Q. What is node.js?

A. Node.js is a javascript runtime build on Chrome's V8 javascript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js has it's own package ecosystem called node package manager or npm, the largest ecosystem of open source libraries in the world.

Q. What is Chrome V8?

A. V8 is google's open source hight-performance javascript engine, written in C++ and used in google chrome, the open source browser form google. It implements ECMAScript as specified in ECMA-262, 3rd edition, and runs on Windows XP or later, Mac OSX 10.5+, and Linux sustems that use IA-32, ARM or MIPS precessors. V8 can run standalone, or can be embedded into any C++ applicaiton.

Q. What is npm?

A. npm stands for node package manager. npm provides following two main functionalities:

- Online repositories for node packages/modules where we can search for different moudles.
- Command line utility to install packages, do version management and dependency management of nade.js packages.

Q. What is an error-first callback?

A. Error-first callbacks are used to pass errors and data. The first arguments is always an error object that the programmer has

to check if something went wrong. Additional arguments are user to pass data.

```
fs.readFile( filePath, function( err, data ) {
    if( err ) {
        //handle the error
    }
    //use the data object
});
```

Q. How can you avoid callback hells?

A. To avoid callback hell we have many options in hand –

1. Modularization – break callbacks into independent functions. (Good)
2. Promises (Better) –

A promise represents a proxy for a value not necessarily known when the promise is created. It allows you to assoicate handlers to an asynchronous action's eventual success value of failure reason. THis lets asynchronous methods return values like synchronous methods: insted of the final value, the asynchronous method returns a promise of having a value at some point in the future.

A promise is in one of these states:

- pending: initial state, not fulfilled or rejected.
- fulfilled: meaning that the operation completed successfully.
- rejected:

mening that the operation failed.

A pending promise can become either fulfilled with a value, or rejected with reson(error). When either of these happens, the associated handlers queued up by promise's then method are called. (If the promise has already been fulfulled or rejected when a corresponding handle is attached, the handler will be called, so there is no race condition between an asyncronous operation completing and its handlers being attached.)

As the promise.prototypeli.then() and promise.prototype.catch() mehtods return promises, they can be chained - an operation called compositon.

Library that uses promise - jQuery, Bluebird, Q, When etc.

3. Generators (Best)

Genrators are functions which can be exited and later re-entered. Their context (variable bingdings) will be saved across re-entrances.

Calling generator function does not execute its body immediately; and iterator object for the function is returned insted. When the iterator's next() method is called, the generator functions's body is executed until the first yield expression, which specifies the value to be returned from the iterator or, with yield*, delegates to another generator function. The next() method returns an object whith a value property containing the yielded value and a done property which

indicates whether the generator has yielded its last value.

It's not supported by most of the browsers yet, but we can use google's traceur to convert it to es6 hermony and then brower will understand it and run it.

We will have to use type=module while calling the script from html file so that google traceur can take it and convert it for the browser. <script src='generator.js' type='module></script>

And in our code we will use funciton* inside promise.corutine -

```
Promise.corutine( function* () {
                                        var
value = yield function(){....};
                                        //do
something with the value
                                        var
value2 = yield function(){....};
                                        //do
something with the value2
                                    })
().catch(function(errs) {
                                        //
handle errors
                                    });
```

In this example on every yield program will wait untill the function attached to the yield returns a value.

Q. How can we listen on port 80 with node?

A. It's not a good idea to listen to port 80 (in unix-like systems) with node applicaiton but we can do it and we will need superuser rights. We can also run our application on any port above 1024 and put a reverse procy like nginx in front of it.

Q. What's the event loop?

A. According to 'Trevor Norris' "It's a magical place filled with unicorns and rainbows.

Node.js runs using a single thread, at least from a Node.js developer's point of view. Under the hood Node.js uses many threads through libuv.

