NODE PROGRAM

NODE BASICS

NODE.JS VERSION: 5.1
LAST UPDATED: JAN 2016

NODE.JS

"EVERYTHING IS A CALLBACK... IN NODE EVERYTHING IS NON-BLOCKING AND SO IT DOESN'T ALLOW YOU TO JUST SIT THERE AND THEN RETURN THE RESPONSE. "

– RYAN DAHL HTTP://WWW.INFOQ.COM/
INTERVIEWS/NODE-RYAN-DAHL

NODE LANGUAGE

NODE IS NOT JAVASCRIPT, BUT BOTH JAVASCRIPT AND NODE ARE ECMASCRIPT.

global OBJECT

THE GLOBAL OBJECT IS SIMILAR TO WINDOW OBJECT IN THE BROWSER JAVASCRIPT.

```
global.require() === require()
global.process === process
global.console === console
```

process OBJECT

IN NODE, INTERACTION WITH THE CURRENT NODE PROCESS TAKES PLACE
VIA THE process OBJECT.

AS THE process OBJECT IS A GLOBAL, IT IS ACCESSIBLE FROM ANYWHERE
IN THE APPLICATION SOURCE CODE.

NOTE: IT IS AN INSTANCE OF EVENTEMITTER – LOGIC CAN

ENVIRONMENT VARIABLES

ENVIRONMENT VARIABLES CAN BE ACCESSED VIA THE env ATTRIBUTE:

```
console.log(process.env);

{ SHELL: '/bin/bash',
  USER: 'jordan',
  HOME: '/home/jordan',
  ...
}
```

COMMAND-LINE ARGUMENTS

SHELL COMMANDS ACCEPT ARGUMENTS TO ALTER THEIR BEHAVIOUR:

```
$ echo "Hello World!" // "Hello World" is the argument here
```

NODE MAKES THESE ARGUMENTS ACCESSIBLE VIA process.argv

COMMAND-LINE OPTIONS

THE `argv` PROPERTY IS AN ARRAY.

THE FIRST TWO ELEMENTS ARE 'NODE' AND THE APPLICATION'S NAME:

```
$ node app.js arg1 arg2 arg3=val3

/*
    process.argv => [
        'node', 'app.js', 'arg1',
        'arg2', 'arg3=val3'
    ]
*/
```

EXITING A PROCESS

TO EXIT A PROCESS, USE THE `exit` FUNCTION

```
process.exit();
```

EXITING WITH ERRORS

EXIT CODES CAN ALSO BE SPECIFIED

```
// this process exits successfully
process.exit(0);

// this process failed
process.exit(1);

// this process failed with a different code
process.exit(127);
```

EXITING WITH ERRORS

NOTE:

> DIFFERENT FAILURE CODES CAN BE USED TO DIFFERENTIATE TYPES OF
FAILURE

> KNOWING HOW AN APPLICATION FAILED ALLOWS THE DEVELOPERS THE MEANS
TO PROGRAM AN APPROPRIATE RESPONSE

CHILD PROCESSES

A CHILD PROCESS IS A PROCESS CREATED BY ANOTHER PROCESS.

TO HAVE NODE APPLICATIONS RUN OTHER PROCESSES, USE THE `child process` MODULE.

EXECUTE A PROCESS

THE `exec` FUNCTION RUNS A SHELL COMMAND, AND INVOKES A CALLBACK
WITH REFERENCES TO THE CHILD PROCESS' STANDARD OUTPUT AND ERROR

```javascript
var cp = require('child_process');
var ps = cp.exec('ps aux', function (err, stdout, stderr) {
    console.log('STDOUT: ', stdout); // data written to stdout
    console.log('STDERR: ', stderr); // data written to stderr
});
```

EXEC CALLBACK

THE exec CALLBACK ALSO PROVIDES AN ERROR OBJECT AS ITS FIRST ARGUMENT, WHICH CAN BE ANALYZED IN THE EVENT PROCESS EXECUTION FAILS.

```javascript
var ps = cp.exec('nonexistant-command', function (err, stdout, stderr) {
    if (err) {
        // stack trace
        console.log(err.stack);

        // exit code
        console.log(err.code);
    }
});
```

