KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

2가지 모듈을 활용하자

동기와 비동기 함수를 제공한다

파일 관리하기 - path, File System 모듈

[KB] IT's Your Life

임의의 파일(사용자가 드래그 앤 드랍 직접하지 않는)
이
프로그램으로 열리는 것은 원천 차단함.

해당 모듈을 통해서 백엔드에서 해당 작업을 할 수 있다.

KB국민은행

path 모듈

- 파일 경로나 디렉터리 경로를 다룸
  - 운영체제 간에 경로를 구분하는 구분자가 다름 → 경로 구분자를 통일
- 경로를 나누거나 합칠 수 있음

경로 결합, 특정 부분 추출, 절대 경로 <->상대경로 변환

- 절대경로와 상태 경로

윈도우는 패스 표현시 \
리눅스 맥은 /

해당 모듈에서는 / 로 통일해서 사용해라
이유는 윈동의 패스 구분자는 이스케이프로 해석될 수 있어서

1 path 모듈

경로를 다루는 주요 함수

해당 교재 단원에서느 commonjs 방식을 따르고 있다. (ES 방식 말고)

path 모듈 가져오기

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

모듈명만 있으니까? 써드파티 혹은 내장 표준 모듈에서 검색 시작

경로 합치기

가변 매개변수

```
path.join(경로1, 경로2, ...)
```

결합시 운영체제에 맞게끔 변환도 해준다,
반환시 문자열

chapter03/sec01/path.js

```
const path = require('path');          혹은 const {join> = require('path'); //분할 할당 기법 사용해도 됨. 하나만 필요시

// 경로 연결하기
const fullPath = path.join('some', 'work', 'ex.txt');
console.log(fullPath);
```

```
some\work\ex.txt
```

1 path 모듈

경로만 추출하기 – dirname 함수

- path.dirname(경로)

chapter03/sec01/path.js

```
const path = require('path'); 내장 모듈로 인식
                              ./path 내가 만든 모듈                    전역변수 __dirname __filename 활용하기

// 경로 연결하기
const fullPath = path.join('some', 'work', 'ex.txt');
console.log(fullPath);

// 절대 경로
console.log(`파일 절대 경로: ${__filename}`);__filename 변수 내용 확인

// 경로 추출하기
const dir = path.dirname(__filename);  해당 파일에 해당하는 패쓰만 추출
console.log(`경로만: ${dir}`);  확인
```

```
some\work\ex.txt
파일 절대 경로: c:\workspace\node\src\03\path.js
경로만: c:\workspace\node\src\03
```

파일 이름 추출하기 – basename 함수

- path.basename(경로)
  - 경로를 제외한 파일명만 리턴
- path.basename(경로, 확장자)
  - 지정한 확장자를 제외한 파일명 리턴

파일명을 노드에서는 basename이라고 한다

확장자는 윈도우os에만 잇는 개념.

리눅스나 맥에서는 확장자라는 개념 없음. 그냥 다 파일명의 일부

chapter03/sec01/path.js

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


…

// 파일 이름 추출하기
const fn = path.basename(__filename);
const fn2 = path.basename(__filename, '.js');

console.log(`파일 이름: ${fn}`);
console.log(`파일 이름(확장자 제외): ${fn2}`);
```

해당 방법은 .js라는 확장자를 아는 상태에서 제거하는 상황.

확장자가 뭔지 모르겠으면?
확장자 추출함수 이용

```
…
파일 이름: path.js
파일 이름(확장자 제외): path
```

1 path 모듈

- **확장자 추출하기 – extname 함수**
  - path.extname(경로)

chapter03/sec01/path.js

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

…


// 파일 확장자 추출
const ext = path.extname(__filename);
console.log(`파일 확장자: ${ext}`);
console.log(path.basename(__filename, ext));
```

```
…
파일 확장자: .js
path
```

경로를 객체로 반환하기 – parse 함수

- path.parse(경로)

내장된 파서 함수

```
{
  root,       // 루트 디렉터리
  dir,        // 디렉터리 경로
  base, // 파일명.확장명
  ext,        // 확장명
  name // 파일명
}
```

chapter03/sec01/path.js

앞서 본 path 관련 함수들은

파싱만 하지

진짜 잇는지는 별로 중요 ㄴㄴ

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

…
                                파서함수로 경로 분해 하여 여러 정보 추출하기
// 경로 분해하기
const parsedPath = path.parse(__filename); 여러 정보들을 객체 형태로 반환한다
console.log(parsedPath);
```

