

**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ «СИНЕРГИЯ»**

**Факультет (Институт)** Интернет-профессий

(наименование факультета/ института)

**Направление подготовки /специальность:** 02.03.03 Математическое обеспечение и администрирование информационных систем.

(код и наименование направления подготовки /специальности)

**Профиль/специализация:** Разработка программного обеспечения (Full-stack разработка)

(наименование профиля/специализации)

**Форма обучения:** Очно-заочная

(очная, очно-заочная, заочная)

**ОТЧЕТ**

**ПО УЧЕБНОЙ ПРАКТИКЕ**

(вид практики)

**ТЕХНОЛОГИЧЕСКАЯ (ПРОЕКТНО-ТЕХНОЛОГИЧЕСКАЯ) ПРАКТИКА**

(тип практики)

**\_\_\_\_ семестр**

|  |  |  |  |
| --- | --- | --- | --- |
| **Обучающийся** |  |  |  |
|  | (ФИО) |  | (подпись) |

**Москва 20 г.**

**Практические кейсы-задачи, необходимые для оценки знаний, умений, навыков и (или) опыта деятельности по итогам практики**

| **№ п/п** | **Подробные ответы обучающегося на практические кейсы-задачи** |
| --- | --- |
| Кейс-задача № 1 | **Full-stack разработка** - это методология разработки программного обеспечения, в которой разработчики работают как на фронтенде, так и на бэкенде приложения. Она охватывает полный стек технологий, необходимых для создания и развертывания веб-приложений, включая работу с базами данных, серверной логикой, пользовательским интерфейсом и т. д. Вот основные аспекты Full-stack разработки:  **Основные понятия и принципы:**  Основные понятия и принципы Full-stack разработки включают в себя следующие аспекты:   * ***Полный стек технологий (Full-stack):*** Full-stack разработчик работает на всех уровнях разработки приложения, включая фронтенд, бэкенд и базы данных. Он имеет навыки и знания, чтобы создавать и поддерживать полноценные приложения от начала до конца. * ***Универсальность:*** Full-stack разработчики обладают широким спектром навыков и знаний, что позволяет им эффективно выполнять различные задачи в рамках проекта. Они могут работать с различными технологиями и инструментами на разных уровнях стека технологий. * ***Адаптивность:*** Full-stack разработчики способны быстро адаптироваться к изменяющимся требованиям и условиям проекта. Они могут легко переключаться между разными технологиями и задачами, обеспечивая гибкость и эффективность в работе. * ***Полное понимание приложения:*** Поскольку Full-stack разработчики работают на всех уровнях приложения, у них есть полное понимание его архитектуры, функциональности и взаимодействия с пользователем. Это позволяет им принимать обоснованные решения и эффективно решать проблемы на разных уровнях. * ***Коммуникация и сотрудничество:*** Full-stack разработчики часто взаимодействуют с другими членами команды, такими как дизайнеры, тестировщики и менеджеры продукта. Они должны быть готовы обмениваться информацией, решать проблемы и работать в команде для достижения общих целей проекта.   В целом, Full-stack разработка обеспечивает универсальность, адаптивность и полное понимание приложения, что делает разработчика более эффективным и продуктивным на всех этапах разработки.  **Роли в команде разработки:**  В команде разработки программного обеспечения (ПО) роли распределены таким образом, чтобы обеспечить эффективное выполнение всех этапов проекта. Вот основные роли в команде разработки:   * ***Full-stack разработчик:*** Это универсальный специалист, который обладает навыками как на фронтенде, так и на бэкенде приложения, а также имеет опыт работы с базами данных. Он может разрабатывать полноценное приложение от начала до конца. * ***Фронтенд-разработчик:*** Специалист, который отвечает за создание пользовательского интерфейса (UI) и его взаимодействие с пользователем. Фронтенд-разработчик использует HTML, CSS и JavaScript для создания интерфейса, который пользователи видят и взаимодействуют с ним. * ***Бэкенд-разработчик:*** Это специалист, который работает на серверной стороне приложения и отвечает за обработку запросов от клиентов, взаимодействие с базами данных, разработку бизнес-логики и обеспечение безопасности и масштабируемости приложения. * ***Специалист по базам данных:*** Этот член команды отвечает за проектирование и управление базами данных, а также за оптимизацию