## Middleware -

Functions that execute during the request-response cycle. It is used for tasks such as logging, authentication, error handling etc.

- **Application-level Middleware :-**
  Bound to an instance of 'express'

```
app.use((req, res, next) => {
        console.log('Time:', Date.now());
        next();
});
```

- **Router level middleware :**

```
const router = express.Router();
router.use((req, res, next) => {
console.log('Router Time:', Date.now());
next();
});
```

- **Error-handling middleware :**

```
app.use((err, req, res, next) => {
        console.error(err.stack);
        res.status(500).send('Something
                              broke!');
});
```

- **Built in middleware :**
  'express.static', 'express.json', 'express.urlencoded'.
- **Third-party middleware :**
  'morgan', 'cors', 'body-parser', etc.

## ASYNCHRONOUS PROGRAMMING :

- **Callback functions :** Functions passed as arguments to be executed later.
- **Promises :** objects representing the eventual completion or failure of an asynchronous operation.

* Async/wait → Syntactic sugar over promises, making asynchronous code look synchronous.

EVENT LOOP →

It handles asynchronous operations, allowing non-blocking I/O.

Phases :—

Timers: Executes callbacks scheduled by 'setTimeout' and 'setInterval',

Pending Callbacks: Executes I/O callbacks deferred to the next loop.

Idle, Prepare: Internal use.

Poll: Retrieves new I/O events; executes I/O-related callbacks.

Check: Executes 'setImmediate' callbacks.

Close callbacks: Executes close event callbacks, eg :—

'Socket.on('close', --);',

Execution →

```
setTimeout(() => console.log('timeout'), 0);
setImmediate(() => console.log('immediate'));
process.nextTick(() => console.log('nextTick'));
// output : nextTick, timeout, immediate
```

Streams →

It handles reading, writing data piece-by-piece (chunks) instead of all at once.

Types :—

✓ Readable streams : 'fs.createReadStream()'

✓ Writable Streams : 'fs.createWriteStream()'

✓ Duplex Streams : Both readable & writable (eg. 'net.socket')

✓ Transform Streams : Modify or transform data while

reading or writing (eg: 'zlib.createGzip()')

Ex:

```
const fs = require('fs');
const readStream = fs.createReadStream('file.txt');
const writeStream = fs.createWriteStream('file-copy.txt');
readStream.pipe(writeStream);
readStream.on('end', () => console.log('File copied
                                    successfully'));
```

## Buffer and Crypto

✓ Buffer is raw binary data allocated outside V8 heap memory.

```
const buf = Buffer.from('hello');
console.log(buf.toString());
// output: Hello.
```

✓ Crypto provides cryptographic functionality

- Hashing
- Encryption

## Child Processes

It Runs scripts and commands in a different process.

Methods → .exec : Executes command & buffers the output

- spawn : Launches a new process with a given command
- fork : A special case of 'spawn' to create child processes specially to run Node.js scripts