## What is Node.js?

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009 and its latest version is v0.10.36. The definition of Node.js as supplied by its [official documentation](https://nodejs.org/) is as follows −

Node.js is a platform built on [Chrome's JavaScript runtime](https://code.google.com/p/v8/) for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

## Features of Node.js

Following are some of the important features that make Node.js the first choice of software architects.

* Asynchronous and Event Driven − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a 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 the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
* No Buffering − Node.js applications never buffer any data. These applications simply output the data in chunks.
* License − Node.js is releas
* ed under the [MIT license](https://raw.githubusercontent.com/joyent/node/v0.12.0/LICENSE).

## Who Uses Node.js?

Following is the link on github wiki containing an exhaustive list of projects, application and companies which are using Node.js. This list includes eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wikipins, Yahoo!, and Yammer to name a few.

* [Projects, Applications, and Companies Using Node](https://github.com/joyent/node/wiki/projects,-applications,-and-companies-using-node)

## Concepts

The following diagram depicts some important parts of Node.js which we will discuss in detail in the subsequent chapters.



Node.js uses some of the following concepts,

1. *Debugger*  
   Statement inserting debugger; in the code of java script it will help to enable break point.
2. *Modules*  
   Node.js supports simple module loading system.
3. *Console*  
   Console module provides debugging feature that is similar in Java script console by browser.
4. *Streaming*  
   By various objects in node js it is abstract interface implemented.
5. *Cluster*  
   it allows us to create child process easily that shares server port.
6. *DNS*  
   DNS module contains functions.
7. *Add-ons*  
   It is a dynamically linked shared object.
8. *Domain*  
   It provides a way to handle multiple different IO operations as a single group.
9. *Buffer*  
   It is similar to array of integers but corresponds to fixed-sized.
10. *Global*  
    It is available for all modules.
11. *Net*  
    It provides asynchronous network wrapper.
12. *Call backs*  
    It is called when given task will be completed.
13. *Error handling*  
    It supports various types of categories error.
14. *Crypto  
    It provides cryptographic functionality that includes a set of wrappers for Open SSL’s hash.*

**Where to Use Node.js?**

Following are the areas where Node.js is proving itself as a perfect technology partner.

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

## Where Not to Use Node.js?

It is not advisable to use Node.js for CPU intensive applications.

# Node.js - Environment Setup

## Try it Option Online

You really do not need to set up your own environment to start learning Node.js. Reason is very simple, we already have set up Node.js environment online, so that you can execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

Try following example using Try it option available at the top right corner of the below sample code box:

/\* Hello World! program in Node.js \*/

console.log("Hello World!");

## Local Environment Setup

If you are still willing to set up your environment for Node.js, you need the following two software’s available on your computer, (a) Text Editor and (b) The Node.js binary installable.

## Text Editor

This will be used to type your program. Examples of few editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi.

Name and version of text editor can vary on different operating systems. For example, Notepad will be used on Windows, and vim or vi can be used on windows as well as Linux or UNIX.

The files you create with your editor are called source files and contain program source code. The source files for Node.js programs are typically named with the extension ".js".

Before starting your programming, make sure you have one text editor in place and you have enough experience to write a computer program, save it in a file, and finally execute it.

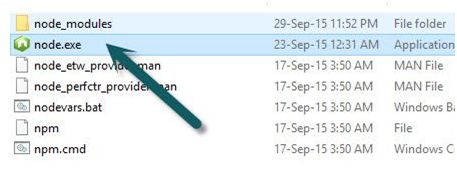
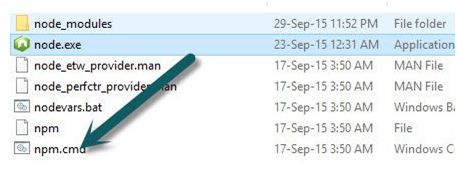
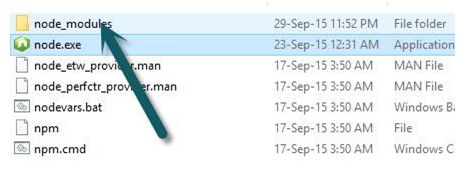
## The Node.js Runtime

The source code written in source file is simply JavaScript. The Node.js interpreter will be used to interpret and execute your JavaScript code.

Node.js distribution comes as a binary installable for SunOS , Linux, Mac OS X, and Windows operating systems with the 32-bit (386) and 64-bit (amd64) x86 processor architectures.

Following section guides you on how to install Node.js binary distribution on various OS.

**Question 1: Where can I install Node.js?**  
We can download Node.js software from the [website](https://nodejs.org/en/).  
  
  
  
After successful installation look at the installation location, and then you will see the three important parts.

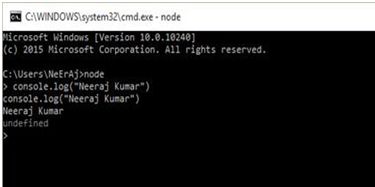
1. *Node*  
     
   This file starts a Node.js JavaScript engine. You can execute any JavaScript file with this node file.  
     
   
2. *npm*  
     
   npm is used for managing the Node.js Package.  
     
   
3. *node\_module*  
     
   This folder contains the installation Node.js Package. This package acts as a Library.  
     
   

**Verifying Node.js Installation**  
  
After successful installation of Node.js, you can check if it’s working.  
Open Command Prompt and write the following command.

1. node

Next, the Node.js prompt will appear. After the prompt is visible, write the following command.

1. console.log(“NeErAjKuMaR”);

Press Enter  
  
  
  
  
**Question 2: What is Node.js?**Node.js is a server-side JavaScript platform which is built on Google Chrome’s JavaScript V8 engine. It is an open source and cross platform application to develop server side and networking applications. Anyone can develop the Node.js application by writing code in JavaScript and it can be run on Microsoft Windows, Linux, or OS X.  
  
Node.js is not a new language and also it is not just a framework built on JavaScript. It is built on Chrome's JavaScript Runtime, so the code is written and executed very similarly to the browser.   
 **Features of Node.js**  
Following are some important features of Node.js.

* *Fast in Code execution*  
  It is very fast in code execution.
* *Highly scalable*  
  Node.js uses a single thread model for event looping. Events respond to the server without blocking other operations. This makes Node.js highly scalable. Traditional servers create limited threads to handle requests and Node.js creates a single thread that provides service to much larger numbers of requests.
* *Event Driven and Asynchronous*  
  All API of Node.js are asynchronous. It means that the server is moving the next API call without waiting for returning data of previous requests.
* *No Buffering*  
  Node.js never buffers any data.

Many of us are confused about Node.js. It is not a web server like Apache. Node.js provides a new way to execute our code. It is JavaScript runtime. Node.js provides the facility to create an HTTP server and we can host our application on the same.  
  
**Question 3: What are the key features of Node.js?**  
Here are the following key features of Node.js:

* Real time Data intensive.  
  *Example* -Multiplayer Games, Stock Trading, Chat App etc.
* Highly scalable servers for Web Applications.   
  *Example* - E-Commerce, Social Media, REST API etc.
* Builds fast and scalable network Applications.  
  *Example* - Proxy Server, Backend web services, HTTP Web Server etc.

**Node.js main features**

* Node.js version 6 focuses on performance improvements, increased reliability and better security for its 3.5 million users.
* Event driven and Asynchronous.
* Node.js is emerging as a universal platform used for Web Applications, IoT, mobile, Enterprise Application development and micro service architectures.
* Full stack JavaScript developers can use it for front end, back end, mobile and IoT projects.
* Node.js v6 comes equipped with v8 JavaScript engine 5.0, which has improved ECMAScript 2015 (ES6) support. Ninety-three percent of ES6 features are also now supported in the Node.js v6 release.
* Very Fast.
* Single Threaded but highly Scalable.

**Question 4: What are the advantages of Node.js?**Node.js is an open source, cross-platform runtime environment for server-side and networking applications. Node.js applications are written in JavaScript and can be run within the Node.jsruntime on OS X, Microsoft Windows, Linux, FreeBSD, NonStop and IBM.  
  
**Advantages**

1. Web development is done in a dynamic language (JavaScript) on a VM that is incredibly fast (V8). It is much faster than Ruby, Python, or Perl.
2. Ability to handle thousands of concurrent connections with minimal overhead on a single process.
3. JavaScript is perfect for event loops with first class function objects and closures. People already know how to use it this way having used it in the browser to respond to user-initiated events.
4. Many people already know JavaScript, even people who do not claim to be programmers. It is arguably the most popular programming language.
5. Using JavaScript on a web server as well as the browser reduces the impedance mismatch between the two programming environments that can communicate data structures via JSON that work the same on both sides of the equation. Duplicate form validation code can be shared between server and client and so on.

**Question 5: Explain REPL in Node.js.**The REPL stands for **Read Eval Print Loop**, which is a simple program that accepts the commands, evaluates the commands, and prints their results. It represents a computer environment like Unix/Linux shell or a window console in which we can enter the command and the system responds with output. REPL performs the following tasks:

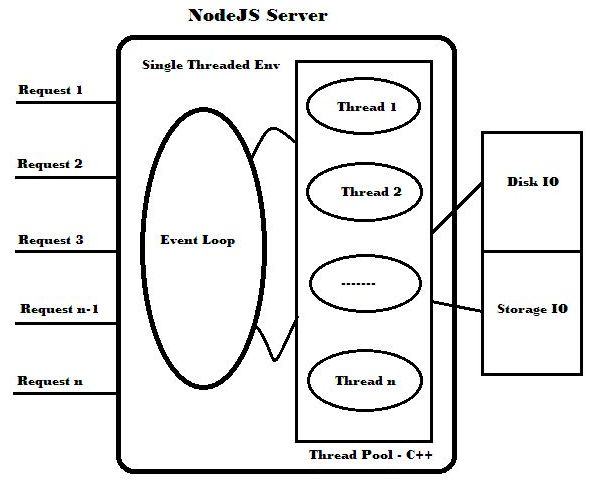
* *READ*  
  Read the input from user, parse it into JavaScript data structure and store it in memory.
* *EVAL*  
  Execute the data structure.
* *RINT*  
  Print the result
* *LOOP*  
  Loop the command until user presses Ctrl+C two times.

**Question 6: What is Closure?**

Closure is a function defined within another scope that has access to all the variables within the outer scope. A Closure allows us a free environment for the outer function to access the inner functions and inner variables without any scope restrictions.  
  
