NODEJS INTERVIEW QUESTIONS:

**1)      What is node.js?**

Node.js is a Server side scripting which is used to build scalable programs. Its multiple advantages over other server side languages, the prominent being non-blocking I/O.

**2)      How node.js works?**

Node.js works on a v8 environment, it is a virtual machine that utilizes JavaScript as its scripting language and achieves high output via non-blocking I/O and single threaded event loop.

**3)      What do you mean by the term I/O ?**

I/O is the shorthand for input and output, and it will access anything outside of your application. It will be loaded into the machine memory to run the program, once the application is started.

**4)      What does event-driven programming mean?**

In computer programming, event driven programming is a programming paradigm in which the flow of the program is determined by events like messages from other programs or threads. It is an application architecture technique divided into two sections 1) Event Selection 2) Event Handling

**5)      Where can we use node.js?**

Node.js can be used for the following purposes

a)      Web applications ( especially real-time web apps )

b)      Network applications

c)       Distributed systems

d)      General purpose applications

**6)      What is the advantage of using node.js?**

a)      It provides an easy way to build scalable network programs

b)      Generally fast

c)       Great concurrency

d)      Asynchronous everything

e)      Almost never blocks

**7)      What are the two types of API functions in Node.js ?**

The two types of API functions in Node.js are

a)      Asynchronous, non-blocking functions

b)      Synchronous, blocking functions

**8)      What is control flow function?**

A generic piece of code which runs in between several asynchronous function calls is known as control flow function.

**9)      Explain the steps how “Control Flow” controls the functions calls?**

a)      Control the order of execution

b)      Collect data

c)       Limit concurrency

d)      Call the next step in program

**10)   Why Node.js is single threaded**?

For async processing, Node.js was created explicitly as an experiment. It is believed that more performance and scalability can be achieved by doing async processing on a single thread under typical web loads than the typical thread based implementation.

**11)   Does node run on windows?**

Yes – it does. Download the MSI installer from <http://nodejs.org/download/>

**12)   Can you access DOM in node?**

No, you cannot access DOM in node.

**13)   Using the event loop what are the tasks that should be done asynchronously?**

a)      I/O operations

b)      Heavy computation

c)       Anything requiring blocking

**14)   Why node.js is quickly gaining attention from JAVA programmers?**

Node.js is quickly gaining attention as it is a loop basedomei server for JavaScript. Node.js gives user the ability to write the JavaScript on the server, which has access to things like HTTP stack, file I/O, TCP and databases.

**15)   What are the two arguments that async.queue takes?**

The two arguments that async.queue takes

a)      Task function

b)      Concurrency value

**16)   What is an event loop in Node.js ?**

To process and handle external events and to convert them into callback invocations an event loop is used. So, at I/O calls, node.js can switch from one request to another .

**17)   Mention the steps by which you can async in Node.js?**

By following steps you can async Node.js

a)      First class functions

b)      Function composition

c)       Callback Counters

d)      Event loops

**18)    What are the pros and cons of Node.js?**

**Pros:**

a)      If your application does not have any CPU intensive computation, you can build it in Javascript top to bottom, even down to the database level if you use JSON storage object DB like MongoDB.

b)      Crawlers receive a full-rendered HTML response, which is far more SEO friendly rather than a single page application or a websockets app run on top of Node.js.

**Cons:**

a)       Any intensive CPU computation will block node.js responsiveness, so a threaded platform is a better approach.  
b)      Using relational database with Node.js is considered less favourable

**19)   How Node.js overcomes the problem of blocking of I/O operations?**

Node.js solves this problem by putting the event based model at its core, using an event loop instead of threads.

**20)   What is the difference between Node.js vs Ajax?**

The difference between Node.js and Ajax is that, Ajax (short for Asynchronous Javascript and XML) is a client side technology, often used for updating the contents of the page without refreshing it. While,Node.js is Server Side Javascript, used for developing server software. Node.js does not execute in the browser but by the server.

**21)   What are the Challenges with Node.js ?**

Emphasizing on the technical side, it’s a bit of challenge in Node.js to have one process with one thread to scale up on multi core server.

**22)** **What does it mean “non-blocking” in node.js?**

In node.js “non-blocking” means that its IO is non-blocking.  Node uses “libuv” to handle its IO in a platform-agnostic way. On windows, it uses completion ports for unix it uses epoll or kqueue etc. So, it makes a non-blocking request and upon a request, it queues it within the event loop which call the JavaScript ‘callback’ on the main JavaScript thread.

**23)   What is the command that is used in node.js to import external libraries?**

Command “require” is used for importing external libraries, for example, “var http=require (“http”)”.  This will load the http library and the single exported object through the http variable.

**24)   Mention the framework most commonly used in node.js?**

“Express” is the most common framework used in node.js

**25)   What is ‘Callback’ in node.js?**

Callback function is used in node.js to deal with multiple requests made to the server. Like if you have a large file which is going to take a long time for a server to read and if you don’t want a server to get engage in reading that large file while dealing with other requests, call back function is used. Call back function allows the server to deal with pending request first and call a function when it is finished.

What is Node.js?

Node.js is a web application framework built on Google Chrome's JavaScript Engine(V8 Engine).

Node.js comes with runtime environment on which a Javascript based script can be interpreted and executed (It is analogus to JVM to JAVA byte code). This runtime allows to execute a JavaScript code on any machine outside a browser. Because of this runtime of Node.js, JavaScript is now can be executed on server as well.

Node.js also provides a rich library of various javascript modules which eases the developement of web application using Node.js to great extents.

Node.js = Runtime Environment + JavaScript Library

What do you mean by Asynchronous API?

All APIs of Node.js library are aynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get response from the previous API call.

What are the benefits of using Node.js?

Following are main benefits of using Node.js

* **Aynchronous and Event Driven**All APIs of Node.js library are aynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get response from the previous API call.
* **Very Fast** Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* **Single Threaded but highly Scalable** - Node.js uses a single threaded model with event looping. Event mechanism helps server to respond in a non-bloking ways and makes server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and same program can services much larger number of requests than traditional server like Apache HTTP Server.
* **No Buffering** - Node.js applications never buffer any data. These applications simply output the data in chunks.

Is Node a single threaded application?

* Yes! Node uses a single threaded model with event looping.

What is REPL in context of Node?

REPL stands for Read Eval Print Loop and it represents a computer environment like a window console or unix/linux shell where a command is entered and system responds with an output. Node.js or Node comes bundled with a REPL environment. It performs the following desired tasks.

* **Read** - Reads user's input, parse the input into JavaScript data-structure and stores in memory.
* **Eval** - Takes and evaluates the data structure
* **Print** - Prints the result
* **Loop** - Loops the above command until user press ctrl-c twice.

What is npm?

npm stands for Node Package Manager. npm provides following two main functionalities:

* Online repositories for node.js packages/modules which are searchable on [search.nodejs.org](http://search.nodejs.org/)
* Command line utility to install packages, do version management and dependency management of Node.js packages.

What is global installation of dependencies?

Globally installed packages/dependencies are stored in **<user-directory>**/npm directory. Such dependencies can be used in CLI (Command Line Interface) function of any node.js but can not be imported using require() in Node application directly. To install a Node project globally use -g flag.

C:\Nodejs\_WorkSpace>npm install express -g

What is local installation of dependencies?

By default, npm installs any dependency in the local mode. Here local mode refers to the package installation in node\_modules directory lying in the folder where Node application is present. Locally deployed packages are accessible via require(). To install a Node project locally following is the syntax.

C:\Nodejs\_WorkSpace>npm install express

How to check the already installed dependencies which are globally installed using npm?

Use the following command:

C:\Nodejs\_WorkSpace>npm ls -g

What is Package.json?

package.json is present in the root directory of any Node application/module and is used to define the properties of a package.

Name some of the attributes of package.json?

Following are the attributes of Package.json

* **name** - name of the package
* **version** - version of the package
* **description** - description of the package
* **homepage** - homepage of the package
* **author** - author of the package
* **contributors** - name of the contributors to the package
* **dependencies** - list of dependencies. npm automatically installs all the dependencies mentioned here in the node\_module folder of the package.
* **repository** - repository type and url of the package
* **main** - entry point of the package
* **keywords** - keywords

How to uninstall a dependency using npm?

Use following command to uninstall a module.

C:\Nodejs\_WorkSpace>npm uninstall dependency-name

How to update a dependency using npm?

Update package.json and change the version of the dependency which to be updated and run the following command.

C:\Nodejs\_WorkSpace>npm update

What is Callback?

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All APIs of Node are written is such a way that they supports callbacks. For example, a function to read a file may start reading file and return the control to execution environment immidiately so that next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process high number of request without waiting for any function to return result.

What is a blocking code?

