Server-Side Web Development - Exercise 01

Student: Markus Ijäs

Help to complete the tasks of this exercise can be found from the chapter 0 ”Setting Up Node.js and the JavaScript Engine” & chapter 2 “Running a Node.js Application” by Jonathan Wexler and from the chapter “About Node.js” of the supplementary course book “Node.js Web Development” by David Herron. The aims of the exercise are to learn to create a Node.js development environment, give you basic understanding of Node.js, and to help you to run your first Node.js application.

Embed your theory answers, drawings, codes, and screenshots directly into this document. Always immediately after the relevant question. Return the document in itsLearning by the deadline.

It’s also recommendable to use Internet sources to supplement the information provided by the course book.

The maximum number of points you can earn from this exercise is 10.

Tasks:

1. Explain with your own words what is Node.js and why it is so popular to learn? (1 point)

Node.js is an asynchronous event-driven JavaScript runtime (a library basically) for building scalable applications. Since Node.js is event-driven with properly implemented architecture it makes it easy to develop efficient multi-user applications without having to worry about threading and it's difficulties.

2. Node.js event loop (4 * 0,5 = 2 points)

Explain with your own words what Node.js’ event loop is.

Node.js is a single-threaded application, which means that by default it's only able to run one thing at a time. Since this is obviously a huge problem, Node.js as an event loop that gives developers an ability to call different Node.js APIs (and their own things as well) asynchronously, allowing things to be run concurrently without, say, a long file operation halting execution of all the other things.

Node.js' event loop is constucted in an Observer pattern. This is basically a pattern where events are listened to by some amount of individual Observers, and whenever an event occurs, all the Observers get notified. This pattern makes communicating between different parts of a program very efficient and fast.

What is executed in the event loop, what is delegated?

Event loop basically:

- Executes timers
- Executes pending callbacks (from the previous cycle)

- Has a preparation/idling stage (to basically prevent hogging cpu, among other things)
- Polls for incoming connections, data etc.
- Check and invokes `setImmediate()` callbacks
- Close callbacks in need of closing

All the actual work is delegated to different callbacks.

What does the event loop do when a delegated task is finished?

Nothing specific. It just handles delegating events, whenever they come.

What does it mean that Node.js is single threaded?

It basically runs on single CPU core, without an ability to do parallel work. (Actually Node.js has a few threads, separated to basically the Event Loop, and some Worker threads actually executing our commands)

3. Explain the types of application where Node.js is efficient. When it might not be efficient at all? (2 * 0,5 = 1 point)

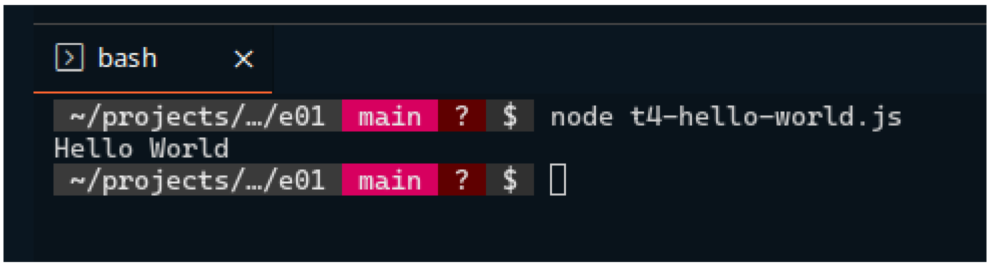
Node.js is basically efficient for things that "break down" into small individual tasks, while inefficient for example in applications requiring lots of complex time-consuming calculations. Basically anything CPU/IO-bound is not well suited for Node.js, those are the things you'd want to delegate to more efficient "data processors".

4. Run your first Node.js application “Hello World!”.

Write the necessary code into a file. Display the greeting on the terminal window. (1 point)

```
console.log("Hello World")
```

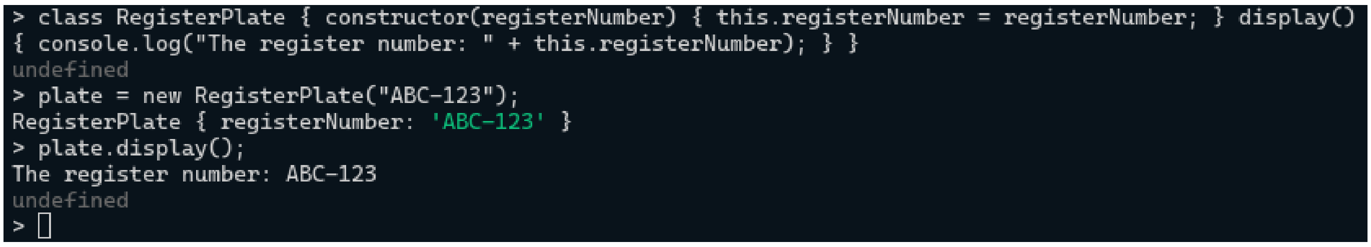
And screenshot of execution:



```
~/projects/…/e01  main  ?  $  node t4-hello-world.js
Hello World
~/projects/…/e01  main  ?  $
```

5. Create a JavaScript class RegisterPlate.

Create a constructor that takes an argument registerNumber. Create a method display that prints the registerNumber at the terminal window. Do all this by using REPL. (1 point)



```
> class RegisterPlate { constructor(registerNumber) { this.registerNumber = registerNumber; } display()
{ console.log("The register number: " + this.registerNumber); } }
undefined
> plate = new RegisterPlate("ABC-123");
RegisterPlate { registerNumber: 'ABC-123' }
> plate.display();
The register number: ABC-123
undefined
> 
```

