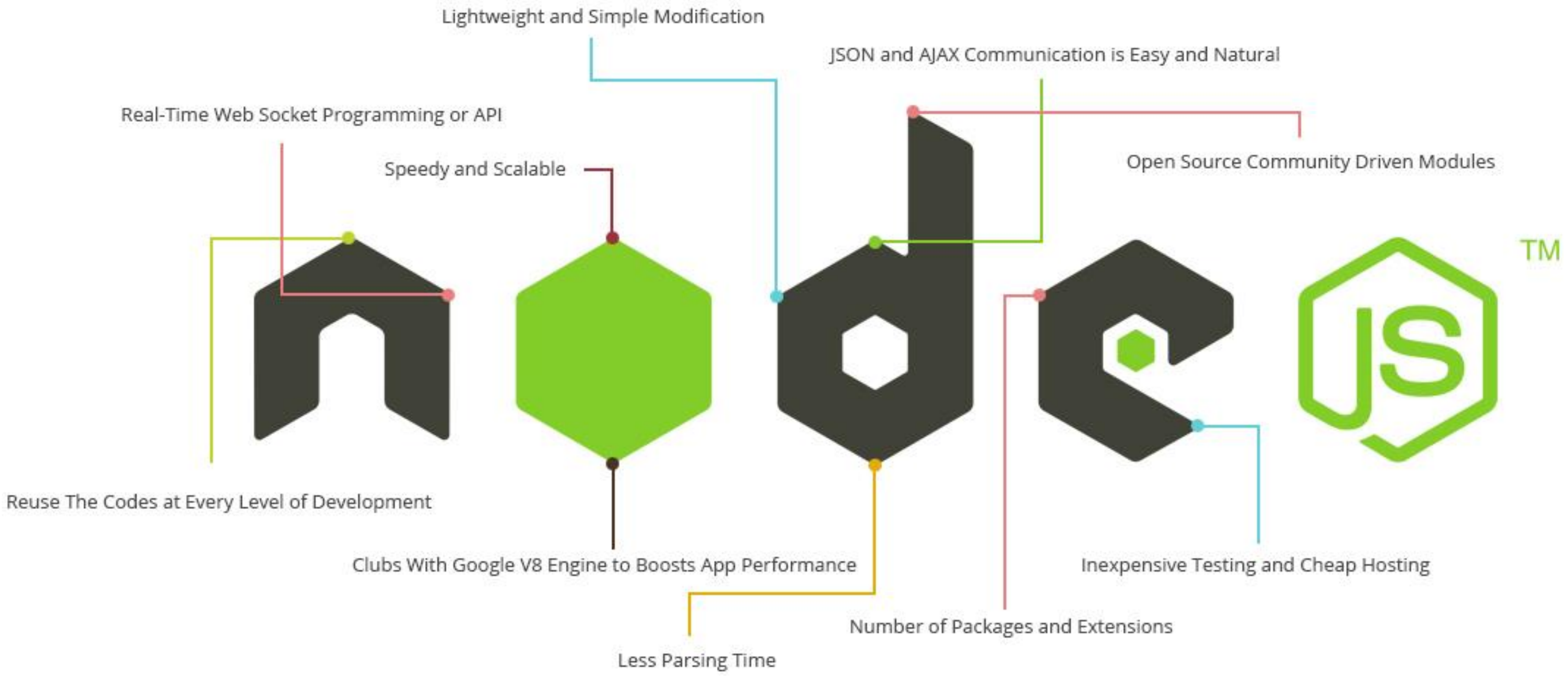
JavaScript Everywhere!

The Server-side JavaScript

Eng. Niveen Nasr El-Den
SD & Gaming CoE.
iTi

Day 1


Lightweight and Simple Modification
JSON and AJAX Communication is Easy and Natural
Real-Time Web Socket Programming or API
Speedy and Scalable
Open Source Community Driven Modules
TM
Reuse The Codes at Every Level of Development
Clubs With Google V8 Engine to Boosts App Performance
Inexpensive Testing and Cheap Hosting
Number of Packages and Extensions
Less Parsing Time

http://www.unifiedinfotech.net/node-js-development/

INTRODUCTION

- **Node** is the official name of the project, **NodeJS** to avoid confusion
- Created by **Ryan Dahl** in 2009
- Sponsored by **Joyent** (now a part of Samsung)
- Stable release is **10.15.1**
- Its an Open Source, Cross-platform runtime environment for server-side and a high-performance networking applications framework

https://nodejs.org/en/
https://github.com/nodejs/node

INTRODUCTION

- You can write a JavaScript file that will run in either Node or the browser.
- Node.js is designed for DIRT applications
  - DIRT stands for : *data-intensive real-time* applications
  - e.g. video streaming, SPAs, networking apps
- Node.js is a *platform* for JavaScript applications running out side browser
- It's a **command line** tool.
- Its built in v8 chrome engine.

Companies Backing Node.js

Microsoft IBM

YAHOO!

UBER

ebay™ Joyent

GoDaddy™

airbnb

Google intel®

Linked in® PayPal JS

What can you do with Node.js ?

- You can create an **HTTP server**
- You can create a **TCP server** similar to HTTP server
- You can create a **Web Chat Application**
- Node.js can also be used for creating online games, collaboration tools or anything which sends updates to the user in **real-time.**

What can't do with Node?

- Node is a platform for writing JavaScript applications outside web browsers.
  - This is not the JavaScript we are familiar with in web browsers.

- There is no DOM built into Node, nor any other browser capability.

