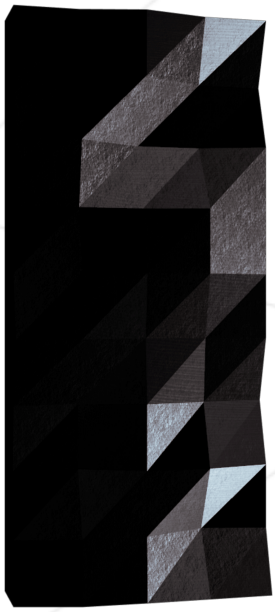

# hello_node

This is a beginning for server side javascript

*Summary: This document is the subject for node.js @42Tokyo.*

# Contents

# Chapter I

# Instructions

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ...`.
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- your node version should be == 12.X.X.
- Your program should not CRASH.
- If you use any external module, then during the evaluation, you should explain why you use that external module and install in front of them.
- You should do this subject on the 42's iMac(Guacamole), use brew to download npm and node.
- Evaluation will be done on 42's iMac (Guacamole).

# Chapter II

# Foreword

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

# Chapter III

# Exercise 00 : Hello World

|  | Exercise 00 |
|---|---|
| Only the best know how to display Hello World | |
| Turn-in directory : $ex00/$ | |
| Files to turn in : `hello-world.js` | |
| Allowed functions : `console.log` | |

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

```
?>node hello-world.js
HELLO WORLD
?>
```

# Chapter IV

# Exercise 01 : Variables

|  | Exercise 01 |
|---|---|
| What kind of variables are there? | |
| Turn-in directory : $ex01/$ | |
| Files to turn in : `vars.js` | |
| Allowed functions : `console.log` | |

- Write a program that prints the same output as below.

```
?>node vars.js
42 is a string.
42 is a number.
42 is an object.
[object Object] is an object.
true is a boolean.
undefined is an undefined.
?>
```

`typeof`

# Chapter V

# Exercise 02 : Args

|  | Exercise 02 |
|---|---|
| What are args anyway ? | |
| Turn-in directory : $ex02/$ | |
| Files to turn in : `sum_args.js` | |
| Allowed functions : `console.log, parseInt` | |

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

```
?>node sum_args.js 1 2 3
6
?>
```

# Chapter VI

# Exercise 03 : I/O

|  | Exercise 03 |
|---|---|
| I/O | |
| Turn-in directory : $ex03/$ | |
| Files to turn in : `io.js` | |
| Allowed functions : `require, fs.readFileSync, console.log, toString` | |

- 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.
- You should use "fs.readFileSync" inside your program.

```
?>node io.js /etc/passwd
108
?>
```

# Chapter VII

# Exercise 04 : Async I/O

|  | Exercise 04 |
|---|---|
| Async I/O | |
| Turn-in directory : $ex04/$ | |
| Files to turn in : `asyncio.js` | |
| Allowed functions : `require, fs.readFile, console.log, toString, split` | |

- 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 (i.e., process.argv[2]). You do not need to make your own test file.
- You should use "fs.readFile" inside your program.

```
?>node asnycio.js /etc/passwd
108
?>
```

`Callback`

# Chapter VIII

# Exercise 05 : HTTP Client

|  | Exercise 05 |
|---|---|
| HTTP Client | |
| Turn-in directory : $ex05/$ | |
| Files to turn in : `http-client.js` | |
| Allowed functions : * | |

- 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).

```
?>node http-client.js http://www.google.com
<!doctype html><html itemscope=""
[...]</body></html>
?>
```

Event

# Chapter IX

# Exercise 06 : HTTP Collect

|  | Exercise 06 |
|---|---|
| HTTP Collect | |
| Turn-in directory : $ex06/$ | |
| Files to turn in : `http-collect.js` | |
| Allowed functions : * | |

- 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.

```
?>node http-collect.js http://www.google.com
50724
<!doctype html><html itemscope=""
[...]</body></html>
?>
```

`npm install bl`

# Chapter X

# Exercise 07 : Async HTTP Collect

|  | Exercise 07 |
|---|---|
| Async HTTP Collect | |
| Turn-in directory : *ex*07/ | |
| Files to turn in : `async-http-collect.js` | |
| Allowed functions : * | |

- 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.

```
?>node async-http-collect.js http://www.google.com http://www.youtube.com http://www.gmail.
    com
<!doctype html><html itemscope=""
[...]
</BODY></HTML>

?>
```

Queue the results

# Chapter XI

# Exercise 08 : Time server

|  | Exercise 08 |
|---|---|
| Time server | |
| Turn-in directory : $ex08/$ | |
| Files to turn in : `time-server.js` | |
| Allowed functions : `*` | |

- 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:

  `"2020-12-15 17:28"`

  After sending the string, close the connection.

```
?>node time-server.js 8080
```

```
?>curl localhost:8080
2020-12-15 17:28
?>
```

`require('net')`

Maybe you can use the previous exercice to get some content from the server..?

# Chapter XII

# Exercise 09 : JSON server

|  | Exercise 09 |
|---|---|
| JSON server | |
| Turn-in directory : $ex09/$ | |
| Files to turn in : `http-json-api-server.js` | |
| Allowed functions : `*` | |

- 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.
- 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.

```
?>node http-json-api-server.js 8080
```

```
?>curl 'localhost:8080/api/parsetime?iso=2020-12-15T17:10:15.474Z'
{"hour":17,"minute":10,"second":15}
?>
```

```
?>curl 'localhost:8080/api/unixtime?iso=2020-12-15T17:10:15.474Z'
{"unixtime":1608052215474}
?>
```

```
require('http')
```

Maybe you can use the previous exercice to get some content from the server..?