



# NodeJS HandBook

## Contents

What is Javascript?.....	3
Advance Javascript?.....	3
Disadvantages of Javascript.....	3
The modern mode, "use strict".....	3
Variables.....	3
Code quality.....	4
Debugging in Chrome.....	4
Why we need tests?.....	6
Coding Style.....	7
Datatype.....	7
Polyfills.....	7
Objects: the basics.....	8
Objects.....	8
Symbol Type Symbols.....	8
Global Symbols:.....	9
Constructor ,operator.....	9
Advance Working With Function.....	12
Advance Working With Function Recursion and stack.....	12
Advance Working With Function Rest parameters and spread operator.....	13
Advance Working With Function Closure.....	13
Advance Working With Function.....	14
Function binding.....	14
Arrow functions revisited.....	14
Forms, Controls, Regular Expression Promises, async/await.....	15
Node JS.....	15
V8 Engine.....	15
LIBUV.....	16
What is Node Js?.....	16
Advantages of Node.js.....	16

# TOPS Technologies Pvt.Ltd

---

Features of Node.js.....	17
Who Use Node JS.....	17
Why Node JS?.....	18
How to Install Node JS.....	18
The Node.js Runtime.....	18
App Hello World.....	21
What are Modules in Node Js?.....	21
Create NPM Module.....	23
When use Node JS?.....	23
Create Webserver in Node JS.....	23
How Webserver Make?.....	24
Handling get Request in Node.....	25
What is Callback?.....	26
Event Loop.....	26
Event Loop Explained.....	27
EventEmitter Class.....	28
Buffer.....	29
What is Express Js.....	29
Why Express Js.....	29
How Express Js.....	30
What are Routes?.....	30
Web Server Using Express.js.....	30
Web Server Using Express.js.....	31
What is Mongo DB?.....	31
Why Mongo DB?.....	31
HOW use Mongo DB?.....	31
What is Promises?.....	33
Create Custom Promise.....	33

## What is Javascript?

- The programs in this language are called *scripts*. They can be written right in a web page's HTML and executed automatically as the page loads.
- Scripts are provided and executed as plain text. They don't need special preparation or compilation to run.
- The browser has an embedded engine sometimes called a "JavaScript virtual machine".
- Different engines have different "codenames". For example:
  - V8 – in Chrome and Opera.
  - SpiderMonkey – in Firefox.

## Advance Javascript?

- Add new HTML to the page, change the existing content, modify styles.
- React to user actions, run on mouse clicks, pointer movements, key presses.
- Send requests over the network to remote servers, download and upload files (so-called AJAX and COMET technologies).
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side ("local storage").

## Disadvantages of Javascript

- JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS system functions.
- There are ways to interact with camera/microphone and other devices, but they require a user's explicit permission.
- Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other if they come from different sites (from a different domain, protocol or port)

JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that's a safety limitation

## The modern mode, "use strict"

- This was the case until 2009 when ECMAScript 5 (ES5) appeared.
- It added new features to the language and modified some of the existing ones.

- To keep the old code working, most modifications are off by default.
- You need to explicitly enable them with a special directive: "use strict".
- Example:
  - "use strict"; // this code works the modern way ...

## Variables

- A variable is a “named storage” for data. We can use variables to store goodies, visitors, and other data.
- To create a variable in JavaScript, use the let keyword.
- Example:

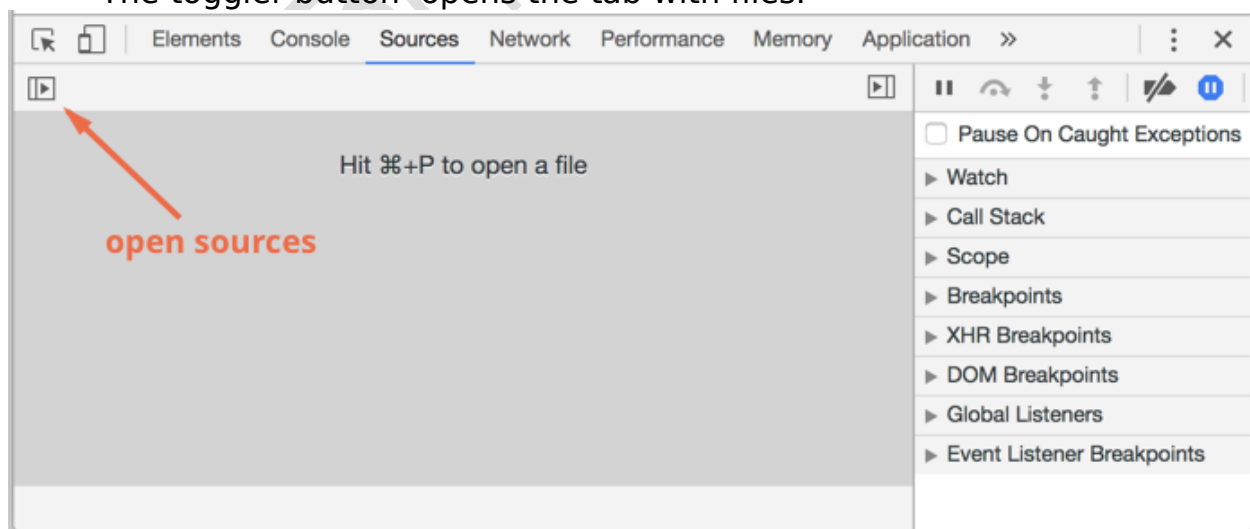
```
<script>
'use strict'; let message;
    message = 'Hello!';
alert(message); // shows the variable content
</script>
```

## Code quality

- Debugging in Chrome
- Coding Style
- Comments
- Ninja code
- Automated testing with mocha
- [Polyfills](#)

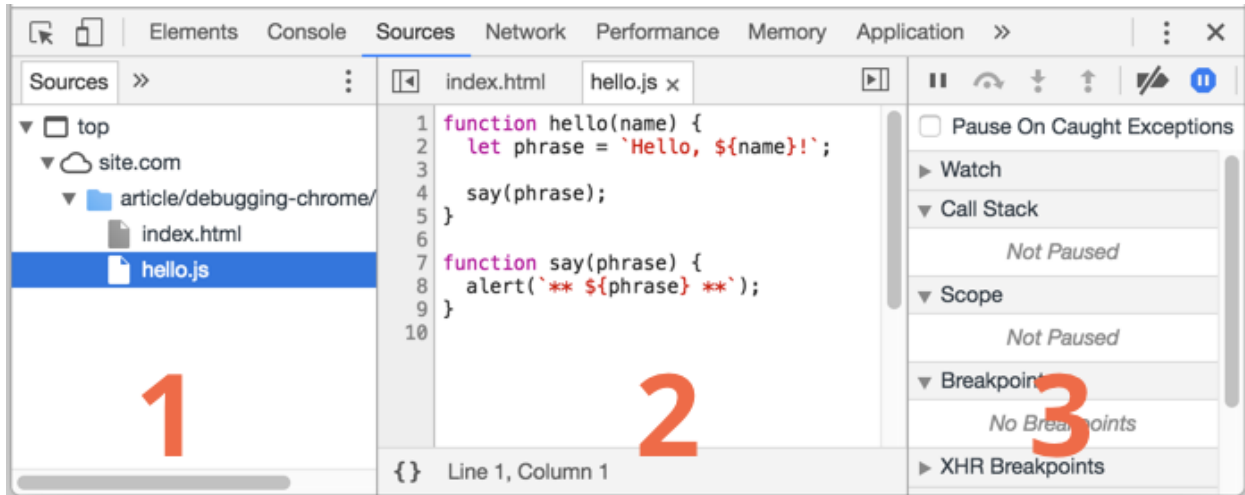
## Debugging in Chrome

- All modern browsers and most other environments support “debugging” - a special UI in developer tools that makes finding and fixing errors much easier.
- The toggler button opens the tab with files.

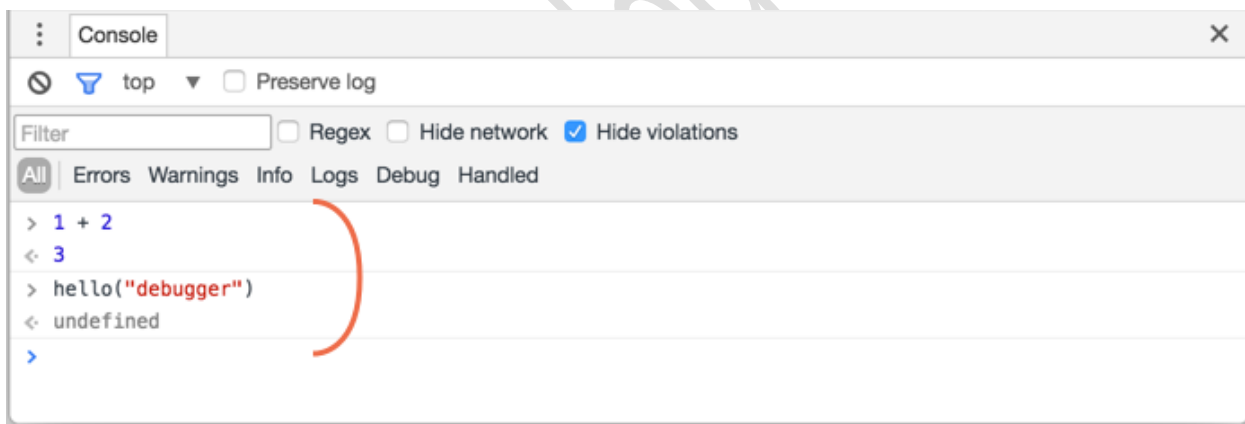


- Let's click it and select index.html and then hello.js in the tree view. Here's what should show up:
- Here we can see three zones:

