Оглавление

[1. правила и структура 2](#_Toc195313477)

[2. индекс начинаем писать 3](#_Toc195313478)

[3. как сделать динамический заголовок страницы 7](#_Toc195313479)

[4. шапка сайта 11](#_Toc195313480)

[5. шапка сайта – форматирование 4 пункта 16](#_Toc195313481)

[6. главный блок начальной страницы 20](#_Toc195313482)

[7. главный блок начальной страницы –форматирование 24](#_Toc195313483)

[7. главный блок начальной страницы –форматирование. 28](#_Toc195313484)

# 1. правила и структура

Да, я полностью настроен помочь тебе в этом проекте и сделать всё максимально качественно! 😊 Твои правила диалога я понял и обязательно буду следовать им:

1. \*\*Комментарии к коду\*\*: Я буду объяснять каждую строку, каждый символ, каждую переменную, чтобы ты точно понимала, зачем это нужно.

2. \*\*Альтернативные варианты решения\*\*: После базового объяснения я предложу улучшенный или профессиональный вариант кода, если он возможен. Это поможет тебе не только освежить знания, но и углубиться в тонкости разработки.

3. \*\*Качественное выполнение работы\*\*: Я буду стараться писать чистый, оптимизированный и современный код, используя лучшие практики.

Теперь, когда мы договорились о правилах, можем начинать с верстки сайта. Давай пошагово разберем процесс:

### План действий:

1. \*\*HTML-структура\*\*: Начнем с базовой структуры HTML. Если у тебя есть макет, то можно сразу обсудить, какие секции нужны (например, header, main, footer) и как их лучше организовать.

2. \*\*CSS-стилизация\*\*: После HTML добавим базовые стили. Здесь я также объясню, почему стоит использовать те или иные теги (например, `<section>`, `<article>` вместо `<div>`).

3. \*\*JavaScript\*\*: Когда HTML и CSS будут готовы, добавим динамические элементы через JavaScript.

4. \*\*PHP и MySQL\*\*: После завершения фронтенда подключим бэкенд и базу данных.

Если ты готова, давай начнем с HTML. Расскажи или покажи, как выглядит макет сайта, и мы вместе составим структуру. Например:

- Есть ли шапка (header)?

- Какие блоки на главной странице?

- Какие страницы вообще нужны (главная, контакты, каталог и т.д.)?

# 2. индекс начинаем писать

Твой код выглядит логично и соответствует базовым принципам структурирования PHP-проекта. Однако давай разберем его подробно, добавим комментарии и предложим улучшения, чтобы сделать его более профессиональным и удобным для дальнейшего развития.

---

### Разбор твоего кода

```php

<?php

session\_start(); // Запускаем сессию. Это необходимо для хранения данных пользователя (например, авторизован ли он).

require\_once("database/dbconnect.php"); // Подключаем файл для работы с базой данных. Функция require\_once гарантирует, что файл будет подключен только один раз.

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"> <!-- Устанавливаем кодировку UTF-8 для корректного отображения символов -->

<meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Делаем сайт адаптивным для мобильных устройств -->

<title>Document</title> <!-- Заголовок страницы. Позже можно заменить на динамический заголовок -->

</head>

<body>

<main>

<?php include "includes/header.php"; ?> <!-- Подключаем шапку сайта из отдельного файла. Это позволяет легко редактировать шапку в одном месте -->

<div class="main">

<?php include "includes/main\_container.php"; ?> <!-- Подключаем главный блок контента. Его содержимое может меняться в зависимости от страницы -->

</div>

<?php include "includes/footer.php"; ?> <!-- Подключаем подвал сайта из отдельного файла -->

</main>

</body>

</html>

```

---

### Комментарии к коду:

1. \*\*`session\_start()`\*\*:

- Сессии действительно нужны, если ты планируешь работать с авторизацией или другими данными пользователя. Всё верно.

- Совет: Если сессии не используются на всех страницах, можно их запускать только там, где это нужно. Но пока оставим как есть.

2. \*\*`require\_once("database/dbconnect.php")`\*\*:

- Хорошо, что ты используешь `require\_once`. Это гарантирует, что файл с подключением к базе данных будет подключен только один раз, и если файл не найден, скрипт завершится с ошибкой (что лучше, чем продолжать выполнение без БД).

3. \*\*HTML-структура\*\*:

- Тег `<main>` используется правильно, так как он обозначает основное содержимое страницы.

- Однако внутри `<main>` лучше использовать семантические теги, такие как `<header>`, `<section>` и `<footer>`, вместо простых `<div>`. Это улучшает читаемость кода и помогает поисковым системам лучше понимать структуру страницы.

4. \*\*Подключение файлов через `include`\*\*:

- Использование `include` для подключения частей страницы — это хорошая практика. Это делает код модульным и упрощает поддержку.

- Однако, если файлы шапки, подвала или контента критически важны для работы страницы, лучше использовать `require` вместо `include`. Разница в том, что `require` вызывает фатальную ошибку, если файл не найден, а `include` только выведет предупреждение.

---

### Улучшенный вариант кода

```php

<?php

session\_start(); // Запускаем сессию для работы с данными пользователя

require\_once("database/dbconnect.php"); // Подключаем файл для работы с базой данных

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"> <!-- Устанавливаем кодировку UTF-8 -->

<meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Делаем сайт адаптивным -->

<title>Образовательный курс</title> <!-- Заменили "Document" на осмысленный заголовок -->

</head>

<body>

<header>

<?php require "includes/header.php"; ?> <!-- Подключаем шапку сайта. Используем require, так как шапка критически важна -->

</header>

<main>

<section class="main-content">

<?php require "includes/main\_container.php"; ?> <!-- Подключаем главный блок контента. Используем require -->

</section>

</main>

<footer>

<?php require "includes/footer.php"; ?> <!-- Подключаем подвал сайта. Используем require -->

</footer>

</body>

</html>

```

### Что изменилось и почему:

1. \*\*Семантические теги\*\*:

- Заменил `<div class="main">` на `<section class="main-content">`. Это более семантично и показывает, что это основной контент страницы.

- Добавил `<header>` и `<footer>` вместо включения их внутрь `<main>`. Это правильная структура HTML5.

2. \*\*Использование `require`\*\*:

- Заменил `include` на `require` для подключения файлов шапки, контента и подвала. Это гарантирует, что страница не будет работать без этих критически важных элементов.

3. \*\*Заголовок страницы\*\*:

- Заменил `"Document"` на `"Образовательный курс"`. Это делает заголовок страницы более информативным. Позже можно сделать его динамическим (например, менять в зависимости от страницы).

4. \*\*Комментарии\*\*:

- Добавил подробные комментарии к каждому элементу, чтобы было понятно, зачем они нужны.

---

### Дополнительные советы:

1. \*\*CSS-классы\*\*:

- Класс `main-content` можно использовать для стилизации главного блока. Например, задать ему отступы или ширину.

2. \*\*Динамический заголовок\*\*:

- Позже можно сделать заголовок страницы динамическим, передавая его через PHP. Например:

```php

<title><?php echo $pageTitle; ?></title>

```

Где `$pageTitle` — переменная, которую ты можешь задавать на каждой странице.

3. \*\*Адаптивность\*\*:

- Убедись, что макет сайта будет адаптивным. Для этого используй CSS-фреймворки (например, Bootstrap) или пиши медиа-запросы.

---

Если всё понятно, можем перейти к следующему шагу: написанию кода для `header.php`, `main\_container.php` и `footer.php`. Какой файл ты хочешь разобрать первым?

# 3. как сделать динамический заголовок страницы

Чтобы задавать динамический заголовок страницы, можно использовать переменную PHP, которая будет хранить название страницы. Эта переменная будет передаваться в `<title>` через PHP. Давай разберем этот процесс пошагово.

---

### 1. \*\*Основная идея\*\*

Мы создадим переменную `$pageTitle` на каждой странице и будем подставлять её значение в тег `<title>`. Например:

- На главной странице заголовок будет `"Главная страница"`.

- На странице учебника заголовок будет `"Учебник - Образовательный курс"`.

- На странице авторизации заголовок будет `"Авторизация - Образовательный курс"`.

---

### 2. \*\*Реализация\*\*

#### Шаг 1: Изменение основного шаблона (`index.php`)

В основном файле (например, `index.php`) мы добавим переменную `$pageTitle` и используем её в теге `<title>`:

```php

<?php

session\_start(); // Запускаем сессию

require\_once("database/dbconnect.php"); // Подключаем базу данных

// Устанавливаем значение переменной $pageTitle. Если она не задана, используем значение по умолчанию.

$pageTitle = isset($pageTitle) ? $pageTitle : "Образовательный курс";

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"> <!-- Устанавливаем кодировку -->

<meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Делаем сайт адаптивным -->

<title><?php echo $pageTitle; ?></title> <!-- Динамический заголовок -->

</head>

<body>

<header>

<?php require "includes/header.php"; ?> <!-- Подключаем шапку -->

</header>

<main>

<section class="main-content">

<?php require "includes/main\_container.php"; ?> <!-- Подключаем контент -->

</section>

</main>

<footer>

<?php require "includes/footer.php"; ?> <!-- Подключаем подвал -->

</footer>

</body>

</html>

```

#### Комментарии:

- Мы используем тернарный оператор `isset($pageTitle) ? $pageTitle : "Образовательный курс"`, чтобы проверить, была ли переменная `$pageTitle` определена. Если нет, то используется значение по умолчанию `"Образовательный курс"`.