Every I/O requires a callback-once they are done they are pushed onoto the event loop for executon. Event loop is something when user-1 sends request your server takes the request and starts working on it. But thanks to node.js it doesn't hold the port...it keeps listening. When user-2 sends a request, your server takes the request and start working on it as it did for user-1. When your server is done with user-1's request, it sends a responds with necessary information to user-1. And it will do the same thing for user-2. But if the server finish working with user-2's request before finishing user-1's request, it will send user-2 a response and then to user-1.

Node is a single threaded application but it support concurrency via concept of event and callbacks. As every API of node are asynchronous and being a single threaded, it uses async function call 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.

Q. What is event emitter?

A. EventEmitter class lies in events module. When an evenemitter instance faces any error, it emits and 'error' event. When new listener is added, 'newListener' event is fired and when a listener is removed, 'removeListener even 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.

Q. What is libuv?

A. 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.

Q. How node.js handle child threads?

A. Node.js in it's essence, is a single thread process. It does not expose child threads and thread management methods to the develoer. 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, not 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. We can do it using this:

```
const spawn = require('child_process').spawn;
```

Q. What does event-driven programming mean?

A. 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 applicaiton architecture technique divided into two sections

- Event selection
- Event Handling

Q. Where we can use node.js?

A. Node.js can be used for the following purposes:

- Web applicaiotns (especially real-time web apps)
- Network applications
- Distributed systems
- General purpose applicaitons

Q. Pros and cons of node.js

A. Pros:

1. Asychronous event driven I/O helps concurrent request handling.
2. Uses javascript which is easy to learn.
3. Share the same piece of code with both server and client side.

4. npm has already become huge and still growing.

5. Active and vibrant community, with lots of code shared via guthub etc.

6. Can stream big files.

Cons:

1. Node.js is not suited for CPU-intensive tasks. It is suited for I/O stuff only (like web servers).

2. Dealing with relational databse is a pain if you are using node.

3. Everytime using a callback end up with tons of nasted callbacks which also called callback hell.

4. Without diving in depth of javascript if someone starts node, he may face conceptual problems.

Q. What is ‘Callback’ in node.js?

A. 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, callback function is used. Callback function allows the server to deal with pending request first and call a function when it is finished.

Q. What is purpose of Buffer class in

node?

A. Buffer class is a global class and can be accessed in applicaiton without importing buffer module. Prior to the introduction of TypedArry in ES2015, the javascript language had no mechanism for reading or manipulatin streams of binary data. The Buffer class was introduced as part of the Node.js API to make it posssible to interact with octet streams in the context of things like TCP streams and file system operations.

Now that TypedArray has been added in ES6, the Buffer class implements the Uint8Array API in a manner that in more optimized and suitable for Node.js' use cases.

Instances of the Buffer class are similar to arrays of integers but correspond to fixed-sized, raw memory allocations outside the V* heap. The size of the Buffer is established when it is created and cannot be resized. As it is global so there is no need to use require('buffer') to use it.

Q. What is piping in node?

A. 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.

Q. Which module is used for file based operations?

A. fs moudle is used for file base operation

```
var fs = require('fs');
```

Q. What module is used for web based operations?

A. http module is used for web based operations−

```
var http = require('http');
```

Q. Is it true that fs module provides both synchronous and asynchronous methods?

A. True. Every method in fs module have synchronous as well as asynchronous form. It is preferred to use asynchronous method instead of synchronous method as it never blocks the program execution.

Q. What are streams?

A. Streams are objects that let you read data form a source or write data to a estination in continous fasion.

Q. How many types of streams are present in node?

A. In node there are four types of streams.

Readable − Stream which is used for read operations.

Writable − Stream which is used for white operations.

Duplex − Stream which can be used for both rand and write operations.

Transform − A type of duplex stream where the output is computed based on input.

Q. Nane sine events fired by streams.

A. Each 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 available to read.

end – this event is fired when there is no more data to read.

error – fired when there is any error receiving or writing data.

finish – fired when all data had been flushed to underlying system.