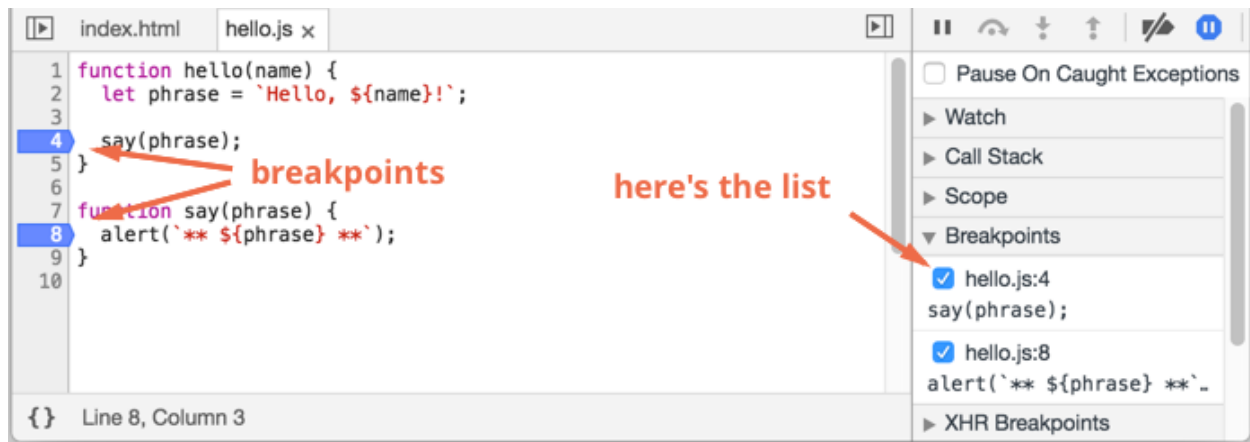
- The **Resources zone** lists HTML, JavaScript, CSS and other files, including images that are attached to the page. Chrome extensions may appear here too.
- The **Source zone** shows the source code.
- The **Information and control zone** is for debugging, we'll explore it soon.



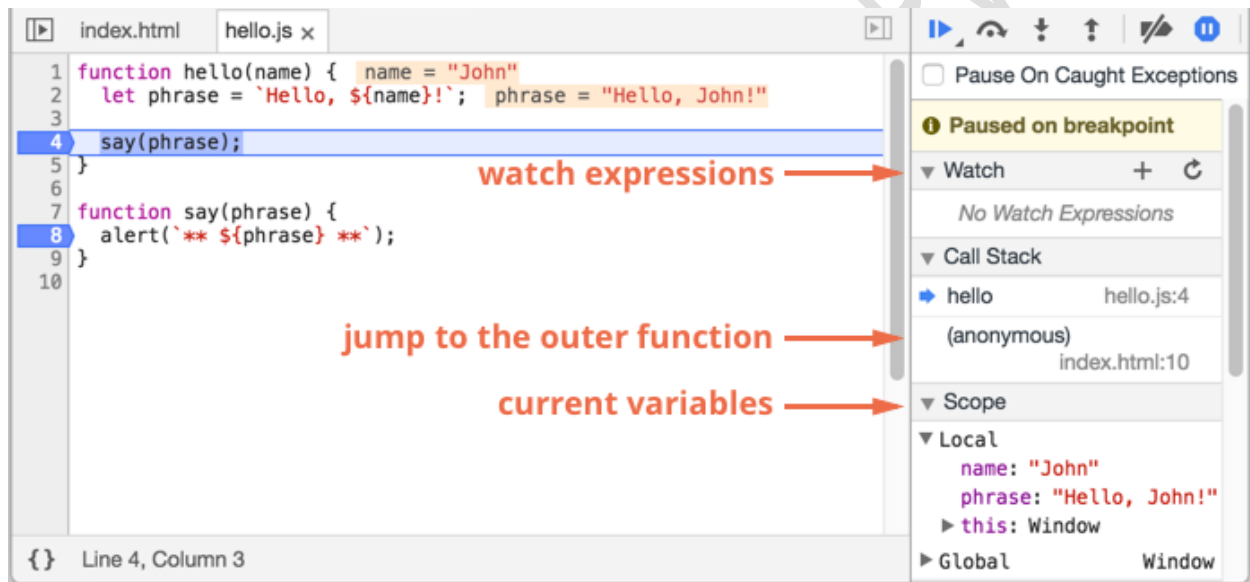
### Console



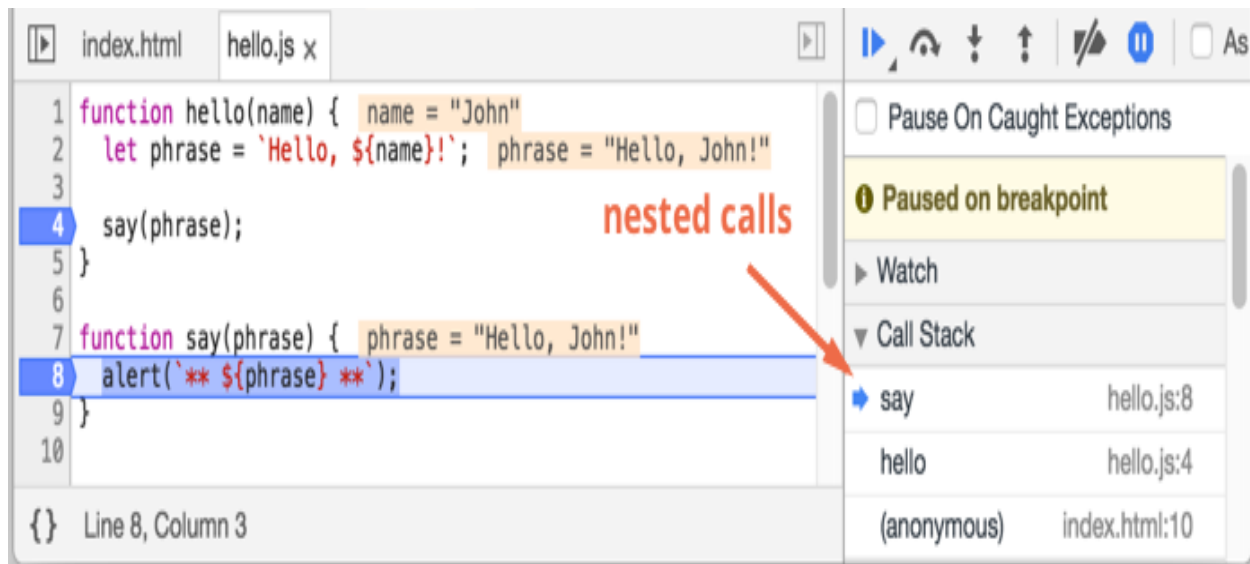
### Breakpoints



## Pause and look Around Command



## Tracing the Execution



Automated testing with mocha

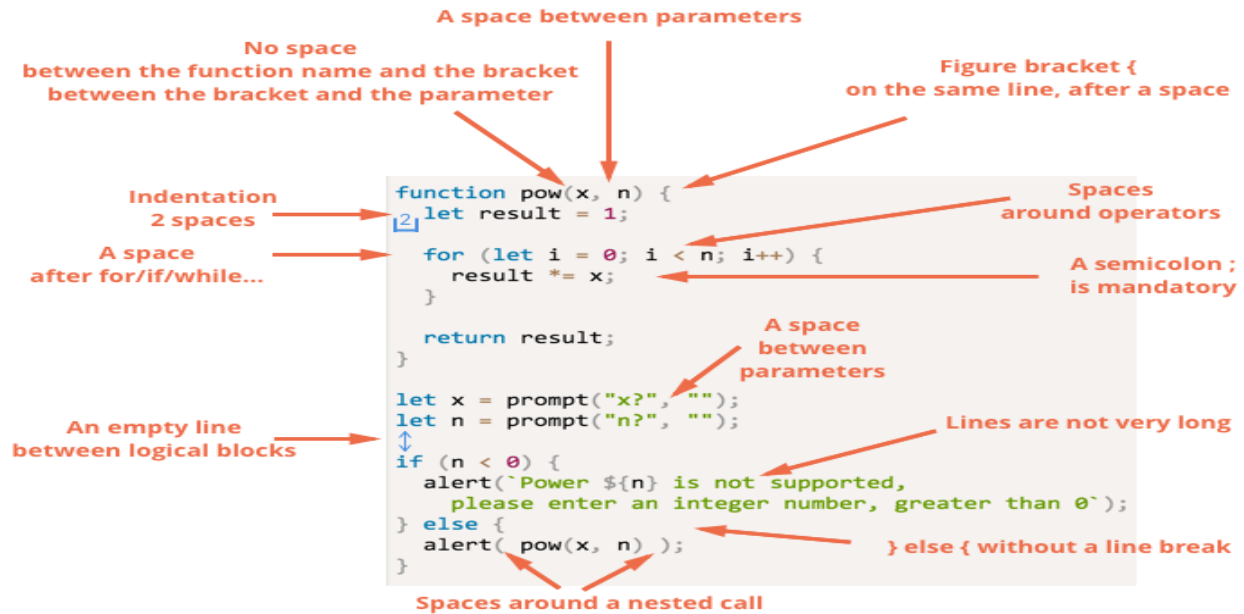
- Automated testing will be used in further tasks.
- It's actually a part of the "educational minimum" of a developer.

## Why we need tests?

- When we write a function, we can usually imagine what it should do: which parameters give which results.
- During development, we can check the function by running it and comparing the outcome with the expected one. For instance, we can do it in the console.



