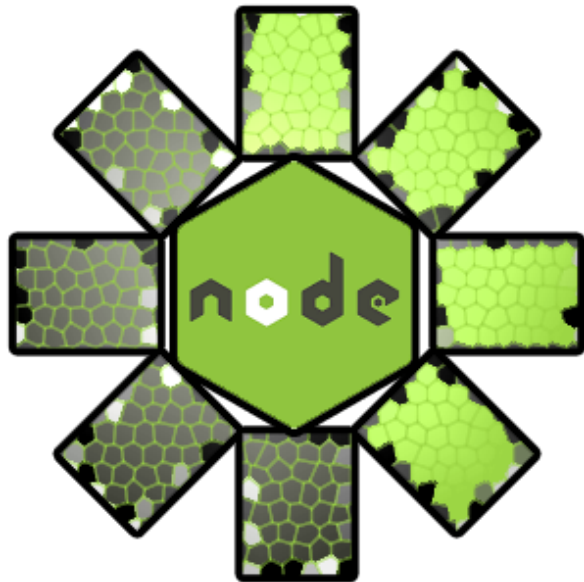


# JavaScript Callbacks vs Promises vs Async Await



JavaScript is a powerful programming language with its ability for closure, first class functions,

and many other features. JavaScript is often used for Asynchronous Programming, or programming in a style that uses callbacks. While powerful, this can lead to what many now refer to as **Callback Hell**. Callback hell is also affectionately referred to as the Pyramid Of Doom thanks to the many levels of indentation which make your code look like a difficult to read pyramid. Let's review the evolution of asynchronous programming in JavaScript from callbacks to async/await as this will help us in our Node.js programming.

## What are JavaScript Callbacks Used For?

Callback functions in JavaScript are functions that can be passed to other functions as a parameter. They are also known as higher-order functions and are called (or executed) inside the function they were passed to. A callback is a convention, not an actual thing in the JavaScript language. **Callbacks** are used when you need to do things in your code which result in I/O, or input/output operations. Things like talking to a database, downloading a photo, or writing a file to disk are examples of I/O operations. These are asynchronous operations. In other words, they may take some time, or happen in the future. Check out an **example callback function**.

Let's look at a code example here.

**P-SILVER CORE**  
INTEL® XEON® SILVER 4110

1 CPU (8C/16T) @2.1 GHZ

1 GBPS BANDWIDTH

2 x 240 GB SSD

64 GB RAM DDR4

FROM **€169<sup>99</sup>** / MTH  
EX. VAT

**DISCOVER**

ikoula we host with care

NO COMMITMENT

www.ikoula.com

Advertise Here



[Node.js Express Rest Api Tutorial](#)



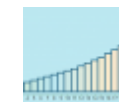
[How To Use wp\\_list\\_pages\(\) To Create Parent and Child Page Menus](#)



[Flarum Forum Software](#)



[Laravel Eloquent ORM Tutorial](#)



[Create A Bar Chart With D3 JavaScript](#)



[The Most Awesome PHP Script to Highlight Your Code](#)



[ES6 Promises Tutorial](#)



[The Top 9 Most Popular PHP String Functions](#)



[D3 DOM Manipulation And Data Binding With D3 JavaScript](#)



[Add Featured Image Support To Your WordPress Theme](#)



[Python Dictionaries](#)

```
1 const getBlogPost = () => {
2   setTimeout(() => {
3     return {
4       title: 'JavaScript Callbacks'
5     }
6   }, 2000);
7 };
8
9 const blogpost = getBlogPost();
10 console.log(blogpost.title);
```

What do you think will happen when this code is run? Let's see.

```
express-rest $node index.js
C:\node\express-rest\index.js:21
console.log(blogpost.title);
               ^
TypeError: Cannot read property 'title' of undefined
    at Object. (C:\node\express-rest\index.js:21:22)
    at Module._compile (module.js:643:30)
    at Object.Module._extensions..js (module.js:654:10)
    at Module.load (module.js:556:32)
    at tryModuleLoad (module.js:499:12)
    at Function.Module._load (module.js:491:3)
    at Function.Module.runMain (module.js:684:10)
    at startup (bootstrap_node.js:187:16)
    at bootstrap_node.js:608:3
```

The application says it can not read a property of undefined. But why? JavaScript does not wait for the async operation to complete to keep moving forward through the code execution. It simply runs each line of the program with no stopping. In other words it is non blocking. The getBlogPost() function runs immediately and sets the value of **blogpost** to undefined. This is why callbacks are needed when programming JavaScript and Node. We can rewrite this example using a callback so we get the result we want.

```
1 const getBlogPost = (callbackfunc) => {
2   setTimeout(() => {
3     callbackfunc({
4       title: 'JavaScript Callbacks'
5     });
6   }, 2000);
7 };
8
9 getBlogPost((blogpost) => {
10   console.log(blogpost.title);
11 })
```



