Node.js Tutorial

[❮ Home](https://www.w3schools.com/default.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_intro.asp)

Node.js is an open source server environment.

Node.js allows you to run JavaScript on the server.

[Start learning Node.js now »](https://www.w3schools.com/nodejs/nodejs_intro.asp)

Learning by Examples

Our "Show Node.js" tool makes it easy to learn Node.js, it shows both the code and the result.

Example

var http = require('http');  
  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/plain'});  
  res.end('Hello World!');  
}).listen(8080);

[Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_intro)

**Click on the "Run example" button to see how it works.**

Examples Running in the Command Line Interface

In this tutorial there will be some examples that are better explained by displaying the result in the command line interface.

When this happens, The "Show Node.js" tool will show the result in a black screen on the right:

Example

console.log('This example is different!');  
console.log('The result is displayed in the Command Line Interface');

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_intro_cmd)

**Click on the "Run example" button to see how it works.**

Node.js Reference

Node.js has a set of built-in modules.

[Built-in Modules](https://www.w3schools.com/nodejs/ref_modules.asp)

Download Node.js

Download Node.js from the official Node.js web site: [https://nodejs.org](https://nodejs.org/)

Node.js Introduction

[❮ Previous](https://www.w3schools.com/nodejs/default.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_get_started.asp)

What is Node.js?

* Node.js is an open source server environment
* Node.js is free
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* Node.js uses JavaScript on the server

Why Node.js?

**Node.js uses asynchronous programming!**

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

What Can Node.js Do?

* Node.js can generate dynamic page content
* Node.js can create, open, read, write, delete, and close files on the server
* Node.js can collect form data
* Node.js can add, delete, modify data in your database

What is a Node.js File?

* Node.js files contain tasks that will be executed on certain events
* A typical event is someone trying to access a port on the server
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"
* Node.js Get Started
* [❮ Previous](https://www.w3schools.com/nodejs/nodejs_intro.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_modules.asp)
* Download Node.js
* The official Node.js website has installation instructions for Node.js: [https://nodejs.org](https://nodejs.org/)
* Getting Started
* Once you have downloaded and installed Node.js on your computer, let's try to display "Hello World" in a web browser.
* Create a Node.js file named "myfirst.js", and add the following code:
* myfirst.js
* var http = require('http');  
    
  http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.end('Hello World!');  
  }).listen(8080);
* Save the file on your computer: C:\Users\*Your Name*\myfirst.js
* The code tells the computer to write "Hello World!" if anyone (e.g. a web browser) tries to access your computer on port 8080.
* For now, you do not have to understand the code. It will be explained later.
* Command Line Interface
* Node.js files must be initiated in the "Command Line Interface" program of your computer.
* How to open the command line interface on your computer depends on the operating system. For Windows users, press the start button and look for "Command Prompt", or simply write "cmd" in the search field.
* Navigate to the folder that contains the file "myfirst.js", the command line interface window should look something like this:
* C:\Users\*Your Name*>\_
* Initiate the Node.js File
* The file you have just created must be initiated by Node.js before any action can take place.
* Start your command line interface, write node myfirst.js and hit enter:
* Initiate "myfirst.js":
* C:\Users\*Your Name*>node myfirst.js
* Now, your computer works as a server!
* If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!
* Start your internet browser, and type in the address: [http://localhost:8080](http://localhost:8080/)
* Node.js Modules
* [❮ Previous](https://www.w3schools.com/nodejs/nodejs_get_started.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_http.asp)
* What is a Module in Node.js?
* Consider modules to be the same as JavaScript libraries.
* A set of functions you want to include in your application.
* Built-in Modules
* Node.js has a set of built-in modules which you can use without any further installation.
* Look at our [Built-in Modules Reference](https://www.w3schools.com/nodejs/ref_modules.asp) for a complete list of modules.
* Include Modules
* To include a module, use the require() function with the name of the module:
* var http = require('http');
* Now your application has access to the HTTP module, and is able to create a server:
* http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.end('Hello World!');  
  }).listen(8080);
* Create Your Own Modules
* You can create your own modules, and easily include them in your applications.
* The following example creates a module that returns a date and time object:
* Example
* Create a module that returns the current date and time:
* exports.myDateTime = function () {  
    return Date();  
  };
* Use the exports keyword to make properties and methods available outside the module file.
* Save the code above in a file called "myfirstmodule.js"
* Include Your Own Module
* Now you can include and use the module in any of your Node.js files.
* Example
* Use the module "myfirstmodule" in a Node.js file:
* var http = require('http');  
  **var dt = require('./myfirstmodule');**  
  http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write("The date and time are currently: " + **dt.myDateTime()**);  
    res.end();  
  }).listen(8080);
* [Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_module)
* Notice that we use ./ to locate the module, that means that the module is located in the same folder as the Node.js file.
* Save the code above in a file called "demo\_module.js", and initiate the file:
* Initiate demo\_module.js:
* C:\Users\*Your Name*>node demo\_module.js
* If you have followed the same steps on your computer, you will see the same result as the example: [http://localhost:8080](http://localhost:8080/)
* Node.js HTTP Module
* [❮ Previous](https://www.w3schools.com/nodejs/nodejs_modules.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_filesystem.asp)
* The Built-in HTTP Module
* Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).
* To include the HTTP module, use the require() method:
* var http = require('http');
* Node.js as a Web Server
* The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
* Use the createServer() method to create an HTTP server:
* Example
* var http = require('http');  
    
  //create a server object:  
  http.createServer(function (req, res) {  
    res.write('Hello World!'); //write a response to the client  
    res.end(); //end the response  
  }).listen(8080); //the server object listens on port 8080
* [Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_http)
* The function passed into the http.createServer() method, will be executed when someone tries to access the computer on port 8080.
* Save the code above in a file called "demo\_http.js", and initiate the file:
* Initiate demo\_http.js:
* C:\Users\*Your Name*>node demo\_http.js
* If you have followed the same steps on your computer, you will see the same result as the example: [http://localhost:8080](http://localhost:8080/)
* Add an HTTP Header
* If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:
* Example
* var http = require('http');  
  http.createServer(function (req, res) {  
  **res.writeHead(200, {'Content-Type': 'text/html'});**  res.write('Hello World!');  
    res.end();  
  }).listen(8080);
* [Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_http_header)
* The first argument of the res.writeHead() method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.
* Read the Query String
* The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).
* This object has a property called "url" which holds the part of the url that comes after the domain name:
* demo\_http\_url.js
* var http = require('http');  
  http.createServer(function (**req**, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write(**req.url**);  
    res.end();  
  }).listen(8080);
* Save the code above in a file called "demo\_http\_url.js" and initiate the file:
* Initiate demo\_http\_url.js:
* C:\Users\*Your Name*>node demo\_http\_url.js
* If you have followed the same steps on your computer, you should see two different results when opening these two addresses:
* <http://localhost:8080/summer>
* Will produce this result:
* /summer
* [Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_querystring_summer)
* <http://localhost:8080/winter>
* Will produce this result:
* /winter
* [Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_querystring_winter)
* Split the Query String
* There are built-in modules to easily split the query string into readable parts, such as the URL module.
* Example
* Split the query string into readable parts:
* var http = require('http');  
  **var url = require('url');**  
  http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
  **var q = url.parse(req.url, true).query;**  var txt = **q.year** + " " + **q.month**;  
    res.end(txt);  
  }).listen(8080);
* Save the code above in a file called "demo\_querystring.js" and initiate the file:
* Initiate demo\_querystring.js:
* C:\Users\*Your Name*>node demo\_querystring.js
* The address:
* <http://localhost:8080/?year=2017&month=July>
* Will produce this result:
* 2017 July
* [Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_querystring_query)
* Read more about the URL module in the [Node.js URL Module](https://www.w3schools.com/nodejs/nodejs_url.asp) chapter.

Node.js File System Module

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_http.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_url.asp)

Node.js as a File Server

The Node.js file system module allows you to work with the file system on your computer.

To include the File System module, use the require() method:

var fs = require('fs');

Common use for the File System module:

* Read files
* Create files
* Update files
* Delete files
* Rename files

Read Files

The fs.readFile() method is used to read files on your computer.

Assume we have the following HTML file (located in the same folder as Node.js):

demofile1.html

<html>  
<body>  
<h1>My Header</h1>  
<p>My paragraph.</p>  
</body>  
</html>

Create a Node.js file that reads the HTML file, and return the content:

Example

var http = require('http');  
var fs = require('fs');  
http.createServer(function (req, res) {  
**fs.readFile('demofile1.html', function(err, data) {**    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write(data);  
    return res.end();  
  });  
}).listen(8080);

[Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_readfile)

Save the code above in a file called "demo\_readfile.js", and initiate the file:

Initiate demo\_readfile.js:

C:\Users\*Your Name*>node demo\_readfile.js

If you have followed the same steps on your computer, you will see the same result as the example: [http://localhost:8080](http://localhost:8080/)

Create Files

The File System module has methods for creating new files:

* fs.appendFile()
* fs.open()
* fs.writeFile()

The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created:

Example

Create a new file using the appendFile() method:

var fs = require('fs');  
  
fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {  
  if (err) throw err;  
  console.log('Saved!');  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_fs_append)

The fs.open() method takes a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created:

Example

Create a new, empty file using the open() method:

var fs = require('fs');  
  
fs.open('mynewfile2.txt', 'w', function (err, file) {  
  if (err) throw err;  
  console.log('Saved!');  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_fs_open)

The fs.writeFile() method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

Example

Create a new file using the writeFile() method:

var fs = require('fs');  
  
fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {  
  if (err) throw err;  
  console.log('Saved!');  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_fs_writefile)

Update Files

The File System module has methods for updating files:

* fs.appendFile()
* fs.writeFile()

The fs.appendFile() method appends the specified content at the end of the specified file:

Example

Append "This is my text." to the end of the file "mynewfile1.txt":

var fs = require('fs');  
  
fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {  
  if (err) throw err;  
  console.log('Updated!');  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_fs_update)

The fs.writeFile() method replaces the specified file and content:

Example

Replace the content of the file "mynewfile3.txt":

var fs = require('fs');  
  
fs.writeFile('mynewfile3.txt', 'This is my text', function (err) {  
  if (err) throw err;  
  console.log('Replaced!');  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_fs_replace)

Delete Files

To delete a file with the File System module,  use the fs.unlink() method.

The fs.unlink() method deletes the specified file:

Example

Delete "mynewfile2.txt":

var fs = require('fs');  
  
fs.unlink('mynewfile2.txt', function (err) {  
  if (err) throw err;  
  console.log('File deleted!');  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_fs_unlink)

Rename Files

To rename a file with the File System module,  use the fs.rename() method.

The fs.rename() method renames the specified file:

Example

Rename "mynewfile1.txt" to "myrenamedfile.txt":

var fs = require('fs');  
  
fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {  
  if (err) throw err;  
  console.log('File Renamed!');  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_fs_rename)

Upload Files

You can also use Node.js to upload files to your computer.

Read how in our [Node.js Upload Files](https://www.w3schools.com/nodejs/nodejs_uploadfiles.asp) chapter.

Node.js URL Module

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_filesystem.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_npm.asp)

The Built-in URL Module

The URL module splits up a web address into readable parts.

To include the URL module, use the require() method:

var url = require('url');

Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

Example

Split a web address into readable parts:

var url = require('url');  
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';  
var q = url.parse(adr, true);  
  
console.log(q.host); //returns 'localhost:8080'  
console.log(q.pathname); //returns '/default.htm'  
console.log(q.search); //returns '?year=2017&month=february'  
  
var qdata = q.query; //returns an object: { year: 2017, month: 'february' }  
console.log(qdata.month); //returns 'february'

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_url)

Node.js File Server

Now we know how to parse the query string, and in the previous chapter we learned how to make Node.js behave as a file server. Let us combine the two, and serve the file requested by the client.

Create two html files and save them in the same folder as your node.js files.

summer.html

<!DOCTYPE html>  
<html>  
<body>  
<h1>Summer</h1>  
<p>I love the sun!</p>  
</body>  
</html>

winter.html

<!DOCTYPE html>  
<html>  
<body>  
<h1>Winter</h1>  
<p>I love the snow!</p>  
</body>  
</html>

Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error:

demo\_fileserver.js:

var http = require('http');  
var url = require('url');  
var fs = require('fs');  
  
http.createServer(function (req, res) {  
  var q = url.parse(req.url, true);  
  var filename = "." + q.pathname;  
  fs.readFile(filename, function(err, data) {  
    if (err) {  
      res.writeHead(404, {'Content-Type': 'text/html'});  
      return res.end("404 Not Found");  
    }   
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write(data);  
    return res.end();  
  });  
}).listen(8080);

Remember to initiate the file:

Initiate demo\_fileserver.js:

C:\Users\*Your Name*>node demo\_fileserver.js

If you have followed the same steps on your computer, you should see two different results when opening these two addresses:

<http://localhost:8080/summer.html>

Will produce this result:

Summer

I love the sun!

<http://localhost:8080/winter.html>

Will produce this result:

Winter

I love the snow!

Node.js NPM

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_url.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_events.asp)

What is NPM?

NPM is a package manager for Node.js packages, or modules if you like.

[www.npmjs.com](https://www.npmjs.com/) hosts thousands of free packages to download and use.

The NPM program is installed on your computer when you install Node.js

NPM is already ready to run on your computer!

What is a Package?

A package in Node.js contains all the files you need for a module.

Modules are JavaScript libraries you can include in your project.

Download a Package

Downloading a package is very easy.

Open the command line interface and tell NPM to download the package you want.

I want to download a package called "upper-case":

Download "upper-case":

C:\Users\*Your Name*>npm install upper-case

Now you have downloaded and installed your first package!

NPM creates a folder named "node\_modules", where the package will be placed. All packages you install in the future will be placed in this folder.

My project now has a folder structure like this:

C:\Users\*My Name*\node\_modules\upper-case

Using a Package

Once the package is installed, it is ready to use.

Include the "upper-case" package the same way you include any other module:

var uc = require('upper-case');

Create a Node.js file that will convert the output "Hello World!" into upper-case letters:

Example

var http = require('http');  
var uc = require('upper-case');  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write(**uc.upperCase("Hello World!")**);  
  res.end();  
}).listen(8080);

[Run example »](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_uppercase)

Save the code above in a file called "demo\_uppercase.js", and initiate the file:

Initiate demo\_uppercase:

C:\Users\*Your Name*>node demo\_uppercase.js

If you have followed the same steps on your computer, you will see the same result as the example: [http://localhost:8080](http://localhost:8080/)

Node.js Events

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_npm.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_uploadfiles.asp)

Node.js is perfect for event-driven applications.

Events in Node.js

Every action on a computer is an event. Like when a connection is made or a file is opened.

Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file:

Example

var fs = require('fs');  
var rs = fs.createReadStream('./demofile.txt');  
rs.on('open', function () {  
  console.log('The file is open');  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_events_open)

Events Module

Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.

To include the built-in Events module use the require() method. In addition, all event properties and methods are an instance of an EventEmitter object. To be able to access these properties and methods, create an EventEmitter object:

var events = require('events');  
var eventEmitter = new events.EventEmitter();

The EventEmitter Object

You can assign event handlers to your own events with the EventEmitter object.

In the example below we have created a function that will be executed when a "scream" event is fired.

To fire an event, use the emit() method.

Example

var events = require('events');  
var eventEmitter = new events.EventEmitter();  
  
//Create an event handler:  
var myEventHandler = function () {  
  console.log('I hear a scream!');  
}  
  
//Assign the event handler to an event:  
eventEmitter.on('scream', myEventHandler);  
  
//Fire the 'scream' event:  
eventEmitter.emit('scream');

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_eventemitter)

Node.js Upload Files

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_events.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_email.asp)

The Formidable Module

There is a very good module for working with file uploads, called "Formidable".

The Formidable module can be downloaded and installed using NPM:

C:\Users\*Your Name*>npm install formidable

After you have downloaded the Formidable module, you can include the module in any application:

var formidable = require('formidable');

Upload Files

Now you are ready to make a web page in Node.js that lets the user upload files to your computer:

Step 1: Create an Upload Form

Create a Node.js file that writes an HTML form, with an upload field:

Example

This code will produce an HTML form:

var http = require('http');  
  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');  
  res.write('<input type="file" name="filetoupload"><br>');  
  res.write('<input type="submit">');  
  res.write('</form>');  
  return res.end();  
}).listen(8080);

Step 2: Parse the Uploaded File

Include the Formidable module to be able to parse the uploaded file once it reaches the server.

When the file is uploaded and parsed, it gets placed on a temporary folder on your computer.

Example

The file will be uploaded, and placed on a temporary folder:

var http = require('http');  
**var formidable = require('formidable');**  
http.createServer(function (req, res) {  
**if (req.url == '/fileupload') {  
    var form = new formidable.IncomingForm();  
    form.parse(req, function (err, fields, files) {  
      res.write('File uploaded');  
      res.end();  
    });**  } else {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');  
    res.write('<input type="file" name="filetoupload"><br>');  
    res.write('<input type="submit">');  
    res.write('</form>');  
    return res.end();  
  }  
}).listen(8080);

Step 3: Save the File

When a file is successfully uploaded to the server, it is placed on a temporary folder.

The path to this directory can be found in the "files" object, passed as the third argument in the parse() method's callback function.

To move the file to the folder of your choice, use the File System module, and rename the file:

Example

Include the fs module, and move the file to the current folder:

var http = require('http');  
var formidable = require('formidable');  
**var fs = require('fs');**  
http.createServer(function (req, res) {  
  if (req.url == '/fileupload') {  
    var form = new formidable.IncomingForm();  
    form.parse(req, function (err, fields, files) {  
**var oldpath = files.filetoupload.path;  
      var newpath = 'C:/Users/*Your Name*/' + files.filetoupload.name;  
      fs.rename(oldpath, newpath, function (err) {  
        if (err) throw err;  
        res.write('File uploaded and moved!');  
        res.end();  
      });** });  
  } else {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');  
    res.write('<input type="file" name="filetoupload"><br>');  
    res.write('<input type="submit">');  
    res.write('</form>');  
    return res.end();  
  }  
}).listen(8080);

Node.js Send an Email

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_uploadfiles.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql.asp)

The Nodemailer Module

The Nodemailer module makes it easy to send emails from your computer.

The Nodemailer module can be downloaded and installed using npm:

C:\Users\*Your Name*>npm install nodemailer

After you have downloaded the Nodemailer module, you can include the module in any application:

var nodemailer = require('nodemailer');

Send an Email

Now you are ready to send emails from your server.

Use the username and password from your selected email provider to send an email. This tutorial will show you how to use your Gmail account to send an email:

Example

var nodemailer = require('nodemailer');  
  
var transporter = nodemailer.createTransport({  
  service: 'gmail',  
  auth: {  
    user: '*youremail@gmail.com*',  
    pass: '*yourpassword*'  
  }  
});  
  
var mailOptions = {  
  from: '*youremail@gmail.com*',  
  to: '*myfriend@yahoo.com*',  
  subject: 'Sending Email using Node.js',  
  text: 'That was easy!'  
};  
  
transporter.sendMail(mailOptions, function(error, info){  
  if (error) {  
    console.log(error);  
  } else {  
    console.log('Email sent: ' + info.response);  
  }  
});

And that's it! Now your server is able to send emails.

Multiple Receivers

To send an email to more than one receiver, add them to the "to" property of the mailOptions object, separated by commas:

Example

Send email to more than one address:

var mailOptions = {  
  from: '*youremail*@gmail.com',  
**to:**'***myfriend@yahoo.com*, *myotherfriend@yahoo.com*',**  subject: 'Sending Email using Node.js',  
  text: 'That was easy!'  
}

Send HTML

To send HTML formatted text in your email, use the "html" property instead of the "text" property:

Example

Send email containing HTML:

var mailOptions = {  
  from: '*youremail*@gmail.com',  
  to: '*myfriend@yahoo.com*',  
  subject: 'Sending Email using Node.js',  
**html: '<h1>Welcome</h1><p>That was easy!</p>'**}

Node.js MySQL

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_email.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_create_db.asp)

Node.js can be used in database applications.

One of the most popular databases is MySQL.

MySQL Database

To be able to experiment with the code examples, you should have MySQL installed on your computer.

You can download a free MySQL database at <https://www.mysql.com/downloads/>.

Install MySQL Driver

Once you have MySQL up and running on your computer, you can access it by using Node.js.