## Coding Style



## Datatype

- A variable is a “named storage” for data. We can use variables to store goodies, visitors, and other data.
- To create a variable in JavaScript, use the `let` keyword.
- Example:  
 <script>  
 'use strict'; let message;  
 message = 'Hello!';  
 alert(message); // shows the variable content  
 </script>

## Polyfills

- The transpiler rewrites the code, so syntax features are covered.
- But for new functions we need to write a special script that implements them. JavaScript is a highly dynamic language, scripts may not just add new functions, but also modify built-in ones, so that they behave according to the modern standard.
- There's a term “polyfill” for scripts that “fill in” the gap and add missing implementations.
- Two interesting polyfills are:
  - [babel polyfill](#) that supports a lot, but is big.
  - [polyfill.io](#) service that allows to load/construct polyfills on-demand, depending on the features we need.

## Objects: the basics

- Objects- Symbol type
- Garbage collection
- Object methods, "this"
- Object to primitive conversion
- Constructor, operator "new"

## Objects

- An object can be created with figure brackets {...} with an optional list of *properties*. A property is a "key: value" pair, where key is a string (also called a "property name"), and value can be anything.
- We can imagine an object as a cabinet with signed files. Every piece of data is stored in its file by the key. It's easy to find a file by its name or add/remove a file.
- Example:
  - `let user = new Object();` // "object constructor"
  - syntax
    - `let user = {};` // "object literal" syntax

## Symbol Type Symbols

- "Symbol" value represents a unique identifier.
- A value of this type can be created using `Symbol()`:

Example:

```
// id is a new symbol
```

```
let id = Symbol();
```

Hidden Property:

```
<script>
```

```
'use strict';
```

```
let user = { name: "John" };
```

```
let id = Symbol("id");
```

```
user[id] = "ID Value";
```

```
alert( user[id] ); // we can access the data using the symbol as the key
```

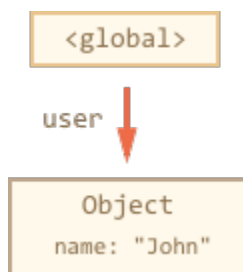
```
</script>
```

## Global Symbols:

```
<script> 'use strict'; // read from the global registry let id =  
Symbol.for("id"); // if the symbol did not exist, it is created // read it  
again let idAgain = Symbol.for("id"); // the same symbol alert( id  
=== idAgain ); // true </script>
```

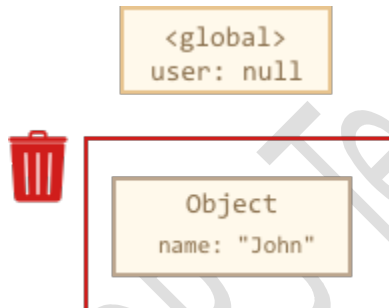
### Garbage Collection

- Memory management in JavaScript is performed automatically and invisibly to us. We create primitives, objects, functions... All that takes memory.
- Example:
  - // user has a reference to the object
    - let user = { name: "John" };



Here the arrow depicts an object reference. The global variable "user" references the object {name: "John"} (we'll call it John for brevity). The "name" property of John stores a primitive, so it's painted inside the object.

If the value of user is overwritten, the reference is lost:



```
user = null;
```

## Constructor ,operator

- The regular {...} syntax allows to create one object. But often we need to create many similar objects, like multiple users or menu items and so on.
- That can be done using constructor functions and the "new" operator.
- Constructor function
  - Constructor functions technically are regular functions. There are two conventions though:
  - They are named with capital letter first.

- They should be executed only with "new" operator.

Example:

```
<script>

    'use strict';

    function User(name) {

        this.name = name;

        this.isAdmin = false;

    }

    let user = new User("Jack");

    alert(user.name); // Jack

    alert(user.isAdmin); // false

</script>
```

### Classes and Inheritance

- The “class” construct allows to define prototype-based classes with a clean, nice-looking syntax.
- Example:

```
    □ <script>

    'use strict';

    class User

    {

        constructor(name)

        {   this.name = name;   }

        sayHi()

        {   alert(this.name);   }

    }

    let user = new User("John");

    user.sayHi();
```

</script>

### Classes and Inheritance

- Classes can extend one another. There's a nice syntax, technically based on the prototypal **inheritance**.
- To inherit from another class, we should specify "extends" and the parent class before the brackets {..}.

- Here Rabbit inherits from Animal:

- <script>

- 'use strict';

- class Animal {

- constructor(name) {

- this.speed = 0;

- this.name = name;

- }

- run(speed) {

- this.speed += speed;

- alert(` \${this.name} runs with speed \${this.speed}.`);

- stop() {

- this.speed = 0;

- alert(` \${this.name} stopped.`);

- }

// Inherit from Animal

class Rabbit extends Animal {

hide() { alert(` \${this.name} hides!`); }

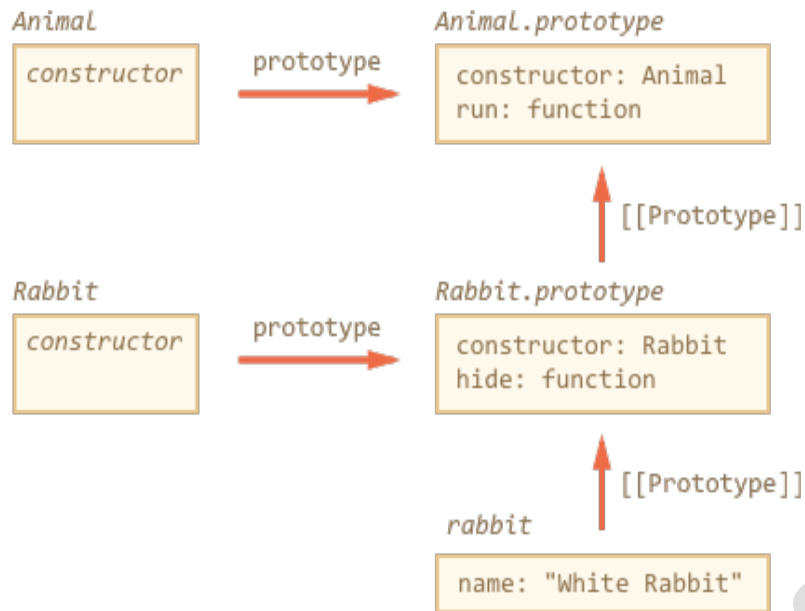
}

let rabbit = new Rabbit("White Rabbit");

rabbit.run(5); // White Rabbit runs with speed 5.

rabbit.hide(); // White Rabbit hides!

</script>



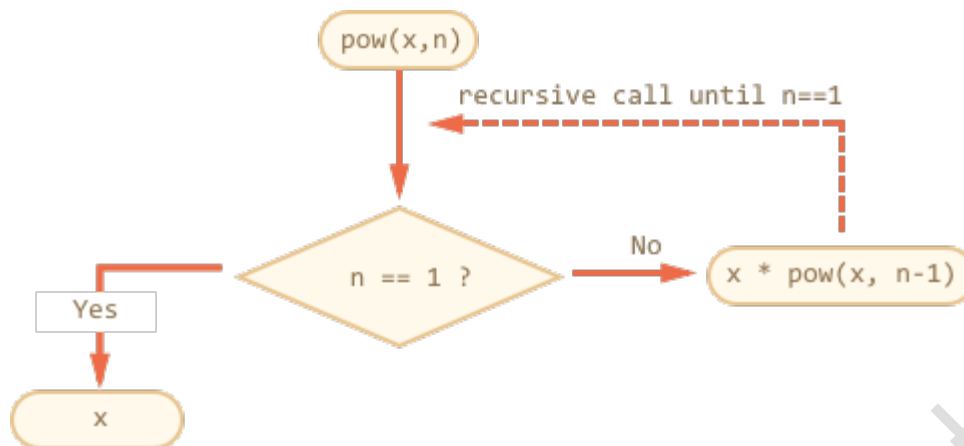
## Advance Working With Function

- Recursion and stack
- Rest parameters and spread operator
- Closure
- Global Object & function Object
- Decorators and forwarding, call/apply
- Function binding
- Arrow functions revisited

## Advance Working With Function Recursion and stack

- Recursion is a programming pattern that is useful in situations when a task can be naturally split into several tasks of the same kind, but simpler. Or when a task can be simplified into an easy action plus a simpler variant of the same task. Or, as we'll see soon, to deal with certain data structures.
- When a function solves a task, in the process it can call many other functions. A partial case of this is when a function calls *itself*. That's called *recursion*.
- Example:

```
<script>
'use strict';
function pow(x, n) {
    let result = 1;
    // multiply result by x n times in the loop
    for (let i = 0; i < n; i++)
    {
        result *= x;
    }
    return result;
}
alert( pow(2, 3) ); // 8
</script>
```



## Advance Working With Function Rest parameters and spread operator

- A function can be called with any number of arguments, no matter how it is defined.
- Example:

```
<script>  'use strict';
        function sum(a, b)
        {
            return a + b;
        }
        alert( sum(1, 2, 3, 4, 5) );
</script>
```

- There will be no error because of “excessive” arguments. But of course in the result only the first two will be counted.