и обеспечение безопасности данных. Он занимается созданием таблиц, индексов, запросов и обеспечивает надежность и производительность баз данных. * ***UI/UX дизайнер:*** Это специалист, который занимается проектированием пользовательского интерфейса (UI) и созданием приятного и удобного опыта взаимодействия с пользователем (UX). Он обеспечивает эстетику, удобство использования и соответствие дизайна бренду. * ***Тестировщик (QA инженер):*** Этот член команды отвечает за тестирование приложения на различных этапах разработки, чтобы обеспечить его качество, надежность и соответствие требованиям. Он выполняет функциональное, интеграционное, модульное и другие виды тестирования. * ***Продуктовый владелец (Product Owner):*** Это представитель заказчика или бизнеса, который определяет требования к продукту, приоритизирует задачи и обеспечивает соответствие разрабатываемого продукта потребностям пользователей и бизнесу.   Роли в команде разработки могут варьироваться в зависимости от размера и сложности проекта, а также от методологии разработки, используемой командой (например, Agile, Scrum, Waterfall). Каждый член команды вносит свой вклад в процесс разработки, чтобы обеспечить успешное выполнение проекта.  **Различия между Full-stack и другими подходами к разработке программного обеспечения:**  Различия между Full-stack разработкой и другими подходами к разработке программного обеспечения могут быть существенными и касаться различных аспектов процесса разработки. Вот основные различия:   * **Область ответственности:**   + ***Full-stack разработка:*** Full-stack разработчик отвечает как за фронтенд, так и за бэкенд приложения, а также за базы данных. Он способен работать на всех уровнях технологического стека.   + ***Фронтенд-разработка:*** Фокусируется исключительно на создании пользовательского интерфейса (UI) и взаимодействии с пользователем без углубления в бэкенд и базы данных.   + ***Бэкенд-разработка:*** Специализируется на разработке серверной логики, обработке запросов и управлении данными, не затрагивая напрямую фронтенд. * **Навыки и компетенции:**   + ***Full-stack разработка:*** Требует широкого спектра навыков и знаний, включая знания фронтенд- и бэкенд-технологий, баз данных, архитектуры приложений и других аспектов разработки.   + ***Специализированные подходы:*** Позволяют разработчикам сосредоточиться на определенных аспектах разработки и стать экспертами в своей области (например, фронтенд или бэкенд). * **Уровень глубины понимания:**   + ***Full-stack разработка:*** Требует глубокого понимания как фронтенда, так и бэкенда приложения, а также их взаимодействия между собой.   + ***Специализированные подходы:*** Разработчики могут иметь более узкую специализацию и глубокое понимание конкретных аспектов разработки (например, работы с базами данных или проектирования пользовательского интерфейса). * **Гибкость и адаптивность:**   + ***Full-stack разработка:*** Позволяет разработчикам быть более гибкими и адаптивными к изменениям в проекте, так как они могут работать на разных уровнях технологического стека.   + ***Специализированные подходы:*** Могут быть менее гибкими в адаптации к изменениям, так как разработчики сосредоточены на определенных аспектах разработки. * **Коллаборация и коммуникация:**   + ***Full-stack разработка:*** Позволяет разработчикам лучше понимать и взаимодействовать с другими членами команды, так как они имеют обширное представление о всем процессе разработки.   + ***Специализированные подходы:*** Могут привести к более узкому видению и пониманию только определенных аспектов разработки, что может затруднить коммуникацию и сотрудничество в команде. |
