# IT 기술실무
# Node.js 14주차

목표 : CLI(Command Line Interface)에 대하여 이해할 수 있다.

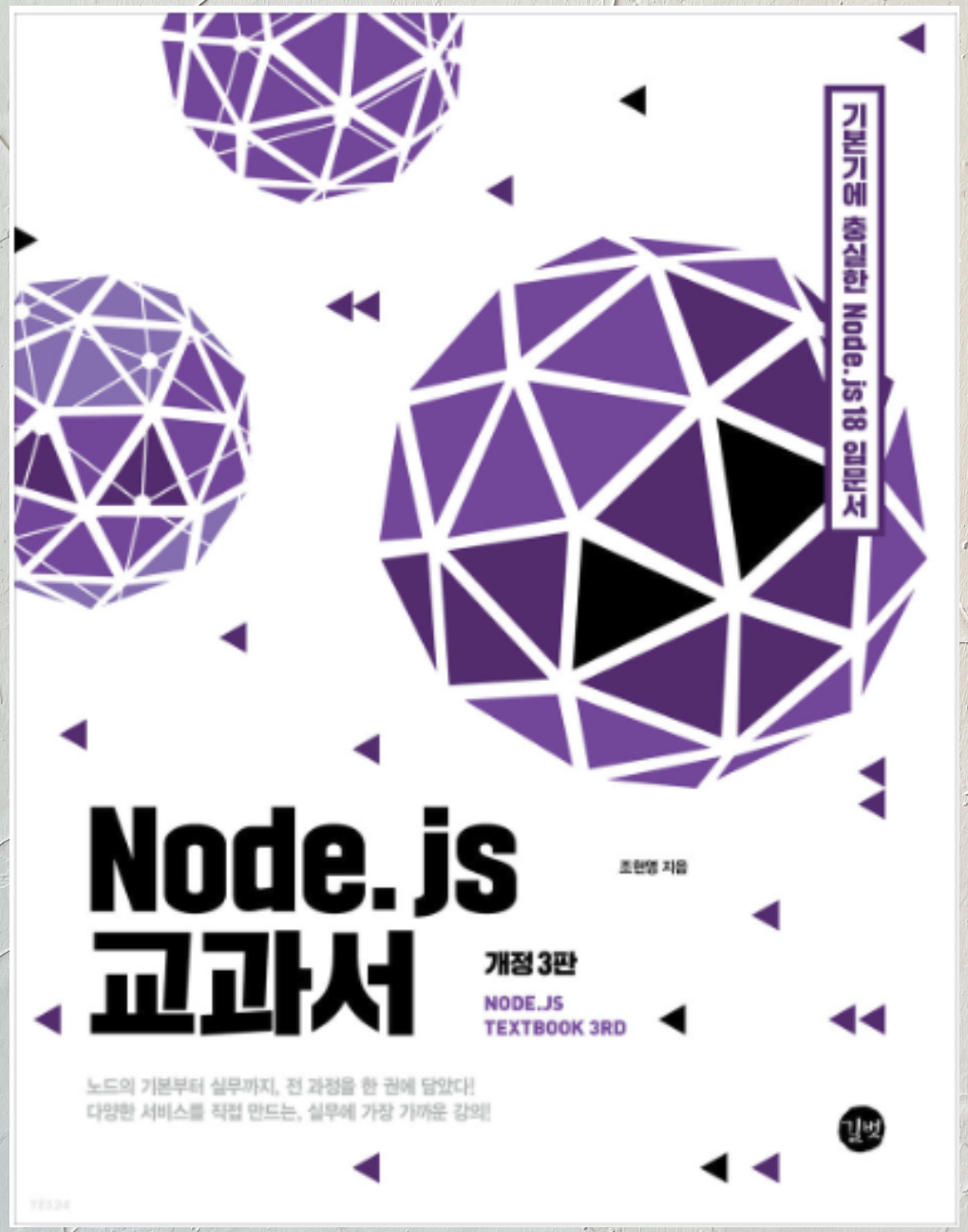
# 강의개요

| 주 | 주차별 목표 | 강의소개 | 교재 |
|---|---|---|---|
| 1 | 강의소개 등 | 강의소개 및 필수사항 공유 | 주/보조 |
| 2 | Abstract | 핵심 개념, 서버로서의 노드, 서버 외의 노드, 개발 환경 구축(23~64) | 주교재 |
| 3 | ES2015+ : ECMAScript 6 | ES2015+ 및 프런트엔드 자바스크립트(65~92) | 주교재 |
| 4 | Node's Functions | REPL, JS 파일 실행하기, 모듈로 만들기, 노드 내장 객체/모듈 등(93~178) | 주교재 |
| 5 | http Module | 요청/응답, REST와 라우팅, 쿠키/세션, https/http2, cluster(179~216) | 주교재 |
| 6 | Package Manager | npm, package.json, 패키지 배포하기(217~240) | 주교재 |
| 7 | Express Web Server | 미들웨어, req/res, 템플릿 엔진 사용하기(241~290) | 주교재 |
| 8 | 중간시험 | 중간시험 및 웹어플리케이션 추가 강의 | 주/보조 |
| 9 | Databases : MySQL | MySQL 설치, 데이터베이스/테이블, CRUD, 시퀄라이즈(291~364) | 주교재 |
| 10 | Databases : MongoDB | 몽고디비 설치, 데이터베이스/컬렉션, CRUD, 몽구스(365~416) | 주교재 |
| 11 | Web API Server | API 서버, JWT 토큰 인증하기, 다른 서비스 호출하기, SNS API, CORS 등(475~520) | 주교재 |
| 12 | Node Service Test | 테스트 준비하기, 유닛/커버리지/통합/부하(521~560) | 주교재 |
| 13 | Web Socket | 웹 소켓 이해하기, ws 모듈, Socket.io, 미들웨어와 소켓, 채팅 구현하기(561~608) | 주교재 |
| 14 | CLI | 간단한 콘솔 명령어, Commander, Inquirer 사용하기(649~676) | 주교재 |
| 15 | 기말시험 | 기말시험 및 웹개발자 로드맵 추가 강의 | 주/보조 |