## Advance Working With Function Closure

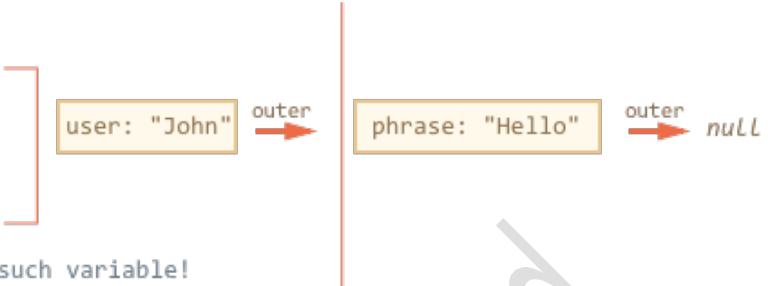
- JavaScript is a very function-oriented language. It gives us a lot of freedom. A function can be created at one moment, then copied to another variable or passed as an argument to another function and called from a totally different place later.
- We know that a function can access variables outside of it; this feature is used quite often.
- The examples above concentrated on functions. But Lexical Environments also exist for code blocks `{...}`.
- They are created when a code block runs and contain block-local variables. Here are a couple of examples.



```
let phrase = "Hello";

if (true) {
  let user = "John";
  alert(`${phrase}, ${user}`);
}

alert(user); // Error, can't see such variable!
```



## Advance Working With Function Global Object

- When JavaScript was created, there was an idea of a “global object” that provides all global variables and functions. It was planned that multiple in-browser scripts would use that single global object and share variables through it.
- Since then, JavaScript greatly evolved, and that idea of linking code through global variables became much less appealing. In modern JavaScript, the concept of modules took its place.
- But the global object still remains in the specification.
- Example:
- `<script>`

```
    var phrase = "Hello";
    function sayHi() {    alert(phrase);    }    // can read
from window
    alert( window.phrase ); // Hello (global var)
    alert( window.sayHi ); //

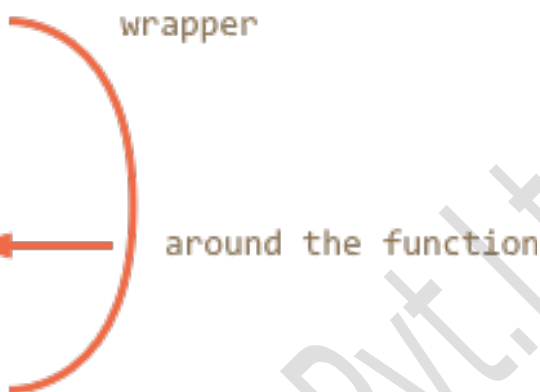
function (global function declaration)    // can write to window
(creates a new global variable)    window.test = 5;    alert(test); // 5
</script>
```

## Advance Working With Function

Decorators and forwarding, call/apply

- JavaScript gives exceptional flexibility when dealing with functions. They can be passed around, used as objects, and now we’ll see how to *forward* calls between them and *decorate* them.

```
function cachingDecorator(func) {  
  let cache = new Map();  
  
  return function(x) {  
    if (cache.has(x)) {  
      return cache.get(x);  
    }  
  
    let result = func(x);  
  
    cache.set(x, result);  
    return result;  
  };  
}
```



## Function binding

- When using `setTimeout` with object methods or passing object methods along, there's a known problem: "losing this".
- Suddenly, this just stops working right. The situation is typical for novice developers, but happens with experienced ones as well.
- Example:  
let user = { firstName: "John" };  
function func() { alert(this.firstName); }      let funcUser =  
func.bind(user);      funcUser(); // John

## Arrow functions revisited

- Arrow functions are not just a "shorthand" for writing small stuff.
- JavaScript is full of situations where we need to write a small function, that's executed somewhere else.
- As we remember from the chapter Object methods, "this", arrow functions do not have this. If this is accessed, it is taken from the outside.
- Example:  
let group = { title: "Our Group", students: ["John", "Pete", "Alice"],  
 showList() {  
 this.students.forEach(      student => alert(this.title + ': ' + student)  
 );  
} };  
group.showList();

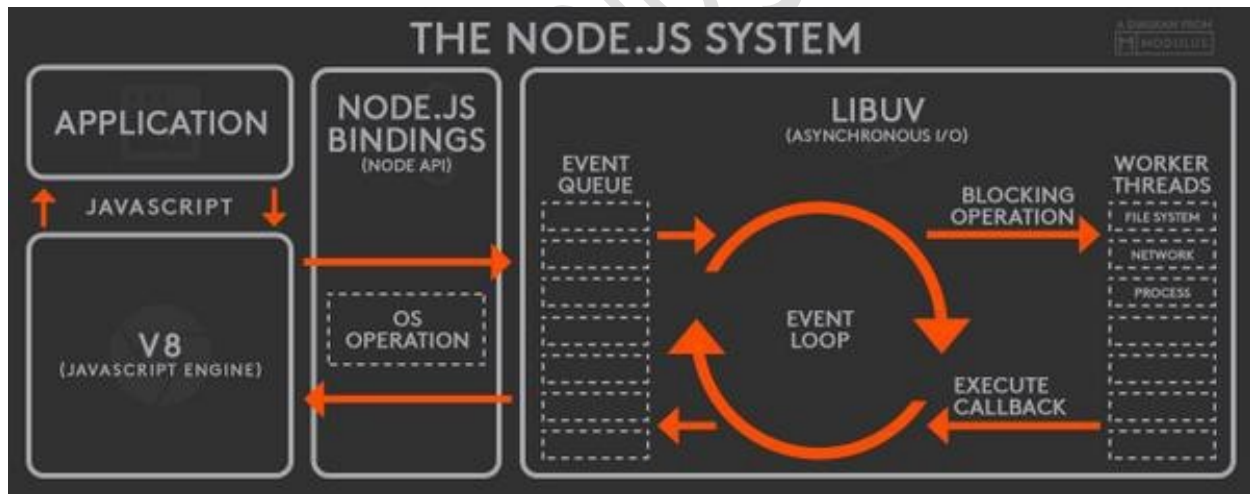
## Forms, Controls, Regular Expression Promises, async/await

Introduction: callbacks

- Many actions in JavaScript are *asynchronous*.
- For instance, take a look at the function loadScript(src):
- Example:

```
<script> 'use strict';
function loadScript(src, callback) {
  let script = document.createElement('script');
  script.src = src;
  script.onload = () => callback(script);
  document.head.append(script);
}
loadScript('https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js', script =>
{
  alert(`Cool, the ${script.src} is loaded`);
  alert(_); // function declared in the loaded script
});
</script>
```

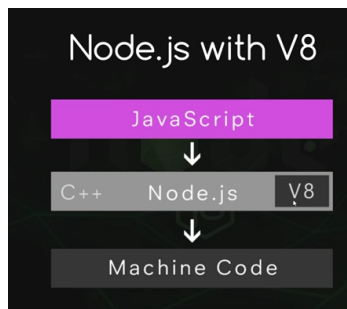
## Node JS



## V8 Engine

The Chrome V8 engine :

- The V8 engine is written in C++ and used in Chrome and Nodejs.
- It implements ECMAScript as specified in ECMA-262.
- The V8 engine can run standalone we can embed it with our own C++ program.



## LIBUV

LIBUV (Unicorn Velociraptor Library ) is a multi-platform C library that provides support for asynchronous I/O based on event loops.



It is primarily designed for use in Node.js but it is also used by other software projects

## What is Node Js?

Node.js is an open-source, cross-platform runtime environment used for development of server-side web applications. Node.js applications are written in JavaScript and can be run on a wide variety of operating systems.

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

Node.js was written and introduced by Ryan Dahl in 2009. Visit Wikipedia to know the history of Node.js.

Node.js official web site: <https://nodejs.org>

Node.js on github: <https://github.com/nodejs/node>

Node.js community conference <http://nodeconf.com>

## Advantages of Node.js

Node.js is an open-source framework under MIT license. (MIT license is a free software license originating at the Massachusetts Institute of Technology (MIT).)

Uses JavaScript to build entire server side application.

Lightweight framework that includes bare minimum modules. Other modules can be included as per the need of an application.

Asynchronous by default. So it performs faster than other frameworks.

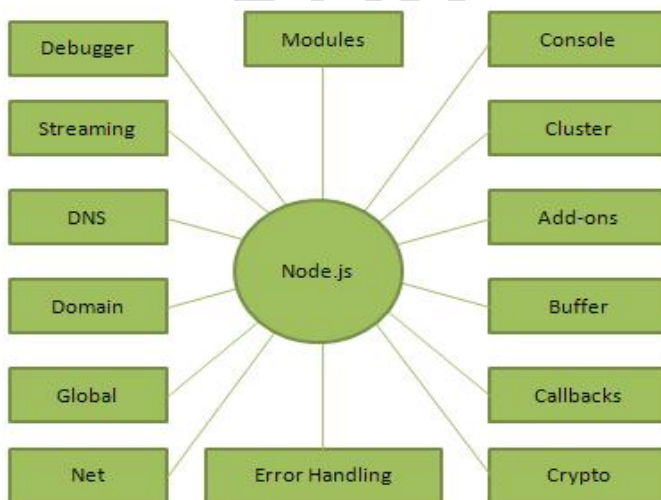
Cross-platform framework that runs on Windows, MAC or Linux

## Features of Node.js

1. **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.
2. **Very Fast** – Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
3. **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.
4. **No Buffering** – Node.js applications never buffer any data. These applications simply output the data in chunks.
5. **License** – Node.js is released under the MIT license

## Concepts

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



## Who Use Node JS

Node.js is used by a variety of large companies. Below is a list of a few of them.

Paypal – A lot of sites within Paypal have also started the transition onto Node.js.

LinkedIn - LinkedIn is using Node.js to power their\_Mobile\_Servers, which powers the iPhone, Android, and Mobile Web products.