```
…
{
  root: 'c:\\',
  dir: ' c:\\workspace\\node\\src\\03',
  base: 'path.js',
  ext: '.js',
  name: 'path'
}
```

FS 모듈 살펴보기

Fie System 모듈의 약자

실제 i/o가 일어남

비동기 처리 방법에 따라 사용하는 함수가 다름

- 동기 처리 함수
- 콜백 처리 함수 비동기죠 제공 교재에서는 콜백 방식만 제공됨
- Promise API 프로미스로 동작하는 비동기방식

FS 모듈 가져오기

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

콜백함수

fs.함수명

```
fs.readFile('example.txt', (err, data) => { ... });
```

콜백함수의 매개변수는 항상 첫번째 매개변수는 err객체, 두번째부터 결과데이터 매개변수

현재 디렉터리 읽기

파일 목록을 추출한다는 의미

동기 처리로 디렉터리 읽기 – readdirSync 함수

fs.readdirSync(경로 [, 옵션])

- 경로: 파일 목록을 표시할 경로를 지정 ==디렉토리 경로
- 옵션
  - encoding: 기본값 utf8

node에서는
동기와 비동기함수를 구분하는 관례가 있다.

함수명에서 sync가 붙으면 동기, 안붙으면 비동기 함수

chapter03/sec02/list-1.js

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

let files = fs.readdirSync('./');
console.log(files);
```

./ 현재 디렉토리 == 프로세스의 working directory

```
[ 'package.json', 'node_modules', 'package-lock.json', '01', '03' ]
```

결과.

왜 list-1.js가 안나오고 저게 나올까(루트 디렉토리의 내용이 왜 나올까)

vscode 터미널에서 node llist-1.js시
해당 프로세스의 working directorysms

해당 프로세스의 부모 프로세스의 working directory를
따른다. 상ㅇ속 받는다
즉 해당 프로세스의 working direcotry는 부모 프로세스의 그것과 같다.

vscode말고 다른 터미널이면
해당 터미널의 working 디렉토리로 바뀐다.

해당 터미널에서 cd명령어로 해당 프로세스의 working dir가
바뀌므로

그럼 원래 의도 대로 원하는 디렉토리 정보를 얻으려면

터미널의 workingdirectory를cd명령어로 옮겨야 한다.

불편하다 그 때는 경로 설정 './'로 하지말고 __dirname이라는 모듈 전역변수를 사용하자. __dirname변수는 현재 파일의 경로를 의미하는 변수다

정리 : working directory기분이면 상대경로로. 현재 파일의 경로를 기준으로하고 싶을 땐 __dirname변수로

현재 디렉터리 읽기

비동기 처리로 디렉터리 읽기 – readdir 함수

fs.readdir(경로[, 옵션], 콜백)

- 경로: 파일 목록을 표시할 경로를 지정
- 옵션
  - encoding: 기본값 utf8
  - withFileTypes: 기본값은 false, true면 반환값이 몬자열로 된 배열이 아닌 디렉터리 항목으로된 배열로 반환
- 콜백함수: (err, files)

매개변수 형식

1번째 err객체, 2번째 결과 데이터

동기함수의 에러 처리는

try catch로 감싸서 처리해야함

err객체에 대한 정보를 받지 않기 때문에

chapter03/sec02/list-2.js

번외 js에서 false로 해석하는 값
false 0 '' undefined null

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

fs.readdir('./', (err, files) => {
  if (err) {
    console.error(err);
    return;
  }
  console.log(files);
});
```

err가 생기면...

```
[ 'package.json', 'node_modules', 'package-lock.json', '01', '03' ]
```

파일 읽기 – readFileSync 함수, readFile 함수

동기 처리로 파일 읽기 – readFileSync 함수

fs.readFileSync(경로 [,옵션])

- 경로
- 옵션
  - encoding: 기본값 null default encode는 utf-8, 윈도우에서는 다른 문자셋
  - flag: 기본값 r, r+(읽기&쓰기), w(쓰기), a(추가) 등

파일 오픈 옵션

vscode에서 텍스트파일을 만들면 utf-8 charset을 사용해 생성한다

chapter03/sec03/example.txt

```
Node.js is an open-source, cross-platform JavaScript runtime environment.
Node.js는 Chrome v8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다.
```

옵션으로 들어가는 encode옵션 매개변수는

열려는 파일의 charset과 일치해야한다

chapter03/sec03/read-1.js

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

const data = fs.readFileSync('./example.txt');
console.log(data);
```

bin형식으로 읽어라.

