1. 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.

2. 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.

3. What are the key features of Node.js?

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.

4. 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.

5. What is libuv?

libuv is a C library that is used to abstract non-blocking I/O operations to a consistent interface across all supported platforms. It provides mechanisms to handle file system, DNS, network, child processes, pipes, signal handling, polling and streaming.
It also includes a thread pool for offloading work for some things that can't be done asynchronously at the operating system level.

6. Are you familiar with differences between Node.js modules and ES6 modules?

The modules used in Node.js follow a module specification known as the CommonJS specification.
The recent updates to the JavaScript programming language, in the form of ES6, specify changes to the language, adding things like new class syntax and a module system.
This module system is different from Node.js modules. To import ES6 module, we'd use the
ES6 import functionality.

Now ES6 modules are incompatible with Node.js modules. This has to do with the way modules
are loaded differently between the two formats. If you use a compiler like Babel, you can mix
and match module formats.

7. How does Node.js handle child threads?

Node.js, in its essence, is a single thread process. It does not expose child threads and thread management methods to the developer. Technically, Node.js does spawn child threads
for certain tasks such as asynchronous I/O, but these run behind the scenes and do not execute any application JavaScript code, nor block the main event loop.

If threading support is desired in a Node.js application, there are tools available
to enable it, such as the ChildProcess module.

8. How to use Buffer in Node.js?

Buffer is used to process binary data, such as pictures, mp3, database files, etc.
Buffer supports a variety of encoding and decoding, binary string conversion.

9. What is Stream 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.
if we’re piping into a duplex stream, we can chain pipe calls just like we do in Linux:

The pipe method returns the destination stream, which enabled us to do the chaining above.
For streams a (readable), b and c (duplex), and d (writable), we can:

10. 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.

11. What is difference between synchronous and asynchronous method of fs module?

Every method in fs module has synchronous as well as asynchronous form.

1) Asynchronous methods takes a last parameter as completion function callback and first parameter of the callback function is error. It is preferred to use asynchronous method instead of
2) Synchronous method as former never blocks the program execution where the latter one does.

12. What is the preferred method of resolving unhandled exceptions in Node.js?

Unhandled exceptions in Node.js can be caught at the Process level by attaching a handler
for uncaughtException event.

```
process.on('uncaughtException', function(err) {
  console.log('Caught exception: ' + err);
});
```

However, uncaughtException is a very crude mechanism for exception handling and may be removed
from Node.js in the future. An exception that has bubbled all the way up to the Process level
means that your application, and Node.js may be in an undefined state, and the only sensible
approach would be to restart everything.

The preferred way is to add another layer between your application and the Node.js process
which is called the domain.

Domains provide a way to handle multiple different I/O operations as a single group.
So, by having your application, or part of it, running in a separate domain, you can safely handle exceptions at the domain level, before they reach the Process level.

13. What's the Event Loop?

The event loop is what allows Node.js to perform non-blocking I/O operations — despite the fact that JavaScript is single-threaded — by offloading operations to the system kernel whenever possible.
Every I/O requires a callback - once they are done they are pushed onto the event loop for
execution. Since most modern kernels are multi-threaded, they can handle multiple operations
executing in the background. When one of these operations completes, the kernel tells Node.js
so that the appropriate callback may be added to the poll queue to eventually be executed.

14. When should we use Node.js?

Node.js is well suited for applications that have a lot of concurrent connections and
each request only needs very few CPU cycles because the event loop (with all the other
clients) is blocked during execution of a function.
Node.js is best suited for real-time applications:

online games,
collaboration tools,
chat rooms,
API's
or anything where what one user does with the application needs to be seen by other users
immediately, without a page refresh.

15. Can Node.js work without V8?
No. The current node.js binary cannot work without V8. It would have no Javascript engine
and thus no ability to run code which would obviously render it non-functional.
Node.js was not designed to run with any other Javascript engine and, in fact, all the
native code bindings that come with node.js (such as the fs module or the net module) all
rely on the specific V8 interface between C++ and Javascript.

There is an effort by Microsoft to allow the Chakra Javascript engine
(that's the engine in Edge) to be used with node.js. Node.js can actually function to
some extent without V8, through use of the node-chakracore project. There is ongoing work
to reduce the tight coupling between V8 and Node, so that different JavaScript engines
can be used in-place.

16. How would you handle errors for async code in Node.js?
Handling async errors in callback style (error-first approach) is probably the fastest way
to hell (a.k.a the pyramid of doom). It's better to use a reputable promise library or
async-await instead which enables a much more compact and familiar code syntax like
try-catch.

