширование паролей.

Идеи для самостоятельных проектов

1. **Блог** — публикация статей с категориями и комментариями.
2. **Каталог товаров** — фильтрация, карточки товаров, форма заказа.
3. **Список задач (To-Do)** — добавление, выполнение, удаление задач с сохранением в базе.
4. **Галерея изображений** — загрузка файлов, отображение сеткой.
5. **Система голосования** — выбор варианта, подсчёт голосов, защита от накрутки.
6. **Личный органайзер** — заметки с возможностью поиска и сортировки.

Начните с одного проекта, реализуйте минимум функций, а затем постепенно улучшайте: добавьте валидацию, красивый дизайн, мобильную версию, авторизацию.

Веб-разработка — это не только технологии, но и **практика**. Чем больше вы пишете, тем увереннее становитесь.