- В теге `<title>` мы выводим значение переменной `$pageTitle` с помощью `<?php echo $pageTitle; ?>`.

---

#### Шаг 2: Задание заголовка для конкретных страниц

Теперь на каждой странице, перед подключением основного шаблона, мы будем задавать значение переменной `$pageTitle`. Например:

##### Главная страница (`home.php`):

```php

<?php

$pageTitle = "Главная страница - Образовательный курс"; // Задаем заголовок

include "index.php"; // Подключаем основной шаблон

?>

```

##### Страница учебника (`course.php`):

```php

<?php

$pageTitle = "Учебник - Образовательный курс"; // Задаем заголовок

include "index.php"; // Подключаем основной шаблон

?>

```

##### Страница авторизации (`login.php`):

```php

<?php

$pageTitle = "Авторизация - Образовательный курс"; // Задаем заголовок

include "index.php"; // Подключаем основной шаблон

?>

```

---

### 3. \*\*Как это работает\*\*

1. Когда пользователь заходит на страницу (например, `home.php`), сначала выполняется код этой страницы.

2. Переменная `$pageTitle` задается в начале файла.

3. Затем подключается основной шаблон (`index.php`), где значение `$pageTitle` подставляется в тег `<title>`.

---

### 4. \*\*Пример результата\*\*

- Если пользователь заходит на `home.php`, заголовок страницы будет:

```

Главная страница - Образовательный курс

```

- Если пользователь заходит на `course.php`, заголовок страницы будет:

```

Учебник - Образовательный курс

```

- Если страница не задает `$pageTitle`, заголовок будет:

```

Образовательный курс

```

---

### 5. \*\*Дополнительные улучшения\*\*

1. \*\*Добавление суффикса\*\*:

- Если ты хочешь, чтобы к каждому заголовку автоматически добавлялся суффикс (например, `" - Образовательный курс"`), можно сделать это в основном шаблоне:

```php

$pageTitle = isset($pageTitle) ? $pageTitle . " - Образовательный курс" : "Образовательный курс";

```

2. \*\*Использование констант\*\*:

- Если суффикс одинаковый для всех страниц, его можно вынести в константу:

```php

define("SITE\_NAME", "Образовательный курс");

$pageTitle = isset($pageTitle) ? $pageTitle . " - " . SITE\_NAME : SITE\_NAME;

```

---

Если всё понятно, можем двигаться дальше. Какой следующий шаг тебя интересует? Например, написание кода для `header.php` или `main\_container.php`? 😊

Продолжение   
Да, конечно, помню! Динамические заголовки страниц — это отличная идея. Они помогут сделать сайт более удобным для пользователей и поисковых систем. Мы можем реализовать их так, чтобы заголовок менялся в зависимости от того, какая страница открыта в главном блоке.

---

### 1. \*\*Как это работает\*\*

Основная идея:

- Мы будем задавать заголовок страницы (`<title>`) динамически на основе параметра `page` из URL.

- Для каждой страницы (например, `home`, `course`, `profile`) будет свой заголовок.

- Если параметр `page` не указан или страница не найдена, будет использоваться заголовок по умолчанию.

---

### 2. \*\*Реализация\*\*

#### Шаг 1: Добавление переменной `$pageTitle`

Мы создадим переменную `$pageTitle` в каждом файле страницы (например, `home.php`, `course.php`, `profile.php`). Эта переменная будет содержать заголовок для конкретной страницы.

##### Пример файла `pages/home.php`:

```php

<?php

$pageTitle = "Главная страница - Образовательный курс"; // Заголовок для главной страницы

?>

<h2>Добро пожаловать на образовательный курс!</h2>

<p>Здесь вы найдете полезные материалы для обучения.</p>

```

##### Пример файла `pages/course.php`:

```php

<?php

$pageTitle = "Учебник - Образовательный курс"; // Заголовок для страницы учебника

?>

<h2>Учебник</h2>

<p>Это страница с материалами курса.</p>

```

##### Пример файла `pages/profile.php`:

```php

<?php

$pageTitle = "Личный кабинет - Образовательный курс"; // Заголовок для личного кабинета

?>

<h2>Личный кабинет</h2>

<p>Здесь вы можете управлять своим профилем.</p>

```

---

#### Шаг 2: Использование `$pageTitle` в основном шаблоне

Теперь мы добавим переменную `$pageTitle` в тег `<title>` в основном файле (`index.php`).

##### Код в `index.php`:

```php

<?php

session\_start(); // Запускаем сессию

require\_once("database/dbconnect.php"); // Подключаем базу данных

// Устанавливаем значение переменной $pageTitle. Если она не задана, используем значение по умолчанию.

$pageTitle = isset($pageTitle) ? $pageTitle : "Образовательный курс";

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"> <!-- Устанавливаем кодировку -->

<meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Делаем сайт адаптивным -->

<title><?php echo $pageTitle; ?></title> <!-- Динамический заголовок -->

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

<link rel="stylesheet" href="styles.css"> <!-- Подключаем стили -->

</head>

<body>

<header>

<?php require "includes/header.php"; ?> <!-- Подключаем шапку -->

</header>

<main>

<section class="main-content">

<?php require "includes/main\_container.php"; ?> <!-- Подключаем главный блок -->

</section>

</main>

<footer>

<?php require "includes/footer.php"; ?> <!-- Подключаем подвал -->

</footer>

</body>

</html>

```

---

#### Шаг 3: Обработка ошибок (страница не найдена)

Если пользователь передает недопустимый параметр `page` (например, `?page=secret`), мы можем задать специальный заголовок для ошибки.

##### Изменения в `main\_container.php`:

```php

<div class="main\_\_container">

<?php

// Определяем, какой контент нужно показать

if (isset($\_GET['page'])) { // Используем параметр "page"

$allowed\_pages = ['home', 'course', 'profile']; // Список разрешенных страниц

$page = basename($\_GET['page']); // Защита от пути к файлу

if (in\_array($page, $allowed\_pages)) {

include "pages/$page.php"; // Подключаем только разрешенные файлы

} else {

$pageTitle = "Страница не найдена - Образовательный курс"; // Заголовок для ошибки

echo "<p>Страница не найдена.</p>"; // Сообщение об ошибке

}

} else {

// Содержимое по умолчанию (главная страница)

include "pages/home.php";

}

?>

</div>

```

---

### 3. \*\*Пример работы\*\*

1. \*\*Главная страница\*\*:

- URL: `http://example.com/index.php?page=home`

- Заголовок: `"Главная страница - Образовательный курс"`

2. \*\*Страница учебника\*\*:

- URL: `http://example.com/index.php?page=course`

- Заголовок: `"Учебник - Образовательный курс"`

3. \*\*Личный кабинет\*\*:

- URL: `http://example.com/index.php?page=profile`

- Заголовок: `"Личный кабинет - Образовательный курс"`

4. \*\*Недопустимый параметр\*\*:

- URL: `http://example.com/index.php?page=secret`

- Заголовок: `"Страница не найдена - Образовательный курс"`

5. \*\*Без параметра\*\*:

- URL: `http://example.com/index.php`

- Заголовок: `"Главная страница - Образовательный курс"`

---

### 4. \*\*Дополнительные улучшения\*\*

1. \*\*Добавление суффикса\*\*:

Если ты хочешь, чтобы к каждому заголовку автоматически добавлялся суффикс (например, `" - Образовательный курс"`), можно сделать это в основном шаблоне:

```php

$pageTitle = isset($pageTitle) ? $pageTitle . " - Образовательный курс" : "Образовательный курс";

```

2. \*\*Использование констант\*\*:

Если суффикс одинаковый для всех страниц, его можно вынести в константу:

```php

define("SITE\_NAME", "Образовательный курс");

$pageTitle = isset($pageTitle) ? $pageTitle . " - " . SITE\_NAME : SITE\_NAME;

```

---

### 5. \*\*Итог\*\*

Теперь у тебя есть:

1. \*\*Динамические заголовки\*\*, которые меняются в зависимости от открытой страницы.

2. \*\*Защита от ошибок\*\*: Если страница не найдена, выводится соответствующий заголовок.

3. \*\*Чистый и понятный код\*\*: Каждый файл страницы задает свой заголовок через переменную `$pageTitle`.

Если всё понятно, можем двигаться дальше. Например, можно добавить боковое меню или начать работу с базой данных. Какой следующий шаг тебя интересует? 😊

Tot  
Спасибо за уточнение! Давай разберем, почему заголовок страницы (`<title>`) не меняется при ошибке "Страница не найдена". Проблема заключается в том, что переменная `$pageTitle` задается внутри блока `main\_container.php`, но она не передается обратно в основной файл `index.php`. Чтобы это исправить, нужно немного изменить логику работы.

---

### 1. \*\*Почему это происходит\*\*

В PHP переменные имеют область видимости. Переменная `$pageTitle`, которую мы задаем в `main\_container.php`, существует только внутри этого файла. Она не доступна в `index.php`, где используется для генерации тега `<title>`.

Чтобы решить эту проблему, нужно:

1. Задать переменную `$pageTitle` в основном файле (`index.php`).

2. Передавать её значение из `main\_container.php` через глобальную или сессионную переменную.

