1. global - хранит var-данные на уровне модуля

process - информация о среде выполнения, о текущем процессе

buffer - класс Buffer – предназначен для работы с двоичными данными.

console - используется для печати в stdout и stderr

2. Асинхронная функция – это функция, после вызова которой JavaScript приложение продолжает работать, потому что функция сразу выполняет возврат.

3. process.stdin - поток на чтение содержит стандартный системный поток ввода.

process.stdout - поток на запись, содержащий стандартный системный вывод.

process.stderr - поток на запись, содержащий стандартный системный вывод ошибок.

4. setImmediate() - переданынй callback будет выполнен на следующей итерации цикла событий.

callback`и помещаются в в цикл обработки событий

nextTick() - переданный callback будет выполнен после пердыдущей итерации цикла событий и до следующей

колбэки не помещаются в цикл обрабоки событий

1. Callback (коллбэк) - это функция, которая передается в другую функцию в качестве аргумента и вызывается внутри этой функции в определенный момент времени или по выполнении определенного условия. Коллбэки используются для обработки асинхронных операций, обратного вызова событий или передачи данных между компонентами системы.
2. Минусы использования коллбэков:

* Коллбэк хранит состояние и контекст выполнения, что может привести к сложному пониманию и отладке кода.
* Если коллбэки используются множественно или вложенно (так называемый "callback hell"), код может стать трудным для чтения и поддержки.
* При использовании коллбэков может возникнуть проблема с обработкой ошибок и управлением потоком выполнения.

Некоторые способы решения этих проблем включают использование промисов, асинхронных функций (async/await) или библиотек, таких как async.js или RxJS, которые предлагают более удобные средства для управления асинхронным кодом.

1. Promise (обещание) - это объект, представляющий результат (успех или отказ) асинхронной операции. Он позволяет выполнять асинхронные операции последовательно, комбинировать их и обрабатывать ошибки. Promise имеет три возможных состояния: ожидание (pending), выполнено (fulfilled) и отклонено (rejected).
2. Состояния Promise:

* Ожидание (pending): Исходное состояние, когда промис не разрешен и не отклонен. Промис находится в ожидании выполнения асинхронной операции.
* Выполнено (fulfilled): Состояние, когда промис успешно разрешен и имеет конечное значение.
* Отклонено (rejected): Состояние, когда промис отклонен и имеет ошибку или отказ.

1. Состояние Promise меняется автоматически. Оно может измениться с ожидания на выполнено или отклонено, но после этого оно остается неизменным. Разработчик не может изменить состояние промиса вручную.
2. Значение Promise может быть изменено только один раз. При выполнении асинхронной операции, когда промис разрешается, ему можно передать значение, которое станет конечным результатом промиса. Однако после разрешения или отклонения промиса его значение не может быть изменено.
3. Цепочка промисов (promise chaining) - это способ последовательного выполнения асинхронных операций с использованием промисов. Результат одной операции может быть передан в следующую операцию в цепочке, что позволяет выполнять операции последовательно и обрабатывать результаты.

При использовании цепочек промисов можно применять методы .then() или .catch() для обработки разрешенных или отклоненных промисов соответственно. Это позволяет последовательно применять функции к результатам промисов и передавать значения по цепочке.

1. Два способа обработки ошибок в Promise:

* Использование метода .catch(): Метод .catch() применяется к промису и используется для обработки отклоненных промисов в цепочке. Если в цепочке промисов происходит отклонение, выполнение переходит к ближайшему блоку .catch(), где можно обработать ошибку.
* Использование метода .then() вторым аргументом: Метод .then() может принимать два аргумента: первый для обработки разрешенного промиса, второй для обработки отклоненного промиса. Второй аргумент .then() можно использовать для обработки ошибок, аналогично использованию метода .catch().

1. Метод Promise.all() используется для выполнения нескольких промисов параллельно и ожидания их разрешения. Он принимает массив промисов в качестве аргумента и возвращает новый промис, который разрешается, когда все промисы в массиве разрешены, или отклоняется, если хотя бы один промис отклоняется. Результатом разрешенного промиса является массив значений, соответствующих разрешенным промисам в том же порядке.
2. Методы Promise.race() и Promise.any() различаются в своем поведении:

* Promise.race() принимает массив промисов в качестве аргумента и возвращает новый промис, который разрешается или отклоняется в зависимости от того, какой промис разрешится или отклонится первым. Результатом разрешенного промиса является значение первого разрешенного или отклоненного промиса.
* Promise.any() также принимает массив промисов в качестве аргумента и возвращает новый промис, который разрешается, если хотя бы один промис в массиве разрешается. Он отклоняется только в том случае, если все промисы отклоняются. Результатом разрешенного промиса является значение первого разрешенного промиса.

1. async/await - это синтаксический сахар в JavaScript, который облегчает написание асинхронного кода и управление промисами. async используется для объявления асинхронной функции, которая всегда возвращает промис. await используется внутри асинхронной функции для ожидания разрешения промиса и получения его результата.

Когда await используется перед промисом, выполнение асинхронной функции приостанавливается до тех пор, пока промис не разрешится, и затем возвращается его результат. Это позволяет писать асинхронный код, который выглядит более синхронно и линейно.