Q. What is chaining in Node?

A. Chaining is a mechanism to connect output of one stream to another stream and creat a chain of multiple stream operations. It is mormally used with piping operations.

Q. How will you open a file using node?

A. We can use fs.open(path, flags[, mode], callback) to open a file in asynchronous mode.

path – a string having file name including path.

flags – tells the behavior of the file to be opened. All possible values have been mentioned below.

mode – sets the file mode(permission and sticky bits), but only if the file was created. It defaults to 0666, readable and writeable.

callback – this is the callback function which gets two arguments err and fd.

Q. How will you read a file using node?

A. fs.read(fd, buffer, offset, length, position, callback)

fd – file descriptor returned by file fs.open() methid.

buffer – this is the buffer that the data will be written to.

offset – an integer specifying the number of bytes to read.

position – and integer specifying where to begain reading from in the file. If position is null, data will be read from the current file position.

callback – callback function which takes three arguments, err, bytesRead, buffer.

Q. How will you write a file using node?

A. fs.writeFile(path, data[, options], callback)

path – string having filename including path

data – string or buffer to be written in the file.

options – an object which will hold {encoding, mode, flag}. By default encoding is utf8, mode is octal value 0666 and flag is 'w'

callback – callback function which takes a single parameter err.

Q. How to close a file in node?
A. fs.close(fd, callback)
fd – file descriptor returned by file fs.open() methid.
callback – callback function that takes no argument.

Q. How to delete a file using node?
A. fs.unlink(path, callback)

Q. How to create a directory in node?
A. fs.mkdir(path[, mode], callback)

Q. How to delete directory in node?
A. fs.rmdir(path, callback)

Q. How to read directory in node?
A. fs.readdir(path, callback)

Q. What is purposr of __filename variable?
A. The __foelname represents the filename of the code being executed. This is the resolved absolte path of this code file. For a main program this is not necessarily the same filename used in the command line. the value inside a module is the path to the mdoule file.

Q. What is purposr of __dirname?
A. The __dirname represents the name of the directory that the currently executing script resides in.

Q. What is purpose of setTimeout function?

A. The setTImeout(cb, ms) global function is used to run callback cb after at least ms milliseconds. The actual delay depends on external factors like OS timer granularity and system load. A timer cannot span for more than 24.8 days.

This function returns an opaque value that represents the timer which can be used to clear the timer.

Q. What is purpose of setInterval function?

A. The setInterval(cd, ms) is same a ssetTimeout where instead of doing it once it does repeatedly after ms milliseconds.

Q. What is purpose of process object?

A. Precess object is used to get information on current process. Provies multiple events related to process activities.

Q. What is the preferred method of resolving unhandled exceptions in node?

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

```
precess.on('uncaughtException', function(err) {

console.log('Caught exception: ' + err);
                        });
```

However, uncaughtException is a

very crude mechanism for exception handling and may be removed form node.js in future. AN ecveption that had bubbled all the way up to the Process level means that you application, and node may be in an undefined state, and the only sensible approch would be to restart everything.

The preferred way is to add another layer between your application and the node 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 applicaiotn, or part of it, running in a separate domain, you can safely handle exceptions at the domain level, before they reach the Process level. Although currently the domain is marked as depricated by node.js .

Q. How does node support multi-processor platforms, and does it fully utilize all processor resources?

A. Since node.js is by default a single thread applicaiton, it will run on a single processor core and will not take full advantage of multiple core resources. However , node.js provides support for deployment on multiple-core systems, to take greater advantage of the hardware. The Cluster module is one of the core node.js module and it allows running multiple node.js worker processes that will share same port.

Q. What tools can be used to assure consistent style?

A. We have plenty of options to do so:

JSLint, JSHint, ESLint, JSCS

These tools are really helpful when developing code in teams, to enforce a given style guid and to catch common errors using static analysis.