Closure is the local variable for a function, kept alive after the function has returned; or it's a stack-frame that is not deallocated when the function returns. A Closure is an inner function that has access to the outer function variable's scope chain. The Closure has 3 scope chains; the first one is to access to its own scope (variables defined within the curly brackets) and the second one is to access the outer function's variables and the third one is to access the global variables.  
  
Points to remember about Closures,

* Whenever you use a function inside another function, a Closure is used.
* Closure in JavaScript is like keeping a copy of all the local variables.
* A Closure is created just on entry to a function and the local variables are added to that Closure.
* A new set of local variables are kept every time a function with a Closure is called.
* Closure is the term for both the functions along with the variables that are captured.

**Question 7: What is NPM? What is the need of NPM in Node.js?**  
NPM stands for Node.js Package Management. It comes with Node.js platform and allows us to install various packages for Node.js. This package manages and supports various commands to install and remove the modules. Here one important note is we require either **package.json** file or the **node\_modules** folder to install modules locally.  
One of the best things about NPM is that it locally stores all dependencies. For example, if module X uses module A version 1.0, and module Y uses module A version 1.5, then both X and Y will have their own local copies of module A.  
  
**Need of NPM package**  
  
while I was working on a real node.js application I encountered many instances where I needed to write a few libraries. Recently, I thought these libraries could be useful for others. Afterward, I discovered the NPM package.  
  
**Question 8: Explain event loop architecture of NodeJS.**When a request arrives, it is queued up into the Event Loop with a JavaScript Closure that includes the event (request, response) and corresponding callback.

1. The event is then given to the one worker thread if the job seems to take a long time to complete that is from a C++ Thread Pool, handled by the libuv library.
2. Once the job is done, the corresponding callback is fired to return back the response to the Main thread.
3. The Event Loop returns the response to the client.  
   

For example:

1. var fs = require('fs');
2. fs.readFile('avator.png', function(avator) {
3. console.log(‘image **is** loaded…’);
4. });
5. fs.writeFile('log.txt', 'Done', function() {
6. console.log(‘Done !..’);
7. });

The execution flow goes as follows:

1. Our code tells the node to do the two tasks, read() and write(), and then takes a rest.
2. The read() and write() operations are then queued into the Event Loop and distributed to the worker thread.
3. Once the worker threads are done with the job, it will fire the callbacks to return the response to the Event Loop.
4. Then the Event Loop returns the response to the client.

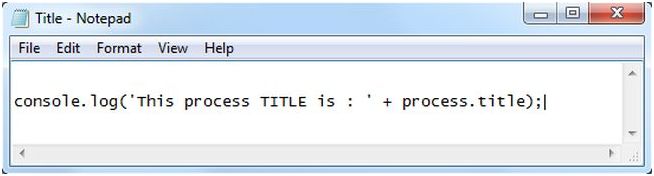
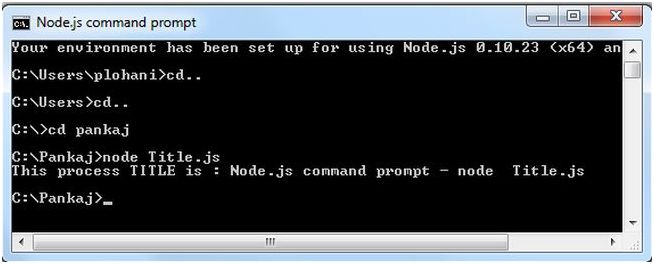
The callbacks of read() and write() will be executed in the order of which one completes first. But note that only one of the callbacks will be executed at a time so that the Node environment ensures that there won't be a deadlock or race condition. So it makes sure that the NodeJS provides **Non-blocking IO**.  
  
We could consider the Post Office environment for the NodeJS.  
  
When we post (request) something, the Post Master (Node) asks the Postmen to deliver (get the job done) the post to the corresponding address.  
  
Once the postmen deliver the posts, they report, one-by-one, to the Post Master that it's completed.  
  
The Post Master may even assign some work to the postman at the time of reporting that he is free.  
  
**Question 9: Explain Node Package Manager.**The Node Package Manager is a Command line utility. It allows you to find, install, remove, publish and do a lot of other things related to Node Packaged Modules. The Node package Manager provides a link between the Node Package Registry and development environment. Node Package Manager lists some of the Command line options.

|  |  |  |
| --- | --- | --- |
| **Option** | **Description** | **Example** |
| **Search** | Finds Module packages in the repository | npm search express |
| **Install** | Installs a package using a package.json | npm install |
| **Install -g** | Install a package in a globally accessible location | npm installexpress –g |
| **remove** | Removes a module | npm remove express |
| **pack** | Packages the module defined by the package.json file into a .tgz file | npmpack |
| **view** | Displays module details | npm view express |
| **publish** | Publish the module defined by a package.json file to the registry | Npm publish |
| **unpublish** | Unpublished a module you have published | npmunpublishMyModule |
| **owner** | Allows you to add, remove, and list owners of a package in the repository | npm add neerajMyModule npmrmneerajMyModule npm Is MyModule |

**Question 10: Explain Process Object in Node.js.**  
The process object is the global object in Node. It can be accessed from anywhere; it is an instance of Event Emitter. Each Node.js has a set of built-in functionalities, accessible through the global process object. The process object provides the standard input/output (stdio) streams stdin, stdout and stderr (as in C/C++) as in the following:

* ***stdin***  
  A readable stream for reading input from the user.
* ***stdout***  
  A writable stream, either synchronously or asynchronously.
* ***stderr***  
  A blocking synchronous writable stream intended for error messages.

The stdout or non-blocking functions are: console.log, console.info, util.puts, util.print and Stderr. The blocking functions are: console.warn, console.error, util.debug and process.stdin (a readable stream for getting user input).  
 **Properties of Process Object**

* *process.title*  
    
  By default a process title is NODE but you can change it.  
    
    
    
  

**Question 12: What is Cluster Server Object in Node.JS?**A single instance of Node runs in a single thread. The cluster module allows you to create child processes that all share server ports. When you call server.listen() in a worker, it serializes the argument and passes the request to the master process. If the master process already has a listening server matching the worker's requirement, then it passes the handle to the server.  
  
If it does not already have a listening server matching that requirement then it will create one and pass the handle to the worker.  
  
**Cluster Events**

* fork
* online
* listening
* disconnect

***Fork****:* the fork event is fired when the master attempts to fork a new child; it receives a worker object.  
  
***Online****:* the online event is fired when the master receives a notification that the child is fully bound; it receives a worker object.  
  
***Listening****:* when the worker performs an action that requires a listen() call such as HTTP server. The event emits two arguments, the first one is a worker object, and the address object contains the address, port and address type value of the connection.  
  
***Disconnect****:* called whenever a child is disconnected from the cluster; that can happen either through a process or exit event.

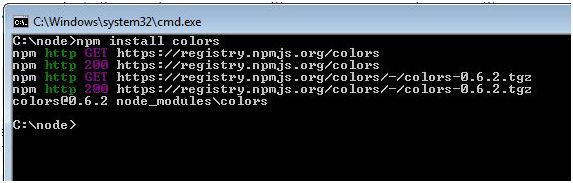
**Question 13: Why do we use Node.js and how does Node.js work?**Node.js is specially designed for building fast, efficient, scalable network applications and uses an event-driven, non-blocking I/O model to maximize efficiency. The callback function is what Node calls a listener function and it is called by the server whenever a new request comes in.  
  
**How it works**  
  
The following describes how Node.js works,

* You use your web browser to make a request for "about.html" on a Node.js web server.
* The Node server accepts your request and calls a function to retrieve that file from disk.
* While the Node server is waiting for the file to be retrieved, it services the next web request.
* When the file is retrieved, there is a callback function inserted into the Node server's queue.
* The Node server executes that function that in this case would render the HTML page and send it back to your browser.

**Platforms for Node.js**

* Runs on OSX, Linux, and Windows.
* Clickable installer for Windows and MacOSX.

**Question 14: Explain Process.nextTick() function in Node.js.**  
The Process.nextTick() function typically runs before any other I/O events fire. As we know, every node application runs on a single thread, in other words only one task or event is processed by Node's event loop. In an event loop, a queue of callbacks are processed by Node on every Tick of the event loop. On the next loop around the event loop calls these callbacks.  
  
The Process.nextTick() function defers the function until a completely new stack. You can call as many functions as you want to in the current stack. When the event loop is looking for a new event to execute the nextTick function will be in the queue and will execute an entire new stack. The Process.nextTick() function defers the execution of an action until the next pass around the event loop.

**Question 15: Explain Colors Module in Node.js.**  
The color module uses the popular color.js. Since the color module is not a built-in module of node.js, we need to install it using the node package manager.  
  
**npm install colors**  
Once the command has executed successfully, you will find the following screen. In my case it's showing that the colors module was successfully installed in the application and is ready for use.  
  
  
  
The beauty of the colors module is that we can beautify the output with various colors.

**Color**  
  
Let's implement a few small examples to understand how the colors module works with node.js. Have a look at the following example.

1. var http = require('http');
2. var colors = require('colors');
3. var server = http.createServer(function (req,res) {
4. console.log('This String Will Display RED'.red);
5. });
6. server.listen(9090);

In the second line we are loading the colors module into our application. After loading the colors module, we can use it in our application. Have a look at the body of the createServer() function. The module will provide freedom to use a color associated with the (.) property of the object. Like, “*sring*”.colorname  
  
In this example we are using the red color by giving (.) after the object. Here is the sample output.  
  
  
  
Not only red, we can provide many colors in the same way. Here is the use of the colors module.  
**Usage**  
This package extends the global String prototype with additional getters that apply terminal colors to your texts. The available styles are,

* ***Emphasis***  
  bold, italic, underline, inverse
* ***Colors***  
  yellow, cyan, white, magenta, green, red, grey, blue
* ***Sequencers***  
  rainbow, zebra, random

So, let's try another example. Here is some sample code.