If application has to wait for some I/O operation in order to complete its execution any further then the code responsible for waiting is known as blocking code.

How Node prevents blocking code?

By providing callback function. Callback function gets called whenever corresponding event triggered.

What is Event Loop?

Node js is a single threaded application but it support concurrency via concept of event and callbacks. As every API of Node js are asynchronous and being a single thread, it uses async function calls to maintain the concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever any task get completed, it fires the corresponding event which signals the event listener function to get executed.

What is Event Emmitter?

EventEmitter class lies in **events** module. It is accessibly via following syntax:

//import events module

var events = require('events');

//create an eventEmitter object

var eventEmitter = new events.EventEmitter();

When an EventEmitter instance faces any error, it emits an 'error' event. When new listener is added, 'newListener' event is fired and when a listener is removed, 'removeListener' event is fired.

EventEmitter provides multiple properties like **on** and **emit**. **on** property is used to bind a function with the event and **emit** is used to fire an event.

What is purpose of Buffer class in Node?

Buffer class is a global class and can be accessed in application without importing buffer module. A Buffer is a kind of an array of integers and corresponds to a raw memory allocation outside the V8 heap. A Buffer cannot be resized.

What is Piping in Node?

Piping is a mechanism to connect output of one stream to another stream. It is normally used to get data from one stream and to pass output of that stream to another stream. There is no limit on piping operations. Consider the above example, where we've read test.txt using readerStream and write test1.txt using writerStream. Now we'll use the piping to simplify our operation or reading from one file and writing to another file.

Which module is used for file based operations?

fs module is used for file based operations.

var fs = require("fs")

Which module is used for buffer based operations?

buffer module is used for buffer based operations.

var buffer = require("buffer")

Which module is used for web based operations?

http module is used for web based operations.

var http = require("http")

What are streams?

Streams are objects that let you read data from a source or write data to a destination in continous fashion.

How many types of streams are present in Node.

In Node.js, there are four types of streams.

* **Readable** - Stream which is used for read operation.
* **Writable** - Stream which is used for write operation.
* **Duplex** - Stream which can be used for both read and write operation.
* **Transform** - A type of duplex stream where the output is computed based on input.

Name some of the events fired by streams.

Each type of Stream is an **EventEmitter** instance and throws several events at different instance of times. For example, some of the commonly used events are:

* **data** - This event is fired when there is data is available to read.
* **end** - This event is fired when there is no more data to read.
* **error** - This event is fired when there is any error receiving or writing data.
* **finish** - This event is fired when all data has been flushed to underlying system

What is Chaining in Node?

Chanining is a mechanism to connect output of one stream to another stream and create a chain of multiple stream operations. It is normally used with piping operations.

Q-1. What Is Node.Js?

**Answer.**

Node.js is a JavaScript runtime or platform which is built on Google Chrome’s JavaScript v8 engine. This runtime allows executing the JavaScript code on any machine outside a browser (this means that it is the server that executes the Javascript and not the browser).

Node.js is single-threaded, that employs a concurrency model based on an event loop. It doesn’t block the execution instead registers a callback which allows the application to continue. It means Node.js can handle concurrent operations without creating multiple threads of execution so can scale pretty well.

It uses JavaScript along with C/C++ for things like interacting with the filesystem, starting up HTTP or TCP servers and so on. Due to it’s extensively fast growing community and NPM, Node.js has become a very popular, open source and cross-platform app. It allows developing very fast and scalable network app that can run on Microsoft Windows, Linux, or OS X.

Following are the areas where it’s perfect to use Node.js.

* I/O bound Applications
* Data Streaming Applications
* Data Intensive Real-time Applications (DIRT)
* JSON APIs based Applications
* Single Page Applications

At the same time, it’s not suitable for heavy applications involving more of CPU usage.

### Q-2. What Are The Key Features Of Node.Js?

* **Asynchronous event driven IO helps concurrent request handling –** All APIs of Node.js are asynchronous. This feature means that if a Node receives a request for some Input/Output operation, it will execute that operation in the background and continue with the processing of other requests. Thus it will not wait for the response from the previous requests.
* **Fast in Code execution –** Node.js uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node.js also become faster.
* **Single Threaded but Highly Scalable –** Node.js uses a single thread model for event looping. The response from these events may or may not reach the server immediately. However, this does not block other operations. Thus making Node.js highly scalable. Traditional servers create limited threads to handle requests while Node.js creates a single thread that provides service to much larger numbers of such requests.
* **Node.js library uses JavaScript –** This is another important aspect of Node.js from the developer’s point of view. The majority of developers are already well-versed in JavaScript. Hence, development in Node.js becomes easier for a developer who knows JavaScript.
* **There is an Active and vibrant community for the Node.js framework –** The active community always keeps the framework updated with the latest trends in the web development.
* **No Buffering –** Node.js applications never buffer any data. They simply output the data in chunks.

### Q-3. Explain How Do We Decide, When To Use Node.Js And When Not To Use It?

#### When Should We Use Node.Js?

It’s ideal to use Node.js for developing streaming or event-based real-time applications that require less CPU usage such as.

* Chat applications.
* Game servers.

Node.js is good for fast and high-performance servers, that face the need to handle thousands of user requests simultaneously.

#### Good For A Collaborative Environment.

It is suitable for environments where multiple people work together. For example, they post their documents, modify them by doing check-out and check-in of these documents.

Node.js supports such situations by creating an event loop for every change made to the document. The “Event loop” feature of Node.js enables it to handle multiple events simultaneously without getting blocked.

# Top 30 Node.Js Interview Questions With Answers

[Web Development](http://www.techbeamers.com/category/interview-questions/web-development/)  Published: February 15, 2017  [Meenakshi Agarwal](http://www.techbeamers.com/author/meenakshi/" \o "Posts by Meenakshi Agarwal)

Are you a MEAN stack developer and have an interview lined up? Don’t worry, just read the top 30 Node.js interview questions we’ve outlined in this post.

We’ve tried our best to keep the answers easy to understand and simple to remember. All of our questions are thoroughly researched and have a potential to surface in most Node.js interviews.

However, you would also prepare yourself for **[AngularJS interview questions](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/" \t "_blank)** as it is the next most important skill you’ll need to become an ideal MEAN stack developer. In out next posts, we’ll also cover other areas where you need to centralize your focus.

Let’s now begin to read to the most important Node.js interview questions and answers.

## Top 30 Node.Js Interview Questions & Answers.

**Prepare for Node.js Interview.**

### Q-1. What Is Node.Js?

**Answer.**

Node.js is a JavaScript runtime or platform which is built on Google Chrome’s JavaScript v8 engine. This runtime allows executing the JavaScript code on any machine outside a browser (this means that it is the server that executes the Javascript and not the browser).

Node.js is single-threaded, that employs a concurrency model based on an event loop. It doesn’t block the execution instead registers a callback which allows the application to continue. It means Node.js can handle concurrent operations without creating multiple threads of execution so can scale pretty well.

It uses JavaScript along with C/C++ for things like interacting with the filesystem, starting up HTTP or TCP servers and so on. Due to it’s extensively fast growing community and NPM, Node.js has become a very popular, open source and cross-platform app. It allows developing very fast and scalable network app that can run on Microsoft Windows, Linux, or OS X.

Following are the areas where it’s perfect to use Node.js.

* I/O bound Applications
* Data Streaming Applications
* Data Intensive Real-time Applications (DIRT)
* JSON APIs based Applications
* Single Page Applications

At the same time, it’s not suitable for heavy applications involving more of CPU usage.

### Q-2. What Are The Key Features Of Node.Js?

**Answer.**

Let’s look at some of the key features of Node.js.

* **Asynchronous event driven IO helps concurrent request handling –** All APIs of Node.js are asynchronous. This feature means that if a Node receives a request for some Input/Output operation, it will execute that operation in the background and continue with the processing of other requests. Thus it will not wait for the response from the previous requests.
* **Fast in Code execution –** Node.js uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node.js also become faster.
* **Single Threaded but Highly Scalable –** Node.js uses a single thread model for event looping. The response from these events may or may not reach the server immediately. However, this does not block other operations. Thus making Node.js highly scalable. Traditional servers create limited threads to handle requests while Node.js creates a single thread that provides service to much larger numbers of such requests.
* **Node.js library uses JavaScript –** This is another important aspect of Node.js from the developer’s point of view. The majority of developers are already well-versed in JavaScript. Hence, development in Node.js becomes easier for a developer who knows JavaScript.
* **There is an Active and vibrant community for the Node.js framework –** The active community always keeps the framework updated with the latest trends in the web development.
* **No Buffering –** Node.js applications never buffer any data. They simply output the data in chunks.

### Q-3. Explain How Do We Decide, When To Use Node.Js And When Not To Use It?

**Answer.**

#### When Should We Use Node.Js?

It’s ideal to use Node.js for developing streaming or event-based real-time applications that require less CPU usage such as.