---

### 2. \*\*Исправленный код\*\*

#### Шаг 1: Основной файл (`index.php`)

Мы будем задавать `$pageTitle` в `index.php` и позволять `main\_container.php` изменять её значение.

```php

<?php

session\_start(); // Запускаем сессию

require\_once("database/dbconnect.php"); // Подключаем базу данных

// Устанавливаем значение переменной $pageTitle по умолчанию

$pageTitle = "Образовательный курс";

// Подключаем главный контент

ob\_start(); // Включаем буферизацию вывода

require "includes/main\_container.php";

ob\_end\_flush(); // Очищаем буфер

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"> <!-- Устанавливаем кодировку -->

<meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Делаем сайт адаптивным -->

<title><?php echo $pageTitle; ?></title> <!-- Динамический заголовок -->

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

<link rel="stylesheet" href="styles.css"> <!-- Подключаем стили -->

</head>

<body>

<header>

<?php require "includes/header.php"; ?> <!-- Подключаем шапку -->

</header>

<main>

<section class="main-content">

<?php require "includes/main\_container.php"; ?> <!-- Подключаем главный блок -->

</section>

</main>

<footer>

<?php require "includes/footer.php"; ?> <!-- Подключаем подвал -->

</footer>

</body>

</html>

```

---

#### Шаг 2: Главный блок (`main\_container.php`)

Теперь мы будем изменять глобальную переменную `$pageTitle` внутри `main\_container.php`.

```php

<div class="main\_\_container">

<?php

// Определяем, какой контент нужно показать

if (isset($\_GET['page'])) { // Используем параметр "page"

$allowed\_pages = ['home', 'course', 'profile']; // Список разрешенных страниц

$page = basename($\_GET['page']); // Защита от пути к файлу

if (in\_array($page, $allowed\_pages)) {

include "pages/$page.php"; // Подключаем только разрешенные файлы

} else {

// Если страница не найдена

$GLOBALS['pageTitle'] = "Страница не найдена - Образовательный курс"; // Меняем заголовок

echo "<p>Страница не найдена.</p>"; // Сообщение об ошибке

}

} else {

// Содержимое по умолчанию (главная страница)

include "pages/home.php";

}

?>

</div>

```

---

### 3. \*\*Как это работает\*\*

1. \*\*Буферизация вывода\*\*:

- Мы используем функцию `ob\_start()` в `index.php`, чтобы временно сохранить вывод `main\_container.php`.

- Это позволяет нам изменить переменную `$pageTitle` до того, как будет сгенерирован тег `<title>`.

2. \*\*Глобальная переменная\*\*:

- В `main\_container.php` мы используем `$GLOBALS['pageTitle']`, чтобы изменить значение переменной `$pageTitle`, объявленной в `index.php`.

3. \*\*Заголовок страницы\*\*:

- Теперь, если страница не найдена, переменная `$pageTitle` изменяется на `"Страница не найдена - Образовательный курс"`, и этот текст отображается в теге `<title>`.

---

### 4. \*\*Пример работы\*\*

1. \*\*Главная страница\*\*:

- URL: `http://example.com/index.php?page=home`

- Заголовок: `"Главная страница - Образовательный курс"`

2. \*\*Страница учебника\*\*:

- URL: `http://example.com/index.php?page=course`

- Заголовок: `"Учебник - Образовательный курс"`

3. \*\*Личный кабинет\*\*:

- URL: `http://example.com/index.php?page=profile`

- Заголовок: `"Личный кабинет - Образовательный курс"`

4. \*\*Недопустимый параметр\*\*:

- URL: `http://example.com/index.php?page=secret`

- Заголовок: `"Страница не найдена - Образовательный курс"`

- Выводится сообщение: `"Страница не найдена."`

5. \*\*Без параметра\*\*:

- URL: `http://example.com/index.php`

- Заголовок: `"Главная страница - Образовательный курс"`

---

### 5. \*\*Дополнительные улучшения\*\*

1. \*\*Перенаправление на 404-страницу\*\*:

Если ты хочешь сделать более профессиональную обработку ошибок, можно создать отдельный файл `404.php` и перенаправлять пользователя на него:

```php

header("HTTP/1.0 404 Not Found");

include "pages/404.php";

exit;

```

2. \*\*Очистка URL\*\*:

Если ты хочешь, чтобы адресная строка всегда была чистой (например, без `?page=home`), можно использовать `.htaccess` для красивых URL (например, `/home` вместо `?page=home`).

---

### 6. \*\*Итог\*\*

Теперь заголовок страницы (`<title>`) будет меняться корректно:

- Для каждой страницы — свой заголовок.

- При ошибке "Страница не найдена" — соответствующий заголовок.

Если всё понятно, можем двигаться дальше. Например, можно добавить боковое меню или начать работу с базой данных. Какой следующий шаг тебя интересует? 😊

Теперь я понял, в чём проблема! Дело в том, что заголовок страницы (`<title>`) генерируется в `<head>` файла `index.php`, а содержимое главного блока (`main\_container.php`) подключается уже \*\*после\*\* объявления `$pageTitle`. Это создает конфликт: если мы убираем одно из подключений `main\_container.php`, то либо заголовки не работают, либо контент выводится в неправильном месте.

Давай исправим это так, чтобы:

1. Заголовок страницы (`<title>`) работал корректно.

2. Содержимое главного блока выводилось только один раз и в правильном месте (внутри `<main>`).

---

### 1. \*\*Почему это происходит\*\*

Проблема возникает из-за того, что:

- Заголовок страницы (`<title>`) генерируется в `<head>` файла `index.php`.

- Переменная `$pageTitle` должна быть изменена до того, как будет сгенерирован тег `<title>`.

- Однако содержимое главного блока (`main\_container.php`) должно выводиться внутри `<main>`.

Чтобы решить эту проблему, нужно выполнить следующие шаги:

1. Выполнить логику обработки параметра `page` \*\*до\*\* генерации `<head>`.

2. Вывести содержимое главного блока только один раз, внутри `<main>`.

---

### 2. \*\*Исправленный код\*\*

#### Шаг 1: Логика обработки параметра `page` в начале файла

Мы будем обрабатывать параметр `page` в самом начале файла `index.php`, чтобы определить значение `$pageTitle` до генерации `<head>`.

##### `index.php`:

```php

<?php

session\_start(); // Запускаем сессию

require\_once("database/dbconnect.php"); // Подключаем базу данных

// Устанавливаем значение переменной $pageTitle по умолчанию

$pageTitle = "Образовательный курс";

// Определяем, какой контент нужно показать

if (isset($\_GET['page'])) { // Используем параметр "page"

$allowed\_pages = ['home', 'course', 'profile']; // Список разрешенных страниц

$page = basename($\_GET['page']); // Защита от пути к файлу

if (in\_array($page, $allowed\_pages)) {

// Подключаем файл для изменения $pageTitle

include "pages/$page.php";

} else {

// Если страница не найдена

$pageTitle = "Страница не найдена - Образовательный курс"; // Меняем заголовок

}

} else {

// Содержимое по умолчанию (главная страница)

include "pages/home.php";

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"> <!-- Устанавливаем кодировку -->

<meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Делаем сайт адаптивным -->

<title><?php echo $pageTitle; ?></title> <!-- Динамический заголовок -->

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

<link rel="stylesheet" href="styles.css"> <!-- Подключаем стили -->

</head>

<body>

<header>

<?php require "includes/header.php"; ?> <!-- Подключаем шапку -->

</header>

<main>

<section class="main-content">

<?php require "includes/main\_container.php"; ?> <!-- Подключаем главный блок -->

</section>

</main>

<footer>

<?php require "includes/footer.php"; ?> <!-- Подключаем подвал -->

</footer>

</body>

</html>

```

---

#### Шаг 2: Главный блок (`main\_container.php`)

Теперь в `main\_container.php` нам нужно только вывести содержимое страницы. Логика обработки параметра `page` уже выполнена в `index.php`.

##### `main\_container.php`:

```php

<div class="main\_\_container">

<?php

// Определяем, какой контент нужно показать

if (isset($\_GET['page'])) { // Используем параметр "page"

$allowed\_pages = ['home', 'course', 'profile']; // Список разрешенных страниц

$page = basename($\_GET['page']); // Защита от пути к файлу

if (in\_array($page, $allowed\_pages)) {

include "pages/$page.php"; // Подключаем только разрешенные файлы

} else {

// Если страница не найдена

echo "<p>Страница не найдена.</p>"; // Сообщение об ошибке

}

} else {

// Содержимое по умолчанию (главная страница)

include "pages/home.php";

}

?>

</div>

```

---

### 3. \*\*Как это работает\*\*

1. \*\*Логика обработки параметра `page`\*\*:

- В начале файла `index.php` мы проверяем параметр `page` и подключаем соответствующий файл из папки `pages`.

- Если файл существует, он может изменить значение `$pageTitle`.

- Если файл не существует, устанавливается заголовок `"Страница не найдена"`.

2. \*\*Генерация `<title>`\*\*:

- Переменная `$pageTitle` уже задана на момент генерации `<head>`, поэтому заголовок страницы будет корректным.

3. \*\*Вывод контента\*\*:

- Содержимое главного блока выводится только один раз, внутри `<main>`.