| Кейс-задача № 2 | **JavaScript** - это высокоуровневый, интерпретируемый язык программирования, который широко используется для создания интерактивных веб-страниц и веб-приложений. Вот общее описание основных аспектов языка JavaScript:  **Синтаксис:**  JavaScript имеет синтаксис, подобный другим языкам программирования, таким как Java и C. Вот основные конструкции синтаксиса JavaScript:   * Объявление переменных: var, let, const. * Условные операторы: if, else, switch. * Циклы: for, while, do...while. * Функции: объявление с помощью function или стрелочные функции () => {}. * Объектно-ориентированный синтаксис для работы с объектами и классами. * Обработка исключений с помощью конструкции try...catch. * Работа с массивами, объектами, строками и другими встроенными типами данных.   **Типы данных:**  JavaScript имеет динамическую типизацию, что означает, что типы данных определяются автоматически во время выполнения программы. Основные типы данных в JavaScript:   * Примитивные типы данных: number, string, boolean, null, undefined. * Объекты: object, array, function. * Специальные значения: NaN, Infinity.   **Операторы:**  Операторы в языке JavaScript представляют собой символы или ключевые слова, которые выполняют операции над операндами. Операнды могут быть значениями, переменными или выражениями. Вот основные категории операторов в JavaScript:   * **Арифметические операторы:**   + +: Сложение.   + -: Вычитание.   + \*: Умножение.   + /: Деление.   + %: Остаток от деления.   + ++: Инкремент.   + --: Декремент. * **Операторы сравнения:**   + ==: Проверка на равенство (без учета типов).   + ===: Строгая проверка на равенство (с учетом типов).   + !=: Проверка на неравенство (без учета типов).   + !==: Строгая проверка на неравенство (с учетом типов).   + >: Больше.   + <: Меньше.   + >=: Больше или равно.   + <=: Меньше или равно. * **Логические операторы:**   + &&: Логическое И.   + ||: Логическое ИЛИ.   + !: Логическое НЕ. * **Операторы присваивания:**   + =: Присваивание значения переменной.   + +=, -=, \*=, /=: Присваивание с операцией. * **Операторы битового сдвига:**   + <<: Сдвиг влево.   + >>: Сдвиг вправо.   + >>>: Сдвиг вправо с заполнением нулями. * **Операторы условия:**   + ? :: Тернарный оператор. * **Операторы типов:**   + typeof: Возвращает тип операнда.   + instanceof: Возвращает true, если объект является экземпляром класса. * **Операторы для работы с строками и массивами:**   + +: Конкатенация строк.   + []: Обращение к элементам массива/свойствам объекта.   + in: Проверка наличия свойства в объекте.   + new Array(): Создание массива.   + new Object(): Создание объекта.   Это основные операторы в JavaScript, которые позволяют выполнять различные операции и контролировать логику выполнения программы.  **Функции:**  В языке JavaScript функции являются основным строительным блоком для организации кода. Они могут быть объявлены как именованные или анонимные, и использоваться для выполнения определенных действий. Вот основные аспекты функций в JavaScript:   * **Объявление функций:**   + Функции могут быть объявлены с использованием ключевого слова function или с помощью стрелочной нотации () => {}. * **Параметры и аргументы:**   + Функции могут принимать параметры, которые указываются в скобках при объявлении. Аргументы передаются функции при вызове и используются внутри функции. * **Возвращаемые значения:**   + Функции могут возвращать значения с помощью оператора return. Если оператор return не указан, функция возвращает undefined. * **Анонимные функции:**   + Анонимные функции - это функции без имени, которые могут быть присвоены переменным или использоваться как аргументы других функций. * **Стрелочные функции:**   + Стрелочные функции - это сокращенный синтаксис для объявления функций, который предоставляет более краткую запись для определения анонимных функций. * **Функции как объекты:**   + В JavaScript функции являются объектами первого класса, что означает, что они могут быть переданы как аргументы других функций, присвоены переменным и возвращены из других функций.   Функции в JavaScript играют важную роль и используются для организации кода, выполнения определенных задач и структурирования приложений.  **Классы:**  JavaScript поддерживает объектно-ориентированный подход к программированию с помощью классов. Классы используются для создания объектов с определенными свойствами и методами. Классы могут иметь конструкторы, методы и наследование. В языке программирования JavaScript классы - это синтаксический сахар над прототипами, введенный в стандарте ECMAScript 2015 (ES6), который облегчает создание и организацию объектно-ориентированного кода. Вот основные аспекты классов в JavaScript:   * **Объявление классов:**   + Классы объявляются с использованием ключевого слова class, за которым следует имя класса. Тело класса содержит методы и конструктор. * **Конструктор:**   + Конструктор - это специальный метод класса, который вызывается при создании нового объекта этого класса. Он инициализирует свойства объекта. * **Методы:**   + Методы - это функции, объявленные в теле класса, которые представляют собой действия или операции, которые может выполнить объект этого класса. * **Создание экземпляров класса:**   + Для создания новых объектов на основе класса используется ключевое слово new, за которым следует имя класса и аргументы для конструктора. * **Наследование:**   + Классы могут наследовать свойства и методы других классов с помощью ключевого слова extends. Дочерний класс может переопределять методы родительского класса или добавлять новые методы. * **Статические методы:**   + Статические методы принадлежат самому классу, а не его экземплярам. Они вызываются непосредственно на классе, а не на объекте.   Классы в JavaScript обеспечивают более удобный и понятный способ организации кода, позволяя создавать шаблоны объектов с помощью конструкторов и методов.  **Модули:**  Модули в JavaScript - это механизм организации кода, который позволяет разделить код на отдельные файлы и импортировать экспортированные значения из одного модуля в другой. Модули помогают создавать модульные и масштабируемые приложения, обеспечивая изоляцию кода и предотвращая конфликты имен. Вот основные аспекты модулей в JavaScript:   * **Создание модуля:**   + Модуль представляет собой отдельный файл с JavaScript кодом, который экспортирует определенные значения для использования в других модулях. Для этого используется ключевое слово export. * **Импорт из модуля:**   + Для использования экспортированных значений из модуля в другом модуле используется ключевое слово import. Можно импортировать одно или несколько значений из модуля. * **Импорт по умолчанию:**   + Модуль может экспортировать значение по умолчанию, которое можно импортировать без указания его имени. * **Псевдонимы импорта:**   + При импорте значения из модуля можно использовать псевдонимы для переименования имени. * **Импорт всего модуля:**   + Можно импортировать весь модуль целиком, используя ключевое слово \* as.   Модули в JavaScript позволяют организовывать код в логические блоки, что делает его более структурированным и поддерживаемым. Они также позволяют повторно использовать код и уменьшают вероятность конфликтов имен между различными частями приложения.  **Асинхронное программирование:**  Асинхронное программирование в JavaScript позволяет выполнять операции, которые занимают время, не блокируя основной поток выполнения программы. Это особенно полезно для операций ввода-вывода (I/O), таких как загрузка данных из сети или чтение файлов, которые могут занимать значительное время, но не требуют постоянного участия пользователя. Вот основные аспекты асинхронного программирования в JavaScript:   * **Callback-функции:**   + Callback-функции - это функции, которые передаются в качестве аргументов другим функциям и вызываются после завершения определенной операции. * **Промисы (Promises):**   + Промисы - это объекты, которые представляют результат асинхронной операции и позволяют управлять ее успешным выполнением или ошибкой. * **async/await:**   + async/await - это синтаксический сахар над промисами, который позволяет писать асинхронный код в более читаемой и линейной форме. * **Обратный вызов (Callback Hell):**   + Callback Hell - это ситуация, когда множество асинхронных операций вложены друг в друга, что делает код трудным для чтения и поддержки. Промисы и async/await помогают избежать этой проблемы.   Асинхронное программирование в JavaScript помогает создавать эффективные и отзывчивые приложения, обрабатывая асинхронные операции без блокировки основного потока выполнения. Это важный аспект для работы с сетевыми запросами, базами данных и другими операциями, которые требуют времени на выполнение. |