To access a MySQL database with Node.js, you need a MySQL driver. This tutorial will use the "mysql" module, downloaded from NPM.

To download and install the "mysql" module, open the Command Terminal and execute the following:

C:\Users\*Your Name*>npm install mysql

Now you have downloaded and installed a mysql database driver.

Node.js can use this module to manipulate the MySQL database:

var mysql = require('mysql');

Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database.

demo\_db\_connection.js

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "*yourusername*",  
  password: "*yourpassword*"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_connection)

Save the code above in a file called "demo\_db\_connection.js" and run the file:

Run "demo\_db\_connection.js"

C:\Users\*Your Name*>node demo\_db\_connection.js

Which will give you this result:

Connected!

Now you can start querying the database using SQL statements.

Query a Database

Use SQL statements to read from (or write to) a MySQL database. This is also called "to query" the database.

The connection object created in the example above, has a method for querying the database:

con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
**con.query(*sql*, function (err, result) {  
    if (err) throw err;  
    console.log("Result: " + result);  
  });**});

The query method takes an sql statements as a parameter and returns the result.

Learn how to read, write, delete, and update a database in the next chapters.

Read more about SQL statements in our [SQL Tutorial](https://www.w3schools.com/sql/default.asp).

Node.js MySQL Create Database

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_create_table.asp)

Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

Example

Create a database named "mydb":

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "*yourusername*",  
  password: "*yourpassword*"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  con.query("**CREATE DATABASE mydb**", function (err, result) {  
    if (err) throw err;  
    console.log("Database created");  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_create_db)

Save the code above in a file called "demo\_create\_db.js" and run the file:

Run "demo\_create\_db.js"

C:\Users\*Your Name*>node demo\_create\_db.js

Which will give you this result:

Connected!  
Database created

Node.js MySQL Create Table

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_create_db.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_insert.asp)

Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection:

Example

Create a table named "customers":

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "*yourusername*",  
  password: "*yourpassword*",  
**database: "mydb"**  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  var sql = "**CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255)**)";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("Table created");  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_create_table)

Save the code above in a file called "demo\_create\_table.js" and run the file:

Run "demo\_create\_table.js"

C:\Users\*Your Name*>node demo\_create\_table.js

Which will give you this result:

Connected!  
Table created

Primary Key

When creating a table, you should also create a column with a unique key for each record.

This can be done by defining a column as "INT AUTO\_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

Example

Create primary key when creating the table:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "*yourusername*",  
  password: "*yourpassword*",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  var sql = "CREATE TABLE customers (**id INT AUTO\_INCREMENT PRIMARY KEY**, name VARCHAR(255), address VARCHAR(255))";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("Table created");  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_create_table_pk)

If the table already exists, use the ALTER TABLE keyword:

Example

Create primary key on an existing table:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "*yourusername*",  
  password: "*yourpassword*",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  var sql = "**ALTER TABLE customers ADD COLUMN id INT AUTO\_INCREMENT PRIMARY KEY**";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("Table altered");  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_alter_table_pk)

Node.js MySQL Insert Into

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_create_table.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_select.asp)

Insert Into Table

To fill a table in MySQL, use the "INSERT INTO" statement.

Example

Insert a record in the "customers" table:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "*yourusername*",  
  password: "*yourpassword*",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
**var sql = "INSERT INTO customers (name, address) VALUES ('Company Inc', 'Highway 37')";**  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("1 record inserted");  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_insert)

Save the code above in a file called "demo\_db\_insert.js", and run the file:

Run "demo\_db\_insert.js"

C:\Users\*Your Name*>node demo\_db\_insert.js

Which will give you this result:

Connected!  
1 record inserted

Insert Multiple Records

To insert more than one record, make an array containing the values, and insert a question mark in the sql, which will be replaced by the value array:  
INSERT INTO customers (name, address) VALUES ?

Example

Fill the "customers" table with data:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "*yourusername*",  
  password: "*yourpassword*",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  var sql = "INSERT INTO customers (name, address) VALUES ?";  
  var values = [  
    ['John', 'Highway 71'],  
    ['Peter', 'Lowstreet 4'],  
    ['Amy', 'Apple st 652'],  
    ['Hannah', 'Mountain 21'],  
    ['Michael', 'Valley 345'],  
    ['Sandy', 'Ocean blvd 2'],  
    ['Betty', 'Green Grass 1'],  
    ['Richard', 'Sky st 331'],  
    ['Susan', 'One way 98'],  
    ['Vicky', 'Yellow Garden 2'],  
    ['Ben', 'Park Lane 38'],  
    ['William', 'Central st 954'],  
    ['Chuck', 'Main Road 989'],  
    ['Viola', 'Sideway 1633']  
  ];  
  con.query(sql,**[values]**, function (err, result) {  
    if (err) throw err;  
    console.log("Number of records inserted: " + result.affectedRows);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_insert_multiple)

Save the code above in a file called "demo\_db\_insert\_multple.js", and run the file:

Run "demo\_db\_insert\_multiple.js"

C:\Users\*Your Name*>node demo\_db\_insert\_multiple.js

Which will give you this result:

Connected!  
Number of records inserted: 14

The Result Object

When executing a query, a result object is returned.

The result object contains information about how the query affected the table.

The result object returned from the example above looks like this:

{  
  fieldCount: 0,  
  affectedRows: 14,  
  insertId: 0,  
  serverStatus: 2,  
  warningCount: 0,  
  message: '\'Records:14  Duplicated: 0  Warnings: 0',  
  protocol41: true,  
  changedRows: 0  
}

The values of the properties can be displayed like this:

Example

Return the number of affected rows:

console.log(result.affectedRows)

Which will produce this result:

14

Get Inserted ID

For tables with an auto increment id field, you can get the id of the row you just inserted by asking the result object.

**Note:** To be able to get the inserted id, **only one row** can be inserted.

Example

Insert a record in the "customers" table, and return the ID:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "*yourusername*",  
  password: "*yourpassword*",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  var sql = "INSERT INTO customers (name, address) VALUES ('Michelle', 'Blue Village 1')";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("1 record inserted, ID: " + **result.insertId**);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_insert_id)

Save the code above in a file called "demo\_db\_insert\_id.js", and run the file:

Run "demo\_db\_insert\_id.js"

C:\Users\*Your Name*>node demo\_db\_insert\_id.js

Which will give you something like this in return:

1 record inserted, ID: 15

Node.js MySQL Select From

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_insert.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_where.asp)

Selecting From a Table

To select data from a table in MySQL, use the "SELECT" statement.

Example

Select all records from the "customers" table, and display the result object:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  con.query("**SELECT \* FROM customers**", function (err, result, fields) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_select)

**SELECT \*** will return *all* columns

Save the code above in a file called "demo\_db\_select.js" and run the file:

Run "demo\_db\_select.js"

C:\Users\*Your Name*>node demo\_db\_select.js

Which will give you this result:

[  
  { id: 1, name: 'John', address: 'Highway 71'},  
  { id: 2, name: 'Peter', address: 'Lowstreet 4'},  
  { id: 3, name: 'Amy', address: 'Apple st 652'},  
  { id: 4, name: 'Hannah', address: 'Mountain 21'},  
  { id: 5, name: 'Michael', address: 'Valley 345'},  
  { id: 6, name: 'Sandy', address: 'Ocean blvd 2'},  
  { id: 7, name: 'Betty', address: 'Green Grass 1'},  
  { id: 8, name: 'Richard', address: 'Sky st 331'},  
  { id: 9, name: 'Susan', address: 'One way 98'},  
  { id: 10, name: 'Vicky', address: 'Yellow Garden 2'},  
  { id: 11, name: 'Ben', address: 'Park Lane 38'},  
  { id: 12, name: 'William', address: 'Central st 954'},  
  { id: 13, name: 'Chuck', address: 'Main Road 989'},  
  { id: 14, name: 'Viola', address: 'Sideway 1633'}  
]

Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed by the column name.

Example

Select name and address from the "customers" table, and display the return object:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  con.query("**SELECT name, address FROM customers**", function (err, result, fields) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_select2)

Save the code above in a file called "demo\_db\_select2.js" and run the file:

Run "demo\_db\_select2.js"

C:\Users\*Your Name*>node demo\_db\_select2.js

Which will give you this result:

[  
  { name: 'John', address: 'Highway 71'},  
  { name: 'Peter', address: 'Lowstreet 4'},  
  { name: 'Amy', address: 'Apple st 652'},  
  { name: 'Hannah', address: 'Mountain 21'},  
  { name: 'Michael', address: 'Valley 345'},  
  { name: 'Sandy', address: 'Ocean blvd 2'},  
  { name: 'Betty', address: 'Green Grass 1'},  
  { name: 'Richard', address: 'Sky st 331'},  
  { name: 'Susan', address: 'One way 98'},  
  { name: 'Vicky', address: 'Yellow Garden 2'},  
  { name: 'Ben', address: 'Park Lane 38'},  
  { name: 'William', address: 'Central st 954'},  
  { name: 'Chuck', address: 'Main Road 989'},  
  { name: 'Viola', address: 'Sideway 1633'}  
]

The Result Object

As you can see from the result of the example above, the result object is an array containing each row as an object.

To return e.g. the address of the third record, just refer to the third array object's address property:

Example

Return the address of the third record:

console.log(result[2].address);

Which will produce this result:

Apple st 652

The Fields Object

The third parameter of the callback function is an array containing information about each field in the result.

Example

Select all records from the "customers" table, and display the *fields* object:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  con.query("SELECT name, address FROM customers", function (err, result, **fields**) {  
    if (err) throw err;  
    console.log(**fields**);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_select_fields)

Save the code above in a file called "demo\_db\_select\_fields.js" and run the file:

Run "demo\_db\_select\_fields.js"

C:\Users\*Your Name*>node demo\_db\_select\_fields.js

Which will give you this result:

[  
  {  
    catalog: 'def',  
    db: 'mydb',  
    table: 'customers',  
    orgTable: 'customers',  
    name: 'name',  
    orgName: 'address',  
    charsetNr: 33,  
    length: 765,  
    type: 253,  
    flags: 0,  
    decimals: 0,  
    default: undefined,  
    zeroFill: false,  
    protocol41: true  
  },  
  {  
    catalog: 'def',  
    db: 'mydb',  
    table: 'customers',  
    orgTable: 'customers',  
    name: 'address',  
    orgName: 'address',  
    charsetNr: 33,  
    length: 765,  
    type: 253,  
    flags: 0,  
    decimals: 0,  
    default: undefined,  
    zeroFill: false,  
    protocol41: true  
  {  
]

As you can see from the result of the example above, the fields object is an array containing information about each field as an object.

To return e.g. the name of the second field, just refer to the second array item's name property:

Example

Return the name of the second field:

console.log(fields[1].name);

Which will produce this result:

address

Node.js MySQL Where

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_select.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_orderby.asp)

Select With a Filter

When selecting records from a table, you can filter the selection by using the "WHERE" statement:

Example

Select record(s) with the address "Park Lane 38":

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  con.query("SELECT \* FROM customers **WHERE address = 'Park Lane 38'**", function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_where)

Save the code above in a file called "demo\_db\_where.js" and run the file:

Run "demo\_db\_where.js"

C:\Users\*Your Name*>node demo\_db\_where.js

Which will give you this result:

[  
  { id: 11, name: 'Ben', address: 'Park Lane 38'}  
]

Wildcard Characters

You can also select the records that starts, includes, or ends with a given letter or phrase.

Use the '%' wildcard to represent zero, one or multiple characters:

Example

Select records where the address starts with the letter 'S':

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  con.query("SELECT \* FROM customers WHERE address **LIKE 'S%'**", function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_where_s)

Save the code above in a file called "demo\_db\_where\_s.js" and run the file:

Run "demo\_db\_where\_s.js"

C:\Users\*Your Name*>node demo\_db\_where\_s.js

Which will give you this result:

[  
  { id: 8, name: 'Richard', address: 'Sky st 331'},  
  { id: 14, name: 'Viola', address: 'Sideway 1633'}  
]

Escaping Query Values

When query values are variables provided by the user, you should escape the values.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The MySQL module has methods to escape query values:

Example

Escape query values by using the mysql.escape() method:

var adr = 'Mountain 21';  
var sql = 'SELECT \* FROM customers WHERE address = ' + mysql.escape(adr);  
con.query(sql, function (err, result) {  
  if (err) throw err;  
  console.log(result);  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_where_escape)

You can also use a ? as a placeholder for the values you want to escape.

In this case, the variable is sent as the second parameter in the query() method:

Example

Escape query values by using the placeholder ? method:

var adr = 'Mountain 21';  
var sql = 'SELECT \* FROM customers WHERE address = ?';  
con.query(sql,**[adr]**, function (err, result) {  
  if (err) throw err;  
  console.log(result);  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_where_placeholder)

If you have multiple placeholders, the array contains multiple values, in that order:

Example

Multiple placeholders:

var name = 'Amy';  
var adr = 'Mountain 21';  
var sql = 'SELECT \* FROM customers WHERE name = ? OR address = ?';  
con.query(sql,**[name, adr]**, function (err, result) {  
  if (err) throw err;  
  console.log(result);  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_where_placeholders)

Node.js MySQL Order By

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_where.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_delete.asp)

Sort the Result

Use the ORDER BY statement to sort the result in ascending or descending order.

The ORDER BY keyword sorts the result ascending by default. To sort the result in descending order, use the DESC keyword.

Example

Sort the result alphabetically by name:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  con.query("SELECT \* FROM customers **ORDER BY name**", function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_orderby)

Save the code above in a file called "demo\_db\_orderby.js" and run the file:

Run "demo\_db\_orderby.js"

C:\Users\*Your Name*>node demo\_db\_orderby.js

Which will give you this result:

[  
  { id: 3, name: 'Amy', address: 'Apple st 652'},  
  { id: 11, name: 'Ben', address: 'Park Lane 38'},  
  { id: 7, name: 'Betty', address: 'Green Grass 1'},  
  { id: 13, name: 'Chuck', address: 'Main Road 989'},  
  { id: 4, name: 'Hannah', address: 'Mountain 21'},  
  { id: 1, name: 'John', address: 'Higheay 71'},  
  { id: 5, name: 'Michael', address: 'Valley 345'},  
  { id: 2, name: 'Peter', address: 'Lowstreet 4'},  
  { id: 8, name: 'Richard', address: 'Sky st 331'},  
  { id: 6, name: 'Sandy', address: 'Ocean blvd 2'},  
  { id: 9, name: 'Susan', address: 'One way 98'},  
  { id: 10, name: 'Vicky', address: 'Yellow Garden 2'},  
  { id: 14, name: 'Viola', address: 'Sideway 1633'},  
  { id: 12, name: 'William', address: 'Central st 954'}  
]

ORDER BY DESC

Use the DESC keyword to sort the result in a descending order.

Example

Sort the result reverse alphabetically by name:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  con.query("SELECT \* FROM customers **ORDER BY name DESC**", function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_orderby_desc)

Save the code above in a file called "demo\_db\_orderby\_desc.js" and run the file:

Run "demo\_db\_orderby\_desc.js"

C:\Users\*Your Name*>node demo\_db\_orderby\_desc.js

Which will give you this result:

[  
  { id: 12, name: 'William', address: 'Central st 954'},  
  { id: 14, name: 'Viola', address: 'Sideway 1633'},  
  { id: 10, name: 'Vicky', address: 'Yellow Garden 2'},  
  { id: 9, name: 'Susan', address: 'One way 98'},  
  { id: 6, name: 'Sandy', address: 'Ocean blvd 2'},  
  { id: 8, name: 'Richard', address: 'Sky st 331'},  
  { id: 2, name: 'Peter', address: 'Lowstreet 4'},  
  { id: 5, name: 'Michael', address: 'Valley 345'},  
  { id: 1, name: 'John', address: 'Higheay 71'},  
  { id: 4, name: 'Hannah', address: 'Mountain 21'},  
  { id: 13, name: 'Chuck', address: 'Main Road 989'},  
  { id: 7, name: 'Betty', address: 'Green Grass 1'},  
  { id: 11, name: 'Ben', address: 'Park Lane 38'},  
  { id: 3, name: 'Amy', address: 'Apple st 652'}  
]

Node.js MySQL Delete

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_orderby.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_drop_table.asp)

Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

Example

Delete any record with the address "Mountain 21":

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
**var sql = "DELETE FROM customers WHERE address = 'Mountain 21'";**  con.query(**sql,** function (err, result) {  
    if (err) throw err;  
    console.log("Number of records deleted: " + result.affectedRows);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_delete)

**Notice the WHERE clause in the DELETE syntax:** The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Save the code above in a file called "demo\_db\_delete.js" and run the file:

Run "demo\_db\_delete.js"

C:\Users\*Your Name*>node demo\_db\_delete.js

Which will give you this result:

Number of records deleted: 1

The Result Object

When executing a query, a result object is returned.

The result object contains information about how the query affected the table.

The result object returned from the example above looks like this:

{  
  fieldCount: 0,  
  affectedRows: 1,  
  insertId: 0,  
  serverStatus: 34,  
  warningCount: 0,  
  message: '',  
  protocol41: true,  
  changedRows: 0  
}

The values of the properties can be displayed like this:

Example

Return the number of affected rows:

console.log(result.affectedRows)

Which will produce this result:

1

Node.js MySQL Drop Table

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_delete.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_update.asp)

Delete a Table

You can delete an existing table by using the "DROP TABLE" statement:

Example

Delete the table "customers":

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
**var sql = "DROP TABLE customers";**  con.query(**sql,** function (err, result) {  
    if (err) throw err;  
    console.log("Table deleted");  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_drop)

Save the code above in a file called "demo\_db\_drop\_table.js" and run the file:

Run "demo\_db\_drop\_table.js"

C:\Users\*Your Name*>node demo\_db\_drop\_table.js

Which will give you this result:

Table deleted

Drop Only if Exist

If the the table you want to delete is already deleted, or for any other reason does not exist, you can use the IF EXISTS keyword to avoid getting an error.

Example

Delete the table "customers" if it exists:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
**var sql = "DROP TABLE IF EXISTS customers";**  con.query(**sql,** function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_drop_if)

Save the code above in a file called "demo\_db\_drop\_table\_if.js" and run the file:

Run "demo\_db\_drop\_table\_if.js"

C:\Users\*Your Name*>node demo\_db\_drop\_table\_if.js

If the table exist, the result object will look like this:

{  
  fieldCount: 0,  
  affectedRows: 0,  
  insertId: 0,  
  serverstatus: 2,  
  warningCount: 0,  
  message: '',  
  protocol41: true,  
  changedRows: 0  
}

If the table does not exist, the result object will look like this:

{  
  fieldCount: 0,  
  affectedRows: 0,  
  insertId: 0,  
  serverstatus: 2,  
  warningCount: 1,  
  message: '',  
  protocol41: true,  
  changedRows: 0  
}

As you can see the only differnce is that the warningCount property is set to 1 if the table does not exist.

Node.js MySQL Update

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_drop_table.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_limit.asp)

Update Table

You can update existing records in a table by using the "UPDATE" statement:

Example

Overwrite the address column from "Valley 345" to "Canyon 123":

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
**var sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'";**  con.query(**sql,** function (err, result) {  
    if (err) throw err;  
    console.log(result.affectedRows + " record(s) updated");  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_update)

**Notice the WHERE clause in the UPDATE syntax:** The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Save the code above in a file called "demo\_db\_update.js" and run the file:

Run "demo\_db\_update.js"

C:\Users\*Your Name*>node demo\_db\_update.js

Which will give you this result:

1 record(s) updated

The Result Object

When executing a query, a result object is returned.

The result object contains information about how the query affected the table.

