Частное учреждение образования

«Колледж бизнеса и права»

|  |
| --- |
| УТВЕРЖДАЮ  Заведующий  методическим кабинетом  \_\_\_\_\_\_\_\_\_\_Е.В.Паскал  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ |

|  |  |
| --- | --- |
| Специальность:  2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Скриптовые языки программирования» |

**Лабораторная работа № 26**

**Инструкционно-технологическая карта**

Тема: Работа с потоками Async, Await.

Цель: сформировать навыки по работе с асинхронными функциями.

Время выполнения: 2 часа

# Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.
2. Реализовать алгоритм решения задачи.
3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.
4. Ответить на контрольные вопросы.

# Теоретические сведения

Существует специальный синтаксис для работы с промисами, который называется «**async/await**». Он удивительно прост для понимания и использования.

## Асинхронные функции

Начнём с ключевого слова **async**. Оно ставится перед функцией, вот так:

async function f() {

    return 1;

}

У слова **async** один простой смысл: эта функция всегда возвращает промис. Значения других типов оборачиваются в завершившийся успешно промис автоматически.

Например, эта функция возвратит выполненный промис с результатом 1:

async function f() {

    return 1;

}

f().then(alert); // 1

Можно и явно вернуть промис, результат будет одинаковым:

async function f() {

    return Promise.resolve(1);

}

f().then(alert); // 1

Так что ключевое слово **async** перед функцией гарантирует, что эта функция в любом случае вернёт промис. Но это ещё не всё. Есть другое ключевое слово – **await**, которое можно использовать только внутри **async**-функций.

## Await

Синтаксис:

// работает только внутри async–функций

let value = await promise;

Ключевое слово **await** заставит интерпретатор JavaScript ждать до тех пор, пока промис справа от **await** не выполнится. После чего оно вернёт его результат, и выполнение кода продолжится.

В этом примере промис успешно выполнится через 1 секунду:

async function f() {

    let promise = new Promise((resolve, reject) => {

        setTimeout(() => resolve("готово!"), 1000)

    });

    let result = await promise; // будет ждать, пока промис не выполнится (\*)

    alert(result); // "готово!"

}

f();

В данном примере выполнение функции остановится на строке (\*) до тех пор, пока промис не выполнится. Это произойдёт через секунду после запуска функции. После чего в переменную **result** будет записан результат выполнения промиса, и браузер отобразит **alert**-окно «готово!».

Обратите внимание, хотя **await** и заставляет **JavaScript** дожидаться выполнения промиса, это не отнимает ресурсов процессора. Пока промис не выполнится, JS-движок может заниматься другими задачами: выполнять прочие скрипты, обрабатывать события и т.п.

По сути, это просто «синтаксический сахар» для получения результата промиса, более наглядный, чем **promise.then**.

**await нельзя использовать в обычных функциях.**

## Обработка ошибок

Когда промис завершается успешно, **await** **promise** возвращает результат. Когда завершается с ошибкой – будет выброшено исключение. Как если бы на этом месте находилось выражение **throw**.

Такой код:

async function f() {

    await Promise.reject(new Error("Упс!"));

}

Делает то же самое, что и такой:

async function f() {

    throw new Error("Упс!");

}

Но есть отличие: на практике промис может завершиться с ошибкой не сразу, а через некоторое время. В этом случае будет задержка, а затем **await** выбросит исключение.

Такие ошибки можно ловить, используя **try...catch**, как с обычным **throw**:

async function f() {

    try {

        let response = await fetch('http://no-such-url');

    } catch (err) {

        alert(err); // TypeError: failed to fetch

    }

}

f();

В случае ошибки выполнение **try** прерывается и управление прыгает в начало блока **catch**. Блоком **try** можно обернуть несколько строк:

async function f() {

    try {

        let response = await fetch('/no-user-here');

        let user = await response.json();

    } catch (err) {

        // перехватит любую ошибку в блоке try: и в fetch, и в response.json

        alert(err);

    }

}

f();

Если у нас нет **try..catch**, асинхронная функция будет возвращать завершившийся с ошибкой промис (в состоянии **rejected**). В этом случае используется метод .**catch** промиса, чтобы обработать ошибку:

async function f() {

    let response = await fetch('http://no-such-url');

}

// f() вернёт промис в состоянии rejected

f().catch(alert); // TypeError: failed to fetch // (\*)

Если забыть добавить **.catch**, то будет сгенерирована ошибка «Uncaught promise error» и информация об этом будет выведена в консоль. Такие ошибки можно поймать глобальным обработчиком.

