The New York
Code + Design
Academy

MODULES AND PACKAGES

Agenda

- Node Modules
- Module Loaders
- Node Packages
- Node Package Manager

Node Modules

What Are Node Modules?

A module encapsulates related code into a single unit of code. When creating a module, this can be interpreted as moving all related functions into a file.[1]

[1] https://www.sitepoint.com/understanding-module-exports-exports-node-js/

Benefits of Modules

- Reusable Code
- Easier to Maintain Code
- Avoid Namespace Pollution

Core Modules

Node comes with certain modules already build it in, which are sometimes referred to as **Core Modules**.

- `fs` - includes events, classes and functions needed to work with file input/output
- `http` - includes events, classes and functions needed to create a HTTP server
- `util` - includes functions that assist the developer with debugging, inspecting and deprecating code.
- `path` - includes fuctions for working with file and directory paths

External Modules

- External modules are sometimes referred to as **Third Party Modules**
- These are modules created by other developers which they make publicly available
- These modules can save developers a lot of time

Custom Modules

- Custom modules are usually specific to a developers app
- This can as simple as a greeting module

```
// module.js
    console.log('Hello Class!');

// app.js
    require('./module.js');
```

Module Loaders

RequireJS

RequireJS is a JavaScript file and module loader. Using a modular script loader like RequireJS will improve the speed and quality of your code.[2]

[2] http://requirejs.org/

`require()`

- `require()` is a function that points to the path of a module that will be used
- If a path is specified, `require` will traverse to it and look for the module there
- `require` supports relative and absolute paths

Importing a Module

- Here, the first `require` looks for a file called `models.js`, located in the same directory as the current file
- The second `require` looks for a file on an absolute path

```
var models = require('./models');
var absolute = require('/some/absolute/path/models');
```

Exporting a Module

- Whatever `module.exports` is set to define, is what is available when including a module is your app
- `module.exports` is returned to the requiring file
- `exports` collects properties, ultimately attaching them to `module.exports`

```
//module_one.js
exports.foo = 'bar';
```

```
//app_one.js
var a = require('./module_one');

console.log(a.foo);
```

```
//module_two.js
module.exports = 'hello';
```

```
//app_two.js
var b = require('./module_two')

console.log(b)
```

Simple Example

- Exports can be a single function

```
//make `helloWorld` callable via `require`!
function helloWorld(){
    return "Hello World!";
};
//export the `helloWorld` method directly
module.exports = helloWorld;
```

Multiple Functions Example

- Exports can also be an object, in this case an object wrapping local functions

```
//make `helloWorld` _and_ `helloPerson` callable via `require`
function helloWorld(){
    return "Hello World!"
};
function helloPerson(name){
    return `Hello ${name}!`;
};
//wrap the methods in an object and export
//note that the functions could be renamed when exporting
module.exports = {
    helloWorld: helloWorld,
    helloPerson: helloPerson
};
```

Pass parameters to Module Example

- Exports could take a parameter, for instance when setting up a configuration

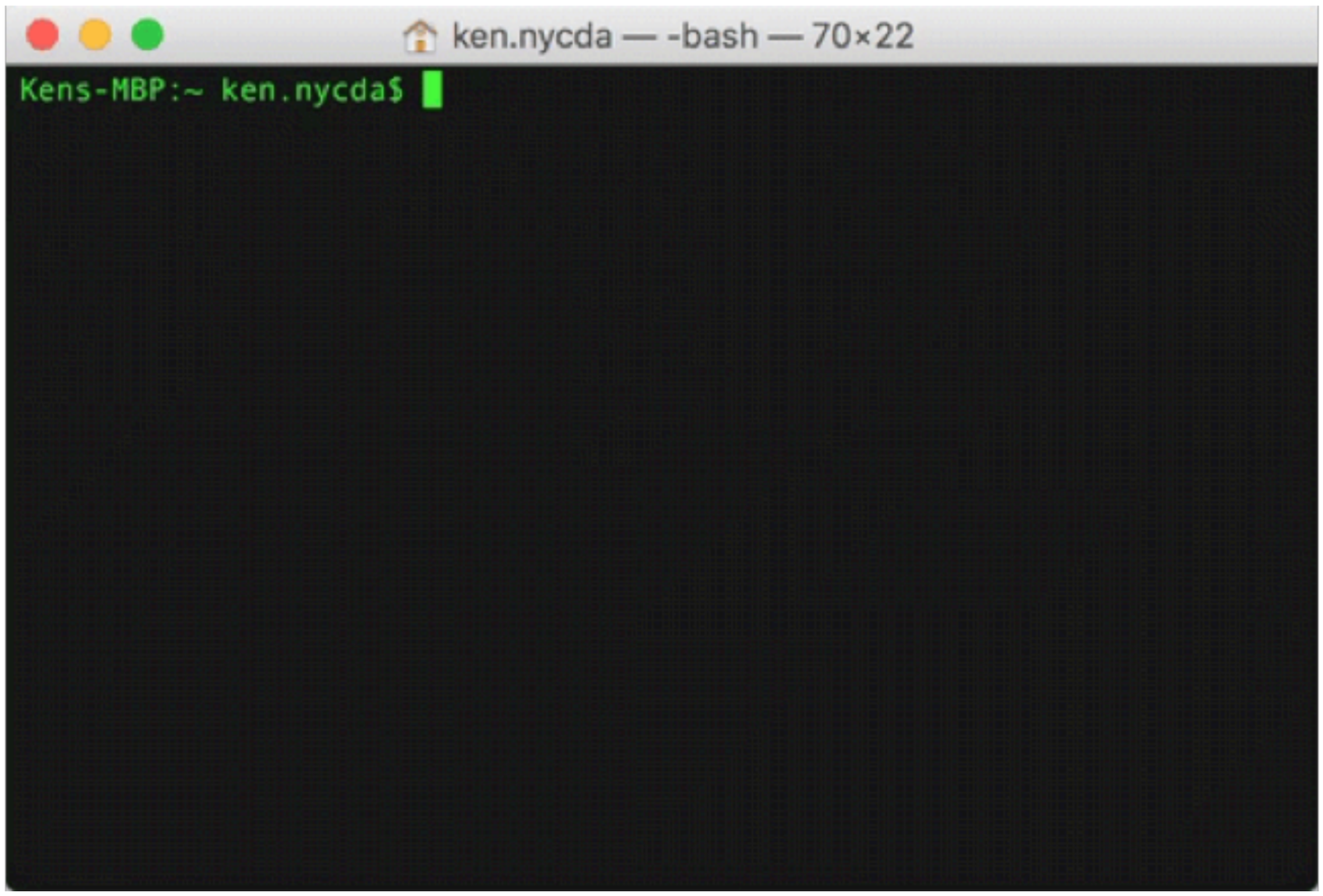
```
//this is in my-module.js
var extraInformation;
function helloWorld(){
    return "Hello world!";
};
function helloPerson(name){
    return `Hello ${name}. ${extraInformation}`;
};

module.exports = function(info){
    extraInformation = info;
    return {
        helloWorld: helloWorld,
        helloPerson: helloPerson
    };
};

//the following is in app.js at the same directory as my-module
var myModule = require('./my-module')(`It's a beautiful day.`);
//prints "Hello Sally.  It's a beautiful day."
console.log(myModule.helloPerson('Sally'));
```

