Node.js Week 2

fs and process

process

- It is a global object which provides information about, and control over, the current Node.js process.
- It is always available to Node.js applications without using require().
- To view the process object run:

```
$ node
> process
```

process.argv (1)

- It is a property of the process object which contains an array.
- The array contains the command line arguments passed when the Node.js process was launched.

process.argv (2)

- By default process.argv contains an array with two elements:
  - Process.arg[0]: contains the process.execPath property
    - process.execPath returns the absolute pathname of the executable that started the Node.js process.
  - Process.arg[1]: contains the PATH to node.js executed file
- We can view the console.log(process.argv);
  - If we call it inside node environment (REPL), the array contains only the process.execPath property

process.argv (3)

- We can use it to find out what arguments are being passed into our program, if a certain argument is supplied.
- So a node.js program is able to do something that it wouldn't do if there weren’t any supplied arguments.
- By typing $ node <filename>.js <arg1, arg2,...>, we can pass whatever arguments we want

*(1,2 + pass args display)

process.argv (4)

Example:

- Command: node index.js one two three
- Returns: [‘< process.execPath>’,

'</[PATH TO]/index.js >,

'one',

'two',

'three']

process.argv (5)

- We can also pass files and folders of directories inside the process.argv array.
- $ node . * : adds all files of the current folder to the array
- $ node . ../*: adds all files of the previous folder to the array
- $ node . ~: adds the root directory path to the array

. indicates the current directory

* indicates all (folders or files)

indicates the home directory

process.argv (6)

- Using single quotes:
  - We can prevent the functionality presented at the previous slide
    - $ node . ‘../*’
  - We can write a string and pass it as a single argument
    - $ node . ‘one two three’

What is a CLI

- CLI : Command Line Interface
- Is a means for the user of interacting with a computer program by issuing commands in the form of lines of text.
- It is handled by a program called command language interpreter (shell).
- Example:
  - Bash shell: The shell of MacOS and linux, which has a CLI to issue commands (ls, cd, mkdir etc.)

Node CLI

- We can view its commands executing : $ node --help
- Process.argv can be used to create new command line interfaces inside node

*e.g. (3,4)

fs

- Fs is an npm library that allow us to work with the file system of a computer
- To use it we should import it (it is already inside node) to our node.js project:

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

fs basic methods (1)

- **fs.readFile()**: Reads a file asynchronously
- **fs.readFileSync()**: Read file synchronously (blocks the execution of the code until it finishes)
- **fs.writeFile()**: Writes to a file asynchronously (replacing the file if it already exists)
- **fs.writeFileSync()**: Writes to a file synchronously (blocks the execution of the code until it finishes)

fs basic methods (2)

- **fs.appendFile()** : Asynchronously append data to a file, creating the file if it does not yet exist
- **fs.appendFileSync()**: Synchronously append data to a file, creating the file if it does not yet exist

Append preserves the content of the file.

https://nodejs.org/api/fs.html

ReadFile methods

- fs.readFile(path[, options], callback)
  - Path: the path of the file
  - Options: Usually the encoding of the file
  - Callback: The args passed are the error, and the data, where data is the contents of the file.
- fs.readFileSync(path[, options])
  - Path: the path of the file
  - Options: Usually the encoding of the file

writeFile methods

- fs.writeFile(file, data[, options], callback)
  - Path: the path of the file
  - Data: the Data to be written (usually string)
  - Options: Usually the encoding of the file
  - Callback: The args passed is the error.
- fs.writeFileSync(file, data[, options])
  - Path: the path of the file
  - Options: Usually the encoding of the file

AppendFile methods

- fs.appendFile(file, data[, options], callback)
  - Path: the path of the file
  - Data: the Data to be written (usually string)
  - Options: Usually the encoding of the file
  - Callback: The args passed is the error.
- fs.appendFileSync(file, data[, options])
  - Path: the path of the file
  - Options: Usually the encoding of the file

Use Sync or Async ?

- Question: “should code run in the background while I’m reading this file?” If yes, use async. Otherwise, use sync.
- When coding for a webserver use Async (non-blocking)
- Prefer Sync when it’s an option, it is faster and simper

*e.g. 5

https://medium.com/@adamhooper/node-synchronous-code-runs-faster-than-asynchronous-code-b0553d5cf54e

CRUD (1)

- CRUD = Create, Read, Update and Delete
- The term is most commonly used at:
  - Databases
  - User Interfaces (like CLI)
- At CLIs it is used in order for the user to:
  - Create or add new entries
  - Read existing entries
  - Update or edit existing entries
  - Delete existing entries

  e.g. todo list

CRUD (2)

- Let’s built a simple CRUD CLI Todo app !!!

****Bonus assignment**:

Add functionality to edit (add and clear the usage commands).

Commands: add-usage, clear-usage

assignment

- https://github.com/SocialHackersCodeSchool/Node.js/blob/master/week2/homework/README.md
- Hints for remove implementation:
  - Think to first read the todo file, then remove the todo user picked from the list array and then display the list.
  - Think of using splice and join at the list array
  - Consider of using parseInt() aswell

Sources

- https://developer.mozilla.org
- https://nodejs.org/
- https://www.w3schools.com/
- https://stackoverflow.com
- https://medium.com
- & more..