[Introduction To AngularJS](#)



[Angular Root Component](#)

[Tutorial](#)



[Making Favorites Part Of The Activity Feed](#)



[What Is Middleware In Laravel 5?](#)



[JavaScript Functions Tutorial](#)



[Express.js Beginner Tutorial](#)



[Underscore JS Map Function](#)



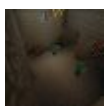
[An Example of JavaScript Closure Using The Date Object](#)



[Php Tutorials For Beginners](#)



[What is a MySQL Join?](#)



[Escape Strings For MySQL To Avoid SQL Injection](#)



[VueJS Bootstrap Pagination Component](#)



[Laravel Vue Component Example](#)



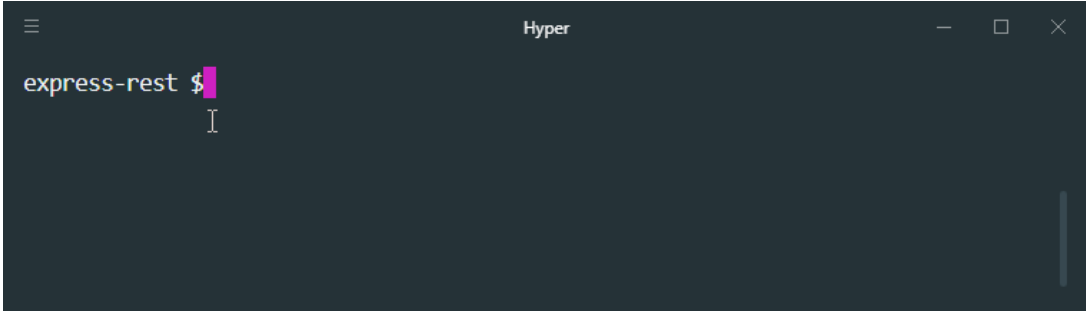
[ES6 Rest Parameters and Spread Operators](#)



[CSS Position Tutorial](#)



[How To Select Raw SQL Using Eloquent](#)



This is more along the lines of what we are looking for. We start the program, and after 2 seconds we see the output on the screen.

## Callback Hell

Let’s think of a situation that could possibly cause callback hell. Imagine in your application that you would like to fetch a user from the database. Once you have that user, you now want to get all of that user’s blog posts. Once you have the blog posts, you also want to get the comments from that blog post. In a synchronous approach this is pretty straight forward. In the asynchronous world, this can start to cause some problems. The highlighted code here is what we refer to as the [Pyramid Of Doom](#)! You could also call it callback hell, but pyramid of doom just sounds so killer, haha!



[Send a Tweet with Laravel](#)



[How Does The Filter Function Work In Underscore JS?](#)



[What are Getters and Setters?](#)



[Angular Structural Directives](#)



[Laravel File Structure](#)



[Command Types In Linux](#)



[How To Add Search To A WordPress Theme](#)



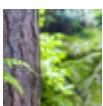
[How To Bind Events In AngularJS](#)



[Setting Up A Database With Seeding](#)



[What Are PHP Arrays?](#)



[What Are Migrations In Laravel?](#)



[Check Authorization With Policies Before Delete Function](#)



[Embedding Expressions In JSX](#)



[Route Model Binding In Laravel](#)



[How To Use View Composers](#)



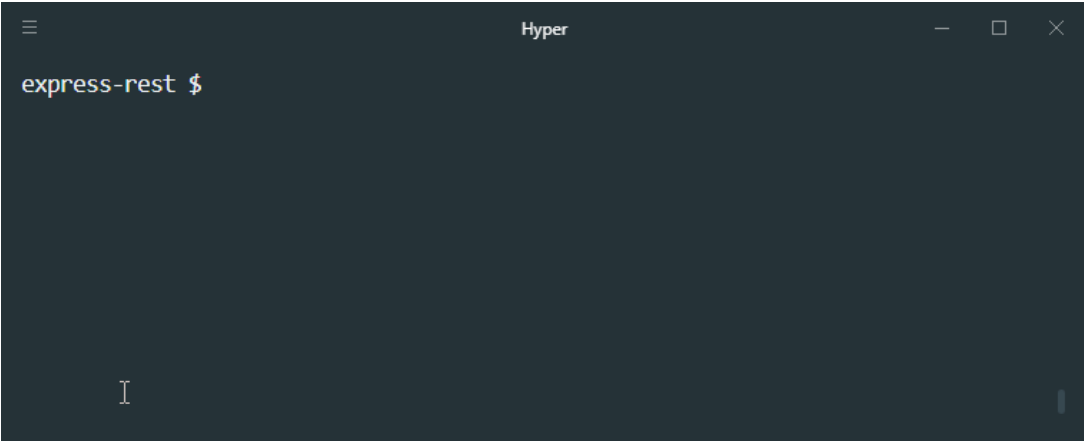
[Liskov Substitution Principle](#)



