Лабораторная работа 03б

ПвИ

**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**

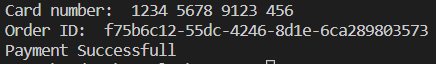
1. Разработайте функцию ***createOrder***, в которой будет создаваться Promise. Эта функция должна принимать в качестве параметра номер карты покупателя, проверять ее, а также назначать идентификатор заказу.

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**

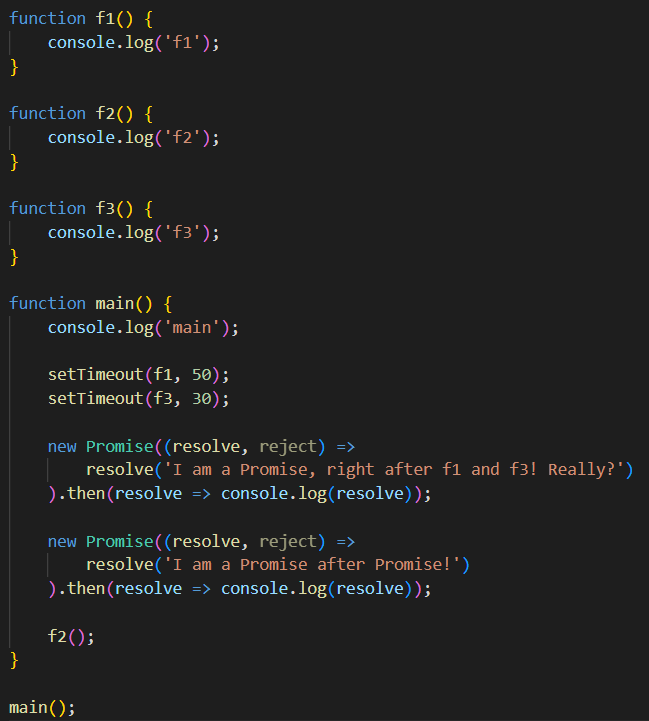
1. Разработайте три отдельные функции для вычисления квадрата, куба и четвертой степени заданного числа. Каждая из функций должна возвращать Promise, который либо разрешается с вычисленным значением, либо отклоняется с сообщением об ошибке, если ввод недействителен.
2. Далее используйте Promise.all() 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.

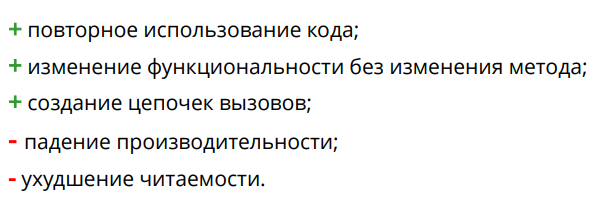


**Задание 08.** Ответьте следующие на вопросы.

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?
12. **Что такое callback?**

Callback - это функция, которая передается в другую функцию в качестве аргумента и выполняется после завершения выполнения этой функции. Callback-функции используются для обработки результатов асинхронных операций, таких как чтение файла или выполнение запроса к базе данных. Когда операция завершается, вызывается callback-функция, которая обрабатывает результаты операции.

1. **В чем минусы использования коллбэков? Какие есть способы их решения?**



Разбиение коллбека на функции??????

Использование колбэков в Node.js может привести к так называемому "callback hell" - глубокой вложенности колбэков друг в друга, что делает код менее читабельным и управляемым. Кроме того, при использовании колбэков может быть сложно обрабатывать ошибки.

Два распространенных способа решения этой проблемы в Node.js - использование Promise и async/await. Promise позволяют лучше структурировать код, обрабатывать ошибки и делать его более читабельным. Async/await, в свою очередь, предоставляют синтаксический сахар над Promise, делая код более простым и понятным.