# 목차

table of contents





# 14.1 간단한 콘솔 명령어 만들기

# 1. CLI

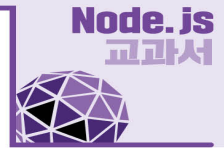


- CLI(Command Line Interface) 기반 노드 프로그램을 제작해보기
  - 콘솔 창을 통해서 프로그램을 수행하는 환경
  - 반대 개념으로는 GUI(그래픽 유저 인터페이스)가 있음
  - 리눅스의 셸이나 브라우저 콘솔, 명령 프롬프트 등이 대표적인 CLI 방식 소프트웨어
  - 개발자에게는 CLI 툴이 더 효율적일 때가 많음

그림 14-1 CLI 프로그램 화면

```
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli tmpl html -f main -d public/html
public\html\main.html 생성 완료
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli tmpl html -f main -d public/html
이미 해당 파일이 존재합니다
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli tmpl express-router
이미 해당 파일이 존재합니다
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli tmpl js
html 또는 express-router 둘 중 하나를 입력하세요.
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli
? 템플릿 종류를 선택하세요. html
? 파일의 이름을 입력하세요. hell
? 파일이 위치할 폴더의 경로를 입력하세요. .
? 생성하시겠습니까? Yes
hell.html 생성 완료
터미널을 종료합니다.
```

# 2. 콘솔 명령어




- 노드 파일을 실행할 때 node [파일명] 명령어를 콘솔에 입력함
  - node나 npm. nodemon처럼 콘솔에서 입력하여 어떠한 동작을 수행하는 명령어를 콘솔 명령어라고 부름.
  - node와 npm 명령어는 노드를 설치해야만 사용할 수 있음
  - nodemon, rimraf같은 명령어는 npm i –g 옵션으로 설치하면 명령어로 사용 가능
  - 패키지 명과 콘솔 명령어를 다르게 만들 수도 있음(sequelize-cli는 sequelize 명령어 사용)
  - 이러한 명령어를 만드는 게 이 장의 목표

# 3. 프로젝트 시작하기



» node-cli 폴더 안에 package.json과 index.js 생성

- index.js 첫 줄의 주석에 주목(윈도에서는 의미 없음)
- 리눅스나 맥 같은 유닉스 기반 운영체제에서는 /usr/bin/env에 등록된 node 명령어로 이 파일을 실행하라는 뜻

package.json

```
{
  "name": "node-cli",
  "version": "0.0.1",
  "description": "nodejs cli program",
  "main": "index.js",
  "author": "ZeroCho",
  "license": "ISC"
}
```

index.js

```
#!/usr/bin/env node
console.log('Hello CLI');
```

# 4. CLI 프로그램으로 만들기



- package.json에 다음 줄을 추가
  - bin 속성이 콘솔 명령어와 해당 명령어 호출 시 실행 파일을 설정하는 객체
  - 콘솔 명령어는 cli, 실행 파일은 index.js

package.json

```
{
  ...
  "license": "ISC",
  "bin": {
    "cli": "./index.js"
  }
}
```

# 5. 콘솔 명령어 사용하기



» npm i -g로 설치 후 cli로 실행

- 보통 전역 설치할 때는 패키지 명을 입력하지만 현재 패키지를 전역 설치할 때는 적지 않음

콘솔

```
$ npm i -g
+ node-cli@0.0.1
added 1 package from 1 contributor in 0.069s
```

- 리눅스나 맥에서는 명령어 앞에 sudo를 붙여야 할 수도 있음
- 전역 설치한 것이기 때문에 node_modules가 생기지 않음

콘솔

```
$ cli
Hello CLI
```

# 6. 명령어에 옵션 붙이기



» process.argv로 명령어에 어떤 옵션이 주어졌는지 확인 가능(배열로 표시)

index.js

```
#!/usr/bin/env node
console.log('Hello CLI', process.argv);
```

- 코드가 바뀔 때마다 전역 설치할 필요는 없음
- package.json 내용이 바뀌면 다시 전역 설치해야 함
- 배열의 첫 요소는 노드의 경로, 두 번째 요소는 cli 명령어의 경로, 나머지는 옵션

콘솔

```
$ cli one two three four
Hello CLI [
'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\speak\\AppData\\Roaming\\npm\\node_modules\\node-cli\\index.js',
  'one',
  'two',
  'three',
  'four'
]
```

# 7. 사용자로부터 입력 받기



## 노드 내장 모듈 readline 사용

- createInterface 메서드로 rl 객체를 만듦
- process.stdin, process.stdout은 각각 콘솔을 통해 입력받고 출력한다는 의미
- question 메서드로 질문을 표시하고 답변이 들어오면 콜백 함수가 실행됨
- 답변은 answer 매개변수에 담김

index.js

```
#!/usr/bin/env node
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.question('예제가 재미있습니까? (y/n) ', (answer) => {
  if (answer === 'y') {
    console.log('감사합니다!');
  } else if (answer === 'n') {
    console.log('죄송합니다!');
  } else {
    console.log('y 또는 n만 입력하세요.');
  }
  rl.close();
});
```

콘솔

```
$ npx cli
예제가 재미있습니까? (y/n) y
감사합니다!
```

# 8. 콘솔 내용 지우기



## console.clear로 콘솔 내용 지우기

- 프로그램 시작 시와, 잘못된 답변 후에 콘솔 지움

index.js

```
#!/usr/bin/env node
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

console.clear();
const answerCallback = (answer) => {
  if (answer === 'y') {
    console.log('감사합니다!');
    rl.close();
  } else if (answer === 'n') {
    console.log('죄송합니다!');
    rl.close();
  } else {
    console.clear();
    console.log('y 또는 n만 입력하세요.');
    rl.question('예제가 재미있습니까? (y/n) ', answerCallback);
  }
};

rl.question('예제가 재미있습니까? (y/n) ', answerCallback);
```

콘솔

```
$ npx cli
(화면 정리 및 입력 시작)
예제가 재미있습니까? (y/n) [y나 n 외의 다른 답변]
(화면 정리됨)
y 또는 n만 입력하세요.
예제가 재미있습니까? (y/n) n
죄송합니다!
(입력 종료)
```

# 9. 템플릿을 만들어주는 명령어 만들기



## template.js 작성

template.js

```
#!/usr/bin/env node
const fs = require('fs');
const path = require('path');

const type = process.argv[2];
const name = process.argv[3];
const directory = process.argv[4] || '.';
const htmlTemplate = `
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Template</title>
  </head>
  <body>
    <h1>Hello</h1>
    <p>CLI</p>
  </body>
</html>
`;

const routerTemplate = `
const express = require('express');
const router = express.Router();

router.get('/', (req, res, next) => {
  try {
    res.send('ok');
  } catch (error) {
    console.error(error);
    next(error);
  }
});

module.exports = router;
`;

const exist = (dir) => { // 폴더 존재 확인 함수
  try {
    fs.accessSync(dir, fs.constants.F_OK | fs.constants.R_OK | fs.constants.W_OK);
    return true;
  } catch (e) {
    return false;
  }
};

const mkdirp = (dir) => { // 경로 생성 함수
  const dirname = path
    .relative('.', path.normalize(dir))
    .split(path.sep)
    .filter(p => !!p);
  dirname.forEach((d, idx) => {
    const pathBuilder = dirname.slice(0, idx + 1).join(path.sep);
    if (!exist(pathBuilder)) {
      fs.mkdirSync(pathBuilder);
    }
  });
};

const makeTemplate = () => { // 템플릿 생성 함수
  mkdirp(directory);
  if (type === 'html') {
    const pathToFile = path.join(directory, `${name}.html`);
    if (exist(pathToFile)) {
      console.error('이미 해당 파일이 존재합니다');
    } else {
      fs.writeFileSync(pathToFile, htmlTemplate);
      console.log(pathToFile, '생성 완료');
    }
  } else if (type === 'express-router') {
    const pathToFile = path.join(directory, `${name}.js`);
    if (exist(pathToFile)) {
      console.error('이미 해당 파일이 존재합니다');
    } else {
      fs.writeFileSync(pathToFile, routerTemplate);
      console.log(pathToFile, '생성 완료');
    }
  } else {
    console.error('html 또는 express-router 둘 중 하나를 입력하세요.');
  }
};

const program = () => {
  if (!type || !name) {
    console.error('사용 방법: cli html|express-router 파일명 [생성 경로]');
  } else {
    makeTemplate();
  }
};

program(); // 프로그램 실행부
```

# 10. 템플릿을 만들어주는 명령어 만들기



- 디렉토리가 존재하는지 확인하는 exist 함수와 디렉토리를 생성하는 mkdirp 함수를 만듦
- program이라는 함수는 template.js의 실행부, makeTemplate은 옵션을 읽어서 알맞은 템플릿을 작성해주는 함수
- 옵션에 따라 다른 동작을 하도록 분기 처리
- package.json의 명령어를 바꿔주고 전역 재설치

package.json

```
{
  ...
  "bin": {
    "cli": "./template.js"
  }
}
```

콘솔

```
$ npm i -g
$ cli
사용 방법: cli html|express-router 파일명 [생성 경로]
$ cli js main ./public
html 또는 express-router 둘 중 하나를 입력하세요.
$ cli html main public/html
public\html\main.html 생성 완료
$ cli express-router index ./routes
routes\index.js 생성 완료
$ cli express-router index ./routes
이미 해당 파일이 존재합니다.
```

# 11. 단계적 명령어 만들기


- template.js를 다음과 같이 수정
  - https://github.com/ZeroCho/nodejs-book/blob/master/ch14/14.1/node-cli/template.js
  - 옵션을 입력하지 않는 경우 readline 모듈로 단계적으로 질문을 해 옵션을 외울 필요가 없도록 함
  - 옵션을 입력하는 경우 예전과 마찬가지로 동작
  - dirAnswer, nameAnswer, typeAnswer는 각각 디렉터리, 파일명, 템플릿 종류에 대해 사용자 입력을 받는 함수(코드의 순서가 역순)

콘솔

```
$ npx cli
어떤 템플릿이 필요하십니까? html
파일명을 설정하세요. test
저장할 경로를 설정하세요.(설정하지 않으면 현재경로) public
public\test.html 생성 완료
```

Note ≡ CLI 프로그램 삭제 방법

CLI 프로그램을 삭제하는 방법은 간단합니다. npm 전역 제거 명령어를 호출하면 됩니다.

