Walk-through 006

WRITING ASYNCHRONOUS CODE WITH ASYNC.JS

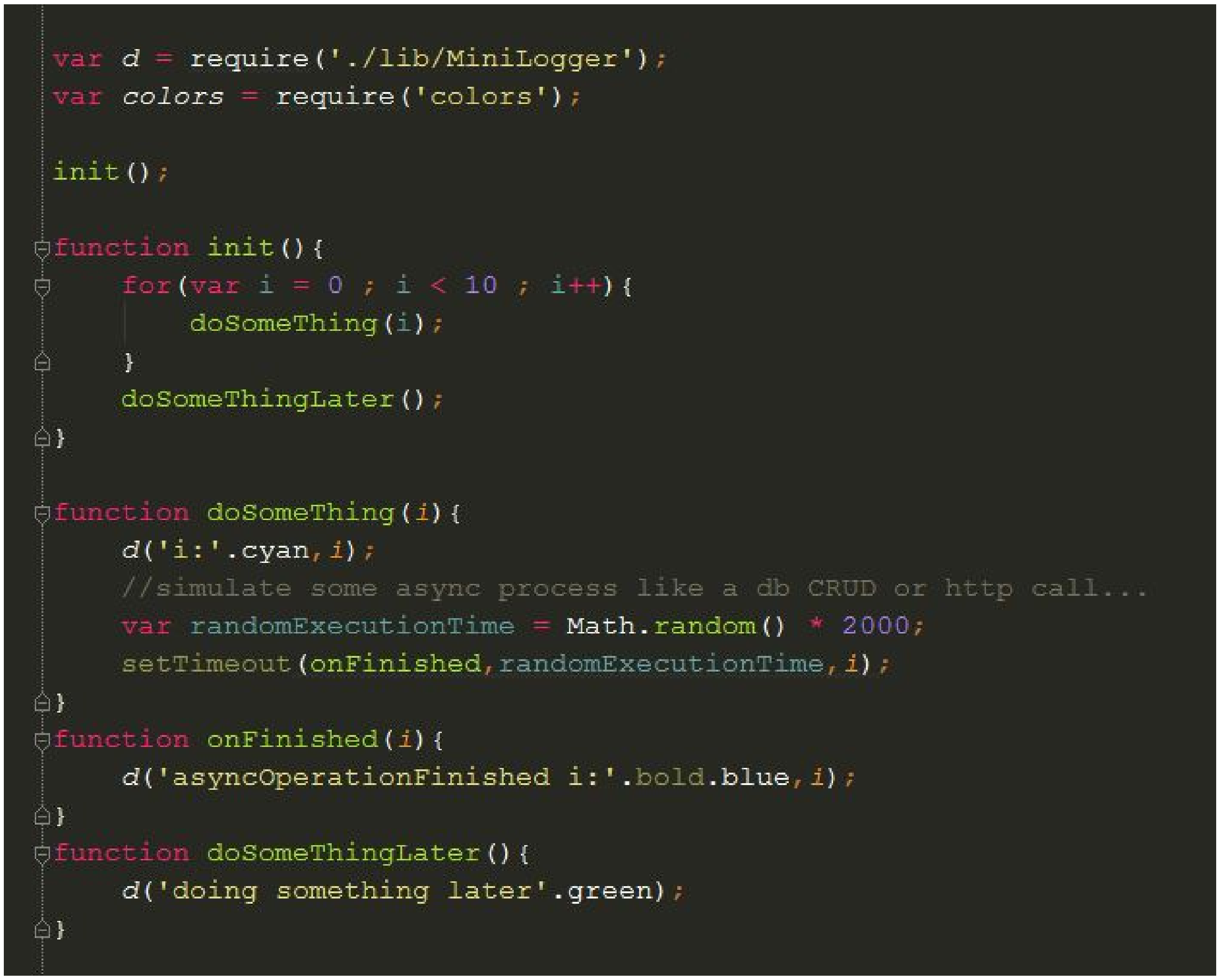
Objectives:

- Avoiding common pitfalls and getting to know one of the most essential packages out there when writing node apps - async.js

Steps:

