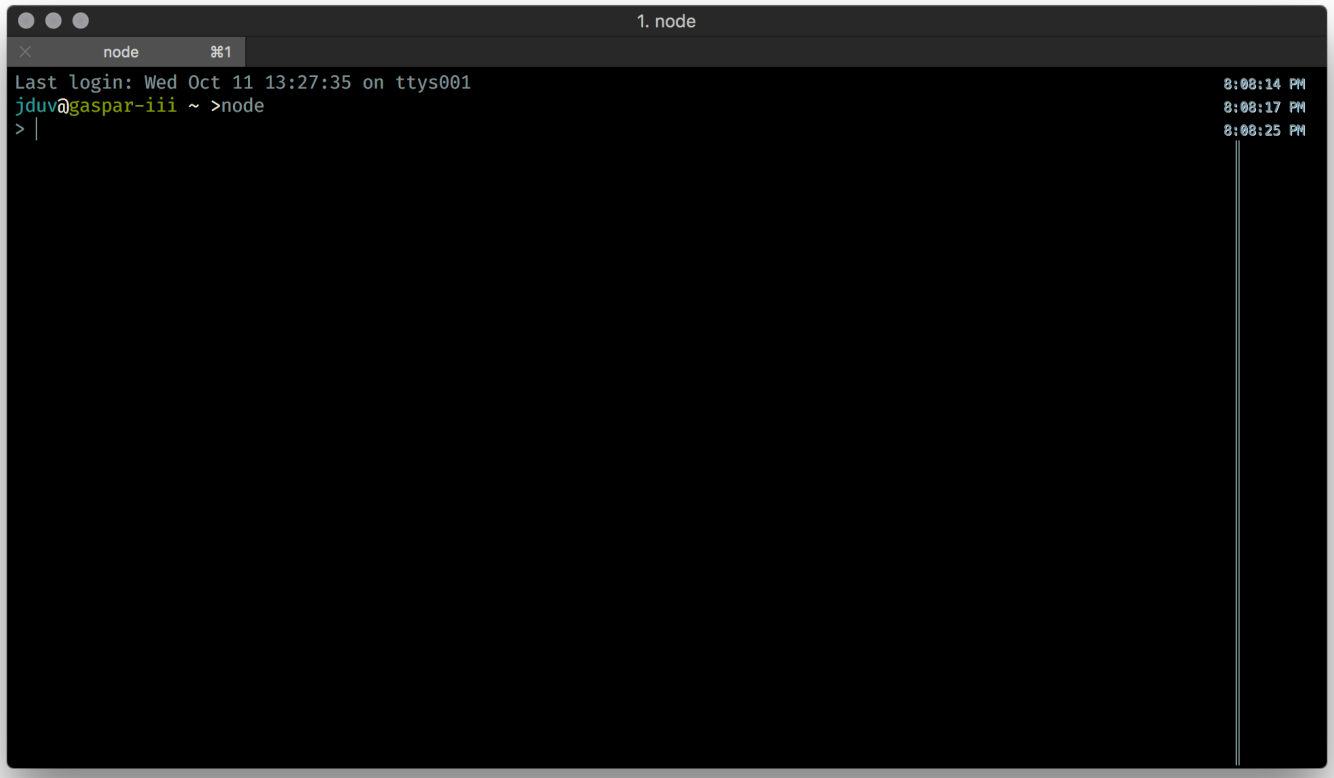
Node.js

The basics

Node is a run time environment similar to python. Instead of running scripts that are `.py` we run scripts that end in `.js`. Which means, of course, that any code we write that runs on the node framework is written in JavaScript.

