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. **Callback** - это функция, которая передается как аргумент в другую функцию и выполняется после завершения выполнения этой функции. Основное назначение коллбэков - асинхронное выполнение кода, например, при завершении асинхронных операций, таких как чтение файла или запрос к серверу.
13. **Минусы использования коллбэков**:

* **Callback Hell**: Вложенные коллбэки могут привести к созданию громоздкого и трудноподдерживаемого кода.
* **Сложность отладки**: Отслеживание и отладка ошибок может быть сложной из-за асинхронной природы коллбэков.
* **Проблемы с обработкой ошибок**: Обработка ошибок в коллбэках может быть неудобной и часто приводит к неотлавливаемым исключениям.
* **Отсутствие контроля над потоком выполнения**: Коллбэки могут выполняться в непредсказуемом порядке.

Для решения этих проблем можно использовать **Promise** и **async/await**.

1. **Promise** - это объект, представляющий асинхронную операцию, которая может завершиться успешно (resolved) или с ошибкой (rejected). Он предоставляет более удобный способ обработки асинхронных операций и управления потоком выполнения кода.
2. **Promise** может находиться в одном из трех состояний:

* **Pending**: Начальное состояние, обещание не выполнено и не отклонено.
* **Resolved**: Обещание выполнено успешно.
* **Rejected**: Обещание отклонено с ошибкой.

1. Состояние **Promise** изменяется автоматически при завершении асинхронной операции. Для изменения состояния **Promise** используются функции **resolve** и **reject**.
2. **Promise** не может изменить свое значение после разрешения или отклонения. Он либо хранит успешный результат, либо ошибку.
3. **Цепочка промисов** - это последовательное выполнение асинхронных операций с использованием промисов. Результат одной операции передается в следующую. Это позволяет управлять потоком выполнения кода и делает его более читаемым.
4. Два способа обработки ошибок в **Promise**:

* Использование метода **.catch()** для перехвата ошибок в любой части цепочки промисов.
* Использование блока **try/catch** внутри асинхронной функции с **await** для обработки ошибок.

1. Метод **Promise.all()** используется для ожидания выполнения нескольких промисов и возврата массива результатов, когда все промисы разрешены. Если хотя бы один из промисов отклонен, он возвращает ошибку первого отклоненного промиса.
2. **Promise.race()** и **Promise.any()** работают по-разному:

* **Promise.race()** возвращает результат первого разрешенного или отклоненного промиса.
* **Promise.any()** возвращает результат первого разрешенного промиса, игнорируя отклоненные, если все промисы отклонены, то возвращает ошибку.

1. **async/await** - это синтаксический сахар над **Promise**, который делает асинхронный код более читаемым и понятным. **async** используется для объявления функции, как асинхронной, а **await** используется для ожидания выполнения промиса и получения его результата.