1. Let's illustrate the problem first

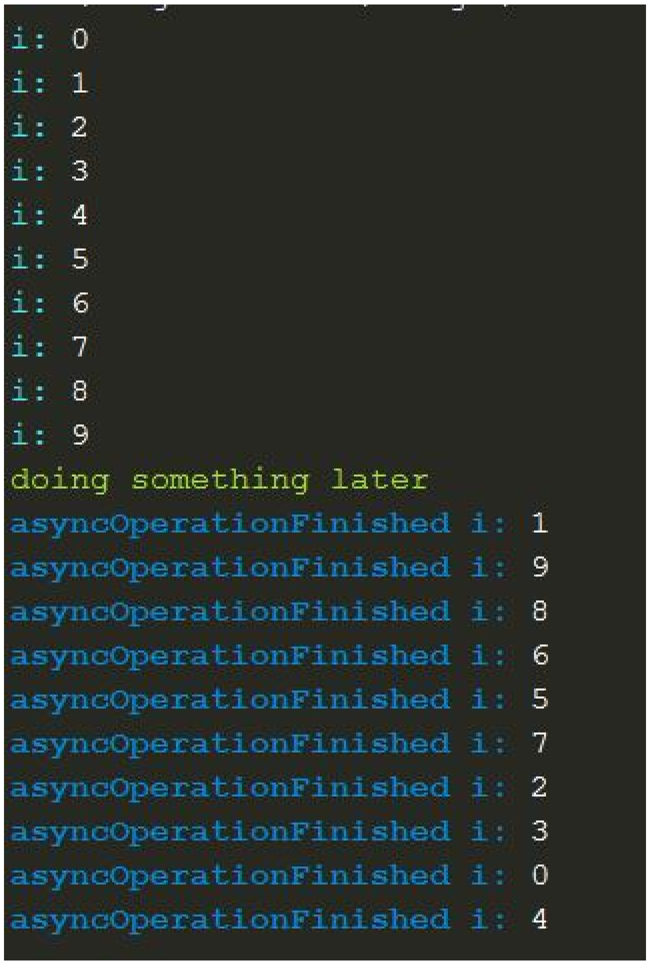
   Write the following code:



```
var d = require('./lib/MiniLogger');
var colors = require('colors');

init();

function init(){
    for(var i = 0 ; i < 10 ; i++){
        doSomeThing(i);
    }
    doSomeThingLater();
}

function doSomeThing(i){
    d('i:'.cyan,i);
    //simulate some async process like a db CRUD or http call...
    var randomExecutionTime = Math.random() * 2000;
    setTimeout(onFinished,randomExecutionTime,i);
}
function onFinished(i){
    d('asyncOperationFinished i:'.bold.blue,i);
}
function doSomeThingLater(){
    d('doing something later'.green);
}
```

Assume that **doSomething()** function stands for an asynchronous process like writing to a database or calling an http request – You do not have any way to guaranty when the process had finished, and you might want to also control the order in which the processes run. It might be meaningful that all of them run one after the other, or in parallel or one-by-one but within a que or another logic. Assume you want to call the **doSomeThingLater()** only after all the async processes had finished, you may have different requirements in case all of the processes completed or some of them failed.