The result object returned from the example above looks like this:

{  
  fieldCount: 0,  
  affectedRows: 1,  
  insertId: 0,  
  serverStatus: 34,  
  warningCount: 0,  
  message: '(Rows matched: 1 Changed: 1 Warnings: 0',  
  protocol41: true,  
  changedRows: 1  
}

The values of the properties can be displayed like this:

Example

Return the number of affected rows:

console.log(result.affectedRows)

Which will produce this result:

1

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_drop_table.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_limit.asp)

Node.js MySQL Limit

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_drop_table.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mysql_join.asp)

Limit the Result

You can limit the number of records returned from the query, by using the "LIMIT" statement:

Example

Select the 5 first records in the "customers" table:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  var sql = "SELECT \* FROM customers **LIMIT 5**";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_limit)

Save the code above in a file called "demo\_db\_limit.js" and run the file:

Run "demo\_db\_limit.js"

C:\Users\*Your Name*>node demo\_db\_limit.js

Which will give you this result:

[  
  { id: 1, name: 'John', address: 'Highway 71'},  
  { id: 2, name: 'Peter', address: 'Lowstreet 4'},  
  { id: 3, name: 'Amy', address: 'Apple st 652'},  
  { id: 4, name: 'Hannah', address: 'Mountain 21'},  
  { id: 5, name: 'Michael', address: 'Valley 345'}  
]

Start From Another Position

If you want to return five records, starting from the third record, you can use the "OFFSET" keyword:

Example

Start from position 3, and return the next 5 records:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  var sql = "SELECT \* FROM customers **LIMIT 5 OFFSET 2**";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_offset)

**Note:** "OFFSET 2", means starting from the third position, not the second!

Save the code above in a file called "demo\_db\_offset.js" and run the file:

Run "demo\_db\_offset.js"

C:\Users\*Your Name*>node demo\_db\_offset.js

Which will give you this result:

[  
  { id: 3, name: 'Amy', address: 'Apple st 652'},  
  { id: 4, name: 'Hannah', address: 'Mountain 21'},  
  { id: 5, name: 'Michael', address: 'Valley 345'},  
  { id: 6, name: 'Sandy', address: 'Ocean blvd 2'},  
  { id: 7, name: 'Betty', address: 'Green Grass 1'}  
]

Shorter Syntax

You can also write your SQL statement like this "LIMIT 2, 5" which returns the same as the offset example above:

Example

Start from position 3, and return the next 5 records:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  var sql = "SELECT \* FROM customers **LIMIT 2, 5**";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_limit_short)

**Note:** The numbers are reversed: "LIMIT 2, 5" is the same as "LIMIT 5 OFFSET 2"

Node.js MySQL Join

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_limit.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb.asp)

Join Two or More Tables

You can combine rows from two or more tables, based on a related column between them, by using a JOIN statement.

Consider you have a "users" table and a "products" table:

users

[  
  { id: 1, name: 'John', favorite\_product: 154},  
  { id: 2, name: 'Peter', favorite\_product: 154},  
  { id: 3, name: 'Amy', favorite\_product: 155},  
  { id: 4, name: 'Hannah', favorite\_product:},  
  { id: 5, name: 'Michael', favorite\_product:}  
]

products

[  
  { id: 154, name: 'Chocolate Heaven' },  
  { id: 155, name: 'Tasty Lemons' },  
  { id: 156, name: 'Vanilla Dreams' }  
]

These two tables can be combined by using users' favorite\_product field and products' id field.

Example

Select records with a match in both tables:

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword",  
  database: "mydb"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  var sql = "SELECT users.name AS user, products.name AS favorite FROM users **JOIN products ON users.favorite\_product = products.id**";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_join)

**Note:** You can use INNER JOIN instead of JOIN. They will both give you the same result.

Save the code above in a file called "demo\_db\_join.js" and run the file:

Run "demo\_db\_join.js"

C:\Users\*Your Name*>node demo\_db\_join.js

Which will give you this result:

[  
  { user: 'John', favorite: 'Chocolate Heaven' },  
  { user: 'Peter', favorite: 'Chocolate Heaven' },  
  { user: 'Amy', favorite: 'Tasty Lemons' }  
]

As you can see from the result above, only the records with a match in both tables are returned.

Left Join

If you want to return *all* users, no matter if they have a favorite product or not, use the LEFT JOIN statement:

Example

Select all users and their favorite product:

SELECT users.name AS user,  
products.name AS favorite  
FROM users **LEFT JOIN** products ON users.favorite\_product = products.id

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_left_join)

Which will give you this result:

[  
  { user: 'John', favorite: 'Chocolate Heaven' },  
  { user: 'Peter', favorite: 'Chocolate Heaven' },  
  { user: 'Amy', favorite: 'Tasty Lemons' },  
  { user: 'Hannah', favorite: null },  
  { user: 'Michael', favorite: null }  
]

Right Join

If you want to return all products, and the users who have them as their favorite, even if no user have them as their favorite, use the RIGHT JOIN statement:

Example

Select all products and the user who have them as their favorite:

SELECT users.name AS user,  
products.name AS favorite  
FROM users **RIGHT JOIN** products ON users.favorite\_product = products.id

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_db_right_join)

Which will give you this result:

[  
  { user: 'John', favorite: 'Chocolate Heaven' },  
  { user: 'Peter', favorite: 'Chocolate Heaven' },  
  { user: 'Amy', favorite: 'Tasty Lemons' },  
  { user: null, favorite: 'Vanilla Dreams' }  
]

**Note:** Hannah and Michael, who have no favorite product, are not included in the result.

Node.js MongoDB

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mysql_join.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_create_db.asp)

Node.js can be used in database applications.

One of the most popular NoSQL database is MongoDB.

MongoDB

To be able to experiment with the code examples, you will need access to a MongoDB database.

You can download a free MongoDB database at [https://www.mongodb.com](https://www.mongodb.com/).

Or get started right away with a MongoDB cloud service at <https://www.mongodb.com/cloud/atlas>.

Install MongoDB Driver

Let us try to access a MongoDB database with Node.js.

To download and install the official MongoDB driver, open the Command Terminal and execute the following:

Download and install mongodb package:

C:\Users\*Your Name*>npm install mongodb

Now you have downloaded and installed a mongodb database driver.

Node.js can use this module to manipulate MongoDB databases:

var mongo = require('mongodb');

Node.js MongoDB

Create Database

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_createcollection.asp)

Creating a Database

To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

MongoDB will create the database if it does not exist, and make a connection to it.

Example

Create a database called "mydb":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/mydb";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  console.log("Database created!");  
  db.close();  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_create_mongo_db)

Save the code above in a file called "demo\_create\_mongo\_db.js" and run the file:

Run "demo\_create\_mongo\_db.js"

C:\Users\*Your Name*>node demo\_create\_mongo\_db.js

Which will give you this result:

Database created!

**Important:** In MongoDB, a database is not created until it gets content!

MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

Node.js MongoDB Create Collection

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_create_db.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_insert.asp)

A **collection** in MongoDB is the same as a **table** in MySQL

Creating a Collection

To create a collection in MongoDB, use the createCollection() method:

Example

Create a collection called "customers":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.createCollection("customers", function(err, res) {  
    if (err) throw err;  
    console.log("Collection created!");  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_createcollection)

Save the code above in a file called "demo\_mongodb\_createcollection.js" and run the file:

Run "demo\_mongodb\_createcollection.js"

C:\Users\*Your Name*>node demo\_mongodb\_createcollection.js

Which will give you this result:

Collection created!

**Important:** In MongoDB, a collection is not created until it gets content!

MongoDB waits until you have inserted a document before it actually creates the collection.

Node.js MongoDB Insert

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_createcollection.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_find.asp)

Insert Into Collection

To insert a record, or *document* as it is called in MongoDB, into a collection, we use the insertOne() method.

A **document** in MongoDB is the same as a **record** in MySQL

The first parameter of the insertOne() method is an object containing the name(s) and value(s) of each field in the document you want to insert.

It also takes a callback function where you can work with any errors, or the result of the insertion:

Example

Insert a document in the "customers" collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var myobj = { name: "Company Inc", address: "Highway 37" };  
  dbo.collection("customers").insertOne(myobj, function(err, res) {  
    if (err) throw err;  
    console.log("1 document inserted");  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_insert)

Save the code above in a file called "demo\_mongodb\_insert.js" and run the file:

Run "demo\_mongodb\_insert.js"

C:\Users\*Your Name*>node demo\_mongodb\_insert.js

Which will give you this result:

1 document inserted

**Note:** If you try to insert documents in a collection that do not exist, MongoDB will create the collection automatically.

Insert Multiple Documents

To insert multiple documents into a collection in MongoDB, we use the insertMany() method.

The first parameter of the insertMany() method is an array of objects, containing the data you want to insert.

It also takes a callback function where you can work with any errors, or the result of the insertion:

Example

Insert multiple documents in the "customers" collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var myobj = [  
    { name: 'John', address: 'Highway 71'},  
    { name: 'Peter', address: 'Lowstreet 4'},  
    { name: 'Amy', address: 'Apple st 652'},  
    { name: 'Hannah', address: 'Mountain 21'},  
    { name: 'Michael', address: 'Valley 345'},  
    { name: 'Sandy', address: 'Ocean blvd 2'},  
    { name: 'Betty', address: 'Green Grass 1'},  
    { name: 'Richard', address: 'Sky st 331'},  
    { name: 'Susan', address: 'One way 98'},  
    { name: 'Vicky', address: 'Yellow Garden 2'},  
    { name: 'Ben', address: 'Park Lane 38'},  
    { name: 'William', address: 'Central st 954'},  
    { name: 'Chuck', address: 'Main Road 989'},  
    { name: 'Viola', address: 'Sideway 1633'}  
  ];  
  dbo.collection("customers").insertMany(myobj, function(err, res) {  
    if (err) throw err;  
    console.log("Number of documents inserted: " + res.insertedCount);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_insert_multiple)

Save the code above in a file called "demo\_mongodb\_insert\_multiple.js" and run the file:

Run "demo\_mongodb\_insert\_multiple.js"

C:\Users\*Your Name*>node demo\_mongodb\_insert\_multiple.js

Which will give you this result:

Number of documents inserted: 14

The Result Object

When executing the insertMany() method, a result object is returned.

The result object contains information about how the insertion affected the database.

The object returned from the example above looked like this:

{  
  result: { ok: 1, n: 14 },  
  ops: [  
    { name: 'John', address: 'Highway 71', \_id: 58fdbf5c0ef8a50b4cdd9a84 },  
    { name: 'Peter', address: 'Lowstreet 4', \_id: 58fdbf5c0ef8a50b4cdd9a85 },  
    { name: 'Amy', address: 'Apple st 652', \_id: 58fdbf5c0ef8a50b4cdd9a86 },  
    { name: 'Hannah', address: 'Mountain 21', \_id: 58fdbf5c0ef8a50b4cdd9a87 },  
    { name: 'Michael', address: 'Valley 345', \_id: 58fdbf5c0ef8a50b4cdd9a88 },  
    { name: 'Sandy', address: 'Ocean blvd 2', \_id: 58fdbf5c0ef8a50b4cdd9a89 },  
    { name: 'Betty', address: 'Green Grass 1', \_id: 58fdbf5c0ef8a50b4cdd9a8a },  
    { name: 'Richard', address: 'Sky st 331', \_id: 58fdbf5c0ef8a50b4cdd9a8b },  
    { name: 'Susan', address: 'One way 98', \_id: 58fdbf5c0ef8a50b4cdd9a8c },  
    { name: 'Vicky', address: 'Yellow Garden 2', \_id: 58fdbf5c0ef8a50b4cdd9a8d },  
    { name: 'Ben', address: 'Park Lane 38', \_id: 58fdbf5c0ef8a50b4cdd9a8e },  
    { name: 'William', address: 'Central st 954', \_id: 58fdbf5c0ef8a50b4cdd9a8f },  
    { name: 'Chuck', address: 'Main Road 989', \_id: 58fdbf5c0ef8a50b4cdd9a90 },  
    { name: 'Viola', address: 'Sideway 1633', \_id: 58fdbf5c0ef8a50b4cdd9a91 } ],  
  insertedCount: 14,  
  insertedIds: [  
    58fdbf5c0ef8a50b4cdd9a84,  
    58fdbf5c0ef8a50b4cdd9a85,  
    58fdbf5c0ef8a50b4cdd9a86,  
    58fdbf5c0ef8a50b4cdd9a87,  
    58fdbf5c0ef8a50b4cdd9a88,  
    58fdbf5c0ef8a50b4cdd9a89,  
    58fdbf5c0ef8a50b4cdd9a8a,  
    58fdbf5c0ef8a50b4cdd9a8b,  
    58fdbf5c0ef8a50b4cdd9a8c,  
    58fdbf5c0ef8a50b4cdd9a8d,  
    58fdbf5c0ef8a50b4cdd9a8e,  
    58fdbf5c0ef8a50b4cdd9a8f  
    58fdbf5c0ef8a50b4cdd9a90,  
    58fdbf5c0ef8a50b4cdd9a91 ]  
}

The values of the properties can be displayed like this:

Example

Return the number of inserted documents:

console.log(res.insertedCount)

Which will produce this result:

14

The \_id Field

If you do not specify an \_id field, then MongoDB will add one for you and assign a unique id for each document.

In the example above no \_id field was specified, and as you can see from the result object, MongoDB assigned a unique \_id for each document.

If you *do* specify the \_id field, the value must be unique for each document:

Example

Insert three records in a "products" table, with specified \_id fields:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var myobj = [  
    { **\_id: 154**, name: 'Chocolate Heaven'},  
    { **\_id: 155**, name: 'Tasty Lemon'},  
    { **\_id: 156**, name: 'Vanilla Dream'}  
  ];  
  dbo.collection("products").insertMany(myobj, function(err, res) {  
    if (err) throw err;  
    console.log(res);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_insert_id)

Save the code above in a file called "demo\_mongodb\_insert\_id.js" and run the file:

Run "demo\_mongodb\_insert\_id.js"

C:\Users\*Your Name*>node demo\_mongodb\_insert\_id.js

Which will give you this result:

{  
  result: { ok: 1, n: 3 },  
  ops: [  
    { \_id: 154, name: 'Chocolate Heaven },  
    { \_id: 155, name: 'Tasty Lemon },  
    { \_id: 156, name: 'Vanilla Dream } ],  
  insertedCount: 3,  
  insertedIds: [  
    154,  
    155,  
    156 ]  
}

Node.js MongoDB Find

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_insert.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_query.asp)

In MongoDB we use the **find** and **findOne** methods to find data in a collection.

Just like the **SELECT** statement is used to find data in a table in a MySQL database.

Find One

To select data from a collection in MongoDB, we can use the findOne() method.

The findOne() method returns the first occurrence in the selection.

The first parameter of the findOne() method is a query object. In this example we use an empty query object, which selects all documents in a collection (but returns only the first document).

Example

Find the first document in the customers collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").findOne({}, function(err, result) {  
    if (err) throw err;  
    console.log(result.name);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_findone)

Save the code above in a file called "demo\_mongodb\_findone.js" and run the file:

Run "demo\_mongodb\_findone.js"

C:\Users\*Your Name*>node demo\_mongodb\_findone.js

Which will give you this result:

Company Inc.

Find All

To select data from a table in MongoDB, we can also use the find() method.

The find() method returns all occurrences in the selection.

The first parameter of the find() method is a query object. In this example we use an empty query object, which selects all documents in the collection.

No parameters in the find() method gives you the same result as **SELECT \*** in MySQL.

Example

Find all documents in the customers collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").find({}).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_find)

Save the code above in a file called "demo\_mongodb\_find.js" and run the file:

Run "demo\_mongodb\_find.js"

C:\Users\*Your Name*>node demo\_mongodb\_find.js

Which will give you this result:

[  
  { \_id: 58fdbf5c0ef8a50b4cdd9a84 , name: 'John', address: 'Highway 71'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a85 , name: 'Peter', address: 'Lowstreet 4'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a86 , name: 'Amy', address: 'Apple st 652'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a87 , name: 'Hannah', address: 'Mountain 21'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a88 , name: 'Michael', address: 'Valley 345'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a89 , name: 'Sandy', address: 'Ocean blvd 2'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8a , name: 'Betty', address: 'Green Grass 1'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8b , name: 'Richard', address: 'Sky st 331'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8c , name: 'Susan', address: 'One way 98'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8d , name: 'Vicky', address: 'Yellow Garden 2'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8e , name: 'Ben', address: 'Park Lane 38'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8f , name: 'William', address: 'Central st 954'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a90 , name: 'Chuck', address: 'Main Road 989'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a91 , name: 'Viola', address: 'Sideway 1633'}  
]

Find Some

The second parameter of the find() method is the projection object that describes which fields to include in the result.

This parameter is optional, and if omitted, all fields will be included in the result.

Example

Return the fields "name" and "address" of all documents in the customers collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").find({},**{ projection: { \_id: 0, name: 1, address: 1 } }**).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_find_fields)

Save the code above in a file called "demo\_mongodb\_find\_fields.js" and run the file:

Run "demo\_mongodb\_find\_fields.js"

C:\Users\*Your Name*>node demo\_mongodb\_find\_fields.js

Which will give you this result:

[  
  { name: 'John', address: 'Highway 71'},  
  { name: 'Peter', address: 'Lowstreet 4'},  
  { name: 'Amy', address: 'Apple st 652'},  
  { name: 'Hannah', address: 'Mountain 21'},  
  { name: 'Michael', address: 'Valley 345'},  
  { name: 'Sandy', address: 'Ocean blvd 2'},  
  { name: 'Betty', address: 'Green Grass 1'},  
  { name: 'Richard', address: 'Sky st 331'},  
  { name: 'Susan', address: 'One way 98'},  
  { name: 'Vicky', address: 'Yellow Garden 2'},  
  { name: 'Ben', address: 'Park Lane 38'},  
  { name: 'William', address: 'Central st 954'},  
  { name: 'Chuck', address: 'Main Road 989'},  
  { name: 'Viola', address: 'Sideway 1633'}  
]

You are not allowed to specify both 0 and 1 values in the same object (except if one of the fields is the \_id field). If you specify a field with the value 0, all other fields get the value 1, and vice versa:

Example

This example will exclude "address" from the result:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").find({},**{ projection: { address: 0 } }**).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_find_fields4)

To exclude the \_id field, you must set its value to 0:

Example

This example will return only the "name" field:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").find({},**{ projection: { \_id: 0, name: 1 } }**).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_find_fields3)

Example

This example will give you the same result as the first example; return all fields except the \_id field:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").find({},**{ projection: { \_id: 0 } }**).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_find_fields2)

Example

You get an error if you specify both 0 and 1 values in the same object (except if one of the fields is the \_id field):

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").find({},**{ projection: { name: 1, address: 0 } }**).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_find_fields_error)

The Result Object

As you can see from the result of the example above, the result can be converted into an array containing each document as an object.

To return e.g. the address of the third document, just refer to the third array object's address property:

Example

Return the address of the third document:

console.log(result[2].address);

Which will produce this result:

Apple st 652

Node.js MongoDB Query

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_find.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_sort.asp)

Filter the Result

When finding documents in a collection, you can filter the result by using a query object.

The first argument of the find() method is a query object, and is used to limit the search.

Example

Find documents with the address "Park Lane 38":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
**var query = { address: "Park Lane 38" };**  dbo.collection("customers").find(**query**).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_query)

Save the code above in a file called "demo\_mongodb\_query.js" and run the file:

Run "demo\_mongodb\_query.js"

C:\Users\*Your Name*>node demo\_mongodb\_query.js

Which will give you this result:

[  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8e , name: 'Ben', address: 'Park Lane 38' }  
]

Filter With Regular Expressions

You can write regular expressions to find exactly what you are searching for.

**Regular expressions can only be used to query *strings*.**

To find only the documents where the "address" field starts with the letter "S", use the regular expression /^S/:

Example

Find documents where the address starts with the letter "S":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var query = { address: **/^S/** };  
  dbo.collection("customers").find(query).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_query_s)

Save the code above in a file called "demo\_mongodb\_query\_s.js" and run the file:

Run "demo\_mongodb\_query\_s.js"

C:\Users\*Your Name*>node demo\_mongodb\_query\_s.js

Which will give you this result:

[  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8b , name: 'Richard', address: 'Sky st 331' },  
  { \_id: 58fdbf5c0ef8a50b4cdd9a91 , name: 'Viola', address: 'Sideway 1633' }  
]

Node.js MongoDB Sort

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_query.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_delete.asp)

Sort the Result

Use the sort() method to sort the result in ascending or descending order.

The sort() method takes one parameter, an object defining the sorting order.

