1.What is Callback?

A callback is a function that is to be executed after another function has finished executing hence the name 'call back'. More complexly put: In JavaScript, functions are objects. Any function that is passed as an argument is called a callback function.

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action. Here is a quick example: ... The above example is a synchronous callback, as it is executed immediately.

2.What is Callback Hell

Callbacks are just the name of a convention for using JavaScript functions. There isn't a special thing called a 'callback' in the JavaScript language, it's just a convention. Instead of immediately returning some result like most functions, functions that use callbacks take some time to produce a result. The word 'asynchronous', aka 'async' just means 'takes some time' or 'happens in the future, not right now'. Usually callbacks are only used when doing I/O, e.g. downloading things, reading files, talking to databases, etc.

How do I resolve the resolve of callback hell?

a. Keep your code shallow

## b. Modularize

## c. Handle every single error

## 3. What are promises? Advantages of Promises?

## Promises. A promise is an object that wraps an asynchronous operation and notifies when it's done. This sounds exactly like callbacks, but the important differences are in the usage of Promises.

## Advantages:

In my view, there are four key advantages that the promises have over simple callback handlers. They are:

* Better definition of control flow of asynchronous logic
* Reduced coupling
* Better error handling
* Improved readability

## JavaScript promises let your async call return a value like synchronous function, that value is an object that promises success or failure value. This small change make it really powerful to use.

## 4. Different Promise Libraries and difference between each of them?

The core idea behind promises is that a promise represents the result of an asynchronous operation. A promise is in one of three different states:

* pending - The initial state of a promise.
* fulfilled - The state of a promise representing a successful operation.
* rejected - The state of a promise representing a failed operation.

Once a promise is fulfilled or rejected, it is immutable (i.e. it can never change again).

## Constructing a promise

Once all of the APIs return promises, it should be relatively rare that you need to construct one by hand. In the meantime, we need a way to polyfill existing APIs. For example:

function readFile(filename, enc){

return new Promise(function (fulfill, reject){

fs.readFile(filename, enc, function (err, res){

if (err) reject(err);

else fulfill(res);

});

});}

**Bluebird Promise Library:**

It is a fully-featured Promise library for JavaScript. The strongest feature of Bluebird is that it allows you to "promisify" other Node modules in order to use them asynchronously. Promisify is a concept applied to callback functions. This concept is used to ensure that every callback function which is called returns some sort of value.

So if a Node JS module contains a callback function which does not return a value, if we Promisify the node module, all the function's in that specific node module would automatically be modified to ensure that it returns a value.

So you can use BlueBird to make the[MongoDB](https://www.guru99.com/mongodb-tutorials.html)module run asynchronously. This just adds another level of ease when writing Node.js applications.

**Q promises Library**

The Q module can be loaded as:

* A <script> tag (creating a Q global variable):

It provides a fail shorthand for then when you are only interested in handling the error:

Example:

var output Promise = getInputPromise()

.fail(function (error) {

});

5. Understanding Async/Await?

There’s a special syntax to work with promises in a more comfortable fashion, called “async/await”. It’s surprisingly easy to understand and use.

## [Async functions](https://javascript.info/async-await" \l "async-functions)

Let’s start with the async keyword. It can be placed before a function, like this:

async function f() {

return 1;

}

The word “async” before a function means one simple thing: a function always returns a promise. If the code has return <non-promise> in it, then JavaScript automatically wraps it into a resolved promise with that value.

For instance, the code above returns a resolved promise with the result of 1, let’s test it:

async function f() {

return 1;

}

f().then(alert);

Similarly we could explicitly return a promise, that would be the same:

async function f() {

return Promise.resolve(1);

}

f().then(alert);

So, async ensures that the function returns a promise, and wraps non-promises in it. Simple enough, right? But not only that. There’s another keyword, await, that works only inside async functions, and it’s pretty cool.

**Await**:

async function f() {

let promise = new Promise((resolve, reject) => {

setTimeout(() => resolve("done!"), 1000)

});

let result = await promise; // wait till the promise resolves (\*)

alert(result); // "done!"

}

f();

await literally makes JavaScript wait until the promise settles, and then go on with the result. That doesn’t cost any CPU resources, because the engine can do other jobs meanwhile: execute other scripts, handle events etc.

6.What is NPM? List some popular packages and describe each package.

**npm** is the world's largest **Software Registry**.

The registry contains over 800,000 **code packages**.