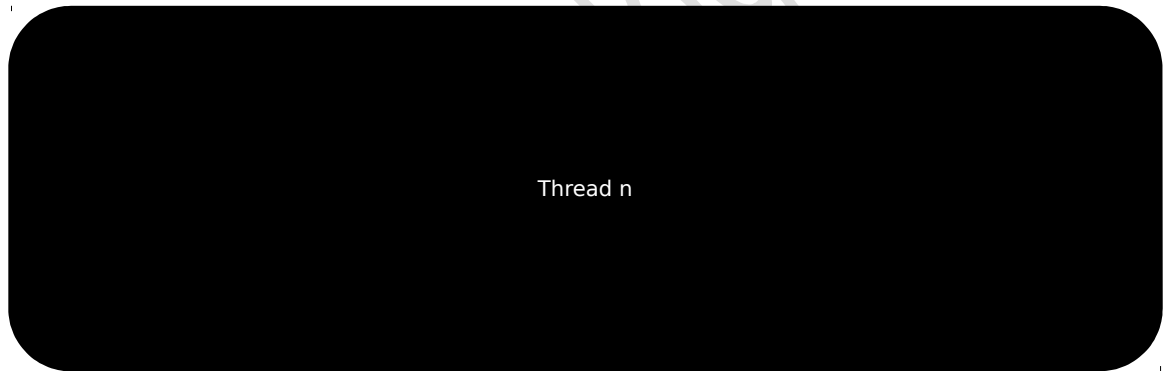
Mozilla has implemented Node.js to support browser APIs which has half a billion installs.

Ebay hosts their HTTP API service in Node.js

## Why Node JS?

In these sort of applications, it is up to the developer to ensure the right code was put in place to ensure the state of web session was maintained while the user was working with the system.

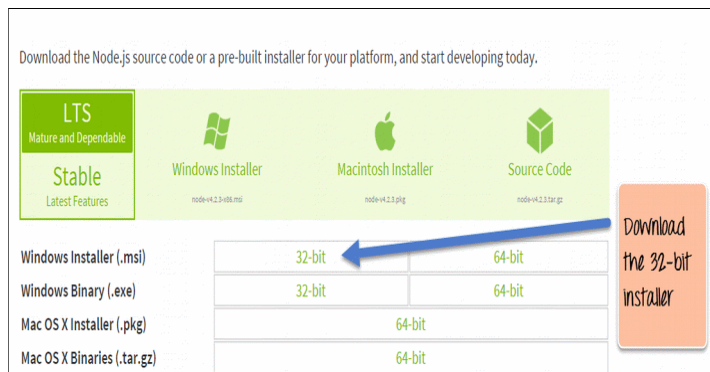
But with Node.js web applications, you can now work in real-time and have a 2-way communication. The state is maintained, and the either the client or server can start the communication.



Thread n

## How to Install Node JS

Step 1) Go to the site <https://nodejs.org/en/download/> and download the necessary binary files. In our example, we are going to the download the 32-bit setup files for Node.js.

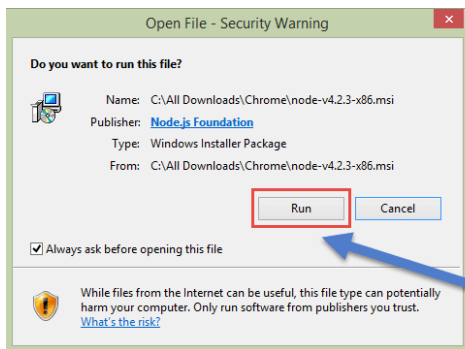


## The Node.js Runtime

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.

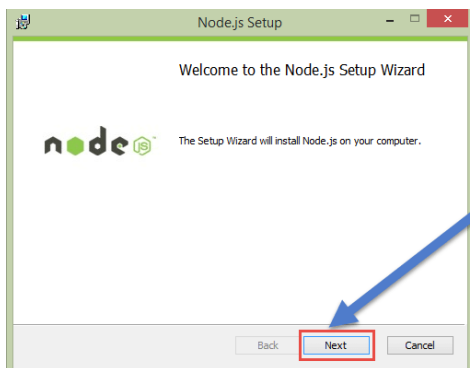
OS	Archive name
Windows	node-v6.3.1-x64.msi
Linux	node-v6.3.1-linux-x86.tar.gz
Mac	node-v6.3.1-darwin-x86.tar.gz
SunOS	node-v6.3.1-sunos-x86.tar.gz

Step 2) Double click on the downloaded .msi file to start the installation. Click the Run button in the first screen to begin the installation.



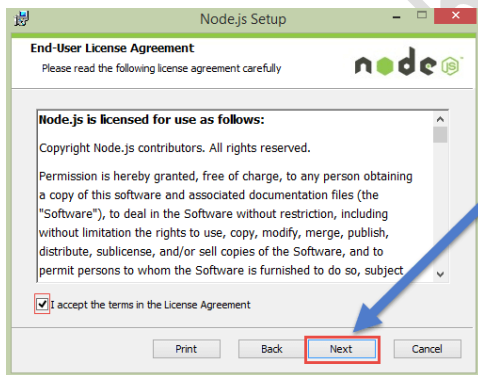
Click the  
Run button

**Step 3)** In the next screen, click the "Next" button to continue with the installation



Click the  
Next button

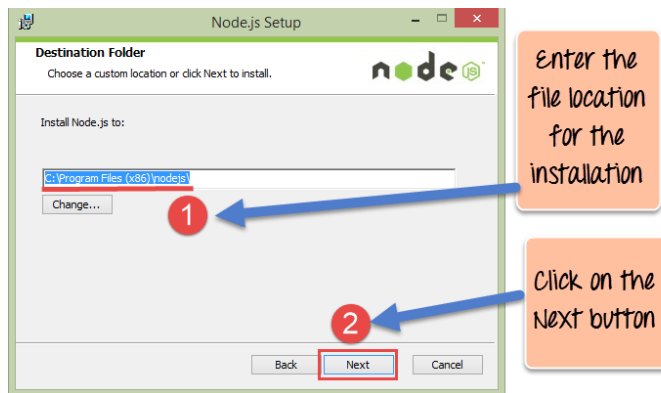
**Step 4)** In the next screen Accept the license agreement and click on the Next button.



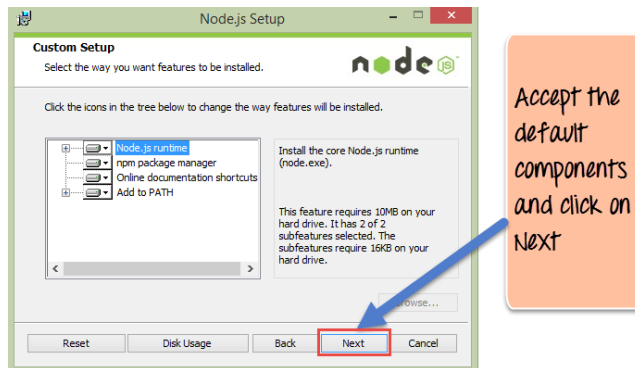
Accept the  
license  
agreement  
and click  
the Next

**Step 5)** In the next screen, choose the location where Node.js needs to be installed and then click on the Next button.

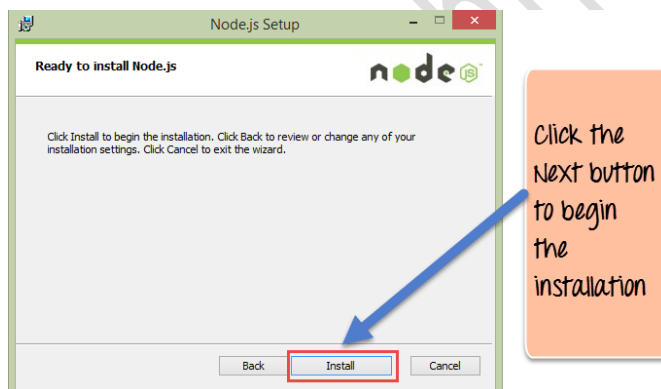




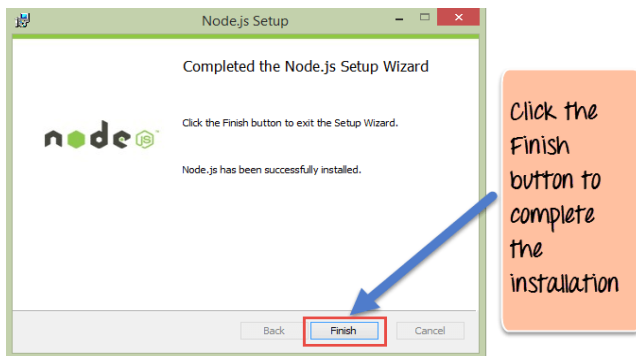
**Step 6)** Accept the default components and click on the next button.



**Step 7)** In the next screen, click the Install button to start the installation.



**Step 8)** Click the Finish button to complete the installation.



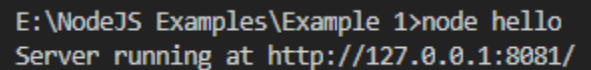
## App Hello World

Create index.html and hello.js file  
Add HTTP Server Listen code in to that.  
Run app from console / terminal window.  
Index.html:

```
<script src="hello.js"></script>
```

hello.js:

```
var http = require("http");  
http.createServer(function (request,  
response) {  
  response.writeHead(200, {'Content-Type': 'text/html'});  
  response.end('<h1>Hello World\n</h1>');  
}).listen(8081);  
console.log('Server running at http://127.0.0.1:8081/');
```



## What are Modules in Node Js?

Node js are a way of encapsulating code in a separate logical unit. There are many readymade modules available in the market which can be used within Node js. Below are some of the popular modules which are used in a Node js application  
Express framework - Express is a minimal and flexible Node js web application framework that provides a robust set of features for the web and Mobile applications.