Example

Sort the result alphabetically by name:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
**var mysort = { name: 1 };**  dbo.collection("customers").find()**.sort(mysort)**.toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_sort)

Save the code above in a file called "demo\_sort.js" and run the file:

Run "demo\_sort.js"

C:\Users\*Your Name*>node demo\_sort.js

Which will give you this result:

[  
  { \_id: 58fdbf5c0ef8a50b4cdd9a86, name: 'Amy', address: 'Apple st 652'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8e, name: 'Ben', address: 'Park Lane 38'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8a, name: 'Betty', address: 'Green Grass 1'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a90, name: 'Chuck', address: 'Main Road 989'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a87, name: 'Hannah', address: 'Mountain 21'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a84, name: 'John', address: 'Highway 71'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a88, name: 'Michael', address: 'Valley 345'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a85, name: 'Peter', address: 'Lowstreet 4'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8b, name: 'Richard', address: 'Sky st 331'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a89, name: 'Sandy', address: 'Ocean blvd 2'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8c, name: 'Susan', address: 'One way 98'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8d, name: 'Vicky', address: 'Yellow Garden 2'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a91, name: 'Viola', address: 'Sideway 1633'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8f, name: 'William', address: 'Central st 954'}  
]

Sort Descending

Use the value -1 in the sort object to sort descending.

{ name: 1 } // ascending  
{ name: -1 } // descending

Example

Sort the result reverse alphabetically by name:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
**var mysort = { name: -1 };**  dbo.collection("customers").find()**.sort(mysort)**.toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_sort_desc)

Save the code above in a file called "demo\_sort\_desc.js" and run the file:

Run "demo\_sort\_desc.js"

C:\Users\*Your Name*>node demo\_sort\_desc.js

Which will give you this result:

[  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8f, name: 'William', address: 'Central st 954'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a91, name: 'Viola', address: 'Sideway 1633'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8d, name: 'Vicky', address: 'Yellow Garden 2'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8c, name: 'Susan', address: 'One way 98'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a89, name: 'Sandy', address: 'Ocean blvd 2'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8b, name: 'Richard', address: 'Sky st 331'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a85, name: 'Peter', address: 'Lowstreet 4'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a88, name: 'Michael', address: 'Valley 345'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a84, name: 'John', address: 'Highway 71'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a87, name: 'Hannah', address: 'Mountain 21'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a90, name: 'Chuck', address: 'Main Road 989'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8a, name: 'Betty', address: 'Green Grass 1'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8e, name: 'Ben', address: 'Park Lane 38'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a86, name: 'Amy', address: 'Apple st 652'}  
]

Node.js MongoDB Delete

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_sort.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_drop.asp)

Delete Document

To delete a record, or document as it is called in MongoDB, we use the deleteOne() method.

The first parameter of the deleteOne() method is a query object defining which document to delete.

**Note:** If the query finds more than one document, only the first occurrence is deleted.

Example

Delete the document with the address "Mountain 21":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
**var myquery = { address: 'Mountain 21' };**  dbo.collection("customers").deleteOne(myquery, function(err, obj) {  
    if (err) throw err;  
    console.log("1 document deleted");  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_delete)

Save the code above in a file called "demo\_delete.js" and run the file:

Run "demo\_delete.js"

C:\Users\*Your Name*>node demo\_delete.js

Which will give you this result:

1 document deleted

Delete Many

To delete more than one document, use the deleteMany() method.

The first parameter of the deleteMany() method is a query object defining which documents to delete.

Example

Delete all documents were the address starts with the letter "O":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
**var myquery = { address: /^O/ };**  dbo.collection("customers").deleteMany(myquery, function(err, obj) {  
    if (err) throw err;  
    console.log(obj.result.n + " document(s) deleted");  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_delete_many)

Save the code above in a file called "demo\_delete\_many.js" and run the file:

Run "demo\_delete\_many.js"

C:\Users\*Your Name*>node demo\_delete\_many.js

Which will give you this result:

2 document(s) deleted

The Result Object

The deleteMany() method returns an object which contains information about how the execution affected the database.

Most of the information is not important to understand, but one object inside the object is called "result" which tells us if the execution went OK, and how many documents were affected.

The result object looks like this:

{ n: 2, ok: 1 }

You can use this object to return the number of deleted documents:

Example

Return the number of deleted documents:

console.log(obj.result.n);

Which will produce this result:

2

Node.js MongoDB Drop

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_delete.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_update.asp)

Drop Collection

You can delete a table, or collection as it is called in MongoDB, by using the drop() method.

The drop() method takes a callback function containing the error object and the result parameter which returns true if the collection was dropped successfully, otherwise it returns false.

Example

Delete the "customers" table:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").drop(function(err, delOK) {  
    if (err) throw err;  
    if (delOK) console.log("Collection deleted");  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_drop)

Save the code above in a file called "demo\_drop.js" and run the file:

Run "demo\_drop.js"

C:\Users\*Your Name*>node demo\_drop.js

Which will give you this result:

Collection deleted

db.dropCollection

You can also use the dropCollection() method to delete a table (collection).

The dropCollection() method takes two parameters: the name of the collection and a callback function.

Example

Delete the "customers" collection, using dropCollection():

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb"); **dbo.dropCollection("customers", function(err, delOK) {**    if (err) throw err;  
    if (delOK) console.log("Collection deleted");  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_dropcollection)

Save the code above in a file called "demo\_dropcollection.js" and run the file:

Run "demo\_dropcollection.js"

C:\Users\*Your Name*>node demo\_dropcollection.js

Which will give you this result:

Collection deleted

Node.js MongoDB Update

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_drop.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_limit.asp)

Update Document

You can update a record, or document as it is called in MongoDB, by using the updateOne() method.

The first parameter of the updateOne() method is a query object defining which document to update.

**Note:** If the query finds more than one record, only the first occurrence is updated.

The second parameter is an object defining the new values of the document.

Example

Update the document with the address "Valley 345" to name="Mickey" and address="Canyon 123":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://127.0.0.1:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var myquery = { address: "Valley 345" };  
  var newvalues = { $set: {name: "Mickey", address: "Canyon 123" } };  
  dbo.collection("customers").updateOne(myquery, newvalues, function(err, res) {  
    if (err) throw err;  
    console.log("1 document updated");  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_update_one)

Save the code above in a file called "demo\_update\_one.js" and run the file:

Run "demo\_update\_one.js"

C:\Users\*Your Name*>node demo\_update\_one.js

Which will give you this result:

1 document updated

Update Only Specific Fields

When using the $set operator, only the specified fields are updated:

Example

Update the address from "Valley 345" to "Canyon 123":

...  
  var myquery = { address: "Valley 345" };  
  var newvalues = { **$set: { address: "Canyon 123" }** };  
  dbo.collection("customers").updateOne(myquery, newvalues, function(err, res) {  
...

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_update_set)

Update Many Documents

To update *all* documents that meets the criteria of the query, use the updateMany() method.

Example

Update all documents where the name starts with the letter "S":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://127.0.0.1:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var myquery = { address: /^S/ };  
  var newvalues = {$set: {name: "Minnie"} };  
  dbo.collection("customers").updateMany(myquery, newvalues, function(err, res) {  
    if (err) throw err;  
    console.log(res.result.nModified + " document(s) updated");  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_update_many)

Save the code above in a file called "demo\_update\_many.js" and run the file:

Run "demo\_update\_many.js"

C:\Users\*Your Name*>node demo\_update\_many.js

Which will give you this result:

2 document(s) updated

The Result Object

The updateOne() and the updateMany() methods return an object which contains information about how the execution affected the database.

Most of the information is not important to understand, but one object inside the object is called "result" which tells us if the execution went OK, and how many documents were affected.

The result object looks like this:

{ n: 1, nModified: 2, ok: 1 }

You can use this object to return the number of updated documents:

Example

Return the number of updated documents:

console.log(res.result.nModified);

Which will produce this result:

2

Node.js MongoDB Limit

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_update.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_mongodb_join.asp)

Limit the Result

To limit the result in MongoDB, we use the limit() method.

The limit() method takes one parameter, a number defining how many documents to return.

Consider you have a "customers" collection:

customers

[  
  { \_id: 58fdbf5c0ef8a50b4cdd9a84 , name: 'John', address: 'Highway 71'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a85 , name: 'Peter', address: 'Lowstreet 4'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a86 , name: 'Amy', address: 'Apple st 652'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a87 , name: 'Hannah', address: 'Mountain 21'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a88 , name: 'Michael', address: 'Valley 345'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a89 , name: 'Sandy', address: 'Ocean blvd 2'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8a , name: 'Betty', address: 'Green Grass 1'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8b , name: 'Richard', address: 'Sky st 331'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8c , name: 'Susan', address: 'One way 98'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8d , name: 'Vicky', address: 'Yellow Garden 2'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8e , name: 'Ben', address: 'Park Lane 38'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a8f , name: 'William', address: 'Central st 954'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a90 , name: 'Chuck', address: 'Main Road 989'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a91 , name: 'Viola', address: 'Sideway 1633'}  
]

Example

Limit the result to only return 5 documents:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").find()**.limit(5)**.toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_limit)

Save the code above in a file called "demo\_mongodb\_limit.js" and run the file:

Run "demo\_mongodb\_limit.js"

C:\Users\*Your Name*>node demo\_mongodb\_limit.js

Which will give you this result:

customers

[  
  { \_id: 58fdbf5c0ef8a50b4cdd9a84 , name: 'John', address: 'Highway 71'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a85 , name: 'Peter', address: 'Lowstreet 4'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a86 , name: 'Amy', address: 'Apple st 652'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a87 , name: 'Hannah', address: 'Mountain 21'},  
  { \_id: 58fdbf5c0ef8a50b4cdd9a88 , name: 'Michael', address: 'Valley 345'}  
]

As you can see from the result above, only the 5 first documents were returned.

Node.js MongoDB Join

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_limit.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_raspberrypi.asp)

Join Collections

MongoDB is not a relational database, but you can perform a left outer join by using the $lookup stage.

The $lookup stage lets you specify which collection you want to join with the current collection, and which fields that should match.

Consider you have a "orders" collection and a "products" collection:

orders

[  
  { \_id: 1, product\_id: 154, status: 1 }  
]

products

[  
  { \_id: 154, name: 'Chocolate Heaven' },  
  { \_id: 155, name: 'Tasty Lemons' },  
  { \_id: 156, name: 'Vanilla Dreams' }  
]

Example

Join the matching "products" document(s) to the "orders" collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://127.0.0.1:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection('orders').aggregate([  
**{ $lookup:  
       {  
         from: 'products',  
         localField: 'product\_id',  
         foreignField: '\_id',  
         as: 'orderdetails'  
       }  
     }**    ]).toArray(function(err, res) {  
    if (err) throw err;  
    console.log(JSON.stringify(res));  
    db.close();  
  });  
});

[Run example »](https://www.w3schools.com/nodejs/shownodejs_cmd.asp?filename=demo_mongodb_join)

Save the code above in a file called "demo\_mongodb\_join.js" and run the file:

Run "demo\_mongodb\_join.js"

C:\Users\*Your Name*>node demo\_mongodb\_join.js

Which will give you this result:

[  
  { "\_id": 1, "product\_id": 154, "status": 1, "orderdetails": [  
    { "\_id": 154, "name": "Chocolate Heaven" } ]  
  }  
]

As you can see from the result above, the matching document from the products collection is included in the orders collection as an array.

Node.js and Raspberry Pi

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_mongodb_join.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_raspberrypi_gpio_intro.asp)

Raspberry Pi is a small, multi-use computer.

With Node.js you can do amazing things with your Raspberry Pi.

What is the Raspberry Pi?

The Raspberry Pi is a small, affordable, and amazingly capable, credit card size computer.

It is developed by the Raspberry Pi Foundation, and it might be the most versatile tech ever created.

Creator Eben Upton's goal was to create a low-cost device that would improve programming skills and hardware understanding.

Due to the small size and price of the device, it has become the center of a wide range of projects by tinkerers, makers, and electronics enthusiasts.

Raspberry Pi and Node.js

The Raspberry Pi has a row of GPIO (General Purpose input/output) pins, and these can be used to interact in amazing ways with the real world. This tutorial will focus on how to use these with Node.js.

What Do I Need?

For this tutorial you need a Raspberry Pi. In our examples we use a Raspberry Pi 3, but this tutorial should work for most versions.

Hardware needed:

* Raspberry Pi computer
* MicroSD memory card (We recommend a class 10 with 16 GB or higher)
* MicroSD to SD memory card adapter (usually included with the MicroSD card)
* Micro USB power supply to power the Raspberry Pi (2.5A or greater recommended)
* WiFi/Ethernet Cable connection for the Raspberry Pi (Not needed for Raspberry Pi 3 as it has built in WiFi)
* A working computer with internet and SD memory card reader (used to get the OS (Operating System) for the Raspberry Pi onto the memory card). In our tutorial we use a Windows computer for this, but you can use a Mac or Linux computer if you prefer
* HDMI monitor, USB keyboard (we need these only temporarily for the first boot of the Raspberry Pi)

For later chapters in this tutorial we will use special sensors or devices that we connect to the Raspberry Pi. We will specify these as special requirements in the relevant chapters.

If you already have a Raspberry Pi set up with Raspbian, internet and enabled SSH, you can skip to the step "Install Node.js on Raspberry Pi".

Write Raspbian OS Image to MicroSD Card

Before we can start using our Raspberry Pi for anything, we need to get a OS installed.

Raspbian is a free operating system based on Debian Linux, and it is optimized Raspberry Pi.

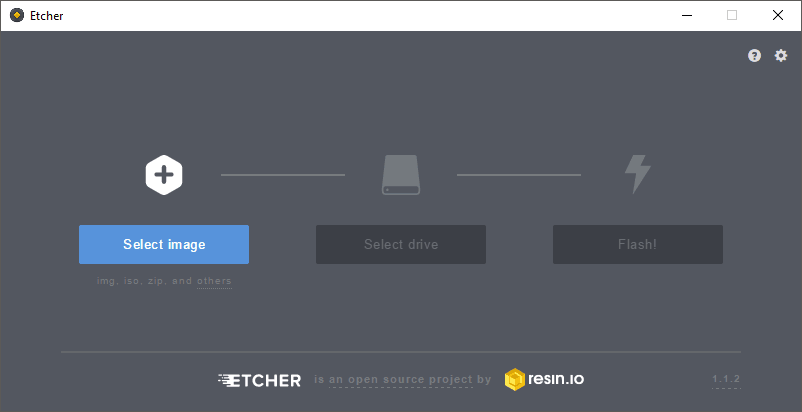
Download the latest Raspbian image from <https://www.raspberrypi.org/downloads/raspbian/> to your computer.

We use the "LITE" version in our tutorial, since we are setting the Raspberry Pi up as a headless server (we will connect to it through SSH, without having a keyboard/display connected to it). You can use whichever version you want, but this tutorial is written with the "LITE" version as its focus.

Insert the MicroSD memory card in your computer (via the SD adapter if needed). Open File Explorer to verify that it is operational.

Etcher is a program for flashing images to memory cards. Download and install Etcher from: <https://etcher.io/>

Launch Etcher:



Click "Select image" button and find the Raspbian zip file that you downloaded.

Click the "Select drive" button and specify the memory card as the target location.

Click the "Flash!" button to write the image to the memory card.

After Etcher is finished writing the image to the memory card, remove it from your computer.

Set up Your Raspberry Pi

To get the Raspberry Pi ready to boot we need to:

1. Insert the MicroSD memory card into the Raspberry Pi
2. Connect the USB keyboard
3. Connect the HDMI cable
4. Connect the USB Wi-Fi adapter (or Ethernet cable). Skip this step if you are using a Raspberry Pi 3
5. Connect the micro USB power supply
6. The Raspberry Pi should now be booting up

When the Raspberry Pi is finished booting up, log in using username: pi and password: raspberry

Set Up Network on the Raspberry Pi

If you will use a Ethernet cable to connect your Raspberry Pi to the internet, you can skip this step.

For this section we will assume you have a Raspberry Pi 3, with built in WiFi.

Start by scanning for wireless networks:

pi@raspberrypi:~ $ sudo iwlist wlan0 scan

This will list all of the available WiFi networks. (It also confirms that your WiFi is working)

Now we need to open the wpa-supplicant file, to add the network you want to connect to:

pi@raspberrypi:~ $ sudo nano /etc/wpa\_supplicant/wpa\_supplicant.conf

This will open the file in the Nano editor. Add the following to the bottom of the file (change wifiName and wifiPassword with the actual network name and password):

network={  
  ssid="wifiName"  
  psk="wifiPassword"  
}

Press "Ctrl+x" to save the code. Confirm with "y", and confirm the name with "Enter".

And reboot the Raspberry Pi:

pi@raspberrypi:~ $ sudo reboot

After reboot, log in again, and confirm that the WiFi is connected and working:

pi@raspberrypi:~ $ ifconfig wlan0

If the WiFi is working propery, the information displayed should include an IP address, similar to this:

inet addr:192.168.1.50

Write down that IP address, as we will use it to connect to the Raspberry Pi via SSH.

Enable SSH, Change Hostname and Password

Now your Raspberry Pi is connected to the internet, it is time to enable SSH.

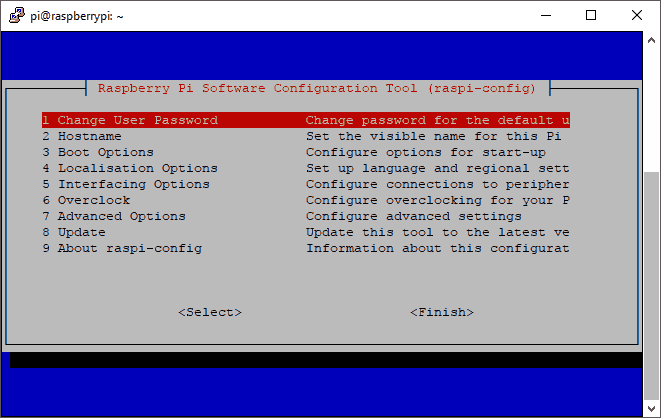
SSH allows you up use the Raspberry Pi without having a monitor and keyboard connected to it.

(You will need a SSH client for this on your non-Raspberry Pi computer. We use [PuTTY](http://www.putty.org/) for windows)

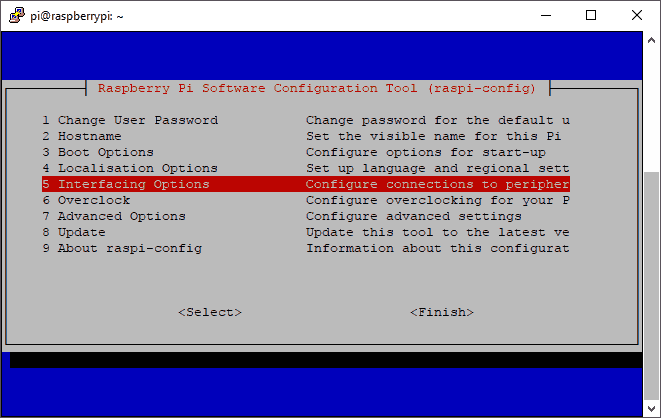
Open the Raspberry Pi Software Configuration Tool:

pi@raspberrypi:~ $ sudo raspi-config

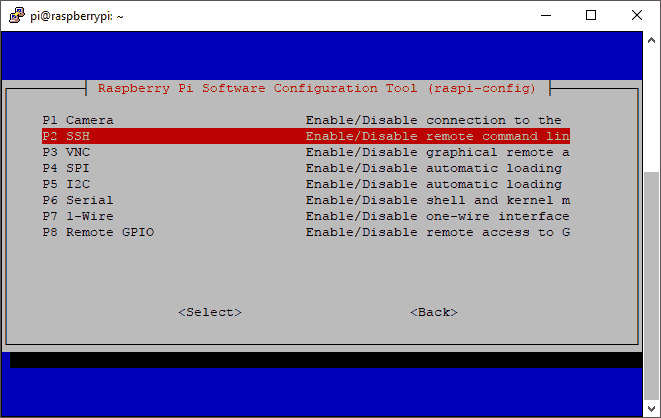
You should see a menu like this:



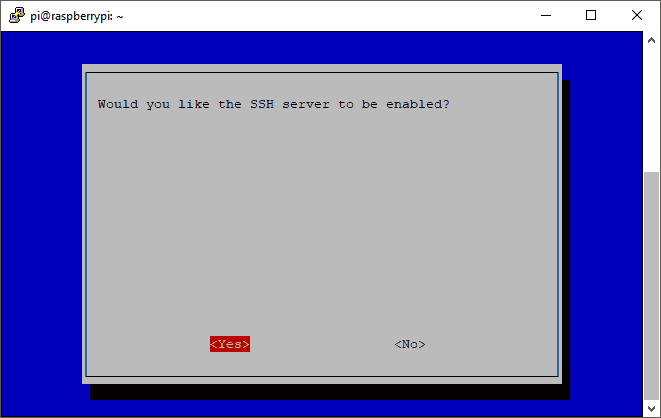
Select option 5 Interfacing Options:



Select option P2 SSH, to activate SSH:

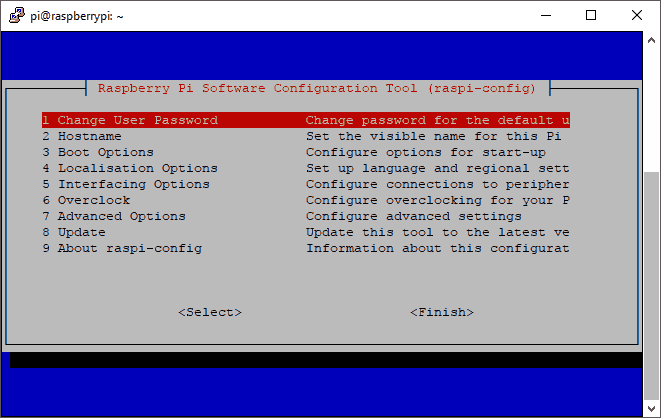


Confirm with YES, to activate SSH:



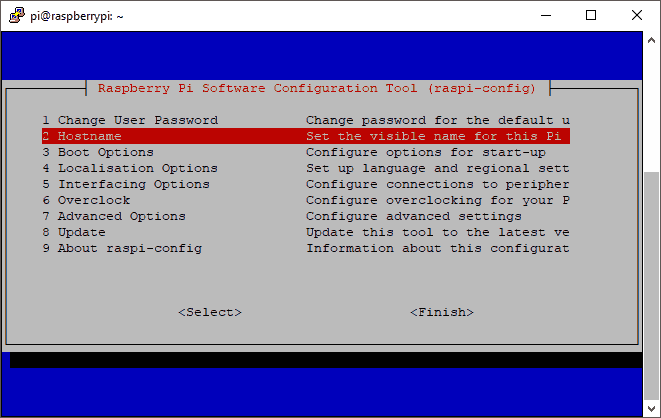
SSH is now enabled, and you should be in the main menu again.

