C1- Code & Go

C-WEB-150

Node

Day 01

CODING ACADEMY
BY EPITECH

Foreword

This day will be spent learning about Node.

Let me introduce your new friend ===> https://nodejs.org/dist/latest-v6.x/docs/api/

As you will notice, there is plenty to read and learn, it can be hard at times. Getting a good understanding of Node will be a good thing for your career as a developer. Node is a powerful tool.

Please read thoroughly each exercise, there is everything you need to understand the expected console outputs

Ask yourself the right questions and, most importantly, think.

You will create a repository named **MEAN_Pool_Day_01/**

Each Exercise will have its own directory.
E.g.: Exercise 01 sources will be in ex_01
Today, the exercises follow each other and are not independent. They are to be done in order.

CODING ACADEMY
BY EPITECH

Exercise 01

1 Pt

File to hand in: ex_01.js
Restriction: Use of any external module is forbidden

Your program should retrieve command line arguments and display on standard output those that can be converted into a number, followed by a new line.

Example:

```
Terminal
~/C-WEB-150> node ex_01.js toto 5 martin 0 | cat -e
5$
0$
~/C-WEB-150>
```


CODING ACADEMY
BY EPITECH

Exercise 02

1 Pt

File to hand in: ex_02.js
Restriction: Use Node synchronous functions (functions ending by *Sync())

For this exercise, you will use synchronous functions of Node's FS (File System) module.

Your script named ex_02.js takes a number as argument. Upon execution, your script will create a **garbage** directory. This directory must be filled with a number of empty files equal to the number passed as argument.

The files name will be numbered from 1 to the entered number. Each time a file is created, you must display "Created file x", x being the name of the created file.

If the garbage directory already exists, it must be emptied beforehand (without deleting it).
The last line of your script MUST BE **console.log("Done")**.

In case an error occurs, you will display "Error: Critical failure" ("Done" will still be displayed).

Example 1:

```
Terminal
~/C-WEB-150> node ex_02.js
Done
~/C-WEB-150> ls -R
.:
ex_02.js garbage
./garbage:
~/C-WEB-150>
```


CODING ACADEMY
BY EPITECH

Example 2:

```
Terminal
~/C-WEB-150> node ex_02.js 5
Created file 1
Created file 2
Created file 3
Created file 4
Created file 5
Done
~/C-WEB-150> ls -R
.:
ex_02.js garbage ./garbage
1 2 3 4 5
```


CODING ACADEMY
BY EPITECH

Exercise 03

1 Pt

From now on, no more synchronous functions. Node strength lies in asynchronous functions. Let's do things in the Node way!!!!!

File to hand in: ex_03.js
Restriction: Use asynchronous functions of the FS module

Your script will take two file names as argument. Upon execution, the first file content will be copied into the second one.

If the second file already exists, its content must be replaced.
If it does not, it must be created and filled.

If the first file does not exist, it does not have to be created.

If an error happens, you must display "Error:_Critical_failure."

Exemple:

```
Terminal
~/C-WEB-150> cat toto
Salutation!  Je suis Toto.
~/C-WEB-150> node ex_03.js toto toto_copy
~/C-WEB-150> cat toto_copy
Salutation!  Je suis Toto.
~/C-WEB-150>
```


CODING ACADEMY
BY EPITECH

Exercise 04

1 Pt

Filtered listing

File to hand in: ex_04.js
Restriction: Use asynchronous functions of the FS module

Your script will be provided a directory name as the first argument and a file extension as the second argument. Your script will print the list of files matching this extension within the given directory.

The list of files should be printed on the console, one file per line.

If an error happens, you must display "Error: Critical failure".

Warning: the second argument will not come prefixed with ".".
Example:

Terminal

```
~/C-WEB-150> ls home/path/some_dir
foo.js bar.js gecko.md think.md understood.txt ~/C-WEB-150> node ex_04.js
home/path/some_dir md
gecko.md
think.md
~/C-WEB-150> node ex_04.js home/path/some_dir txt
understood.txt
```


CODING ACADEMY
BY EPITECH

Exercise 05

3 Pts

It is time to retrace your steps, and make amends!!!

File to hand in: ex_05.js
Restriction: Use asynchronous functions of the FS module

Go back to exercise 02 but only use asynchronous functions this time.

The results will have to be identical to exercise 02, except for the "Done" that will be displayed at the beginning instead of the end.

(Of course, you will not move **console.log("Done")** which will remain the last line of your script.)

CODING ACADEMY
BY EPITECH

Exercise 06

1 Pt

Modularity

File to hand in: ex_06.js, check_type.js
Restriction: Use the "type-of-is" module

As you may have noticed in the previous exercises, callbacks can quickly make your code difficult to read and to review.

This is why it can be interesting to organize your code into modules in order to encapsulate functionalities.

In this exercise, you will need to create a module named **check_types**, which will contain a **check** function. That function takes an array of values as parameter and returns an array containing the result of the string method of the **type-of-is** module for each passed value.

Please note that you do not have the **type-of-is** module on your VM.
It's up to you to install it. (**Tip:** "npm").

Example:

```
Terminal
~/C-WEB-150> cat ex_06.js
var ct = ...  //import your check_types module
console.log(ct.check(["chat", [0, 0], 5]))
~/C-WEB-150> node ex_06.js
['String', 'Array', 'Number' ]
~/C-WEB-150>
```


CODING ACADEMY
BY EPITECH

Exercise 07

2 Pts

Behold! Here starts your love story with Node!

File to hand in: ex_07.js, server.js
Restriction: Use HTTP module (asynchronous)

You will create a server module which will have a start method taking a listening port as parameter.
Your ex_07.js will start a server with an appropriate port passed as argument.