2. Execute your code - The result would be something like this:



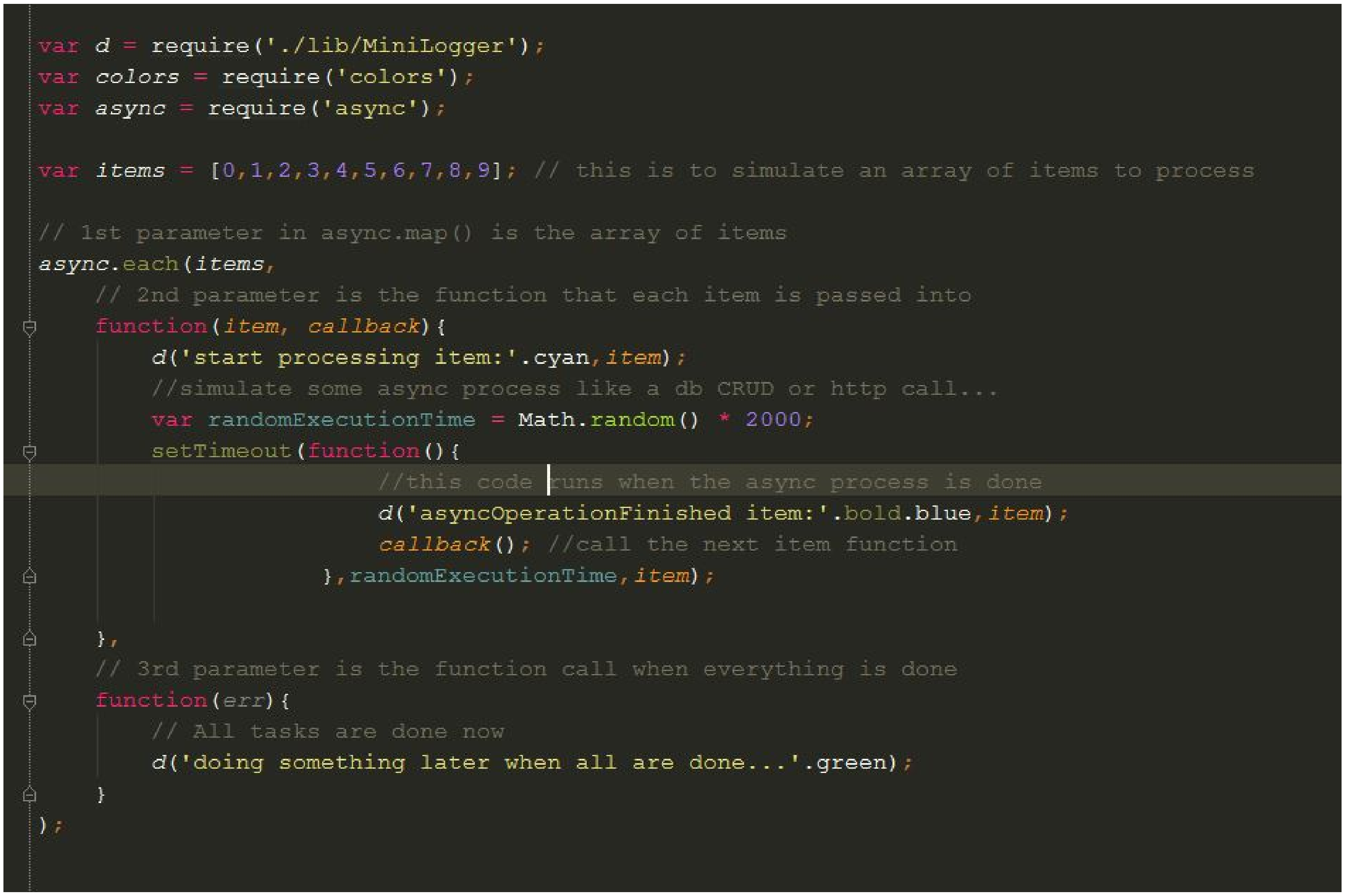
```
i: 0
i: 1
i: 2
i: 3
i: 4
i: 5
i: 6
i: 7
i: 8
i: 9
doing something later
asyncOperationFinished i: 1
asyncOperationFinished i: 9
asyncOperationFinished i: 8
asyncOperationFinished i: 6
asyncOperationFinished i: 5
asyncOperationFinished i: 7
asyncOperationFinished i: 2
asyncOperationFinished i: 3
asyncOperationFinished i: 0
asyncOperationFinished i: 4
```

So instead of keeping a counter or use custom boilerplate code for each of these generic situations – use async.js!