Select 1 Change User Password, and follow the instructions to change the password. Choose a secure password, but something you will remember:



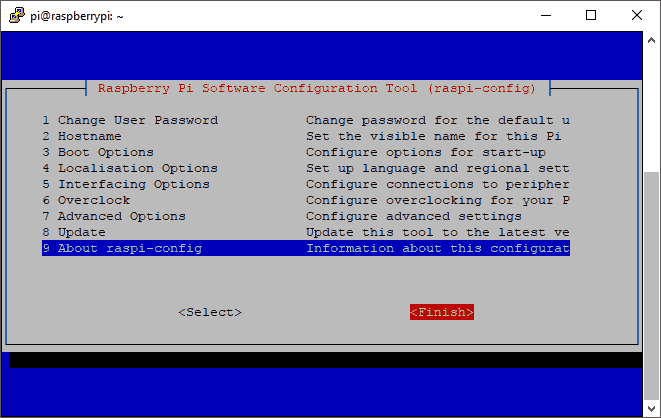
After you have finished changing the password, you should be back in the main menu.

Select 2 Hostname, and follow the instructions to change the hostname:

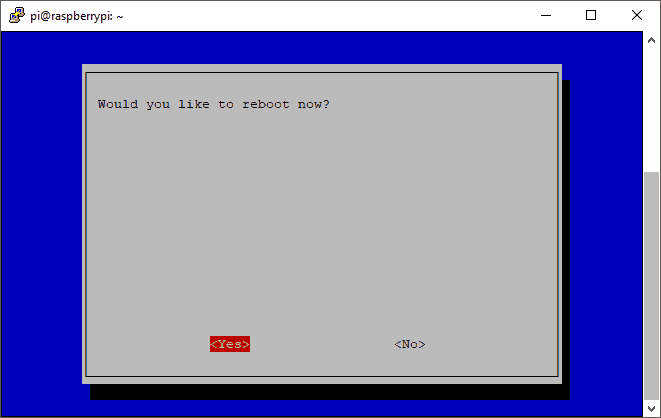


After you have finished changing the hostname, you should be back in the main menu.

Now we will close the menu and save the changes:

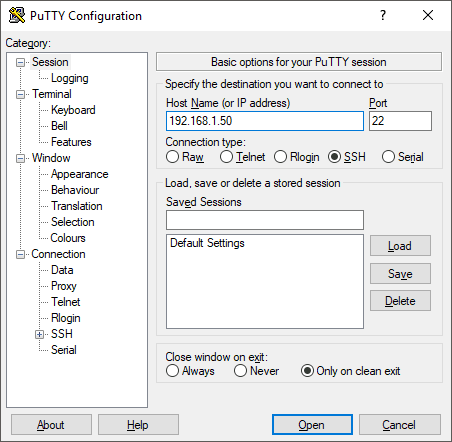


When selecting Finish, you will get the option to reboot. Select Yes to reboot the Raspberry Pi.



You can now unplug the monitor and keyboard from the Raspberry Pi, and we can log in using out SSH client.

Open PuTTY, type in the IP address for your Raspberry Pi, and click Open:



Log in using the username pi and the new password you specified.

You should now see a command line like this: (we used w3demopi as our hostname)

pi@w3demopi:~ $

You are now able to run your Raspberry Pi in "Headless-mode", meaning you do not need a monitor or keyboard. And if you have a WiFi connection, you do not need a ethernet cable either, just the power cable!

Install Node.js on Raspberry Pi

With the Raspberry Pi properly set up, login in via SSH, and update your Raspberry Pi system packages to their latest versions.

Update your system package list:

pi@w3demopi:~ $ sudo apt-get update

Upgrade all your installed packages to their latest version:

pi@w3demopi:~ $ sudo apt-get dist-upgrade

Doing this regularly will keep your Raspberry Pi installation up to date.

To download and install newest version of Node.js, use the following command:

pi@w3demopi:~ $ curl -sL https://deb.nodesource.com/setup\_8.x | sudo -E bash -

Now install it by running:

pi@w3demopi:~ $ sudo apt-get install -y nodejs

Check that the installation was successful, and the version number of Node.js with:

pi@w3demopi:~ $ node -v

Get Started with Raspberry Pi and Node.js

Now you have a Raspberry Pi with Node.js installed!

If you want to learn more about Node.js, follow our tutorial: <https://www.w3schools.com/nodejs/>

In the next chapter we will get to know the GPIO and how to use it with Node.js.

Node.js Raspberry Pi - GPIO Introduction

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_raspberrypi.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_raspberrypi_blinking_led.asp)

What is GPIO?

GPIO stands for General Purpose Input Output.

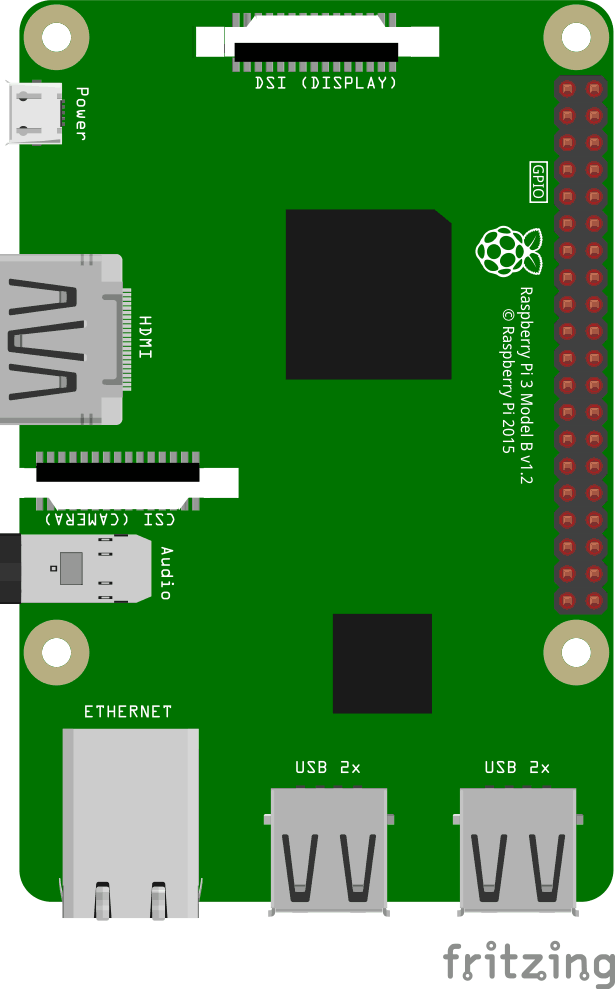
The Raspberry Pi has two rows of GPIO pins, which are connections between the Raspberry Pi, and the real world.

Output pins are like switches that the Raspberry Pi can turn on or off (like turning on/off a LED light). But it can also send a signal to another device.

Input pins are like switches that you can turn on or off from the outside world (like a on/off light switch). But it can also be a data from a sensor, or a signal from another device.

That means that you can interact with the real world, and control devices and electronics using the Raspberry PI and its GPIO pins!

Taking a Closer Look at the GPIO Pins



This is an illustration of the Raspberry Pi 3.

The GPIO pins are the small red squares in two rows on the right side of the Raspberry Pi, on the actual Raspberry Pi they are small metal pins.

The Raspberry Pi 3 has 26 GPIO pins, the rest of the pins are power, ground or "other".

The pin placements correspond with the table below.

Raspberry Pi B+, 2, 3 & Zero

|  |  |  |  |
| --- | --- | --- | --- |
| 3V3 | 1 | 2 | 5V |
| GPIO 2 | 3 | 4 | 5V |
| GPIO 3 | 5 | 6 | GND |
| GPIO 4 | 7 | 8 | GPIO 14 |
| GND | 9 | 10 | GPIO 15 |
| GPIO 17 | 11 | 12 | GPIO 18 |
| GPIO 27 | 13 | 14 | GND |
| GPIO 22 | 15 | 16 | GPIO 23 |
| 3V3 | 17 | 18 | GPIO 24 |
| GPIO 10 | 19 | 20 | GND |
| GPIO 9 | 21 | 22 | GPIO 25 |
| GPIO 11 | 23 | 24 | GPIO 8 |
| GND | 25 | 26 | GPIO 7 |
| DNC | 27 | 28 | DNC |
| GPIO 5 | 29 | 30 | GND |
| GPIO 6 | 31 | 32 | GPIO 12 |
| GPIO 13 | 33 | 34 | GND |
| GPIO 19 | 35 | 36 | GPIO 16 |
| GPIO 26 | 37 | 38 | GPIO 20 |
| GND | 39 | 40 | GPIO 21 |

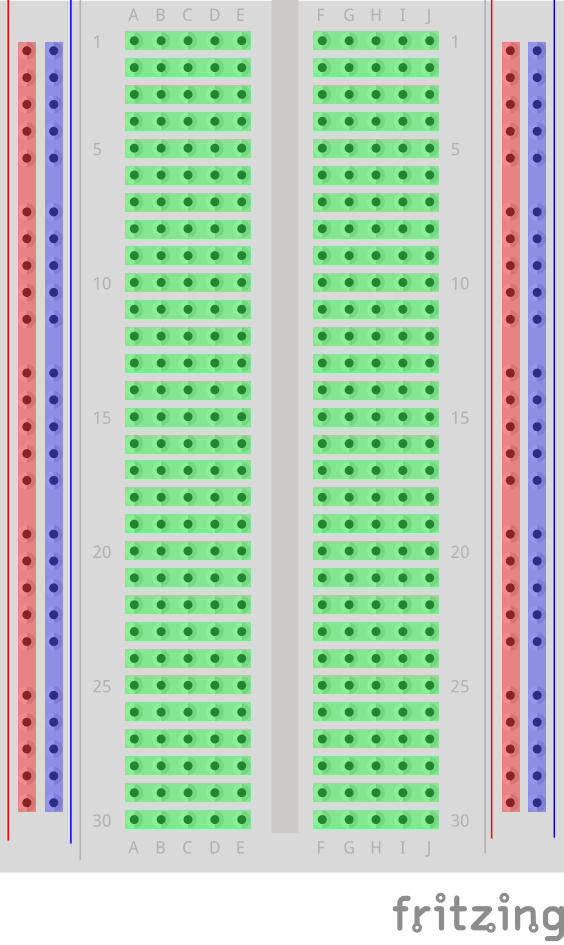
Legend

|  |
| --- |
| Physical Pin Number |
| Power + |
| Ground |
| UART |
| I2C |
| SPI |
| GPIO |
| Do Not Connect |

Taking a Closer Look at the Breadboard

A breadboard is used for prototyping electronics, it allows you to create circuits without soldering. It is basically a plastic board, with a grid of tie-points (holes). Inside the board there are metal strips connecting the different tie-points in specific ways.

In the illustration below we have highlighted some of the sections with different colors. This is to show you how the grid is connected.



The different sections of the breadboard:

* On the left, and right, side there are 2 columns of tie-points. All the tie points in each of these columns are connected.
* The Power Bus - The columns highlighted with red. There are usually used to connect power to the Breadboard. Since the entire column is connected, you can connect power to any of the tie-points in the column.
* The Ground Bus - The columns highlighted with blue. There are usually used to connect Ground to the Breadboard. Since the entire column is connected, you can connect ground to any of the tie-points in the column.
* Rows of connected Tie-Points - The rows highlighted with green. The tie-points of each of these rows are connected, but not the entire row! The left side tie-points are connected (A-B-C-D-E), and the right side tie-points are connected (F-G-H-I-J).
* In the center of the Breadboard there is a Trench, this separates the left and right rows. The width of the trench is designed so that many Integrated Circuits fit across it.

Install the onoff Module

To interface with the GPIO on the Raspberry Pi using Node.js, we will use a Module called "onoff".

Install the onoff module using npm:

pi@w3demopi:~ $ npm install onoff

Now onoff should be installed and we can interact with the GPIO of the Raspberry Pi.

Node.js Raspberry Pi GPIO - Blinking LED

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_raspberrypi_gpio_intro.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_raspberrypi_led_pushbutton.asp)

Using the GPIO for Output

In this chapter we will use a Raspberry Pi and its GPIO to make a LED blink.

We use Node.js with the onoff module to control the GPIO.

To get a LED light to turn on, we use a GPIO pin as "Output", and create a script to turn it on and off (blinking).

What do we need?

In this chapter we will create a simple example where we control a LED light.

For this you need:

* A Raspberry Pi with Raspian, internet, SSH, with Node.js installed
* The [onoff module](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#onoff) for Node.js
* 1 x [Breadboard](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#breadboard)
* 1 x [68 Ohm resistor](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#resistor68ohm)
* 1 x [Through Hole LED](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#throughHoleLED)
* 2 x [Female to male jumper wires](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#jumperWireFemaletoMale)

Click the links in the list above for descriptions of the different components.

**Note:** The resistor you need can be different from what we use depending on the type of LED you use. Most small LEDs only need a small resistor, around 200-500 ohms. It is generally not critical what exact value you use, but the smaller the value of the resistor, the brighter the LED will shine.

Building the Circuit

Now it is time to build the circuit on our Breadboard.

If you are new to electronics, we recommend you turn off the power for the Raspberry Pi. And use an anti-static mat or a grounding strap to avoid damaging it.

Shut down the Raspberry Pi properly with the command:

pi@w3demopi:~ $ sudo shutdown -h now

After the LEDs stop blinking on the Raspberry Pi, then pull out the power plug from the Raspberry Pi (or turn off the power strip it is connected to).

Just pulling the plug without shutting down properly may cause corruption of the memory card.

Node.js Raspberry Pi GPIO - LED and Pushbutton

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_raspberrypi_blinking_led.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_raspberrypi_flowing_leds.asp)

Using both Input and Output

In the previous chapter we learned how to use a Raspberry Pi and its GPIO to make a LED blink.

For that we used a GPIO pin as "Output".

In this chapter we will use another GPIO pin as "Input".

Instead of blinking for 5 seconds, we want the LED to light up when you push a button connected to the breadboard.

What do we need?

In this chapter we will create a simple example where we control a LED light with a Push Button.

For this you need:

* A Raspberry Pi with Raspian, internet, SSH, with Node.js installed
* The [onoff module](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#onoff) for Node.js
* 1 x [Breadboard](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#breadboard)
* 1 x [68 Ohm resistor](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#resistor68ohm)
* 1 x [1k Ohm resistor](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#resistor1kohm)
* 1 x [Through Hole LED](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#throughHoleLED)
* 1 x [Push Button](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#switchPushButton)
* 4 x [Female to male jumper wires](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#jumperWireFemaletoMale)
* 1 x [Male to Male jumper wires](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#jumperWireMaletoMale)

Click the links in the list above for descriptions of the different components.

**Note:** The resistor you need can be different from what we use depending on the type of LED you use. Most small LEDs only need a small resistor, around 200-500 ohms. It is generally not critical what exact value you use, but the smaller the value of the resistor, the brighter the LED will shine.

In this chapter we will build on the circuit we built in last chapter, so you will recognize some of the parts in the list above.

Building the Circuit

Now it is time to build the circuit on our Breadboard. We will use the [circuit we created in the last chapter](https://www.w3schools.com/nodejs/nodejs_raspberrypi_gpio_led.asp) as a starting point.

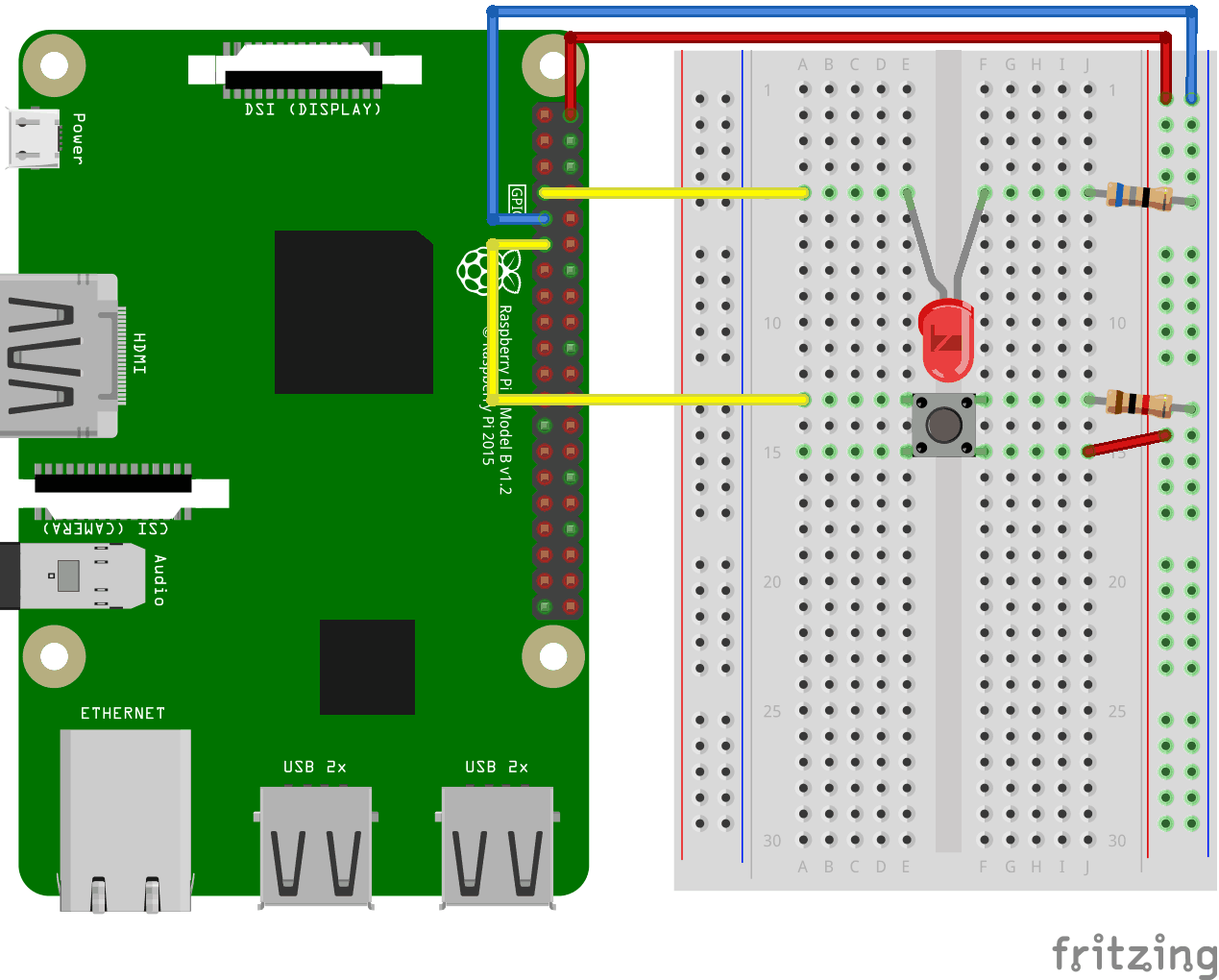
If you are new to electronics, we recommend you turn off the power for the Raspberry Pi. And use an anti-static mat or a grounding strap to avoid damaging it.