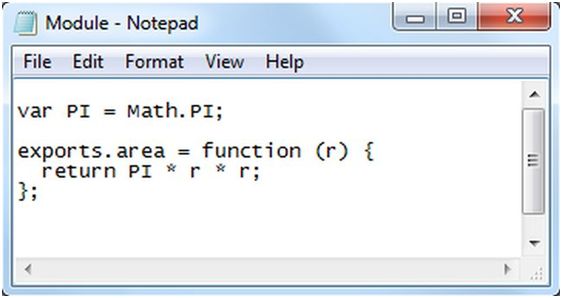
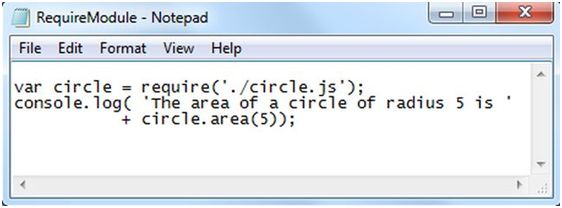
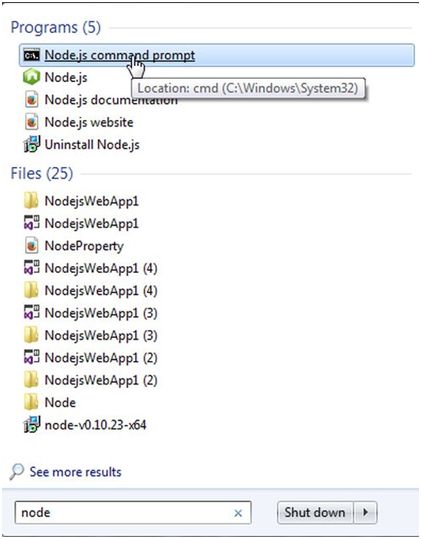
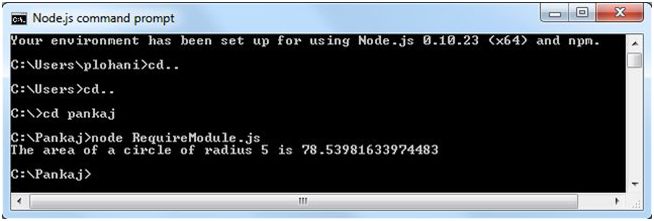
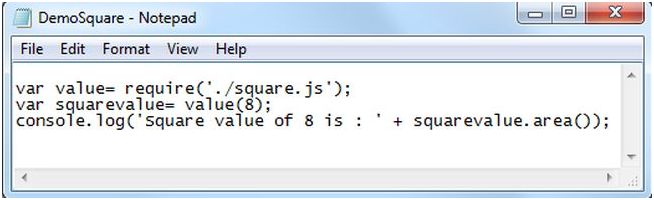
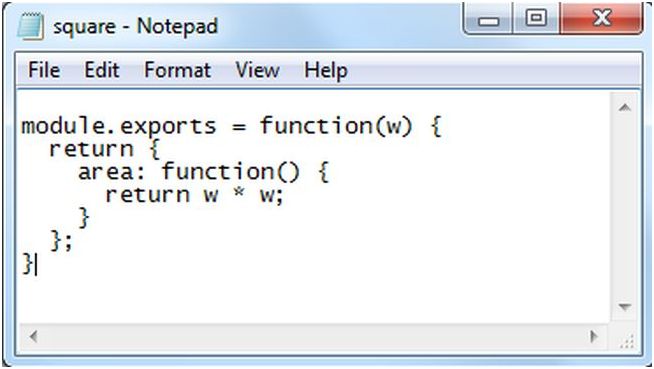
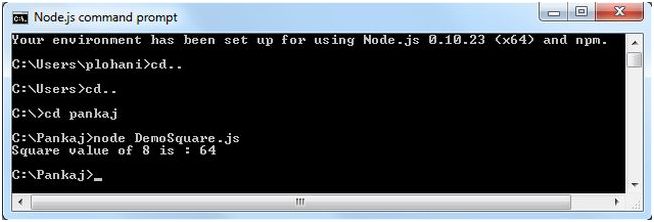
1. var http = require('http');
2. var colors = require('colors');
3. var server = http.createServer(function (req,res) {
4. console.log('hello'.green); // outputs green text
5. console.log('i like cake and pies'.underline.red) // outputs red underlined text
6. console.log('inverse the color'.inverse); // inverses the color
7. console.log('OMG Rainbows!'.rainbow); //
8. });
9. server.listen(9090);

The output of this example is:  
  
  
  
**Question 16: What Is NodeJS Module?**  
A module encapsulates related code into a single unit of code. When creating a module, this can be interpreted as moving all related functions into a file. Node modules run in their own scope so that they do not conflict with other modules. Node relatedly provides global access to help facilitate module interoperability. The primary 2 items that we are concerned with here are required and exports. You require other modules that you wish to use in your code and your module exports anything that should be exposed publicly. Let's see an example: save the below code as test.js,

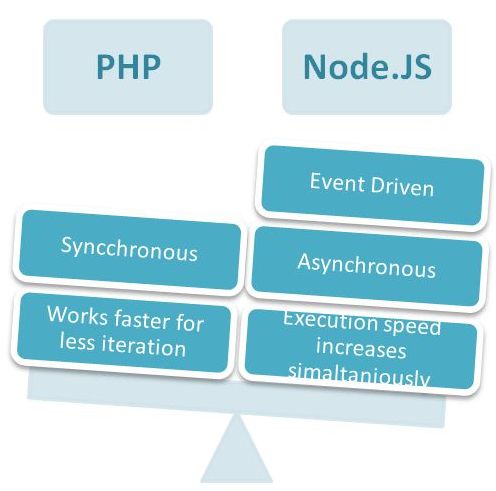
1. exports.name = function() {
2. console.log('My name is Shekhar Patidar');
3. };

**Question 17: What is Cluster Worker Object in Node.js?**A Worker Object is a script that runs in a background process and does not block the other code processes attached to the page. A Worker Object contains all the public information and methods about a worker. The master can use "cluster.workers." If a worker throws an exception and does not handle the exception itself, the exception will create an event that you can listen for as an onerror event.  
  
**Worker.id**  
A unique id is provided to each worker. This unique id resides in the "id". While a worker is alive, on the basis of the unique id we can get the indexes from the cluster.workers.  
  
**Worker.process**  
Workers are created using "child\_process.fork()", the returned object from this function is stored as ".process". Child nodes are entire new instances of V8. Allow at least 30ms startup and 10mb for each new node. A ChildProcess class is not intended to be used directly; use the "spawn()" or "fork()"methods to create child process instances.  
  
**Methods of Worker Class**

* *Worker*  
  send(message,[sendHandle])
* *Worker.kill()*  
  This function kills the workers.
* *Worker*  
  Disconnect()

**Question 18: What are the Modules in Node.js?**Modules are part of the program. Programs are composed of one or more independently developed modules that are not combined until the programs are linked. The structure of the module is identical to the structure of the program. Node has a simple module loading system. In Node there is a one-to-one correspondence of files and modules. As I have shown you in the example below, Module.js loads the module RequireModule.js in the same directory.  
  
Write the following code in Notepad and save the file with the .**js** extension.  
  
  
  
Now define the module RequireModule.js that is loaded into Module.js,  
  
  
  
Now open the Node.js command prompt and execute the following code,  
  
  
  
Now write the following command in the Command Prompt,  
  
  
  
In the module.js module we have defined the function area(). To add functions and objects to the root of your module you can use the "exports" object. Variables are in modules as private as though the module was wrapped in a function. In the example above the variable PI is private to requiremoduel.js. If you want to export an object in one assignment then assign it to *"****module.exports***" instead of "*exports*".  
  
For example,  
  
Create a file and save this file with a .js extension as in the following figure,  
  
   
  
Now create a module in a square.js file as in the following figure,  
  
  
  
Now in a node.js command prompt execute the module as in the following figure  
  
  
  
In the example given above you see the bar.js file uses the square module and the square module is defined in Square.js.  
  
**Core Module**  
The core modules are defined in node's "lib" folder.

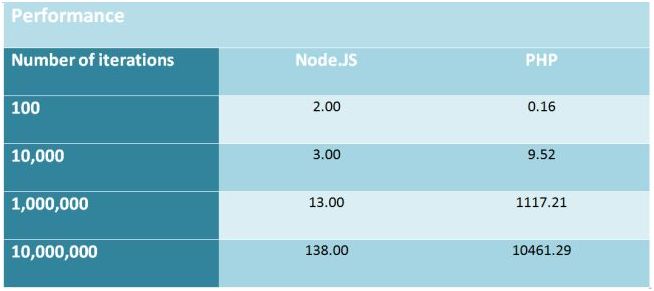
* Node has several modules compiled into the binary.
* Core modules are loaded if their identifier is passed to require().

**Question 19: Difference between Node.js and PHP.**Node.JS is an environment where you need to do everything yourself. It may sound crazy to you. The thing is there is no default HTTP server or any other server for the several required purposes.  
  
  
  
In the node.js environment one script handles all the communication with the respective clients. This considerably reduces the number of resources used by an application.  
  
This fact can be overwhelming for a beginner.  
  
Here I am comparing both with an example:  
  
**Node.JS | Code Snippet**

1. vari, l, m, n, maxval;
2. maxval = 10000000;
3. var v = Date.now();
5. **for**(i = 0; i<maxval; i++)
7. {
8. l = 365 + 1024 + i;
9. m = 365 \* 1024 + i;
10. n = 365 / 5 + i;
11. }
13. console.log(Date.now() - v);

**PHP | Code Snippet**

1. $l = **null**;
2. $m = **null**;
3. $n = **null**;
4. $i = **null**;
5. $maxval = 10000000;
7. $v = microtime(**true**);
9. **for**(i = 0; i<maxval; i++)
11. {
12. $l = 365 + 1024 + $i;
13. $m = 365 \* 1024 + $i;
14. $n = 365 / 5 + $i;
15. }
17. var\_dump(microtime(**true**) - $v);

**Explanation**  
  
I did execute the preceding program using CMD so that I can see the delay in code snippet execution. I performed this execution 10 times and averaged the result, and after execution I was shocked due to the performance results.  
  
Now have a look at this table, and get ready for dropping your jaw. In this table I mentioned the performance measure of both PHP and Node.JS. (The time is in milliseconds.)  
  
Here is the table containing the performance measurements between these two:  
  
  
  
**Question 20: What are the main components of Node.js?**The main components of NodeJs are APIs, a V8 Engine and Libuv.  
  
**Libuv Library**  
  
Libuv is a multi-platform support library for asynchronous I/O. It was developed for Node.js using C and C++. But it's also used by Mozilla's Rust language, Luvit, Julia, pyuv and others.  
  
This libuv library is the main part for I/O related operations like reading files and interacting with the OS.  
  
You can check it out on GitHub for more information about the libuv library.  
  
**V8 Engine**  
  
From Google: “V8 is Google's open-source high-performance JavaScript engine, written in C++ and used in Google Chrome, the open source browser from Google. It implements ECMAScript as specified in ECMA-262, 3rd edition and runs on Windows XP and Vista, Mac OS X 10.5+, and Linux systems that use IA-32, ARM or MIPS processors. V8 can run standalone, or can be embedded into any C++ application.”  
  
