**Lesson 1**

Write a program that prints the text "HELLO WORLD" to the console (stdout)

**Solution 1**

console.log("HELLO WORLD");

**Lesson 2**

Write a program that accepts one or more numbers as command-line arguments and prints the sum of those numbers to the console (stdout).

**Solution 2**

var data = process.argv

var info = data.slice(2);

function sum(array){

var result = array.reduce(function(a,b){

a = parseInt(a);

b = parseInt(b);

return a+b;

},0);

console.log(result);

};

sum(info);

**Lesson 3 -** MY FIRST I/O

Write a program that uses a single synchronous filesystem operation to

read a file and print the number of newlines (\n) it contains to the

console (stdout), similar to running cat file | wc -l.

The full path to the file to read will be provided as the first

command-line argument (i.e., process.argv[2]). You do not need to make

your own test file.

**Notes:** All synchronous (or blocking) filesystem methods in the fs module end with 'Sync'.

**Solution 3**

var linkto = process.argv[2]; // get path for the file

var fs = require("fs");

var data = fs.readFileSync(linkto); // read file

var info = data.toString(); // turn file into string

var count = info.split("\n").length-1; // split the string by \n, and the count the number of iten in the array.

console.log(count);

**Lesson 4 -** MY FIRST ASYNC I/O! (Exercise 4 of 13)

Write a program that uses a single asynchronous filesystem operation to read a file and print the number of newlines it contains to the console (stdout), similar to running cat file | wc -l.

The full path to the file to read will be provided as the first command-line argument.

**Solution 4**

var linkto = process.argv[2]; // get path for the file

var fs = require("fs");

fs.readFile(linkto, function(err,data){

if(err){

console.log(err);

};

var info = data.toString(); // turn file into string

var count = info.split("\n").length-1; // split the string by \n, and the count the number of iten in the array.

console.log(count);

});

**Lesson 5 –**  ## FILTERED LS

Create a program that prints a list of files in a given directory, filtered by the extension of the files. You will be provided a directory name as the first argument to your program (e.g. '/path/to/dir/') and a file extension to filter by as the second argument.

For example, if you get 'txt' as the second argument then you will need to filter the list to only files that end with .txt. Note that the second argument will not come prefixed with a '.'.

Keep in mind that the first arguments of your program are not the first values of the process.argv array, as the first two values are reserved for system info by Node.

The list of files should be printed to the console, one file per line. You must use asynchronous I/O.

**Solution 5**

var fs = require("fs");

var linkto = process.argv[2]; // get path for the file

var ending = "."+process.argv[3] // get the ending extension to match with

var endinglength=ending.length; //determine the length of ending

fs.readdir(linkto, function(err,data){

if (err){

console.log(err);

};

var filtered = data.filter(function(word){

if(word.substr(-endinglength)==ending){ // test if ending of the word matches with the ending we want

console.log(word)

return

};

});

});

**Lesson 6 –**  ## MAKE IT MODULAR

This problem is the same as the previous but introduces the concept of modules. You will need to create two files to solve this.

Create a program that prints a list of files in a given directory, filtered by the extension of the files. The first argument is the directory name and the second argument is the extension filter. Print the list of files (one file per line) to the console. You must use asynchronous I/O.

You must write a module file to do most of the work. The module must export a single function that **takes three arguments**: the **directory name**, the **filename extension string** and **a callback** function, in that order. The filename extension argument must be the same as what was passed to your program. Don't turn it into a RegExp or prefix with "." or do anything except pass it to your module where you can do what you need to make your filter work.

The callback function must be called using the idiomatic node(err, data) convention. This convention stipulates that unless there's an error, the first argument passed to the callback will be null, and the second will be your data. In this exercise, the data will be your filtered list of files, as an Array. If you receive an error, e.g. from your call to fs.readdir(), the callback must be called with the error, and only the error, as the first argument.

You must not print directly to the console from your module file, only from your original program.

In the case of an error bubbling up to your original program file, simply check for it and print an informative message to the console.