[How To Create A Feature Test In Laravel](#)












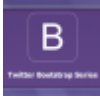

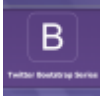



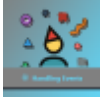
```
1  getUser(1, (user) => {
2    getBlogPosts(user.name, (blogposts) => {
3      getComments(blogposts[0], (comments) => {
4        console.log(user, blogposts[0], comments);
5      })
6    })
7  });
8
9  function getUser(id, callbackfunc) {
10   setTimeout(() => {
11     console.log('Getting the user from the database...');
12     callbackfunc({
13       id: id,
14       name: 'Vegibit'
15     });
16   }, 1000);
17 }
18
19 function getBlogPosts(username, callbackfunc) {
20   setTimeout(() => {
21     console.log('Calling WordPress Rest API for posts');
22     callbackfunc(['Post1', 'post2', 'post3']);
23   }, 1000);
24 }
25
26 function getComments(post, callbackfunc) {
27   setTimeout(() => {
28     console.log('Calling WordPress Rest API for comments');
29     callbackfunc(['comments for ' + post]);
30   }, 1000);
31 }
```

This code works, but it's not that easy to read nor look at. Worse, if you keep indenting further levels of callbacks you certainly will get lost in the code, create bugs, and simply create anti patterns that are not good for you!



# Reducing Callback Hell With Named Functions

JavaScript makes it easy to shoot yourself in the foot if you are not careful. One such feature of the language which helps you do this are anonymous functions. Getify is

- [Node.js Blog Tutorial](#)
- [Drawing Scalable Vector Graphics With D3 JavaScript](#)
- [How To Use VueJS with Laravel Blade](#)
- [Building A Forum With Laracasts](#)
- [D3 DOM Selection With D3 JavaScript](#)
- [Autoloading For Code Organization](#)
- [Angular Table Filter Component](#)
- [Render HTML In Node.js](#)
- [C# Classes And Objects](#)
- [Introduction to MySQL](#)
- [jQuery Selectors and Filters](#)
- [Create a Twitter Bootstrap Page](#)
- [CSS Transitions and Transformations](#)
- [Install Twitter Bootstrap](#)
- [The Most Popular Content Manipulation Methods in jQuery](#)
- [Best Way To Learn Node.js](#)
- [HTML Form Tutorial](#)
- [How To Handle Events In React](#)



known for suggesting that you [always name your functions](#) and he has a point. [Debugging anonymous functions](#) is next to impossible. In any event, let's modify our code so far to use named functions and see how that helps out readability. Here we refactor the code to use named functions. It's better, but still not that great.

```

1  getUser(1, getUserBlogPosts);
2
3  function getUser(id, callbackfunc) {
4      setTimeout(() => {
5          console.log('Getting the user from the database...');
6          callbackfunc({
7              id: id,
8              name: 'Vegibit'
9          });
10     }, 1000);
11 }
12
13 function getUserBlogPosts(user) {
14     getBlogPosts(user.name, getPosts);
15 }
16
17 function displayComments(comments) {
18     console.log(comments);
19 }
20
21 function getPosts(blogposts) {
22     getComments(blogposts[0], displayComments);
23 }
24
25 function getBlogPosts(username, callbackfunc) {
26     setTimeout(() => {
27         console.log('Calling WordPress Rest API for posts');
28         callbackfunc(['Post1', 'post2', 'post3']);
29     }, 1000);
30 }
31
32 function getComments(post, callbackfunc) {
33     setTimeout(() => {
34         console.log('Calling WordPress Rest API for comments');
35         callbackfunc(['comments for ' + post]);
36     }, 1000);
37 }

```

## How [JavaScript Promises](#) Can Help



[C# Methods And Properties](#)



[What Is Guzzle PHP?](#)



[Conditional Rendering In React](#)



[Process Returned MySQL Query Results In PHP](#)



[PHP URL Encode Example](#)



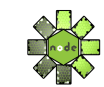
[PHP Simple HTML DOM Parser vs FriendsOfPHP Goutte](#)



