1. **Перечислите основные свойства глобальные объекты Node.js и поясните их предназначение.**

Node.js предоставляет специальный объект global, который предоставляет доступ к глобальным, то есть доступным из каждого модуля приложения, переменным и функциям. Примерным аналогом данного объекта в javascript для браузера является объект window.

Эти объекты доступны во всех модулях.

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

Существуют только в составе модулей:

exports - ссылка на module.exports

module - ссылка на текущий модуль

require - используется для импорта модуле

1. **Поясните понятие «асинхронная функция»**

**Promise – это специальный объект, который содержит своё состояние**

Асинхронные функции не являются заменой промисов(. Они работают сообща. Асинхронная функция ожидает (await) исполнения промиса и всегда возвращает промис.Промис, возвращаемый асинхронной функцией, будет разрешен (resolve) с тем значением, которое вернет функция.

Асинхронная функция – это функция, после вызова которой JavaScript приложение продолжает работать, потому что функция сразу выполняет возврат. Результат работы асинхронной функции становится известным позже, и для того, чтобы оповестить наше приложении о полученных значениях, асинхронная функция вызывает другую функцию (callback), которую мы передаем в аргументах при запуске.

1. **Поясните понятие стандартные «системные потоки».**

Поток – байтовая последовательность, передаваемая в процессе ввода-вывода.

Стандартные потоки ввода-вывода в [системах](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) типа [UNIX](https://ru.wikipedia.org/wiki/UNIX) (и некоторых других) — [потоки](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [процесса](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), имеющие номер (дескриптор), зарезервированный для выполнения некоторых «стандартных» функций

Эти потоки поставляются с node.js и могут быть использованы без подключения

дополнительных библиотек.

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.

Stdin, stdout, stderr, stdprn, stdaux.

1. **Поясните назначение функций process.nextTick, setImmediate, поясните в чем разница.**

setImmediate() и setTimeout() похожи, но ведут себя по-разному в том, когда они вызываются.

* setImmediate() предназначен для выполнения сценария после завершения текущей фазы опроса.
* setTimeout() планирует запуск сценария после истечения минимального порога в миллисекундах.

Оглядываясь назад на нашу диаграмму, каждый раз, когда вы вызываете process.nextTick() на данной фазе, все коллбэки, переданные процессу process.nextTick(), будут разрешаться до того, как цикл событий продолжится. Это может создать некоторые плохие ситуации, потому что это позволяет «замораживать» ваш ввод/вывод, делая рекурсивные вызовы process.nextTick(), что не даёт циклу событий достичь фазы опроса.

У нас есть два вызова, которые похожи для пользователей, но их имена сбивают с толку.

* process.nextTick() срабатывает сразу на той же фазе
* setImmediate() срабатывает на следующей итерации или «тике» цикла событий

Node.js является событийно-ориентированной системой. Все, что происходит в ноде, является реакцией на события и события вызывают каскад колбеков. Этот механизм работает на основе библиотеки libuv и называется циклом событий (event loop).

Цикл событий представляет собой процесс, состоящий из этапов (групп задач), которые выполняются по очереди.

setImmediate() предназначен для выполнения сценария после завершения текущей фазы опроса, отправляется к конец очереди.

process.nextTick отправляется в начало очереди событий, чтобы она выполнялась сразу после завершения текущей функции.

Основное различие между setImmediate и process.nextTick заключается в том, что обратные вызовы setImmediate помещаются в цикл обработки событий, а process.nextTick нет.

process.nextTick() технически не является частью цикла событий. Вместо этого nextTickQueue будет обрабатываться после завершения текущей операции, независимо от текущей фазы цикла событий.

process.nextTick() срабатывает сразу на той же фазе

setImmediate() срабатывает на следующей итерации или «тике» цикла событий