Q. Difference between operational and programm errors

A. Operation errors are not bugs, but problems with the system, like request timeout or hardware failure.

On the other hand program errors are actual bugs.

Q. Why npm shrinkwrap is useful?

A. This command looks down the versions of a package's dependencies so that you can control exactly which versions of each dependency will be used when your package is installed.

It's usefull when you are deploying your node.js applications – with tit you can be sure which versions of your dependencies are going to be deployed.

Q. What's a stub? Name a use case.

A. Stubs are functions/programs that simulate the behavours 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:

var fs = require('fs');

```
                var readFileStub =
sinon.stub(fs, 'readFile', function(path, cb) {
                        reutrn cb(null,
'filecontent');
                });

expect(readFileStub).to.be.called;
                readFileStub.restore();
```

Q. What's a test pyramid? How can you implement it when talking about HTTP APIs?

A. A test puramid describes that when writings test cases there should be a lot more low-level unit test than high level end-to-end tests.

When talking about HTTP APIs, it may come down to this:

- a lot of low-level unit tests for your models
- less integration tests, where your test how your models interact with each other
- a lot less acceptance thests, where you test the actual HTTP endpoints

Q. What are the benefits of using node.js?

A. Following are main benefits of using node.js

Asynchronous and Even Driven: All APIs of node.js library are asynchronous that is non-blocking. It essentially means a node.js base 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 build 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 sacalable as opposed to tradtional servers which create limited threads to handle requests. Node.js uses single threaded program and same progam can services much larger number of requests than traditional server like Apache HTTP Server.

No Buffering – Node.js applicaitons never buffer any data. These applications simply output the data in chunks.

Q. What is REPL in context of Node?

A. REPL stand 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 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 untill user press ctrl+c twice.

Q. What is the difference of using and not using var in REPL?

A. We can use var in REPL to store result and use it later. But if we just want the result to be printed and not to be stored then there is no point to use var.

Q. What is the use of Underscore(_) variable in REPL?

A. Underscore(_) is user to get the last result printed in console.

Q. What is global installation of dependencies?

A. It means installing modules/ dependencies globally where the modules are saved in user's root directory. Such modules can be used in CLI (Command line interface) functions of any node app but can not be imported by require() in app directly. TO install a node module globally we need to use -g flag.

npm install -g module_name

Q. What is local installation of dependencies?

A. If we do not use -g flag while installing npm moudle then by default npm installs it localy. A local module can be required from the app localy. But we can not use

it in CLI as we can for global modules. To install a module localy –

npm install module_name

Q. How to chack installed modules?
A. For local moules: npm ls
for glabal moules: npm ls –g

Q. What is package.json?
A. package.json is a file present in the root directory of node app which contains app properties and dependency information.

Q. Name some apptributes of package.json
A. name – name of the app
version – version of the app
description – description of the app
homepage – homepage of the app
author – author name of the app
contributors – names of contributors
dependencies – list of dependencies or npm modules used in the app
repository – url of repository where app is stored
main – entry point of the app
keywords – keywords related to the app

Q. How to uninstall npm modules?
A. For local mdoules: npm uninstall moudle_name

for global modules: npm uninstall -g moudle_name

Q. How to update npm moudles?
A. For local modules: npm update
for glabal modules: npm update -g

Q. {

```
console.time('loop');
                                for(var i = 0; i< 1000000; i += 1) {
                                        //do nothing
                                }

console.timeEnd('loop');
```

}
Time required to run a for loop in google chrome in considerably more then node.js
A. Within a web browser such as chrome, declaring the variable i outside of any function's scope makes it global and therefor binds it as a property of the window object. As a result, running this code in a web browser requires repeatedly resolving the proverty i within the heavily populated window namespace in each iteration of the for loop.
In node, however, declaring any variable outside of any function's scope binds it only to the module's own scope which therefor makes it faster to resolve.