Socket.io - Socket.IO enables real-time bidirectional event-based communication. This module is good for creation of chatting based applications.

Jade - Jade is a high-performance template engine and implemented with Javascript for node and browsers.

MongoDB - The MongoDB Node.js driver is the officially supported node.js driver for MongoDB.

Restify - restify is a lightweight framework, similar to express for building REST APIs

Bluebird - Bluebird is a fully featured promise library with focus on innovative features and performance

## Create NPM Module

**Step 1)** Create a file called "Addition.js" and include the below code. This file will contain the logic for your module.

```
1 var exports = module.exports={};  
  exports.AddNumber=function(a,b) 2  
  {  
    return a+b; 3  
  };
```

Defining a module

Creating a function in our module

Returning a value back to the calling function

**Step 2)** Create a file called "app.js," which is your main application file and add the below code

```
var Addition=require('./Addition.js'); 1  
console.log(Addition.AddNumber(1,2)); 2
```

Calling the AddNumber function in our module

Using require to include the Addition module

## When use Node JS?

- Chat applications
- Game servers
- Good for collaborative environment
- Advertisement servers
- Streaming servers

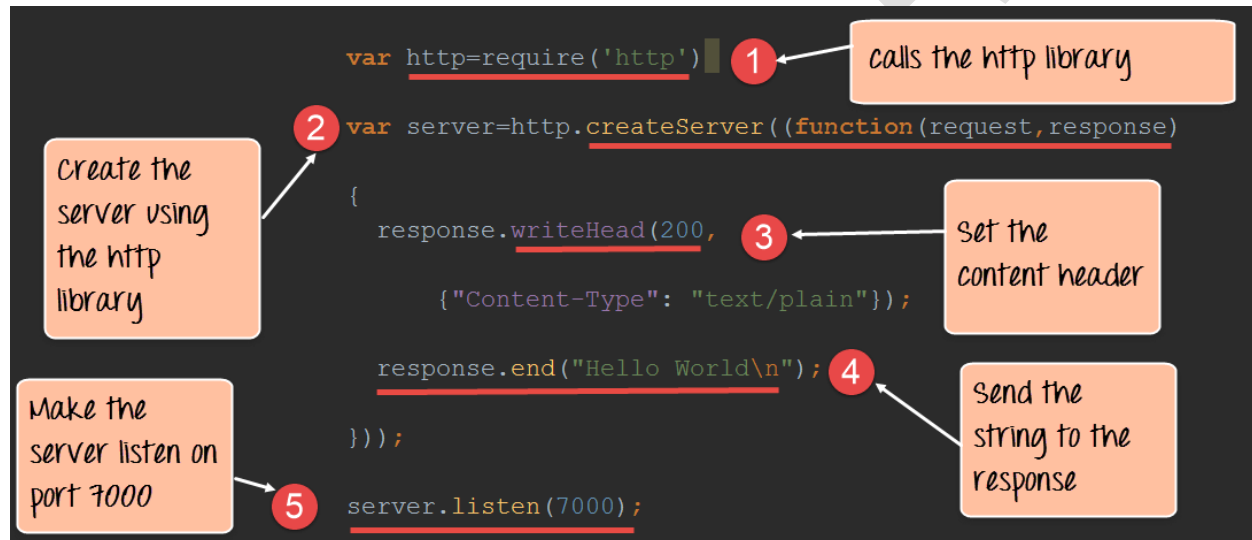
## Create Webserver in Node JS

The Node.js framework is mostly used to create server based applications. The framework can easily be used to create web servers which can serve content to users.

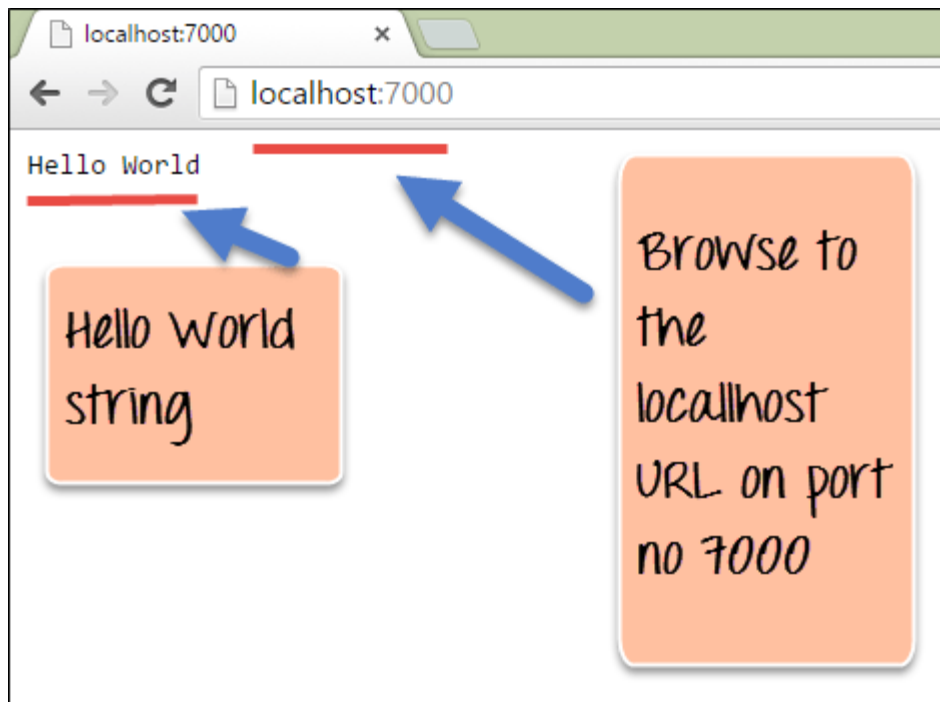
There are a variety of modules such as the "http" and "request" module, which helps in processing server related requests in the web server space. We will have a look at how we can create a basic web server application using Node js.

## How Webserver Make?

Our application is going to create a simple server module which will listen on port no 7000. If a request is made through the browser on this port no, then server application will send a 'Hello' World' response to the client.



OutPut

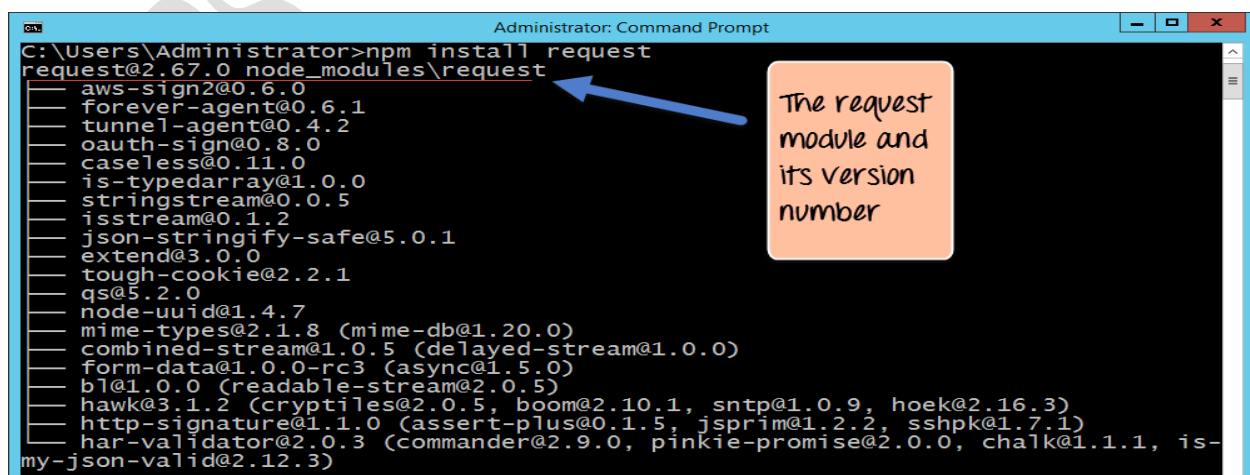


## Handling get Request in Node

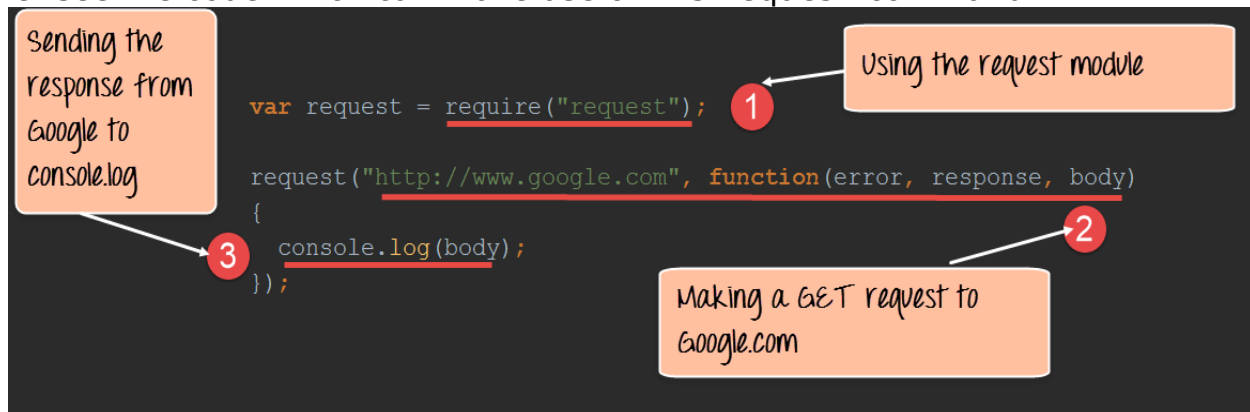
Making a GET Request to get the data from another site is relatively very simple in Node.js. To make a Get request in the node, we need to first have the request module installed. This can be done by executing the following line in the command line.