---

### 4. \*\*Пример работы\*\*

1. \*\*Главная страница\*\*:

- URL: `http://example.com/index.php?page=home`

- Заголовок: `"Главная страница - Образовательный курс"`

- Контент: Отображается содержимое файла `pages/home.php`.

2. \*\*Страница учебника\*\*:

- URL: `http://example.com/index.php?page=course`

- Заголовок: `"Учебник - Образовательный курс"`

- Контент: Отображается содержимое файла `pages/course.php`.

3. \*\*Недопустимый параметр\*\*:

- URL: `http://example.com/index.php?page=secret`

- Заголовок: `"Страница не найдена - Образовательный курс"`

- Контент: Отображается сообщение: `"Страница не найдена."`

4. \*\*Без параметра\*\*:

- URL: `http://example.com/index.php`

- Заголовок: `"Главная страница - Образовательный курс"`

- Контент: Отображается содержимое файла `pages/home.php`.

---

### 5. \*\*Итог\*\*

Теперь всё работает корректно:

- Заголовок страницы (`<title>`) меняется динамически.

- Содержимое главного блока выводится только один раз и в правильном месте (внутри `<main>`).

- Никаких дублирований или ошибок в расположении контента.

Если всё работает, можем двигаться дальше. Например, можно добавить боковое меню или начать работу с базой данных. Какой следующий шаг тебя интересует? 😊

# 4. шапка сайта

Отлично! Давай начнем с верстки шапки (`header.php`). Мы создадим HTML-структуру для шапки, добавим стили CSS и объясним каждую часть кода пошагово.

### Шаг 1: Создание базовой структуры HTML

Вот как может выглядеть HTML-код для шапки:

```html

<!-- includes/header.php -->

<header class="site-header">

<div class="header-container">

<!-- Иконка сайта (возвращает на главную страницу) -->

<a href="/" class="site-logo">

<img src="images/logo.png" alt="Логотип сайта" width="30" height="30">

</a>

<!-- Название "Учебник" (переходит на страницу учебника) -->

<h1 class="site-title">

<a href="/course.php">Учебник</a>

</h1>

<!-- Иконка пользователя (переходит в личный кабинет) -->

<a href="/profile.php" class="user-icon">

<img src="images/user-icon.png" alt="Иконка пользователя" width="30" height="30">

</a>

</div>

</header>

```

#### Комментарии к коду:

1. \*\*`<header>`\*\*:

- Это семантический тег, который обозначает шапку документа.

- Класс `site-header` используется для стилизации.

2. \*\*`.header-container`\*\*:

- Это контейнер, который поможет нам центрировать или выравнивать элементы внутри шапки.

3. \*\*Иконка сайта (`site-logo`)\*\*:

- Ссылка `<a>` ведет на главную страницу (`/`).

- Внутри ссылки находится изображение логотипа (`<img>`).

4. \*\*Название "Учебник" (`site-title`)\*\*:

- Текст "Учебник" помещен в заголовок `<h1>`, чтобы подчеркнуть его важность.

- Ссылка `<a>` ведет на страницу учебника (`/course.php`).

5. \*\*Иконка пользователя (`user-icon`)\*\*:

- Ссылка `<a>` ведет в личный кабинет (`/profile.php`).

- Внутри ссылки находится изображение иконки пользователя (`<img>`).

---

### Шаг 2: Добавление стилей CSS

Теперь давайте напишем стили для этой шапки. Мы сделаем её чёрной с белым текстом и добавим простое позиционирование элементов.

```css

/\* styles.css \*/

.site-header {

background-color: #000; /\* Чёрный фон \*/

color: #fff; /\* Белый текст \*/

padding: 10px 20px; /\* Поля вокруг содержимого \*/

}

.header-container {

display: flex; /\* Используем Flexbox для горизонтального расположения элементов \*/

justify-content: space-between; /\* Распределяем элементы по краям \*/

align-items: center; /\* Выравниваем элементы по вертикали \*/

}

.site-logo img,

.user-icon img {

vertical-align: middle; /\* Выравнивание изображений по середине \*/

}

.site-title {

font-size: 1.5em; /\* Увеличиваем размер текста \*/

margin: 0; /\* Убираем отступы заголовка \*/

}

.site-title a {

text-decoration: none; /\* Убираем подчеркивание у ссылки \*/

color: inherit; /\* Наследуем цвет текста от родителя \*/

}

```

#### Комментарии к стилям:

1. \*\*`.site-header`\*\*:

- Фоновый цвет установлен в чёрный (`#000`), а текст — в белый (`#fff`).

- Поля (`padding`) добавлены для создания пространства между границами шапки и её содержимым.

2. \*\*`.header-container`\*\*:

- Мы используем Flexbox (`display: flex`) для горизонтального размещения элементов.

- `justify-content: space-between` распределяет элементы по краям контейнера.

- `align-items: center` выравнивает элементы по вертикали.

3. \*\*Иконки (`site-logo img` и `user-icon img`)\*\*:

- `vertical-align: middle` гарантирует, что изображения будут выровнены по центру относительно текста.

4. \*\*`.site-title`\*\*:

- Размер текста увеличен до `1.5em` для большего акцента на название "Учебник".

- Отступы (`margin`) сброшены, так как заголовки обычно имеют стандартные отступы.

5. \*\*Ссылки (`site-title a`)\*\*:

- Подчеркивание (`text-decoration`) сброшено, чтобы ссылка не выделялась.

- Цвет ссылки наследуется от родительского элемента (`color: inherit`), чтобы он соответствовал общему цвету текста.

---

### Шаг 3: Полная реализация

#### HTML (`includes/header.php`):

```html

<header class="site-header">

<div class="header-container">

<a href="/" class="site-logo">

<img src="images/logo.png" alt="Логотип сайта" width="30" height="30">

</a>

<h1 class="site-title">

<a href="/course.php">Учебник</a>

</h1>

<a href="/profile.php" class="user-icon">

<img src="images/user-icon.png" alt="Иконка пользователя" width="30" height="30">

</a>

</div>

</header>

```

#### CSS (`styles.css`):

```css

.site-header {

background-color: #000;

color: #fff;

padding: 10px 20px;

}

.header-container {

display: flex;

justify-content: space-between;

align-items: center;

}

.site-logo img,

.user-icon img {

vertical-align: middle;

}

.site-title {

font-size: 1.5em;

margin: 0;

}

.site-title a {

text-decoration: none;

color: inherit;

}

```

---

### Результат

Шапка будет выглядеть следующим образом:

- Левая часть: иконка сайта (переходит на главную страницу).

- Центральная часть: название "Учебник" (переходит на страницу учебника).

- Правая часть: иконка пользователя (переходит в личный кабинет).

Если ты хочешь добавить дополнительные стили (например, hover-эффекты или адаптивность), можем сделать это в следующем шаге. Как тебе такой вариант? Если есть вопросы или нужно уточнить что-то — спрашивай! 😊

# 5. шапка сайта – форматирование 4 пункта

Отлично, я понял твои вопросы и пожелания! Давай разберем всё по порядку: начнем с объяснения ссылок (`<a href="/course.php">`), затем создадим общие стили для всего сайта (включая шрифты, отступы и размер текста), а потом уточним форматирование только для блоков шапки.

---

### 1. \*\*Объяснение ссылок (`<a href="/course.php">`)\*\*

Ссылка `<a>` в HTML используется для создания гиперссылок. Вот как это работает:

#### Синтаксис:

```html

<a href="путь\_к\_файлу">Текст ссылки</a>

```

- \*\*`href`\*\* — это атрибут, который указывает адрес, куда ведет ссылка.

- \*\*Текст ссылки\*\* — это то, что пользователь видит на странице и может кликнуть.

#### Пример:

```html

<a href="/course.php">Учебник</a>

```

- \*\*`/course.php`\*\*: Это путь к файлу `course.php`, который находится в корневой директории сайта.

- Если сайт расположен в папке `http://example.com/`, то `/course.php` будет означать `http://example.com/course.php`.

- Косая черта (`/`) в начале пути означает, что путь начинается с корня сайта. Без неё путь будет относительным (например, `course.php` будет искаться в текущей папке).

#### Что происходит при клике:

Когда пользователь нажимает на ссылку, браузер отправляет запрос на сервер по указанному адресу (`/course.php`). Сервер возвращает содержимое файла `course.php`, и браузер отображает его.

#### Важные моменты:

1. Если ты используешь PHP для подключения файлов (например, через `include`), то ссылки должны быть абсолютными (начинаться с `/`).

2. Если файл находится в другой папке, нужно указать путь до него. Например:

- `href="/pages/course.php"` — файл `course.php` находится в папке `pages`.

---

### 2. \*\*Общие стили для всего сайта\*\*

Для того чтобы задать общий стиль для всего сайта, мы создадим базовые правила CSS. Эти правила будут применяться ко всем элементам, если они не переопределены в других частях кода.

#### Общие стили:

```css

/\* styles.css \*/

/\* Обнуление стандартных отступов \*/

\* {

margin: 0; /\* Убираем внешние отступы \*/

padding: 0; /\* Убираем внутренние отступы \*/

box-sizing: border-box; /\* Все отступы считаются внутри элемента \*/

}

/\* Базовые стили для body \*/

body {

font-family: 'Roboto', Arial, sans-serif; /\* Используем красивый шрифт Roboto \*/

font-size: 16px; /\* Размер шрифта по умолчанию \*/

line-height: 1.6; /\* Межстрочный интервал \*/

color: #333; /\* Цвет текста \*/

background-color: #f9f9f9; /\* Цвет фона страницы \*/

padding: 20px; /\* Отступы по краям страницы \*/

}

```

#### Комментарии:

1. \*\*`\*`\*\*:

- Это универсальный селектор, который применяется ко всем элементам на странице.