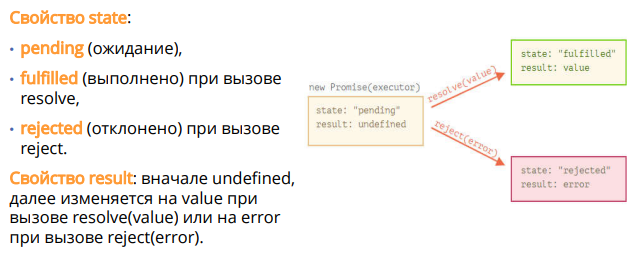
Например, вместо множества вложенных колбэков, можно использовать Promise.all(), который позволяет выполнить несколько асинхронных операций параллельно и дождаться завершения всех. Promise.race() позволяет выполнить несколько асинхронных операций параллельно и вернуть результат первой завершившейся операции. Promise.any() позволяет выполнить несколько асинхронных операций параллельно и вернуть результат первой операции, которая завершилась успешно.

1. **Что такое Promise и как он работает?**

**Promise (Обещание)** - объект, используемый для выполнения отложенных и асинхронных вычислений. Представляет собой операцию, которая еще не завершена, но ожидается в будущем.

Обработчик промиса возвращает новый промис с неопределенным result (можно его задать с помощью оператора return), поэтому их можно объединять в цепочки.

1. **В каких состояниях может находиться Promise?**



1. **Как изменить состояние Promise?**

Чтобы изменить состояние Promise в Node.js, вы можете использовать методы resolve() и reject(). Метод resolve() вызывается, когда операция, обернутая в Promise, завершилась успешно, а метод reject() вызывается, когда операция завершилась с ошибкой.

После вызова методов resolve() или reject() Promise переходит в соответствующее состояние: fulfilled (успешно выполнен) или rejected (выполнен с ошибкой). Вы можете обрабатывать эти состояния, используя методы then() и catch():

1. **Как изменить значение Promise?**

Вы можете изменить состояние Promise, вызвав один из методов then() или catch()

1. **Что такое цепочки промисов и как они работает?**

Цепочки промисов - это метод организации асинхронного кода в Node.js, который позволяет последовательно выполнять асинхронные операции. Каждый промис в цепочке представляет собой асинхронную операцию, которая должна быть выполнена после завершения предыдущей операции.

При использовании цепочек промисов, результат предыдущего промиса передается в следующий промис в цепочке с помощью метода then(). Если в каком-либо промисе происходит ошибка, то она передается в следующий промис с помощью метода catch().

В результате, цепочка промисов позволяет писать более чистый и понятный код, который легче отлаживать и поддерживать.

1. **Назовите два способа обработки ошибок в Promise.**

Использование метода catch() в конце цепочки методов then(). Этот метод позволяет обработать ошибки, которые могут возникнуть в любом из предыдущих методов then().

Использование метода reject() внутри Promise. Этот метод вызывается, если произошла ошибка, и позволяет передать ошибку в метод catch().

1. **Для чего нужен метод Promise.all()?**

Метод Promise.all() используется для выполнения нескольких промисов параллельно и ожидания результата выполнения всех промисов. Он принимает массив промисов в качестве аргумента и возвращает новый промис, который разрешается, когда все промисы в массиве разрешаются. Если хотя бы один промис из массива отклоняется, то возвращаемый промис тоже будет отклонен. Это очень полезно, когда необходимо выполнить несколько асинхронных операций одновременно и дождаться их завершения, чтобы продолжить выполнение кода.

1. **В чем отличия методов Promise.race() и Promise.any()?**

Race() вернет результат первого промиса в списке

Any() вернет результат первого ПРАВИЛЬНОГО в списке

Метод Promise.race() возвращает промис, который завершится с результатом первого промиса из списка переданных ему промисов, который завершится, будь то успешно или с ошибкой. Тогда как метод Promise.any() возвращает промис, который завершится с результатом первого успешно завершенного промиса из списка переданных ему промисов.

Если все промисы завершатся с ошибкой, Promise.any() вернет ошибку типа AggregateError, содержащую массив ошибок, которые произошли при выполнении всех переданных промисов.

1. **Что такое async/await?**

Синтаксис для обработки нескольких промисов в режиме синхронного кода.

**async** – перед объявлением функции, возвращает промис.

**await** – блокирует код до тех пор, пока промис не будет разрешен или отклонен.