3. Cd to your project directory and install the async package adding it to your json file:

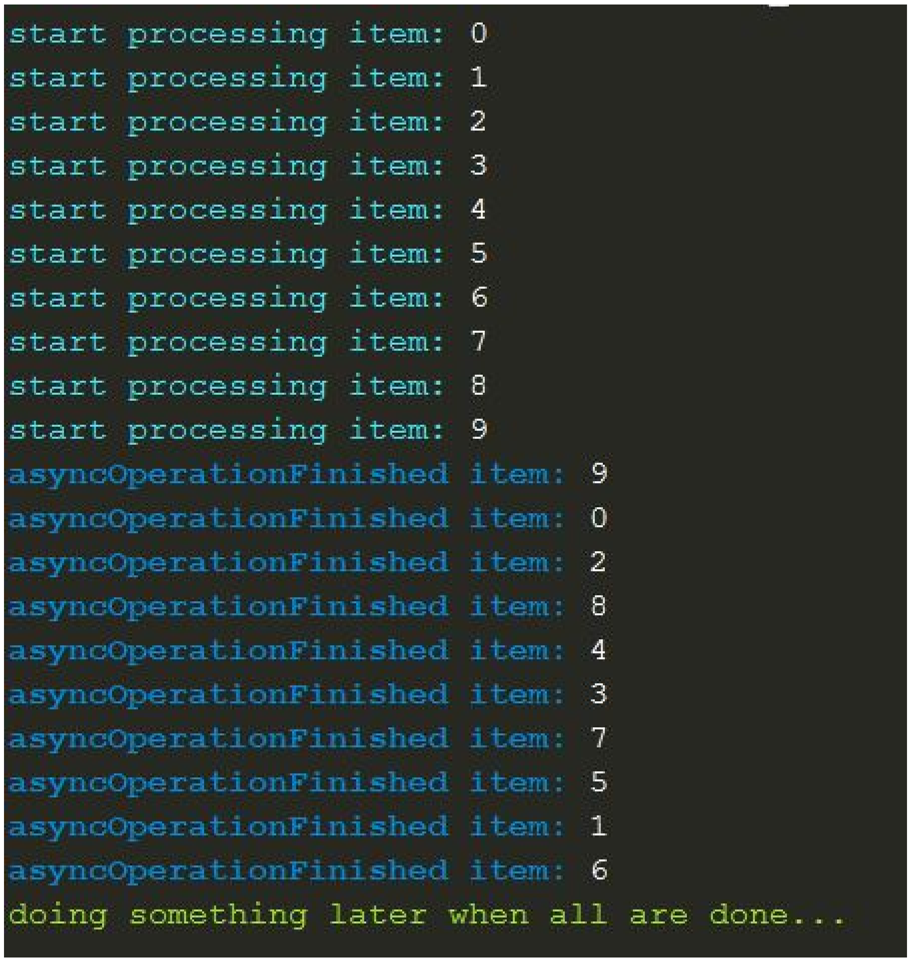
   npm install async --save

4. Modify your code to use **async.each()** like so:



```
var d = require('./lib/MiniLogger');
var colors = require('colors');
var async = require('async');

var items = [0,1,2,3,4,5,6,7,8,9]; // this is to simulate an array of items to process

// 1st parameter in async.map() is the array of items
async.each(items,
    // 2nd parameter is the function that each item is passed into
    function(item, callback){
        d('start processing item:'.cyan,item);
        //simulate some async process like a db CRUD or http call...
        var randomExecutionTime = Math.random() * 2000;
        setTimeout(function(){
                        //this code runs when the async process is done
                        d('asyncOperationFinished item:'.bold.blue,item);
                        callback(); //call the next item function
                    },randomExecutionTime,item);

    },
    // 3rd parameter is the function call when everything is done
    function(err){
        // All tasks are done now
        d('doing something later when all are done...'.green);
    }
);
```

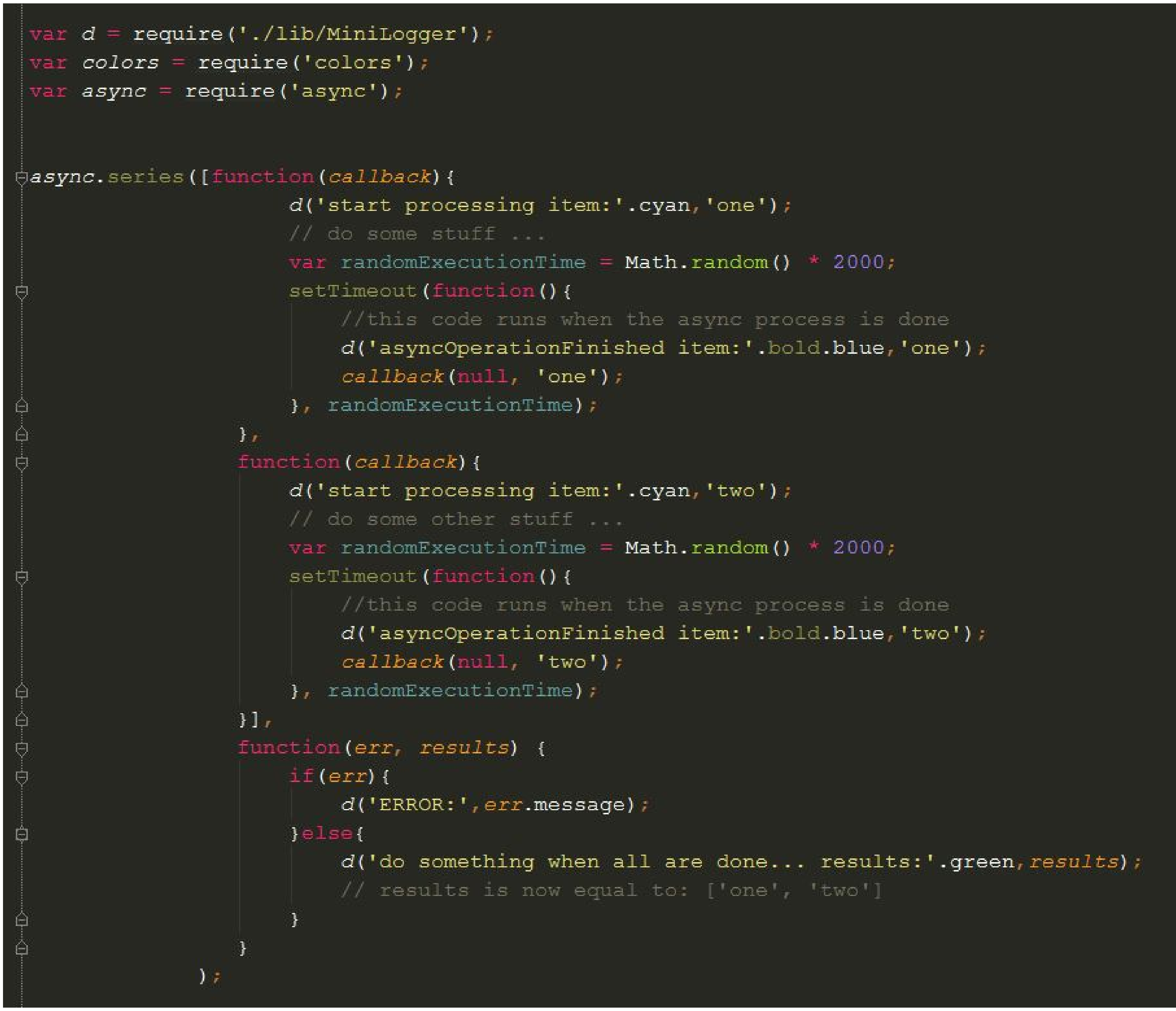
Run the code, you'll get something like this:



```
start processing item:  0
start processing item:  1
start processing item:  2
start processing item:  3
start processing item:  4
start processing item:  5
start processing item:  6
start processing item:  7
start processing item:  8
start processing item:  9
asyncOperationFinished item:  9
asyncOperationFinished item:  0
asyncOperationFinished item:  2
asyncOperationFinished item:  8
asyncOperationFinished item:  4
asyncOperationFinished item:  3
asyncOperationFinished item:  7
asyncOperationFinished item:  5
asyncOperationFinished item:  1
asyncOperationFinished item:  6
doing something later when all are done...
```