- Мы обнуляем `margin` и `padding`, чтобы убрать стандартные отступы браузера.

- `box-sizing: border-box` гарантирует, что все отступы (например, `padding`) учитываются внутри элемента, а не добавляются к его ширине.

2. \*\*`body`\*\*:

- `font-family`: Мы используем шрифт `Roboto`, который хорошо читается на русском языке. Если он недоступен, используются резервные шрифты (`Arial`, `sans-serif`).

- `font-size`: Размер шрифта по умолчанию установлен в `16px`. Это оптимальный размер для чтения.

- `line-height`: Межстрочный интервал установлен в `1.6`, чтобы текст был удобочитаемым.

- `color`: Цвет текста — темно-серый (`#333`), чтобы он был мягче, чем чёрный.

- `background-color`: Фон страницы — светло-серый (`#f9f9f9`), чтобы глаза не уставали.

- `padding`: Отступы по краям страницы добавлены для удобства.

---

### 3. \*\*Стили для шапки\*\*

Теперь давай добавим стили только для шапки. Чтобы избежать конфликтов с другими блоками, мы будем использовать классы.

#### HTML (`includes/header.php`):

```html

<header class="site-header">

<div class="header-container">

<a href="/" class="site-logo">

<img src="images/logo.png" alt="Логотип сайта" width="30" height="30">

</a>

<h1 class="site-title">

<a href="/course.php">Учебник</a>

</h1>

<a href="/profile.php" class="user-icon">

<img src="images/user-icon.png" alt="Иконка пользователя" width="30" height="30">

</a>

</div>

</header>

```

#### CSS (`styles.css`):

```css

.site-header {

background-color: #000; /\* Чёрный фон \*/

color: #fff; /\* Белый текст \*/

padding: 10px 20px; /\* Поля вокруг содержимого \*/

}

.header-container {

display: flex; /\* Flexbox для горизонтального расположения элементов \*/

justify-content: space-between; /\* Распределяем элементы по краям \*/

align-items: center; /\* Выравниваем элементы по вертикали \*/

}

.site-logo img,

.user-icon img {

vertical-align: middle; /\* Выравнивание изображений по середине \*/

}

.site-title {

font-size: 1.5em; /\* Увеличиваем размер текста \*/

margin: 0; /\* Убираем отступы заголовка \*/

}

.site-title a {

text-decoration: none; /\* Убираем подчеркивание у ссылки \*/

color: inherit; /\* Наследуем цвет текста от родителя \*/

}

```

---

### 4. \*\*Добавление шрифта Roboto\*\*

Чтобы использовать шрифт `Roboto`, нужно подключить его через Google Fonts. Для этого добавь следующую строку в `<head>` основного файла (`index.php`):

```html

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

```

#### Комментарии:

- `wght@400;700` означает, что мы подключаем два варианта шрифта: обычный (`400`) и жирный (`700`).

- `display=swap` гарантирует, что шрифт будет загружаться асинхронно, чтобы страница не "мигала".

---

### 5. \*\*Результат\*\*

Теперь у тебя есть:

1. \*\*Общие стили для всего сайта\*\*: шрифт `Roboto`, удобочитаемый размер текста, отступы и фон.

2. \*\*Стили для шапки\*\*: чёрный фон, белый текст, выровненные элементы.

3. \*\*Понятные ссылки\*\*: ты знаешь, как работают пути в `href`.

# 6. главный блок начальной страницы

Отлично, давай разберем этот код и улучшим его, чтобы он был более понятным, безопасным и профессиональным. Главный блок (`main\_container.php`) будет динамически подгружать контент в зависимости от параметров, которые передаются через URL.

### 1. \*\*Объяснение текущего кода\*\*

Твой код работает так:

1. Если в URL есть параметр `main\_\_container` (например, `?main\_\_container=about.php`), то PHP подключает указанный файл.

2. Если параметра нет, то отображается содержимое по умолчанию (например, главная страница с рекламой и слайдером).

#### Пример работы:

- URL: `http://example.com/index.php?main\_\_container=about.php`

- В этом случае PHP подключит файл `about.php`.

- URL: `http://example.com/index.php`

- В этом случае отобразится содержимое по умолчанию.

### 2. \*\*Проблемы текущего кода\*\*

1. \*\*Безопасность\*\*:

- Использование параметра из URL напрямую (`include "$main\_\_container";`) может быть опасным. Пользователь может передать путь к любому файлу на сервере, что приведет к уязвимости.

- Нужно проверять и фильтровать входные данные.

2. \*\*Читаемость\*\*:

- Код можно сделать более структурированным и понятным.

3. \*\*Функциональность\*\*:

- Можно добавить больше контроля над тем, какие файлы могут быть подключены.

---

### 3. \*\*Улучшенный вариант кода\*\*

Вот как можно улучшить код:

```php

<div class="main\_\_container">

<?php

// Определяем, какой контент нужно показать

if (isset($\_GET['page'])) { // Используем параметр "page" вместо "main\_\_container"

$allowed\_pages = ['home', 'course', 'profile']; // Список разрешенных страниц

$page = basename($\_GET['page']); // Защита от пути к файлу

if (in\_array($page, $allowed\_pages)) {

include "pages/$page.php"; // Подключаем только разрешенные файлы

} else {

echo "<p>Страница не найдена.</p>"; // Сообщение об ошибке

}

} else {

// Содержимое по умолчанию (главная страница)

include "pages/home.php";

}

?>

</div>

```

---

### 4. \*\*Разбор улучшенного кода\*\*

#### Что изменилось:

1. \*\*Параметр `page`\*\*:

- Вместо `main\_\_container` мы используем параметр `page`. Это более логичное название.

- Пример URL: `http://example.com/index.php?page=course`.

2. \*\*Защита от уязвимостей\*\*:

- Мы создаем массив `$allowed\_pages`, который содержит список разрешенных страниц (например, `home`, `course`, `profile`).

- Функция `basename()` удаляет все пути из строки, оставляя только имя файла. Это предотвращает возможность передачи пути к другим файлам на сервере.

3. \*\*Проверка на разрешенные страницы\*\*:

- Функция `in\_array()` проверяет, есть ли запрошенная страница в списке разрешенных.

- Если страница разрешена, она подключается через `include "pages/$page.php"`.

- Если страница не найдена, выводится сообщение об ошибке.

4. \*\*Содержимое по умолчанию\*\*:

- Если параметр `page` не указан, подключается файл `pages/home.php`.

---

### 5. \*\*Структура папок\*\*

Для удобства организуем файлы следующим образом:

```

/project

/includes

header.php

footer.php

main\_container.php

/pages

home.php // Главная страница

course.php // Страница учебника

profile.php // Личный кабинет

/images

logo.png

user-icon.png

index.php

styles.css

```

---

### 6. \*\*Пример содержимого файлов\*\*

#### `pages/home.php` (Главная страница):

```php

<h2>Добро пожаловать на образовательный курс!</h2>

<p>Здесь вы найдете полезные материалы для обучения.</p>

```

#### `pages/course.php` (Страница учебника):

```php

<h2>Учебник</h2>

<p>Это страница с материалами курса.</p>

```

#### `pages/profile.php` (Личный кабинет):

```php

<h2>Личный кабинет</h2>

<p>Здесь вы можете управлять своим профилем.</p>

```

---

### 7. \*\*Как это работает\*\*

1. \*\*URL без параметра\*\*:

- `http://example.com/index.php`

- Подключается файл `pages/home.php`.

2. \*\*URL с параметром\*\*:

- `http://example.com/index.php?page=course`

- Подключается файл `pages/course.php`.

3. \*\*URL с недопустимым параметром\*\*:

- `http://example.com/index.php?page=secret`

- Выводится сообщение: "Страница не найдена."

---

### 8. \*\*Стили для главного блока\*\*

Добавим базовые стили для главного блока:

```css

.main\_\_container {

background-color: #fff; /\* Белый фон \*/

padding: 20px; /\* Отступы внутри блока \*/

border-radius: 8px; /\* Закругленные углы \*/

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); /\* Тень для эффекта карточки \*/

margin: 20px auto; /\* Отступы сверху и снизу, центрирование \*/

max-width: 1200px; /\* Максимальная ширина блока \*/

}

```

---

### 9. \*\*Итог\*\*

Теперь у тебя есть:

1. \*\*Безопасный и гибкий код\*\* для подключения контента в главном блоке.

2. \*\*Организованная структура папок\*\*, которая упрощает управление файлами.

3. \*\*Стилизованный главный блок\*\*, который выглядит современно.

Если всё понятно, можем двигаться дальше. Например, можно добавить боковое меню или начать работу с базой данных. Какой следующий шаг тебя интересует? 😊

# 7. главный блок начальной страницы –форматирование

Отлично, что ты задаешь такие важные вопросы! Давай разберем, почему лучше выделить страницы в отдельную папку (например, `pages`) и как это влияет на логику проекта. Если же ты предпочитаешь держать всё в папке `includes`, мы можем адаптировать код для этого случая.