Shut down the Raspberry Pi properly with the command:

pi@w3demopi:~ $ sudo shutdown -h now

After the LEDs stop blinking on the Raspberry Pi, then pull out the power plug from the Raspberry Pi (or turn of the power strip it is connected to).

Just pulling the plug without shutting down properly may cause corruption of the memory card.



Look at the above illustration of the circuit.

1. Starting with the circuit we created in the last chapter:  
   On the Raspberry Pi, connect the female leg of a jumper wire to a 5V power pin. In our example we used Physical Pin 2 (5V, row 1, right column)
2. On the Breadboard, connect the male leg of the jumper wire connected to the 5V power, to the Power Bus on the right side. That entire column of your breadboard is connected, so it doesn't matter which row. In our example we attached it to row 1
3. On the Breadboard, connect the push button so that it fits across the Trench. In our example it connects to rows 13 and 15, columns E and F
4. On the Breadboard, connect one leg of the 1k ohm resistor to the Ground Bus column on the right side, and the other leg to the right side Tie-Point row where it connects to one of the right side legs of the push button. In our example we attached one side to Tie-Point row 13, column J, and the other side to the closest Ground Bus hole
5. On the Breadboard, connect a male-to-male jumper wire from the right Power Bus, to the right Tie-Point row that connects to the other leg of the push button. In our example we attached one side to Tie-Point row 15, column J, and the other side to the closest Power Bus hole
6. On the Raspberry Pi, connect the female leg of a jumper wire to a GPIO pin. In our example we used Physical Pin 11 (GPIO 17, row 6, left column)
7. On the Breadboard, connect the male leg of the jumper wire to left Tie-Point row the Push Button leg that is directly across the Ground connection leg.  In our example we attached it to row 13, column A

Your circuit should now be complete, and your connections should look pretty similar to the illustration above.

Now it is time to boot up the Raspberry Pi, and write the Node.js script to interact with it.

Raspberry Pi and Node.js LED and Button Script

Go to the "nodetest" directory, and create a new file called "buttonled.js":

pi@w3demopi:~ $ nano buttonled.js

The file is now open and can be edited with the built in Nano Editor.

Write, or paste the following:

buttonled.js

var Gpio = require('onoff').Gpio; //include onoff to interact with the GPIO  
var LED = new Gpio(4, 'out'); //use GPIO pin 4 as output  
var pushButton = new Gpio(17, 'in', 'both'); //use GPIO pin 17 as input, and 'both' button presses, and releases should be handled  
  
pushButton.watch(function (err, value) { //Watch for hardware interrupts on pushButton GPIO, specify callback function  
  if (err) { //if an error  
    console.error('There was an error', err); //output error message to console  
  return;  
  }  
  LED.writeSync(value); //turn LED on or off depending on the button state (0 or 1)  
});  
  
function unexportOnClose() { //function to run when exiting program  
  LED.writeSync(0); // Turn LED off  
  LED.unexport(); // Unexport LED GPIO to free resources  
  pushButton.unexport(); // Unexport Button GPIO to free resources  
};  
  
process.on('SIGINT', unexportOnClose); //function to run when user closes using ctrl+c

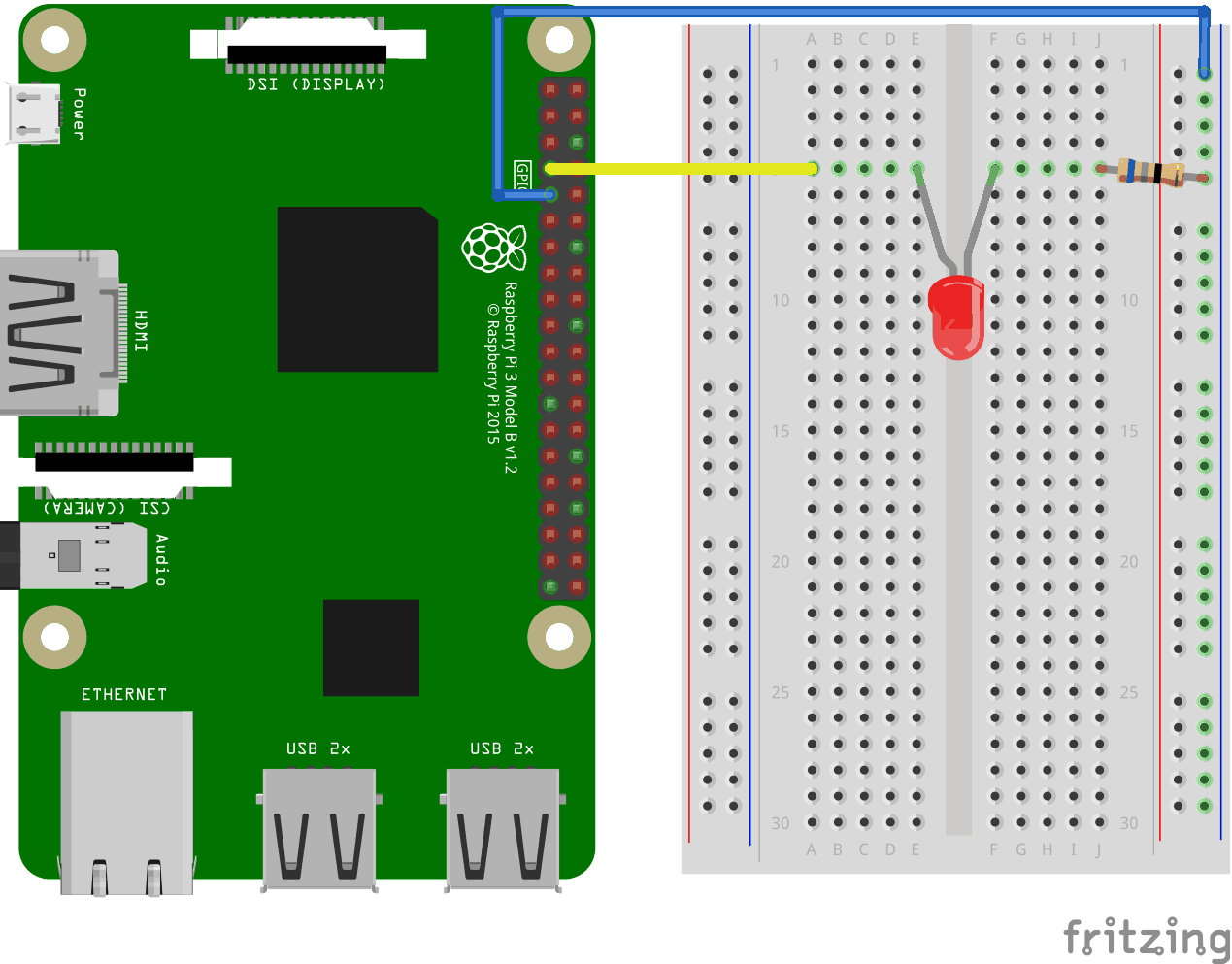
Press "Ctrl+x" to save the code. Confirm with "y", and confirm the name with "Enter".

Run the code:

pi@w3demopi:~ $ node buttonled.js

Now the LED should turn on when you press the button, and turn off when you release it.

End the program with Ctrl+c.



Look at the above illustration of the circuit.

1. On the Raspberry Pi, connect the female leg of the first jumper wire to Ground. You can use any GND pin. In this example we used Physical Pin 9 (GND, row 5, left column)
2. On the Breadboard, connect the male leg of the first jumper wire to the Ground Bus column on the right. That entire column of your breadboard is connected, so it doesn't matter which row. In this example we have attached it to row 1
3. On the Raspberry Pi, connect the female leg of the second jumper cable to a GPIO pin. In this example we used Physical Pin 7 (GPIO 4, row 4, left column)
4. On the Breadboard, connect the male leg of the second jumper wire to the Tie-Point row of your choice. In this example we connected it to row 5, column A
5. On the Breadboard, connect one leg of the resistor to the Ground Bus column on the right side. That entire column of your breadboard is connected, so it doesn't matter which row. In this example we have attached it to row 5
6. On the Breadboard, connect the other leg of the resistor to the right side Tie-Point row of your choice. In this example we have used row 5, column J
7. On the Breadboard, connect the cathode leg (the shortest leg) of the LED to the same Tie-Point row that you connected the resistor from GND to. In this example we used row 5, column F
8. On the Breadboard, connect the anode leg (the longest leg) of the LED to the same Tie-Point row that you connected the jumper from the GPIO pin to. In this example we used row 5, column E

Your circuit should now be complete, and your connections should look pretty similar to the illustration above.

Now it is time to boot up the Raspberry Pi, and write the Node.js script to interact with it.

Raspberry Pi and Node.js Blinking LED Script

Now that we have everything set up, we can write a script to turn the LED on and off.

Start by making a directory where we can keep our Node.js scripts:

pi@w3demopi:~ $ mkdir nodetest

Go to our new directory:

pi@w3demopi:~ $ cd nodetest

Now we will create a new file called "blink.js" using the Nano Editor:

pi@w3demopi:~ $ nano blink.js

The file is now open and can be edited with the built in Nano Editor.

Write, or paste the following code:

blink.js

var Gpio = require('onoff').Gpio; //include onoff to interact with the GPIO  
var LED = new Gpio(4, 'out'); //use GPIO pin 4, and specify that it is output  
var blinkInterval = setInterval(blinkLED, 250); //run the blinkLED function every 250ms  
  
function blinkLED() { //function to start blinking  
  if (LED.readSync() === 0) { //check the pin state, if the state is 0 (or off)  
    LED.writeSync(1); //set pin state to 1 (turn LED on)  
  } else {  
    LED.writeSync(0); //set pin state to 0 (turn LED off)  
  }  
}  
  
function endBlink() { //function to stop blinking  
  clearInterval(blinkInterval); // Stop blink intervals  
  LED.writeSync(0); // Turn LED off  
  LED.unexport(); // Unexport GPIO to free resources  
}  
  
setTimeout(endBlink, 5000); //stop blinking after 5 seconds

Press "Ctrl+x" to save the code. Confirm with "y", and confirm the name with "Enter".

Run the code:

pi@w3demopi:~ $ node blink.js

Now the LED should blink for 5 seconds (10 times) before turning off again!

Node.js Raspberry Pi GPIO - Flowing LEDs

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_raspberrypi_led_pushbutton.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_raspberrypi_webserver_websocket.asp)

Using Array With Output to Create Flowing LEDs

In this chapter we will use several GPIO pins to create a "flowing" effect by turning them on and off in sequence.

What do we need?

For this you need:

* A Raspberry Pi with Raspian, internet, SSH, with Node.js installed
* The [onoff module](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#onoff) for Node.js
* 1 x [Breadboard](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#breadboard)
* 8 x [220 Ohm resistor](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#resistor220ohm)
* 8 x [Through Hole LED](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#throughHoleLED)
* 9 x [Female to male jumper wires](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#jumperWireFemaletoMale)

**Note:** The resistor you need can be different from what we use depending on the type of LEDs you use. Most small LEDs only need a small resistor, around 200-500 ohms. It is generally not critical what exact value you use, but the smaller the value of the resistor, the brighter the LED will shine.

Click the links in the list above for descriptions of the different components.

Building the Circuit

Now it is time to build the circuit on our Breadboard.

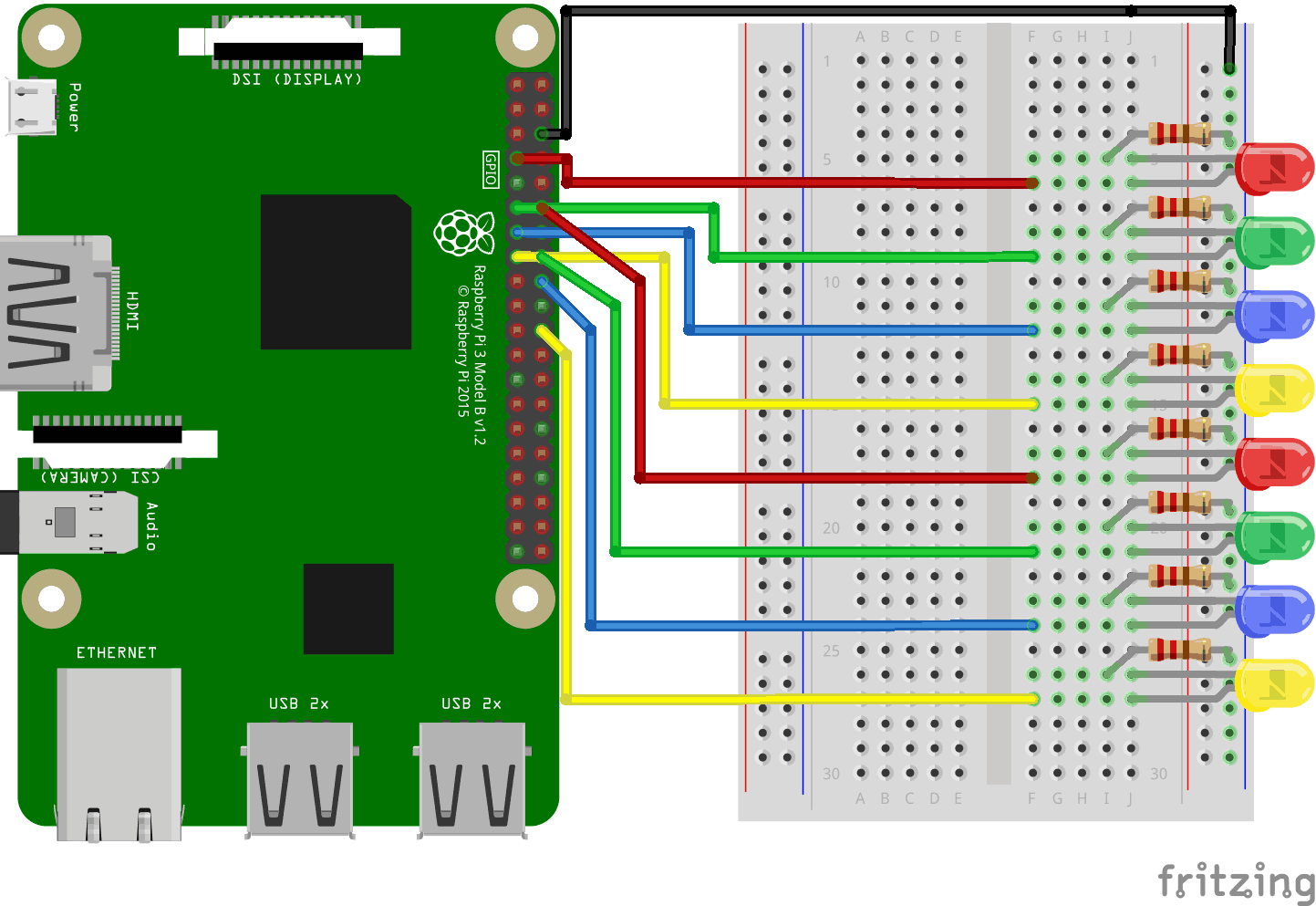
If you are new to electronics, we recommend you turn off the power for the Raspberry Pi. And use an anti-static mat or a grounding strap to avoid damaging it.

Shut down the Raspberry Pi properly with the command:

pi@w3demopi:~ $ sudo shutdown -h now

After the LEDs stop blinking on the Raspberry Pi, then pull out the power plug from the Raspberry Pi (or turn of the power strip it is connected to).

Just pulling the plug without shutting down properly may cause corruption of the memory card.



Look at the above illustration of the circuit.

1. On the Raspberry Pi, connect the female leg of a jumper wire to a GND pin. In our example we used Physical Pin 6 (GND, row 3, right column)
2. On the Breadboard, connect the male leg of the jumper wire connected to the GND power, to the Ground Bus on the right side. That entire column of your breadboard is connected, so it doesn't matter which row. In our example we attached it to row 1
3. For each LED: Connect the LED so that it connects to 2 Tie-Point rows. In our example we connected:
   1. LED1 to rows 5 (cathode) & 6 (anode) column J
   2. LED2 to rows 8 (cathode) & 9 (anode) column J
   3. LED3 to rows 11 (cathode) & 12 (anode) column J
   4. LED4 to rows 14 (cathode) & 15 (anode) column J
   5. LED5 to rows 17 (cathode) & 18 (anode) column J
   6. LED6 to rows 20 (cathode) & 21 (anode) column J
   7. LED7 to rows 23 (cathode) & 24 (anode) column J
   8. LED8 to rows 26 (cathode) & 27 (anode) column J
4. For each LED: Connect one of the legs of a 220 ohm resistor from the the Ground Bus column on the right side, and the other leg to the right side Tie-Point row where it connects to the cathode leg of the LED. In our example we connected:
   1. LED1 to row 5 column I
   2. LED2 to row 8 column I
   3. LED3 to row 11 column I
   4. LED4 to row 14 column I
   5. LED5 to row 17 column I
   6. LED6 to row 20 column I
   7. LED7 to row 23 column I
   8. LED8 to row 26 column I
5. For each LED: Connect the female leg of a jumper wire to a GPIO pin on the Raspberry Pi, and the male leg of the jumper wire to the right side Tie-Point row where it connects to the anode leg of the LED. In our example we connected:
   1. LED1 from Physical Pin 7 (GPIO 4, row 4, left column) to Tie-point row 6 column F
   2. LED2 from Physical Pin 11 (GPIO 17, row 6, left column) to Tie-point row 9 column F
   3. LED3 from Physical Pin 13 (GPIO 27, row 7, left column) to Tie-point row 12 column F
   4. LED4 from Physical Pin 15 (GPIO 22, row 8, left column) to Tie-point row 15 column F
   5. LED5 from Physical Pin 12 (GPIO 18, row 6, right column) to Tie-point row 18 column F
   6. LED6 from Physical Pin 16 (GPIO 23, row 8, right column) to Tie-point row 21 column F
   7. LED7 from Physical Pin 18 (GPIO 24, row 9, right column) to Tie-point row 24 column F
   8. LED8 from Physical Pin 22 (GPIO 25, row 11, right column) to Tie-point row 27 column F

Your circuit should now be complete, and your connections should look pretty similar to the illustration above.

Now it is time to boot up the Raspberry Pi, and write the Node.js script to interact with it.

Raspberry Pi and Node.js Flowing LEDs Script

Go to the "nodetest" directory, and create a new file called "flowingleds.js":

https://youtu.be/FvbiyVa03\_4

pi@w3demopi:~ $ nano flowingleds.js

The file is now open and can be edited with the built in Nano Editor.

Write, or paste the following:

flowingleds.js

var Gpio = require('onoff').Gpio; //include onoff to interact with the GPIO  
var LED04 = new Gpio(4, 'out'), //use declare variables for all the GPIO output pins  
  LED17 = new Gpio(17, 'out'),  
  LED27 = new Gpio(27, 'out'),  
  LED22 = new Gpio(22, 'out'),  
  LED18 = new Gpio(18, 'out'),  
  LED23 = new Gpio(23, 'out'),  
  LED24 = new Gpio(24, 'out'),  
  LED25 = new Gpio(25, 'out');  
  
//Put all the LED variables in an array  
var leds = [LED04, LED17, LED27, LED22, LED18, LED23, LED24, LED25];  
var indexCount = 0; //a counter  
dir = "up"; //variable for flowing direction  
  
var flowInterval = setInterval(flowingLeds, 100); //run the flowingLeds function every 100ms  
  
function flowingLeds() { //function for flowing Leds  
  leds.forEach(function(currentValue) { //for each item in array  
    currentValue.writeSync(0); //turn off LED  
  });  
  if (indexCount == 0) dir = "up"; //set flow direction to "up" if the count reaches zero  
  if (indexCount >= leds.length) dir = "down"; //set flow direction to "down" if the count reaches 7  
  if (dir == "down") indexCount--; //count downwards if direction is down  
  leds[indexCount].writeSync(1); //turn on LED that where array index matches count  
  if (dir == "up") indexCount++ //count upwards if direction is up  
};  
  
function unexportOnClose() { //function to run when exiting program  
  clearInterval(flowInterval); //stop flow interwal  
  leds.forEach(function(currentValue) { //for each LED  
    currentValue.writeSync(0); //turn off LED  
    currentValue.unexport(); //unexport GPIO  
  });  
};  
  
process.on('SIGINT', unexportOnClose); //function to run when user closes using ctrl+cc

Press "Ctrl+x" to save the code. Confirm with "y", and confirm the name with "Enter".

