JSNAD Certification - Practice Exercises

---

1. Node.js CLI

Exercise 1.1:

Write a command to run `app.js` with the environment variable `MODE=production`.

Exercise 1.2:

Use the Node.js REPL to evaluate the expression `5 + 6 * 3`.

Exercise 1.3:

Use a Node.js CLI flag to allow an experimental module feature.

---

2. Buffers

Exercise 2.1:

Create a Buffer containing the string `"Hello, Node"` and log its hexadecimal representation.

Exercise 2.2:

Allocate a 10-byte buffer and fill it with the value `0xff`.

---

3. Streams

Exercise 3.1:

Create a readable stream from a file called `input.txt` and pipe it to a writable stream `output.txt`.

Exercise 3.2:

Implement a Transform stream that converts all incoming data to uppercase.

---

4. Events

Exercise 4.1:

Create an `EventEmitter` that emits an event called `"start"` with a payload.

Exercise 4.2:

Listen for an `"error"` event and log the error message.

---

5. HTTP(S)

Exercise 5.1:

Create an HTTP server that responds `"Hello World"` to any request.

Exercise 5.2:

Make an HTTPS request to `https://example.com` and log the status code.

---

6. Child Processes

Exercise 6.1:

Use `spawn` to run the `ls` command and log the output.

Exercise 6.2:

Use `exec` to run `node -v` and log the result.

---

7. File System (fs)

Exercise 7.1:

Write a file named `greeting.txt` with the content `"Hello, File System!"`.

Exercise 7.2:

Read the contents of `greeting.txt` asynchronously and log it.

Exercise 7.3:

Check if the file `greeting.txt` exists.

---

8. Modules (CommonJS & ES Modules)

Exercise 8.1:

Export a function from `math.js` and import it into `index.js` using CommonJS.

Exercise 8.2:

Create an ES Module that exports a default class `Car` and import it.

---

9. Timers

Exercise 9.1:

Use `setTimeout` to log `"Done!"` after 3 seconds.

Exercise 9.2:

Use `setInterval` to log `"Running"` every second.

---

10. Process / Environment Variables

Exercise 10.1:

Log all environment variables using Node.js.

Exercise 10.2:

Exit the process with status code `1` if a required environment variable `API_KEY` is missing.

---

11. Worker Threads

Exercise 11.1:

Create a Worker that calculates the factorial of a number.

Exercise 11.2:

Send a message to a Worker and receive a reply.

---

12. Error Handling

Exercise 12.1:

Handle an uncaught exception and log a custom error message.

Exercise 12.2:

Create a Promise that rejects and properly handle the rejection.

Solutions (ONLY LOOK IF YOU NEED)

---

1. Node.js CLI

1.1: `MODE=production node app.js`

1.2: `node -e "console.log(5 + 6 * 3)"`

1.3: `node --experimental-modules app.js`

2. Buffers

2.1:

```
const buf = Buffer.from('Hello, Node');
console.log(buf.toString('hex'));
```

2.2:

```
const buf = Buffer.alloc(10, 0xff);
console.log(buf);
```

3. Streams

3.1:

```
const fs = require('fs');
fs.createReadStream('input.txt').pipe(fs.createWriteStream('output.txt'));
```

3.2:

```
const { Transform } = require('stream');
const upperCase = new Transform({
  transform(chunk, encoding, callback) {
    callback(null, chunk.toString().toUpperCase());
  }
});
```

4. Events

4.1:

```
const EventEmitter = require('events');
const emitter = new EventEmitter();
emitter.on('start', data => console.log(data));
emitter.emit('start', { user: 'Alice' });
```

4.2:

```
emitter.on('error', (err) => console.error(err.message));
```

5. HTTP(S)

5.1:

```
const http = require('http');
http.createServer((req, res) => {
  res.end('Hello World');
}).listen(3000);
```

5.2:

```
const https = require('https');
https.get('https://example.com', (res) => {
  console.log(res.statusCode);
});
```

6. Child Processes

6.1:

```
const { spawn } = require('child_process');
const ls = spawn('ls');
ls.stdout.on('data', data => console.log(data.toString()));
```

6.2:

```
const { exec } = require('child_process');
exec('node -v', (err, stdout) => {
  if (err) throw err;
  console.log(stdout);
});
```

7. File System (fs)

7.1:

```
const fs = require('fs');
fs.writeFile('greeting.txt', 'Hello, File System!', err => {
  if (err) throw err;
});
```

7.2:

```
fs.readFile('greeting.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

7.3:

```
fs.access('greeting.txt', fs.constants.F_OK, (err) => {
  console.log(err ? 'Does not exist' : 'Exists');
});
```

8. Modules

8.1:

// math.js

```
module.exports.add = (a, b) => a + b;
```

// index.js

```
const math = require('./math');
console.log(math.add(2, 3));
```

8.2:

// car.mjs

```
export default class Car {
  constructor(model) {
    this.model = model;
  }
}
```

// index.mjs

```
import Car from './car.mjs';
const car = new Car('Tesla');
console.log(car.model);
```

9. Timers

9.1:

```
setTimeout(() => console.log('Done!'), 3000);
```

9.2:

```
setInterval(() => console.log('Running'), 1000);
```

10. Process / Env Vars

10.1:

```
console.log(process.env);
```

10.2:

```
if (!process.env.API_KEY) {
  console.error('API_KEY missing');
  process.exit(1);
}
```

11. Worker Threads

11.1:

```
const { Worker } = require('worker_threads');

// worker.js
const { parentPort } = require('worker_threads');
parentPort.on('message', num => {
  const factorial = n => (n <= 1 ? 1 : n * factorial(n - 1));
  parentPort.postMessage(factorial(num));
});
```

11.2:

```
const worker = new Worker('./worker.js');
worker.postMessage(5);
worker.on('message', msg => console.log('Factorial:', msg));
```

12. Error Handling

12.1:

```
process.on('uncaughtException', err => {
  console.error('Uncaught Exception:', err.message);
  process.exit(1);
});
```

12.2:

```
Promise.reject(new Error('Failure')).catch(err => console.error(err.message));
```