If you are interested in learning more about the V8 engine, please visit here.   
  
**APIs (NodeJS Core Libs)**  
  
The NodeJs APIs are nothing but functions to do something upon your request. By default the NodeJSapis are asynchronous in nature but still you can use NodeJS APIs synchronously.  
  
For example, the http module could be used either synchronously or asynchronously.

1. var fs = require('fs');
2. fs.readFile(‘/files/help.txt’, function(err, buf)
3. {
4. // use fs.readFileSync() for sync operation. console.log(buf.toString());
5. }
6. );

**Question 21: Explain EventEmitter Object in NodeJS.**To access the EventEmitter class, use,  
  
*require('events').EventEmitter*API methods are accessed through an instance of the EventEmitter class. For this you need to create an object of the EventEmitter class, then you can access the method. For example,

1. varobj=**new** EventEmitter();

Using that you can access the API method via an EventEmitter instance. Let's see how in the following:

1. obj.emitEvent();

In Node we do not have the DOM so we use the EventEmitter class. This class has mainly two methods, "on" and "emit". I have given the example and sample use of these methods in the NodeJS environment.  
 **EventEmitter Class**  
The event module contains the EventEmitter class. The Util package provides the way to inherit from one class to another class. The "EventEmitter" class allows us to listen for events and assign actions to run when those events occur. The EventEmitter class is based on a publish/subscribe model because we can subscribe to events and then publish them.  
  
When we use theEventEmitter class we need to require an events module in the program, for example,

1. varev=require("events");

The ev object will then have a single property, which is the EventEmitter class itself.  
  
Objects that emit events are instances of "events.EventEmitter." You can access them using,

1. require("events")

Functions can be attached to objects, to be executed when an event is emitted; these functions are listeners. Inside a listener function, "this" refers to the "EventEmitter" that the listener was attached to.  
  
Features of EventEmitter class,

* This class is for managing events.
* Can be extended to provide event functionality in other classes.
* It emits events.
* It listens to and handles those events.
* Something that drives the two, causing events to be emitted.

**EventEmitter Methods**

* emitobj.on(event, listener)
* emitobj.emit(event,[arg1],[arg2],..[argN])
* emitobj.once(event, listener)
* emitobj.removeListener(event, listener)

**Question 22: Explain Event-Driven Programming in Node.js.**Event-Driven Programming is the term where the flow of the code is determined by events (click, load and so on). It's one of the basic milestones of today's popular programming languages, such as C#, Java and many more; I won't dwell on all of them here. In Node.js and moreover in any kind of JavaScript project, you'll be using or have used event-driven processes. Whether it's a page on load or a button click event, this is something you have done, whether you knew it or not.  
  
Let's make an example to the classic event-driven process and how it's done in NodeJS,

1. result = getJSONfromDestination();
2. binddata(result);

The operation above requires a blocking I/O process (single-threaded operation that waits for the previously working synchronous code).  
  
Now let's have a look at how we do the asynchronous way (a non-blocking I/O process).

1. json\_finished = function(result){
2. binddata(result);
3. }
5. getJSONfromDestination(jsonfinished);

As you can see, this is a non-blocking sample, because json\_finished does not work at first as you can imagine.  
It starts working when you call the getJSONfromDestination method and send param as the function to json\_finished.  
  
**Question 23: What are Request and Cheerio in Node.js NPM?**Request and Cheerio are our NPM packages. Cheerio doesn’t try to emulate a full implementation of the DOM. It specifically focuses on the scenario where you want to manipulate an HTML document using jQuery-like syntax. As such, it compares to jsdom favorably in some cases, but not in every situation.  
  
Cheerio itself doesn’t include a mechanism for making HTTP requests, and that’s something that can be tedious to handle manually. It’s a bit easier to use a module called request to facilitate requesting remote HTML documents. Request handles common tasks like caching cookies between multiple requests, setting the content length on POSTs, and generally makes life easier.  
  
**Question 24: Explain Karma And Jasmine in Node.js.  
  
Karma**Karma is a tool made on top of NodeJS to run JavaScript test cases. This is not a testing framework like Jasmine or Mocha or Chai etc. It only allows us to run JavaScript test cases written using testing frameworks like Jasmine.

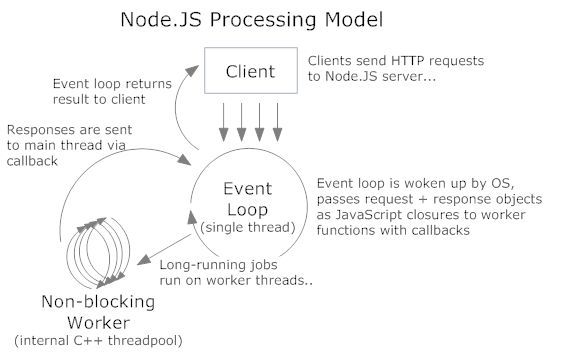
Karma runs JavaScript test cases against real browsers through Command Line Interface (CLI) rather than virtual browser and DOM. Since DOM rendering across browsers vary, for correctness of the test report it uses real browsers. It can configure what browsers need to be used while running Karma.  
  
**Jasmine**  
Jasmine is a **BDD (Behavior Driven Development)** JavaScript testing framework which provides building blocks to write JavaScript unit test cases. It does not require any other JavaScript Framework. As stated in official documentation of Jasmine 2.0.0,  
  
“*Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests.*”  
Here is a code snippet which describes the basic structure of Jasmine Unit Test Spec,

1. describe("simple test", function()
2. {      beforeEach(function(){
3. });
5. afterEach(function(){
6. });
8. it("a is a string", function(){
10. })   })

In the above sample code, there are four key keywords, “describe”, “it”, “beforeEach” and “afterEach”. “describe” and “it” functions accept two parameters.

* *A string*  
  This parameter is used to define a string value which explains the purpose of suite.
* *A function*  
  This is a block of code that implements the suite

**Question 25: What is Grunt in Node.js?**Grunt is a JavaScript Task Runner and this is a command line tool which runs on [NodeJS](http://www.c-sharpcorner.com/technologies/nodejs). It helps us to do all the above tasks very easily with a minimum effort. Grunt ecosystem and its huge list of plugins help us get all front-end code production ready.   
  
**Install Grunt**  
  
To install grunt in your project folder, run the below command in command line:  
***npm install grunt --save-dev***

**Question 26: Describe the Node.js processing model.**Node begins by a limited thread pool to receive the client requests. When a request arrives, it’s placed within a stack called “**Event Queue**”. The thing responsible to catch the request, to process and return it for client is called “Event Loop” that is the heart of Node and it works like a single-thread. However, if there is a process I/O, the Event Loop stops executing and calls an internal thread pool, done in C++, to execute the process and return for Event Loop when finished. But the great difference that we have noted is that the Event Loop does not wait for the return of Internal Thread Pool, it goes to catch other requests. With this concept Node.js is faster than current web servers, such as IIS or Apache.  
  
Below we have a diagram representing how Node works,  
  
  
  
**Question 27: What are the features of Node v5.4.0?**The Node v5.4.0 release also includes the following performance improvements:

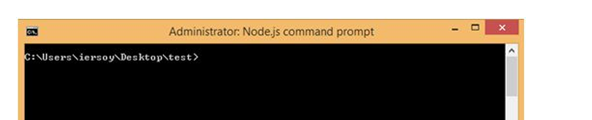
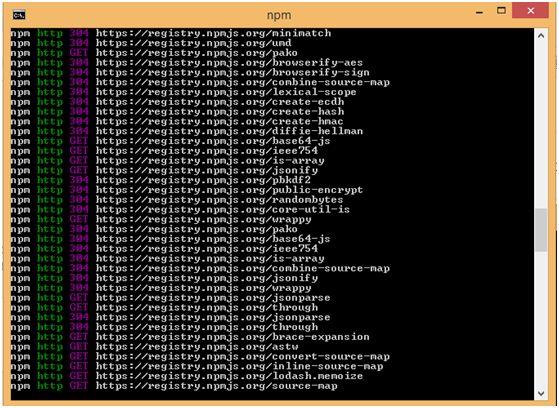
* ***assert***  
  deepEqual is now faster while comparing TypedArrays.
* ***lib***  
  Use arrow functions instead of bind where possible.
* ***node***  
  Enhanced accessor perf of process.env.
* ***node***  
  Enhanced performance of process.hrtime().
* ***node***  
  Enhanced GetActiveHandles performance.
* ***util***  
  Faster iteration in util.format().

**Question 28: Which companies are using Node.js?**Node.js has gained a lot of popularity among a wide variety of companies and in a short time span. The following companies are using the Node.js technology,

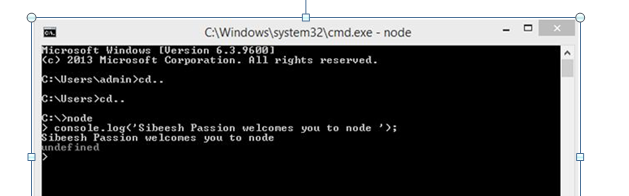
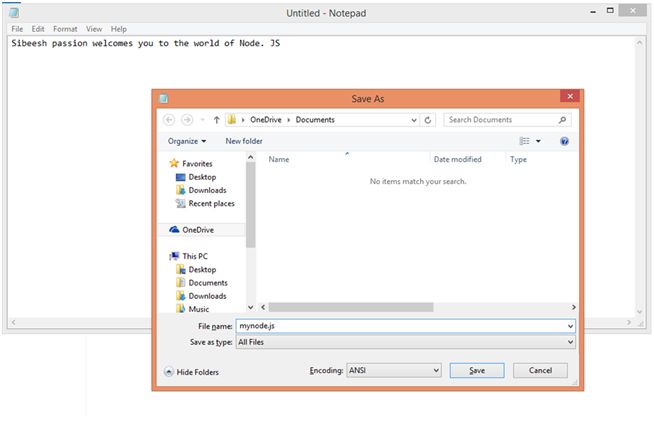
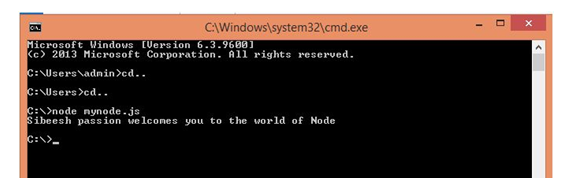
1. Yahoo! 2.LinkedIn 3.eBay 4.Microsoft

**Uses of Node.js,**

1. Web Services API’s 2.Real time multiplayer Games 3.Backend web services 4.Web applications

**Question 29: How can you install a module in Node.js?**Installing a module is easy, let's get back to <https://www.npmjs.com/> and select a module to install browserify:  
  
  
  
Now open your Nodejs Command Prompt and head to the folder you just created.  
  