상대 경로이니
working directory가 어느 위치에 있는지 주의.

```
<Buffer 4e 6f 64 65 2e 6a 73 20 69 73 20 61 6e 20 6f 70 65 6e 2d 73 6f 75 72 63 65 2c 20 63 72
6f 73 73 2d 70 6c 61 74 66 6f 72 6d 20 4a 61 76 61 53 63 72 69 ... 110 more bytes>
```

chapter03/read-2.js

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

const data = fs.readFileSync('./example.txt', 'utf-8');  // 인코딩 지정
console.log(data);
```

```
Node.js is an open-source, cross-platform JavaScript runtime environment.
Node.js는 Chrome v8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다.
```

파일 읽기 – readFileSync 함수, readFile 함수

비동기 처리로 파일 읽기 – readFile 함수

fs.readFile(파일 [, 옵션], 콜백)

- 파일
- 옵션
  - encoding
  - flag
  - signal: 파일을 읽는 데 시간이 너무 걸릴 경오 중간 취소를 위해 설정
- 콜백
  - (erro, data) 매개변수

nodejs 홈페이지에서 공식문서를
살펴보면 더 자세한 정보를 알 수 있다
내장 모듈과 해당 함수에서 더 알아봇자

chapter03/sec03/read-3.js

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

fs.readFile('./example.txt', 'utf-8', (err, data) => {
  if (err) {
    console.error(err);
  }
  console.log(data);
});
```

```
Node.js is an open-source, cross-platform JavaScript runtime environment.
Node.js는 Chrome v8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다.
```

파일에 기록하기 – writeFileSync 함수, writeFile 함수

동기 처리로 파일에 쓰기 – writeFileSync 함수

fs.writeFileSync(파일, 내용 [, 옵션])

- 파일: 내용을 기록할 파일 경로
- 내용: 기록할 내용을 지정
- 옵션
  - encoding: 기본값 utf8
  - flag: 기본값 w
  - mode: 파일 사용자 권한, 기본값 0o666

chapter03/sec03/write-1.js

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

const data = fs.readFileSync('./example.txt', 'utf8');
fs.writeFileSync('./text-1.txt', data);
```

상대 경로니 ㅏ까
working directory 경로 어딘지 주시

해당 파일이 없을 수 있으니 존재 여부 검사 코드

파일에 기록하기 – writeFileSync 함수, writeFile 함수

파일 존재 여부 체크하기 – existsSync 함수

fs.existsSync(파일)

chapter03/sec03/write-2.js

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

const data = fs.readFileSync('./example.txt', 'utf8');

if(fs.existsSync('text-1.txt')) {     // text-1.txt 파일이 있다면
  console.log('file already exist');
} else {                              // text-1.txt 파일이 없다면
  fs.writeFileSync('./text-1.txt', data);
}
```

여기선 쓸 파일이
이미 존재하는지
검사하는 역할로

파일에 기록하기 – writeFileSync 함수, writeFile 함수

비동기 처리로 파일에 쓰기 – writeFile 함수

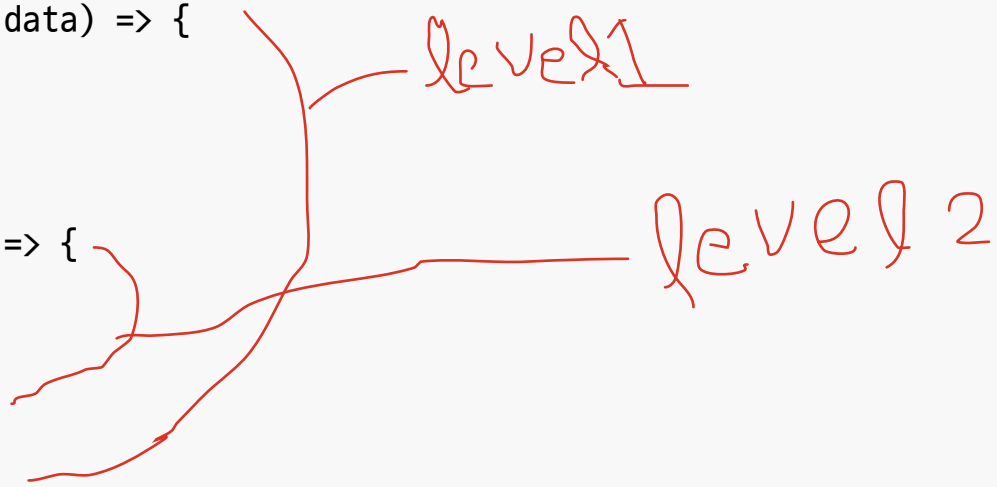
fs.writeFile(파일, 내용[, 옵션], 콜백)