Install RequireJS

```
npm install requirejs
```



Exercise: Build Your Own Custom Module

Step 1: Create a module for each of the following:

- *Set Difference* - Given two arrays of strings, produce a single array of items that are in one or the other but not both.
- *Set Intersection* - Given two arrays of strings, produce a single array of unique items that are in both sets.

Step 2: Create a file called app.js and include the below code:

```
var setDifference = require('./set-difference');
var setIntersection = require('./set-intersection');

var set1 = ['dogs', 'cats', 'red', 'bananas', 'code', 'movies'];
var set2 = ['blue', 'horses', 'dogs', 'code', 'rain'];

var difference = setDifference(set1, set2);
var intersection = setIntersection(set1, set2);

//should print an array with cats, red, bananas, movies, blue, rain as elements
console.log(difference);

//should print an array with dogs and code as elements
console.log(intersection);
```

Node Packages

What are Node Packages?

- Packages are one or more modules that have been packaged together.
- The list of modules in a package is defined in `package.json`

`package.json`

- Defines what packages your package depends on
- Defines the name and version of your package
- Defines what file gets executed when the package is required
- Holds basic licensing information
- Gives author and repository information
- Optionally defines scripts and tasks (ie, `npm run [task]`)

npm Node Package Manager

Remember typing the npm *command when you installed RequireJS?*

- The NPM registry is a central place to get open-source modules, preventing you from reinventing the wheel
- Allows for installation from several types of location including the NPM registry, git, symbolic link, or a tarball archive
- Helps manage installed modules

`npm init`

- Creates a `package.json` for your project
- Interactive prompts for important information about your project - author information, repository, package name, etc

`npm install`: local

- Installs the package locally to `node_modules`
- Once installed, the package can be pulled into your source with `require` by name
- If you run `npm install` with `--save`, the dependency will be saved in your `package.json`

```
npm install --save express
```

Once install is complete:

```
//express is now `require`able from node_modules by name
var express = require('express');
```

`npm install`: global

- Installs the package globally, defaults to `.npm/` under your home directory
- Use the `-g` flag to denote a global install
- Usually used for command-line tools

```
# install `http-server`, a tool that exposes folders and files over HTTP

        npm install -g http-server

# run `http-server` on your home directory

        http-server ~
```

`npm uninstall`

- Follows the same rules as `install`
- Uninstalls (removes) the package from `node_modules`
- `-g` uninstalls from the global packages
- `--save` will remove the dependency from `package.json`

```
# uninstall the global `http-server` from the previous example

        npm uninstall -g http-server

# running `http-server` will now error
```

`npm ls`

- Shows a tree of the installed modules, meaning you can see your dependencies and their dependencies (and so on) at a glance
- Can be run with `-g` to show all globally installed modules
- Can be run with `--depth [number]` to limit how many dependencies-of-dependencies you see

```
npm ls

npm ls --depth 0
```

Exercise #1: Create a package

Step 1: Create a directory called `my-module` with a file called `my-module.js`. Inside of `my-module.js`, make a call to `require` to pull in `express`.

Step 2: When finished, run `npm init` answering the prompts, then install Express, saving it to your `package.json`.

Step 3: Finally, run `npm ls` and `npm ls --depth 0`, observing the differences.

Exercise #2: Countdown Timer

Create a countdown timer app and deploy it using Node HTTP Server

- Create folder on Desktop
- Folder should have an **`index.html`** file and a script file
  - The script file is where your countdown timer code will live
- Install this Node HTTP-Server https://github.com/indexzero/http-server