## Promise ALL

С промисами можно выполнять несколько промисов параллельно с помощью метода Promise.all().

function pause500ms() {

  return new Promise((res) => setTimeout(res, 500));

}

const promise1 = pause500ms();

const promise2 = pause500ms();

Promise.all([promise1, promise2]).then(() => {

  console.log("Я буду выведен через 500 миллисекунд");

});

Если просто перечислить функции, ожидающие в последовательности, они будут выполняться последовательно, так как **await** приостанавливает выполнение остальной части функции.

async function inSequence() {

  await pause500ms();

  await pause500ms();

  console.log("Я буду выведен через 1000 миллисекунд");

}

Это займет 1000 миллисекунд, так как второе ожидание не запустится, пока не завершится первое. Чтобы это обойти, необходимо обращаться к функциям следующим образом:

async function inParallel() {

  const await1 = pause500ms();

  const await2 = pause500ms();

  await await1;

  await await2;

  console.log("Я буду выведен через 500 миллисекунд");

}

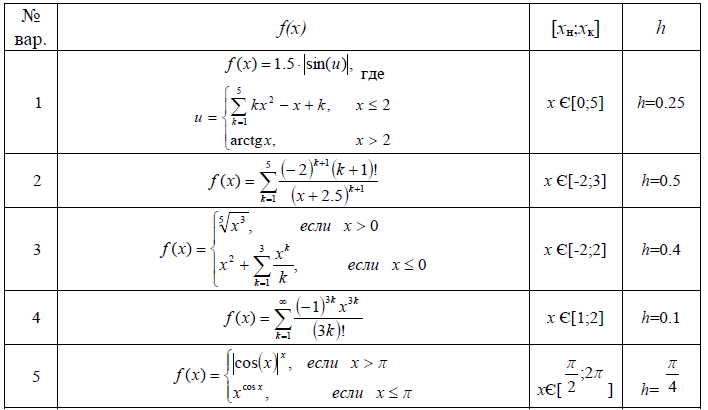
Это займет всего 500 миллисекунд, потому что обе функции pause500ms() выполняются одновременно.

# Контрольные вопросы

1. Что делает ключевое слово async?
2. Что делает ключевое слово await?
3. Что возвращают функции с использованием async/await?
4. Приведите пример обработки ошибки в async\await функции.
5. Какие ошибки выводятся в консоль, если возникают необработанные исключения?

# Задания

Задание 1. Написать асинхронную функцию для решения уравнения, которая будет отдавать свой результат спустя случайное время от 1 до 10 секунд. Вызвать одновременное несколько экземпляров функций с разными входными параметрами.



# Литература

**Диков, А.В.** Клиентские технологии веб программирования: JavaScript и DOM: учебное пособие / А.В. Диков. – СПб: Лань, 2020 – 124 с.

**Читанамбри, Кирупа.** Изучаем React / Кирупа Читанамбри – 2-е изд. – М.: Эксмо, 2019. 368 с.

**Хавербеке, Марейн.** Выразительный JavaScript. Современное веб-программирование. / Марейн Хавербеке – 3-е изд. – СПб.: Питер, 2019. – 480 с.:

Преподаватель Рогалевич А.В.

Рассмотрено на заседании цикловой

комиссии программного обеспечения

информационных технологий №10

Протокол № \_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Председатель ЦК\_\_\_\_\_\_\_\_ В.Ю.Михалевич