- 파일: 내용을 기록할 파일 경로
- 내용: 기록할 내용을 지정
- 옵션
  - encoding: 기본값 utf8
  - flag: 기본값 w
  - mode: 파일 사용자 권한, 기본값 0o666
  - signal: 쓰기 취소 설정
- 콜백: err => {}

해당 함수의 콜백함수는 매개변수가 하나

chapter03/sec03/write-3.js

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

fs.readFile('./example.txt', 'utf8', (err, data) => {
  if (err) {
    console.log(err);
  }

  fs.writeFile('./text-2.txt', data, (err) => {
    if (err) {
      console.log(err);
    }
    console.log('text-2.txt is saved!');
  });
});
```



그냥 콜백방식 만르고
promise 방식을 사용하면 되지 않느나?

비동기 함수 중에는 그래서 콜백함수 방식말고 (앞서 봤더것들)

promise를 반환하는 함수 버전이 또 있다.

모듈로부터 promise를 반환하는 비동기 함수를 추출하려면
requre혹은 imporot시 경로 지정을 특정한 형식으로 하면 된다

비동기 함수에다가 패스지정시 'node:경로/promises' 형식으로 지정하면
return이 promise인 함수를 준다.

파일에 기록하기 – writeFileSync 함수, writeFile 함수

기존 파일에 내용 추가하기 – flag 옵션 사용하기

flag값	설명
"a"	내용을 추가하기 위해 파일을 엽니다. 파일이 없으면 만듭니다.
"ax"	"a"와 같지만 파일이 이미 있으면 실패합니다.
"a+"	파일을 읽고 내용을 추가하기 위해 파일을 엽니다. 파일이 없으면 만듭니다.
"ax+"	"ax"와 같지만 파일이 있을 경우 실패합니다.
"as"	동기 처리로 내용을 추가하기 위해 파일을 엽니다. 파일이 없으면 만듭니다.
"w"	쓰기 위해 파일을 엽니다. 파일이 없으면 만듭니다.
"wx"	"w"와 같지만 파일이 있을 경우 실패합니다.
"w+"	내용을 읽고 쓰기 위해 파일을 엽니다. 파일이 없으면 만듭니다.
"wx+"	"wx"와 같지만 파일이 있을 경우 실패합니다.

append가 아닌
그냥 write에 기본 동작은
기존 데이터를 아예 다 지워버리고 새로 작성
!=
overwrite

chapter03/sec03/write-4.js

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

let content = `
================================
새로운 내용을 추가해 보겠습니다.
================================
`;
                                                    파일 오픈 옵션 플래그 지정
fs.writeFileSync('./text-1.txt', content, { flag: 'a' });
```

```
Node.js is an open-source, cross-platform JavaScript runtime environment.
Node.js는 Chrome v8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다.
================================
새로운 내용을 추가해 보겠습니다.
================================
```

파일에 기록하기 – writeFileSync 함수, writeFile 함수

기존 파일에 내용 추가하기 – appendFileSync, appendFile 함수

fs.appendFileSync(파일, 내용 [, 옵션])
fs.appendFileSync(파일, 내용 [, 옵션], 콜백)

chapter03/sec03/write-5.js

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

fs.appendFile('./text-2.txt', '\n\n 새로운 내용 추가', (err) => {
  if (err) {
    console.log(err);
  }
  console.log('appending to file');
});
```

```
Node.js is an open-source, cross-platform JavaScript runtime environment.
Node.js는 Chrome v8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다.