  
  
Now add these lines to install browserify in the specified folder:  
***npm install browserify***  
  
Press Enter and let it download the dependencies and the browserify module itself.  
  
  
  
Once it's finished, you can have a look at your folder structure being replaced by the browserify module and its dependencies.  
  
**Question 30: How can you perform Testing in Node JS?**To check whether our Node JS is working, we can create a “Hello World” program in either of the following two ways,

* Directly type a message into the command window.
* Type the message in a JavaScript file and execute it in node.

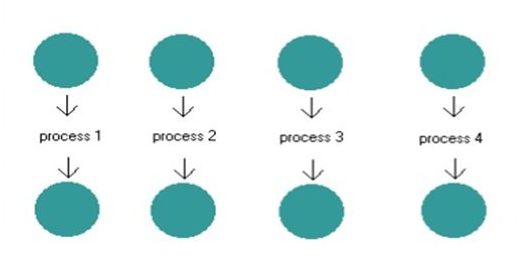
**Directly typing a message in a command window**  
  
Open your command window (press the Window key + R), type cmd in the window. Now a window will be opened, you can type your commands into it.  
  
  
  
Once you pointed out to your drive as shown in the image, type “**node**” and press Enter. Now you can type anything in your console using console.log. Then you will get the output as shown in the preceding image.  
  
Now you may be thinking, why does it return an undefined error? Am I right?  
  
It is because normally Node JS will display a return value for each command. As you all know, console.log does not return anything. So it is displaying "**undefined**".  
  
**Type the message in a JavaScript file and execute it in node**  
Create a JavaScript file manually. (To do so, open Notepad, type some content into it and save it with a .js extension. That is all.)  
  
  
  
I typed the content as “*console.log(‘Sibeesh passion welcomes you to the world of Node’);*”.  
  
Now you can execute that JavaScript file as follows.  
  
  
  
**Question 31: How can you create HTTPS Server WithNodeJS?**  
We use HTTPS when we need secure sessions. A secure session implies that the web browser will encrypt everything you do with a digitally signed certificate. So obviously before you do HTTPS, you need to create certificates. Let's therefore spend some time to see how to create certificates for SSL.  
  
**Creating HTTPS server**  
  
Just like we do for HTTP, an import to node's HTTP module, for HTTPS we import the HTTPS module. Also, since we need to pass in the certificate and key file, we also need to import the filestream (fs) module to enable Node to read the files. The following is the code to create the HTTPS server.

1. var https = require('https');
2. var fs = require('fs');
4. var options = {
5. key: fs.readFileSync('hostkey.pem'),
6. cert: fs.readFileSync('hostcert.pem')
7. };
9. https.createServer(options, function (req, res) {
10. res.writeHead(200);
11. res.end("hello world\n");
12. }).listen(8000);

**Question 32: How can you achieve Scale and Concurrency in Node.js?**This feature is inherited in NODE.js through these functionalities:

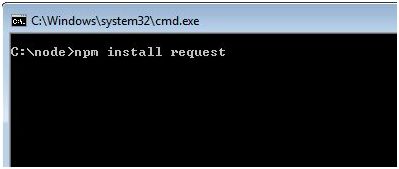
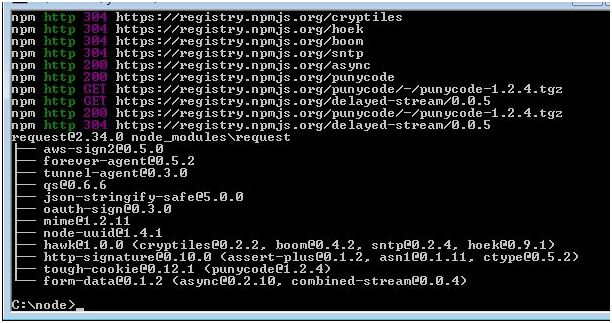
* Single thread and multiple process
* Asynchronous functioning/programming
* Size the thread pool correctly
* Node applications are built

**Single Threaded Environment**  
  
so what do these terms mean?

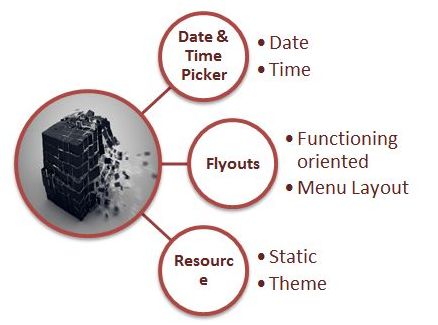
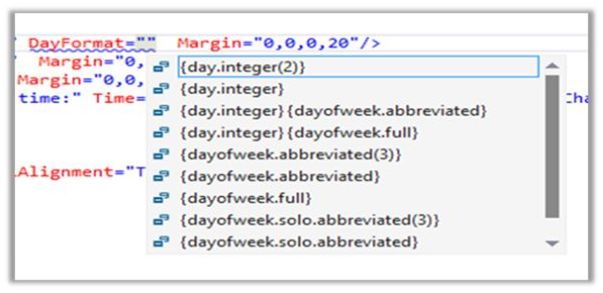
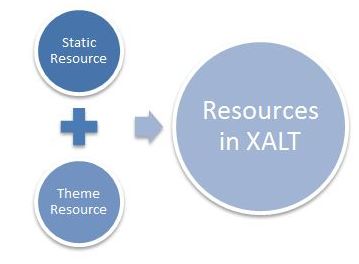
* In the simplest of words we can say that it takes requests from many clients but responds back in one operation or a thread.  
    
  

We can explain these features as,

* Client requests for something
* Server processes the request
* Server forwards a response to the client
* Server is again ready to process a new request

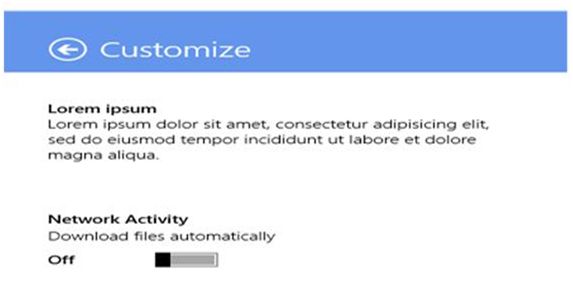
**Question 33: Explain “request” module in node.js.**  
The “*request*” module is not a core module of node.js, so we need to install it on our own.  
This command will install the request module in the current node.js application. Here is the screen to demonstrate the installation.  
  
  
Once we start the installation, we will find that it's downloading many files and installing in the current node.js application. You will see the following screen after finishing the installation.  
  
  
  
Now, we are sure that the “request” module is installed in the application and it's ready to use. Now, let's discuss it. What is the purpose of the request module?   
  
The beauty of node.js is, not only can we build a server but we can also build a client tool using node.js. A client in the sense that we can call some other server from a node.js application and get data from there, and to get data from another server, we can use the “request” module in node.js.   
  
Here is a simple implementation of the “request” module. In this module we are making a HTTP request to the Google server and then we are logging the response in the console. Have a look at the following example.

1. var http = require('http');
2. var request = require("request");
3. var server = http.createServer(function (req, res) {
4. request("http://www.google.com", function (error, response, body) {
5. console.log(body);
6. });
7. });
8. server.listen(9090);

**Question 34: What is XALT in Node.js?**  
These features can be used directly or indirectly in development work.  
These basic functionalities are as in the following:  
  
   
  
**XALT | Work Window**  
  
  
**XALT | Resources**  
  
whenever we talk about resources using XALT, we think about these two basic resources.  
  
  
  
**Question 35: What are the functionalities of XALT?**

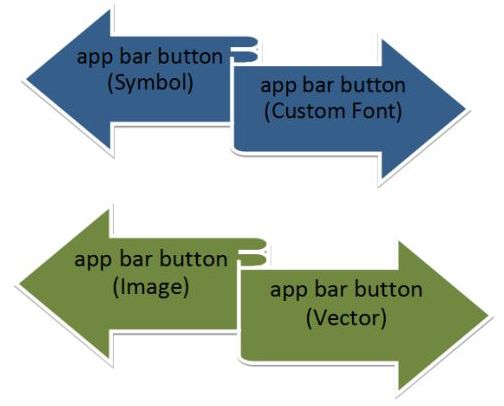
Extended level functionalities of XALT are as follows:

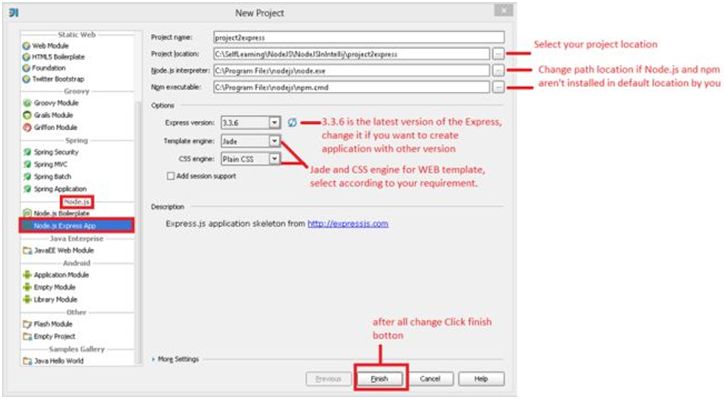
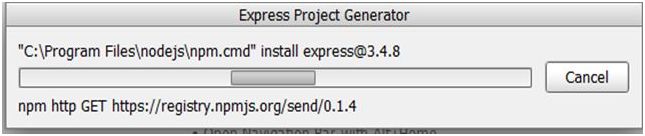
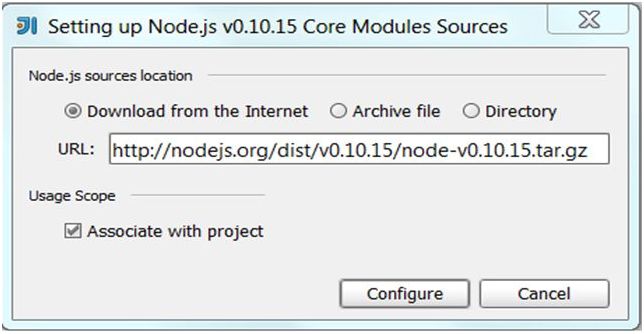
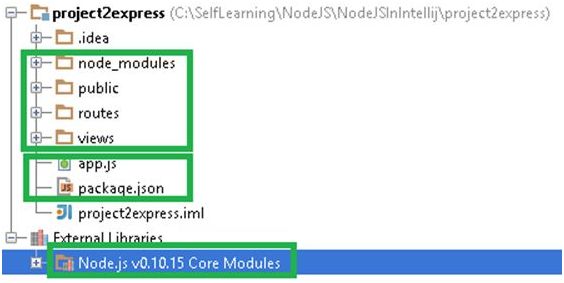
* Placement
* Animation
* Back Navigation
* Header Customization
* Design-Time Experience



**XALT | Controls**  
Controls in XALT can be several. This scenario depends upon the development need along with scope and modeling.  
  