- Node can’t run on GUI, but run on terminal/cmd

Node Js

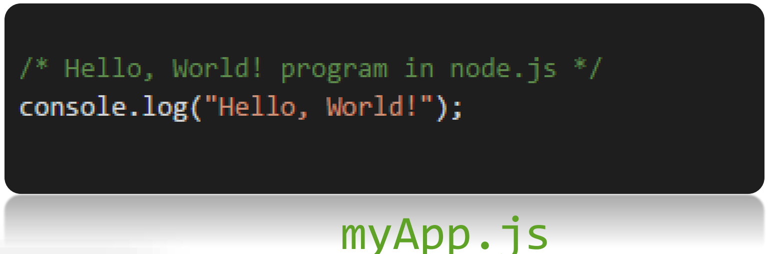
JavaScript running outside a browser

GETTING STARTED & HELLO WORLD

- Install Node.js.
  - `.msi`
  - `npm`
- Use any text editor
- Type your JavaScript code (myApp.js).
- Open cmd and type this:

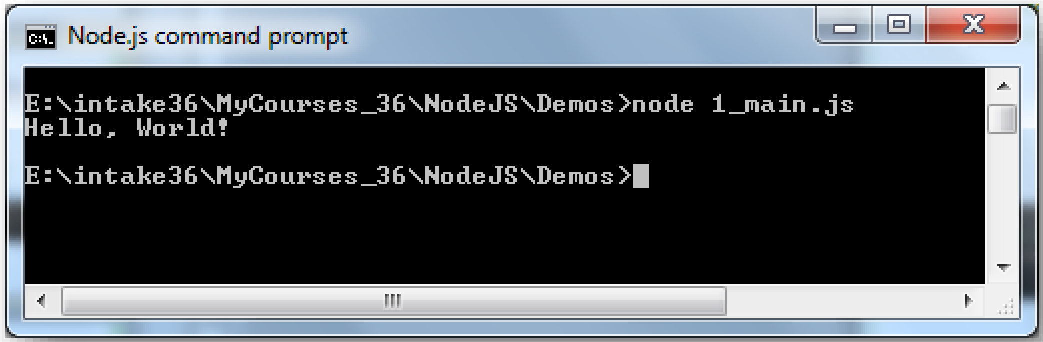
```
node myApp.js
```

`Make sure you are on the same path where myApp.js is saved`



```
/* Hello, World! program in node.js */
console.log("Hello, World!");
```

`myApp.js`



```
E:\intake36\MyCourses_36\NodeJS\Demos>node 1_main.js
Hello, World!

E:\intake36\MyCourses_36\NodeJS\Demos>
```

REPL

- REPL stands for Read Evaluate Print Loop
- REPL is like browser console
- REPL is a quick way to write and execute js without creating a file
- REPL commands
  - Ctrl+c
  - Ctrl+d
  - .editor
  - .break
  - .clear
  - Ctrl+l
  - .load file_nm
  - .save file_nm
  - .exit

Getting Started with Nodejs

Demo1

Node.js facts

- Node.js applications are written in JavaScript, and can be run within the Node.js runtime on different platforms

- It provides a rich library of various JavaScript modules.

- Node.js makes communication between client and server will happen in same language

Node.js facts

- Node.js makes use of **event-loops** via JavaScript's **callback** functionality to implement the non-blocking I/O.
- There is no DOM implementation provided by Node.js
- Everything inside Node.js runs in a **single-thread**.

Node.js is "**Event**" based server

From the official website

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

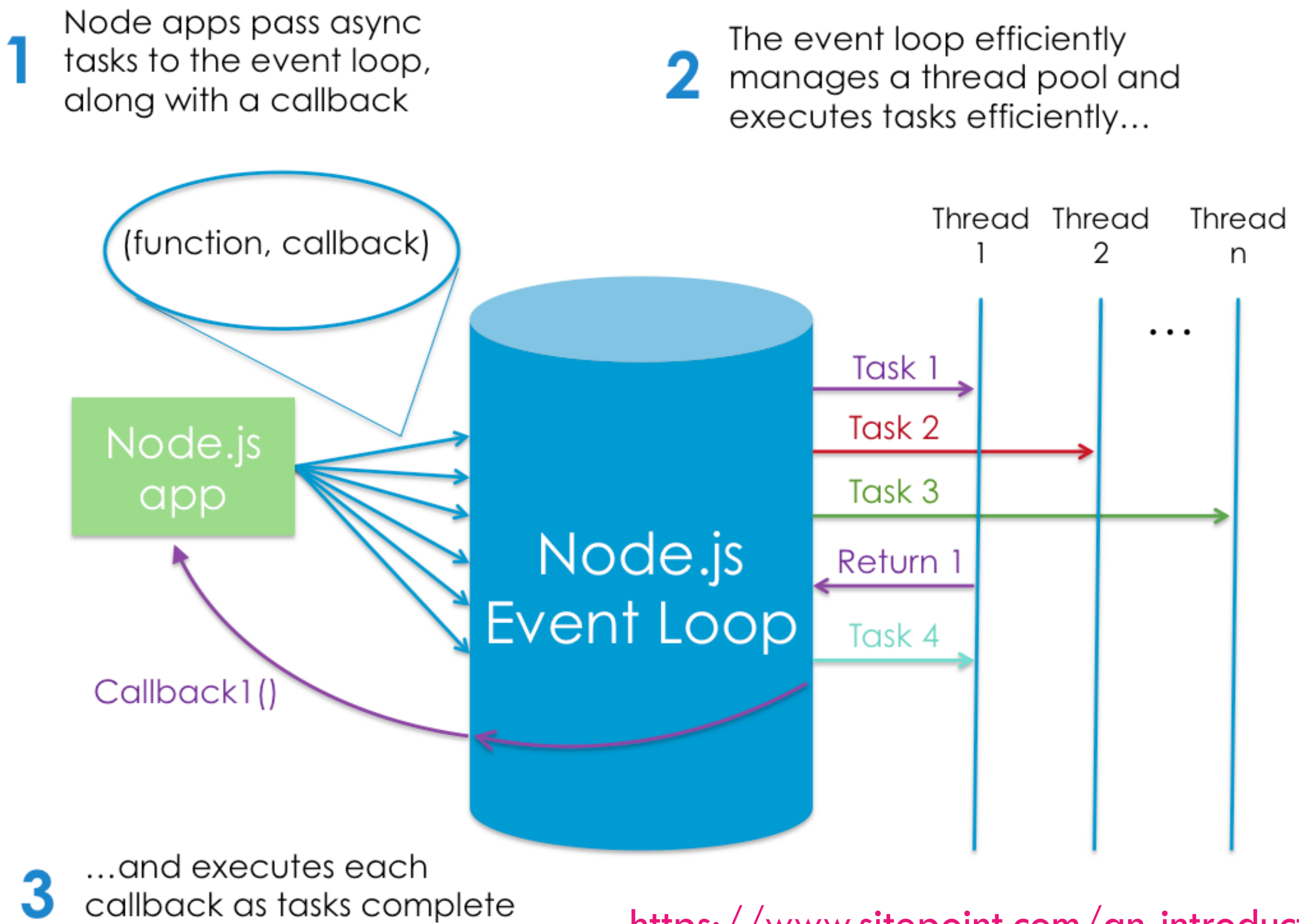
Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