[Digging In To HTML Fundamentals](#)



[Reference Types And Value Types In C#](#)

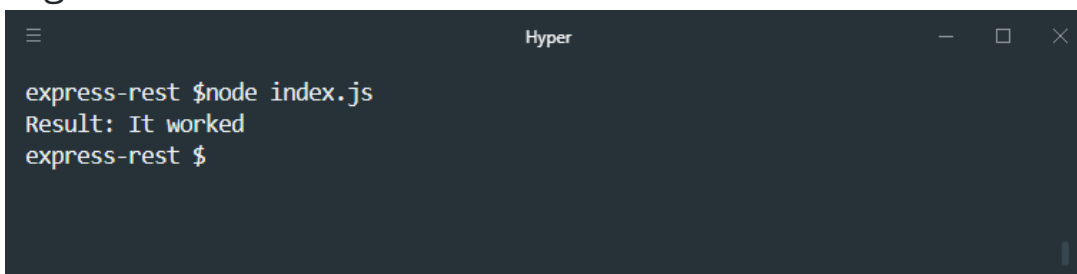


[JavaScript Callbacks vs Promises vs Async Await](#)

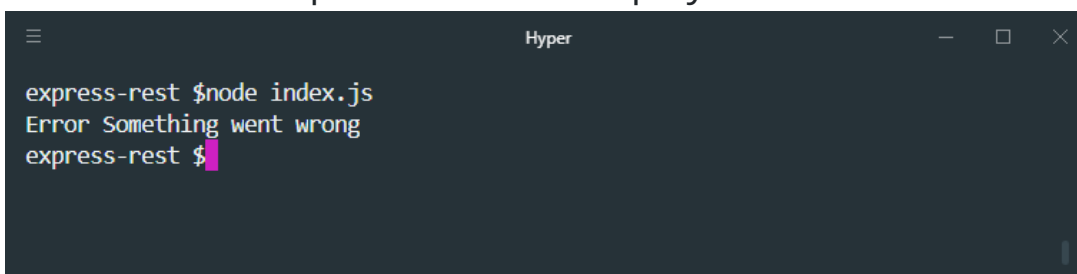
Promises hold the eventual result of an asynchronous operation. When an asynchronous operation completes, it can either result in a value or an error. The three states a promise can hold are Pending, Fulfilled, or Rejected. Here is a fictional example of using a promise to make an API request.

```
1 const promise = new Promise((resolve, reject) => {
2   // Do some asynchronous tasks
3   // ...
4   setTimeout(() => {
5     let apiResponse = 1;
6     if (apiResponse == 1) {
7       resolve('It worked'); // pending => resolved, fulfilled
8     } else {
9       reject(new Error('Something went wrong')); // pending => rejected
10    }
11  }, 1000);
12 });
13
14 promise.then(result => console.log('Result:', result))
15   .catch(err => console.log('Error', err.message));
```

When the API request was successful (apiResponse = 1), all is good.

A terminal window titled 'Hyper' showing the command 'express-rest \$node index.js' and the output 'Result: It worked'. The prompt 'express-rest \$' is visible at the bottom.

When the API request was unsuccessful (apiResponse = 0), the .catch of the promise chain displays an error.

A terminal window titled 'Hyper' showing the command 'express-rest \$node index.js' and the output 'Error Something went wrong'. The prompt 'express-rest \$' is visible at the bottom.

With this understanding, we can now replace asynchronous functions that take a callback, with a returned promise. Let's see how.

## Promises In Place Of Callbacks

In the example code we showed for callback hell, we have the pyramid of doom structure. In other words, we have that deeply nested problem that is hard to read. Here, we can modify the asynchronous functions to now return a promise. Since those async functions return a promise, we can then consume those promises using .then() and .catch() thereby eliminating the callback hell nested pyramid. Note the pyramid of doom is now commented