Your server will have a very simple behavior. Each time a client connects, your server will send him the following content:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF8"/>
    <title>Welcome</title>
  </head>
  <body>
    <p>Greetings Traveler!</p>
  </body>
</html>
```


CODING ACADEMY
BY EPITECH

Exercise 08

2 Pts

As you probably guessed, writing your html in your scripts can quickly become cumbersome to maintain.

File to hand in: ex_08.js, server.js, index.html
Restriction: use HTTP, URL, FS (File System) modules (asynchronous)

First step, copy server.js from ex_07 and move your html code to a file named index.html.

Second step, use Node's File System to retrieve index.html content. This content will be served when needed. (pun intended ;])

Make sure to greet your user properly by obtaining the "name" parameter passed through the URL (GET).

Example: a call to page **localhost:port?name=Martin** will display "Greetings_Martin!" to the user.

If there is no **name** parameter obtained, the greeting will remain "Greetings_Traveler!"

Tip: Think about how you are going to replace your name variable in the HTML page that you will get with File System. It is simply a matter of text manipulation.

CODING ACADEMY
BY EPITECH

Exercise 09

3 Pts

Hopefully, your server is cleaner. Let's add more web pages! As this won't happen overnight, you will need to get your hands dirty.

File to hand in: ex_09.js, server.js, router.js, page.js, index.html, image.html
Restriction: use HTTP, URL, FS (File System) modules (asynchronous)

There is an ugly way to add more pages using multiples if and else, I said ugly! Don't even think about it! We want you to aim for cleanliness. Other developers will thank you for it, you will even thank yourself.

Part 1

Copy your server.js and index.html from the previous exercise.

You will provide a way for your server to differentiate between the different URLs that it receives requests for. Your server must be able to generate custom responses for each of these specific URLs.

To achieve this, you are going to implement a router. Create a module named router.js that will contain the method **route**. This function will take three parameters: an associative array **handle**, a **request** object and a **response** object.

Your router will check if the URL is found in the indexes of the associative array, and if the corresponding value is indeed a function. If the function is found, it will be called with a **request** object and a **response** object.

If no corresponding index is to be found, your router will reply with an error 404 in plain text simply stating: "404 error: Page not found."

CODING ACADEMY
BY EPITECH

Part 2

You will now implement the custom responses by creating a **pages** module that will contain two methods. Each method will manage to display a specific page by receiving as arguments a **request** object and a **response** object.

- **Index** method serves the content of index.html with the "name" passed through the URL as required in the preceding exercise.
- **Image** method serves the content of image.html. (provided in the subject)

Of course in each case the header sent by your server must have a value of 200 and a Content-Type indicated as "text/html".

Lastly, for everything to come into shape, you will modify the **start** method of your server so that it takes three parameters, a **port**, your **route** method and an associative array **handle**. Upon a new connection, your server will call **route** with the right arguments.

In your ex_09.js you will start your server by calling its rightfully named method. It's the only function call (of your own functions) allowed in that file.

Reminder: The array will contain paths as key and your **pages** methods as value. This will be instrumental in handling routing neatly. Obviously this array will be declared only in one place.

CODING ACADEMY
BY EPITECH

Exercise 10

3 Pts

Just a tiny bit more complexity, is it a GET or a POST?

File to hand in: ex_10.js, server.js, router.js, page.js, index.html, image.html
Restriction: use HTTP, URL, FS (File System) modules (asynchronous)

Now that you can add pages to your server in an appropriate way, we want you to handle POST requests.

First step, copy your server.js, router.js, pages.js index.html and image.html.

Second step, create a little form named form.html sending to **/index.html** with an "input" field named "name" and a "Submit" button.

Let's get some work done!
In order to display your form on **/form.html** you will add a method **form** to your **pages** module. This method sole purpose is to serve the desired page.

Update your associative array **handle** by adding the path as key and this method as value.

To check everything works as expected, try the new added path. You should see your form. If you hit the submit button, you are redirected to the index page. Yay! Yay! Look at you all grown up.

Let's not get ahead of ourselves, the content of your form is not handled yet, so there is still some work to do.

You will modify the **index** method of your **pages** module so that when it receives a POST request with a field "name", your "index" page will have to use it instead of GET in order to greet this visitor.

Keep in mind that your **index** method must be able to handle both behaviors.

In your ex_10.js you will start your server by calling its rightfully named method. It's the only function call (of your own functions) allowed in that file.

CODING ACADEMY
BY EPITECH

Exercise 11

2 Pts

LARGEDive in and meet node Raw TCP/IP socket interface

File to hand in: srvr.js, server.js, clt.js, client.js
Restriction: use NET module (asynchronous)

You will create a simple TCP server/client and enjoy a great load of fun.

Part 1
Create a **server** module containing a method called **run** that takes a **port** as parameter.
A call to your **run** method will create a TCP server listening on the port received as argument.

Each time a client connects, the server will write "Client connected" on the standard output and will, of course, write "Client disconnected" upon disconnection.

Since you are polite, your server should greet each new client by sending "Greetings traveler!" upon connection.

Last but not least, each time a client sends a "ping" to the server, the server will answer "pong".
If a different message is sent, the server will answer, "Sorry, I don't know. :(".

Part 2
Create a **client** module containing a method called **connect**, taking a **host** and a **port** as parameter.

As soon as the **connect** method is called, your client will connect on the given address and port, and send a message containing "ping" before disconnecting.

Any answers from the server will be displayed on standard output followed by a new line.

- Srv.js will launch a server instance on the 1337 port
- Clt.js will launch a client instance connecting to your localhost on the 1337 port

CODING ACADEMY
BY EPITECH