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.

1. **process:** информация о среде выполнения
2. **global: *хранит var-данные на уровне модуля***.
3. **buffer:** класс Buffer – предназначен для работы с двоичными данными: набором октетов.
4. **Поясните понятие «асинхронная функция»**

**Асинхронное программирование:** выполнение процесса в неблокирующем режиме системного вызова, что ***позволяет потоку программы продолжить обработку***.

операция называется асинхронной, если ее выполнение осуществляется в 2 фазы: 1) заявка на исполнение; 2) получение результата; при этом участвуют два механизма: A-механизм, формирующий заявку и потом получающий результат; B-механизм, получающий заявку от A, исполняющий операцию и отправляющий результат A; продолжительность исполнения операции B-механизмом, как правило, непредсказуемо; в то время пока B-механизм исполняет операцию, А-**механизм выполняет** собственную работу.

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

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.

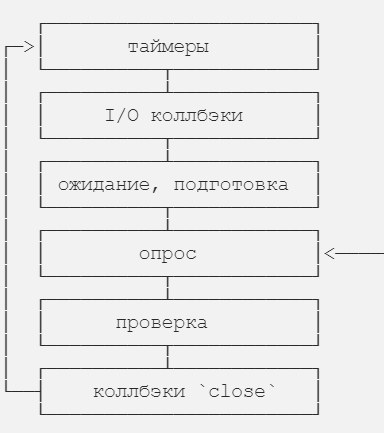
* setImmediate() предназначен для выполнения сценария после завершения текущей фазы опроса.
* setTimeout() планирует запуск сценария после истечения минимального порога в миллисекундах.

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

функция **setTimeout(callback, delay) –** глобальная функция, котораяпринимает функцию обратного вызова и временно сохраняет ее. По истечению времени, заданного в миллисекундах, функция обратного вызова помещается в **очередь коллбэков (callback queue)**. Затем event loop перемещает эту функцию в **стек вызовов (call stack)**, когда стек пуст. После этого осуществляется выполение функция обратного вызова. В основном так работают все веб-API.



setImmediate() — это специальный таймер, который выполняется в отдельной фазе цикла событий. Он использует API libuv, чтобы запланировать коллбэки для выполнения после завершения фазы опроса.

Вообще, когда код исполняется, цикл событий, в конечном счете, попадает на фазу опроса, где он будет ждать входящего соединения, запроса и т.д. Однако, если коллбэк был запланирован с помощью setImmediate(), и фаза опроса переходит в нерабочее состояние, она закончится и начнётся фаза проверки, вместо того, чтобы ждать событий опроса.

После помещения обратного вызова в process.nextTick(), сценарий всё еще имеет возможность выполниться до конца, позволяя инициализировать все переменные, функции и т.д. до вызова коллбэка. Это также имеет то преимущество, что не позволяет циклу событий перейти на следующую фазу. Может быть полезно, чтобы пользователь был предупрежден об ошибке до того, как цикл событий продолжится

В сущности, имена следует поменять местами. process.nextTick() срабатывает быстрее, чем setImmediate(), но это артефакт прошлого, который вряд ли изменится. Такое изменение приведет к поломке большого количества пакетов в npm. Каждый день добавляются новые модули, а это значит, что каждый день пока мы ждем, возникают всё более серьезные потенциальные проблемы от такого изменения. Хотя это сбивают с толку, сами имена не изменятся.

Мы рекомендуем разработчикам использовать setImmediate() во всех случаях, потому что оно легче к пониманию (и также позволяет писать код, который совместим с более широким набором окружений, таких как браузерный JS.)