These four things are the contract that your module must follow.

1. Export a single function that takes exactly the arguments described.

2. Call the callback exactly once with an error or some data as described.

3. Don't change anything else, like global variables or stdout.

4. Handle all the errors that may occur and pass them to the callback.

The benefit of having a contract is that your module can be used by anyone who expects this contract. So your module could be used by anyone else who does learnyounode, or the verifier, and just work.

**Solution 6 -**

**My solution**

(for program.js)

var filteringW = require("./mymodule.js")

var linkto = process.argv[2]; // get path for the file

var ending = process.argv[3] // get the ending extension to match with

filteringW(linkto, ending, function (err, list) {

if (err){

return console.error("There was an error:",err);

};

list.forEach(function(word) { // go through each word in the file

console.log(word);

});

});

(for mymodule.js)

var fs = require('fs');

module.exports = function filteringWords(linkto,ending,callback){

fs.readdir(linkto, function(err, data){

if (err){

return callback(err); // return error due to readdir

};

var data = data.filter(function(word){

return word.substr(-ending.length-1)=="."+ending // test if ending of the word matches with the ending we want

});

callback(null, data)

});

}

**Answer per solution**

(For program.js)

var filterFn = require('./solution\_filter.js')

var dir = process.argv[2]

var filterStr = process.argv[3]

filterFn(dir, filterStr, function (err, list) {

if (err) {

return console.error('There was an error:', err)

}

list.forEach(function (file) {

console.log(file)

})

})

(for solution\_filter.js)

var fs = require('fs')

var path = require('path')

module.exports = function (dir, filterStr, callback) {

fs.readdir(dir, function (err, list) {

if (err) {

return callback(err)

}

list = list.filter(function (file) {

return path.extname(file) === '.' + filterStr

})

callback(null, list)

})

}

**Lesson 7 –**  # HTTP CLIENT

Write a program that performs an HTTP GET request to a URL provided to you as the first command-line argument. Write the String contents of each "data" event from the response to a new line on the console (stdout).

**Solution 7 -**

**My solution**

var http = require("http")

var link= process.argv[2]; // get link

http.get(link, function(response){

response.setEncoding("utf8"); // decode response to utf8

response.on('data', console.log);

response.on("error", console.error) //in case there's an error

}).on('error', function(e) {

console.log("Got error: " + e.message);

});

**Learnyounode solution**

var http = require('http')

http.get(process.argv[2], function (response) {

response.setEncoding('utf8')

response.on('data', console.log)

response.on('error', console.error)

}).on('error', console.error)

**Lesson 8 –**  ## HTTP COLLECT

Write a program that performs an HTTP GET request to a URL provided to you as the first command-line argument. Collect all data from the server (not just the first "data" event) and then write two lines to the console (stdout).

The first line you write should just be an integer representing the number of characters received from the server. The second line should contain the complete String of characters sent by the server.

**Solution 8 -**

**My solution**

var http = require("http")

var bl = require("bl");

var link= process.argv[2]; // get link

http.get(link, function(response){

response.pipe(bl(function (err,data){ // pipe(bl(callback)) will automatically compile the data together

if(err){

console.log(err);

}

data=data.toString(); // convert data to string

console.log(data.length) //get length of the data

return console.log(data);

}));

}).on('error', function(e) {

console.log("Got error: " + e.message);

});

**Learnyounode solution**

var http = require('http')

var bl = require('bl')

http.get(process.argv[2], function (response) {

response.pipe(bl(function (err, data) {

if (err) {

return console.error(err)

}

data = data.toString()

console.log(data.length)

console.log(data)

}))

})

**Lesson 9 –**  ## JUGGLING ASYNC

This problem is the same as the previous problem (HTTP COLLECT) in that you need to use http.get(). However, this time you will be provided with three URLs as the first three command-line arguments.

You must collect the complete content provided to you by each of the URLs and print it to the console (stdout). You don't need to print out the length, just the data as a String; one line per URL. The catch is that you must print them out in the same order as the URLs are provided to you as command-line arguments.

