Базовые конструкции:

1. Переменные:

• Создайте переменную с помощью let и присвойте ей значение. Затем измените это значение и выведите его в консоль.

2. Условия:

• Напишите условие, которое будет проверять, является ли введенное число четным или нечетным, и выводить соответствующее сообщение.

3. Циклы:

• Используйте цикл for для вывода всех четных чисел от 0 до 20 в консоль.

4. Строки:

• Напишите функцию, которая принимает строку и возвращает ее перевернутой (например, "мир" -> "рим").

5. Массивы:

• Создайте массив чисел и напишите функцию, которая возвращает сумму всех элементов массива.

6. Функции:

• Напишите функцию, которая принимает два аргумента и возвращает наибольший из них.

7. Стрелочные функции:

• Перепишите функцию из предыдущего задания в виде стрелочной функции.

8. Объекты:

• Создайте объект car с полями brand, model и year. Добавьте метод, который выводит информацию о машине в формате "Марка: ..., Модель: ..., Год: ...".

9. Работа с массивами:

• Используйте метод filter, чтобы создать новый массив, содержащий только четные числа из исходного массива.

10. Деструктуризация:

• Создайте объект с информацией о книге (название, автор, год издания) и используйте деструктуризацию для извлечения этих значений в отдельные переменные.

11. Шаблонные строки:

• Используя шаблонные строки, создайте строку, которая выводит текущее время в формате "Часы: ..., Минуты: ...".

12. Замыкания:

• Реализуйте счетчик, который увеличивает значение на заданное число при каждом вызове (например, если передано 5, то сначала 5, потом 10, потом 15 и так далее).

Объектно-ориентированное программирование:

1. Класс:

• Создайте класс Person, который будет иметь свойства firstName и lastName. Добавьте метод getFullName(), который возвращает полное имя человека.

2. Наследование:

• Создайте класс Vehicle с полями make и model. Затем создайте класс Car, который наследует Vehicle и добавляет поле year. Создайте метод в Car, который выводит информацию о машине в формате "Марка: ..., Модель: ..., Год: ...".

3. Инкапсуляция:

• Создайте класс BankAccount с приватным свойством balance. Добавьте методы для депозита (deposit(amount)) и снятия (withdraw(amount)), которые изменяют баланс, и метод для получения текущего баланса (getBalance()).

4. Полиморфизм:

• Создайте классы Bird и Fish, которые оба имеют метод move(). Для птицы метод должен выводить "летит", а для рыбы — "плавает". Создайте массив животных, содержащий объекты обоих классов, и вызовите метод move() для каждого из них.

5. Создание объектов:

• Используя класс Dog из вашего примера, создайте несколько объектов собак с разными именами, возрастом и породами. Выведите информацию о каждой собаке с помощью метода speak().

Обработка Ошибок:

1. Задача на try блок:

• Напишите функцию divide(a, b), которая принимает два числа и возвращает результат их деления. Используйте блок try для выполнения деления. Если b равно нулю, выбросьте ошибку с сообщением "Деление на ноль невозможно". Проверьте работу функции с различными значениями a и b.

2. Задача на catch блок:

• Создайте функцию parseJSON(jsonString), которая принимает строку в формате JSON и возвращает соответствующий объект. Используйте блок catch для обработки ошибок парсинга. Если строка не может быть распознана как JSON, выводите сообщение об ошибке "Ошибка парсинга JSON".

DOM и BOM:

1. Придумайте демонстрационный код на порядок выполнения событий.

2. Придумайте демонстрационный код на остановку всплытия: Используйте метод stopPropagation().

3. Придумайте демонстрационный код на добавление новых элементов и обработчиков:

4. Придумайте демонстрационный код на использование атрибута onclick.

Формы:

1. Добавление валидации поля "Подтверждение пароля":

• Добавьте на форму поле для подтверждения пароля.

• Убедитесь, что значение в этом поле совпадает с введенным паролем.

• Если значения не совпадают, выведите сообщение об ошибке.

2. Добавление валидации для номера телефона:

• Добавьте на форму поле для ввода номера телефона.