Run the code:

pi@w3demopi:~ $ node flowingleds.js

Now the LEDs should turn on and off in sequence, creating a flowing effect.

End the program with Ctrl+c.

Node.js and Raspberry Pi - Webserver with WebSocket

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_raspberrypi_flowing_leds.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_raspberrypi_rgb_led_websocket.asp)

What is WebSocket?

WebSocket enables bidirectional communication in real time over the web.

WebSocket can be run together with a normal HTTP server. You can click a button in a web browser, and enable a GPIO on your Raspberry Pi which turns on a light in your house. All in real time, and with communication going both ways!

In this chapter, we will set up a web server with WebSocket. Then create a browser UI to interact with our earlier example of [turning a LED on and off with a button](https://www.w3schools.com/nodejs/nodejs_raspberrypi_led_pushbutton.asp).

What Do I Need?

For this tutorial you need a Raspberry Pi. In our examples we use a a Raspberry Pi 3, but this tutorial should work for most versions.

For this you need:

* A Raspberry Pi with Raspian, internet, SSH, with Node.js installed
* The [onoff module](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#onoff) for Node.js
* The [socket.io module](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#socket.io) for Node.js
* 1 x [Breadboard](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#breadboard)
* 1 x [68 Ohm resistor](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#resistor68ohm)
* 1 x [1k Ohm resistor](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#resistor1kohm)
* 1 x [Through Hole LED](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#throughHoleLED)
* 1 x [Push Button](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#switchPushButton)
* 4 x [Female to male jumper wires](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#jumperWireFemaletoMale)
* 1 x [Male to Male jumper wires](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#jumperWireMaletoMale)

Click the links in the list above for descriptions of the different components.

**Note:** The resistor you need can be different from what we use depending on the type of LED you use. Most small LEDs only need a small resistor, around 200-500 ohms. It is generally not critical what exact value you use, but the smaller the value of the resistor, the brighter the LED will shine.

Compared to our earlier example, the only new thing we need is to set up a web server, and install the socket.io module.

Webserver for Raspberry Pi and Node.js

Following the earlier chapters in this Node.js tutorial, lets set up a web server that can serve HTML files.

In our "nodetest" directory create a new directory we can use for static html files:

pi@w3demopi:~/nodetest $ mkdir public

Now lets set up a webserver. Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error.

pi@w3demopi:~/nodetest $ nano webserver.js

webserver.js:

var http = require('http').createServer(handler); //require http server, and create server with function handler()  
var fs = require('fs'); //require filesystem module  
  
http.listen(8080); //listen to port 8080  
  
function handler (req, res) { //create server  
  fs.readFile(\_\_dirname + '/public/index.html', function(err, data) { //read file index.html in public folder  
    if (err) {  
      res.writeHead(404, {'Content-Type': 'text/html'}); //display 404 on error  
      return res.end("404 Not Found");  
    }  
    res.writeHead(200, {'Content-Type': 'text/html'}); //write HTML  
    res.write(data); //write data from index.html  
    return res.end();  
  });  
}

Go to the folder "public":

pi@w3demopi:~/nodetest $ cd public

And create a HTML file, index.html:

pi@w3demopi:~/nodetest/public $ nano index.html

index.html:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>Control LED light</h1>  
<input id="light" type="checkbox">LED  
  
</body>  
</html>

This file will not have any functionality yet. For now it is just a placeholder. Lets see if the webserver is working:

pi@w3demopi:~/nodetest/public $ cd ..

pi@w3demopi:~/nodetest $ node webserver.js

Open the website in a browser using http://[RaspberryPi\_IP]:8080/:

The webserver should now be up and running, and we can move on to the WebSocket part.

Install socket.io for Node.js

With the webserver set up, update your Raspberry Pi system packages to their latest versions.

Update your system package list:

pi@w3demopi:~ $ sudo apt-get update

Upgrade all your installed packages to their latest version:

pi@w3demopi:~ $ sudo apt-get dist-upgrade

Doing this regularly will keep your Raspberry Pi installation up to date.

To download and install newest version of socket.io, use the following command:

pi@w3demopi:~ $ npm install socket.io --save

Adding WebSocket to our Webserver

Now we can use WebSocket in our application. Lets update our index.html file:

index.html:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>Control LED light</h1>  
<p><input type="checkbox" id="light"></p>  
  
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.3/socket.io.js"></script> <!-- include socket.io client side script -->  
<script>  
var socket = io(); //load socket.io-client and connect to the host that serves the page  
window.addEventListener("load", function(){ //when page loads  
  var lightbox = document.getElementById("light");  
  lightbox.addEventListener("change", function() { //add event listener for when checkbox changes  
    socket.emit("light", Number(this.checked)); //send button status to server (as 1 or 0)  
  });  
});  
socket.on('light', function (data) { //get button status from client  
  document.getElementById("light").checked = data; //change checkbox according to push button on Raspberry Pi  
  socket.emit("light", data); //send push button status to back to server  
});  
</script>  
  
</body>  
</html>

And our webserver.js file:

webserver.js:

var http = require('http').createServer(handler); //require http server, and create server with function handler()  
var fs = require('fs'); //require filesystem module  
var io = require('socket.io')(http) //require socket.io module and pass the http object (server)  
  
http.listen(8080); //listen to port 8080  
  
function handler (req, res) { //create server  
  fs.readFile(\_\_dirname + '/public/index.html', function(err, data) { //read file index.html in public folder  
    if (err) {  
      res.writeHead(404, {'Content-Type': 'text/html'}); //display 404 on error  
      return res.end("404 Not Found");  
    }  
    res.writeHead(200, {'Content-Type': 'text/html'}); //write HTML  
    res.write(data); //write data from index.html  
    return res.end();  
  });  
}  
  
io.sockets.on('connection', function (socket) {// WebSocket Connection  
  var lightvalue = 0; //static variable for current status  
  socket.on('light', function(data) { //get light switch status from client  
    lightvalue = data;  
    if (lightvalue) {  
      console.log(lightvalue); //turn LED on or off, for now we will just show it in console.log  
    }  
  });  
});

Lets test the server:

pi@w3demopi:~ $ node webserver.js

Open the website in a browser using http://[RaspberryPi\_IP]:8080/:

Now the server should output all the changes to the checkbox to the console on the Raspberry Pi.

The client is sending the changes to the server, and the server is responding.

Lets add the [push button controlled LED](https://www.w3schools.com/nodejs/nodejs_raspberrypi_led_pushbutton.asp) from a previous chapter.

Adding Hardware, and sending a response to the Client

Lets update our webserver.js file again. We will use a lot of the code from the Pushbutton controlled LED chapter.

https://youtu.be/iedgqPJ0xX8

webserver.js:

var http = require('http').createServer(handler); //require http server, and create server with function handler()  
var fs = require('fs'); //require filesystem module  
var io = require('socket.io')(http) //require socket.io module and pass the http object (server)  
var Gpio = require('onoff').Gpio; //include onoff to interact with the GPIO  
var LED = new Gpio(4, 'out'); //use GPIO pin 4 as output  
var pushButton = new Gpio(17, 'in', 'both'); //use GPIO pin 17 as input, and 'both' button presses, and releases should be handled  
  
http.listen(8080); //listen to port 8080  
  
function handler (req, res) { //create server  
  fs.readFile(\_\_dirname + '/public/index.html', function(err, data) { //read file index.html in public folder  
    if (err) {  
      res.writeHead(404, {'Content-Type': 'text/html'}); //display 404 on error  
      return res.end("404 Not Found");  
    }  
    res.writeHead(200, {'Content-Type': 'text/html'}); //write HTML  
    res.write(data); //write data from index.html  
    return res.end();  
  });  
}  
  
io.sockets.on('connection', function (socket) {// WebSocket Connection  
  var lightvalue = 0; //static variable for current status  
  pushButton.watch(function (err, value) { //Watch for hardware interrupts on pushButton  
    if (err) { //if an error  
      console.error('There was an error', err); //output error message to console  
      return;  
    }  
    lightvalue = value;  
    socket.emit('light', lightvalue); //send button status to client  
  });  
  socket.on('light', function(data) { //get light switch status from client  
    lightvalue = data;  
    if (lightvalue != LED.readSync()) { //only change LED if status has changed  
      LED.writeSync(lightvalue); //turn LED on or off  
    }  
  });  
});  
  
process.on('SIGINT', function () { //on ctrl+c  
  LED.writeSync(0); // Turn LED off  
  LED.unexport(); // Unexport LED GPIO to free resources  
  pushButton.unexport(); // Unexport Button GPIO to free resources  
  process.exit(); //exit completely  
});

Lets test the server:

pi@w3demopi:~ $ node webserver.js

Open the website in a browser using http://[RaspberryPi\_IP]:8080/:

Now the server should output all the changes to the checkbox to the console on the Raspberry Pi.

The client is sending the changes to the server, and the server is responding.

End the program with Ctrl+c.

Node.js Raspberry Pi RGB LED with WebSocket

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_raspberrypi_webserver_websocket.asp)[Next ❯](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp)

Using Pulse-Width Modulation

In the previous chapters we have learned how to use WebSocket, and how to use GPIO to turn LEDs on and off.

In this we will use chapter we use a RGB LED, with PWM (Pulse-width modulation) to display different colors based on user input via WebSocket.

An RGB LED is a LED with 3 different colors. It has a RED, GREEN and BLUE LED (RGB LED).

And using PWM, we can set the individual strength of the 3 LEDs. This will allow us to mix them, to set a color.

What do we need?

In this chapter we will create an example where we control an RGB LED with a web page via WebSocket.

For this you need:

* A Raspberry Pi with Raspian, internet, SSH, with Node.js installed
* The [pigpio module](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#pigpio)for Node.js
* The [socket.io module](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#socket.io) for Node.js
* 1 x [Breadboard](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#breadboard)
* 3 x [220 Ohm resistor](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#resistor220ohm)
* 1 x [RGB LED](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#RGBLED) (common anode or common cathode)
* 4 x [Female to male jumper wires](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp#jumperWireFemaletoMale)

Click the links in the list above for descriptions of the different components.

**Note:** The resistor you need can be different from what we use depending on the type of LED you use. Most small LEDs only need a small resistor, around 200-500 ohms. It is generally not critical what exact value you use, but the smaller the value of the resistor, the brighter the LED will shine.

Install the pigpio Module

Earlier, we have used the "onoff" module, which works great for just turning on and off. Now we want to set the set the strength of the LEDs, so we need a GPIO Module with a bit more functionality.

We will use the "pigpio" Node.js module, as this allows for PWM.

With PWM we can set the strength of a LED from 0 to 255.

The "pigpio" Node.js module is based on the pigpio C library.

If you are using the "Lite" version of Raspbian, this is most likely not included and must be manually installed.

Update your system package list:

pi@w3demopi:~ $ sudo apt-get update

Install the pigpio C library:

pi@w3demopi:~ $ sudo apt-get install pigpio

Now we can install the "pigpio" Node.js module using npm:

pi@w3demopi:~ $ npm install pigpio

Now the "pigpio" module should be installed and we can use it to interact with the GPIO of the Raspberry Pi.

**Note:** Since the "pigpio" module uses the pigpio C library, it requires root/sudo privileges to access hardware peripherals (like the GPIO).

Building the Circuit

Now it is time to build the circuit on our Breadboard.

If you are new to electronics, we recommend you turn off the power for the Raspberry Pi. And use an anti-static mat or a grounding strap to avoid damaging it.

Shut down the Raspberry Pi properly with the command:

pi@w3demopi:~ $ sudo shutdown -h now

After the LEDs stop blinking on the Raspberry Pi, then pull out the power plug from the Raspberry Pi (or turn of the power strip it is connected to).

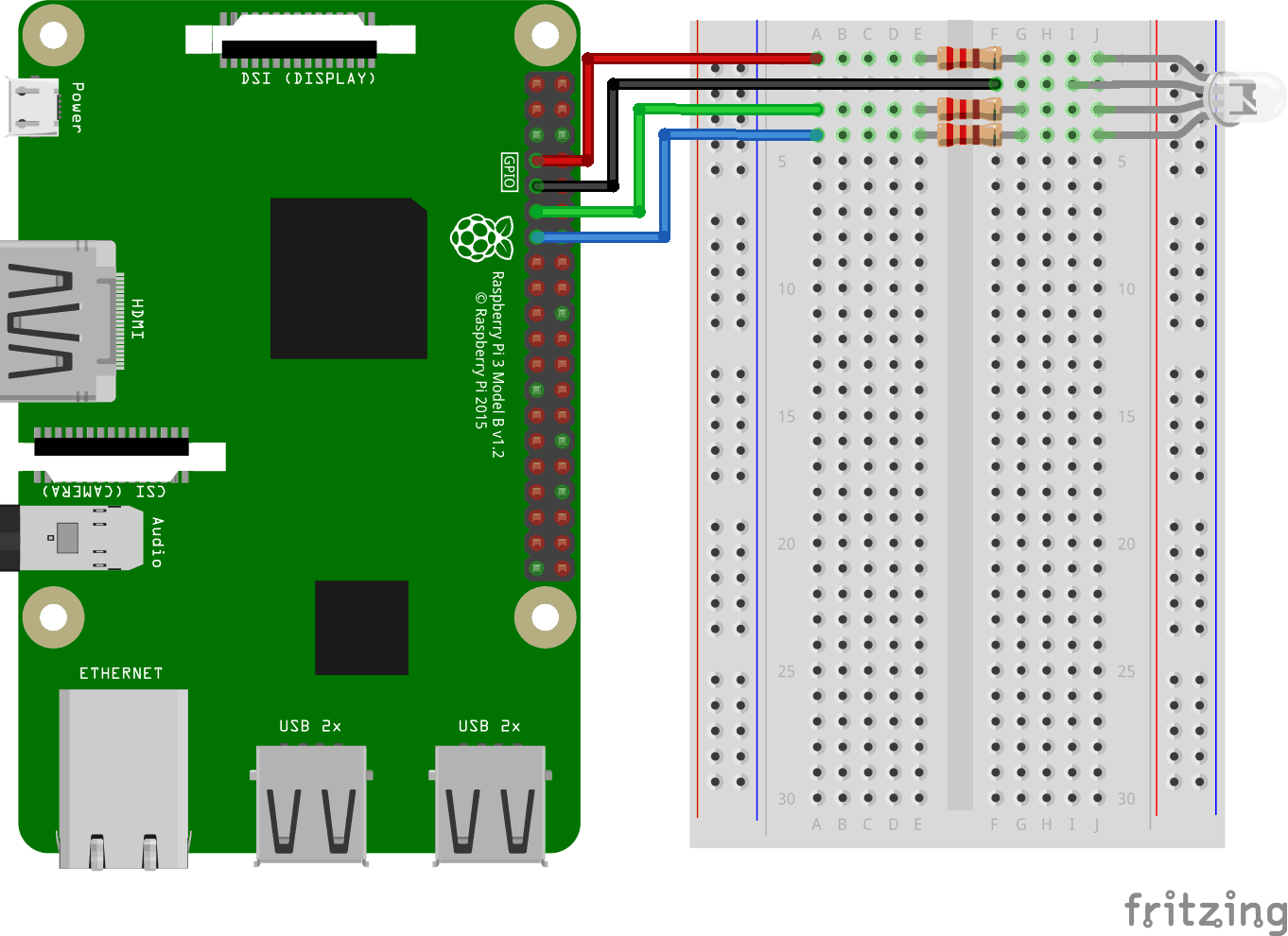
Just pulling the plug without shutting down properly may cause corruption of the memory card.

In building this Circuit it is important to know if you have a common anode, or common cathode, RGB LED:

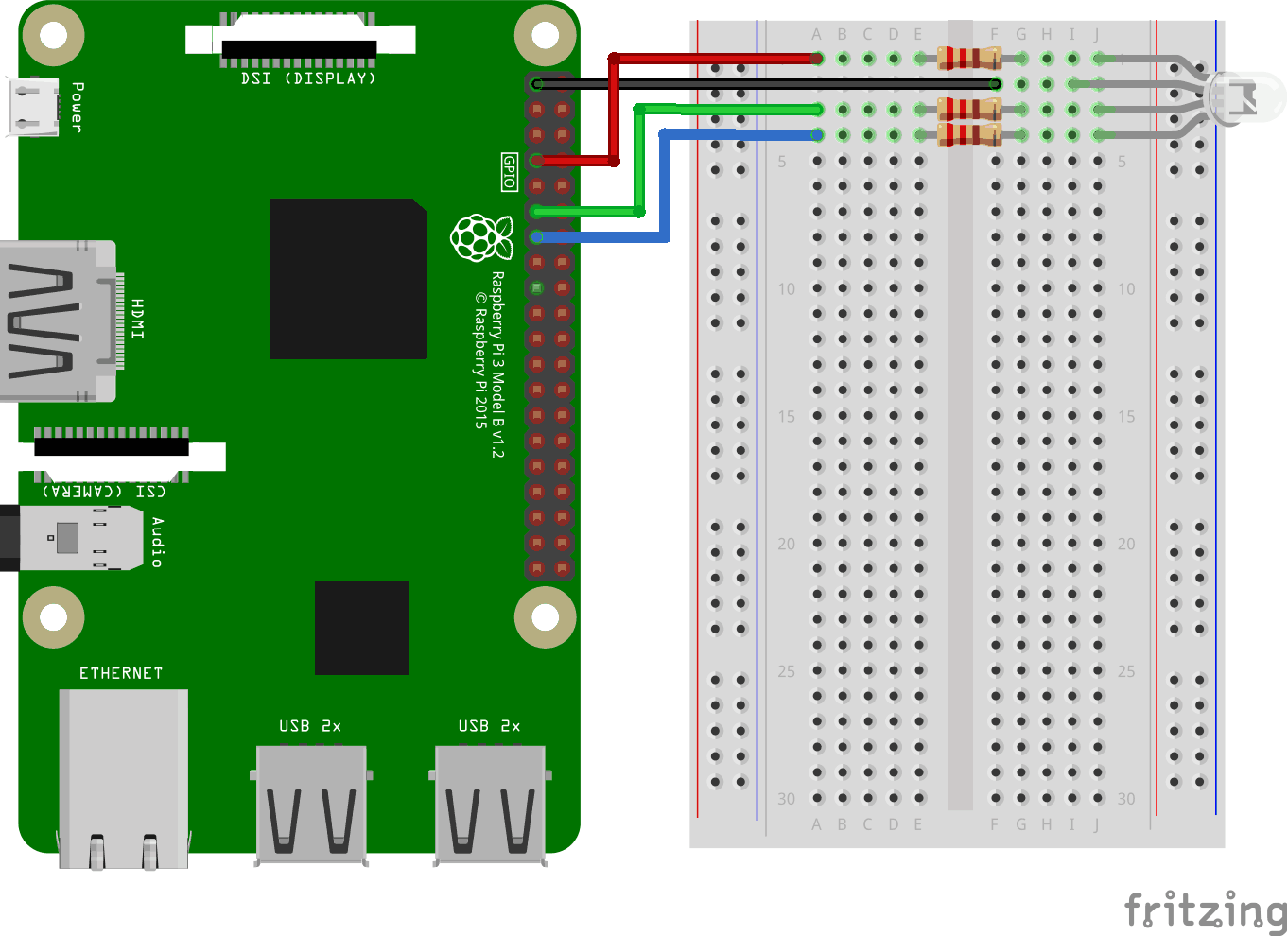
You can check with your provider, or test it yourself:

Connect cables to GND and 3.3V pin. Connect GND to the longest leg of the RGB LED and the 3.3 V to any other leg. If the it lights up, your RGB LED has a common cathode. If not, it has a common anode.