콘솔

```
$ npm rm -g node-cli
```

# 14.2 Commander, Inquirer 사용하기

# 1. 패키지로 쉽게 CLI 프로그램 만들기



- npm에는 CLI 프로그램을 위한 라이브러리가 많이 준비되어 있음
  - commander(CLI)와 inquirer(사용자와 상호작용), chalk(콘솔에 컬러)를 사용해서 예제를 만들어 봄
  - 14.1의 프로그램을 commander와 inquirer로 재작성할 것
  - chalk는 터미널에 색을 입히기 위한 용도

콘솔

```
$ npm i commander@9 inquirer@8
```

# 2. commander 사용하기



## command.js 파일 작성

- version: 프로그램의 버전 설정(--version 또는 -v로 확인)
- usage: 프로그램 사용 방법 기입(--help로 또는 –h로 확인)
- command: 명령어 등록(template <type>과 * 등록함)
  - <>는 필수 옵션을 의미
  - []는 선택 옵션을 의미
- description: 명령어에 대한 설명을 설정하는 메서드
- alias: 명령어에 대한 별칭
- option: 명령어에 대한 옵션을 등록
  - --옵션 [값] 또는 –옵션 <값> 형식
  - 두 번째 인자는 설명, 세 번째 인자는 기본값
- Action: 명령어가 실행될 때 수행할 동작 등록
- parse: process.argv를 파싱하여 옵션 등록

command.js

```
#!/usr/bin/env node
const { program } = require('commander');

program
  .version('0.0.1', '-v, --version')
  .name('cli');

program
  .command('template <type>')
  .usage('<type> --filename [filename] --path [path]')
  .description('템플릿을 생성합니다.')
  .alias('tmpl')
  .option('-f, --filename [filename]', '파일명을 입력하세요.', 'index')
  .option('-d, --directory [path]', '생성 경로를 입력하세요', '.')
  .action((type, options, command) => {
    console.log(type, options.filename, options.directory);
  });

program
  .command('*', { noHelp: true })
  .action(() => {
    console.log('해당 명령어를 찾을 수 없습니다.');
    program.help();
  });

program
  .parse(process.argv);
```

# 3. commander 프로그램 실행하기



## 프로그램 전역 재설치 후 실행해보기

- -v, -h로 버전, 설명 확인 가능하고 필수 옵션도 자동으로 체크해줌

**package.json**

```
{
  ...
  "bin": {
    "cli": "./command.js"
  },
  ...
}
```

**콘솔**

```
$ npm i -g
```

**콘솔**

```
$ npx cli -v
0.0.1
$ npx cli -h
Usage: cli [options] [command]

Options:
  -v, --version                   output the version number
  -h, --help                      display help for command

Commands:
  template|tmpl [options] <type>  템플릿을 생성합니다.
  help [command]                  display help for command
$ npx cli template -h
Usage: cli template|tmpl <type> --filename <filename> --path [path]

템플릿을 생성합니다.

Options:
  -f, --filename [filename]  파일명을 입력하세요. (default: "index")
  -d, --directory [path]     생성 경로를 입력하세요 (default: ".")
  -h, --help                  display help for command
$ npx cli template
error: missing required argument `type'
```

# 4. template.js를 커맨더로 전환하기



## command.js 수정

- template.js를 붙여 넣은 후 첫 require 부분과 끝 program 부분만 수정하면 됨

command.js

```
#!/usr/bin/env node
const { program } = require('commander');
const fs = require('fs');
const path = require('path');

program
  .version('0.0.1', '-v, --version')  .name('cli');

program
  .command('template <type>')
  .usage('<type> --filename [filename] --path [path]')
  .description('템플릿을 생성합니다.')
  .alias('tmpl')
  .option('-f, --filename [filename]', '파일명을 입력하세요.', 'index')
  .option('-d, --directory [path]', '생성 경로를 입력하세요', '.')
  .action((type, options) => {
    makeTemplate(type, options.filename, options.directory);
  });

program
  .command('*', { noHelp: true })
  .action(() => {
    console.log('해당 명령어를 찾을 수 없습니다.');
    program.help();
  });

program
  .parse(process.argv);
```

# 5. 전환된 파일 실행하기



» 명령어들을 커맨더 식으로 전환함

- 옵션들은 순서를 바꿔서 입력해도 됨

콘솔

```
$ npx cli template html -d public/html -f new
public\html\new.html 생성 완료
$ npx cli copy
해당 명령어를 찾을 수 없습니다.
Usage: cli [options] [command]

Options:
  -v, --version                   output the version number
  -h, --help                      display help for command

Commands:
  template|tmpl [options] <type>  템플릿을 생성합니다.
  help [command]                  display help for command
$ cli
(결과 없음)
```