SPAWN A PROCESS

TO ACCESS A CHILD PROCESS' STREAMS, USE THE spawn FUNCTION

```javascript
var spawn = require('child_process').spawn;
var echo  = spawn('echo', ['Manage child processes in Node']);
var tee   = spawn('tee', ['spawn.txt']);

echo.stdout.on('data', function (data) {
  tee.stdin.write(data);
});

echo.stderr.on('data', function (data) {
  console.log('echo stderr: ' + data);
});

echo.on('close', function (code) {
  tee.stdin.end();
});
```

NOTES ON CHILD PROCESS

> `spawn` RETURNS AN INSTANCE OF THE `ChildProcess` CLASS

> `ChildProcess` INHERITS FROM `EventEmitter` AND IS A PRIVATE CLASS

> TO WRITE TO A SPAWNED PROCESS, USE ITS STDIN'S `write` FUNCTION

> CLOSE THE INPUT STREAM TO ALLOW THE SPAWNED PROCESS

MODULES

MODULES IN BROWSER

DON'T EXIST NATIVELY UNTIL ES6, I.E., NO BUILT-IN MODULE SUPPORT!

MODULES IN BROWSER WORKAROUNDS

> `<script>`

> COMMONJS

> AMD (REQUIREJS)

> ES6

Module Loaders in Browser

> SystemJS

> RequireJS

> Browserify

> ES6-module-loader

More info: http://mzl.la/1IEU8ZM and http://mzl.la/1IEU7VZ

MODULES IN NODE

BUILT-IN MODULES WITH require A COMMONJS NOTATIOIN! 💝

NODE REQUIRE EXAMPLE:

```js
var express = require('express');
var app = express();
```

REQUIRING MODULES

MODULES CAN LIVE DIFFERENT PLACES WITH JAVASCRIPT. THEY CAN BE ON LOCAL MACHINES, VIRTUAL MACHINES, SERVERS, REMOTE URI LOCATIONS, OR ANYWHERE REALLY.

LOADING NODE MODULES

THESE MODULES CAN BE LOADED WITH MODULE LOADERS LIKE REQUIRE OR VIA INVERSION OF CONTROL PATTERNS.

```js
var filesystem = require('fs'),
    databaseConfigs = require('./configs/database.json'),
    routes = require('../routes'),
    server = require('./boot/server.js')
```

CREATING A MODULE WITH A "CLASS" EXAMPLE:

```javascript
function UserController() {
  var username, password;
  function doLogin(user,pw) {
    username = user;
    password = pw;
    // do the rest of the login work
  }
  var publicAPI = {
    login: doLogin
  };
  return publicAPI;
}
// create a `UserController` instance
var ctrl = UserController();
ctrl.login( "fred", "12Battery34!" );
```

NODE PATTERNS FOR MODULE EXPORTS

> `module.exports = function(ops) {...}`

> `module.exports = {...}`

> `exports.methodA = function(ops) {...}`

> `exports.obj = {...}`

FUNCTION PATTERN

module.js:

```javascript
module.exports = function(options) {
  var limit = 100
  if (options.type === 'foobar') {
    limit = 200;
  }
  return {
    name: 'request',
    limit: limit,
    type: options.type,
    method: function(data) { return data; }
  };
};
```

FUNCTION PATTERN

main.js:

```
var mod = require('./module.js');
var request = mod({
  type:'foobar'
});
request.method({
  x: 10,
  y: 20
});
```

PROTOTYPAL INHERITANCE

validator.js:

```js
var Validator = exports.Validator = function() {
    this._rules = [];
};


Validator.prototype.addRule = function(rule) {
    this._rules.push(rule)
};


Validator.prototype.validate = function(instance) {
    ...
};
```