* Chat applications.
* Game servers.

Node.js is good for fast and high-performance servers, that face the need to handle thousands of user requests simultaneously.

#### Good For A Collaborative Environment.

It is suitable for environments where multiple people work together. For example, they post their documents, modify them by doing check-out and check-in of these documents.

Node.js supports such situations by creating an event loop for every change made to the document. The “Event loop” feature of Node.js enables it to handle multiple events simultaneously without getting blocked.

#### Advertisement Servers.

Here again, we have servers that handle thousands of request for downloading advertisements from a central host. And Node.js is an ideal solution to handle such tasks.

#### Streaming Servers.

Another ideal scenario to use Node.js is for multimedia streaming servers where clients fire request’s towards the server to download different multimedia contents from it.

To summarize, it’s good to use Node.js, when you need high levels of concurrency but less amount of dedicated CPU time.

Last but not the least, since Node.js uses JavaScript internally, so it fits best for building client-side applications that also use JavaScript.

#### When To Not Use Node.Js?

However, we can use Node.js for a variety of applications. But it is a single threaded framework, so we should not use it for cases where the application requires long processing time. If the server is doing some calculation, it won’t be able to process any other requests. Hence, Node.js is best when processing needs less dedicated CPU time.

# Top 30 Node.Js Interview Questions With Answers

[Web Development](http://www.techbeamers.com/category/interview-questions/web-development/)  Published: February 15, 2017  [Meenakshi Agarwal](http://www.techbeamers.com/author/meenakshi/" \o "Posts by Meenakshi Agarwal)

Are you a MEAN stack developer and have an interview lined up? Don’t worry, just read the top 30 Node.js interview questions we’ve outlined in this post.

We’ve tried our best to keep the answers easy to understand and simple to remember. All of our questions are thoroughly researched and have a potential to surface in most Node.js interviews.

However, you would also prepare yourself for **[AngularJS interview questions](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/" \t "_blank)** as it is the next most important skill you’ll need to become an ideal MEAN stack developer. In out next posts, we’ll also cover other areas where you need to centralize your focus.

Let’s now begin to read to the most important Node.js interview questions and answers.

## Top 30 Node.Js Interview Questions & Answers.

**Prepare for Node.js Interview.**

### Q-1. What Is Node.Js?

**Answer.**

Node.js is a JavaScript runtime or platform which is built on Google Chrome’s JavaScript v8 engine. This runtime allows executing the JavaScript code on any machine outside a browser (this means that it is the server that executes the Javascript and not the browser).

Node.js is single-threaded, that employs a concurrency model based on an event loop. It doesn’t block the execution instead registers a callback which allows the application to continue. It means Node.js can handle concurrent operations without creating multiple threads of execution so can scale pretty well.

It uses JavaScript along with C/C++ for things like interacting with the filesystem, starting up HTTP or TCP servers and so on. Due to it’s extensively fast growing community and NPM, Node.js has become a very popular, open source and cross-platform app. It allows developing very fast and scalable network app that can run on Microsoft Windows, Linux, or OS X.

Following are the areas where it’s perfect to use Node.js.

* I/O bound Applications
* Data Streaming Applications
* Data Intensive Real-time Applications (DIRT)
* JSON APIs based Applications
* Single Page Applications

At the same time, it’s not suitable for heavy applications involving more of CPU usage.

### Q-2. What Are The Key Features Of Node.Js?

**Answer.**

Let’s look at some of the key features of Node.js.

* **Asynchronous event driven IO helps concurrent request handling –** All APIs of Node.js are asynchronous. This feature means that if a Node receives a request for some Input/Output operation, it will execute that operation in the background and continue with the processing of other requests. Thus it will not wait for the response from the previous requests.
* **Fast in Code execution –** Node.js uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node.js also become faster.
* **Single Threaded but Highly Scalable –** Node.js uses a single thread model for event looping. The response from these events may or may not reach the server immediately. However, this does not block other operations. Thus making Node.js highly scalable. Traditional servers create limited threads to handle requests while Node.js creates a single thread that provides service to much larger numbers of such requests.
* **Node.js library uses JavaScript –** This is another important aspect of Node.js from the developer’s point of view. The majority of developers are already well-versed in JavaScript. Hence, development in Node.js becomes easier for a developer who knows JavaScript.
* **There is an Active and vibrant community for the Node.js framework –** The active community always keeps the framework updated with the latest trends in the web development.
* **No Buffering –** Node.js applications never buffer any data. They simply output the data in chunks.

### Q-3. Explain How Do We Decide, When To Use Node.Js And When Not To Use It?

**Answer.**

#### When Should We Use Node.Js?

It’s ideal to use Node.js for developing streaming or event-based real-time applications that require less CPU usage such as.

* Chat applications.
* Game servers.

Node.js is good for fast and high-performance servers, that face the need to handle thousands of user requests simultaneously.

#### Good For A Collaborative Environment.

It is suitable for environments where multiple people work together. For example, they post their documents, modify them by doing check-out and check-in of these documents.

Node.js supports such situations by creating an event loop for every change made to the document. The “Event loop” feature of Node.js enables it to handle multiple events simultaneously without getting blocked.

#### Advertisement Servers.

Here again, we have servers that handle thousands of request for downloading advertisements from a central host. And Node.js is an ideal solution to handle such tasks.

#### Streaming Servers.

Another ideal scenario to use Node.js is for multimedia streaming servers where clients fire request’s towards the server to download different multimedia contents from it.

To summarize, it’s good to use Node.js, when you need high levels of concurrency but less amount of dedicated CPU time.

Last but not the least, since Node.js uses JavaScript internally, so it fits best for building client-side applications that also use JavaScript.

#### When To Not Use Node.Js?

However, we can use Node.js for a variety of applications. But it is a single threaded framework, so we should not use it for cases where the application requires long processing time. If the server is doing some calculation, it won’t be able to process any other requests. Hence, Node.js is best when processing needs less dedicated CPU time.

### Q-4. What IDEs Can You Use For Node.Js Development?

**Answer.**

Here is the list of most commonly used IDEs for developing node.js applications.

#### Cloud9.

It is a free, cloud-based IDE that supports, application development, using popular programming languages like Node.js, PHP, C++, Meteor and more. It provides a powerful online code editor that enables a developer to write, run and debug the app code.

#### JetBrains WebStorm.

WebStorm is a lightweight yet powerful JavaScript IDE, perfectly equipped for doing client-side and server-side development using Node.js. The IDE provides features like intelligent code completion, navigation, automated and safe refactorings. Additionally, we can use the debugger, VCS, terminal and other tools present in the IDE.

#### JetBrains InteliJ IDEA.

It is a robust IDE that supports web application development using mainstream technologies like Node.js, Angular.js, JavaScript, HTML5 and more. To enable the IDE that can do Node.js development we have to install a Node.js plugin. It provides features, including syntax highlighting, code assistance, code completion and more. We can even run and debug Node.js apps and see the results right in the IDE. It’s JavaScript debugger offers conditional breakpoints, expression evaluation, and other features.

#### Komodo IDE.

It is a cross-platform IDE that supports development in main programming languages, like Node.js, Ruby, PHP, JavaScript and more. It offers a variety of features, including syntax highlighting, keyboard shortcuts, collapsible Pane, workspace, auto indenting, code folding and code preview using built-in browser.

#### Eclipse.

It is a popular cloud-based IDE for web development using Java, PHP, C++ and more. You can easily avail the features of Eclipse IDE using the Node.js plug-in, which is <**nodeclipse**>.

#### Atom.

It is an open source application built with the integration of HTML, JavaScript, CSS, and Node.js. It works on top of Electron framework to develop cross-platform apps using web technologies. Atom comes pre-installed with four UI and eight syntax themes in both dark and light colors. We can also install themes created by the Atom community or create our own if required.

### Q-5. Explain How Does Node.Js Work?

**Answer.**

A Node.js application creates a single thread on its invocation. Whenever Node.js receives a request, it first completes its processing before moving on to the next request.

Node.js works asynchronously by using the event loop and callback functions, to handle multiple requests coming in parallel. An Event Loop is a functionality which handles and processes all your external events and just converts them to a callback function. It invokes all the event handlers at a proper time. Thus, lots of work is done on the back-end, while processing a single request, so that the new incoming request doesn’t have to wait if the processing is not complete.

While processing a request, Node.js attaches a callback function to it and moves it to the back-end. Now, whenever its response is ready, an event is called which triggers the associated callback function to send this response.

