Tutorial 0.2

Node modules with examples

>>> Node@1 :: what we will cover in this tutorial?

1. Node modules:
    a. Core
    b. Third Party
    c. Local
2. Introduction to ‘fs’ module
3. Introduction to ‘http’ module

>>> Node modules

>>> Node@2 :: what is node module?

Node modules can be considered as a JavaScript library. A set of functions you want to include in your application.

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.

Node.js implements CommonJS modules standard. CommonJS is a group of volunteers who define JavaScript standards for web server, desktop, and console application.

>>> Node@3 :: types of node modules?

Node.js includes three types of modules:

1. Core Modules
2. Local Modules
3. Third Party Modules

>>> Node@4 :: CORE modules

Node.js is a lightweight framework. The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and load automatically when Node.js process starts. However, you need to import the core module first in order to use it in your application.

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

>>> Node@5 :: CORE modules

In order to use Node.js core (or NPM modules), you first need to import it using require() function as shown below.

```
var module = require('module_name');
```

As per above syntax, specify the module name in the require() function. The require() function will return an object, function, property or any other JavaScript data type, depending on what the specified module returns. You will see examples in the next slides with [fs] and [http] modules.

>>> Node@6 :: THIRD PARTY modules for node

We usually download third party libraries from online/cloud package repositories. To download and use third party modules in our project/app we can use NPM.

The biggest cloud repository for third party modules is npmjs.com

>>> Node@7 :: THIRD PARTY modules for node

To download third party modules we use NPM command: `npm install <package_name> --save`

Note that to use `--save` you must have `package.json` file created inside your project folder.

>>> Node@8 :: THIRD PARTY modules for node :: practice

1. Create project folder <any_name>
2. Create inside this folder package.json file with proper structure.
3. Install <animal-sounds> third party library using command npm install animal-sounds --save

   Check https://www.npmjs.com/package/animal-sounds **USAGE** part to see what it does and how you can use it.
4. Use it in your app.

More detailed steps for previous slide

1. Run cmd.exe
2. Use command `dir` to list files and folder in current directory
3. Navigate to desired folder using commands e.g. `cd <directory name>`
4. Create a folder using command e.g. `mkdir task1`
5. Navigate to newly created folder using command `cd task1`
6. Run command `npm init` to create package.json file and specify the structure by answering the questions that follow
7. Run command `npm install animal-sounds --save` to install <animal-sounds> third party library
8. Check now your folder, by running command dir (or in a Window interface) and find animals-sounds folder in node_modules, which contains newly installed animal sounds
9. Now write some code in index.js file where you will use this library
   NB: Don't forget to import the module, e.g. `var animalSounds = require ('animal-sounds');`

>>> Node@9 :: LOCAL modules

Local modules are the modules created locally in your Node.js application. These modules include different functionalities of your application in separate files and folders.

Basically, these modules are .js files with different functions that can be used in your main app file.

To use them you must export them (make available for other .js files) and require them where needed.

>>> Node@11 :: LOCAL modules :: example

logger.js

```
var log =  {
             info    : function (info) { console.log('Info: ' + info); },
             warning: function (warning) { console.log('Warning: ' + warning); },
             error   : function (error) { console.log('Error: ' + error); }
           };


module.exports = log
```

app.js

```
var log = require(‘./logger.js’)


log.info(‘Hello World!’)
```

>>> Node@12 :: Local modules for node :: practice

1. Create project folder <any_name>
2. Inside your folder create app.js -> this will be your main file
3. Also create random.js file -> this will be your local module for node app. Inside this file create function that will return random number and export it.
4. Require it in your main app.js file and generate some random numbers.

>>> Introduction to ‘fs’ core module

>>> Node@13 :: File system node core module

Node.js includes **fs** module to access physical file system. The **fs** module is responsible for all the asynchronous or synchronous file I/O operations.

To include the fs module, use the `require('fs')` method.

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

>>> Node@14 :: File system node core module

One of the functions of fs module is writeFile(filename, content, callback) which can create and write data to a file. Below example:

app.js

```
var fs = require('fs')

fs.writeFile ('text.txt', 'Hello', function (err) {
      if (err) {
            console.log(err)
      }

      console.log('Data written to file text.txt')
})
```

terminal

```
> node app.js
> Data written to file text.txt
```

text.txt

```
Hello
```

>>> Node@15 :: Some of the file system module functions

Function	Description
fs.readFile(fileName [,options], callback)	Reads existing file.
fs.writeFile(filename, data[, options], callback)	Writes to the file. If file exists then overwrite the content otherwise creates new file.
fs.open(path, flags[, mode], callback)	Opens file for reading or writing.
fs.rename(oldPath, newPath, callback)	Renames an existing file.
fs.stat(path, callback)	Returns fs.stat object which includes important file statistics.
fs.rmdir(path, callback)	Renames an existing directory.
fs.mkdir(path[, mode], callback)	Creates a new directory.
fs.readdir(path, callback)	Reads the content of the specified directory.
fs.exists(path, callback)	Determines whether the specified file exists or not.
fs.appendFile(file, data[, options], callback)	Appends new content to the existing file.

>>> Node@16 :: Core library for node :: practice

1. Create project folder <any_name>
2. Inside your folder create app.js -> this will be your main file
3. Inside app.js require ‘fs’ module. Using writeFile function of fs module create file and save some data in it.

>>> Introduction to ‘http’ core module

>>> Node@17 :: HTTP node core module

HTTP module allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

To include the HTTP module, use the `require('http')` method.

>>> Node@18 :: HTTP node core module

One of the functions of http module is createServer() that helps you to run a simple server and serve for instance html code. Below example:

app.js

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

var server = http.createServer (function (request, response)
{
 response.write('<h1>Hello World!<h1>')
 response.end()
})
server.listen(5000, function () {
 console.log('Server is running on port 5000')
})
```

terminal

```
> node app.js
> Server is running on port 5000
```

Browser

Hello World!

>>> Node@19 :: Core library for node :: practice

1. Create project folder <any_name>
2. Inside your folder create app.js -> this will be your main file
3. Inside app.js require ‘http’ module. Using createServer function http module run local server on port 3000 and serve html code.
4. Create index.html inside your project folder. Add some html code inside this file. Serve this file using your local server. In this case you have to use ‘fs’ module’s readFile() or readFileSync() functions to read the content of index.html file.

```
>>> See you
```