'Node's goal is to provide an easy way to build scalable network programs'

Node.JS Architecture

- Node is single-process
- Node.js handles requests with a **single** thread (the event loop)
- Allows us to easily write asynchronous code without heavy thread management
- Worth mentioning again: Any JavaScript code in a single context is Synchronous

Node.JS Architecture


1
Node apps pass async tasks to the event loop, along with a callback
2
The event loop efficiently manages a thread pool and executes tasks efficiently…
(function, callback)
Node.js app
Node.js Event Loop
Thread 1
Thread 2
Thread n
…
Task 1
Task 2
Task 3
Return 1
Task 4
Callback1()
3
…and executes each callback as tasks complete

https://www.sitepoint.com/an-introduction-to-node-js/

Event Loop

- Event-loops are the core of event-driven programming, almost all the UI programs use event-loops to track the user event, e.g. Clicks, Ajax Requests etc.

- Instead of threads Node.js uses an event loop with a stack (EventQueue)

Non-blocking I/O

- Servers do nothing but I/O
  - Scripts waiting on I/O requests degrades performance
- To avoid blocking, Node makes use of the event driven nature of JS by attaching callbacks to I/O requests
- Scripts waiting on I/O waste no space because they get popped off the stack when their non-I/O related code finishes executing

Non-Blocking is

Doing other things without stopping your program from running, this is possible by doing things asynchronous.