#### Let’s Take An Example Of A Grocery Delivery.

Usually, the delivery boy goes to each and every house to deliver the packet. Node.js works in the same way and processes one request at a time. The problem arises when any one house is not open. The delivery boy can’t stop at one house and wait till it gets opened up. What he will do next, is to call the owner and ask him to call when the house is open. Meanwhile, he is going to other places for delivery. Node.js works in the same way. It doesn’t wait for the processing of the request to complete (house is open). Instead, it attaches a callback function (call from the owner of the house) to it. Whenever the processing of a request completes (the house is open), an event gets called, which triggers the associated callback function to send the response.

# Top 30 Node.Js Interview Questions With Answers

[Web Development](http://www.techbeamers.com/category/interview-questions/web-development/)  Published: February 15, 2017  [Meenakshi Agarwal](http://www.techbeamers.com/author/meenakshi/" \o "Posts by Meenakshi Agarwal)

Are you a MEAN stack developer and have an interview lined up? Don’t worry, just read the top 30 Node.js interview questions we’ve outlined in this post.

We’ve tried our best to keep the answers easy to understand and simple to remember. All of our questions are thoroughly researched and have a potential to surface in most Node.js interviews.

However, you would also prepare yourself for **[AngularJS interview questions](http://www.techbeamers.com/latest-angularjs-interview-questions-answers/" \t "_blank)** as it is the next most important skill you’ll need to become an ideal MEAN stack developer. In out next posts, we’ll also cover other areas where you need to centralize your focus.

Let’s now begin to read to the most important Node.js interview questions and answers.

## Top 30 Node.Js Interview Questions & Answers.

**Prepare for Node.js Interview.**

### Q-1. What Is Node.Js?

**Answer.**

Node.js is a JavaScript runtime or platform which is built on Google Chrome’s JavaScript v8 engine. This runtime allows executing the JavaScript code on any machine outside a browser (this means that it is the server that executes the Javascript and not the browser).

Node.js is single-threaded, that employs a concurrency model based on an event loop. It doesn’t block the execution instead registers a callback which allows the application to continue. It means Node.js can handle concurrent operations without creating multiple threads of execution so can scale pretty well.

It uses JavaScript along with C/C++ for things like interacting with the filesystem, starting up HTTP or TCP servers and so on. Due to it’s extensively fast growing community and NPM, Node.js has become a very popular, open source and cross-platform app. It allows developing very fast and scalable network app that can run on Microsoft Windows, Linux, or OS X.

Following are the areas where it’s perfect to use Node.js.

* I/O bound Applications
* Data Streaming Applications
* Data Intensive Real-time Applications (DIRT)
* JSON APIs based Applications
* Single Page Applications

At the same time, it’s not suitable for heavy applications involving more of CPU usage.

### Q-2. What Are The Key Features Of Node.Js?

**Answer.**

Let’s look at some of the key features of Node.js.

* **Asynchronous event driven IO helps concurrent request handling –** All APIs of Node.js are asynchronous. This feature means that if a Node receives a request for some Input/Output operation, it will execute that operation in the background and continue with the processing of other requests. Thus it will not wait for the response from the previous requests.
* **Fast in Code execution –** Node.js uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node.js also become faster.
* **Single Threaded but Highly Scalable –** Node.js uses a single thread model for event looping. The response from these events may or may not reach the server immediately. However, this does not block other operations. Thus making Node.js highly scalable. Traditional servers create limited threads to handle requests while Node.js creates a single thread that provides service to much larger numbers of such requests.
* **Node.js library uses JavaScript –** This is another important aspect of Node.js from the developer’s point of view. The majority of developers are already well-versed in JavaScript. Hence, development in Node.js becomes easier for a developer who knows JavaScript.
* **There is an Active and vibrant community for the Node.js framework –** The active community always keeps the framework updated with the latest trends in the web development.
* **No Buffering –** Node.js applications never buffer any data. They simply output the data in chunks.

### Q-3. Explain How Do We Decide, When To Use Node.Js And When Not To Use It?

**Answer.**

#### When Should We Use Node.Js?

It’s ideal to use Node.js for developing streaming or event-based real-time applications that require less CPU usage such as.

* Chat applications.
* Game servers.

Node.js is good for fast and high-performance servers, that face the need to handle thousands of user requests simultaneously.

#### Good For A Collaborative Environment.

It is suitable for environments where multiple people work together. For example, they post their documents, modify them by doing check-out and check-in of these documents.

Node.js supports such situations by creating an event loop for every change made to the document. The “Event loop” feature of Node.js enables it to handle multiple events simultaneously without getting blocked.

#### Advertisement Servers.

Here again, we have servers that handle thousands of request for downloading advertisements from a central host. And Node.js is an ideal solution to handle such tasks.

#### Streaming Servers.

Another ideal scenario to use Node.js is for multimedia streaming servers where clients fire request’s towards the server to download different multimedia contents from it.

To summarize, it’s good to use Node.js, when you need high levels of concurrency but less amount of dedicated CPU time.

Last but not the least, since Node.js uses JavaScript internally, so it fits best for building client-side applications that also use JavaScript.

#### When To Not Use Node.Js?

However, we can use Node.js for a variety of applications. But it is a single threaded framework, so we should not use it for cases where the application requires long processing time. If the server is doing some calculation, it won’t be able to process any other requests. Hence, Node.js is best when processing needs less dedicated CPU time.

### Q-4. What IDEs Can You Use For Node.Js Development?

**Answer.**

Here is the list of most commonly used IDEs for developing node.js applications.

#### Cloud9.

It is a free, cloud-based IDE that supports, application development, using popular programming languages like Node.js, PHP, C++, Meteor and more. It provides a powerful online code editor that enables a developer to write, run and debug the app code.

#### JetBrains WebStorm.

WebStorm is a lightweight yet powerful JavaScript IDE, perfectly equipped for doing client-side and server-side development using Node.js. The IDE provides features like intelligent code completion, navigation, automated and safe refactorings. Additionally, we can use the debugger, VCS, terminal and other tools present in the IDE.

#### JetBrains InteliJ IDEA.

It is a robust IDE that supports web application development using mainstream technologies like Node.js, Angular.js, JavaScript, HTML5 and more. To enable the IDE that can do Node.js development we have to install a Node.js plugin. It provides features, including syntax highlighting, code assistance, code completion and more. We can even run and debug Node.js apps and see the results right in the IDE. It’s JavaScript debugger offers conditional breakpoints, expression evaluation, and other features.

#### Komodo IDE.

It is a cross-platform IDE that supports development in main programming languages, like Node.js, Ruby, PHP, JavaScript and more. It offers a variety of features, including syntax highlighting, keyboard shortcuts, collapsible Pane, workspace, auto indenting, code folding and code preview using built-in browser.

#### Eclipse.

It is a popular cloud-based IDE for web development using Java, PHP, C++ and more. You can easily avail the features of Eclipse IDE using the Node.js plug-in, which is <**nodeclipse**>.

#### Atom.

It is an open source application built with the integration of HTML, JavaScript, CSS, and Node.js. It works on top of Electron framework to develop cross-platform apps using web technologies. Atom comes pre-installed with four UI and eight syntax themes in both dark and light colors. We can also install themes created by the Atom community or create our own if required.

### Q-5. Explain How Does Node.Js Work?

**Answer.**

A Node.js application creates a single thread on its invocation. Whenever Node.js receives a request, it first completes its processing before moving on to the next request.

Node.js works asynchronously by using the event loop and callback functions, to handle multiple requests coming in parallel. An Event Loop is a functionality which handles and processes all your external events and just converts them to a callback function. It invokes all the event handlers at a proper time. Thus, lots of work is done on the back-end, while processing a single request, so that the new incoming request doesn’t have to wait if the processing is not complete.

While processing a request, Node.js attaches a callback function to it and moves it to the back-end. Now, whenever its response is ready, an event is called which triggers the associated callback function to send this response.

#### Let’s Take An Example Of A Grocery Delivery.

Usually, the delivery boy goes to each and every house to deliver the packet. Node.js works in the same way and processes one request at a time. The problem arises when any one house is not open. The delivery boy can’t stop at one house and wait till it gets opened up. What he will do next, is to call the owner and ask him to call when the house is open. Meanwhile, he is going to other places for delivery. Node.js works in the same way. It doesn’t wait for the processing of the request to complete (house is open). Instead, it attaches a callback function (call from the owner of the house) to it. Whenever the processing of a request completes (the house is open), an event gets called, which triggers the associated callback function to send the response.

To summarize, Node.js does not process the requests in parallel. Instead, all the back-end processes like, I/O operations, heavy computation tasks, that take a lot of time to execute, run in parallel with other requests.

### Q-6. Explain REPL In Node.Js?

**Answer.**

The REPL stands for “Read Eval Print Loop”. It is a simple program that accepts the commands, evaluates them, and finally prints the results. REPL provides an environment similar to that of Unix/Linux shell or a window console, in which we can enter the command and the system, in turn, responds with the output. REPL performs the following tasks.