### Npm install request

When your npm module has been installed successfully, the command line will show the installed module name and version: <name>@<version>.



Now let's Handling get Request in Node  
let see the code which can make use of this 'request' command.



The Node.js framework can be used to develop web servers using the 'http' module. The application can be made to listen on a particular port and send a response to the client whenever a request is made to the application. The 'request' module can be used to get information from web sites. The information would contain the entire content of the web page requested from the relevant web site.

## What is Callback?

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

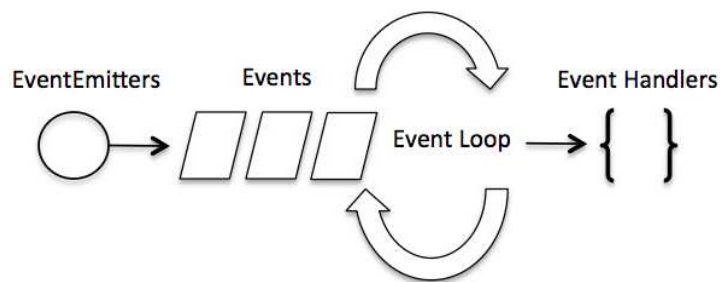
## Event Loop

- Node.js is a single-threaded application, but it can support concurrency via the concept of event and callbacks.
- Every API of Node.js is asynchronous and being single-threaded, they use async function calls to maintain concurrency.

- Node uses observer pattern. Node thread keeps an event loop and whenever a task gets completed, it fires the corresponding event which signals the event-listener function to execute.

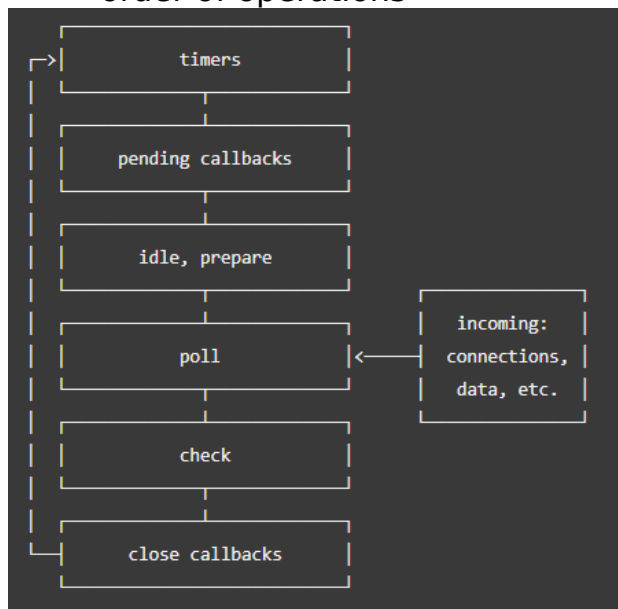
### Event-Driven Programming

- Node.js uses events heavily and it is also one of the reasons why Node.js is pretty fast compared to other similar technologies.
- As soon as Node starts its server, it simply initiates its variables, declares functions and then simply waits for the event to occur.
- Although events look quite similar to callbacks, the difference lies in the fact that callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern.
- The functions that listen to events act as Observers. Whenever an event gets fired, its listener function starts executing.
- Node.js has multiple in-built events available through events module and EventEmitter class which are used to bind events and event-listeners



## Event Loop Explained

- When Node.js starts, it initializes the event loop,
- processes the provided input script (or drops into the REPL, which is not covered in this document) which may make async API calls, schedule timers, or call `process.nextTick()`, then begins processing the event loop.
- The following diagram shows a simplified overview of the event loop's order of operations



- Each phase has a FIFO queue of callbacks to execute.
- While each phase is special in its own way, generally, when the event loop enters a given phase, it will perform any operations specific to that phase, then execute callbacks in that phase's queue until the queue has been exhausted or the maximum number of callbacks has executed.
- When the queue has been exhausted or the callback limit is reached, the event loop will move to the next phase, and so on.
- Since any of these operations may schedule more operations and new events processed in the poll phase are queued by the kernel, poll events can be queued while polling events are being processed.
- As a result, long running callbacks can allow the poll phase to run much longer than a timer's threshold. See the timers and poll sections for more details.

### Phases Overview

- **timers:** this phase executes callbacks scheduled by `setTimeout()` and `setInterval()`.



- pending callbacks: executes I/O callbacks deferred to the next loop iteration.
- idle, prepare: only used internally.
- poll: retrieve new I/O events; execute I/O related callbacks (almost all with the exception of close callbacks, the ones scheduled by timers, and setImmediate()); node will block here when appropriate.
- check: setImmediate() callbacks are invoked here.
- close callbacks: some close callbacks, e.g. socket.on('close', ...).

### EventEmitter Class

- As we have seen in the previous section, EventEmitter class lies in the events module. It is accessible via the following code –  

```
var events = require('events');  
var EventEmitter = new events.EventEmitter();
```
- 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.

Sr.No.	Method & Description
1	<b>addListener(event, listener)</b> Adds a listener at the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. Multiple calls passing the same combination of event and listener will result in the listener being added multiple times. Returns emitter, so calls can be chained.
2	<b>on(event, listener)</b> Adds a listener at the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. Multiple calls passing the same combination of event and listener will result in the listener being added multiple times. Returns emitter, so calls can be chained.
3	<b>once(event, listener)</b> Adds a one time listener to the event. This listener is invoked only the next time the event is fired, after which it is removed. Returns emitter, so calls can be chained.
4	<b>removeListener(event, listener)</b> Removes a listener from the listener array for the specified event. <b>Caution</b> – It changes the array indices in the listener array behind the listener. removeListener will remove, at most, one instance of a listener from the listener array. If any single listener has been added multiple times to the listener array for the specified event, then removeListener must be called multiple times to remove each instance. Returns emitter, so calls can be chained.

5	<b>removeAllListeners([event])</b> Removes all listeners, or those of the specified event. It's not a good idea to remove listeners that were added elsewhere in the code, especially when it's on an emitter that you didn't create (e.g. sockets or file streams). Returns emitter, so calls can be chained.
6	<b>setMaxListeners(n)</b> By default, EventEmitters will print a warning if more than 10 listeners are added for a particular event. This is a useful default which helps finding memory leaks. Obviously not all Emitters should be limited to 10. This function allows that to be increased. Set to zero for unlimited.
7	<b>listeners(event)</b> Returns an array of listeners for the specified event.
8	<b>emit(event, [arg1], [arg2], [...])</b> Execute each of the listeners in order with the supplied arguments. Returns true if the event had listeners, false otherwise.

## Buffer

- Pure JavaScript is Unicode friendly, but it is not so for binary data.
- While dealing with TCP streams or the file system, it's necessary to handle octet streams.
- Node provides Buffer class which provides instances to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap.
- Buffer class is a global class that can be accessed in an application without importing the buffer module.

### Writing to Buffers

**buf.write(string[, offset][, length][, encoding])**

#### Parameters

Here is the description of the parameters used –

- string – This is the string data to be written to buffer.
  - offset – This is the index of the buffer to start writing at. Default value is 0.
  - length – This is the number of bytes to write. Defaults to buffer.length.
  - encoding – Encoding to use. 'utf8' is the default encoding.
- Find **example write buffer here**

## What is Express Js

Express.js is a Node js web application server framework, which is specifically designed for building single-page, multi-page, and hybrid web applications.

## Why Express Js

It has become the standard server framework for node.js. Express is the backend part of something known as the MEAN stack.

The **MEAN** is a free and open-source Javascript software stack for building dynamic web sites and web applications which has the following components:

- **MongoDB** - The standard NoSQL database
- **Express.js** - The default web applications framework
- **Angular.js** - The JavaScript MVC framework used for web applications
- **Node.js** - Framework used for scalable server-side and networking applications.

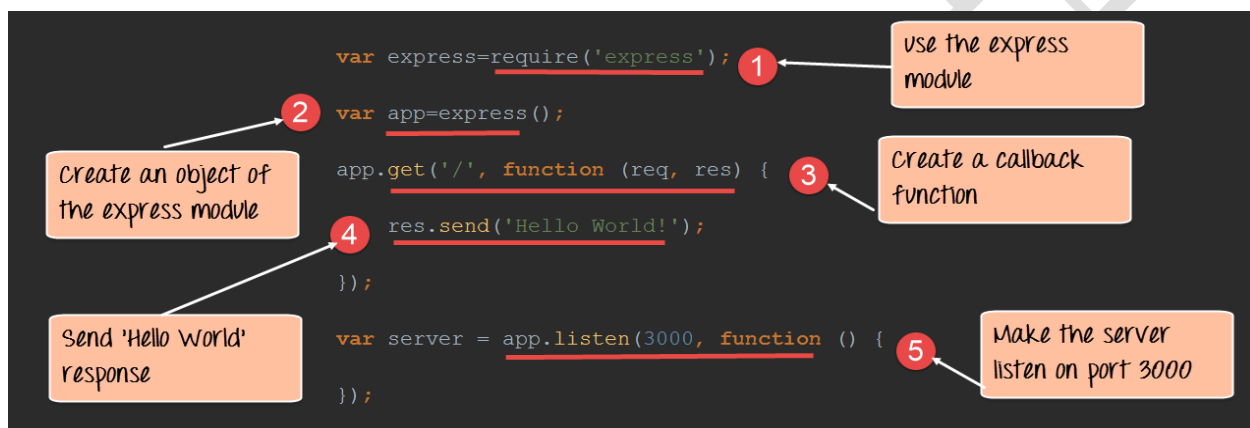
The Express.js framework makes it very easy to develop an application which can be used to handle multiple types of requests like the GET, PUT, and POST and DELETE requests.