How it works

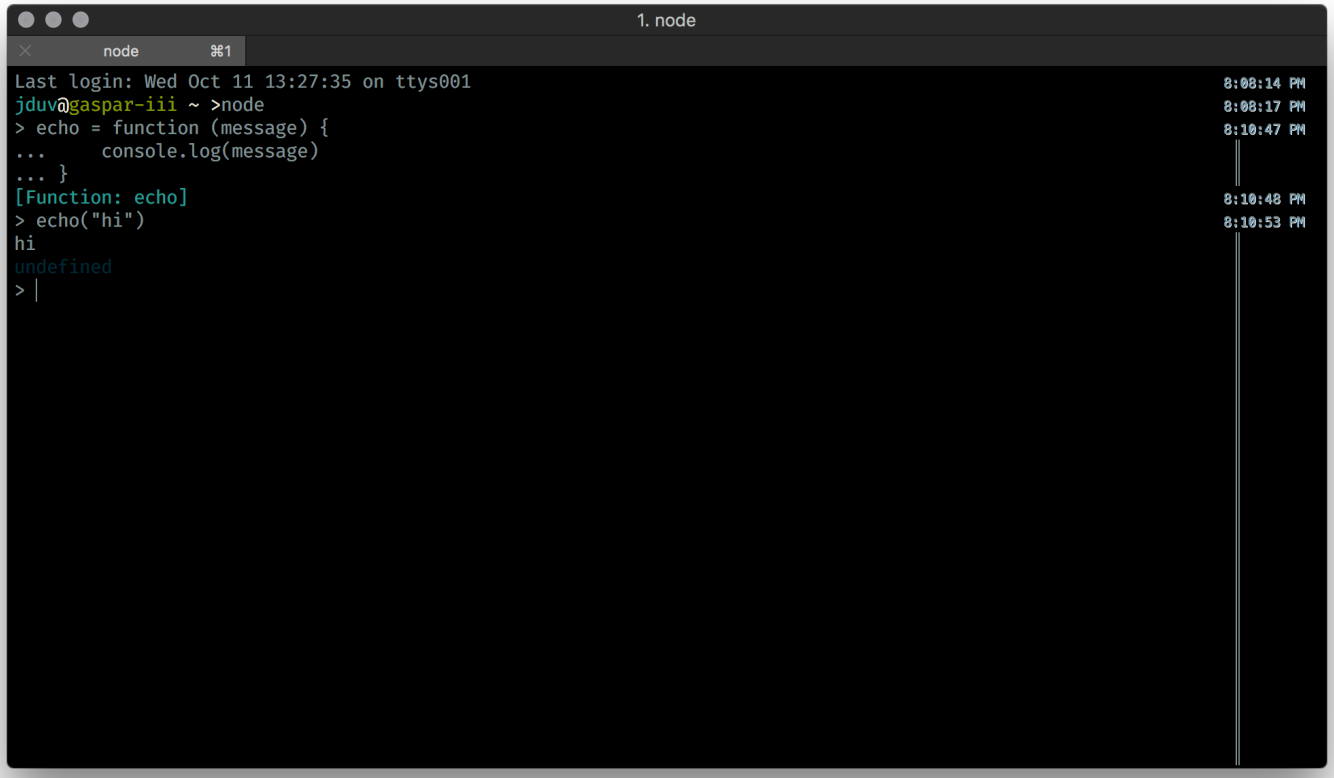
First, let's fire up the node interpreter and play around with it. This is important because it will help you understand, just like we did in python, how code runs: top to bottom left to right.



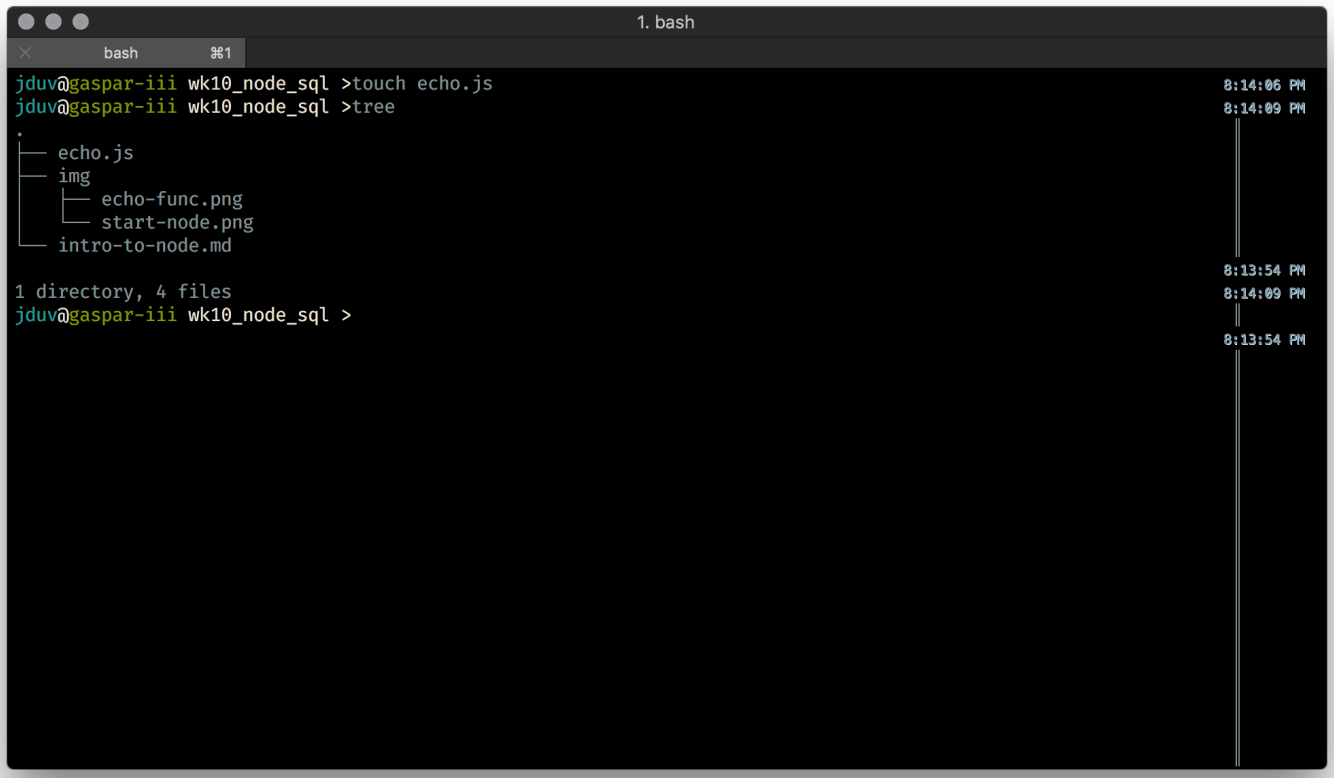
Let's write a simple program that writes a message to the console. The program looks like this:

```
echo = function (message) {
    console.log(message)
}
```

Paste or write this functin line for line in your node.js interpreter. You'll get something like this:



This function will remain in the interpreters memory as long as you have the current shell open. So, if you wanted to keep this function safe we might as well save it in the current directory as a `.js` file. Like so:



Paste the function along with a line executing the function into `echo.js` like so:

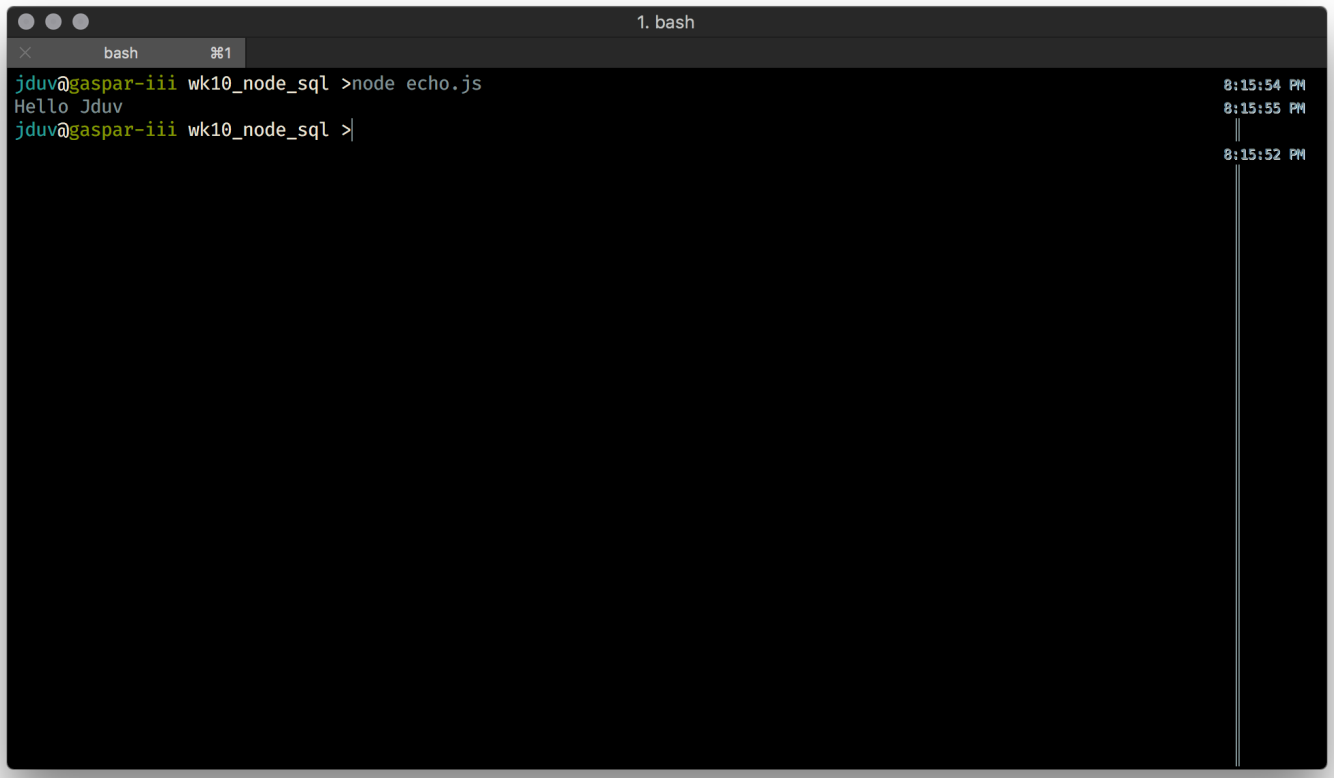
```
echo = function (message) {
    console.log(message)
}

echo("Hello Jduv")
```

Now, run the file like so:

```
> node echo.js
```

You should see something like the terminal below:



Excercise 1:: Using the node interpreter, write a function that doubles a number. Write another function that reverses a string.

Exercise 2:: Save your work to files called `double.js` and `reverse.js`. Run them using the node interpreter from the command line.

As a framework

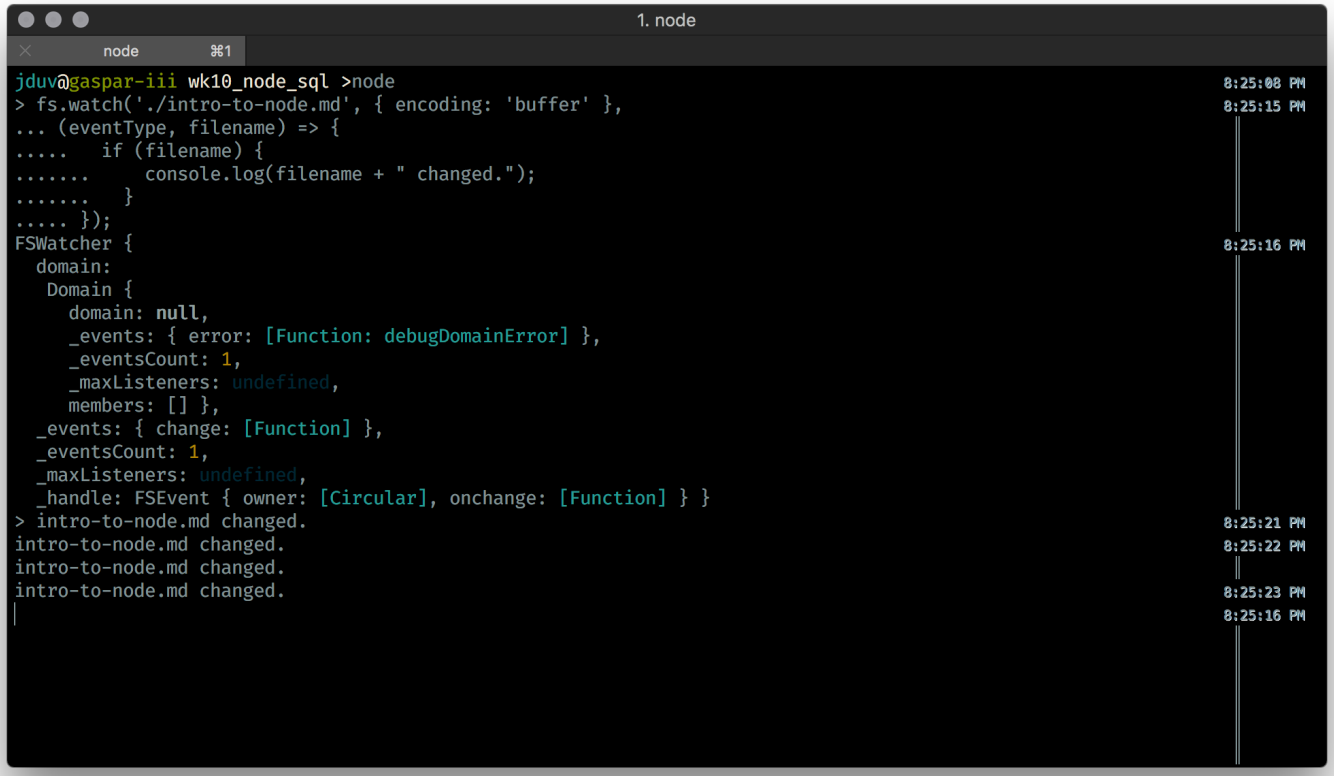
Just like python, node ships with lots of built in libraries that do interesting things. Take, for example, the `fs` package. What does it do? Well, a lot. At least based on the documentation linked above. But, let's explore an interesting piece of it called a `FSWatcher` object.

Code along solution

```
// Example when handled through fs.watch listener
fs.watch('./intro-to-node.md', { encoding: 'buffer' },
(eventType, filename) => {
  if (filename) {
    console.log(filename + " changed.");
```

```
    }
  });
```

The `fs.watch()` function will, using system calls and voodoo magic, watch the target file for any modifications and execute the given callback function on a change. Note that the implementation of this module is operating system specific because the semantics of file systems are native to their respective operating systems. Therefore, you may get different results on Windows, Linux, and Mac OSX systems.



Note a few interesting things about this code and it's execution: the code runs immediately, then returns and gives you back control. This is because the `fs.watch()` function runs *asynchronously* to your node terminal. That is, you can do other stuff while this code runs in the background. In order to stop a watcher, according to the documentation you need to call `watcher.close()`. So, let's modify our code a bit so we can do so:

```
// Example when handled through fs.watch listener
watcher = fs.watch('./intro-to-node.md', { encoding: 'buffer' },
(eventType, filename) => {
```

```
    if (filename) {
      console.log(filename + " changed.");
    }
  });
```

Now that we have stored our watcher in a variable, we can access it! So, let's close the watcher like so: