Node.js 치트시트

1. Node.js 기본 개념

- Node.js: V8 JavaScript 엔진으로 빌드된 JavaScript 런타임
- 비동기 I/O: Non-blocking I/O 모델로 높은 성능 제공
- 이벤트 기반: 이벤트 루프를 통한 비동기 처리
- 단일 스레드: 메인 스레드는 하나지만 내부적으로 스레드 풀 사용
- NPM: Node Package Manager, 패키지 관리 도구

2. Node.js 설치 및 버전 관리

- Node.js 설치: https://nodejs.org에서 다운로드
- 버전 확인: `node --version` 또는 `node -v`
- NPM 버전 확인: `npm --version` 또는 `npm -v`
- nvm 설치: `curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash`
- nvm으로 Node 설치: `nvm install node` (최신), `nvm install 18.17.0` (특정 버전)
- 버전 전환: `nvm use 18.17.0`
- 기본 버전 설정: `nvm alias default 18.17.0`

3. NPM 기본 명령어

패키지 관리

- 초기화: `npm init` (대화형), `npm init -y` (기본값)
- 패키지 설치: `npm install <package>` 또는 `npm i <package>`
- 전역 설치: `npm install -g <package>`
- 개발 의존성: `npm install --save-dev <package>` 또는 `npm i -D <package>`
- 프로덕션 의존성: `npm install --save <package>` 또는 `npm i -S <package>`
- 패키지 제거: `npm uninstall <package>`
- 모든 의존성 설치: `npm install`

패키지 정보

- 설치된 패키지 목록: `npm list` (로컬), `npm list -g` (전역)
- 패키지 정보: `npm info <package>`
- 취약점 검사: `npm audit`
- 취약점 수정: `npm audit fix`
- 업데이트: `npm update` (모든 패키지), `npm update <package>` (특정 패키지)

4. 핵심 모듈

File System (fs)

```
const fs = require('fs');
// 동기 읽기
const data = fs.readFileSync('file.txt', 'utf8');
// 비동기 읽기
fs.readFile('file.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log(data);
});
// Promise 기반
const fsPromises = require('fs').promises;
const data = await fsPromises.readFile('file.txt', 'utf8');
```

HTTP 모듈

```
const http = require('http');
const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World!');
});
server.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

Path 모듈

```
const path = require('path');
path.join('/users', 'john', 'documents'); // /users/john/documents
path.resolve('file.txt'); // 절대 경로 반환
path.extname('file.txt'); // .txt
path.basename('/path/to/file.txt'); // file.txt
path.dirname('/path/to/file.txt'); // /path/to
```

OS 모듈

```
const os = require('os');
os.platform(); // darwin, linux, win32
os.arch(); // x64, arm64
os.cpus(); // CPU 정보
os.freemem(); // 사용 가능한 메모리
os.totalmem(); // 전체 메모리
os.homedir(); // 홈 디렉토리
```

5. 비동기 프로그래밍

Callback

```
function asyncOperation(callback) {
  setTimeout(() => {
    callback(null, 'Success');
  }, 1000);
}
asyncOperation((err, result) => {
  if (err) throw err;
  console.log(result);
});
```

Promise

```
function asyncOperation() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Success');
    }, 1000);
  });
}
asyncOperation()
  .then(result => console.log(result))
  .catch(err => console.error(err));
```

Async/Await

```
async function main() {
  try {
    const result = await asyncOperation();
    console.log(result);
  } catch (err) {
    console.error(err);
  }
}
```

6. Express.js 기본

설치 및 기본 설정

```
// npm install express
const express = require('express');
const app = express();
const PORT = 3000;

app.use(express.json()); // JSON 파싱
app.use(express.static('public')); // 정적 파일

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

라우팅

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

// POST 요청
app.post('/users', (req, res) => {
  const user = req.body;
  res.json(user);
});

// 경로 매개변수
app.get('/users/:id', (req, res) => {
  const id = req.params.id;
  res.json({ id });
});

// 쿼리 매개변수
app.get('/search', (req, res) => {
  const query = req.query.q;
  res.json({ query });
});
```

미들웨어

```
// 로깅 미들웨어
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  next();
});

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

7. 환경변수 및 설정

dotenv 사용

```
// npm install dotenv
require('dotenv').config();

// .env 파일
// PORT=3000
// DB_URL=mongodb://localhost:27017/myapp

const port = process.env.PORT || 3000;
const dbUrl = process.env.DB_URL;
```

process 객체

```
process.env.NODE_ENV; // 환경 (development, production)
process.argv; // 명령행 인수
process.cwd(); // 현재 작업 디렉토리
process.exit(0); // 프로세스 종료
```

8. 디버깅 및 로깅

디버깅

- Node.js 디버거: `node --inspect app.js`
- Chrome DevTools: chrome:
- VS Code 디버깅: launch.json 설정
- nodemon: `npm install -g nodemon`, 자동 재시작

로깅

```
// 기본 console 사용
console.log('Info message');
console.error('Error message');
console.warn('Warning message');

// Winston 라이브러리
// npm install winston
const winston = require('winston');
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' })
  ]
});
```

9. 테스팅

Jest

```
// npm install --save-dev jest
// package.json
{
  "scripts": {
    "test": "jest"
  }
}

// sum.test.js
function sum(a, b) {
  return a + b;
}

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

Mocha & Chai

```
// npm install --save-dev mocha chai
// test/test.js
const chai = require('chai');
const expect = chai.expect;

describe('Array', () => {
  it('should return -1 when the value is not present', () => {
    expect([1, 2, 3].indexOf(4)).to.equal(-1);
  });
});
```

10. 유용한 NPM 패키지

개발 도구

- nodemon: 파일 변경 시 자동 재시작
- concurrently: 여러 명령어 동시 실행
- cross-env: 크로스 플랫폼 환경변수 설정
- eslint: JavaScript 린팅 도구
- prettier: 코드 포매터

유틸리티

- lodash: JavaScript 유틸리티 라이브러리
- moment/dayjs: 날짜 처리 라이브러리
- axios: HTTP 클라이언트
- bcrypt: 비밀번호 해싱
- jsonwebtoken: JWT 토큰 생성/검증
- validator: 데이터 검증
- uuid: 고유 ID 생성

데이터베이스

- mongoose: MongoDB ODM
- sequelize: SQL ORM (PostgreSQL, MySQL, MariaDB, SQLite, Microsoft SQL Server)
- redis: Redis 클라이언트
- pg: PostgreSQL 클라이언트

11. 성능 최적화

클러스터링

```
const cluster = require('cluster');
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 {
  // 워커 프로세스에서 서버 실행
  require('./app.js');
}
```

스트림 사용

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

// 대용량 파일 처리
const readStream = fs.createReadStream('large-file.txt');
const rl = readline.createInterface({
  input: readStream,
  crlfDelay: Infinity
});

rl.on('line', (line) => {
  console.log(line);
});
```

메모리 사용량 모니터링

```
console.log(process.memoryUsage());
// {
//   rss: 4935680,      // 실제 메모리 사용량
//   heapTotal: 1826816, // 힙 총 크기
//   heapUsed: 650472,   // 힙 사용량
//   external: 49879     // V8 외부 메모리
// }
```

12. 배포 및 프로덕션

PM2 프로세스 매니저

```
# 설치
npm install -g pm2

# 앱 시작
pm2 start app.js

# 클러스터 모드
pm2 start app.js -i max

# 상태 확인
pm2 status

# 로그 보기
pm2 logs

# 재시작
pm2 restart app

# 중지
pm2 stop app

# 삭제
pm2 delete app
```

Docker

```
# Dockerfile
FROM node:18-alpine

WORKDIR /app
```

```
COPY package*.json ./
RUN npm ci --only=production

COPY . .

EXPOSE 3000

USER node

CMD ["node", "app.js"]
```

환경별 설정

```
// config/index.js
const config = {
  development: {
    port: 3000,
    db: 'mongodb://localhost:27017/app-
dev'
  },
  production: {
    port: process.env.PORT || 8080,
    db: process.env.DB_URL
  }
};

module.exports =
config[process.env.NODE_ENV ||
'development'];
```