## How Express Js

Express gets installed via the Node Package manager. This can be done by executing the following line in the command line

### npm install express

Our application is going to create a simple server module which will listen on port no 3000. In our example, if a request is made through the browser on this port no, then server application will send a 'Hello' World' response to the client.



## What are Routes?

Routing refers for determining the way in which an application responds to a client request to a particular endpoint

A client can make a GET, POST, PUT or DELETE http request for various URLs such as the ones shown below;

<http://localhost:3000/Books>

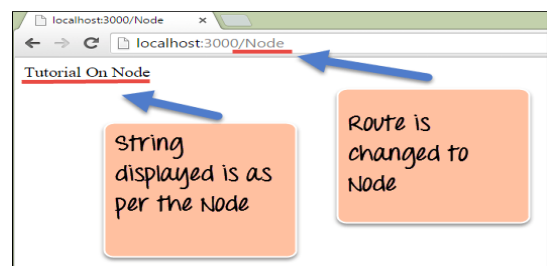
<http://localhost:3000/Students>

## Web Server Using Express.js

When creating quick on-the-fly Node applications, an easy and fast way is to use templates for the application. There are many frameworks available in the market for making templates. In our case, we will take the example of the jade framework for templating.

Jade gets installed via the Node Package manager. This can be done by executing the following line in the command line

### npm install jade



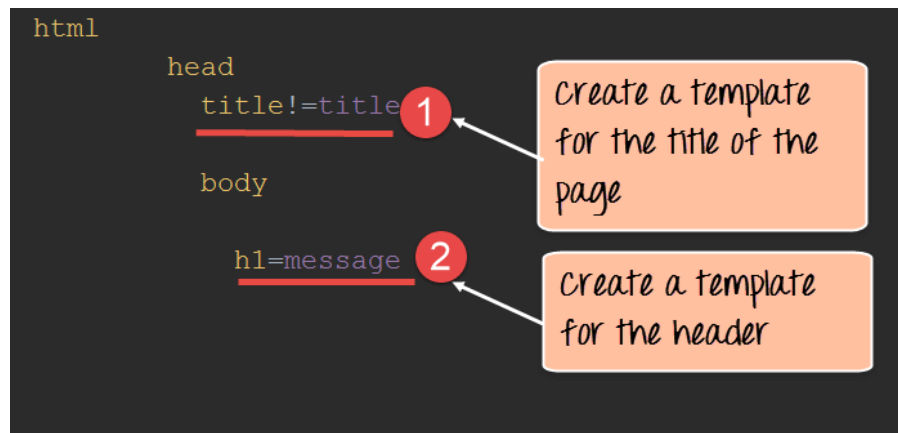
The above command requests the Node package manager to download the required jade modules and install them accordingly.

## Web Server Using Express.js

Let's use our newly installed jade framework and create some basic templates.

**Step 1)** The first step is to create a jade template. Create a file called index.jade and insert the below code

```
html
  head
    title!=title
  body
    h1=message
```

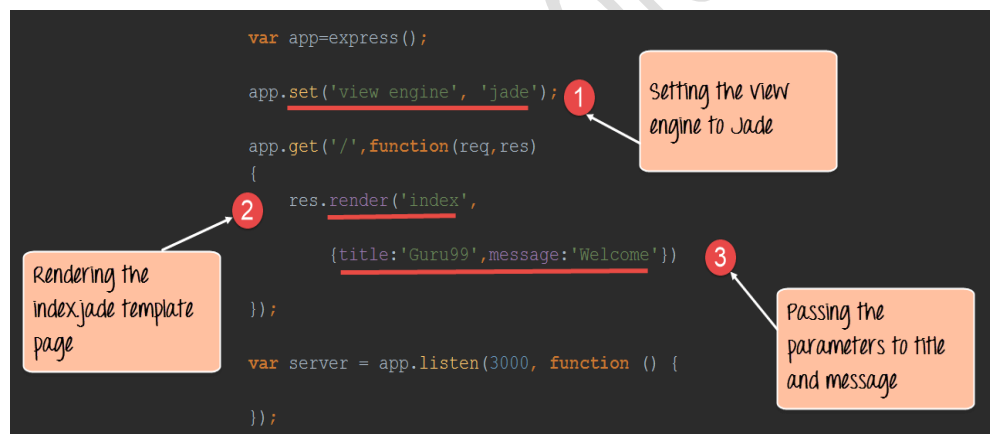


```
var app=express();

app.set('view engine', 'jade');

app.get('/',function(req,res)
{
  res.render('index',
    {title:'Guru99',message:'Welcome'})
});

var server = app.listen(3000, function () {
});
```



## What is Mongo DB?

The Node js framework has the ability to work with databases which are commonly required by most modern day web applications. Node js can work with both relational (such as Oracle and MS SQL Server) and non-relational databases (such as MongoDB).

## **Why Mongo DB?**

The ability of these databases to store any sort of content and particularly in any sort of format is what makes these databases so famous.

## **HOW use Mongo DB?**

TOPS Technologies Pvt.Ltd

The Node js framework has the ability to work with databases which are commonly required by most modern day web applications. Node js can work with both relational (such as Oracle and MS SQL Server) and non-relational databases (such as MongoDB).

For MySQL, the required module is called "mysql" and for using MongoDB the required module to be installed is "Mongoose."

With these modules, you can perform the following operations in Node.js

1. Manage the connection pooling – Here is where you can specify the number of MySQL database connections that should be maintained and saved by Node.js.
2. Create and close a connection to a database. In either case, you can provide a callback function which can be called whenever the "create" and "close" connection methods are executed.
3. Queries can be executed to get data from respective databases to retrieve data.
4. Data manipulation such as inserting data, deleting and updating data can also be achieved with these modules.

How we can establish connections with a MongoDB database

How we can establish connections with a MongoDB database

1. **Installing the NPM Modules :** To access Mongo from within a Node application, a driver is required. There are number of Mongo drivers available, but MongoDB is among the most popular. To install the MongoDB module, run the below command

**npm install mongodb**

2. **Creating and closing a connection to a MongoDB database.** The below code snippet shows how to create and close a connection to a MongoDB database.

3. **Querying for data in a MongoDB database** – Using the MongoDB driver we can also fetch data from the MongoDB database.

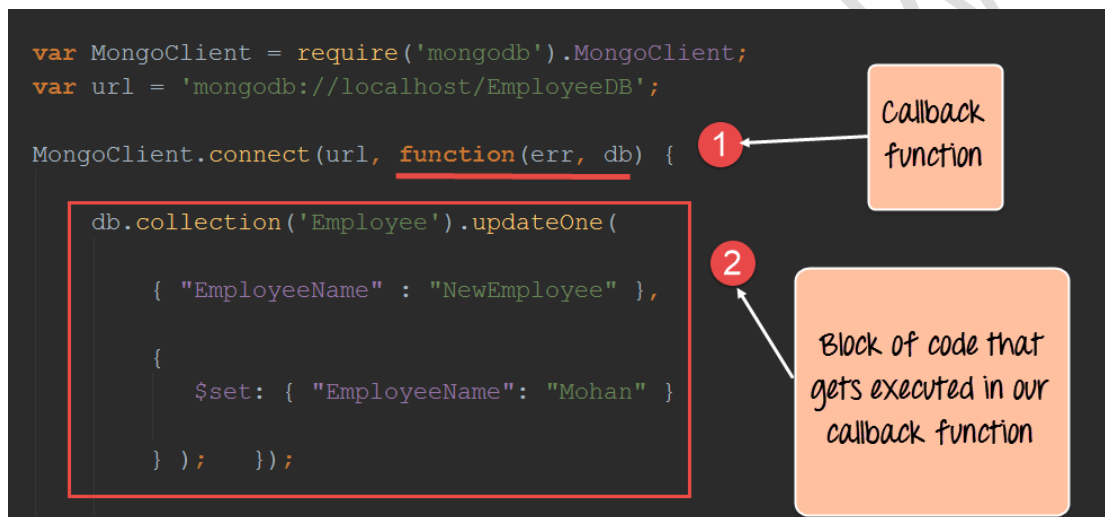
4. **Inserting documents in a collection** – Documents can be inserted into a collection using the insertOne method provided by the MongoDB library. The below code snippet shows how we can insert a document into a mongoDB collection.

. **Updating documents in a collection** - Documents can be updated in a collection using the updateOne method provided by the MongoDB library. The below code snippet shows how to update a document in a mongoDB collection.

6. **Deleting documents in a collection** - Documents can be deleted in a collection using the "deleteOne" method provided by the MongoDB library. The below code snippet shows how to delete a document in a mongoDB collection.

### What is Promises?

Before we start with promises, let's first revisit what are "callback" functions in Node.js. We have seen these callback functions a lot in the previous chapters, so let's quickly go through one of them.



### Create Custom Promise.

A custom promise can be created by using a node module called 'q.' The 'q' library needs to be downloaded and installed using the node package manager. After using the 'q' library, the method "denodeify" can be called which will cause any function to become a function which returns a promise

Let's follow the below steps to creating our custom function to return a promise.

**Step 1)** Installing the NPM Modules

**npm install q**

**Step 2)** Define the following code which will be used to create the custom promise.

**Step 2)** Define the following code which will be used to create the custom promise.