17. Is Node.js entirely based on a single-thread?
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.

18. Is it possible to use Class in Node.js?
With ES6, you are able to make "actual" classes and export them just like anything else
and once imported into another module, then you can treat it as if it were defined
in that file

19. 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

20. What are the Timing features of Node.js?
The Timers module in Node.js contains functions that execute code after a set period of time.

setTimeout/clearTimeout - can be used to schedule code execution after a designated amount of milliseconds
setInterval/clearInterval - can be used to execute a block of code multiple times
setImmediate/clearImmediate - will execute code at the end of the current event loop cycle
process.nextTick - used to schedule a callback function to be invoked in the next iteration of the Event Loop

21. What's a stub? Name a use case.
Stubs are functions/programs that simulate the behaviours of components/modules.
Stubs provide canned answers to function calls made during test cases.
Also, you can assert on with what these stubs were called.
A use-case can be a file read, when you do not want to read an actual file:

22. When to not use Node.js?
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.

23. Can Node.js use other engines than V8?
Yes. Microsoft Chakra is another JavaScript engine which can be used with Node.js.
It's not officially declared yet.

24. Explain what is Reactor Pattern in Node.js?
Reactor Pattern is an idea of non-blocking I/O operations in Node.js.
This pattern provides a handler(in case of Node.js, a callback function) that is associated with each I/O operation. When an I/O request is generated, it is submitted to a demultiplexer.

This demultiplexer is a notification interface that is used to handle concurrency
in non-blocking I/O mode and collects every request in form of an event and queues each event in a queue. Thus, the demultiplexer provides the Event Queue.

At the same time, there is an Event Loop which iterates over the items in the Event Queue.
Every event has a callback function associated with it, and that callback function is invoked when the Event Loop iterates.

25. How V8 compiles JavaScript code?

V8 has two compilers:

A “Full” Compiler that can generate good code for any JavaScript: good but not great JIT code.
The goal of this compiler is to generate code quickly. To achieve its goal,
it doesn’t do any type analysis and doesn’t know anything about types.
Instead, it uses an Inline Caches or “IC” strategy to refine knowledge about types while the program runs. IC is very efficient and brings about 20 times speed improvment.

An Optimizing Compiler that produces great code for most of the JavaScript language.
It comes later and re-compiles hot functions. The optimizing compiler takes types from the Inline Cache and make decisions about how to optimize the code better.
However, some language features are not supported yet like try/catch blocks for instance.
(The workaround for try/catch blocks is to write the “non stable” code in a function and call the function in the try block)

V8 also supports de-optimization: the optimizing compiler makes optimistic assumptions from
the Inline Cache about the different types, de-optimization comes if these assumptions are
invalid. For example, if a hidden class generated was not the one expected, V8 throws away
the optimized code and comes back to the Full Compiler to get types again from the Inline
Cache. This process is slow and should be avoided by trying to not change functions after
they are optimized.

26. How many threads does Node actually create?

4 extra threads are for use by V8. V8 uses these threads to perform various tasks,
such as GC-related background tasks and optimizing compiler tasks.

27. What is V8 Templates?

A template is a blueprint for JavaScript functions and objects.
You can use a template to wrap C++ functions and data structures within JavaScript objects.
V8 has two types of templates: Function Templates and Object Templates.

1) Function Template is the blueprint for a single function.
You create a JavaScript instance of template by calling the template’s GetFunction method
from within the context in which you wish to instantiate the JavaScript function.
You can also associate a C++ callback with a function template which is called when
the JavaScript function instance is invoked.

2) Object Template is used to configure objects created with function template as their constructor. You can associate two types of C++ callbacks with object templates: accessor
callback and interceptor callback. Accessor callback is invoked when a specific object property
is accessed by a script. Interceptor callback is invoked when any object property is accessed
by a script. In a nutshell, you can wrap C++ objects\structures within JavaScript objects.

28. What is the difference between process.nextTick() and setImmediate()?

The difference between process.nextTick() and setImmediate() is that process.nextTick() defers the execution of an action till the next pass around the event loop or it simply calls the callback function once the ongoing execution of the event loop is finished whereas setImmediate() executes a callback on the next cycle of the event loop and it gives back to the event loop for executing any I/O operations.

29. Why Node.js devs tend to lean towards the Module Requiring vs Dependency Injection?

Dependency injection is somewhat the opposite of normal module design.
In normal module design, a module uses require() to load in all the other modules that it needs with the goal of making it simple for the caller to use your module.
The caller can just require() in your module and your module will load all the other things it needs.

With dependency injection, rather than the module loading the things it needs, the caller is required to pass in things (usually objects) that the module needs.
This can make certain types of testing easier and it can make mocking certain things for testing purposes easier.

Modules and dependency injection are orthogonal: if you need dependency injection for testability or extensibility then use it. If not, importing modules is fine.
The great thing about JS is that you can modify just about anything to achieve what you want.
This comes in handy when it comes to testing.

30. Why should you separate Express app and server?

Keeping the API declaration separated from the network related configuration (port, protocol, etc) allows testing the API in-process, without performing network calls, with all the benefits that it brings to the table: fast testing execution and getting coverage metrics of the code. It also allows deploying the same API under flexible and different network conditions. Bonus: better separation of concerns and cleaner code.