Node.js Cheat Sheet in Simple Terms

Node.js Intro

Purpose: Node.js is a powerful JavaScript runtime built on Chrome's V8 JavaScript engine. It's designed to build scalable network applications efficiently. Node.js uses non-blocking, event-driven architecture, enabling it to handle many connections simultaneously without straining the server. This makes it ideal for developing server-side applications, APIs, and real-time web services.

Key Features:

- **Minimalist**: Express is lightweight and doesn't dictate how you should structure your application.
- **Flexible**: It has a robust set of features that can be extended with additional modules.
- **Robust Routing:** Allows you to manage different HTTP routes via a simple and intuitive API.

Node.js Get Started

Purpose: Step-by-step guide to setting up a Node.js environment and creating your first application.

Example Usage: Install Node.js, set up a simple server using HTTP module, and run your first script.

Node.js Modules

Purpose: Modules are reusable blocks of code whose existence does not accidentally impact.
Example Usage: Use require() to include modules like http, fs in your Node.js scripts.

Node.js HTTP Module

Purpose: Allows you to create web servers that listen for HTTP requests from clients.
Example Usage:

```
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);
```

Node.js File System

Purpose: Provides a way to perform file operations like read, write, update, and delete.
Example Usage:

```
const fs = require('fs');
fs.readFile('input.txt', function (err, data) {
  if (err) throw err;
  console.log(data.toString());
});
```

Node.js URL Module

Purpose: Parses URL strings and returns an object with accessible properties to the parts of the URL.
Example Usage:

```
const url = require('url');
const myUrl = new URL('http://example.com?name=abc&status=active');
console.log(myUrl.hostname); // 'example.com'
```

Node.js NPM

Purpose: Node Package Manager for sharing and borrowing packages, and also managing dependencies in your projects.
Example Usage:

```
Install packages using npm install <package_name>, and manage them
with a package.json file.
```

Node.js Events

Purpose: Allows you to create, fire, and listen for your own events.
Example Usage:

```
const events = require('events');
const eventEmitter = new events.EventEmitter();
eventEmitter.on('myEvent', function () {
  console.log('Event Fired!');
});
eventEmitter.emit('myEvent');
```

Node.js Upload Files

Purpose: Handling file uploads from the client-side.
Example Usage:

```
const formidable = require('formidable');
const http = require('http');
const form = new formidable.IncomingForm();

http.createServer((req, res) => {
  form.parse(req, (err, fields, files) => {
    res.write('File uploaded');
    res.end();
  });
}).listen(3000);
```

Node.js Email

Purpose: Sending emails using Node.js via SMTP protocol.
Example Usage :

```
const nodemailer = require('nodemailer');
let transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'your-email@gmail.com',
    pass: 'your-password'
  }
});
 let mailOptions = {
  from: 'your-email@gmail.com',
  to: 'to-email@gmail.com',
  subject: 'Sending Email using Node.js',
  text: 'That was easy!'
};
 transporter.sendMail(mailOptions, function(error, info){
  if (error) {
    console.log(error);
  } else {
    console.log('Email sent: ' + info.response);
  }
});
```

Built-in Modules

Purpose: Node.js comes with a variety of built-in modules that require no additional installation.
Example Modules :

- **File System (fs)**: Express is lightweight and doesn't dictate how you should structure your application.
- **HTTP (http)**: It has a robust set of features that can be extended with additional modules.
- **URL (url):** Allows you to manage different HTTP routes via a simple and intuitive API.
- **Events (events):** Provides the ability to create and handle custom events.
- **Util (util):** Helps in accessing utility functions useful for programming.

Node.js Editor

Purpose: Any text editor or Integrated Development Environment (IDE) that supports JavaScript can be used to write Node.js code.
Recommended Editor :

- **Visual Studio Code:** Popular for its excellent support for JavaScript and Node.js with helpful extensions like ESLint, Nodemon, and others.

Node.js Compiler

Purpose: Node.js does not use a traditional compiling stage; instead, it interprets JavaScript code via the V8 JavaScript engine.
Example Usage: Node.js compiles JavaScript code to machine code using just-in-time (JIT) compilation during execution to improve performance.

Node.js Server

Purpose: Setting up a server to handle web requests and serve responses in a non-blocking manner.
Example Usage:

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

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

server.listen(8000, 'localhost', () => {
  console.log('Server running at http://localhost:8000/');
});
```

This creates a basic HTTP server that listens on port 8000.

Node.js MongoDB

Purpose: Integrate MongoDB with Node.js applications using MongoDB Node.js Driver.
Example Usage :

```
const { MongoClient } = require('mongodb');
const uri = "mongodb+srv://your-cluster-url";
const client = new MongoClient(uri);
async function run() {
  try {
    await client.connect();
    console.log("Connected correctly to server");
  } finally {
    await client.close();
  }
}
run().catch(console.dir);
```