ESMAD | TSIW | PWII
Exercise Sheet nº1 – Introduction to Node.js

Parameters and Environment variables

1. Create a `01_sum.js` script that accepts two numbers passed as command line arguments and displays the sum.

```
CALL: node 01_sum.js
          OUTPUT: USAGE: node 01_sum.js <number1> <number2>
CALL: node 01_sum.js  hello 1
          OUTPUT: ERROR: One or both arguments are not valid numbers
CALL: node 01_sum.js 1 2
          OUTPUT: The sum of 1 and 2 is 3.
```

2. Create a `02_welcome.js` script that reads an environment variable called USER_NAME and displays a custom message.
   Steps:
   a) Set the environment variable before running the script:
   ```
   SET USER_NAME=Teresa && node 02_welcome
   ```
   b) The script should display the following message:
   ```
   Hello, Teresa! Welcome to Node!
   ```
   c) If the USER_NAME variable is not set, the script should display:
   ```
   Hello, visitor! Set the USER_NAME variable to a personalized greeting.
   ```

Node modules

3. Create a custom module (`03_message.js`) that exports the function `getMessage(username)` to display the same custom message as the previous exercise. Use the command-line arguments to define the name of the user and the colour name. Use the cli-color module to colour the output (if no colour is provided, or it is an invalid colour name, it defaults to a predefined colour). Colour name supported by cli-color are: black, red, green, yellow, blue, magenta, cyan and white.

4. Implement a Node module to export colour codes and corresponding HTML name. In your module, add the following array of objects with some colours:

```
const allColours = [
    { name: 'brightred', code: '#E74C3C' },
    { name: 'soothingpurple', code: '#9B59B6' },
    { name: 'skyblue', code: '#5DADE2' },
```

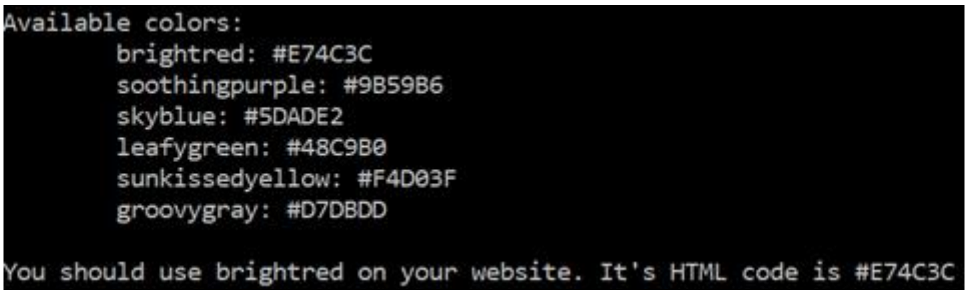
```
    { name: 'leafygreen', code: '#48C9B0' },
    { name: 'sunkissedyellow', code: '#F4D03F' },
    { name: 'groovygray', code: '#D7DBDD' }
];
```

Complete the module, by exporting the following functionalities:

1) **getRandomColour**: randomly exports one colour from the array

3) **getAllColours**: exports all colours

Import the created module into a Node application so that the following information is displayed into the command prompt: all available colours and one random colour for a website.



```
Available colors:
        brightred: #E74C3C
        soothingpurple: #9B59B6
        skyblue: #5DADE2
        leafygreen: #48C9B0
        sunkissedyellow: #F4D03F
        groovygray: #D7DBDD

You should use brightred on your website. It's HTML code is #E74C3C
```

5. Implement a Node module to export the 4 basic mathematical operations between 2 numbers: addition, subtraction, multiplication, and division. The module must provide an error message if the divider of a division operation is zero.

   Create a Node application that reads the command-line arguments with the format **`operation number1 number2`** and outputs the result of the operation:

```
> node ex2 + 1 2
1 + 2 = 3
> node ex2 - 1 2
1 - 2 = -1
> node ex2 . 1 2
Sorry, could not recognize the operation
> node ex2 + 1
Sorry, must provide three arguments
```

6. Create a Node.js script that:
   a. Read a `data.txt` file (create one: `echo DATA > data.txt`).
   b. Add a new line with the current date and time (use `new Date().toLocaleString()`).

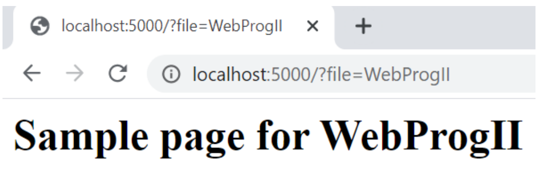
7. Write a script that uses the os module to print information about the operating system. Create a file and write the operating system information.

8. Create an HTTP server with Node.js that recognizes three routes:
   a. GET / → Responds with "Welcome to my server!".
   b. GET /about → Responds with "This is a Node.js server".
   c. GET /time → Responds with the current time.

   For all other requests, the server response with “Route not found”.

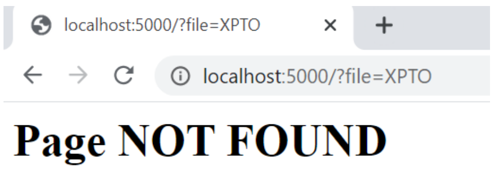
9. Create a Node.js HTTP server that:
    a. Listens on port 3000.
    b. Reads a filename from the URL (e.g., http://localhost:3000/file.txt).
    c. If the file exists inside the `public/` directory, serves the file. Otherwise, returns a 404 error and lists the available files inside the `public/` directory.

10. Build a web server accessed through port 5000. It must serve static HTML pages by obtaining the name of the HTML file in the query string of the HTTP request. If the request corresponds to a non-existent file, the server must return a `NOT_FOUND.html` page, with error code 404. Add some sample HTML files into your application directory (including one `NOT_FOUND.html` page) and test it.


localhost:5000/?file=WebProgII

localhost:5000/?file=WebProgII

Sample page for WebProgII


localhost:5000/?file=XPTO

localhost:5000/?file=XPTO

Page NOT FOUND

11. Create a Node.js server that serves a simple HTML page when receiving a GET request. This page must contain a form with an input field and a button to enter a message and send the message via POST request to /message:
`<form action='/message' method='POST'><input type='text' name='message' required><button type='submit'>Send</button>`
The server must:
    a. Handle the POST request at /message by receiving the submitted message and appending it to a `messages.txt` file.
    b. After storing the message, respond with status code of 302 (Redirection) and set header “Location” to value (“/”) (this will redirect the client to the server’s homepage).