6. Create an array containing the names of some of your Favorite songs.

The length of the array is six. Create a small Node.js application that randomly selects one of the songs and then displays it at the terminal window. (2 points)

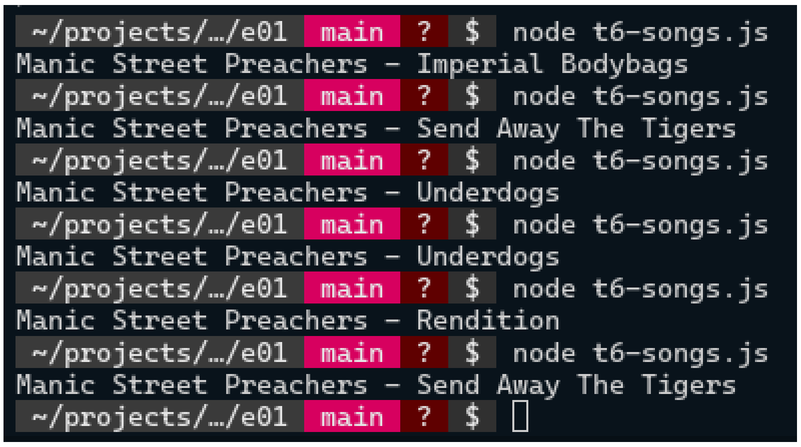
Code:

```
const songs = [
  "Manic Street Preachers - Send Away The Tigers",
  "Manic Street Preachers - Imperial Bodybags",
  "Manic Street Preachers - Autumnsong",
  "Manic Street Preachers - Indian Summer",
  "Manic Street Preachers - Rendition",
  "Manic Street Preachers - Underdogs",
];

function getRandomSong(songs) {
  let randomInt = Math.floor(Math.random() * songs.length);
  return songs[randomInt];
}

console.log(getRandomSong(songs));
```

Screenshot:



```
 ~/projects/…/e01  main  ?  $  node t6-songs.js
Manic Street Preachers - Imperial Bodybags
 ~/projects/…/e01  main  ?  $  node t6-songs.js
Manic Street Preachers - Send Away The Tigers
 ~/projects/…/e01  main  ?  $  node t6-songs.js
Manic Street Preachers - Underdogs
 ~/projects/…/e01  main  ?  $  node t6-songs.js
Manic Street Preachers - Underdogs
 ~/projects/…/e01  main  ?  $  node t6-songs.js
Manic Street Preachers - Rendition
 ~/projects/…/e01  main  ?  $  node t6-songs.js
Manic Street Preachers - Send Away The Tigers
 ~/projects/…/e01  main  ?  $ 
```

7. Asynchronous programming. (4 * 0,5 = 2 points, you can leave one unanswered)

Program the blocking code and non-blocking code examples

that can be found at the address Node.js - Callbacks Concept.

input.txt:

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

main-bloking.js:

```
var fs = require("fs");
var data = fs.readFileSync("input.txt");

console.log(data.toString());
console.log("Program Ended");
```

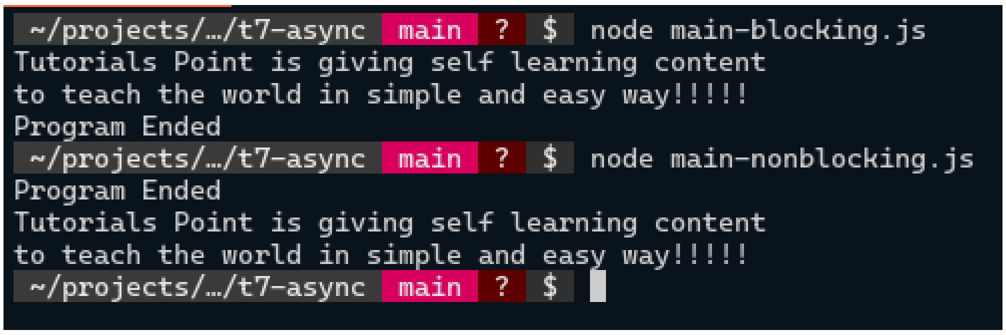
main-nonblocking.js:

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

fs.readFile("input.txt", function (err, data) {
  if (err) return console.error(err);
  console.log(data.toString());
});

console.log("Program Ended");
```

Execute the examples and explain the possible differences in results.



```
 ~/projects/…/t7-async  main  ?  $  node main-blocking.js
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
Program Ended
 ~/projects/…/t7-async  main  ?  $  node main-nonblocking.js
Program Ended
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
 ~/projects/…/t7-async  main  ?  $ 
```

In the blocking example, the reading of the file blocks the execution of the "Program ended" console logging. On the other hand in the non-blocking example, the actual `main-nonblocking.js` finishes execution faster than the reading of the file completes. After the file reading is complete, the callback function given to `fs.readFile` is executed thus logging to console after the `Program ended` console log.

Explain the concept of a callback function.

Basically a function called after some event gets fired, or some action gets completed.

What part of the code is a callback function?

The anonymous function given to `fs.readFile` as the second argument.

How the concept of non-blocking is relevant here?

It allows us to do other work while waiting the completion of file reading operation. We could, for example, show a message that the file is still being read, to the user. We could also initiate some other file read operations, send some data to somewhere, or basically do anything we want while waiting.