• Создайте регулярное выражение для проверки формата номера телефона (например, формат "+1234567890").

• Выведите сообщение об ошибке, если номер телефона не соответствует формату.

3. Показать/скрыть пароль:

• Добавьте чекбокс, который позволяет пользователю показывать или скрывать введенный пароль.

• Реализуйте функциональность, которая изменяет тип поля пароля с "password" на "text" и обратно при изменении состояния чекбокса.

4. Динамическая проверка ввода:

• Реализуйте динамическую проверку ввода, которая будет проверять правильность заполнения полей формы по мере ввода данных (например, используя событие input).

• Отображайте сообщение об ошибке в реальном времени, если данные не соответствуют требованиям.

5. Отправка данных формы на сервер:

• После успешной валидации формы отправьте данные на сервер с помощью fetch или XMLHttpRequest.

• Обработайте ответ сервера и выведите сообщение о результате отправки (успех или ошибка).

6. Локализация сообщений об ошибках:

• Реализуйте поддержку нескольких языков для сообщений об ошибках.

• Позвольте пользователю выбирать язык интерфейса и выводите сообщения об ошибках на выбранном языке.

7. Стилизация сообщений об ошибках:

• Используйте CSS для стилизации сообщений об ошибках.

• Сделайте так, чтобы сообщения выглядели более заметными и привлекали внимание пользователя.

Cookie:

1. Создание и получение нескольких cookie:

• Создайте несколько cookie с разными именами и значениями (например, user, sessionID, preferences).

• Напишите функцию для получения всех значений этих cookie и выведите их в консоль.

2. Обновление значения cookie:

• Создайте cookie с именем theme и значением light.

• Напишите функцию для обновления значения этого cookie на dark.

• Проверьте, что значение было успешно обновлено, используя функцию получения cookie.

3. Установка cookie с ограниченной областью видимости:

• Создайте cookie, доступный только для определенного пути (например, /admin).

• Попробуйте получить значение этого cookie из другого пути и убедитесь, что это невозможно.

4. Работа с датой истечения cookie:

• Создайте cookie с коротким сроком действия (например, 10 секунд).

• Напишите скрипт, который проверяет наличие этого cookie через 15 секунд после его установки и выводит сообщение о его отсутствии.

5. Функция удаления всех cookie:

• Напишите функцию, которая удаляет все cookie, установленные на текущем сайте.

• Проверьте работу этой функции, установив несколько cookie и вызвав ее.

6. Сохранение пользовательских настроек:

• Создайте форму на странице, позволяющую пользователю выбрать тему (например, светлая или темная) и язык.

• Сохраните эти настройки в cookie и примените их при загрузке страницы.

7. Секьюрные cookie:

• Изучите, как устанавливать secure и HttpOnly флаги для cookie.

• Объясните, как эти флаги могут повысить безопасность работы с cookie.

Особенности HTML5:

Local Storage

1. Сохранение и загрузка пользовательских данных:

• Создайте простую форму с полями для ввода имени и электронной почты.

• Сохраните введенные данные в Local Storage при отправке формы.

• При загрузке страницы автоматически заполняйте форму данными из Local Storage, если они существуют.

2. Управление списком задач:

• Создайте приложение для управления списком задач (todo list).

• Реализуйте возможность добавления, удаления и отметки задач как выполненных.

• Сохраните состояние списка задач в Local Storage, чтобы оно сохранялось при перезагрузке страницы.

Geolocation API

3. Определение и отображение местоположения на карте:

• Используйте Geolocation API для получения координат пользователя.

• Интегрируйте карту (например, Google Maps или OpenStreetMap) и отметьте на ней текущее местоположение пользователя.

4. Погодное приложение:

• Используйте Geolocation API для определения текущего местоположения пользователя.

• Получите данные о погоде для этого местоположения из открытого API (например, OpenWeatherMap) и отобразите их на странице.

Canvas API

5. Рисование простых фигур:

• Создайте функцию, которая рисует круги разных цветов и размеров в случайных местах на холсте при каждом обновлении страницы.