callbacks

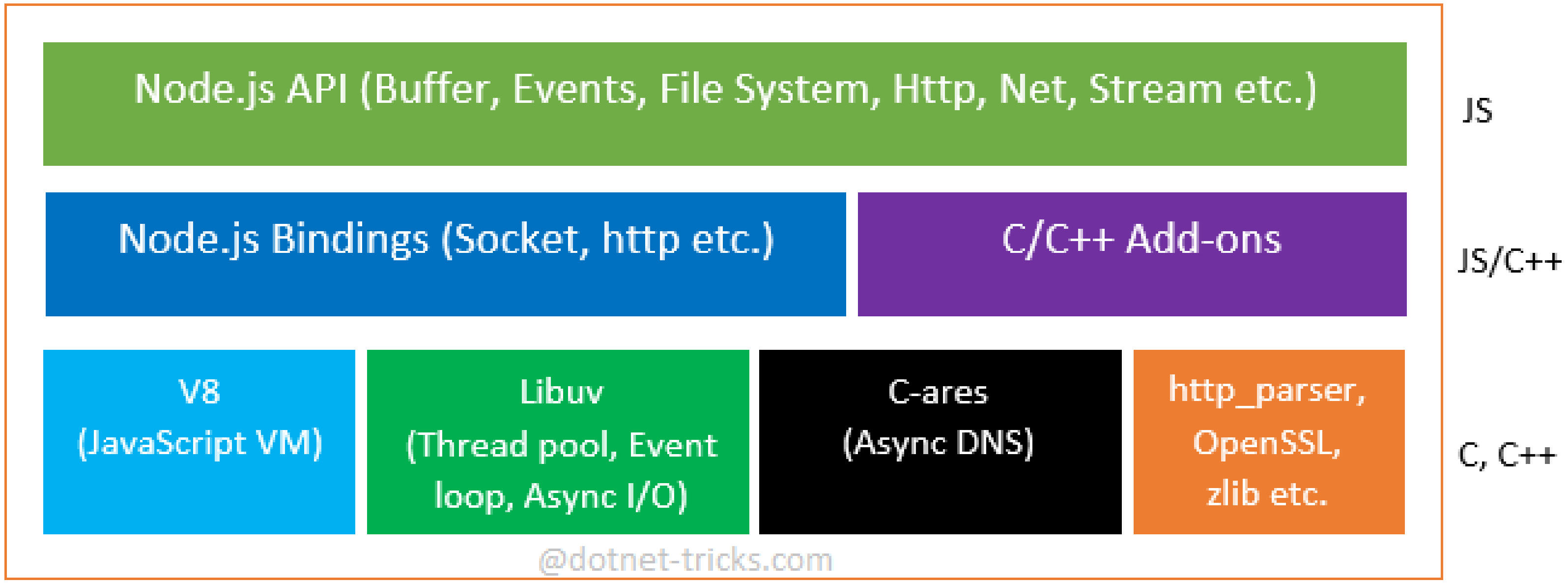
Promises

EventEmitters

Node.js Architecture

- Node.js includes
  - **V8**
    - open source JavaScript engine that resides in Chrome
  - **Unicorn Velociraptor Library (libuv),** "UV"
    - houses the Node.js event loop and the internal mechanisms used to process registered callback functions.
  - **Set of supporting libraries**
    - allow Node.js to perform common operations, such as opening a socket, interfacing with the file system, or starting an HTTP server. While
- libuv and the Node.js supporting libraries are written in C++

Node.js Architecture


Node.js API (Buffer, Events, File System, Http, Net, Stream etc.)
JS
Node.js Bindings (Socket, http etc.)
C/C++ Add-ons
JS/C++
V8 (JavaScript VM)
Libuv (Thread pool, Event loop, Async I/O)
C-ares (Async DNS)
http_parser, OpenSSL, zlib etc.
C, C++
@dotnet-tricks.com

Node.js 4.2.4 Architecture

http://www.dotnettricks.com/learn/nodejs/exploring-nodejs-architecture

Node Globals

- **global** – is the global object similar to window in DOM
- **process** – object providing information and methods for the current process
  - Its a global object and can be accessed from anywhere.
  - It is an instance of EventEmitter.
- **console** – allows printing to stdout
- **require()** – function to load a module
- **module** – refers to the current module
- **exports** – refers to an object that will be exposed as a module
- **__filename and __dirname**

NODE.JS ECOSYSTEM

- Node.js heavily relies on **modules** in order to load built-in APIs, third party modules or custom local module
- Module is a self contained series of one or more .js files presented by an object
- Modules is where we can encapsulate related functionality into a single file.
- Should be accessible from outside the file

Modules

- Modules allow Node to be extended.
- Modules act as libaries
  - It's a set reusable code that adds extra functionality to our application
- We can include a module with the **global require** function, **require('module');**
- Every module has its own scope

Modules

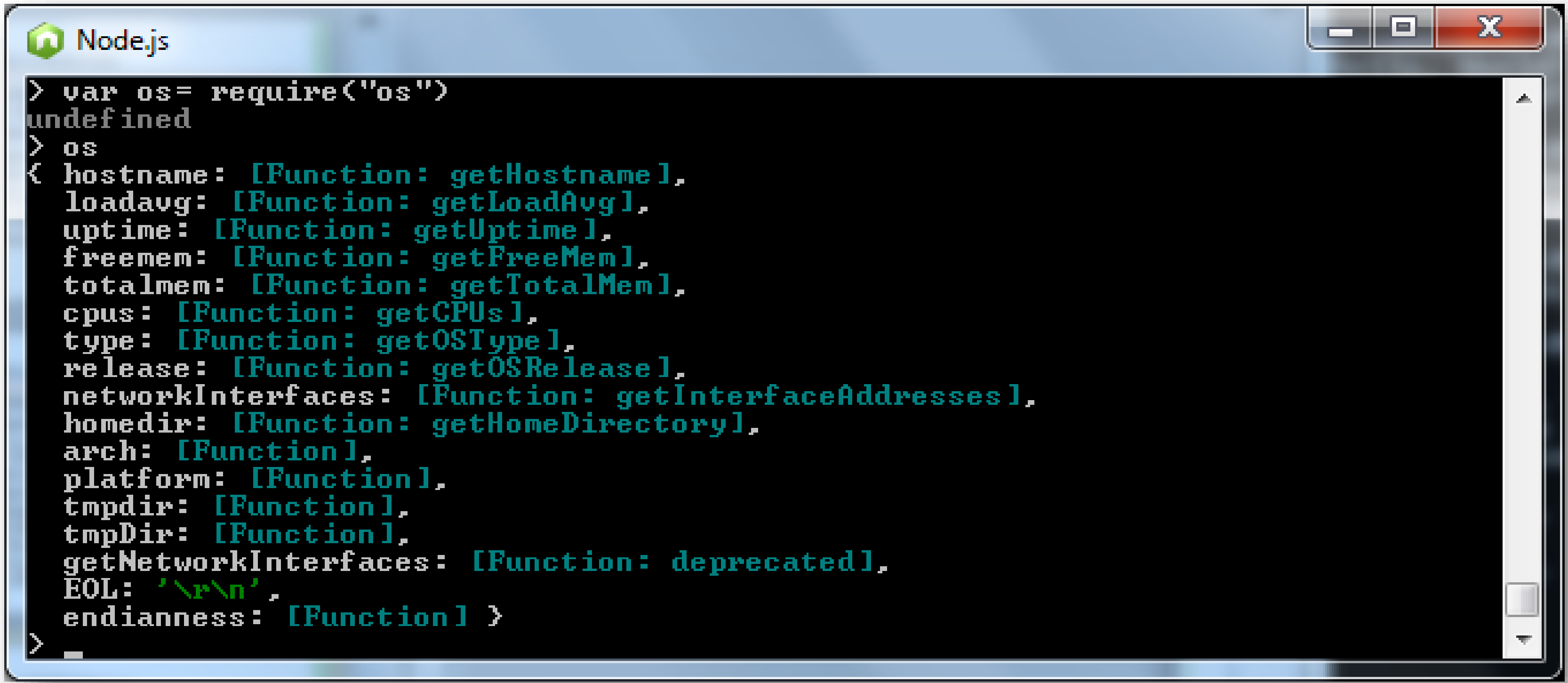
- We can install helping module via npm

- Node provides core modules that can be included by their name:
  - File System – **require('fs')**
  - Http – **require('http')**
  - Utilities – **require('util')**
  - **Etc..**