PROTOTYPAL INHERITANCE

specificValidator.js (UTIL.INHERITS):

```
var SpecificValidator = function() {
  Validator.call(this);
};


require('util').inherits(SpecificValidator, Validator);

SpecificValidator.prototype.filter = function(instance) {
  ...
};
```
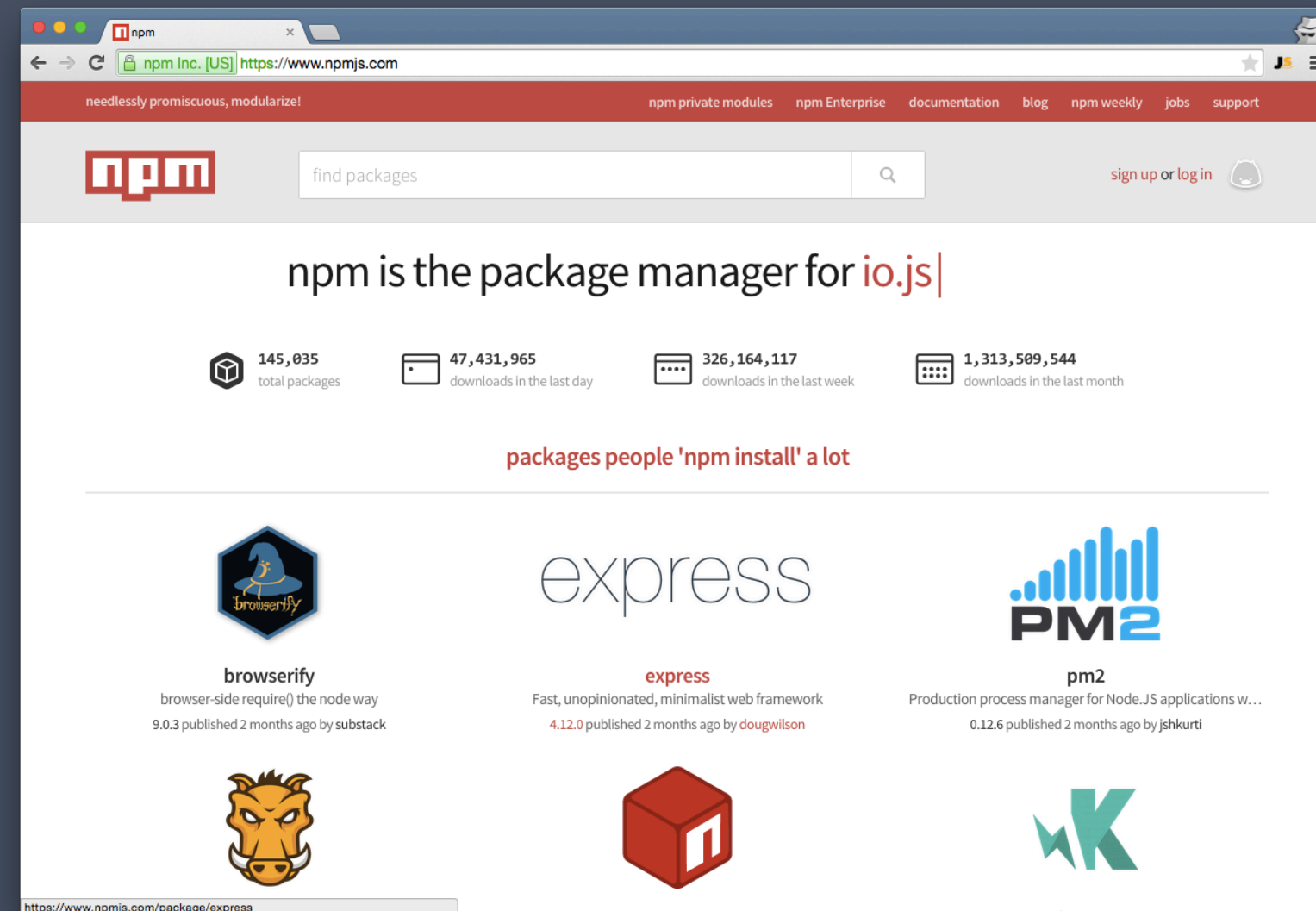
PROTOTYPAL INHERITANCE

client.js:

```
var validator = new SpecificValidator();

validator.filter( { ... } );    // Calls function on derived object

validator.validate( { ... } ); // Calls function on base object
```

HOW-TO EXAMPLES

> DAY1-06-MODULES-STEP1.JS

> DAY1-07-MODULES-STEP2.JS (WHICH REFERENCES THE GREETINGS.JS FILE)

NPM

MEET THE BEAST!