# 6. inquirer 사용하기



» 여전히 옵션들을 외워야 하는 불편함

- inquirer로 상호 작용 추가

command.js

```
#!/usr/bin/env node
const { program } = require('commander');
const fs = require('fs');
const path = require('path');
const inquirer = require('inquirer');
```

```
program
  .action((options, command) => {
    if (command.args.length !== 0) {
      console.log('해당 명령어를 찾을 수 없습니다.');
      program.help();
    } else {
      inquirer.prompt([{
        type: 'list',
        name: 'type',
        message: '템플릿 종류를 선택하세요.',
        choices: ['html', 'express-router'],
      }, {
        type: 'input',
        name: 'name',
        message: '파일의 이름을 입력하세요.',
        default: 'index',
      }, {
        type: 'input',
        name: 'directory',
        message: '파일이 위치할 폴더의 경로를 입력하세요.',
        default: '.',
      }, {
        type: 'confirm',
        name: 'confirm',
        message: '생성하시겠습니까?',
      }])
        .then((answers) => {
          if (answers.confirm) {
            makeTemplate(answers.type, answers.name, answers.directory);
            console.log('터미널을 종료합니다.');
          }
        });
    }
  })
  .parse(process.argv);
```

# 7. inquirer API



## readline보다 간결해짐

- 커맨더의 액션이 실행되지 않은 경우 triggered가 false라서 inquirer가 실행됨
- prompt 메서드로 상호작용 창 띄울 수 있음
  - type: 질문의 종류( input, checkbox, list, password, confirm 등)
  - 예제에서는 input(평범한 답변), list(다중 택일), confirm(Yes or No) 사용
  - name: 질문의 이름, 답변 객체 속성명으로 질문의 이름을, 속성 값으로 질문의 답을 가짐
  - message: 사용자에게 표시되는 문자열(여기에 질문을 적음)
  - choices: type이 checkbox, list 등인 경우 선택지를 넣는 곳(배열로)
  - default: 답 적지 않았을 때 기본값
- 예제에서는 질문 네 개를 연달아 하고 있음
- 질문의 name이 type, name, directory라서 각각의 답변이 answers.type, answers.name, answers.director에 들어 있음.

콘솔

```
$ npx cli
? 템플릿 종류를 선택하세요. (Use arrow keys)
> html
  express-router
```

콘솔

```
? 템플릿 종류를 선택하세요. html
? 파일의 이름을 입력하세요. new
? 파일이 위치할 폴더의 경로를 입력하세요. public/html
? 생성하시겠습니까? y
이미 해당 파일이 존재합니다
터미널을 종료합니다.
```

# 8. chalk

콘솔

```
$ npm i chalk@4
```

## 콘솔에 색을 추가함

- 소스 코드는 https://github.com/ZeroCho/nodejs-book/blob/master/ch14/14.2/node-cli/command.js
- console.log과 console.error에 chalk 적용
- 문자열을 chalk 객체의 메서드로 감싸면 됨
- chalk.green, chalk.yellow, chalk.red 등등
- chalk.rgb(12, 34, 56)(문자열) 또는 chalk.hex('#123456')(텍스트)도 가능
- 배경색도 지정 가능해서 chalk.bgGreen, chalk.bgYellow, chalk.bgRgb, chalk.bgHex
- 동시에 지정하려면 chalk.red.bgBlue.bold처럼 하면 됨

```
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli tmpl html -f main -d public/html
public\html\main.html 생성 완료
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli tmpl html -f main -d public/html
이미 해당 파일이 존재합니다
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli tmpl express-router
이미 해당 파일이 존재합니다
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli tmpl js
html 또는 express-router 둘 중 하나를 입력하세요.
PS C:\Users\speak\WebstormProjects\nodejs-book\ch14\14.3\node-cli> npx cli
? 템플릿 종류를 선택하세요. html
? 파일의 이름을 입력하세요. hell
? 파일이 위치할 폴더의 경로를 입력하세요. .
? 생성하시겠습니까? Yes
hell.html 생성 완료
터미널을 종료합니다.
```

# 9. 프로그램을 공유하고 싶다면?



- 만든 CLI 프로그램을 공유하고 싶다면 5장의 과정대로 npm에 배포하면 됨
  - 다른 사용자가 npm i –g <패키지명>을 한다면 다운로드 받아 사용할 수 있음

# 감사합니다.

**단순하게 설명할 수 없다면**
**제대로 이해하지 못한 것이다.**

아인슈타인