 새로운 내용 추가
```

파일 삭제하기 – unlinkSync 함수, unlink 함수

파일 링크 개념은 리눅스에 잇는 개념이다.

동기 처리로 파일 삭제하기

fs.unlinkSync(파일)

실제 파일 데이터와 파일명참조자의 연결을 끊어내느 것을 unlink라고 한다.

unlinke가 파일 데이터를 직접 삭제하는 연산은 아니지만

실제 파일 데이터를 가리키는 파일명 참조자가
하나도 없으면
실제 파일 데이터는 삭제된다.

chapter03/sec03/unlink-1.js

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

fs.unlinkSync('./text-1.txt');
console.log('file deleted');
```

chapter03/sec03/unlink-2.js

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

if (!fs.existsSync('./text-1.txt')) {   // 파일이 없다면
  console.log('file does not exist');
} else {                                // 파일이 있다면
  fs.unlinkSync('./text-1.txt');
  console.log('file deleted');
}
```

파일 삭제하기 – unlinkSync 함수, unlink 함수

- 비동기 처리로 파일 삭제하기

  fs.unlink(파일, 콜백)

chapter03/unlink-3.js

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

if (!fs.existsSync('./text-2.txt')) { // 파일이 없다면
  console.log('file does not exist');
} else {                              // 파일이 있다면
  fs.unlinkSync('./text-2.txt', (err) => {
    if(err) {
      return console.error(err);
    }
    console.log('file deleted');
  });

}
```

디렉터리 만들기 및 삭제하기

디렉터리 만들기 – mkdirSync, mkdir 함수

fs.mkdirSync(경로 [, 옵션])

fs.mkdir(경로 [, 옵션], 콜백)

- 경로
- 옵션
  - recursive: 중간 경로가 존재하지 않으면 생성할지 여부, 기본값 false
  - mode: 사용자 권한, 기본값 0o777
- 콜백: err => {}

chapter03/sec04/dir-1.js

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

if (fs.existsSync('./test')) { // 디렉터리가 있다면
  console.log('folder already exists');
} else {                       // 디렉터리가 없다면
  fs.mkdir('./test', (err) => {
    if (err) {
      return console.error(err);
    }
    console.log('folder created');
  });
}
```

chapter03/dir-2.js

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

if (fs.existsSync('./test2/test3/test4')) {  // 디렉터리가 있다면
  console.log('folder already exists');
} else {                                     // 디렉터리가 없다면
  fs.mkdir('./test2/test3/test4', { recursive: true }, (err) => {
    if (err) {
      return console.error(err);
    }
    console.log('folder created');
  });
}
```

디렉터리 만들기 및 삭제하기

빈 디렉터리 삭제하기 – rmdirSync, rmdir 함수

fs.rmdirSync(경로 [, 옵션])

fs.rmdir(경로 [, 옵션], 콜백)

- 경로
- 옵션
  - maxRetries: 오류 발생시 재시도 횟수, 디폴트 0
  - retryDelay: 재시도 회수를 지정했을 때 개기 시간(밀리초), 기본값 100
- 콜백: err => {}

chapter03/sec04/dir-3.js

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

if (fs.existsSync('./test')) {
  // 삭제할 디렉토리가 있다면
  fs.rmdir('./test', (err) => {
    if (err) return console.error(err);
    console.log('folder deleted');
  });
} else {
  // 삭제할 디렉토리가 없다면
  console.log('folder does not exist');
}
```

디렉터리 만들기 및 삭제하기

파일 삭제 및 내용이 있는 디렉터리 삭제하기 – rmSync, rm 함수

fs.rmSync(경로 [, 옵션])

fs.rm(경로[, 옵션], 콜백)

- 경로
- 옵션
  - force: 파일이나 디렉터리를 강제로 삭제할지 여부, 기본값 false
  - maxRetries: 최대 재시도 횟수
  - retryDelay: maxRetries 설정시 대기 시간. 기본값 100 밀리초
  - recursive: 하위 폴더까지 삭제할지 여부, 기본값 false
- 콜백: err => {}

chapter03/sec04/dir-4.js

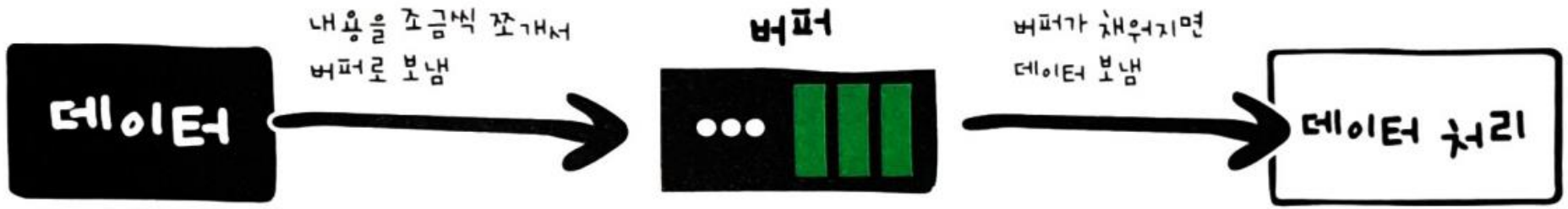
```
fs = require('fs');