* **READ**
  + It Reads the input from the user, parses it into JavaScript data structure and then stores it in the memory.
* **EVAL**
  + It Executes the data structure.
* **PRINT**
  + It Prints the result obtained after evaluating the command.
* **LOOP**
  + It Loops the above command until the user presses Ctrl+C two times.

### Q-7. Is Node.Js Entirely Based On A Single-Thread?

**Answer.**

Yes, it’s true that Node.js processes all requests on a single thread. But it’s just a part of the theory behind Node.js design. In fact, more than the single thread mechanism, it makes use of events and callbacks to handle a large no. of requests asynchronously.

Moreover, Node.js has an optimized design which utilizes both JavaScript and C++ to guarantee maximum performance. JavaScript executes at the server-side by Google Chrome v8 engine. And the C++ lib UV library takes care of the non-sequential I/O via background workers.

To explain it practically, let’s assume there are 100s of requests lined up in Node.js queue. As per design, the main thread of Node.js event loop will receive all of them and forwards to background workers for execution. Once the workers finish processing requests, the registered callbacks get notified on event loop thread to pass the result back to the user.

### Q-10. What Is Callback In Node.Js?

**Answer.**

We may call “callback” as an asynchronous equivalent for a function. Node.js makes heavy use of callbacks and triggers it at the completion of a given task. All the APIs of Node.js are written in such a way that they support callbacks.

For example, suppose we have a function to read a file, as soon as it starts reading the file, Node.js return the control immediately to the execution environment so that the next instruction can be executed. Once file read operation is complete, it will call the callback function and pass the contents of the file as its arguments. Hence, there is no blocking or wait, due to File I/O. This functionality makes Node.js as highly scalable, using it processes a high number of requests without waiting for any function to return the expected result.

### Q-11. What Is Callback Hell?

**Answer.**

Initially, you may praise Callback after learning about it. Callback hell is heavily nested callbacks which make the code unreadable and difficult to maintain.

Let’s see the following code example.

downloadPhoto('http://coolcats.com/cat.gif', displayPhoto)

function displayPhoto (error, photo) {

if (error) console.error('Download error!', error)

else console.log('Download finished', photo)

}

console.log('Download started')

In this scenario, Node.js first declares the “displayPhoto” function. After that, it calls the “downloadPhoto” function and pass the “displayPhoto” function as its callback. Finally, the code prints ‘Download started’ on the console. The “displayPhoto” will be executed only after “downloadPhoto” completes the execution of all its tasks.

### Q-12. How To Avoid Callback Hell In Node.Js?

**Answer.**

Node.js internally uses a single-threaded event loop to process queued events. But this approach may lead to blocking the entire process if there is a task running longer than expected.

Node.js addresses this problem by incorporating callbacks also known as higher-order functions. So whenever a long-running process finishes its execution, it triggers the callback associated. With this approach, it can allow the code execution to continue past the long-running task.

However, the above solution looks extremely promising. But sometimes, it could lead to complex and unreadable code. More the no. of callbacks, longer the chain of returning callbacks would be. Just see the below example.

With such an unprecedented complexity, it’s hard to debug the code and can cause you a whole lot of time. There are four solutions which can address the callback hell problem.

#### 1. Make Your Program Modular.

It proposes to split the logic into smaller modules. And then join them together from the main module to achieve the desired result.

#### 2. Use Async Mechanism.

It is a widely used Node.js module which provides a sequential flow of execution.

The async module has <async.waterfall> API which passes data from one operation to other using the next callback.

Another async API <async.map> allows iterating over a list of items in parallel and calls back with another list of results.

With the async approach, the caller’s callback gets called only once. The caller here is the main method using the async module.

#### 3. Use Promises Mechanism.

Promises give an alternate way to write async code. They either return the result of execution or the error/exception. Implementing promises requires the use of <.then()> function which waits for the promise object to return. It takes two optional arguments, both functions. Depending on the state of the promise only one of them will get called. The first function call proceeds if the promise gets fulfilled. However, if the promise gets rejected, then the second function will get called.

#### 4. Use Generators.

Generators are lightweight routines, they make a function wait and resume via the yield keyword. Generator functions uses a special syntax <function\* ()>. They can also suspend and resume asynchronous operations using constructs such as promises or <thunks> and turn a synchronous code into asynchronous.

### Q-14. What Is The Difference Between Nodejs, AJAX, And JQuery?

**Answer.**

The one common trait between Node.js, AJAX, and jQuery is that all of them are the advanced implementation of JavaScript. However, they serve completely different purposes.

#### Node.Js –

It is a server-side platform for developing client-server applications. For example, if we’ve to build an online employee management system, then we won’t do it using client-side JS. But the Node.js can certainly do it as it runs on a server similar to Apache, Django not in a browser.

#### AJAX (Aka Asynchronous Javascript And XML) –

It is a client-side scripting technique, primarily designed for rendering the contents of a page without refreshing it. There are a no. of large companies utilizing AJAX such as Facebook and Stack Overflow to display dynamic content.

#### JQuery –

It is a famous JavaScript module which complements AJAX, DOM traversal, looping and so on. This library provides many useful functions to help in JavaScript development. However, it’s not mandatory to use it but as it also manages cross-browser compatibility, so can help you produce highly maintainable web applications.

Q-19. List And Explain The Important REPL Commands?

**Answer.**

Following is the list of some of the most commonly used REPL commands.

* **<.help> –** It displays help for all the commands.
* **<tab Keys> –** It displays the list of all the available commands.
* **<Up/Down Keys> –** Its use is to determine what command was executed in REPL previously.
* **<.save filename> –** Save the current REPL session to a file.
* **<.load filename> –** To Load the specified file in the current REPL session.
* **<ctrl + c> –** used to Terminate the current command.
* **<ctrl + c (twice)> –** To Exit from the REPL.
* **<ctrl + d> –** This command perfoms Exit from the REPL.
* **<.break> –** It leads Exitting from multiline expression.
* **<.clear> –** Exit from multiline expression.

Q-20. What Is NPM In Node.Js?

**Answer.**

NPM stands for Node Package Manager. It provides following two main functionalities.

* It works as an Online repository for node.js packages/modules which are present at <nodejs.org>.
* It works as Command line utility to install packages, do version management and dependency management of Node.js packages.

NPM comes bundled along with Node.js installable. We can verify its version using the following command-

$ npm --version

NPM helps to install any Node.js module using the following command.

$ npm install <Module Name>

For example, following is the command to install a famous Node.js web framework module called express-

$ npm install express

### Q-23. What Is Package.Json? Who Uses It?

**Answer.**

#### What Is <Package.Json>?

* It is a plain JSON (JavaScript Object Notation) text file which contains all metadata information about Node.js Project or application.
* This file should be present in the root directory of every Node.js Package or Module to describe its metadata in JSON format.
* The file is named as “package” because Node.js platform treats every feature as a separate component. Node.js calls these as Package or Module.

#### Who Use It?

* NPM (Node Package Manager) uses <package.json> file. It includes details of the Node.js application or package. This file contains a no. of different directives or elements. These directives guide NPM, about how to handle a module or package.

### Q-24. Does Node.Js Support Multi-Core Platforms? And Is It Capable Of Utilizing All The Cores?

**Answer.**

Yes, Node.js would run on a multi-core system without any issue. But it is by default a single-threaded application, so it can’t completely utilize the multi-core system.

However, Node.js can facilitate deployment on multi-core systems where it does use the additional hardware. It packages with a Cluster module which is capable of starting multiple Node.js worker processes that will share the same port.

### Q-26. What Is Chaining Process In Node.Js?

**Answer.**

It’s an approach to connect the output of one stream to the input of another stream, thus creating a chain of multiple stream operations.

Q-30. What Is A Control Flow Function? What Are The Steps Does It Execute?

**Answer.**

It is a generic piece of code which runs in between several asynchronous function calls is known as control flow function.

It executes the following steps.

* Control the order of execution.
* Collect data.
* Limit concurrency.
* Call the next step in the program.

**1) What is Node.js?**

Node.js is a very powerful JavaScript based platform or framework which is built on Google Chrome's JavaScript V8 Engine.

**2) Why to use Node.js?**

It is used to develop I/O intensive web applications like video streaming sites, single page applications (SPA) and other web applications. Node.js is open source and used by thousands of developers around the world.

**3) Who developed Node.js?**

Node.js was developed in 2009 by Ryan Dahl.

**4) What are the features of Node.js?**

Below are the features of Node.js –

* Very Fast
* Event driven and Asynchronous
* Single Threaded but highly Scalable

**5) Explain REPL in Node.js?**

REPL stands for Read Eval Print Loop. Node.js comes with bundled REPL environment which performs the following desired tasks –

* Eval
* Print
* Loop
* Read

**6) Explain variables in Node.js?**

Variables are used to store values and print later like any conventional scripts. If “var” keyword is used then value is stored in variable. You can print the value in the variable using - console.log().

Eg:  
$ node  
> a = 30  
30  
> var b = 50  
undefined  
> a + b  
80  
> console.log("Hi")  
Hi  
undefined

**7) What is the latest version of Node.js available?**

Latest version of Node.js is - v0.10.36.

**8) List out some REPL commands in Node.js?**

Below are the list of REPL commands –

* Ctrl + c - For terminating the current command.
* Ctrl + c twice – For terminating REPL.
* Ctrl + d - For terminating REPL.
* Tab Keys - list of all the current commands.
* .break - exit from multiline expression.
* .save with filename - save REPL session to a file.

**9) Mention the command to stop REPL in Node.js?**

Command - ctrl + c twice is used to stop REPL.

**10) Explain NPM in Node.js?**

NPM stands for Node Package Manager (npm) and there are two functionalities which NPM takes care of mainly and they are –

* Online repositories for node.js modules or packages, which can be searched on search.nodejs.org
* Dependency Management, Version Management and command line utility for installing Node.js packages.

**11) Mention command to verify the NPM version in Node.js?**

Below command can be used to verify the NPM version –

$ npm --version

**12) How you can update NPM to new version in Node.js?**

Below commands can be used for updating NPM to new version –

$ sudo npm install npm -g  
/usr/bin/npm -> /usr/lib/node\_modules/npm/bin/npm-cli.js  
npm@2.7.1 /usr/lib/node\_modules/npm

**13) Explain callback in Node.js?**

Callback is called once the asynchronous operation has been completed. Node.js heavily uses callbacks and all API’s of Node.js are written to support callbacks.

**14) How Node.js can be made more scalable?**

Node.js works good for I/O bound and not CPU bound work. For instance if there is a function to read a file, file reading will be started during that instruction and then it moves onto next instruction and once the I/O is done or completed it will call the callback function. So there will not be any blocking.

**15) Explain global installation of dependencies?**

Globally installed dependencies or packages are stored in <user-directory>/npm directory and these dependencies can be used in Command Line Interface function of any node.js.

**16) Explain local installation of dependencies?**

By default npm will install the dependency in the local mode. Here local mode refers to the package installation in node\_modules directory lying in the folder where Node application is present. “require ()” is used to access the locally deployed packages.

**17) Explain Package.JSON?**

This will be present in the root directory of any Node module/application and will be used to define the properties of a package.

**18) Explain “Callback hell”?**

“Callback hell” will be referred to heavily nested callbacks which has become unreadable or unwieldly.

**19) What are “Streams” in Node.JS?**

“Streams” are objects which will let you read the data from source and write data to destination as a continuous process.

**20) What you mean by chaining in Node.JS?**

It’s a mechanism in which output of one stream will be connected to another stream and thus creating a chain of multiple stream operations.

**21) Explain Child process module?**

Child process module has following three major ways to create child processes –

* spawn  - child\_process.spawn launches a new process with a given command.
* exec  - child\_process.exec method runs a command in a shell/console and buffers the output.
* fork - The child\_process.fork method is a special case of the spawn() to create child processes.

**22) Why to use exec method for Child process module?**

“exec” method runs a command in a shell and buffers the output. Below is the command –

child\_process.exec(command[, options], callback)

**23) List out the parameters passed for Child process module?**

Below are the list of parameters passed for Child Process Module –

child\_process.exec(command[, options], callback)

* command - This is the command to run with space-separated arguments.
* options – This is an object array which comprises one or more following options –
* cwd
* uid
* gid
* killSignal
* maxBuffer
* encoding
* env
* shell
* timeout

callback – This is the function which is gets 2 arguments – stdout, stderr and error.

**24) What is the use of method – “spawn()”?**

This method is used to launch a new process with the given commands. Below is the method signature –

child\_process.spawn(command[, args][, options])

**25) What is the use of method – “fork()”?**

This method is a special case for method- “spawn()” for creating node processes. The method signature –

child\_process.fork(modulePath[, args][, options])

**26) Explain Piping Stream?**

This is a mechanism of connecting one stream to other and this is basically used for getting the data from one stream and pass the output of this to other stream.

**27) What would be the limit for Piping Stream?**

There will not be any limit for piping stream.

**28) Explain FS module ?**

Here FS stands for “File System” and fs module is used for File I/O. FS module can be imported in the following way –

var test = require("fs")

**29) Explain “Console” in Node.JS?**

“Console” is a global object and will be used for printing to stderr and stdout and this will be used in synchronous manner in case of destination is either file or terminal or else it is used in asynchronous manner when it is a pipe.

**30) Explain – “console.log([data][, ...])” statement in Node.JS?**

This statement is used for printing to “stdout” with newline and this function takes multiple arguments as “printf()”.

**31) What you mean by “process”?**

“process” is a global object and will be used to represent a node process.

**32) Explain exit codes in Node.JS? List out some exit codes?**

Exit code will be used when the process needs to be ended with specified code. Below are the list of exit codes in Node.JS –

* Fatal Error
* Non-function Internal Exception Handler
* Internal JavaScript Parse Error
* Uncaught Fatal Exception
* Unused
* Internal JavaScript Evaluation Failure
* Internal Exception Handler Run-Time Failure

**33) List out the properties of process?**

Below are the useful properties of process –

* Platform
* Stdin
* Stdout
* Stderr
* execPath
* mainModule
* execArgv
* config
* arch
* title
* version
* argv
* env
* exitCode

**34) Define OS module?**

OS module is used for some basic operating system related utility functions. Below is the syntax for importing OS module –

var MyopSystem = require("os")

**35) What is the property of OS module?**

os.EOL – Constant for defining appropriate end of line marker for OS.

**36) Explain “Path” module in Node.JS?**

“Path” module will be used for transforming and handling file paths. Below is the syntax of path module –

var mypath = require("path")

**37) Explain “Net” module in Node.JS?**

“Net” module is being used for creating both clients and servers. It will provide asynchronous network wrapper. Below is the syntax of Net module –

var mynet = require("net")

**38) List out the differences between AngularJS and NodeJS?**

AngularJS is a web application development framework. It’s a JavaScript and it is different from other web app frameworks written in JavaScript like jQuery. NodeJS is a runtime environment used for building server-side applications while AngularJS is a JavaScript framework mainly useful in building/developing client-side part of applications which run inside a web browser.

**39) NodeJS is client side server side language?**

NodeJS is a runtime system, which is used for creating server-side applications.

**40) What are the advantages of NodeJS?**

Below are the list of advantages of NodeJS –

* Javascript – It’s a javascript which can be used on frontend and backend.
* Community Driven - NodeJS has great open source community which has developed many excellent modules for NodeJS to add additional capabilities to NodeJS applications.

**41) In which scenarios NodeJS works well?**

NodeJS is not appropriate to use in scenarios where single-threaded calculations are going to be the holdup.

**42) What you mean by JSON?**

JavaScript Object Notation (JSON) is a practical, compound, widely popular data exchange format. This will enable JavaScript developers to quickly construct APIs.

**43) Explain “Stub”?**

Stub is a small program, which substitutes for a longer program, possibly to be loaded later and that is located remotely. Stubs are functions/programs that simulate the behaviors of components/modules.

**44) List out all Node.JS versions available?**

Below are the list of all NodsJS versions supported in operating systems –

|  |  |
| --- | --- |
| **OperatingSystem** | **Node.js version** |
| Windows | node-v0.12.0-x64.msi |
| Linux | node-v0.12.0-linux-x86.tar.gz |
| Mac | node-v0.12.0-darwin-x86.tar.gz |
| SunOS | node-v0.12.0-sunos-x86.tar.gz |

**45) Explain “Buffer class” in Node.JS?**

It is a global class which can be accessed in an application without importing buffer modules.

**46) How we can convert Buffer to JSON?**

The syntax to convert Buffer to JSON is as shown beow

buffer.toJSON()

**47) How to concatenate buffers in NodeJS?**

The syntax to concatenate buffers in NodeJS is

var MyConctBuffer = Buffer.concat([myBuffer1, myBuffer2]);

**48) How to compare buffers in NodeJS?**

To compare buffers in NodeJS, use following code –

Mybuffer1.compare(Mybuffer2);

**49) How to copy buffers in NodeJS?**

Below is the syntax to copy buffers in NodeJS –

buffer.copy(targetBuffer[, targetStart][, sourceStart][, sourceEnd])

**50) What are the differences between “readUIntBE” and “writeIntBE” in Node.JS?**

* readUIntBE - It’s a generalized version of all numeric read methods, which supports up to 48 bits accuracy. Setting noAssert to “true” to skip the validation.
* writeIntBE - This will write the value to the buffer at the specified byteLength and offset and it supports upto 48 bits of accuracy.