- We can create our own custom module

OS Module

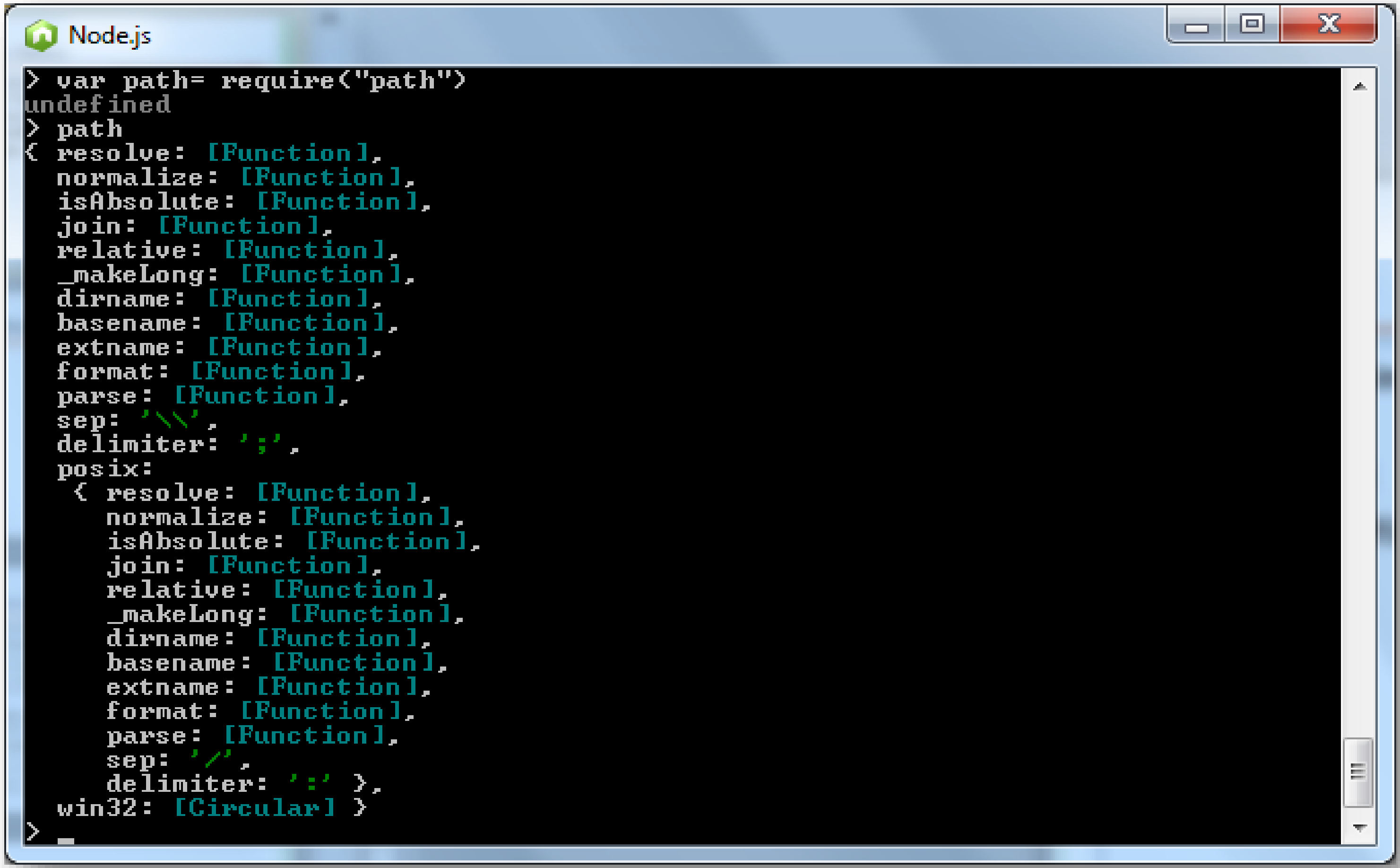
- Module for info about operating system of application
- Useful for statistics



```
> var os= require("os")
undefined
> os
{ hostname: [Function: getHostname],
  loadavg: [Function: getLoadAvg],
  uptime: [Function: getUptime],
  freemem: [Function: getFreeMem],
  totalmem: [Function: getTotalMem],
  cpus: [Function: getCPUs],
  type: [Function: getOSType],
  release: [Function: getOSRelease],
  networkInterfaces: [Function: getInterfaceAddresses],
  homedir: [Function: getHomeDirectory],
  arch: [Function],
  platform: [Function],
  tmpdir: [Function],
  tmpDir: [Function],
  getNetworkInterfaces: [Function: deprecated],
  EOL: '\r\n',
  endianness: [Function] }
> _
```