**XALT | Hub Control**

1. <Hub Header="My App Name"
2. <HubSection Header="Section 1">
3. <DataTemplate>
4. <! -- Your content here -->

**XALT | App Bar Button**  
  
An App bar in XALT can easily be divided into four categories. These two categories have a different significance but the core functioning is the same in both cases.  
  
  
  
**Question 36: What is Express project and how can you create Express project?**

Express is a framework that allows the creation of the MVC web architecture in a Node JS application. There are many ways to create an Express project but in this article I will use Intellij IDEA to create a node Express project.  
  
**Create Express project**  
  
Open Intellij IDE and create a new project as in the following:  
  
  
  
Click on the Finish button and intellij will start downloading the required modules and files for your Express project.  
  
  
  
  
  
Click the "*Configure*" button to include the node core library in your project. This core library will help you to enable intelligence when writing node code.  
  
Finally our Express project is ready with its predefined structure.  
  
  
  
An Express project contains many folders and files. The above structure isn't very hard to understand, let's drill it down.

* ***node\_modules***  
  An Express project uses a MVC framework and it uses many other modules that we call dependency modules for the Express framework. So all dependency modules are stored in the node\_modules folder.
* ***public***  
  This folder contain CSS, JavaScript files and images.
* ***routes***  
  This folder contains a JS file that will help to route user URL. In this folder we can create many JS files and use them in an app.js file.
* ***views***  
  This is the folder where we add our jade page. This jade page will be shown in the browser.
* ***app.js***  
  This file is the heart of the Express framework. In this file we configure the Express application settings.
* ***package.json***  
  It is a JSON configuration file where we can configure project details and its dependencies

**Question 37: What is Asynchronous Programming?**Every function in node.js is asynchronous which means that everything that would normally block the thread is instead executed in the background, in other words it continues running in the background and the flow of control moves to the next line and executes them; the moment the operation is completed it executes a callback that will then be executed. For example if you are reading a file on the file system then you need to specify a callback function that is executed when the read operation has completed.  
  
**Question 38: What is middleware?**Middleware is actually a very simple concept to understand, it is very similar to middleman. What middleman does is he takes the request from one party and gets in touch with the other party; he does what needs to be done and then provides a response back to the party which placed the request in the first place. So middleware are just simple functions with three arguments; request, response and next, they act as a middleman; when express receives a request, express passes this request to each middleware in the chain until it receives a response. In general, there are two kinds of middleware, first one being a normal middleware, and the second one an error handling middleware.  
  
Middleware concept is at the core of express.js and very important to understand, almost everything in express.js is a middleware, from body parsers to loggers, from routes to error handlers. So it is very important to understand what are middlewares and how they work.  
  
A middleware has the following signature,

1. function aMiddleware(req, res, next){
2. //req: an express request object
3. //res: express response object, res.send, res.json,
4. //res.render etc can be used to generate response
6. /\* the voodoo \*/
7. }

**Question 39: How does the Node.js server work?**The require(): Node.js method provides a simple module system. Node.js programs can load individual modules using the require() method. While many modules must be explicitly downloaded, some modules, such as http, called core modules, are bundled with node.js.  
  
The HTTP server is created using the http module’s createServer() method. The createServer() takes a callback function as an argument. This callback function is executed each time the server receives a new request.  
  
The callback function takes two arguments, request and response. The request object contains information regarding the client’s request, such as the URL, HTTP headers, and much more. Similarly, the response object is used to return data back to the client.  
  
The call to listen() causes the server to bind to a port and listen for incoming connections. Here we have defined port to be 8080.  
  
**Question 40: What is Node.js Mean Stack?**It is comprised of four main technologies, with their supporting technologies,

* *Mongo DB*  
  The database.
* *Express*  
  The web framework.
* *AngularJS*  
  The front-end framework.
* *Node.JS*  
  The web server.

Now here comes the point of why we are learning Mean JS.  
  
One great thing about MEAN stack is, it not only uses JavaScript in the browser, it uses JavaScript throughout. Using the MEAN stack, you can code both the front end and back end in the same language.  
  
**Why we use it**

1. It provides good facilities for the backend developer and the frontend developer. It provides good user experience for front end developers and it provides good mechanics for the backend behind other scenes.
2. Mean JS supports 2000 libraries & also frameworks which have become popular in backend and frontend developing.
3. It reduces the server load, thus it reduces the cost.

**Introduction to Node.JS (In MEAN‘N’ is the Node.JS)**  
Node.JS is definitely not a new language, and it is not just a framework on JavaScript. It can be considered as a runtime environment for JavaScript built on top of Google's V8 engine. Node provides a purely event-driven, non-blocking infrastructure for building highly concurrent software.  
  
**Example**

1. var http = require('http');
2. http.createServer(function (req, res)
3. {
4. res.writeHead(200,
5. {
6. 'Content-Type': 'text/plain'
7. });
8. res.end('Hello World\n');
9. }).listen(8080, 'localhost');
10. console.log('Server running at http://localhost:8080');

Why we use Node JS in Mean Stack,

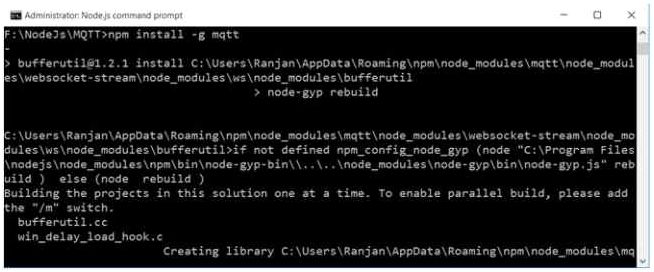
1. It has one main reason that is that Node JS is gaining more popularity so that you write your code in it. If you want to be a full stack developer you have to be proficient in at least several languages -- JavaScript on the frontend and for the backend PHP or Ruby.
2. It enables node.JS application to the server which allows more users on fewer server resources than other mainstream technologies.

**Question 41: What is Web Scraping in Node.js?**Web Scraping is the software technique of extracting the information from server side web applications. In this article we will see how things work by simply creating a web scraper using the DOM Parsing technique, and the tool which I am using is Node.js.

*crapeDataFromHtml* is our function and we create variables in the function for every item that we want to scrape from the website and then the data is serialized from our website in JSON format and then we have it once deserialization is done. In this case the circular red region gives me its relative node in the inspect window.

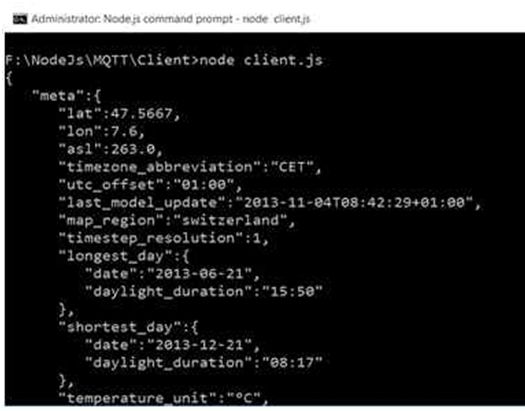
* First we reached url.
* Then we traversed DOM.
* Select our Nodes, the desired data we want to scrape
* Create your function, for instance scrapeDataFromHtml
* In this function, store all the data you want to scrape from website in variables.
* Write your logic and for multiple values you can use an array.
* Span and image are two things we want to scrape.

**Question 42: How can you use Node.js MQTT Library?**

MQTT is the most commonly used protocol in the internet of things. There are two key things one has to understand; i.e., the publisher and the subscriber. Let us see some background about publisher and subscriber.  
  
Subscriber is the one which is interested in receiving messages which are published by the publisher. You can have one or more subscribers registered to receive messages. Each of these clients gets subscribed to messages via topic or topics.   
  
Publisher is the one who is responsible for publishing the messages. All clients who are subscribed to a specific topic or topics will get notified with the message published by the publisher.  
  
We are building a sample “nodejs” client and server application using MQTT library to demonstrate how we can take advantage of how the MQTT nodejs library subscribes and publishes messages with ease. Node.js is very powerful, simple and easy to use. One can really take advantage of building simple and elegant applications using “nodejs”.  
 **Setting up the dependencies and required node.js libraries**  
  
**Step 1 🡪** Download and install [Python 2.7.10](https://www.python.org/downloads/).  
  
**Step 2 🡪** Execute the following command in command prompt.  
  
***npmconfig set python C:\Python\python.exe***  
  
**Step 3**  
  
Install MQTT node.js library by executing the following command.  
 *npm install –g mqtt*  
  
  
  
**Demo App**  
  
Here’s the scenario for which we are coding the node.js application.   
  
For demonstration purposes, we are making use of the following JSON weather forecast sample. 

**MQTT Node.js Client**  
  
The following is the code snippet for MQTT client, where you can see the client is getting subscribed to a specific topic – “*mydevice/forecast*”. The *handleMessage* is something which gets called asynchronously when the client receives a message from the publisher.

1. var http = require('http');
2. http.createServer(function (req, res)
3. {
4. res.writeHead(200,
5. {
6. 'Content-Type': 'text/plain'
7. });
8. res.end('Hello World\n');
9. }).listen(8080, 'localhost');
10. console.log('Server running at http://localhost:8080');

In order to run the client, just execute the *client.js*. You should see the message shortly after running the server code which actually publishes the message.  
The following is the snapshot of the client before receiving the message.   
  
  
After receiving published message from server.  
  
  
  
**MQTT Nodejs Server code**  
  
The following is the code snippet of MQTT Server program. We need to import ‘mqtt’ library and then create a client instance so that it can publish messages. We are going to read a “JSON” file named ‘*forecast.json*’ and send all the contents as part of the message that’s getting published.  
Notice below, the publisher has to use the same broker and topic name in order to publish messages to subscribed clients.