6. Простая игра на Canvas:

• Создайте простую игру, например, Pong или Snake, используя Canvas API.

• Реализуйте управление игрой с помощью клавиатуры и добавьте счетчик очков.

7. Редактор изображений:

• Загрузите изображение на холст и предоставьте пользователю возможность изменять его (например, изменять яркость или контрастность).

• Реализуйте возможность сохранения измененного изображения.

JSON:

1. Сериализация и десериализация сложного объекта:

• Создайте объект, содержащий вложенные объекты и массивы. Например, объект пользователя с адресами и списком друзей.

• Сериализуйте этот объект в строку JSON.

• Десериализуйте строку обратно в объект и убедитесь, что данные не потеряны.

2. Изменение данных в JSON:

• Дана строка JSON, представляющая информацию о книге (название, автор, год издания).

• Напишите функцию, которая принимает строку JSON, изменяет год издания на текущий и возвращает обновленную строку JSON.

3. Фильтрация данных из JSON:

• У вас есть массив объектов в формате JSON, представляющий список студентов с их именами и оценками.

• Напишите функцию, которая принимает этот массив и возвращает новый массив только с теми студентами, у которых оценка выше определенного значения.

4. Обработка ошибок при парсинге:

• Создайте функцию, которая принимает строку JSON и пытается ее распарсить.

• Если строка некорректна, функция должна вернуть сообщение об ошибке вместо выбрасывания исключения.

5. Сравнение объектов до и после сериализации:

• Создайте функцию, которая принимает объект, сериализует его в JSON и затем десериализует обратно.

• Проверьте, что исходный объект и полученный после десериализации идентичны (учтите, что методы объектов не сохраняются в JSON).

6. Локальное хранилище и JSON:

• Напишите функцию, которая сохраняет объект настроек пользователя в Local Storage в формате JSON.

• Реализуйте функцию загрузки этих настроек из Local Storage и восстановления их в виде объекта.

Асинхронные запросы:

Задачи с использованием XMLHttpRequest

1. Получение и отображение данных:

• Используйте XMLHttpRequest для получения списка пользователей с "https://jsonplaceholder.typicode.com/users".

• Отобразите имена пользователей в виде списка на веб-странице.

2. Обработка ошибок:

• Измените URL запроса на несуществующий ресурс и добавьте обработку ошибок.

• Выведите сообщение об ошибке на страницу, если запрос завершился неудачно.

3. POST-запрос:

• Используйте XMLHttpRequest для отправки POST-запроса на "https://jsonplaceholder.typicode.com/posts".

• Отправьте объект с данными (например, заголовок и тело поста) и выведите ответ сервера в консоль.

Задачи с использованием Fetch API

4. Получение и фильтрация данных:

• Используйте Fetch API для получения списка постов с "https://jsonplaceholder.typicode.com/posts".

• Отфильтруйте посты, оставив только те, у которых userId равен 1, и отобразите их заголовки на странице.

5. Асинхронная функция с обработкой ошибок:

• Создайте асинхронную функцию для получения комментариев с "https://jsonplaceholder.typicode.com/comments".

• Добавьте обработку ошибок и выведите сообщение об ошибке на страницу, если запрос завершился неудачно.

6. Отправка данных с Fetch API:

• Используйте Fetch API для отправки POST-запроса на "https://jsonplaceholder.typicode.com/posts".

• Отправьте JSON-объект с данными (например, заголовок и тело поста) и выведите ответ сервера в консоль.

Дополнительные задачи

7. Сравнение XMLHttpRequest и Fetch API:

• Реализуйте одну и ту же функциональность (например, получение списка пользователей) как с помощью XMLHttpRequest, так и Fetch API.

• Сравните кодовые фрагменты и сделайте выводы о различиях в использовании этих подходов.

8. Интеграция с интерфейсом пользователя:

• Создайте форму для добавления нового поста.

• При отправке формы используйте либо XMLHttpRequest, либо Fetch API для отправки данных на сервер.

• Обновите список постов на странице после успешной отправки данных.