Вопросы к лабораторной работе №3

Часть А

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

Основные глобальные объекты Node.js и их назначение:

global: Глобальный объект, предоставляющий доступ к глобальным переменным и функциям в приложении Node.js. Все переменные и функции, объявленные без ключевого слова var, let, или const, автоматически становятся членами объекта global.

process: Этот объект предоставляет информацию и управление текущим процессом Node.js. Он содержит множество полезных свойств и методов, таких как process.env (для доступа к переменным окружения), process.argv (для аргументов командной строки), и process.exit() (для завершения процесса).

console: Глобальный объект, предоставляющий функции для вывода сообщений в консоль. Это основной способ отладки и записи информации в Node.js приложениях.

Buffer: Глобальный объект для работы с бинарными данными. Он используется для создания, чтения и записи бинарных данных, таких как байтовые массивы.

require(): Эта функция используется для подключения модулей в Node.js. Она позволяет импортировать функциональность других файлов и библиотек в ваше приложение.

2. Поясните понятие «асинхронная функция»

Асинхронная функция – это функция, которая выполняется асинхронно, то есть не блокирует выполнение остального кода и может завершиться в будущем. Она инициирует выполнение некоторой задачи и возвращает управление сразу, вместо ожидания завершения этой задачи. Вместо блокировки исполнения, асинхронные функции используют колбэки, промисы или асинхронные/ожидаемые ключевые слова (async/await) для управления последовательностью выполнения.

3. Поясните понятие стандартные «системные потоки».

Стандартные системные потоки - это основные потока ввода-вывода, предоставляемые операционной системой для взаимодействия с программами:

stdin (стандартный поток ввода): Этот поток предназначен для приема ввода от пользователя или других программ. В Node.js, он доступен через объект process.stdin.

stdout (стандартный поток вывода): Этот поток используется для вывода данных и сообщений программы. В Node.js, он доступен через объект process.stdout.

stderr (стандартный поток ошибок): Этот поток используется для вывода ошибок и диагностических сообщений. В Node.js, он доступен через объект process.stderr.

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

Функции process.nextTick и setImmediate используются в Node.js для выполнения асинхронного кода. Они имеют следующие различия:

process.nextTick: Эта функция позволяет выполнить функцию или колбэк как можно скорее после завершения текущей операции Event Loop, но до выполнения других асинхронных операций, таких как таймеры и промисы. Функции, переданные process.nextTick, имеют приоритет над setImmediate. Это может быть полезно, например, для выполнения некоторых операций перед обновлением пользовательского интерфейса в браузере или передачи управления другим компонентам.

setImmediate: Эта функция планирует выполнение функции или колбэка после завершения текущей операции Event Loop, но перед началом следующей итерации Event Loop. Она предпочтительна, когда вам нужно выполнить асинхронный код, который не должен блокировать Event Loop, и который не требует немедленного выполнения.

Обе функции позволяют создавать асинхронные задачи, но разница в их приоритете выполнения и моменте планирования может повлиять на последовательность выполнения кода.

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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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().

7. 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.

8. 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.

9. 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.

10.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.

11.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.