**Solution 9 -**

**My solution**

var http = require("http")

var bl = require("bl");

var link = process.argv; //all 5 links

var strArray=[]; //to put the data in

function printorder(array){

for (var i=0;i<link.length-2;i++){

console.log(array[i]); //to print list

}

}

function jugglingC(i){

http.get(link[i+2], function(response){

response.pipe(bl(function (err,data){ // pipe(bl(callback)) will automatically compile the data together

if(err){

console.log(err);

}

data=data.toString(); // convert data to string

strArray[i]=data; // to put data in order in an array

if(strArray.length==3){

printorder(strArray)

}

}));

}).on('error', function(e) {

console.log("Got error: " + e.message);

});

}

for (var i=0;i<link.length-2;i++){

jugglingC(i); //to input each link

}

**Learnyounode solution**

var http = require('http')

var bl = require('bl')

var results = []

var count = 0

function printResults () {

for (var i = 0; i < 3; i++) {

console.log(results[i])

}

}

function httpGet (index) {

http.get(process.argv[2 + index], function (response) {

response.pipe(bl(function (err, data) {

if (err) {

return console.error(err)

}

results[index] = data.toString()

count++

if (count === 3) {

printResults()

}

}))

})

}

for (var i = 0; i < 3; i++) {

httpGet(i)

}

**Lesson 10 –**  ## TIME SERVER

Write a TCP time server!

Your server should listen to TCP connections on the port provided by the first argument to your program. For each connection you must write the current date & 24 hour time in the format:

"YYYY-MM-DD hh:mm"

followed by a newline character. Month, day, hour and minute must be zero-filled to 2 integers. For example:

"2013-07-06 17:42"

After sending the string, close the connection.

**Solution 10 -**

**My solution**

var net = require('net');

var gport = process.argv[2]; //port to use

function cut2(word){

return word.substr(-2); // to only get the last 2 char

}

function currentTime(){

var date = new Date;

var time="";

time+=date.getFullYear()+"-"; //add year

time+=cut2("0"+(date.getMonth()+1))+"-"; //add Month, **note**:start at 0

time+=cut2("0"+date.getDate())+" "; //add Day

time+=cut2("0"+date.getHours())+":"; //add Hour

time+=cut2("0"+date.getMinutes())+"\n"; //add Minitue

return time;

}

var server = net.createServer(function(socket) { //'connection' listener

socket.write(currentTime());

socket.end(); //end the call

});

server.listen(gport);

**Learnyounode solution**

var net = require('net')

function zeroFill (i) {

return (i < 10 ? '0' : '') + i

}

function now () {

var d = new Date()

return d.getFullYear() + '-' +

zeroFill(d.getMonth() + 1) + '-' +

zeroFill(d.getDate()) + ' ' +

zeroFill(d.getHours()) + ':' +

zeroFill(d.getMinutes())

}

var server = net.createServer(function (socket) {

socket.end(now() + '\n')

})

server.listen(Number(process.argv[2]))

**Lesson 11 –**  ## HTTP FILE SERVER

Write an HTTP server that serves the same text file for each request it receives.

Your server should listen on the port provided by the first argument to your program.

You will be provided with the location of the file to serve as the second command-line argument. You must use the fs.createReadStream() method to stream the file contents to the response.

**Notes**

Where the two arguments are objects representing the HTTP request and the corresponding response for this request. request is used to fetch properties, such as the header and query-string from the request while response is for sending data to the client, both headers and body.

Both request and response are also Node streams! Which means that you can use the streaming abstractions to send and receive data if they suit your use-case.