**51) Why to use “\_\_filename” in Node.JS?**

“\_\_filename” is used to represent the filename of the code which is being executed. It used to resolve the absolute path of file. Below is the sample code for the same –

Console.log(\_\_filename);

**52) Why to use “SetTimeout” in Node.JS?**

This is the global function and it is used to run the callback after some milliseconds.

Syntax of this method –

setTimeout(callbackmethod, millisecs)

**53) Why to use “ClearTimeout” in Node.JS?**

This is the global function and it is used to stop a timer which was created during “settimeout()”.

**54) Explain Web Server?**

It is a software app which will handle the HTTP requests by client (eg: browser) and will return web pages to client as a response. Most of web server supports – server side scripts using scripting languages. Example of web server is Apache, which is mostly used webserver.

**55) List out the layers involved in Web App Architechure?**

Below are the layers used in Web Apps –

* Client - Which makes HTTP request to the server. Eg: Browsers.
* Server – This layer is used to intercept the requests from client.
* Business – It will have application server utilized by web servers for processing.
* Data – This layer will have databases mainly or any source of data.

**56) Explain “Event Emitter” in Node.JS?**

It is a part of Events module. When instance of EventEmitter faces any error, it will emit an 'error' event. “Event Emitters” provides multiple properties like – “emit” and “on”.

* “on” property is used for binding the function with event.
* “emit” property is used for firing an event.

**57) Explain “NewListener” in Node.JS?**

This event is being emitted whenever any listener is added. So when event is triggered the listener may not have been removed from listener array for the event.

**58) Why to use Net.socket in Node.JS?**

This object is an abstraction of a local socket or TCP. net.Socket instances implement a duplex Stream interface. These can be created by the user and used as a client (with connect() function) or they can be created by Node and can be passed to the user through the 'connection' event of a server.

**59) Which events are emitted by Net.socket?**

Below are the list of events emitted by Net.socket –

* Connect
* Lookup
* End
* Data
* Close
* Drain
* Timeout
* Error

**60) Explain “DNS module” in Node.JS?**

This module is used for DNS lookup and to use underlying OS name resolution. This used to provide asynchronous network wrapper. DNS module can be imported like –

var mydns = require("dns")

**61) Explain binding in domain module in Node.JS?**

Below are the bindings in domain modules –

* External Binding
* Internal Binding

**62) Explain RESTful Web Service?**

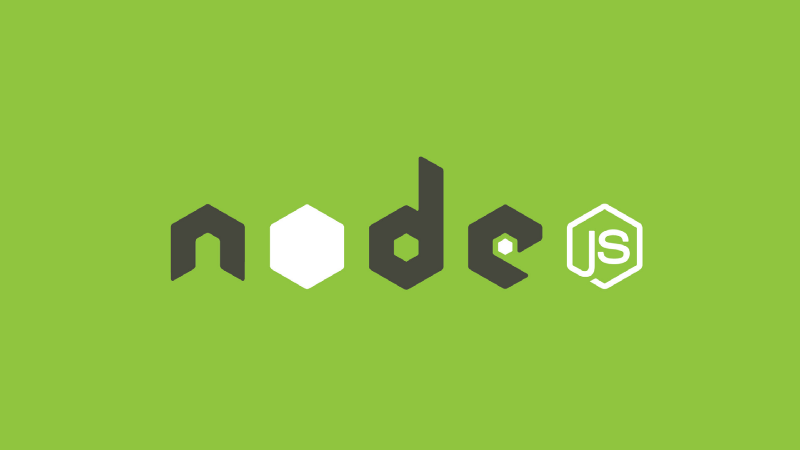
Web services which uses REST architecture will be known as RESTful Web Services. These web services uses HTTP protocol and HTTP methods.

**63) How to truncate the file in Node.JS?**

Below command can be used for truncating the file –

fs.ftruncate(fd, len, callback)

# Frequently asked: Node JS Interview Questions and Answers



**Q1. What is Node.js? What is it used for?**

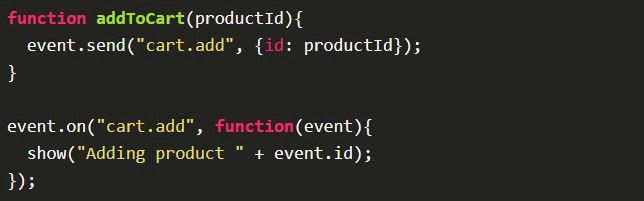
Node.js is a run-time JavaScript environment built on top of Chrome’s V8 engine. It uses an event-driven, non-blocking I/O model. It is lightweight and so efficient. Node.js has a package ecosystem called **npm**.

Node.js can be used to build different types of applications such as web application, real-time chat application, REST API server etc. However, it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET. Node.js was developed by Ryan Dahl in 2009.

**Q2. What is Event-driven programming?**

Event-driven programming is building our application based on and respond to events. When an event occurs, like click or keypress, we are running a callback function which is registered to the element for that event.

Event driven programming follows mainly a publish-subscribe pattern.



**Q3. What is Event loop in Node.js work? And How does it work?**

The Event loop handles all async callbacks. Node.js (or JavaScript) is a single-threaded, event-driven language. This means that we can attach listeners to events, and when a said event fires, the listener executes the callback we provided.

Whenever we are call setTimeout, http.get and fs.readFile, Node.js runs this operations and further conitnue to run other code without waiting for the output. When the operation is finished, it receives the output and runs our callback function.

So all the callback functions are queued in an loop, and will run one-by-one when the response has been received.

**Q4. What is REPL in Node.js?**

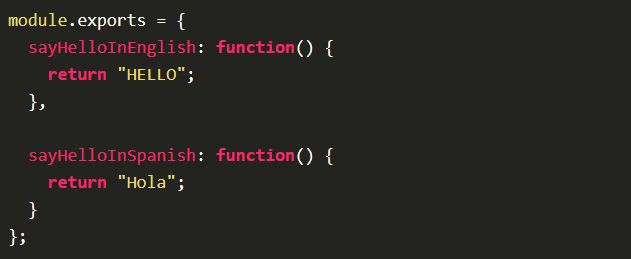
REPL means Read-Eval-Print-Loop. It is a virtual environment that comes with Node.js. We can quickly test our JavaScript code in the Node.js REPL environment.

To launch the REPL in Node.js, just opne the command prompt and type node. It will change the prompt to > in Windows and MAC.

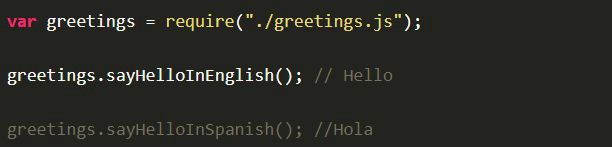
Now we can type and run our JavaScript easily. For example, if we type 10 + 20, it will print 30 in the next line.

**Q5. What is the purpose of module.exports in Node.js?**

A module encapsulates related code into a single unit of code. This can be interpreted as moving all related functions into a file. Imagine that we created a file called greetings.js and it contains the following two functions:



In the above code, module.exports exposes two functions to the outer world. We can import them in another file as follow:



**Q6. What is the difference between Asynchronous and Non-blocking?**

Asynchronous literally means not synchronous. We are making HTTP requests which are asynchronous, means we are not waiting for the server response. We continue with other block and respond to the server response when we received.

The term Non-Blocking is widely used with IO. For example non-blocking read/write calls return with whatever they can do and expect caller to execute the call again. Read will wait until it has some data and put calling thread to sleep.

**Q7. What is Tracing in Node.js?**

Tracing provides a mechanism to collect tracing information generated by V8, Node core and userspace code in a log file. Tracing can be enabled by passing the --trace-events-enabled flag when starting a Node.js application.

https://cdn-images-1.medium.com/max/800/1*gaM6bjJjwacw6o861eTKBA.jpeg

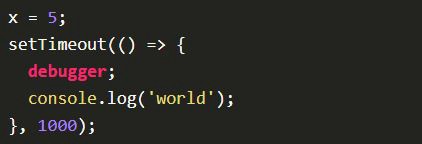
The set of categories for which traces are recorded can be specified using the --trace-event-categories flag followed by a list of comma separated category names. By default the node and v8 categories are enabled.

Running Node.js with tracing enabled will produce log files that can be opened in the chrome://tracing tab of Chrome.

**Q8. How will you debug an application in Node.js?**

Node.js includes a debugging utility called debugger. To enable it start the Node.js with the debug argument followed by the path to the script to debug.

Inserting the statement debugger; into the source code of a script will enable a breakpoint at that position in the code:



**Q9. Difference between setImmediate() vs setTimeout()**

setImmediate() and setTimeout() are similar, but behave in different ways depending on when they are called.