WHAT IS NPM?

A PACKAGE MANAGER FOR NODE.

> WEBSITE: <u>HTTPS://WWW.NPMJS.COM/</u>

> COMMAND-LINE TOOL: npm

> REGISTRIES: PUBLIC AND PRIVATE

Introduction to NPM

Two ways to install a module:

> **Locally: most of your projects' dependencies, e.g., Express, Request, Hapi**

```
npm install module-name
```

> **Globally: comman-line tools only (mostly), e.g., Mocha, Grunt, SLC**

INSTALLING PACKAGES

```
$ npm install express
$ npm install express@4.2.0
$ npm install express@latest
$ npm install express --save
$ npm install mocha --save-dev
$ npm install grunt -g
$ sudo npm install grunt -g
```

PACKAGE.JSON

CREATING PACKAGE.JSON: RUN INIT ACTION TO INTERACTIVELY CREATE A PACKAGE.JSON

```
$ npm init

This utility will walk you through creating a package.json
file.  It only covers the most common items, and tries to
guess sane defaults.

See `npm help json` for definitive documentation on these
fields and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package
and save it as a dependency in the package.json file

Press ^C at any time to quit
name: (my-package-name)
```

PACKAGE.JSON

```json
{
  "name": "my-cool-app",
  "version": "0.1.0",
  "description": "A gret new application",
  "main": "server.js",
  "dependencies": {
    "express": "~4.2.0",
    "ws": "~0.4.25"
  },
  "devDependencies": {
    "grunt": "~0.4.0"
  }
}
```

NPM

WHEN RUNNING `npm install` NAME IN A FOLDER:

> NMP LOOKS FOR NODE_MODULES OR PACKAGE.JSON

> IF NOTHING IS FOUND IT GOES UP THE TREE

THEREFORE, IN AN EMPTY FOLDER, CREATE PACKAGE.JSON OR NODE_MODULES DIR FIRST.

HOW-TO EXAMPLE

SAMPLE CODE:

DAY1-11-NPM-CLI.TXT

PUBLIC MODULES & REGISTRIES

SET CONFIG VALUES...

```
$ npm set init.author.name "Your Name"
$ npm set init.author.email "you@example.com"
$ npm set init.author.url "http://yourblog.com"
```

SIGN UP ON THE NPM WEBSITE AND ADD YOURSELF:

```
$ npm adduser
```

PUBLISHING YOUR MODULE

ADD `package.json` (MAYBE TRY `npm init`)

THEN PUBLISH!

```
$ cd my-cool-app
$ npm publish
```

PRIVATE REGISTRIES

> HOSTED BY NPMJS
> HOSTED BY YOU

ADVANTAGES OF PRIVATE:

CODE IS NOT EXPOSED TO OUTSIDE AND NO EXTERNAL DEPENDENCIES (IF SELF-HOSTED)

(THERE ARE STRATEGIES FOR DEPLOYMENT, E.G., TAR FILE)

TO LIST CURENTLY INSTALLED NPM MODULES, USE THE LS ACTION LS LISTS OUT MODULES LOCAL TO THE CURRENT NODE PROJECT

```
$ npm ls
/home/johndoe/node-app
|__ q@1.0.1
```

TO LIST GLOBALLY INSTALLED MODULES, ADD THE -G FLAG

```
$ npm ls -g
/usr/lib
|__ bower@1.3.11
  |__ abbrev@1.0.5
  |__ archy@0.0.2
|__ semver@4.0.0
```

