Advance Node.js Concepts Cheat sheet

Middleware:

Definition :- Functions that execute during the request-response cycle.

Usage :- Modify req and res objects, end request-response cycle, or call the next middleware.

Example :-

```
app.use((req, res, next) => {
  Console.log('Request Type:', req.method);
  next();
});
```

Asynchronous Programming :-

- Callback :- Functions passed as arguments to be executed later.

```
fs.readfile('file.txt', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

- Promises :- Objects representing eventual completion or failure of async operations.

```
let promise = new promise ((resolve, reject) => {
  setTimeout(() => resolve("done!"), 1000);
});
promise.then(result =>
console.log(result));
```

Async/Await :- Syntactic sugar for promises, making async code look synchronous.

```
async function fetchdata () {
  let response = await fetch ('api/data');
  let data = await response.json ();
  console.log (data);
}
```

Event - Driven Architecture :-

- Event Emitter :- Core module to handle events.

```
const EventEmitter = require ('events');
const emitter = new EventEmitter ();
emitter.on ('event', () =>
console.log ('Event triggered'));
emitter.emit ('event');
```

Streams :-

- Readable & Writable Streams :- Handle data chunks

```
const fs = require ('fs');
let readable =
fs.createReadStream ('file.txt');
let writable =
fs.createWriteStream ('fileCopy.txt');
readable.pipe (writable);
```

- Duplex & Transform Streams :- Both read and write, can modify data.

```
const {Duplex} = require ('stream');
const duplex = new Duplex ({
read (size) {this.push ('data');
this.push (null); },
write (chunk, encoding, callback) {
```

```
        console.log (chunk.toString());
        callback(); }
    });
    duplex.pipe(duplex);
```

Cluster Module:-

- Purpose To exploit multi-core systems

```
const cluster = require('cluster');
Const http = require('http');
Const numCPUs =
require('os').cpus().length;

if (cluster.isMaster) {
 for (let i=0; i< numCPUs; i++) {
 cluster.fork(); }
 cluster.on('exit', (worker, code, signal) => {
   console.log (`worker ${worker.process.pid} died`);
   });
 } else {
   http.createServer((req, res) => {
    res.writeHead(200);
    res.end('Hello world\n');
 }).listen(8000);
}.
```