* setImmediate() is designed to execute a script once the current poll (event loop) phase completes.
* setTimeout() schedules a script to be run after a minimum threshold in ms has elapsed.

The order in which the timers are executed will vary depending on the context in which they are called. If both are called from within the main module, then timing will be bound by the performance of the process.

**Q10. What is process.nextTick()**

setImmediate() and setTimeout() are based on the event loop. But process.nextTick() technically not part of the event loop. Instead, the nextTickQueue will be processed after the current operation completes, regardless of the current phase of the event loop.

Thus, any time you call process.nextTick() in a given phase, all callbacks passed to process.nextTick() will be resolved before the event loop continues.

**Q11. What is package.json? What is it used for?**

This file holds various metadata information about the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.

Some of the fields are: name, name, description, author and dependencies.

When someone installs our project through npm, all the dependencies listed will be installed as well. Additionally, if someone runs npm install in the root directory of our project, it will install all the dependencies to ./node\_modules directory.

**Q12. What is libuv?**

libuv is a multi-platform support library with a focus on asynchronous I/O. It was primarily developed for use by Node.js, but it’s also used by Luvit, Julia, pyuv, and others.

When the node.js project began in 2009 as a JavaScript environment decoupled from the browser, it is using Google’s V8 and Marc Lehmann’s libev, node.js combined a model of I/O – evented – with a language that was well suited to the style of programming; due to the way it had been shaped by browsers. As node.js grew in popularity, it was important to make it work on Windows, but libev ran only on Unix. libuv was an abstraction around libev or IOCP depending on the platform, providing users an API based on libev. In the node-v0.9.0 version of libuv libev was removed.

Some of the features of libuv are:

* Full-featured event loop backed by epoll, kqueue, IOCP, event ports.
* Asynchronous TCP and UDP sockets
* Asynchronous file and file system operations
* Child processes
* File system events

**Q13. What are some of the most popular modules of Node.js?**

There are many most popular, most starred or most downloaded modules in Node.js. Some of them are:

* express
* async
* browserify
* socket.io
* bower
* gulp
* grunt

**Q14. What is EventEmitter in Node.js?**

All objects that emit events are instances of the EventEmitter class. These objects expose an eventEmitter.on() function that allows one or more functions to be attached to named events emitted by the object.

When the EventEmitter object emits an event, all of the functions attached to that specific event are called synchronously.



**Q15. What is Streams in Node.js?**

Streams are pipes that let you easily read data from a source and pipe it to a destination. Simply put, a stream is nothing but an EventEmitter and implements some specials methods. Depending on the methods implemented, a stream becomes Readable, Writable, or Duplex (both readable and writable).

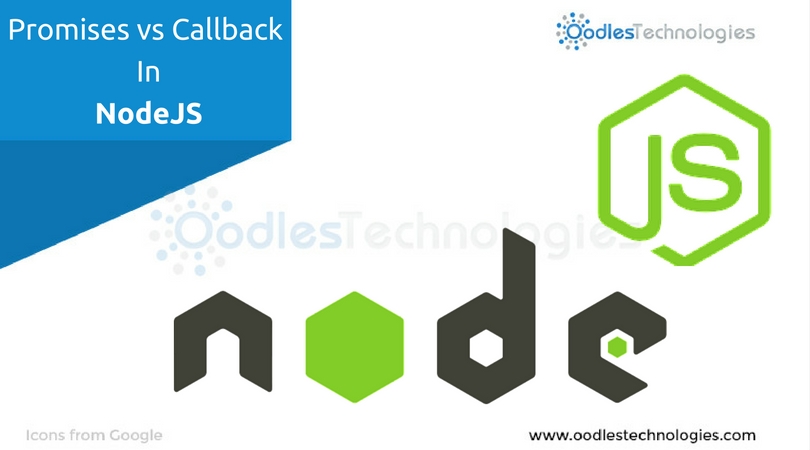
For example, if we want to read data from a file, the best way to do it from a stream is to listen to data event and attach a callback. When a chunk of data is available, the readable stream emits a data event and your callback executes. Take a look at the following snippet:



Types of streams are: Readable, Writable, Duplex and Transform.

# [Promises VS Callback in Nodejs](https://www.oodlestechnologies.com/blogs/Promises-VS-Callback-in-Nodejs)

Posted by [Pankaj Kumar Yadav](https://www.oodlestechnologies.com/blog/filterByAuthor?author=Pankaj+Kumar-Yadav)| Last Updated: 12-Feb-18   
 **41**  **28**  **1**  **8**  **1**  **1**

  
  
Hi All,

In this blog I'm going to show the difference between callback and Promises in Nodejs

First let's start with **callbacks**.

Due to non-blocking I/O, Node is heavy use of callbacks. All the APIs of Nodejs support callbacks.  
A callback is a function called at the completion of a given task. This prevents any blocking, and allows other code to be run in the meantime.

For example - Let there are four functions functionCall, doSomeworkOne, doSomeworkTwo, doSomeworkTwo and they are performing some IO tasks.  
function doSomeworkThree functionCall depends doSomeworkOne, doSomeworkOne depends doSomeworkTwo, doSomeworkTwo depends doSomeworkThree. To make these sync, callback function passed as parameter in all functions.

|  |  |
| --- | --- |
| 1  2  3  4  5 | function functionCall(data, callback){      ...........      ...........      doSomeworkOne(data, callback);  } |
| 1  2  3  4  5 | function doSomeworkOne(data, callback){      ...........      ...........      doSomeworkTwo(otherData, callback);  } |

|  |  |
| --- | --- |
| 1  2  3  4  5 | function doSomeworkTwo(otherData, callback){      ...........      ...........      doSomeworkThree(otherData, callback);  } |
| 1  2  3  4  5  6 | <span style="font-size:16px;"><span style="font-family:arial,helvetica,sans-serif;"> function doSomeworkThree(otherData, callback){      ...........      ...........      callback(result);  }  </span></span> | |

|  |  |
| --- | --- |
| 1  2  3 | function callback(data){      return data  } |

callback is good. The main problem with callbacks is: nested inside of callbacks, nested inside of callbacks. In nested callbacks, it is very tough to test/maintain the codes.  
 Here the Promises comes. Promises provide us with a cleaner and more robust way of handling async code. Instead of using a callback. And also handling errors with promises is very easy.

See the example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | function functionCall(data){      doSomeworkOne(data).then(function(data){          return doSomeworkTwo(data);      }).then(function(data){          return     doSomeworkThree(data);      }).catch(function(e) {          // error handle      });  } |
| 1  2  3  4  5  6  7  8  9  10  11 | function doSomeworkOne(data){      retrun new Promise(function(resolve, reject){          ...........          ...........          if(error){              reject(error);          }else{              resolve(success);          }      })  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | function doSomeworkTwo(data){      retrun new Promise(function(resolve, reject){          ...........          ...........          if(error){              reject(error);          }else{              resolve(success);          }      })  } |
| 1  2  3  4  5  6  7  8  9  10  11 | function doSomeworkThree(data){      retrun new Promise(function(resolve, reject){          ...........          ...........          if(error){              reject(error);          }else{              resolve(success);          }      })  } |

**Note**: Promises and Callbacks are not fundamentally different. Promises is advisable in nested callbacks where you want to perform a series of actions.

avaScript Promises use callback functions actually to specify what to do after a Promise has been **resolved** or **rejected**, so the two are not fundamentally different. The main idea behind Promises is to take callbacks - especially nested callbacks where you want to perform a series of actions, each one after the resolution of the former- and “flatten out” the code required to do this. For example, the below uses JavaScript Promises to load a series of images and perform the desired action after each image load:

1. **function** fetchImage(url){
2. **return** **new** Promise(**function**(resolve, reject){
3. **var** img = **new** Image()
4. img.onload = **function**(){
5. resolve(url)
6. }
7. img.onerror = **function**(){
8. reject(url)
9. }
10. img.src = url
11. })
12. }
14. fetchImage('image1.png').**then**(**function**(url){
15. console.log(url + ' downloaded!')
16. **return** fetchImage('image2.png')
17. }).**then**(**function**(url){
18. console.log(url + ' downloaded!')
19. **return** fetchImage('image3.png')
20. }).**then**(**function**(url){
21. console.log(url + ' downloaded!')
22. **return** fetchImage('image4.png')
23. }).**then**(**function**(url){
24. console.log(url + ' downloaded!')
25. })
27. //Console log:
28. // image1.png downloaded!
29. // image2.png downloaded!
30. // image3.png downloaded!
31. // image4.png downloaded!

Regardless of the number of images I want to load, the level of callback functions used stays at one, which is a lot more maintainable and readable then using callback functions alone. Two excellent guides on JavaScript Promises: