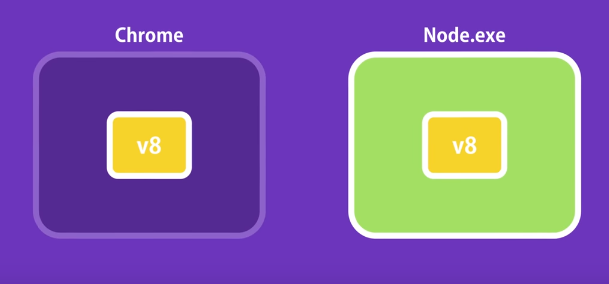
**NODE**

* Node is a runtime environment for executing JavaScript code.
* Before Node JavaScript was used only to build apps that run on a browser.
* Browsers have built-in JavaScript Engines that execute JavaScript
* Browser provides runtime environment to JavaScript code.
* As different browsers have different JavaScript Engine, JS might behave differently on each.
* Node doesn’t have the Window, Document etc. objects



* Node provides environment objects such as

fs – File System

http

* Node apps are highly scalable, data intensive, and real-time apps due to the **Non-Blocking and asynchronous** nature of Node.

Blocking and Synchronous Architecture of apps results in inefficient utilization of resources.

* Node should not be used for CPU intensive applications as that might keep the client requests waiting

It should be used for Data intensive applications.

**Node Installation:**

Jslint

**Working Principle of Node**

* Single Thread
* Multiple Requests
* Event loop
* Blocking request/code
* Non-blocking
* Call backs
* Future functions
* Memory queue - Event Queue
* Libuv compiler - written C++ - converts java script into machine code
* Observer – for every request
* Request Reference
* Sync Process
* Asynchronous - By default
* Resolving blocking Code

**Node Core**

**Node Module System:**

Global Object:

console object - Part of the global scope

setTimeout()

clearTimeout()

setInterval()

clearInterval()

All the above belong to window object in JS.

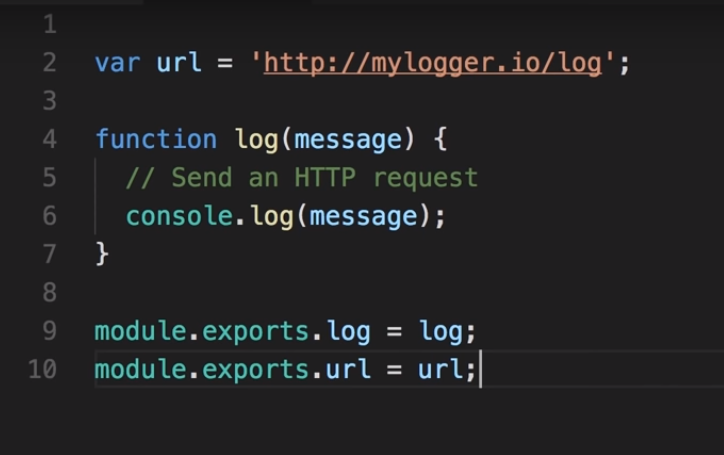
In JS the variables and functions declared are added to global scope.

If two functions are given same name the latest one will override the previous one.

It is not recommended to add the variables and functions into global scope.

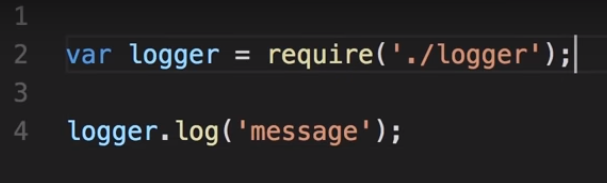
Enter the Modularity:

* With modules the unintentional function overriding can be avoided
* Every Node application will have at least one file or one module which is called the main module
* All the variables and functions defined in a module are available only within the module unless exported explicitly.
* They can be exported by adding to the exports object (as below).



Loading a module:

Using ***require***



var logger = require('./logger');

- returns an object containing exported members and assign to the variable.

const logger = require('./logger');

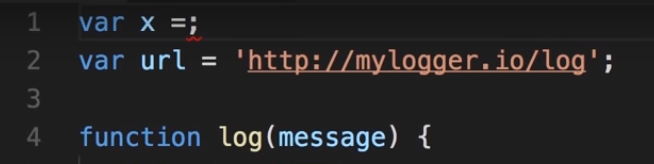
- Best practice to use constant in order to avoid unintentional overriding of the variable

To export a single function, following can be done

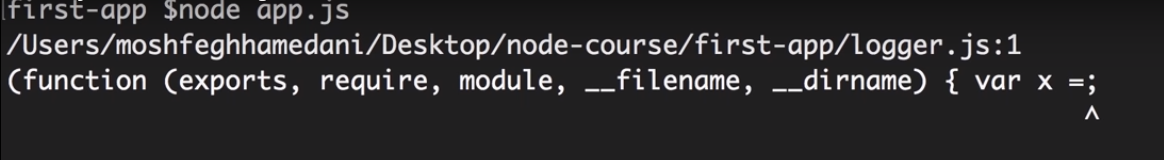
module.exports=log;

**Module Wrapper Function**

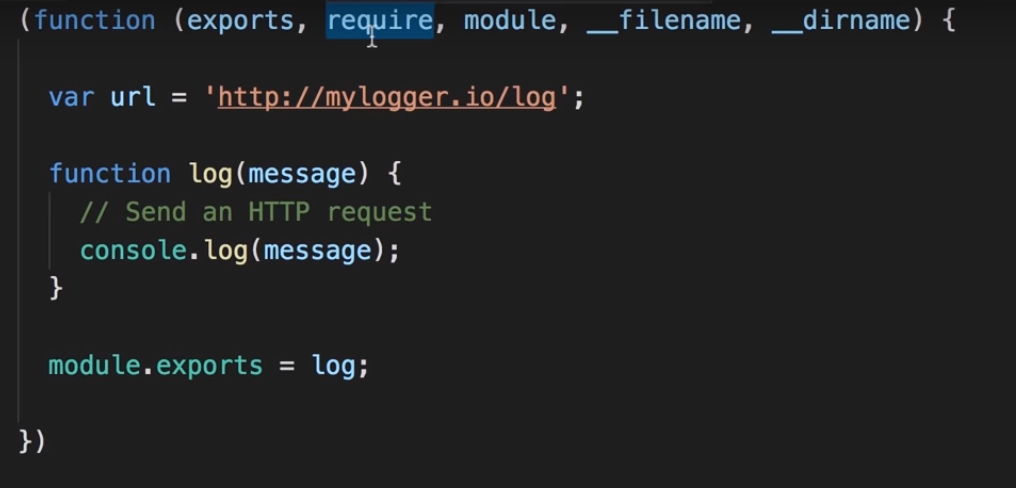
Make a syntactical error



Run the app



Node wraps the Module inside a Wrapper function or IIFE



Built-in Modules

fs

utils

http

url

querystring

path

os

path:

os:

methods

freemem()

totalmem()

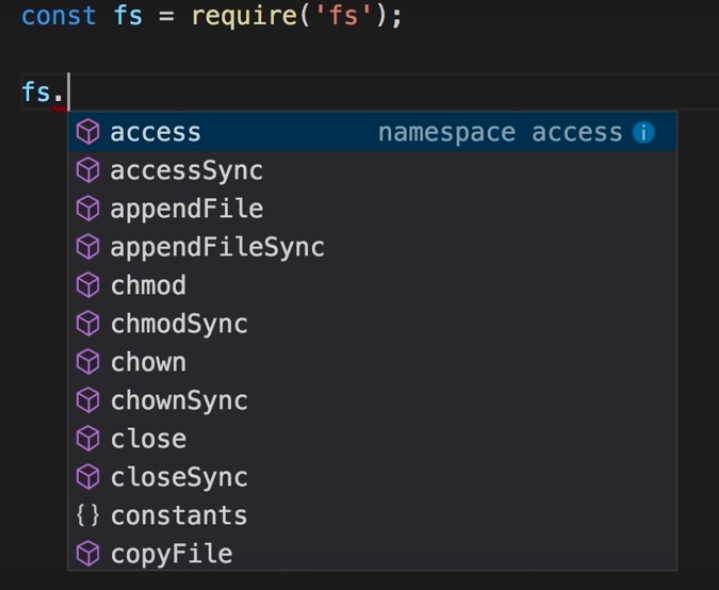
const os = require('os')

console.log(`Free memory: ${os.freemem()}`);

console.log(`Total memory: ${os.totalmem()}`);

fs:

Has both synchronous and asynchronous methods



* Recommended to use asynchronous methods to keep the application safe from blocking operations
* All the asynchronous operations take a function as their last argument which a callback method, will be called/executed, node will call this method when the asynchronous operation completes.

**Modules:**

* Maintainable
* Reusable
* Segregation

2 ways of creating modules

1. Creating the module directly as file.js

Exports

1. creating package and adding the modules and index.js which points to the modules

**Predefined modules**

Fs, utils, http,url,querystring

**Retrieving data from URL**

url

queryString

Parse()

**Run on**

**Promises/Future Functions**

**Writing simple functions**

**Childthread**

**Spawn**

**Exec**

**Creating server**

\_ - (underscore) stores recently created values

writeHead -

Nodeman

response.end()

**To run a node application**

Command – node <app.js>

i case

g – global

parseURL()

OR will return first true value

AND will return last true value

To include HTML – createReadStream

Pipe(response) method

Chunk

Request.on(‘data’, (chunk)=>postData+= chunk);

**Http Methods**

GET

POST

PUT

DELETE

PATCH

POST:

Inside a function

this.var = value - Creates an instance variable

Request object

Request.headers

Request.url

**Custom Events**

* Create a

**Stream and Buffer**

Buffer - is a mechanism that can read write and modify stream of binary data.

Binary data –

charCodeAt

string.charCodeAt – to get ASCII

Stream – Taking input and giving output bit by bit

Youtube example Buffering

Converting Buffer to following

* To string
* To ascii
* To hex
* To JSON

charCodeAt

**Buffer Methods**

toString

concat - will concat buffers

buffer2 = Buffer.