**Вопросы к лабораторной работе №3**

**Часть Б**

1. Что такое **callback**?

Callback - это функция, которая передается как аргумент в другую функцию и выполняется после завершения выполнения этой функции. Callbacks являются одним из способов реализации асинхронности в JavaScript, позволяя выполнять код после завершения асинхронных операций, таких как чтение файла, отправка запроса на сервер или обработка событий.

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

*Минусы использования коллбэков:*

**Callback Hell**: Когда слишком много асинхронных операций выполняется последовательно или вложенно, может возникнуть callback hell, что делает код менее читаемым и поддерживаемым.

**Сложность управления потоком**: Обработка ошибок и управление последовательностью выполнения может быть сложной задачей при использовании коллбэков.

**Отсутствие структуры**: Код с использованием коллбэков может стать плохо структурированным и трудночитаемым.

*Способы решения:*

Promises: Использование промисов упрощает управление асинхронными операциями и устраняет callback hell.

Async/await: С использованием async/await код становится более линейным и понятным, а обработка ошибок более удобной.

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

Promise – это объект, представляющий асинхронную операцию и ее результат. Promise позволяет управлять асинхронными операциями и облегчает их обработку.

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

Состояния Promise:

Ожидание (pending): Исходное состояние, когда Promise создан, но еще не разрешен и не отклонен.

Разрешено (resolved): Когда асинхронная операция завершается успешно, Promise переходит в это состояние, и результат операции доступен через then() обработчики.

Отклонено (rejected): Когда асинхронная операция завершается с ошибкой, Promise переходит в это состояние, и ошибка доступна через catch() обработчики.

1. Как изменить состояние Promise?

Для изменения состояния Promise на ‘разрешено’, вызывается функция resolve() с результатом операции.

Для изменения состояния Promise на ‘отклонено’, вызывается функция reject() с ошибкой.

1. Как изменить значение Promise?

Значение Promise устанавливается в момент разрешения или отклонения, и его нельзя изменить после этого. Promise всегда будет содержать фиксированное значение, которое передается в resolve() или reject().

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

Цепочки промисов (Promise chaining) – это метод использования промисов для выполнения последовательности асинхронных операций. Одна операция начинается сразу после завершения предыдущей, обеспечивая линейный и читаемый код.

1. Назовите два способа обработки ошибок в Promise.

Два способа обработки ошибок в Promise:

Использование метода .catch() после цепочки .then() для обработки ошибок, возникших в предыдущих операциях.

Использование блока try/catch внутри функции, использующей async/await, для обработки ошибок в асинхронном коде.

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

Promise.all() - это метод, который позволяет выполнить массив промисов параллельно и дождаться их завершения. Он возвращает новый промис, который разрешается, когда все переданные промисы разрешаются успешно, или отклоняется, если хотя бы один из промисов отклоняется.

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

**Promise.race():** Если первый разрешенный промис был отклонен, то Promise.race() вернет эту ошибку, игнорируя все остальные успешные промисы или их значения.

**Promise.any():** Если все переданные промисы отклонены, то Promise.any() вернет ошибку, содержащую массив ошибок отклоненных промисов. Если хотя бы один промис разрешится успешно, то Promise.any() вернет результат этого промиса, игнорируя остальные ошибки.

1. Что такое **async/await**?

**async/await** – специальный синтаксис для работы с промисами.

Основные компоненты async/await:

**async функции:** Ключевое слово async перед объявлением функции делает ее асинхронной. Это означает, что эта функция будет возвращать промис и может содержать операторы await.

**await оператор:** Оператор await используется внутри async функции для приостановки выполнения функции до тех пор, пока промис, переданный в await, не разрешится. Затем значение разрешенного промиса будет присвоено переменной.