---

### 1. \*\*Почему лучше выделить страницы в отдельную папку (`pages`)\*\*

Использование отдельной папки для страниц (например, `pages`) — это хорошая практика по нескольким причинам:

1. \*\*Логическая организация файлов\*\*:

- Папка `includes` обычно используется для частей, которые повторяются на всех страницах (например, шапка, подвал, боковое меню).

- Страницы, которые являются самостоятельными контентными блоками (например, главная страница, учебник, профиль), логически относятся к другому типу файлов.

2. \*\*Удобство поддержки\*\*:

- Когда файлы разделены по назначению, их легче найти и изменить.

- Например, если нужно добавить новую страницу, просто кладешь её в папку `pages`.

3. \*\*Безопасность\*\*:

- Если файлы страниц находятся в отдельной папке, их можно защитить от прямого доступа через `.htaccess` или другие методы.

- Это снижает риск того, что пользователь сможет получить доступ к ним напрямую без авторизации.

4. \*\*Масштабируемость\*\*:

- Если проект будет расти, структура с отдельной папкой `pages` позволит легче добавлять новые функциональности (например, разделение на категории или подразделы).

---

### 2. \*\*Если все файлы будут в `includes`\*\*

Если ты всё-таки хочешь хранить все файлы в папке `includes`, это тоже возможно. Вот как можно адаптировать код:

#### Улучшенный вариант кода с использованием `includes`:

```php

<div class="main\_\_container">

<?php

// Определяем, какой контент нужно показать

if (isset($\_GET['page'])) { // Используем параметр "page"

$allowed\_pages = ['home', 'course', 'profile']; // Список разрешенных страниц

$page = basename($\_GET['page']); // Защита от пути к файлу

if (in\_array($page, $allowed\_pages)) {

include "includes/$page.php"; // Подключаем только разрешенные файлы

} else {

echo "<p>Страница не найдена.</p>"; // Сообщение об ошибке

}

} else {

// Содержимое по умолчанию (главная страница)

include "includes/home.php";

}

?>

</div>

```

#### Изменения:

1. Файлы страниц теперь находятся в папке `includes`.

2. Путь к файлам изменен: `include "includes/$page.php"`.

---

### 3. \*\*Структура проекта с `includes`\*\*

Если все файлы находятся в папке `includes`, структура будет выглядеть так:

```

/project

/includes

header.php // Шапка

footer.php // Подвал

main\_container.php // Главный блок

home.php // Главная страница

course.php // Страница учебника

profile.php // Личный кабинет

/images

logo.png

user-icon.png

index.php

styles.css

```

---

### 4. \*\*Пример содержимого файлов\*\*

#### `includes/home.php` (Главная страница):

```php

<h2>Добро пожаловать на образовательный курс!</h2>

<p>Здесь вы найдете полезные материалы для обучения.</p>

```

#### `includes/course.php` (Страница учебника):

```php

<h2>Учебник</h2>

<p>Это страница с материалами курса.</p>

```

#### `includes/profile.php` (Личный кабинет):

```php

<h2>Личный кабинет</h2>

<p>Здесь вы можете управлять своим профилем.</p>

```

---

### 5. \*\*Сравнение двух подходов\*\*

| Критерий | Отдельная папка `pages` | Все файлы в `includes` |

|------------------------|---------------------------------------------|----------------------------------------|

| \*\*Организация файлов\*\* | Логически разделены части сайта и страницы | Все файлы в одной папке, менее удобно |

| \*\*Поддержка\*\* | Легко найти и изменить конкретную страницу | Требует больше внимания при поиске |

| \*\*Безопасность\*\* | Проще защитить страницы от прямого доступа | Нужно дополнительно защищать файлы |

| \*\*Масштабируемость\*\* | Легко добавлять новые разделы | Может стать сложным при росте проекта |

---

### 6. \*\*Какой вариант выбрать?\*\*

Если проект небольшой и ты планируешь его использовать только для демонстрации (например, для дипломной работы), то можно оставить всё в папке `includes`. Это проще и быстрее.

Если ты хочешь сделать проект более профессиональным и готовишься к будущему развитию, лучше выделить страницы в отдельную папку `pages`.

---

### 7. \*\*Итог\*\*

Теперь у тебя есть два варианта:

1. \*\*С отдельной папкой `pages`\*\*:

```php

include "pages/$page.php";

```

2. \*\*Всё в папке `includes`\*\*:

```php

include "includes/$page.php";

```

Какой вариант ты выбираешь? Если остаешься с `includes`, я могу помочь доработать этот подход. 😊

# 7. главный блок начальной страницы –форматирование.

Отлично, что ты выбрала вариант с отдельной папкой `pages`! Это действительно более профессиональный подход. Теперь давай разберем, как сделать так, чтобы страницы (например, `home.php`, `course.php`, `profile.php`) открывались \*\*внутри главного блока\*\*, а не как отдельные страницы.

---

### 1. \*\*Как это работает\*\*

Главная идея заключается в том, что все страницы будут подключаться через PHP в главном блоке (`main\_container.php`). То есть:

- Пользователь переходит по ссылке, например: `http://example.com/index.php?page=course`.

- PHP проверяет параметр `page` и подключает соответствующий файл из папки `pages` внутрь `<div class="main\_\_container">`.

Таким образом, шапка (`header.php`) и подвал (`footer.php`) остаются неизменными, а только содержимое главного блока меняется.

---

### 2. \*\*Улучшенный код для `main\_container.php`\*\*

Вот как будет выглядеть код для `main\_container.php`:

```php

<div class="main\_\_container">

<?php

// Определяем, какой контент нужно показать

if (isset($\_GET['page'])) { // Используем параметр "page"

$allowed\_pages = ['home', 'course', 'profile']; // Список разрешенных страниц

$page = basename($\_GET['page']); // Защита от пути к файлу

if (in\_array($page, $allowed\_pages)) {

include "pages/$page.php"; // Подключаем только разрешенные файлы

} else {

echo "<p>Страница не найдена.</p>"; // Сообщение об ошибке

}

} else {

// Содержимое по умолчанию (главная страница)

include "pages/home.php";

}

?>

</div>

```

---

### 3. \*\*Как работают ссылки\*\*

Чтобы страницы открывались внутри главного блока, нужно правильно настроить ссылки в шапке (`header.php`) или других частях сайта. Например:

#### Пример ссылок в шапке (`header.php`):

```html

<header class="site-header">

<div class="header-container">

<!-- Логотип (переход на главную страницу) -->

<a href="index.php?page=home" class="site-logo">

<img src="images/logo.png" alt="Логотип сайта" width="30" height="30">

</a>

<!-- Название "Учебник" (переход на страницу учебника) -->

<h1 class="site-title">

<a href="index.php?page=course">Учебник</a>

</h1>

<!-- Иконка пользователя (переход в личный кабинет) -->

<a href="index.php?page=profile" class="user-icon">

<img src="images/user-icon.png" alt="Иконка пользователя" width="30" height="30">

</a>

</div>

</header>

```

#### Как это работает:

1. Каждая ссылка указывает на `index.php` с параметром `page`.

- Например: `index.php?page=course` — подключит файл `pages/course.php`.

2. Главный блок (`main\_container.php`) обрабатывает параметр `page` и подключает нужный файл.

---

### 4. \*\*Пример содержимого страниц\*\*

Вот как могут выглядеть файлы в папке `pages`:

#### `pages/home.php` (Главная страница):

```php

<h2>Добро пожаловать на образовательный курс!</h2>

<p>Здесь вы найдете полезные материалы для обучения.</p>

```

#### `pages/course.php` (Страница учебника):

```php

<h2>Учебник</h2>

<p>Это страница с материалами курса.</p>

```

#### `pages/profile.php` (Личный кабинет):

```php

<h2>Личный кабинет</h2>

<p>Здесь вы можете управлять своим профилем.</p>

```

---

### 5. \*\*Как это выглядит на сайте\*\*

1. \*\*Главная страница\*\*:

- URL: `http://example.com/index.php?page=home`

- В главном блоке отображается содержимое файла `pages/home.php`.

2. \*\*Страница учебника\*\*:

- URL: `http://example.com/index.php?page=course`

- В главном блоке отображается содержимое файла `pages/course.php`.

3. \*\*Личный кабинет\*\*:

- URL: `http://example.com/index.php?page=profile`

- В главном блоке отображается содержимое файла `pages/profile.php`.

4. \*\*Недопустимый параметр\*\*:

- URL: `http://example.com/index.php?page=secret`

- В главном блоке отображается сообщение: "Страница не найдена."

---

### 6. \*\*Стили для главного блока\*\*

Добавим базовые стили для главного блока, чтобы он выглядел аккуратно:

```css

.main\_\_container {

background-color: #fff; /\* Белый фон \*/

padding: 20px; /\* Отступы внутри блока \*/

border-radius: 8px; /\* Закругленные углы \*/

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1); /\* Тень для эффекта карточки \*/

margin: 20px auto; /\* Отступы сверху и снизу, центрирование \*/

max-width: 1200px; /\* Максимальная ширина блока \*/

}

```

---

### 7. \*\*Итог\*\*

Теперь у тебя есть:

1. \*\*Динамический главный блок\*\*, который подключает страницы из папки `pages`.

2. \*\*Ссылки в шапке\*\*, которые открывают страницы внутри главного блока.