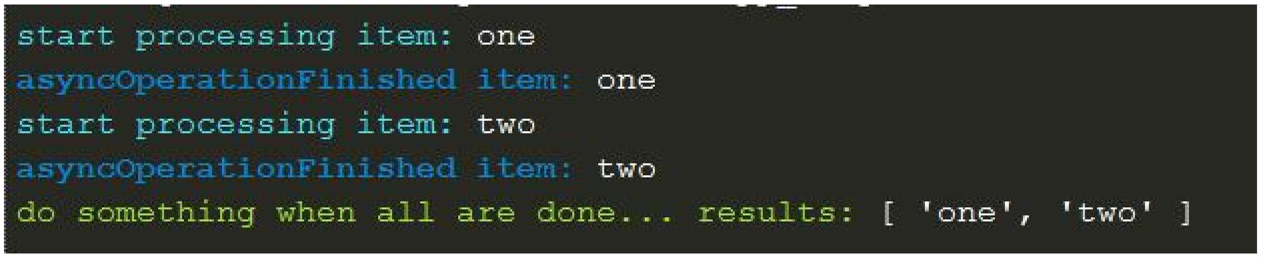
As you can notice – the different operations are done independently regardless of the others, and when all of them are done, we'll execute a final callback.

5. Next, if the order in which the functions are being executed is important and we want to execute one only after the other had finished, we'll use **async.series()**



```
var d = require('./lib/MiniLogger');
var colors = require('colors');
var async = require('async');


async.series([function(callback){
                    d('start processing item:'.cyan,'one');
                    // do some stuff ...
                    var randomExecutionTime = Math.random() * 2000;
                    setTimeout(function(){
                        //this code runs when the async process is done
                        d('asyncOperationFinished item:'.bold.blue,'one');
                        callback(null, 'one');
                    }, randomExecutionTime);
                },
                function(callback){
                    d('start processing item:'.cyan,'two');
                    // do some other stuff ...
                    var randomExecutionTime = Math.random() * 2000;
                    setTimeout(function(){
                        //this code runs when the async process is done
                        d('asyncOperationFinished item:'.bold.blue,'two');
                        callback(null, 'two');
                    }, randomExecutionTime);
                }],
                function(err, results) {
                    if(err){
                        d('ERROR:',err.message);
                    }else{
                        d('do something when all are done... results:'.green,results);
                        // results is now equal to: ['one', 'two']
                    }
                }
             );
```

6. Run the code and get the following:



```
start processing item: one
asyncOperationFinished item: one
start processing item: two
asyncOperationFinished item: two
do something when all are done... results: [ 'one', 'two' ]
```

As you can notice, each process waits for the previous one to finish before continuing.

7. Using the **async** documentation, produce two more examples using the **que** and **waterfall** flows.

Notes:

- Visit async.js on github for a full documentation https://github.com/caolan/async