| Кейс-задача № 3 | HTML (HyperText Markup Language) - это стандартизированный язык  разметки, используемый для создания веб-страниц. Он определяет структуру и содержимое страницы с помощью различных элементов и атрибутов.  CSS (Cascading Style Sheets) - это язык таблиц стилей, используемый для оформления внешнего вида веб-страницы. Он позволяет разработчикам контролировать визуальное представление элементов HTML.  **Основы разметки, семантика:**  Основы разметки в HTML представляют собой основные концепции и элементы, используемые для создания структуры веб-страницы. Эти элементы позволяют организовать содержимое страницы и определить его структуру, иерархию и отношения между различными частями документа. Вот некоторые из основных аспектов разметки:  **Элементы и теги:**  HTML-документ состоит из различных элементов, которые определяют типы содержимого и их структуру. Элементы HTML создаются с помощью открывающих и закрывающих тегов. Примеры элементов и их тегов:   * ***Основные элементы:***   + Параграфы <p>: Используются для отображения текстовых абзацев.   + Заголовки <h1>-<h6>: Определяют уровень заголовков на странице, где <h1> обычно является наиболее важным, а <h6> - наименее важным.   + Ссылки <a>: Создают гиперссылки на другие страницы или ресурсы в интернете.   + Изображения <img>: Используются для вставки изображений на страницу.   + Списки <ul>, <ol>, <li>: Позволяют создавать неупорядоченные (<ul>) и упорядоченные (<ol>) списки, содержащие элементы списка (<li>).   + Формы <form>: Используются для сбора информации от пользователя, такой как текстовые поля, кнопки отправки и т.д. * ***Дополнительные элементы:***   + Контейнеры <div> и <span>: Используются для группировки и стилизации других элементов. <div> обычно используется для блочной группировки, а <span> - для строчной.   + Таблицы <table>, <tr>, <td>: Позволяют создавать таблицы на странице с помощью строк (<tr>) и ячеек (<td>).   + Форматирование текста <strong>, <em>, <u>: Позволяют выделять текст жирным (<strong>), курсивом (<em>) или подчеркиванием (<u>).   + Видео и аудио <video>, <audio>: Позволяют встраивать видео и аудио на страницу. * ***Теги:***   + Теги используются для определения начала и конца элемента. Например, <p> определяет начало параграфа, а </p> - его конец. Некоторые теги, такие как <img> и <input>, не имеют закрывающего тега.   HTML-элементы и теги обеспечивают основу структуры и содержимого веб-страницы, позволяя разработчикам создавать информативные и удобочитаемые веб-документы.  **Вложенность:**  Элементы могут быть вложены друг в друга, создавая иерархическую структуру документа. Например, параграф может содержать ссылку, список может содержать элементы списка, и так далее. Пример вложенности:    **Атрибуты:**  Элементы могут иметь атрибуты, которые предоставляют дополнительную информацию о содержимом элемента или определяют его поведение. Например, атрибут href у тега <a> определяет URL ссылки, а атрибут src у тега <img> указывает источник изображения. Пример использования атрибутов:    **Комментарии:**  HTML позволяет добавлять комментарии в код, которые игнорируются браузером при отображении страницы. Это полезно для добавления пояснений к коду или временного исключения определенных частей. Пример комментария:    **Семантика:**  Семантика в HTML относится к присвоению значений элементам в соответствии с их смыслом или назначением. Использование семантической разметки помогает браузерам и поисковым системам понимать структуру и содержимое страницы, что может повлиять на ее индексацию и рейтинг в поисковых результатах. Примеры семантических элементов:   * <header>: Заголовок страницы или секции. * <nav>: Навигационное меню. * <main>: Основное содержимое страницы. * <article>: Отдельная статья или контент. * <section>: Логическая секция документа. * <footer>: Нижний колонтитул страницы.   Использование семантических элементов помогает улучшить доступность и понятность страницы как для пользователей, так и для поисковых систем. Браузеры и инструменты анализа могут использовать эту информацию для оптимизации отображения и индексации веб-страницы.  **Структура страницы**  Структура страницы в HTML определяет организацию и размещение различных элементов на веб-странице. Правильная структура страницы обеспечивает удобство использования для пользователей, улучшает доступность и SEO, и облегчает поддержку и развитие веб-проекта. Вот основные компоненты структуры страницы:  **Основные элементы структуры страницы:**   * Шапка (<header>): Обычно содержит логотип, название сайта, основное меню навигации или другие важные элементы, относящиеся к верхней части страницы. * Навигация (<nav>): Содержит ссылки на разделы сайта или другие ключевые страницы. Обычно размещается в шапке или под шапкой. * Основное содержимое (<main>): Содержит основной контент страницы, такой как статьи, посты блога, карточки товаров и т. д. Этот элемент может быть только один на странице. * Боковая панель (<aside>): Содержит дополнительную информацию или элементы управления, которые не являются основным содержимым страницы. Например, виджеты, баннеры или блоки рекламы. * Футер (<footer>): Содержит информацию о копирайте, ссылки на социальные сети, контактные данные и другие важные элементы, относящиеся к нижней части страницы.   **Пример структуры страницы (обрезан):**    Пример выше демонстрирует общую структуру страницы, которая может быть настроена и дополнена в зависимости от конкретных потребностей проекта. Структура страницы помогает организовать содержимое сайта таким образом, чтобы оно было понятным и удобным для пользователей, а также легко обрабатывалось браузерами и поисковыми системами.  **Атрибуты, стили**  Атрибуты и стили играют важную роль в HTML и CSS, позволяя определять внешний вид и поведение элементов веб-страницы. Вот более подробное описание этих концепций:  **Атрибуты:**   * class: Используется для определения класса элемента, который может использоваться для стилизации или применения JavaScript. * id: Уникальный идентификатор элемента. Позволяет обращаться к элементу с помощью CSS или JavaScript. * href: Определяет URL-адрес ссылки в элементе <a>. * src: Указывает путь к файлу изображения или другому ресурсу в элементе <img>. * alt: Альтернативный текст для изображения, который отображается в случае, если изображение не может быть загружено.   **Стили:**   * ***Встроенные стили****:* Могут быть определены непосредственно внутри HTML-элемента с помощью атрибута style. Например: * ***Внутренние стили:*** Могут быть определены внутри тега <style> в разделе <head> HTML-документа. Например: * ***Внешние стили:*** Определяются в отдельном файле CSS и подключаются к HTML-документу с помощью тега <link> в разделе <head>. Например:   **Примеры использования:**   * Применение класса и идентификатора: * Определение стилей встроенным способом: * Использование внутренних стилей: * Подключение внешних стилей:   Атрибуты и стили позволяют определять внешний вид, поведение и структуру элементов веб-страницы, обеспечивая гибкость и возможность создания эффективного пользовательского интерфейса.  **Селекторы**  Селекторы в CSS - это паттерны, которые используются для выбора и стилизации элементов на веб-странице. С помощью селекторов можно определить, какие элементы будут стилизованы и какие стили будут к ним применены. Вот некоторые основные типы селекторов:   * **Элементные селекторы:** Стили применяются ко всем элементам определенного типа. Например:    + Этот селектор применяет синий цвет текста ко всем элементам <p>. * **Идентификаторные селекторы:** Стили применяются к элементу с определенным идентификатором. Например:    + Этот селектор применяет желтый цвет фона к элементу с идентификатором my-element. * **Классовые селекторы:** Стили применяются к элементам с определенным классом. Например:    + Этот селектор применяет жирный шрифт ко всем элементам с классом my-class. * **Групповые селекторы:** Позволяют объединить несколько селекторов в один правиловой блок. Например:    + Этот селектор применяет зеленый цвет текста ко всем заголовкам <h1>, <h2> и <h3>. * **Контекстные селекторы:** Позволяют выбрать элементы на основе их расположения в структуре HTML. Например:    + Этот селектор применяет центрирование текста ко всем элементам <p>, которые находятся внутри элемента <div>. * **Псевдоклассы и псевдоэлементы:** Позволяют выбирать элементы в зависимости от их состояния или позиции в документе. Например:    + Этот селектор применяет красный цвет текста к ссылкам при наведении курсора мыши.   Селекторы являются мощным инструментом для стилизации элементов на веб-странице, позволяя разработчикам точно определить, какие стили должны быть применены к различным частям документа.  **Блочная модель**  Блочная модель в CSS описывает способ, с помощью которого браузеры распределяют и отображают элементы на веб-странице. Она состоит из нескольких основных компонентов, которые определяют размеры, положение и внешний вид блочных элементов. Вот основные компоненты блочной модели:  **Основные компоненты блочной модели:**   * ***Content (Содержимое):*** Фактическое содержимое элемента, такое как текст, изображения или другие дочерние элементы. * ***Padding (Внутренний отступ):*** Пространство между содержимым элемента и его границей. Padding добавляется вокруг содержимого и может быть задан с помощью свойства padding. * ***Border (Граница):*** Линия, которая окружает содержимое и отделает его от других элементов. Граница определяется с помощью свойства border и может иметь различные стили, ширину и цвет. * ***Margin (Внешний отступ):*** Пространство вокруг элемента, которое отделяет его от других элементов на странице. Маржа задается с помощью свойства margin и определяет расстояние между элементами.   **Пример блочной модели:**    **Пример HTML-элемента с использованием блочной модели:**    **Объяснение:**   * Этот пример создает блочный элемент с классом box. * Его ширина составляет 200 пикселей, высота - 100 пикселей. * Внутренний отступ (padding) вокруг содержимого равен 20 пикселям. * Граница (border) имеет толщину 1 пиксель и черный цвет. * Внешний отступ (margin) вокруг элемента составляет 10 пикселей.   Блочная модель является основой для создания макетов и управления расположением и отступами элементов на веб-странице. Понимание этой модели помогает разработчикам эффективно стилизовать и компоновать содержимое страницы.  **Flexbox**  Flexbox (Flexible Box) - это мощный инструмент в CSS, который предоставляет гибкий способ управления распределением элементов в контейнере. Он позволяет легко создавать адаптивные и респонсивные макеты без необходимости использовать сложные техники выравнивания и позиционирования. Вот основные концепции и свойства Flexbox:  **Основные концепции Flexbox:**   * ***Контейнер (Container):*** Элемент, который содержит дочерние элементы и на который применяется свойство display: flex; для создания flex-контейнера. * ***Элементы (Items):*** Дочерние элементы внутри flex-контейнера, которые распределяются и выравниваются с помощью Flexbox. * ***Ось главного направления (Main Axis):*** Основная направленность распределения элементов в контейнере. Ось главного направления определяется свойством flex-direction и может быть горизонтальной (row) или вертикальной (column). * ***Ось перпендикулярная главной (Cross Axis):*** Перпендикулярная ось основной направленности. Направление оси перпендикулярной главной зависит от основной оси и может быть вертикальной или горизонтальной.   **Основные свойства Flexbox:**   * ***display: flex;:*** Применяется к контейнеру и создает flex-контейнер. * ***flex-direction:*** Определяет направление главной оси внутри контейнера. Варианты: row, row-reverse, column, column-reverse. * ***justify-content:*** Определяет выравнивание элементов вдоль главной оси. Варианты: flex-start, flex-end, center, space-between, space-around, space-evenly. * ***align-items:*** Определяет выравнивание элементов вдоль оси перпендикулярной главной. Варианты: flex-start, flex-end, center, baseline, stretch. * ***flex-grow:*** Устанавливает способность элемента увеличивать свой размер, чтобы занять доступное пространство в контейнере. * ***flex-shrink:*** Устанавливает способность элемента уменьшать свой размер, чтобы поместиться в доступное пространство в контейнере. * ***flex-basis:*** Определяет начальный размер элемента до распределения свободного пространства в контейнере.   **Пример использования Flexbox:**      Этот пример создает flex-контейнер с тремя дочерними элементами. Элементы будут выровнены по главной оси с равным пространством между ними и центрированы по оси перпендикулярной главной. Flexbox предоставляет мощные инструменты для создания адаптивных и управляемых макетов на веб-страницах.  Grid  CSS Grid Layout - это двумерная система разметки, которая позволяет создавать сетки с помощью рядов и колонок. Grid Layout предоставляет более мощные возможности по сравнению с Flexbox, особенно когда речь идет о создании сложных макетов с определенным порядком элементов. Вот основные концепции и свойства CSS Grid:  **Основные концепции CSS Grid:**   * ***Контейнер (Container):*** Элемент, на который применяется свойство display: grid;, чтобы создать сетку. * ***Строки (Rows):*** Горизонтальные линии, которые определяют, как элементы размещаются по вертикали внутри сетки. * ***Колонки (Columns):*** Вертикальные линии, которые определяют, как элементы размещаются по горизонтали внутри сетки. * ***Ячейки (Cells):*** Области, образованные пересечением строки и колонки, в которых размещаются элементы.   **Основные свойства CSS Grid:**   * ***display: grid;:*** Применяется к контейнеру и создает сетку. * ***grid-template-rows:*** Определяет размеры и количество строк в сетке. * ***grid-template-columns:*** Определяет размеры и количество колонок в сетке. * ***grid-gap:*** Устанавливает промежуток между ячейками сетки. * ***grid-row и grid-column:*** Определяют, на каких строках и колонках должна располагаться ячейка. * ***grid-area:*** Определяет область ячейки в сетке. * ***grid-template-areas:*** Позволяет определить именованные области в сетке, что делает ее более читаемой.   **Пример использования CSS Grid:**      Этот пример создает сетку с тремя колонками: первая имеет фиксированную ширину 100px, вторая - 200px, а третья - автоматическую ширину. Сетка также имеет две строки с высотами 50px и 100px, а между ячейками установлен промежуток в 10px. Элементы сетки (<div class="item">) заполняют ячейки и имеют серый фон и внутренний отступ. CSS Grid предоставляет мощные инструменты для создания сложных и адаптивных макетов на веб-страницах.  **Препроцессоры CSS**  Препроцессоры CSS - это инструменты, которые добавляют дополнительные функции и возможности к стандартному CSS. Они предоставляют более удобный и мощный способ написания стилей, позволяя использовать переменные, вложенные правила, миксины, операции и другие функции, которых нет в обычном CSS. Вот некоторые популярные препроцессоры CSS:  **Основные препроцессоры CSS:**   * ***Sass (Syntactically Awesome Stylesheets):*** Это один из самых популярных препроцессоров CSS. Sass предоставляет множество полезных функций, таких как переменные, вложенные правила, миксины, наследование и многое другое. Он использует расширение .scss или .sass. * ***Less:*** Этот препроцессор CSS также предоставляет множество функций, подобных Sass, таких как переменные, вложенные правила и миксины. Less использует расширение .less. * ***Stylus:*** Это еще один препроцессор CSS с широкими возможностями. Stylus отличается своим лаконичным синтаксисом и поддержкой дополнительных функций. Он использует расширение .styl.   **Основные функции препроцессоров CSS:**   * ***Переменные:*** Позволяют определять и использовать переменные для значений цветов, размеров шрифтов, отступов и других стилей. * ***Вложенные правила:*** Позволяют вкладывать CSS-правила внутрь других, делая код более читаемым и организованным. * ***Миксины:*** Позволяют определять и повторно использовать наборы стилей, что упрощает создание общих шаблонов дизайна. * ***Функции и операции:*** Позволяют выполнять математические операции и другие манипуляции с данными в CSS. * ***Импорт и включение файлов:*** Позволяют разделять стили на несколько файлов и включать их в основной файл стилей для удобства управления.   **Пример использования переменных в Sass:**    **Пример использования миксина в Less:**    Препроцессоры CSS помогают улучшить процесс написания стилей, делая код более читаемым, гибким и поддерживаемым. Они позволяют экономить время и силы разработчиков, упрощая создание и поддержку стилей для веб-проектов. |
| Кейс-задача № 4 | Реализованный кейс находится по ссылке:  https://github.com/white-rabbit-1-sketch/synergy/tree/master/semester\_4/practice/case\_4 |
| Кейс-задача № 5 | Реализованный кейс находится по ссылке:  https://github.com/white-rabbit-1-sketch/synergy/tree/master/semester\_4/practice/case\_5 |

|  |  |
| --- | --- |
| Дата: \_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) (ФИО обучающегося) |