Path Module

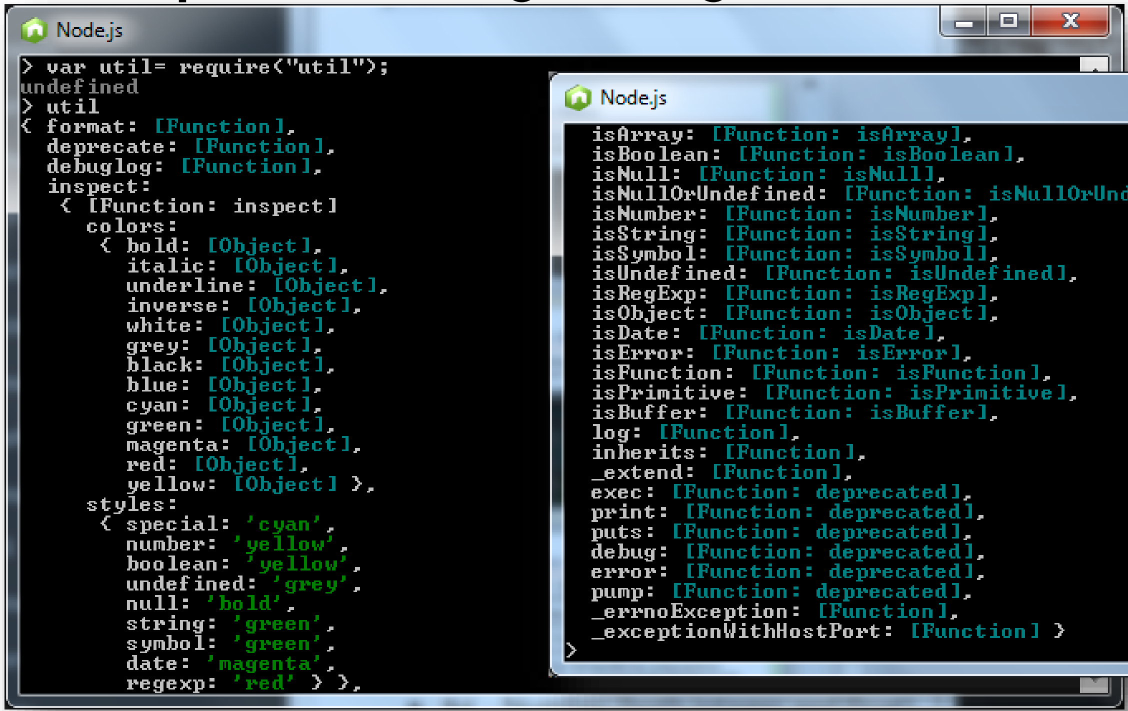
- module contains utilities for handling and transforming file paths



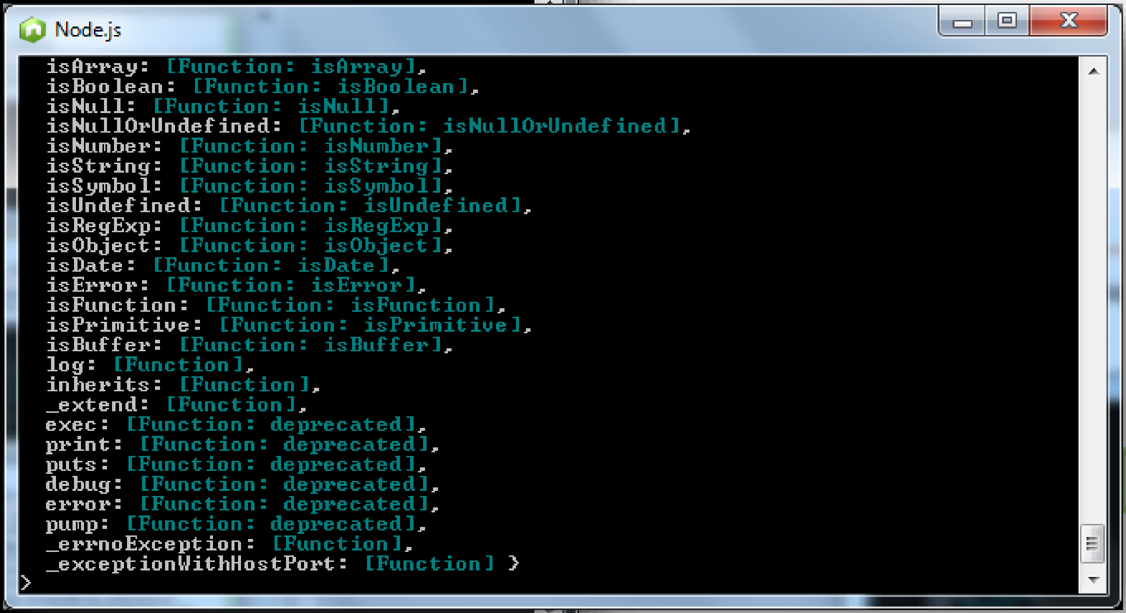
```
> var path= require("path")
undefined
> path
{ resolve: [Function],
  normalize: [Function],
  isAbsolute: [Function],
  join: [Function],
  relative: [Function],
  _makeLong: [Function],
  dirname: [Function],
  basename: [Function],
  extname: [Function],
  format: [Function],
  parse: [Function],
  sep: '\\',
  delimiter: ';',
  posix:
   { resolve: [Function],
     normalize: [Function],
     isAbsolute: [Function],
     join: [Function],
     relative: [Function],
     _makeLong: [Function],
     dirname: [Function],
     basename: [Function],
     extname: [Function],
     format: [Function],
     parse: [Function],
     sep: '/',
     delimiter: ':' },
  win32: [Circular] }
> _
```

