Node.js Cheat Sheet

1. Introduction to Node.js

- **What is Node.js?**

- A runtime environment that allows executing JavaScript on the server side.
- Built on Chrome's V8 JavaScript engine.
- Non-blocking, event-driven I/O model.

- **Installation**

  - **Windows/Mac:** Use the official installer from nodejs.org.

  - **Linux:** Use package managers like `apt`, `yum`, or `nvm` (Node Version Manager).

- **Version Management**

  - **nvm:** `nvm install <version>`, `nvm use <version>`, `nvm ls`

---

2. Core Concepts

Event Loop

- **Concept:**

- Handles asynchronous operations efficiently.
- Executes callbacks in phases: timers, I/O callbacks, idle/prepare, poll, check, close callbacks.

- **Phases:**

  - **Timers:** Executes callbacks scheduled by `setTimeout` and `setInterval`.

  - **I/O Callbacks:** Handles I/O operations like file system and network requests.

  - **Poll:** Retrieves new I/O events.

  - **Check:** Executes `setImmediate` callbacks.

  - **Close Callbacks:** Handles close events.

Modules

- **CommonJS Modules:**

- `require()` to import.

- `module.exports` or `exports` to export.

```
// math.js
module.exports = {
  add: (a, b) => a + b,
};

// app.js
const math = require('./math');
console.log(math.add(2, 3)); // 5
```

- **ES Modules (ESM):**

- `import` and `export` syntax.
- Use `.mjs` extension or set `"type": "module"` in `package.json`.

```
// math.mjs
export const add = (a, b) => a + b;

// app.mjs
import { add } from './math.mjs';
console.log(add(2, 3)); // 5
```

Global Objects

- **`__dirname` and `__filename`:**

- `__dirname`: Current directory path.
- `__filename`: Current file path.

- **`process`:**

- Provides information about the current Node.js process.
- `process.env`: Environment variables.
- `process.argv`: Command-line arguments.

- **`console`:**

- `console.log()`, `console.error()`, `console.warn()`

---

3. Core Modules

`http`

- **Create a Server:**

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, World!\n');
});

server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

`fs`

- **Read/Write Files:**

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

// Synchronous
const data = fs.readFileSync('file.txt', 'utf8');
fs.writeFileSync('output.txt', data);

// Asynchronous
fs.readFile('file.txt', 'utf8', (err, data) => {
  if (err) throw err;
  fs.writeFile('output.txt', data, (err) => {
    if (err) throw err;
  });
});
```

`path`

- **Path Manipulation:**

```
const path = require('path');

const filePath = path.join(__dirname, 'folder', 'file.txt');
const ext = path.extname(filePath); // .txt
const base = path.basename(filePath); // file.txt
```

`os`

- **OS Information:**

```
const os = require('os');

console.log(os.platform()); // e.g., 'linux', 'darwin'
```

```
console.log(os.arch()); // e.g., 'x64'
console.log(os.cpus()); // Array of CPU cores
```

`events`

- **EventEmitter:**

```
const EventEmitter = require('events');
const myEmitter = new EventEmitter();

myEmitter.on('event', () => {
  console.log('an event occurred!');
});

myEmitter.emit('event');
```

---

4. Commonly Used Libraries

`express`

- **Basic Setup:**

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello, World!');
});

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

`mongoose`

- **MongoDB ORM:**

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/test', { useNewUrlParser:
true });

const Cat = mongoose.model('Cat', { name: String });

const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

`socket.io`

- **Real-time Communication:**

```
const http = require('http').createServer();
const io = require('socket.io')(http);

io.on('connection', (socket) => {
  console.log('a user connected');
  socket.on('chat message', (msg) => {
    io.emit('chat message', msg);
  });
});

http.listen(3000, () => {
  console.log('listening on *:3000');
});
```

`axios`

- **HTTP Client:**

```
const axios = require('axios');

axios.get('https://api.example.com/data')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error(error);
  });
```

---

5. Asynchronous Programming

Callbacks

- **Basic Callback:**

```
function fetchData(callback) {
  setTimeout(() => {
    callback('Data fetched');
  }, 1000);
}
```

```
fetchData((data) => {
  console.log(data);
});
```

Promises

- **Basic Promise:**

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Data fetched');
    }, 1000);
  });
}

fetchData().then(data => {
  console.log(data);
}).catch(error => {
  console.error(error);
});
```

Async/Await

- **Async Function:**

```
async function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Data fetched');
    }, 1000);
  });
}

async function main() {
  try {
    const data = await fetchData();
    console.log(data);
  } catch (error) {
    console.error(error);
  }
}

main();
```

---

6. Error Handling

Try/Catch

- **Synchronous Error Handling:**

```
try {
  throw new Error('Something went wrong');
} catch (error) {
  console.error(error.message);
}
```

Error-First Callbacks

- **Asynchronous Error Handling:**

```
function fetchData(callback) {
  setTimeout(() => {
    callback(new Error('Something went wrong'), null);
  }, 1000);
}

fetchData((error, data) => {
  if (error) {
    console.error(error.message);
  } else {
    console.log(data);
  }
});
```

Promise Rejections

- **Promise Error Handling:**

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      reject(new Error('Something went wrong'));
    }, 1000);
  });
}

fetchData().catch(error => {
  console.error(error.message);
});
```

---

7. Debugging and Profiling

Node.js Debugger

- **Basic Debugging:**

```
node inspect myscript.js
```

- **Using Chrome DevTools:**

```
node --inspect-brk myscript.js
```

Profiling with `v8`

- **Heap Profiling:**

```
node --prof myscript.js
node --prof-process isolate-0x102345678-v8.log > processed.txt
```

---

8. Best Practices

Code Organization

- **Modularize Code:**

- Split code into smaller, reusable modules.

- **Use Middleware:**

- In Express, use middleware for common tasks like logging, authentication.

Security

- **Sanitize Input:**

- Validate and sanitize user inputs to prevent injection attacks.

- **Use HTTPS:**

- Always use HTTPS for secure communication.

Performance Optimization

- **Caching:**

- Use caching mechanisms like Redis or in-memory caches.

- **Load Balancing:**

- Distribute load across multiple instances using load balancers.

---

9. Common CLI Commands

- **Node.js Version:**

```
node -v
```

- **Run a Script:**

```
node myscript.js
```

- **Install Dependencies:**

```
npm install
```

- **Start a Server:**

```
npm start
```

---

10. Resources and Further Reading

- **Official Documentation:** [Node.js Documentation](https://nodejs.org/en/docs/)

- **Books:**

- "Node.js Design Patterns" by Mario Casciaro
- "Express in Action" by Evan Hahn

- **Online Courses:**

- [Node.js: The Complete Guide](https://www.udemy.com/course/nodejs-the-complete-guide/)
- [Node.js, Express, MongoDB & More: The Complete Bootcamp](https://www.udemy.com/course/nodejs-express-mongodb-bootcamp/)

This cheat sheet provides a comprehensive overview of Node.js, covering core concepts, modules, libraries, asynchronous programming, error handling, debugging, best practices, and common CLI commands. Use this as a quick reference to enhance your Node.js development skills.

By Ahmed Baheeg Khorshid

ver 1.0