**Solution 11 –**

**My solution**

var http = require('http');

var fs = require("fs");

var gport = process.argv[2]; //port to use

var filedir = process.argv[3]; // file directory to read

var server = http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/plain'});

var rs=fs.createReadStream(filedir);

rs.on("open", function(){ // test to see if the readable stream is valid

rs.pipe(res);

});

rs.on("error", function(err){ // catches any errors that happens while creating readble stream (usually invalid name)

res.end(err);

});

});

server.listen(gport);

**Learnyounode solution**

var http = require('http')

var fs = require('fs')

var server = http.createServer(function (req, res) {

res.writeHead(200, { 'content-type': 'text/plain' })

fs.createReadStream(process.argv[3]).pipe(res)

})

server.listen(Number(process.argv[2]))

**Lesson 12–**  ## HTTP UPPERCASERER

Write an HTTP server that receives only POST requests and converts incoming POST body characters to upper-case and returns it to the client.

Your server should listen on the port provided by the first argument to your program.

**Solution 12 –**

**My solution**

var http = require('http');

var map = require("through2-map");

var gport = process.argv[2]; //port to use

var server = http.createServer(function (req, res) {

req.pipe(map(function (chunk){

return chunk.toString().toUpperCase();

})).pipe(res);

});

server.listen(gport);

**Learnyounode solution (better)**

var http = require('http')

var map = require('through2-map')

var server = http.createServer(function (req, res) {

if (req.method !== 'POST') {

return res.end('send me a POST\n')

}

req.pipe(map(function (chunk) {

return chunk.toString().toUpperCase()

})).pipe(res)

})

server.listen(Number(process.argv[2]))

**Lesson 13 –**  ## HTTP JSON API SERVER

Write an HTTP server that serves JSON data when it receives a GET request to the path '/api/parsetime'. Expect the request to contain a query string with a key 'iso' and an ISO-format time as the value.

For example:

/api/parsetime?iso=2013-08-10T12:10:15.474Z

The JSON response should contain only 'hour', 'minute' and 'second' properties. For example:

{

"hour": 14,

"minute": 23,

"second": 15

}

Add second endpoint for the path '/api/unixtime' which accepts the same query string but returns UNIX epoch time in milliseconds (the number of milliseconds since 1 Jan 1970 00:00:00 UTC) under the property 'unixtime'.

For example:

{ "unixtime": 1376136615474 }

Your server should listen on the port provided by the first argument to your program.

**Solution 13 –**

**My solution**

var http = require('http');

var url = require('url');

var gport = process.argv[2]; //port to use

function printHMS(date){

var dictP ={};

dictP["hour"]=date.getHours();

dictP["minute"]=date.getMinutes();

dictP["second"]=date.getSeconds();

return dictP;

}

function printUnix(date){

var dictP ={};

dictP["unixtime"]=date.getTime();

return dictP;

}

var server = http.createServer(function (req, res) {

var urlParse=url.parse(req.url, true);

var queryinfo = urlParse.query.iso; //locate where the time is

var date = new Date(queryinfo);

var reParse =/\/api\/parsetime/; //to match the path

var reUnixtime =/\/api\/unixtime/; //to match the path

var result;

res.writeHead(200, { 'Content-Type': 'application/json' })

if(reParse.test(req.url)){ //to test the path if true or false

result = JSON.stringify(printHMS(date)); // convert result to json string

res.write(result);

}

if(reUnixtime.test(req.url)){ //to test the path if true or false

result=JSON.stringify(printUnix(date)); // convert result to json string

res.write(result);

}

res.end();

});

server.listen(gport);

**Learnyounode solution (better)**

var http = require('http')

var url = require('url')

function parsetime (time) {

return {

hour: time.getHours(),

minute: time.getMinutes(),

second: time.getSeconds()

}

}

function unixtime (time) {

return { unixtime: time.getTime() }

}

var server = http.createServer(function (req, res) {

var parsedUrl = url.parse(req.url, true)

var time = new Date(parsedUrl.query.iso)

var result

if (/^\/api\/parsetime/.test(req.url)) {

result = parsetime(time)

} else if (/^\/api\/unixtime/.test(req.url)) {

result = unixtime(time)

}

if (result) {

res.writeHead(200, { 'Content-Type': 'application/json' })

res.end(JSON.stringify(result))

} else {

res.writeHead(404)

res.end()

}

})

server.listen(Number(process.argv[2]))