UTIL MODULE

- Designed to support the needs of Node.js's internal APIs.
- prototypical inheritance can be implementing using **util.inherits**() method



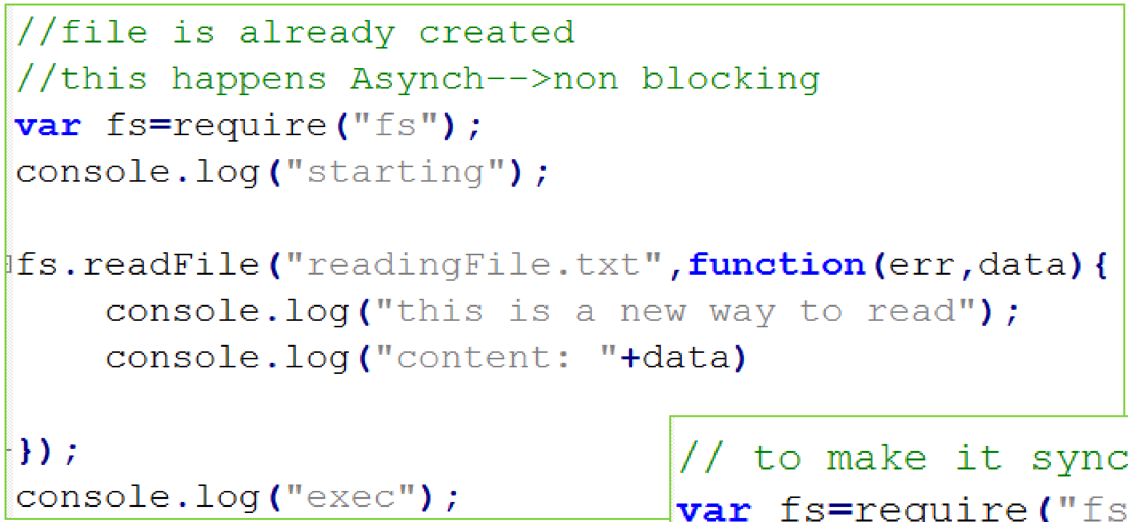
```
> var util= require("util");
undefined
> util
{ format: [Function],
  deprecate: [Function],
  debuglog: [Function],
  inspect:
   { [Function: inspect]
     colors:
      { bold: [Object],
        italic: [Object],
        underline: [Object],
        inverse: [Object],
        white: [Object],
        grey: [Object],
        black: [Object],
        blue: [Object],
        cyan: [Object],
        green: [Object],
        magenta: [Object],
        red: [Object],
        yellow: [Object] },
     styles:
      { special: 'cyan',
        number: 'yellow',
        boolean: 'yellow',
        undefined: 'grey',
        null: 'bold',
        string: 'green',
        symbol: 'green',
        date: 'magenta',
        regexp: 'red' } },
```



```
  isArray: [Function: isArray],
  isBoolean: [Function: isBoolean],
  isNull: [Function: isNull],
  isNullOrUndefined: [Function: isNullOrUndefined],
  isNumber: [Function: isNumber],
  isString: [Function: isString],
  isSymbol: [Function: isSymbol],
  isUndefined: [Function: isUndefined],
  isRegExp: [Function: isRegExp],
  isObject: [Function: isObject],
  isDate: [Function: isDate],
  isError: [Function: isError],
  isFunction: [Function: isFunction],
  isPrimitive: [Function: isPrimitive],
  isBuffer: [Function: isBuffer],
  log: [Function],
  inherits: [Function],
  _extend: [Function],
  exec: [Function: deprecated],
  print: [Function: deprecated],
  puts: [Function: deprecated],
  debug: [Function: deprecated],
  error: [Function: deprecated],
  pump: [Function: deprecated],
  _errnoException: [Function],
  _exceptionWithHostPort: [Function] }
>
```

FILE SYSTEM MODULE

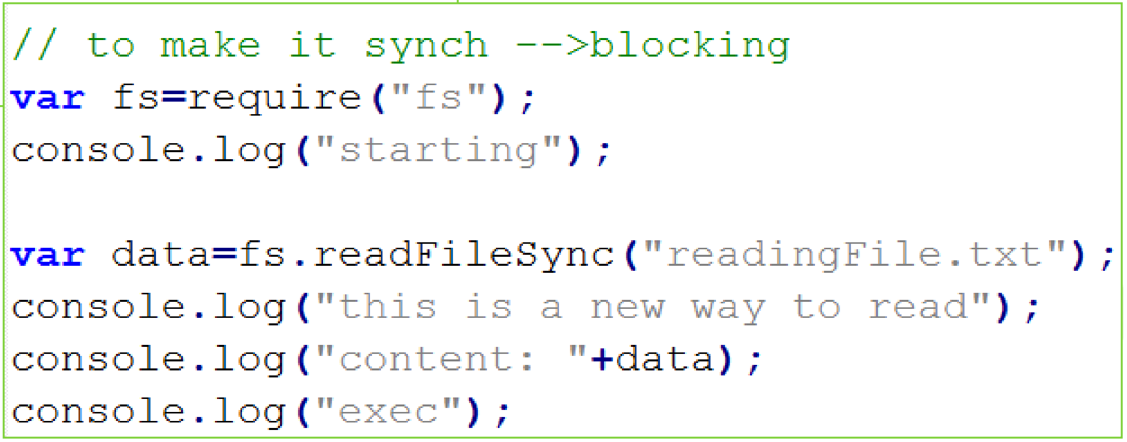
- All its methods have asynchronous and synchronous forms.



```
//file is already created
//this happens Asynch-->non blocking
var fs=require("fs");
console.log("starting");

fs.readFile("readingFile.txt",function(err,data){
    console.log("this is a new way to read");
    console.log("content: "+data)

});
console.log("exec");
```



```
// to make it synch -->blocking
var fs=require("fs");
console.log("starting");

var data=fs.readFileSync("readingFile.txt");
console.log("this is a new way to read");
console.log("content: "+data);
console.log("exec");
```