1. var mqtt = require('mqtt')
2. var fs = require('fs')
3. var broker = 'mqtt://test.mosquitto.org'
4. var client = mqtt.connect(broker)
5. var forecast = fs.readFileSync('../forecast.json').toString();
7. client.publish('mydevice/forecast', forecast)
8. client.end();
10. console.log('Successful!');

In order to run the server program, just key in “*node server.js*” and hit enter. The following is the snapshot of the program running on node.js.  
  
  
  
**Key thing about MQTT broker**  
  
you can find more information on http://test.mosquitto.org//. There’s a provision to set up our own MQTT server and make use of it too.  
  
**Question 43: How can you use Node.js and Angular JS with SQL Server?**  
Installation requirements for this:

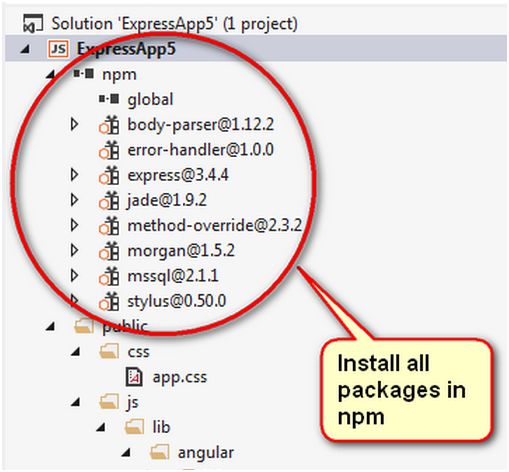
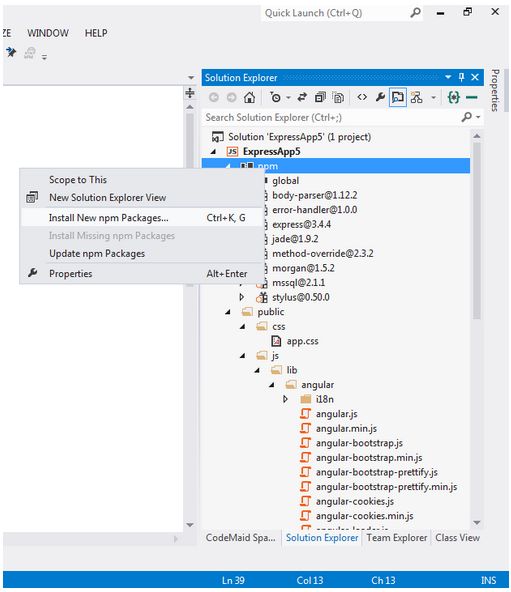
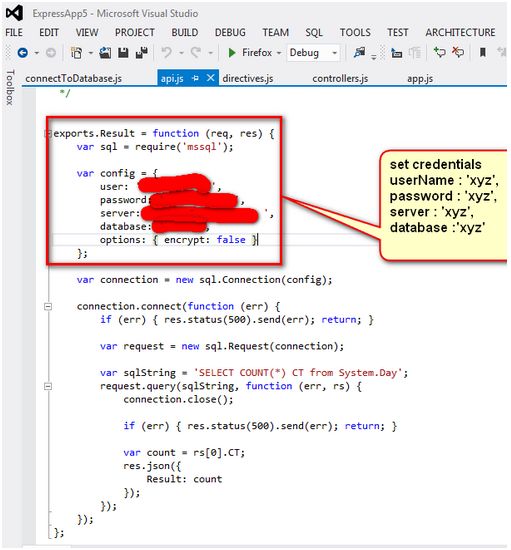
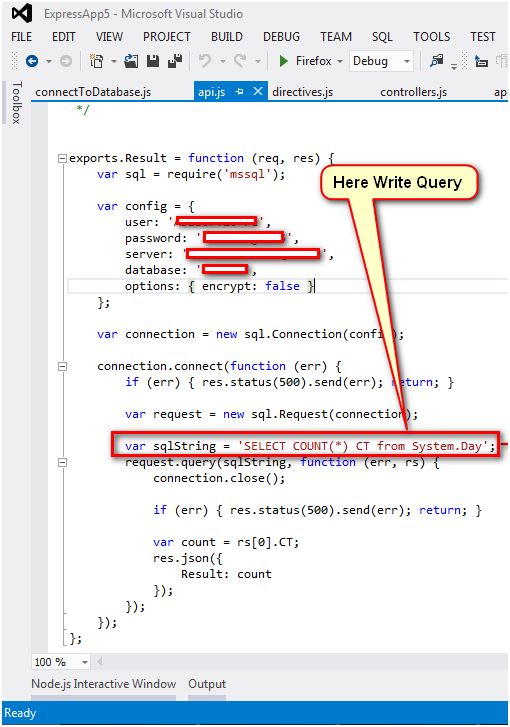
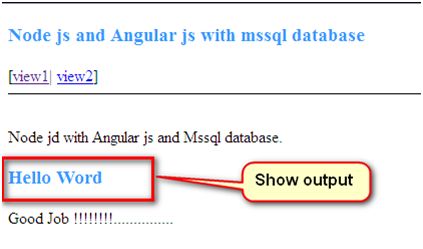
1. Install [npm](https://nodejs.org/en/" \t "_blank) (2).Install Visual Studio 2012 (3).Install SQL Server 2008 r2 (4) NTVS 1.0 [RC 2 Visual Studio 2012](https://nodejstools.codeplex.com/releases/view/612573). (5) Use Angular js library

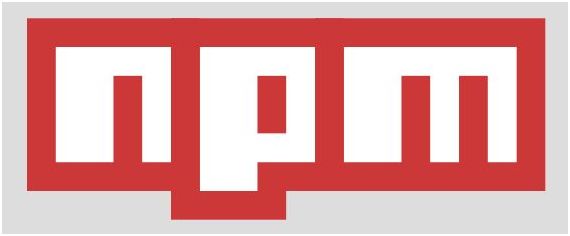
**Procedure**create a table in your database and insert a row as in the following,

1. **CREATE** **TABLE** [dbo].[Nodejs]([Id] [**int**] IDENTITY(1,1) NOT NULL,[Text] [nvarchar](50) NULL,
2. **CONSTRAINT** [PK\_Nodejs] **PRIMARY** **KEY** CLUSTERED
3. (    [Id] **ASC**
4. )**WITH** (PAD\_INDEX = **OFF**, STATISTICS\_NORECOMPUTE = **OFF**, IGNORE\_DUP\_KEY = **OFF**, ALLOW\_ROW\_LOCKS = **ON**, ALLOW\_PAGE\_LOCKS = **ON**) **ON** [**PRIMARY**]
6. ) **ON** [**PRIMARY**]

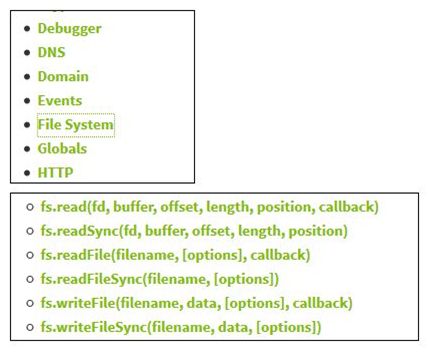
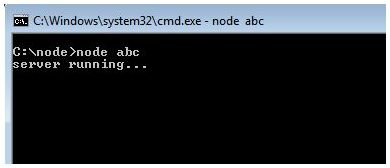
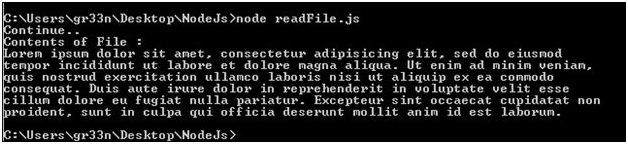
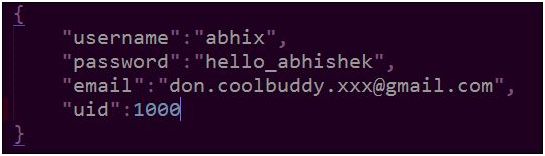
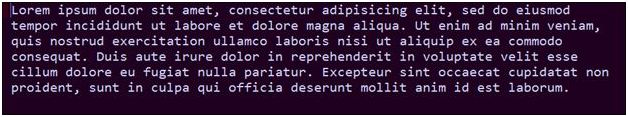
Insert a row into the table as in the following,

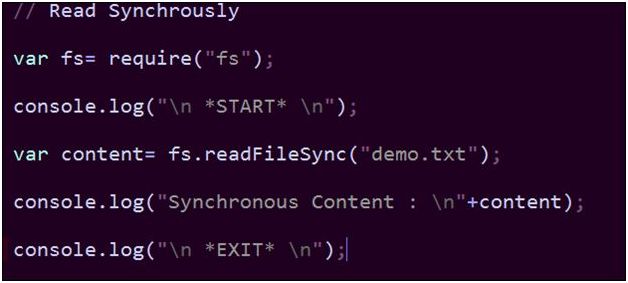
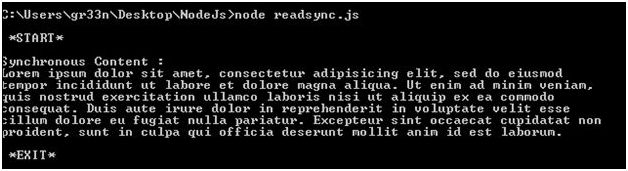
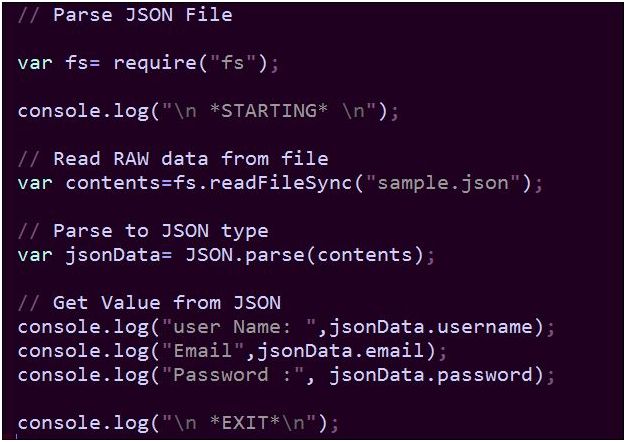
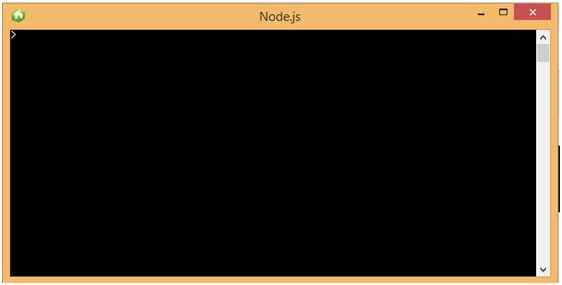
1. **INSERT** **INTO** [IMS1.0].[dbo].[Nodejs]
3. ([Text])
5. **VALUES**
7. ('Hello word!!')
9. GO

Open the project in Visual Studio and you will see the following folder structure,  
  
  
  
**How to install all packages**  
Right-click on npm and select *Install New npm Packages*.  
  
  
  
*<="" style="outline: 0px;">*  
Open the api.js file and set the Microsoft SQL database credentials and database name.  
  