out, and re written using a `.then()` chain right below. Also note the supporting asynchronous functions no longer accept a callback as a parameter since they are now returning a Promise instead. Hopefully it is clear that inside the returned promise is where the async code now goes (Reading a database, or making an API request).

```

1 // getUser(1, (user) => {
2 //   getBlogPosts(user.name, (blogposts) => {
3 //     getComments(blogposts[0], (comments) => {
4 //       console.log(user, blogposts[0], comments);
5 //     })
6 //   })
7 // });
8
9 getUser(1)
10 .then(user => getBlogPosts(user.name))
11 .then(blogposts => getComments(blogposts[0]))
12 .then(comments => console.log(comments))
13 .catch(err => console.log('Error: ', err.message));
14
15 function getUser(id) {
16   return new Promise((resolve, reject) => {
17     setTimeout(() => {
18       console.log('Getting the user from the database');
19       resolve({
20         id: id,
21         name: 'Vegibit'
22       });
23     }, 1000);
24   });
25 }
26
27 function getBlogPosts(username) {
28   return new Promise((resolve, reject) => {
29     setTimeout(() => {
30       console.log('Calling WordPress Rest API for post');
31       resolve(['Post1', 'post2', 'post3']);
32     }, 1000);
33   });
34 }
35
36 function getComments(post) {
37   return new Promise((resolve, reject) => {
38     setTimeout(() => {
39       console.log('Calling WordPress Rest API for comments');
40       resolve(['comments for ' + post]);
41     }, 1000);
42   });
43 }

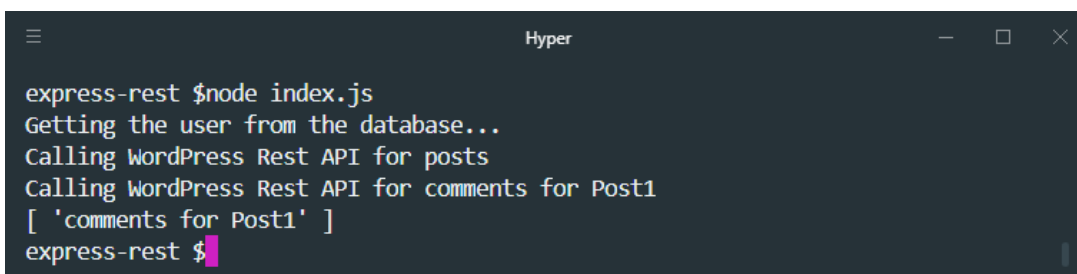
```

# Async Await Syntactic Sugar

Finally, you may consider using the newest solution to callback hell, and that would be [async/await](#). The goal of async/await is to help software developers write asynchronous code as if it were synchronous code, or in a synchronous looking style. To demonstrate, here is the code for both the Promise based and Async/Await approach. Note the Promise approach is commented out.

```
1 // getUser(1)
2 //   .then(user => getBlogPosts(user.name))
3 //   .then(blogposts => getComments(blogposts[0]))
4 //   .then(comments => console.log(comments))
5 //   .catch(err => console.log('Error: ', err.message));
6
7 async function displayComments() {
8   try {
9     const user = await getUser(1);
10    const blogposts = await getBlogPosts(user.name);
11    const comments = await getComments(blogposts[0]);
12    console.log(comments);
13  } catch (err) {
14    console.log('Error', err.message);
15  }
16 }
```

It looks like the code works with the async/await method as well.



```
Hyper
express-rest $node index.js
Getting the user from the database...
Calling WordPress Rest API for posts
Calling WordPress Rest API for comments for Post1
[ 'comments for Post1' ]
express-rest $
```

Some key things to understand here about async/await. You use the **await** keyword when you call a function that is returning a Promise. This allows you to assign the result to a constant or variable just like you would in a synchronous operation. For example the code `const user = await getUser(1);` looks very nice and synchronous, but the `await` keyword gives us an indication that the function is returning a promise. This now facilitates the ability to call `getBlogPosts(user.name)` passing in the user object we just got in the prior line of code. Anytime you see that **await** keyword before a function, you know it is returning a promise. The great thing though is that you can now write that code in a more synchronous looking style. The other takeaway is that since you are using `await` inside of a function, you need to use the **async** keyword on that function. Async and Await are built on top of Promises and are a way to have nicer syntax much like [ES6 Classes](#) are a



nicer syntax for Prototypes and Constructor Functions. If writing asynchronous code makes your head spin, then maybe Async/Await are just what you need.

## Javascript Callbacks vs Promises vs Async Await Summary

When you're writing asynchronous code, it is easy to get yourself into trouble, especially if you've been writing synchronous code for a long time. The truth is, making use of callbacks in a clean and concise way is challenging. You may find yourself in the callback hell pyramid of doom faster than you can blink an eye. You can fight back against call back hell with these tips.

- [Use Modules](#)
- [Use Control Flow Managers](#)
- [Use Promises](#)
- [Use Async Await](#)
- Read a few more tutorials like
  - [Front End Weekly](#)
  - [Tinus Wagner](#)
  - [Hackernoon](#)
  - [Academind](#)
- [Watch a great Traversy Media tutorial](#)

#javascript

← [Node.js Express Rest Api Tutorial](#) | [Install MongoDB With Compass On Windows](#) →

[About](#) [Privacy](#) [Terms](#) [User Site Map](#) © 2013 - 2019 Vegibit.com