Node.js Modules

File System Module Demo

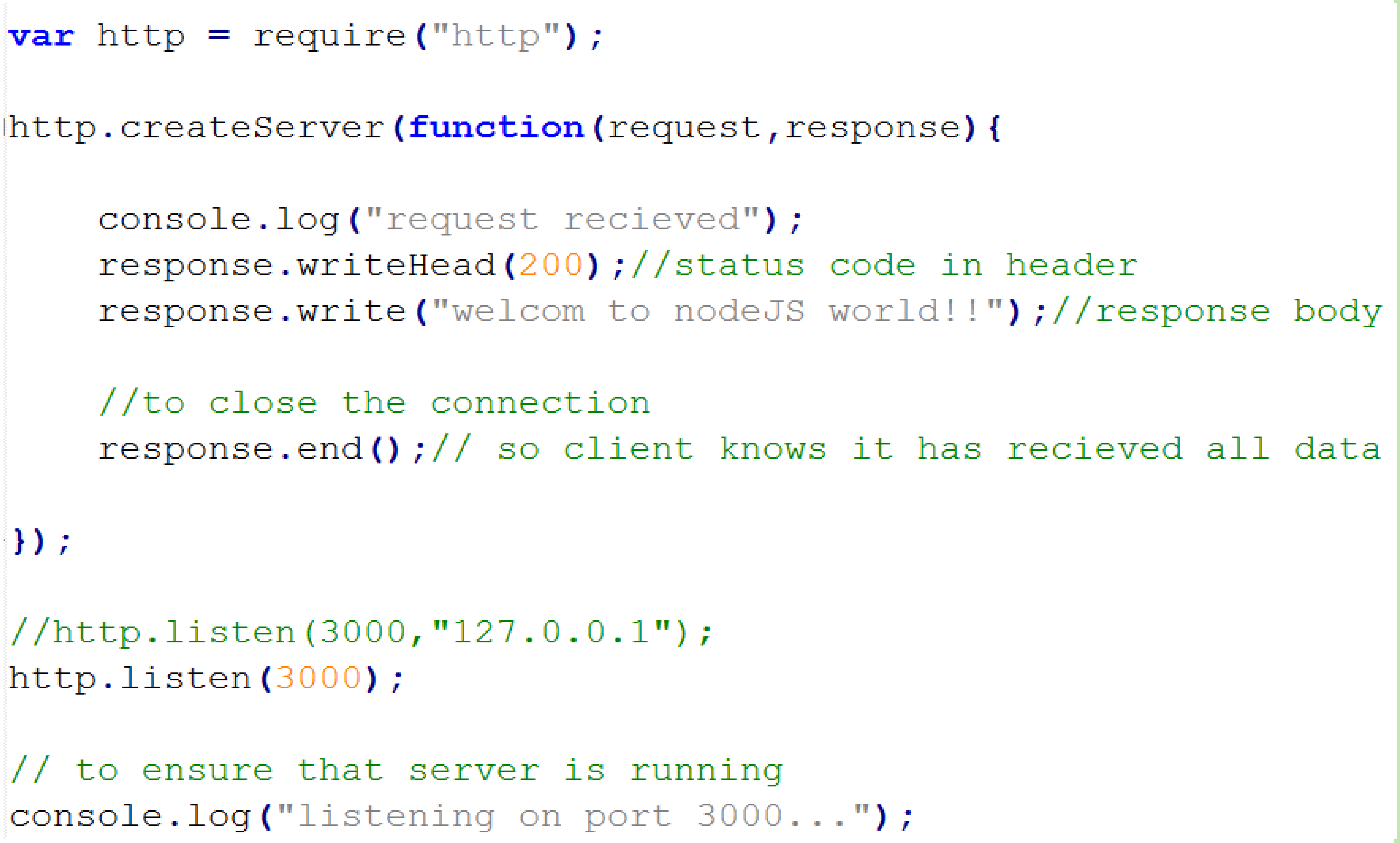
STREAMS

- Streams are like a channels where data can simply flow
- Streams are objects that let you read data from a source or write data to a destination
- All streams are EventEmitters
- Streams can be either Readable, Writable, or both (Duplex)…
- Piping is a mechanism where we provide output of one stream (readable) as the input to another stream (writable).

Node.js Modules

Streams Demo

HTTP Module



```
var http = require("http");

http.createServer(function(request,response){

    console.log("request recieved");
    response.writeHead(200);//status code in header
    response.write("welcom to nodeJS world!!");//response body

    //to close the connection
    response.end();// so client knows it has recieved all data

});

//http.listen(3000,"127.0.0.1");
http.listen(3000);

// to ensure that server is running
console.log("listening on port 3000...");
```

Assignments

Assignment

- Start your http server
- Let the user's request url segment is containing operation (add, sub, multip, div) and 2 values.
- Save in .json the opertation,2 values(from the url) and ,message(custom message indicating the operation of 2 values and its result)
- Send the output result in the response while adding a json object containing {operation,val1,val2,message}
- Bonus: handle operations for more that 2 values