Building the Circuit - Common Cathode



Building the Circuit - Common Anode



Raspberry Pi and Node.js RGB LED and WebSocket Script

Go to the "nodetest" directory, and create a new file called "rgbws.js":

pi@w3demopi:~ $ nano rgbws.js

The file is now open and can be edited with the built in Nano Editor.

Using a Common Cathode RGB LED

Using a Common Anode RGB LED

Raspberry Pi and Node.js WebSocket UI

Now it is time add the HTML that allows for user input via WebSocket.

https://youtu.be/NS3EymTXZm8

For this we want:

* 3 color sliders, one for each color (RGB)
* A color picker
* A div showing the current color

Go to the folder "public":

pi@w3demopi:~/nodetest $ cd public

And create a HTML file, rgb.html:

pi@w3demopi:~/nodetest/public $ nano rgb.html

rgb.html:

<!DOCTYPE html>  
<html>  
<meta name="viewport" content="width=device-width, initial-scale=1">  
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">  
<style>  
.slider {  
  -webkit-appearance: none;  
  width: 100%;  
  height: 15px;  
  border-radius: 5px;  
  background: #d3d3d3;  
  outline: none;  
  opacity: 0.7;  
  -webkit-transition: .2s;  
  transition: opacity .2s;  
}  
  
.slider:hover {opacity: 1;}  
  
.slider::-webkit-slider-thumb {  
  -webkit-appearance: none;  
  appearance: none;  
  width: 25px;  
  height: 25px;  
  border-radius: 50%;  
  cursor: pointer;  
}  
  
.slider::-moz-range-thumb {  
  width: 25px;  
  height: 25px;  
  border-radius: 50%;  
  background: #4CAF50;  
  cursor: pointer;  
}  
#redSlider::-webkit-slider-thumb {background: red;}  
#redSlider::-moz-range-thumb {background: red;}  
#greenSlider::-webkit-slider-thumb {background: green;}  
#greenSlider::-moz-range-thumb {background: green;}  
#blueSlider::-webkit-slider-thumb {background: blue;}  
#blueSlider::-moz-range-thumb {background: blue;}  
</style>  
<body>  
  
<div class="w3-container">  
<h1>RGB Color</h1>  
<div class="w3-cell-row">  
<div class="w3-container w3-cell w3-mobile">  
<p><input type="range" min="0" max="255" value="0" class="slider" id="redSlider"></p>  
<p><input type="range" min="0" max="255" value="0" class="slider" id="greenSlider"></p>  
<p><input type="range" min="0" max="255" value="0" class="slider" id="blueSlider"></p>  
</div>  
<div class="w3-container w3-cell w3-mobile" style="background-color:black" id="colorShow">  
<div></div>  
</div>  
</div>  
<p>Or pick a color: <input type="color" id="pickColor"></p>  
</div>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.3/socket.io.js"></script>  
<script src="https://www.w3schools.com/lib/w3color.js"></script>  
<script>  
var socket = io(); //load socket.io-client and connect to the host that serves the page  
var rgb = w3color("rgb(0,0,0)"); //we use the w3color.js library to keep the color as an object  
window.addEventListener("load", function(){ //when page loads  
  var rSlider = document.getElementById("redSlider");  
  var gSlider = document.getElementById("greenSlider");  
  var bSlider = document.getElementById("blueSlider");  
  var picker = document.getElementById("pickColor");  
  
  rSlider.addEventListener("change", function() { //add event listener for when red slider changes  
    rgb.red = this.value; //update the RED color according to the slider  
    colorShow.style.backgroundColor = rgb.toRgbString(); //update the "Current color"  
    socket.emit("rgbLed", rgb); //send the updated color to RGB LED via WebSocket  
  });  
  gSlider.addEventListener("change", function() { //add event listener for when green slider changes  
    rgb.green = this.value; //update the GREEN color according to the slider  
    colorShow.style.backgroundColor = rgb.toRgbString(); //update the "Current color"  
    socket.emit("rgbLed", rgb); //send the updated color to RGB LED via WebSocket  
  });  
  bSlider.addEventListener("change", function() { //add event listener for when blue slider changes  
    rgb.blue = this.value;  //update the BLUE color according to the slider  
    colorShow.style.backgroundColor = rgb.toRgbString(); //update the "Current color"  
    socket.emit("rgbLed", rgb); //send the updated color to RGB LED via WebSocket  
  });  
  picker.addEventListener("input", function() { //add event listener for when colorpicker changes  
    rgb.red = w3color(this.value).red; //Update the RED color according to the picker  
    rgb.green = w3color(this.value).green; //Update the GREEN color according to the picker  
    rgb.blue = w3color(this.value).blue; //Update the BLUE color according to the picker  
    colorShow.style.backgroundColor = rgb.toRgbString();  //update the "Current color"  
    rSlider.value = rgb.red;  //Update the RED slider position according to the picker  
    gSlider.value = rgb.green;  //Update the GREEN slider position according to the picker  
    bSlider.value = rgb.blue;  //Update the BLUE slider position according to the picker  
   socket.emit("rgbLed", rgb);  //send the updated color to RGB LED via WebSocket  
  });  
});  
</script>  
  
</body>  
</html>

Return to the "nodetest" folder:

pi@w3demopi:~/nodetest $ cd ..

Run the code:

pi@w3demopi:~ $ sudo node rgbws.js

**Note:** Since the "pigpio" module uses the pigpio C library, it requires root/sudo privileges to access hardware peripherals (like the GPIO).

Open the website in a browser using http://[RaspberryPi\_IP]:8080/

Now the RGB LED should change color depending on the user input.

End the program with Ctrl+c.

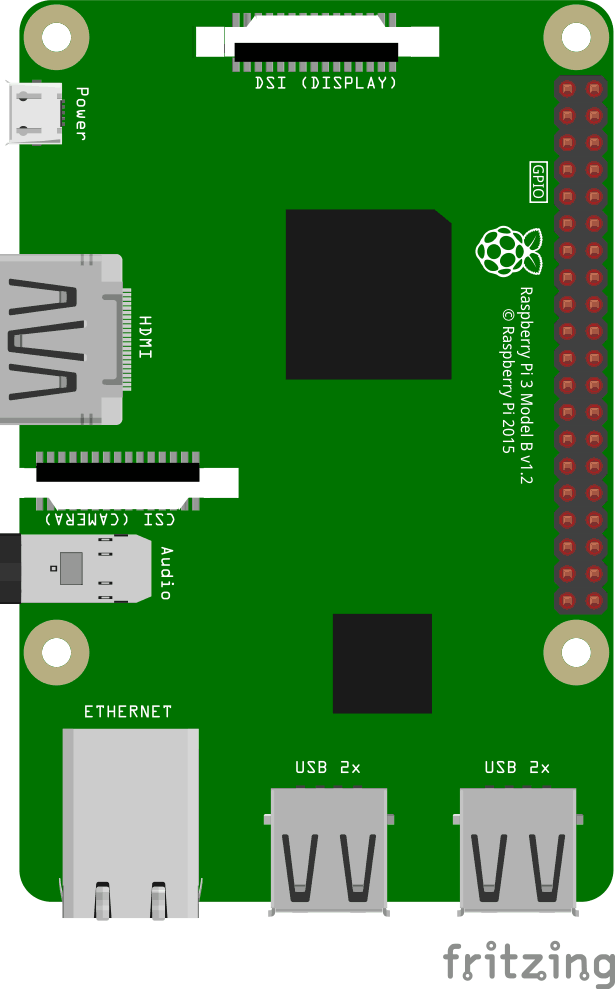
Node.js Raspberry Pi - Components

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_raspberrypi_rgb_let_websocket.asp)[Next ❯](https://www.w3schools.com/nodejs/ref_modules.asp)

What are Components?

Components are parts of a larger whole. In this chapter, we explain the different components we use in our tutorial.

The Raspberry Pi and GPIO Pins



This is an illustration of the Raspberry Pi 3.

The GPIO pins are the small red squares in two rows on the right side of the Raspberry Pi, on the actual Raspberry Pi they are small metal pins.

Input pins are like switches that you can turn on or off from the outside world (like a on/off light switch).

Output pins are like switches that the Raspberry Pi can turn on or off (like turning on/off a LED light).

The Raspberry Pi 3 has 26 GPIO pins, the rest of the pins are power, ground or "other".

The pin placements correspond with the table below.

Raspberry Pi B+, 2, 3 & Zero

|  |  |  |  |
| --- | --- | --- | --- |
| 3V3 | 1 | 2 | 5V |
| GPIO 2 | 3 | 4 | 5V |
| GPIO 3 | 5 | 6 | GND |
| GPIO 4 | 7 | 8 | GPIO 14 |
| GND | 9 | 10 | GPIO 15 |
| GPIO 17 | 11 | 12 | GPIO 18 |
| GPIO 27 | 13 | 14 | GND |
| GPIO 22 | 15 | 16 | GPIO 23 |
| 3V3 | 17 | 18 | GPIO 24 |
| GPIO 10 | 19 | 20 | GND |
| GPIO 9 | 21 | 22 | GPIO 25 |
| GPIO 11 | 23 | 24 | GPIO 8 |
| GND | 25 | 26 | GPIO 7 |
| DNC | 27 | 28 | DNC |
| GPIO 5 | 29 | 30 | GND |
| GPIO 6 | 31 | 32 | GPIO 12 |
| GPIO 13 | 33 | 34 | GND |
| GPIO 19 | 35 | 36 | GPIO 16 |
| GPIO 26 | 37 | 38 | GPIO 20 |
| GND | 39 | 40 | GPIO 21 |

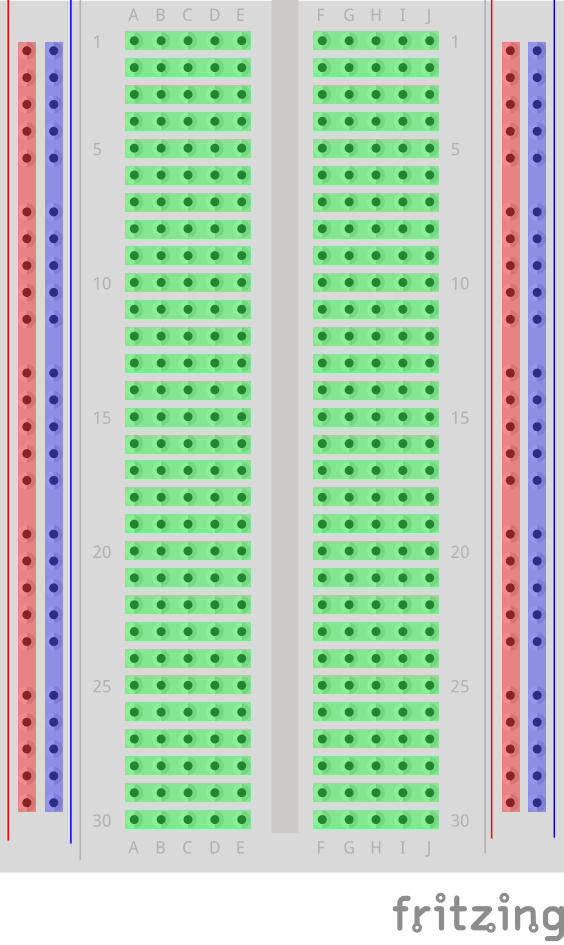
Legend

|  |
| --- |
| Physical Pin Number |
| Power + |
| Ground |
| UART |
| I2C |
| SPI |
| GPIO |
| Do Not Connect |

The Breadboard

A breadboard is used for prototyping electronics, it allows you to create circuits without soldering. It is basically a plastic board, with a grid of tie-points (holes). Inside the board there are metal strips connecting the different tie-points in specific ways.

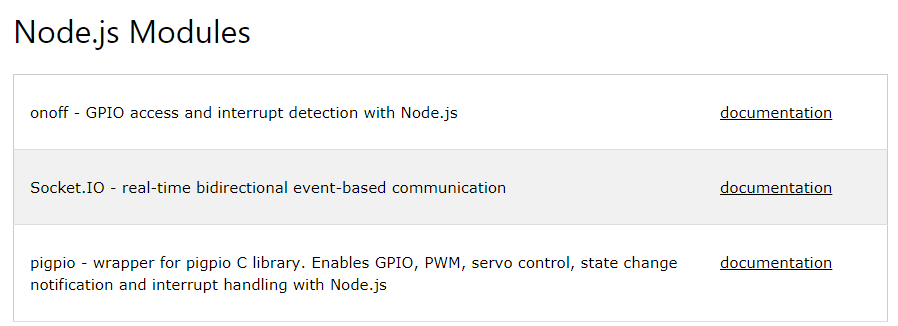
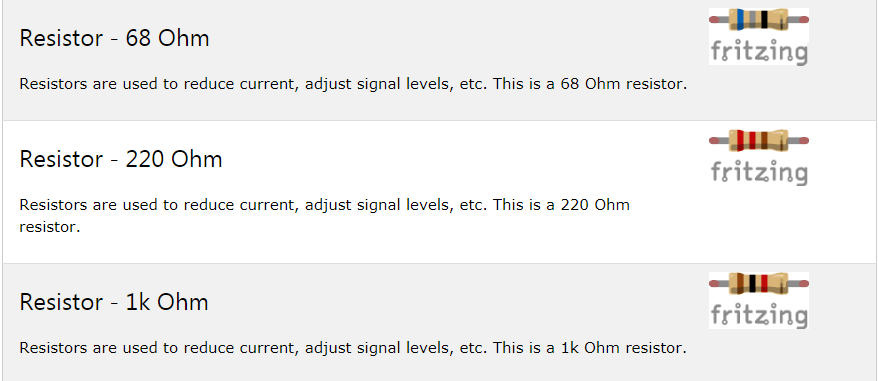
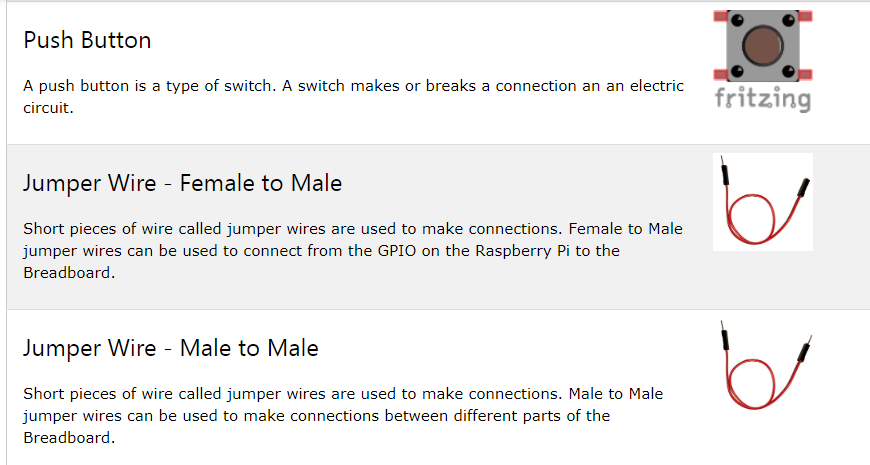
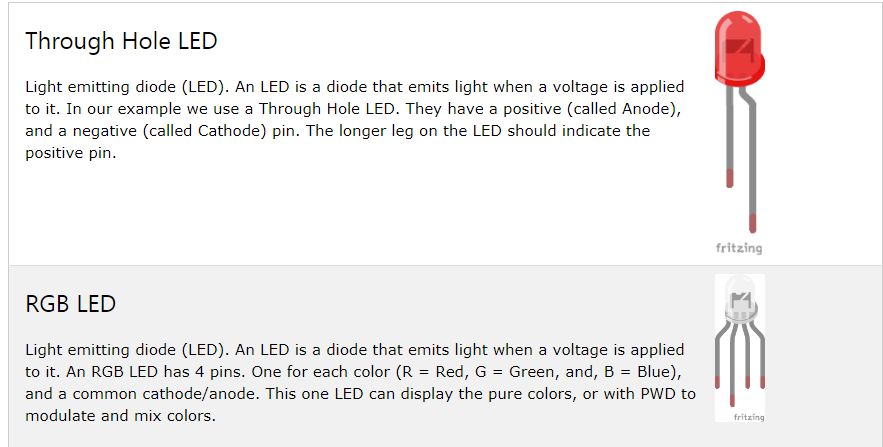
In the illustration below we have highlighted some of the sections with different colors. This is to show you how the grid is connected.



The different sections of the breadboard:

* On the left, and right, side there are 2 columns of tie-points. All the tie points in each of these columns are connected.
* The Power Bus - The columns highlighted with red. There are usually used to connect power to the Breadboard. Since the entire column is connected, you can connect power to any of the tie-points in the column.
* The Ground Bus - The columns highlighted with blue. There are usually used to connect Ground to the Breadboard. Since the entire column is connected, you can connect ground to any of the tie-points in the column.
* Rows of connected Tie-Points - The rows highlighted with green. The tie-points of each of these rows are connected, but not the entire row! The left side tie-points are connected (A-B-C-D-E), and the right side tie-points are connected (F-G-H-I-J).
* In the center of the Breadboard there is a Trench, this separates the left and right rows. The width of the trench is designed so that many Integrated Circuits fit across it.

Other Electrical Components



Node.js Built-in Modules

[❮ Previous](https://www.w3schools.com/nodejs/nodejs_raspberrypi_components.asp)[Next ❯](https://www.w3schools.com/nodejs/default.asp)

Node.js has a set of built-in modules which you can use without any further installation.

Here is a list of the built-in modules of Node.js version 6.10.3:

|  |  |  |  |
| --- | --- | --- | --- |
| MODULE | | DESCRIPTION | |
| assert | | Provides a set of assertion tests | |
| buffer | | To handle binary data | |
| Child\_process | | To run a child process | |
| cluster | | To split a single Node process into multiple processes | |
| [crypto](https://www.w3schools.com/nodejs/ref_crypto.asp) | | To handle OpenSSL cryptographic functions | |
| [dgram](https://www.w3schools.com/nodejs/ref_dgram.asp) | | Provides implementation of UDP datagram sockets | |
| [dns](https://www.w3schools.com/nodejs/ref_dns.asp) | | |  | | --- | | To do DNS lookups and name resolution functions | |  | | |
| domain | | Deprecated. To handle unhandled errors | |
| [events](https://www.w3schools.com/nodejs/ref_events.asp) | | To handle events | |
| [fs](https://www.w3schools.com/nodejs/ref_fs.asp) | | To handle the file system | |
| [http](https://www.w3schools.com/nodejs/ref_http.asp) | | To make Node.js act as an HTTP server | |
| [https](https://www.w3schools.com/nodejs/ref_https.asp) | | To make Node.js act as an HTTPS server. | |
| |  | | --- | | [net](https://www.w3schools.com/nodejs/ref_net.asp) | | | To create servers and clients | |
| |  | | --- | | [os](https://www.w3schools.com/nodejs/ref_os.asp) | | | Provides information about the operation system | |
| [path](https://www.w3schools.com/nodejs/ref_path.asp) | | To handle file paths | |
| punycode | | Deprecated. A character encoding scheme | |
| [querystring](https://www.w3schools.com/nodejs/ref_querystring.asp) | | To handle URL query string | |
| [readline](https://www.w3schools.com/nodejs/ref_readline.asp) | | To handle readable streams one line at the time | |
| [stream](https://www.w3schools.com/nodejs/ref_stream.asp) | | To handle streaming data | |
| [string\_decoder](https://www.w3schools.com/nodejs/ref_string_decoder.asp) | | To decode buffer objects into strings | |
| [timers](https://www.w3schools.com/nodejs/ref_timers.asp) | | To execute a function after a given number of milliseconds | |
| [tls](https://www.w3schools.com/nodejs/ref_tls.asp) | | To implement TLS and SSL protocols | |
| tty | | Provides classes used by a text terminal | |
| [url](https://www.w3schools.com/nodejs/ref_url.asp) | | To parse URL strings | |
| [util](https://www.w3schools.com/nodejs/ref_util.asp) | | To access utility functions | |
| v8 | | To access information about V8 (the JavaScript engine) | |
| [zlib](https://www.w3schools.com/nodejs/ref_zlib.asp) | To compress or decompress files | | |
|  |  | |