**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 -**