SEARCH

SEARCH FOR NPM MODULES VIA THE SEARCH ACTION

```
$ npm search [keyword]
```

THIS ACTION CARRIES OUT SEVERAL TASKS

1. QUERIES THE NPM REGISTRY

2. RETRIEVES SEARCH RESULTS

3. PRINTS IT OUT TO STANDARD OUTPUT

UPDATE

TO UPDATE AN NPM MODULE, USE THE UPDATE COMMAND

```
$ npm update mysql
```

UPDATING ONLY WORKS IF THE MODULE HAS ALREADY BEEN INSTALLED

REMOVE A MODULE

TO REMOVE AN NPM MODULE

```
$ npm rm mysql
```

TO REMOVE A GLOBAL MODULE

```
$ npm rm mysql -g
```

PACKAGING

MODULE PACKAGING IN NODE IS DONE USING A `package.json` FILE
THERE ARE MANY OPTIONS THAT CAN BE CONFIGURED:

> NAME

> VERSION NUMBER

> DEPENDENCIES

PRIVATE MODULES

THE PRIVATE ATTRIBUTE PREVENTS ACCIDENTAL PUBLISHING

```json
{
    "name" : "my-private-module",
    "version": "0.0.1",
    ...
    "private": true,
    ...
}
```

NPM ENTERPRISE

WHEN TO USE -g?

A: ONLY FOR COMMAND-LINE TOOLS. THEY USUALLY HAVE bin IN PACKAGE.JSON:

```
{
  "name": "stream-adventure",
  "version": "4.0.4",
  "description": "an educational stream adventure",
  "bin": {
    "stream-adventure": "bin/cmd.js"
  },
  "dependencies": {
    ...
```

HELLO WORLD

WEB CONTENT

TYPES OF WEB CONTENT

> **STATIC**

> **DYNAMIC**

STATIC CONTENT

STATIC CONTENT IS INCLUSIVE OF THINGS LIKE IMAGE FILES, STATIC HTML FILES THAT ARE ALREADY PUT TOGETHER, AND OTHER RELATED CONTENT THAT IS STORED ON SOME STYLE OF DRIVE STORAGE AND AVAILABLE FOR IMMEDIATE RETURN TO A REQUESTOR VIA GENERAL RESPONSE.

DYNAMIC CONTENT

DYNAMIC CONTENT, WHICH IS THE CONTENT THAT IS PUT TOGETHER – OR GENERATED – BY CODE PULLING TOGETHER DATA FROM DATA SOURCES OR OTHER MEANS, AND THEN PROVIDED TO THE REQUESTOR.

I/O

NODE.JS IS EXCELLENT AT DYNAMIC GENERATION AND RETURNING CONTENT THAT IS PURE I/O IN THE SENSE OF BUILT OR DYNAMIC CONTENT.

FOR STATIC CONTENT LIKE IMAGE FILES AND RELATED CONTENT IT IS ACTUALLY A GREAT BENEFIT TO HAND THAT OFF TO SERVER SOFTWARE THAT CAN HANDLE THE SPECIFIC OS LEVEL REQUEST.

THE SERVER, REQUEST, RESPONSE OBJECTS

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

RUNNING THE APP

RUN WITH:

```
$ node server.js
```

SEND REQUESTS:

```
$ curl http://localhost:1337
```

OR

HTTP://LOCALHOST:1337

HTTP OBJECT

HTTP OBJECT:

HTTPS://NODEJS.ORG/API/HTTP.HTML
HTTPS://NODEJS.ORG/API/
HTTP.HTML#HTTP_CLASS_HTTP_SERVER

```
var server = http.createServer([requestListener])
server.listen(port[, hostname][, backlog][, callback])
```

HTTP RESPONSE

```
response.writeHead(200, {
  'Content-Length': body.length,
  'Content-Type': 'text/plain' });
```

HOW-TO EXAMPLE

SAMPLE CODE (code/node):

1. DAY1-08-NODEJS-APP.JS
2. DAY1-09-NODEJS-APP2.JS
3. DAY1-11-NODEJS-STATIC-SERVER.JS

QUESTIONS AND EXERCISES

👍

WORKSHOP

ONE HAND: HELP, TWO HANDS: FINISHED ONE TASK

```
$ sudo npm install learnyounode -g
```