МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

(КАФЕДРА 43)

|  |
| --- |
| ОТЧЕТ ЗАЩИЩЁН С ОЦЕНКОЙ  РУКОВОДИТЕЛЬ |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ассистент |  |  |  |  |  | Ю. В. Ветрова |
| (должность, учёная степень, звание) |  | (подпись) |  | (дата защиты) |  | (инициалы, фамилия) |

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ПРАКТИЧЕСКОМУ ЗАДАНИЮ №2

Практические навыки работы с DOM и хранением данных в JavaScript

по дисциплине: [JavaScript, его библиотеки и фреймворки в Frontend-разработке](https://lms.guap.ru/course/view.php?id=1099" \l "section-1" \o "JavaScript, его библиотеки и фреймворки в Frontend-разработке)

|  |
| --- |
| РАБОТУ ВЫПОЛНИЛ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4236 |  |  |  | Л. Мвале |
|  |  |  | подпись, дата |  | инициалы, фамилия |

1. **Цель**

Цель работы – формирование практических навыков работы с асинхронным кодом в языке JavaScript, используя callback-функции и объект Promise.

1. **Задание**

1. Напишите функцию `filterArray`, которая принимает массив чисел и callback-функцию. Функция `filterArray` должна вызывать callback функцию для каждого элемента массива и возвращать новый массив, содержащий только те элементы, для которых callback-функция вернула `true`.

Напишите 2 примера применения этой функции. К примеру, для фильтрации четных и нечетных значений массива.

2. Напишите асинхронную функцию `fetchData`, которая принимает URL в качестве параметра и возвращает Promise. Функция должна использовать `fetch` для получения данных с указанного URL. Если запрос прошел успешно (статус ответа 200), Promise должен быть разрешен с полученными данными в виде строки. Если запрос не удался (любой другой статус), Promise должен быть отклонен с сообщением об ошибке.

Пример использования функции:

fetchData('<https://api.example.com/data'>)

  .then(data => {

    console.log('Данные:', data);

  })

  .catch(error => {

    console.error('Ошибка:', error);

  });

скриншоты



Script.js Приложение

// Функция filterArray принимает массив и функцию обратного вызова (callback).

// Она проходит по массиву и добавляет в новый массив только те элементы, для которых callback возвращает true.

*function* filterArray(*arr*, *callback*) {

*const* result = [];

    for (*let* i = 0; i < arr.length; i++) {

        if (callback(arr[i])) {

            result.push(arr[i]);

        }

    }

    return result;

}

// Функция isEven проверяет, является ли число четным.

*const* isEven = (*num*) *=>* num % 2 === 0;

// Исходный массив чисел

*const* numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// Фильтруем четные числа с помощью filterArray

*const* evenNumbers = filterArray(numbers, isEven);

console.log('Even numbers:', evenNumbers); // Ожидаемый вывод: [2, 4, 6, 8, 10]

console.log(""); // Пустая строка для разделения вывода

// Функция isOdd проверяет, является ли число нечетным.

*const* isOdd = (*num*) *=>* num % 2 !== 0;

// Фильтруем нечетные числа

*const* oddNumbers = filterArray(numbers, isOdd);

console.log('Odd numbers:', oddNumbers); // Ожидаемый вывод: [1, 3, 5, 7, 9]

console.log(""); // Пустая строка для разделения вывода

// Функция fetchData принимает URL и возвращает Promise.

// Она выполняет HTTP-запрос с помощью fetch и обрабатывает ответ.

*function* fetchData(*url*) {

    return new *Promise*((*resolve*, *reject*) *=>* {

        fetch(url)

            .then(*response* *=>* {

                if (response.status === 200) { // Если запрос успешен (статус 200)

                    return response.text(); // Преобразуем ответ в строку

                } else {

                    throw new Error(`Ошибка запроса: статус ${response.status}`); // Генерируем ошибку для других статусов

                }

            })

            .then(*data* *=>* resolve(data)) // Разрешаем Promise с полученными данными

            .catch(*error* *=>* reject(error.message)); // Отклоняем Promise в случае ошибки

    });

}

// Пример использования fetchData, который ДОЛЖЕН РАБОТАТЬ (корректный URL)

fetchData('https://jsonplaceholder.typicode.com/posts/1') // Пример запроса

    .then(*data* *=>* {

        console.log('Данные:', data);

    })

    .catch(*error* *=>* {

        console.error('Ошибка:', error);

    });

console.log(""); // Пустая строка для разделения вывода

// Пример использования fetchData, который НЕ БУДЕТ РАБОТАТЬ (некорректный URL)

fetchData('https://api.example.com/data')

    .then((*data*) *=>* {

        console.log('Data:', data);

    })

    .catch((*error*) *=>* {

        console.error('Error:', error);

    });

1. **Выводы**

В ходе работы были приобретены практические навыки работы с асинхронным кодом в JavaScript. Реализованы функции с использованием callback и Promise для обработки данных. Продемонстрированы методы фильтрации массива и асинхронного запроса. Эти знания позволяют эффективно управлять асинхронными операциями и обрабатывать результаты без блокировки выполнения кода.