**Open-source** developers use **npm** to **share** software.

Many organizations also use npm to manage private development.

Some packages in npm are:

1.Express:

All common programming languages share similar structures in the way things are built, one of the fastest ways to get a programming language to serve your needs, is through a framework. Express is the leading Node.js framework for quickly creating and publishing applications, and APIs. The frameworks minimal structure allows any Node.js developer to quickly launch a functional application with the use of Express Generator. The rest is up to you, as Express gives you a solid outline to build your apps on top of. Combine it with any of the other packages we will discuss, and you will quickly realize just how amazing this framework truly is.

2. Asynchronous JavaScript

Asynchronous JavaScript has rebuilt the way JavaScript content interacts with your web pages, allowing you to increase performance by getting rid of render-blocking JavaScript. Render-blocking basically means that any JavaScript content that resides in the “above the fold” parts of a page will not be loaded, until JavaScript itself has been finished loading in the page. This can have certain effects on the performance of your website, but with the helps of libraries such as Async.js — you can open the door to load JavaScript content alongside everything else, sometimes even faster. The library is meant to be used with Node.js, originally, but these days it will work with browsers too, so you can inject it in any project, whether it uses Node.js or not. There are more than twenty functions that this library gives you control over.

3. Request – Simplified HTTP Client

Apps constantly have to deal with external HTTP calls, whether to fetch data, or to load data directly from an external source. Having a reliable way to do HTTP calls is essential, which is why Request has grown to be the most reliable HTTP Client library of them all. With Request, making HTTP calls becomes an easy, secure, and reliable way of accessing external content on the web, in any form imaginable. It works well with authentication methods too.

4. Browserify,

Working with JavaScript libraries can prove to be a difficult process, you constantly have to go back and forth between templates to analyze which libraries and widgets are using the specific scripts, and eventually these scripts begin to add up as you start to lose track of what is working for what.

As JavaScript progresses, the notion that we are just building websites, well.. is gone. In many ways, websites built with JS these days act more like apps — versatile projects that depend on heavy libraries that work independently. There are many uses for Browserify, but the most important one is that Browserify allows us to create the kind of client-side codebase that’s organized, structured, and easy to navigate. Anyone can use Browserify to create solid components that quickly allocate dependencies according to their use at any particular situation. Node.js projects tend to stack up on packages, libraries and scripts, and Browserify helps to structure everything neatly together.

5.Grunt:

Task runners can be avoided for as long as you like, but learning one is going to completely change your programming experience, from day one. With a task runner, you can easily allow yourself to have less tasks to care of for a particular project, and instead automate the process of doing minifications, compile tasks, testing, code linting, and so forth. The less of these you have to do individually on your own, the more time you have to spare for doing actual coding work. Grunt provides a ton of plugins that you can use to automate even the most mundane tasks, without any extra effort from your side. And if you feel that there isn’t a plugin that does what you need, well.. just go ahead and create your own, Grunt lets you publish Grunt-specific plugins through NPM.

6. Socket

Socket lets you build truly real-time communication apps that would require real-time streams of content, whether directly from the data that you are working with, or through an API that comes from an external source. Some example apps that we have seen is a Twitter bot for collecting latest tweets, a Facebook bot for watching news, and other interesting combinations of APIs that work with data in real-time. Just imagine what such communication methods could do for your analytics, truly real-time analytics are still being actively developed even by large companies such as Google Analytics, but with Socket — you just happen to get early access to all that.

7.Mocha

Even more asynchronous action going on here in this Node.js package roundup, this time we have Mocha — a feature-rich JavaScript test framework running on Node.js and the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Testing is so important to understand how well the application is performing, where we can locate any particular leaks, and also to know how we can improve these bugs, problems, and irritations that we experience, testing lets developers to understand better how their code performs, and in turn learn more skills as they continue down their chosen path.

8.Bower:

In (very) old days, websites didn’t consist of much more than HTML and CSS, and in some cases JavaScript. Nowadays, websites almost can’t exist without the use of external libraries, tools, frameworks, and other JS related utilities that make a modern website. Having to keep track of all these utilities can turn into a strenuous task, there is so much to control and keep in check.. it only makes sense that package managers such as Bower exist. Bower manages your components for you, if they are of JS, CSS, or HTML nature. (Fonts, and visual content works too!) Bower carefully allocates all the packages that you are using, and then helps you to keep the updated, and regularly checked against any potential risks. A simple Bower file can help you maintain an application the size of Fortune 500 companies.