  
  
Enter the following query,  
  
  
  
Please download the code and use the following procedure and run the code.  
  
Output Window,  
  
 

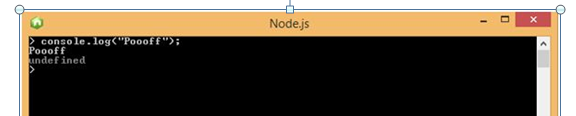
**Question 44: What are the popular modules of NPM in Node.js?**[Node Package Manager](https://www.npmjs.com/) is an online module repository (kind of like Play Store for Node module) that allows you to use publicly available and reusable modules with version and dependency management.  
  
  
  
Popular modules are,

* Express.js ,Connect, Socket.io, Jade, Mongodb, Redis, Coffee-script, Underscore

**Question 45: How can you read Files Using Fs?**When you look at the node.js documentation you will find the file system module.  
  
  
  
There you will find several functions and we will use one of them.  
  
**Code**  
  
  
  
So, start from the first line of code.   
fs is a variable for storing a reference of the file-system. In JavaScript we use the require() function to include an external module.  
Then, we need a callback function that will show your text-file content. So, it takes two arguments, error and data, where data is your content. Thus, we have used the console.log() function to show your content.  
  
And next we have the readFile() function that takes two arguments. First will be your resource-file's path and the second will be your callback function.  
  
And finally, we have written another print statement so that we can see the non-block nature of node.  
  
Before we move to the explanation part:  
  
**Output**  
  
  
  
As we can see, "*Continue*" is in the beginning and then we have the contents of the text file. That is just because node starts executing the code and when it gets to the readFile() function it doesn't stop there, else it moves to the next statement (non-blocking state) and when reading is completed it gets called to print the output. That's why we call that function as callback functions.  
  
**Question 46: How can you handleJSON Files in Node.JS?**  
JSON is a file format that is very reliable when transacting using APIs. And node plays a vital role in giving JSON a new makeover. Or, you can say it's a better alternative to XML.  
We will read the file synchronously. The JSON file is among those files that supports synchronous calls.  
So, first we need a dummy JSON file.  
And save it as sample.json.  
Whereas, the dummy text file would be like:  
  
  
  
Save it as demo.txt.  
  


**Code**  
  
first, we will deal with the synchronous call. It's a conventional way of reading any file. Either in C or PHP, they manage files synchronously. By synchronous, I mean one-way execution where you have a single-flow that executes line-by-line and it will move to the next statement when the statement completes. Unlike an asynchronous call, flow-control moves to the next line without showing any concern over its completion. And, after completion it will call the specified callback function.  
  
This shows the vital difference between blocking (synchronous) and non-blocking (asynchronous) calls.  
  
So, let's explore synchronous calls.  
  
   
  
And, the synchronous output will be like we expect always (single-flow).  
  
  
  
So, there is nothing new in it. But, when we read any JSON file then we can manipulate more. And filter the content of the file.  
  
  
  
Every time, you must use the synchronous way when you are dealing with a JSON file.   
  
And the output will be like:  
  
   
  
**Conclusion**  
  
keeping a constant value in a JSON file is always a good idea. Suppose you are setting a server and you are asked to maintain the credentials of the server. Then you put those credentials in a JSON file, so you can change it in the future.   
  
**Question 47: How can you cast a simple spell in Node.js?**  
Let's install the Node.js now via [http://nodejs.org](http://nodejs.org/). After completing the installation open the “*Node.js*” application with the Green icon to start casting your spells.  
  
   
  
Let's say we want to write “*Poooof*” on the console, here is how,

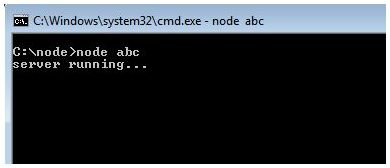
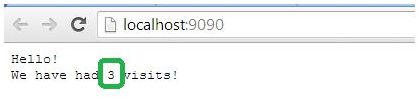
 console.log("Poooff");

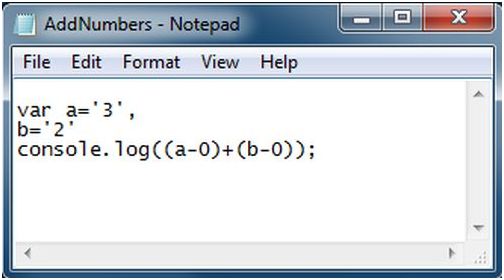
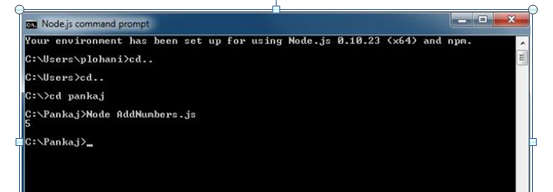
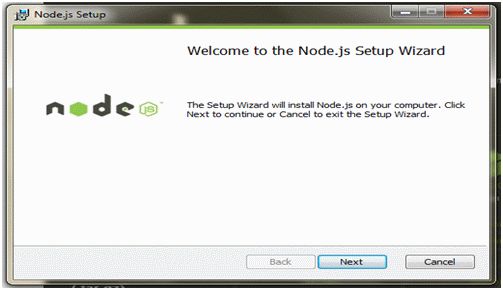
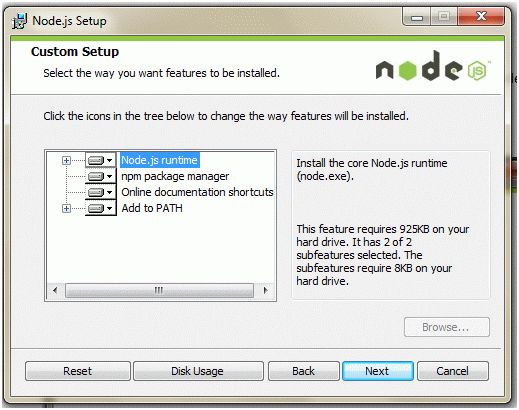
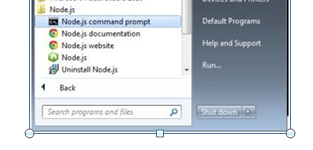
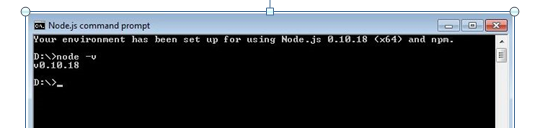
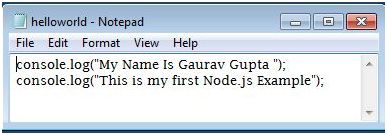
  
  
**Question 48: What are the key points of Node.js?**These are the key points about Node.js,

* Node.js is nothing but server-side JavaScript.
* Single-threaded environment.
* Asynchronous event driven server-side JavaScript.
* Runs on Google JS ver-8 Engine.
* Non-blocking I/O and File API.
* Highly scalable.
* Nodes apps are created.
* Achieving Scale and Concurrency
* Written in C and C++
* A file in node.js is called a Module.
* High performance.

**Question 49: How can you Implement Visitor Counter in Node.js?**We would just like to implement a simple hit count for our web server. For example, we know that in ASP.NET application variables are globally shareable and static in nature, in other words if a user changes a value then all users will see the changed version.  
  
In our example we will count the number of hits in our web server.  
  
Have a look at the following code.

1. var http = require('http');
2. var userCount = 0;
3. var server = http.createServer(function (req, res) {
4. userCount++;
5. res.writeHead(200, { 'Content-Type': 'text/plain' });
6. res.write('Hello!\n');
7. res.write('We have had ' + userCount + ' visits!\n');
8. res.end();
10. });
11. server.listen(9090);
12. console.log('server running...')

In this example we have declared one global variable that is counting the number of hits in our web application. In the server body, at first we are incrementing the variable value and then we are sending the value as a response after setting 200 response statuses.  
  
Let's run the application.  
  
Go to the command prompt and navigate to your server location. Then fire the command,  
  
*node<server filename>*  
  
  
  
We are seeing that our node server is running. Now we will go to a browser and try to access the server.  
  
  
  
At first hit, we see that it's showing that this is the first HTTP request to our server. Now, if we hit a couple of more times then we will see that the number has changed as in the following.  
  
  
  
It's showing the number of HTTP requests after the node server was up and running. The question may come to your mind, for each and every hit, why is the variable not initialized? The reason is, the node.js process won't work in event loop mechanism.  
Once we create the server and it is up, it will never stop and it will continue to listen for events. In our case the server is listening for request and response events. So, when we make one HTTP request , it executes the code that is written within the event scope and since we have declared a variable out of this scope, it is never initialized the second time.  
  
**Question 50: How can you add two numbers in console using Node.js?**When you need to convert strings to real numbers you use the ParseInt() or ParseFloat() functions, but a faster way is to subtract 0 since subtracting from a string is not possible; V8 tries to treat the contents of the string as a number.

**Step 1**Open an editor, for example Notepad and write the following code and save the file with a "*.js*" extension.  
  
  
  
**Step 2**Now open the Node.js Command prompt and execute the file as in the following figure.  
  
  
  
**Question 51: How can you perform the Hello World App using Node.js?**Here are some screenshots of the installation wizard of Node.js,  
  
  
  
  
After installing the package, to ensure that node.js was installed in your machine, from the Start menu go to "*All Programs*" and search for the node.js folder.  
After you find that, open the node.js command prompt from there.  
  
  
  
To see the current version of node.js installed, type the node -v on the node.js command prompt.  
  
  
This will print the current version of node js installed in your machine.  
Now, let's start to create a simple Hello World application using node.js.  
First, open the Notepad editor and write the following code.  
  
  
  
Then save file as "*HelloWorld.js."*  
Now, to execute this js file open the node.js command prompt. 

**Type***node HelloWorld.js*  
This will print the text you write in the text file on the console.  
  