3. \*\*Защита от уязвимостей\*\*: Только разрешенные страницы могут быть подключены.

4. \*\*Стилизованный главный блок\*\*, который выглядит современно.

Если всё понятно, можем двигаться дальше. Например, можно добавить боковое меню или начать работу с базой данных. Какой следующий шаг тебя интересует? 😊

# 8. Семантические/структурные теги в HTML5

* <article> - раздел документа, в котором описана самостоятельная единица контента;
* <aside> - контент, который находится в стороне (сайдбар, боковая колонка сайта);
* <bdi> - используется для изолирование текста от влияния направление текста родителя;
* <details> и <summary> - блок с раскрывающемся контентом;
* <dialog> - всплывающее окно;
* <figure> и <figcaption> - группировка элементов и добавление подписей к ним ([ранее](https://codebra.ru/ru/lessons-html/image/4/1) с помощью его мы группировали изображения и подпись к нему)
* <footer> - определяет нижнюю часть сайт (подвал);
* <header> - определяет верхнюю часть сайта (шапку);
* <main> - основной контент документа;
* <mark> - выделение или подсветка важного фрагмента текста;
* <meter> - отображение числового значения в заданном диапазоне;
* <nav> - ссылки навигации;
* <progress> - индикатор прогресса выполнения задачи;
* <section> - независимый блок/раздел в веб-документе;
* <time> - определение даты/времени;
* <wbr> - место для переноса строки.

<body>

<header>

Шапка

<nav>

Меню

</nav>

</header>

<main>

<section>

<h1>Заголовок первого уровня</h1>

Тест документа

</section>

</main>

<details>

<summary>Спойлер</summary>

<p>Здесь какой-то текст</p>

</details>

<dialog open>

Привет, мир!

</dialog>

<p>

Рейтинг сайта

<meter min="1" max="5" value="3"></meter>

</p>

<p>

Прогресс загрузки

<progress min="1" max="100" value="20"></progress>

</p>

<p>Словобезпробелов<wbr>невлазитнаэкран</p>

<footer>

Подвал

</footer>

</body>

</html> /

# 9. 149. Как сделать плавный переход в CSS

Когда пользователь наводит курсор на объект или нажимает на него, псевдокласс hover срабатывает мгновенно. Чтобы как-то сгладить переход, необходимо использовать анимацию. Приведем простой пример ссылки:

Код HTML

<a href = "/" class = "link">Ссылка</a>

Код CSS

.link {  
 padding: 10px 20px;  
 background: #F1F1F1;  
 color: #000;  
}  
  
.link:hover {  
 background: #AAAAAA;  
}

Теперь сгладим переход цвета фона при наведении курсора на ссылку. Для этого будем использовать три свойства: transition-property, transition-duration и transition-timing-function. Первое указывает на свойство, к которому мы применяем переход. Второе указывает продолжительность перехода. Третье указывает скорость перехода, в зависимости от времени.

Код CSS

.link:hover {  
 background: #AAAAAA;  
 transition-property: background;  
 transition-duration: 1s;  
 transition-timing-function: ease;  
}

В нашем примере устанавливается переход для свойства background, длительностью 1s и с временной функцией ease. Если необходимо установить плавный переход для всех свойств, то замените конкретное свойство на all, то есть таким образом: transition-property: all.

Теперь рассмотрим значения, которые может принимать свойство transition-timing-function. Достаточно тяжело описать работу каждого значения, поэтому просто их перечислю: ease, linear, ease-in, ease-out, ease-in-out и cubicbezier. Чтобы понять, чем они отличаются, попробуйте каждый по очереди. Для упрощения кода эти три строчки можно записать одной. Далее равнозначный пример:

Код CSS

.link:hover {  
 background: #AAAAAA;  
transition: background 1s ease;  
}

Приведенные ранее примеры небыли кроссплатформенными. Чтобы свойство transition корректно работало во всех браузерах, необходимо добавлять специальные приставки (вендорные префиксы). Вот основные:

1. -webkit- для Chrome, Safari и всех браузеров, написанных на движке WebKit;
2. -o- для Opera;
3. -moz- для Firefox;
4. -ms- для браузеров Microsoft.

Используются они следующим способом:

Код CSS

-webkit-transition: background 1s ease;  
-o-transition: background 1s ease;  
-moz-transition: background 1s ease;  
-ms-transition: background 1s ease;

Кстати, в свойстве transition можно перечислять значения через запятую, например, вот так:

Код CSS

transition: background 1s ease, color 2s linear;

Возможно, кто-то задается вопросом, почему бы не использовать JavaScript? Использовать его можно, но зачем усложнять себе жизнь, когда имеется возможность решить задачу средствами CSS.

# 10. 150. Трансформирование элементов при помощи CSS

Введение

CSS3 открыл новые возможности. Сейчас научимся трансформировать элементы, то есть будем вращать, сдвигать и наклонять элементы. Эту возможность открывает свойство transform. Значением свойства transform является функция. Вот их список и принимаемые ими значения:

1. matrix() — любое число;
2. scale(), scaleX(), scaleY() — любое число;
3. rotate() — угол (deg, grad, rad);
4. skew(), skewX(), skewY() — угол (deg, grad, rad);
5. translate(), translateX(), translateY() — единицы длины и проценты.

Пример применения свойства transform:

Код CSS

transform: scaleX(0.5);

Если необходимо использовать сразу несколько функций, то следует перечислять их через пробел:

Код CSS

transform: scaleX(0.5) rotate(20deg);

## Функция matrix

Функция matrix является комбинированной. Она имеет 6 значений: matrix(a, b, c, d, x, y).

1. Значение a: масштаб по горизонтали (если a > 1, то увеличение, иначе уменьшение);
2. Значение b: наклоняет стороны элемента по оси Y (если b > 0, то вверх, иначе вниз);
3. Значение c: наклоняет стороны элемента по оси X (если c > 0, то влево, иначе вправо);
4. Значение d: масштаб по вертикали (если d > 1, то увеличение, иначе уменьшение);
5. Значение x: смещает элемент по оси X (если x > 0, то вправо, иначе влево);
6. Значение y: смещает элемент по оси Y (если y > 0, то вниз, иначе вверх).

## Функция translate

Функции translate(), translateX(), translateY() принимают различные значения длины и проценты. Результатом ее работы является сдвиг элемента относительно обычного положения. Для функции translate() задается два значения, первое – сдвиг по оси X, второе – по Y.

Код CSS

.link:hover {  
 transform: translateY(-10px);  
}

## Функция scale

Функции scale(), scaleX(), scaleY() принимают любые числа (без единиц измерения). Числа от 0 до 1 уменьшают элемент, больше единицы – увеличивают. Для функции scale() первое значение изменяет элемент по ширине, второе по высоте. Значения записываются через запятую. Отрицательные значения отображают элемент зеркально.

Код CSS

.link:hover {  
 transform: scaleX(-1);  
}

## Функция rotate

Функция rotate поворачивает элемент на заданный угол, который может быть как положительным, так и отрицательным. Допускается указывать угол больше 360 градусов.

Код CSS

.link:hover {  
 transform: rotate(360deg);  
}

## Функция skew

Функции skew(), skewX(), skewY() деформируют стороны элемента. Деформация осуществляется относительно осей координат. Для функции skew() первым значением задается угол по оси X, вторым – по оси Y. Если задан только первый угол, то второй будет определен автоматически.

## Точка трансформации

Иногда требуется сместить точку, относительно которой происходит трансформация. По умолчанию этой точкой является центр элемента. Чтобы сместить точку трансформации, добавили свойство transform-origin, которое принимает три значения через пробел:

Код CSS

transform-origin: x y z;

Где буквы означают координаты по одноименным осям. Далее пример:

Код CSS

transform-origin: 0 50%;

## Заключение

Мы познакомились с новыми возможностями CSS3, которые упрощают разработку современного дизайна. Не забывайте использовать вендорные префиксы, которые здесь не упоминались ([о них упоминалось ранее](https://codebra.ru/ru/lessons-css/animation/1/1)). Они нужны для корректной работы свойств во всех браузерах.

# 11. 151. Знакомимся с Flexbox в CSS

## Введение

Продолжаем знакомство с CSS. Перейдем к относительно новой технологии, которая имеет достаточно широкую поддержку браузерами и значительно помогает в верстке веб-страниц. Flexbox предлагает множество инструментов для верстки сложных и в тоже время гибких макетов.

Главной идеей модуля Flexbox является дать контейнеру возможность динамически менять ширину и высоту для наилучшего заполнения пространства.

Так как Flexbox это не одно свойство, а множество, давайте для начала разберемся с понятиями. Для этого обратимся к рисунку 1.

