Лабораторная работа 03б

СТРWП

**Задание 01**

1. Разработайте функцию ***firstJob***, которая будет возвращать Promise.
2. Promise должен разрешаться успешно со значением «Hello World» через 2 секунды после вызова функции ***firstJob***.
3. Вызовите функцию ***firstJob*** и обработайте результат двумя способами: с помощью обработчиков Promise и с помощью конструкции async/await c try/catch.

**Задание 02**

1. Разработайте функцию ***secondJob***, которая будет возвращать Promise.
2. Promise должен отклоняется с сообщением об ошибке через 3 секунды.
3. Вызовите функцию ***secondJob*** и обработайте результат двумя способами: с помощью обработчиков Promise и с помощью конструкции async/await c try/catch.

**Задание 03**

1. Разработайте еще одну функцию ***thirdJob***, принимающую параметр data и возвращающую Promise.
2. Если data не является числом, ***немедленно*** вернуть отклоненный Promise со значением «error».
3. Если data является нечетным числом, вернуть через 1 секунду успешно разрешенный Promise со значением «odd».
4. Если data является четным числом, вернуть через 2 секунды отклоненный Promise со значением «even».
5. Вызовите функцию ***thirdJob*** и обработайте результат двумя способами: с помощью обработчиков Promise и с помощью конструкции async/await c try/catch.

**Задание 04**

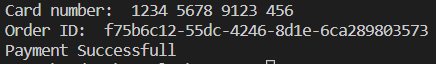
1. Разработайте функцию ***createOrder***, в которой будет создаваться Promise. Эта функция должна принимать в качестве параметра номер карты покупателя, проверять ее, а также назначать идентификатор заказу.

А) Если карта покупателя невалидна, то отклонять Promise с ошибкой «Card is not valid».

Б) Если же карта прошла проверку, то генерировать номер заказа (например, с помощью uuid) и успешно разрешать Promise с этим номером спустя 5 сек.

1. Для проверки карты необходимо создать функцию ***validateCard***. Она должна принимать номер карты, выводить его на консоль и рандомно возвращать true или false.
2. Также разработайте функцию ***proceedToPayment***, которая должна вызываться после ***createOrder***, если проверка карты прошла успешно. В ней необходимо принимать номер заказа, выводить его на консоль и возвращать Promise, который рандомно разрешается либо с успешным значением «Payment successfull», либо с ошибкой «Payment failed».
3. Вызовите функции ***createOrder*** и ***proceedToPayment*** в правильном порядке и обработайте результат двумя способами: с помощью обработчиков Promise и с помощью конструкции async/await c try/catch. Должны получаться следующие результаты:







**Задание 05**

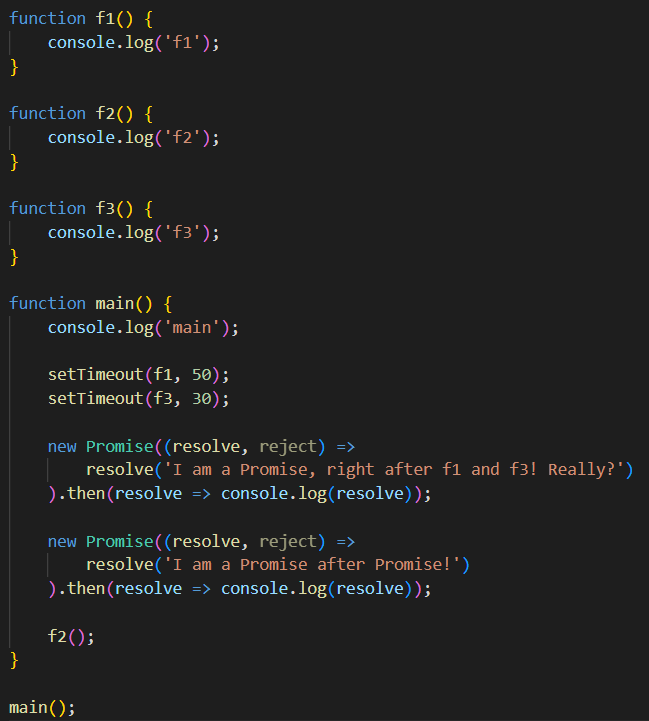
1. Разработайте три отдельные функции для вычисления квадрата, куба и четвертой степени заданного числа. Каждая из функций должна возвращать Promise, который либо разрешается с вычисленным значением, либо отклоняется с сообщением об ошибке, если ввод недействителен.
2. Далее используйте ***Promise.all()*** для вычисления этих функций.
3. Обработайте результат с помощью обработчиков Promise. Протестируйте работу с правильным и неправильным вводом.

**Задание 06**

1. Используйте задание 5. Добавьте к каждой функции разрешение Promise спустя некоторый промежуток времени.
2. Используйте ***Promise.race()*** для вычисления математических функций.
3. Замените использование на ***Promise.any()*** для получения первого Promise, который разрешается.

**Задание 07**

1. Перепишите код со скриншота ниже и выполните его. Объясните полученный результат.
2. На примере этого кода объясните, как работает Event Loop.



**Задание 08.** Ответьте следующие на вопросы.

1. Что такое **callback**?
2. В чем минусы использования коллбэков? Какие есть способы их решения?
3. Что такое **Promise** и как он работает?
4. В каких **состояниях** может находиться Promise?
5. Как изменить состояние Promise?
6. Как изменить значение Promise?
7. Что такое **цепочки промисов** и как они работает?
8. Назовите два способа обработки ошибок в Promise.
9. Для чего нужен метод **Promise.all()**?
10. В чем отличия методов **Promise.race()** и **Promise.any()**?
11. Что такое **async/await**?

Callback - это функция, которая передается в другую функцию в качестве аргумента и вызывается после выполнения определенных операций.

Минусы использования коллбэков:

Коллбэки могут привести к callback hell - ситуации, когда коллбэки вложены друг в друга настолько глубоко, что код становится трудным для чтения и понимания.

Коллбэки могут привести к race conditions - ситуациям, когда результаты операций зависят от порядка их выполнения.

Коллбэки могут привести к непредсказуемому поведению при возникновении ошибок.

Способы решения:

Использование Promise.

Использование async/await.

Promise - это объект, который представляет результат асинхронной операции. Он может находиться в трех состояниях: pending, fulfilled и rejected. Promise позволяет обрабатывать результаты асинхронных операций без использования коллбэков.

Promise может находиться в трех состояниях:

pending: начальное состояние, когда Promise еще не выполнен.

fulfilled: состояние, когда Promise успешно выполнен.

rejected: состояние, когда Promise завершился с ошибкой.

Состояние Promise можно изменить только один раз. Это происходит при вызове метода resolve() или reject().

Значение Promise можно изменить только один раз. Это происходит при вызове метода resolve() или reject().

Цепочка промисов - это последовательность асинхронных операций, которые выполняются одна за другой. Каждая операция возвращает новый Promise, который используется для следующей операции в цепочке.

Два способа обработки ошибок в Promise:

Использование метода catch().

Использование блока try...catch внутри функции, которая возвращает Promise.

Метод Promise.all() используется для выполнения нескольких асинхронных операций параллельно и ожидания их завершения.

Методы Promise.race() и Promise.any() похожи тем, что они оба используются для выполнения нескольких асинхронных операций параллельно и ожидания первого завершения. Однако метод Promise.any() вернет результат первой успешно выполненной операции, а метод Promise.race() вернет результат первой завершившейся операции независимо от того, успешно она завершилась или нет.

Async/await - это синтаксический сахар для работы с промисами. Он позволяет писать асинхронный код так же, как синхронный код.