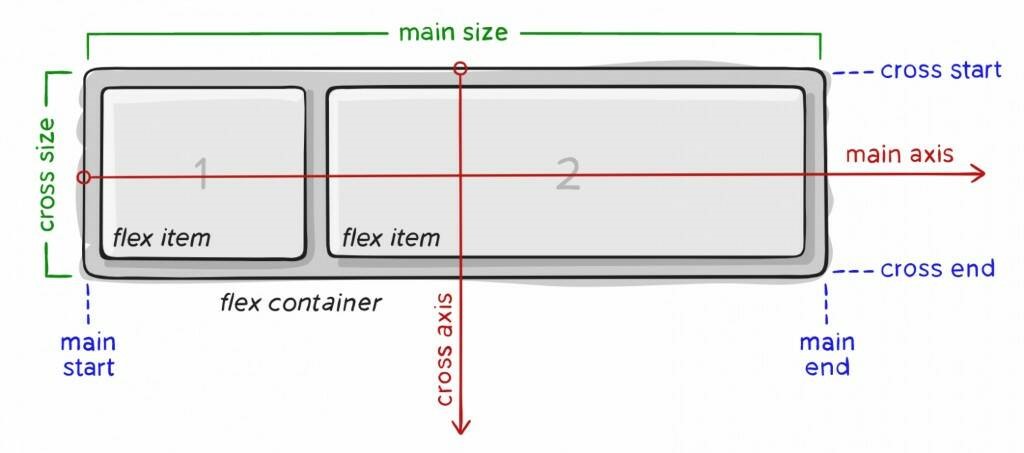


Рисунок 1. Направления в Flexbox

Рисунок взят из спецификации. Поговорим теперь о каждом понятии. Элементы могут располагаться как по main axis (главная ось от main start до main end), так и по cross axis (поперечная ось от cross start до cross end). Flex container (flex-контейнер) – родительский блок, который в последствии мы будем наделять свойством гибкости (flex). Flex item (flex-элемент) – дочерние элементы.

Остальное интуитивно понятно. Так же стоит заметить, что главная ось flex-контейнера не обязательно горизонтальная (этот момент зависит от свойства flex-direction, о чем будет далее).

Создадим пару блоков с разным по объему содержимым и начнем изучать свойства Flexbox.

Пример

<div class = "main">  
 <div class = "block">Текст Текст Текст Текст Текст Текст Текст </div>  
 <div class = "block">Текст Текст </div>  
 <div class = "block">Текст Текст Текст Текст Текст Текст Текст Текст Текст </div>  
 <div class = "block">Текст </div>  
 <div class = "block">Текст Текст Текст </div>  
</div>

И код CSS:

Пример

.main {  
 width: 500px;  
 background: #f1f1f1;  
}  
.main .block {  
 width: 50px;  
 background: #00FF00;  
 margin: 0 0 5px 5px;  
}

Получаем результат как на рисунке 2.

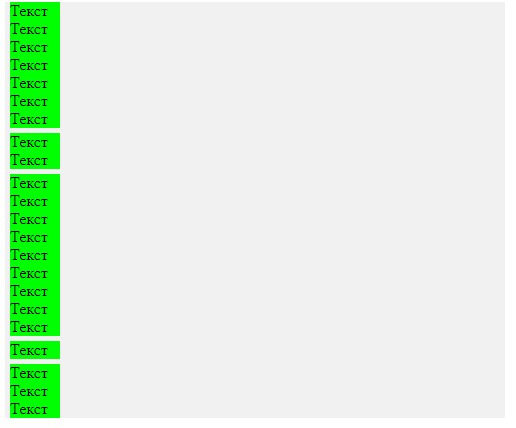


Рисунок 2. Результат верстки

Определим flex-контейнер (в нашем случае это блок с классом .main). Далее код CSS:

Пример

.main {  
 /\* предыдущий код \*/  
 display: flex;  
}

## Свойство flex-direction

По умолчанию свойство flex-direction имеет значение row, поэтому наши блоки разложились по главной оси слева направо. Свойство flex-direction имеет следующие значения:

* row – о нем уже сказали;
* row-reverse – раскладка блоков справа налево по главной оси;
* column – все то же самое, но по вертикальной (поперечной) оси сверху вниз;
* column-reverse – снизу вверх.

## Свойство flex-wrap

Следующее свойство это flex-wrap, которое по умолчанию имеет значение nowrap. Значение nowrap означает, что элементы будут стараться уместиться на одной строке. Теперь уменьшим блок .main до 200 пикселей и посмотрим результат. Как видите, дочерние блоки вышли за пределы родительского блока, так как по умолчанию у свойства flex-wrap значение nowrap. Напишем свойство flex-wrap со значением wrap (означает перенос блоков, если они не помещаются):

Пример

.main {  
 width: 200px;  
 background: #f1f1f1;  
 display: flex;  
 flex-wrap: wrap;  
}

## Свойство flex-flow

Свойства flex-direction и flex-wrap можно сократить, объединив их в одно свойство: flex-flow:

Пример

.main {  
 /\* предыдущий код \*/  
 flex-flow: row wrap;  
}

## Свойство justify-content

Следующим свойством является justify-content, которое определяет выравнивание вдоль главной оси. Данное свойство позволяет распределить оставшееся свободное место. В нашем случае, правая часть родительского контейнера пустует, рисунок 3.

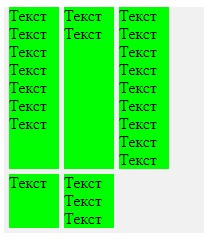


Рисунок 3

По умолчанию свойство justify-content имеет значение flex-start, что означает о выравнивании дочерних элементов по левому краю. Другие значения свойства justify-content:

* flex-end - элементы притянуты к правой части родительского блока;
* space-between - равномерное распределение элементов по главной оси, где первый элемент на линии расположен main start, а последний на main end.
* space-around - элементы расположены на одной линии с одинаковым пространством вокруг них.
* space-evenly - элементы расположены таким образом, что расстояние между двумя любыми элементами одинаковое;
* center - элементы располагаются по центру.

## Свойство align-items

Перейдем к свойству align-items. Это свойство определяет, как будут располагаться элементы относительно горизонтальной (поперечной) оси. Доступны следующие значения свойства align-items:

* stretch – элемент растягивается по высоте и заполняет все доступное пространство;
* flex-start – элементы выравниваются по верхней линии строки;
* flex-end – элементы выравниваются по нижней линии строки;
* center – элементы центрируются по горизонтальной оси;
* baseline – элементы выравниваются по их базовой линии.

## Свойство align-content

Свойство align-content выравнивает на поперечной оси в пределах flex-контейнера, когда есть свободное пространство. Это свойство не сработает, если есть только одна строка.

* flex-start – элементы сдвинуты к верхней части контейнера;
* flex-end – элементы сдвинуты к нижней части контейнера;
* stretch – элементы растягиваются и заполняют все свободное пространство;
* center – элементы выравниваются по центру контейнера;
* space-between – элементы равномерно распределяются в контейнере;
* space-around – элементы распределены с равномерным пространством вокруг;
* space-evenly – элементы растянуты на одинаковое расстояние друг от друга.

## Свойства для дочерних элементов

Перейдем к свойствам для дочерних элементов. Для каждого элемента можно задавать пропорцию, которую он будет занимать при помощи свойства flex-grow. Если задать для всех дочерних элементов значение 1:

Пример

.main .block {  
 /\* предыдущий код \*/  
 flex-grow: 1;  
}

Получится следующий результат, рисунок 4.

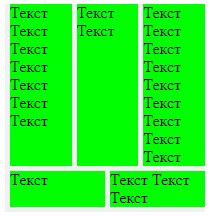


Рисунок 4

Противоположным для flex-grow, является свойство flex-shrink, которое определяет возможность сжатия элемента. Оба свойства не принимают отрицательные свойства.

Следующим свойством для дочерних элементов рассмотрим flex-basis, которое определяет размер перед заполнением оставшегося пространства по умолчанию. В качестве значения принимает длину (50%, 100px и т.д.) или ключевое слово, например, content. Значение по умолчанию у свойства flex-basis: auto.

Кстати, у этих трех свойств есть сокращение. Вместо них можно использовать одно свойство flex. Синтаксис следующий:

Пример

.main .block {  
 /\* предыдущий код \*/  
 flex: none | [ <'flex - grow'> <'flex - shrink'>? || <'flex - basis'> ]  
}

Второй и третий параметры необязательные.

Так же для отдельного элемента можно переопределить выравнивание, которое указано у родителя при помощи свойства align-items. Переопределить можно при помощи свойства align-self.

## Как упростить себе верстку?

Откройте страницу с вашим flexbox’ом, после чего откройте консоль разработчика (кнопка F12). Теперь выберите flex-контейнер, рисунок 5.



Рисунок 5

И теперь найдите CSS свойство display со значением flex. После нажмите на иконку рядом со значением flex, рисунок 6.

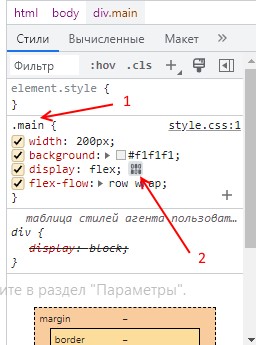


Рисунок 6

Как видите, все что мы сегодня обсуждали очень понятно визуализировано в консоли разработчика, рисунок 7.

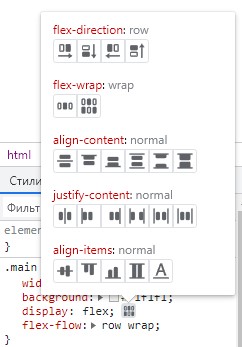


Рисунок 7. Открытая вкладка свойств для flexbox

В этом уроке мы познакомились с относительно новой технологией – Flexbox. Разобрались с базовыми понятиями, основными свойствами для управления компоновкой гибких блоков и свойствами дочерних элементов в контексте данной темы.