fs.rm('./test2', { recursive: true }, (err) => {
  if (err) return console.error(err);

  console.log('folder deleted');
});
```

모듈 관련 끝

버퍼

- 임시 데이터를 저장하는 물리적인 메모리 공간
- 노드의 버퍼 크기는 고정되어 있음

보통 64kb 크기



데이터 임시 저장 장소, 버퍼

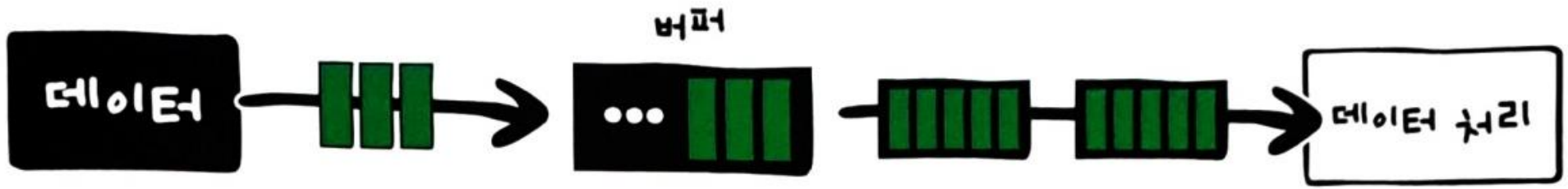
chapter03/sec05/buffer.js

```
const fs = require('fs');
                          옵션 없으니 bin데이터.  data매개변수에는 bin데이터 대입됨. toString메소드로 문자열로 바로 표현 가능함
fs.readFile('example.txt', (err, data) => {
  if (err) return console.log(err);

  console.log(data); // 이진 데이터 표시
  console.log('\n');
  console.log(data.toString()); // 문자열로 변환해서 표시
});
```

```
<Buffer 4e 6f 64 65 2e 6a 73 20 69 73 20 61 6e 20 6f 70 65 6e 2d 73 6f 75 72 63 65 2c 20 63 72 6f 73 73 2d 70 6c 61 74 66 6f 72
6d 20 4a 61 76 61 53 63 72 69 ... 110 more bytes>


Node.js is an open-source, cross-platform JavaScript runtime environment.
Node.js는 Chrome v8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다.
```

스트림

- 한 곳에서 다른 곳으로 데이터가 이동하는 것, 데이터의 흐름



데이터의 흐름, 스트림

종류	설명
리더블 스트림	데이터를 읽기 위한 스트림. 네트워크로 연결해서 데이터를 읽어 오거나 파일에서 데이터를 읽어 올 때 사용
라이터블 스트림	데이터를 쓰기 위한 스트림. 네트워크에 연결한 상태에서 데이터를 기록하거나 파일에 데이터를 기록할 때 사용
듀플렉스 스트림	읽기와 쓰기 모두 가능한 스트림. 리더블 스트림과 라이터블 스트림을 결합한 형태로 실시간 양방향 통신에 사용

리더블 스트림 readable stream

- 데이터를 읽기 위한 스트림으로 주로 서버에서 용량이 큰 데이터를 가져올 때 많이 사용
- createReadStream 함수로 생성

fs.createReadStream(경로, [인코딩, 옵션])

- 경로
- 옵션
  - flags: 기본값 r
  - encoding: 인코딩 방식, 기본값 null
  - fd: 이미 열린 파일의 번호, 지정되면 파일 열기 생략, 기본값 null
  - mode: 사용자 접근 모드. 기본값 0o666
  - autoClose: 읽기가 끝난 후 파일을 자동으로 닫을 지 여부. 기본값 true
  - start: 읽기 시작 위치 지정
  - end: 어디까지 읽을지 지정. 기본값 infinity

리더블 스트림 readable stream

이벤트

이벤트	설명
data	데이터를 읽을 수 있을 때마다 발생하는 이벤트. 스트림에서 읽은 데이터를 처리할 때 data 이벤트를 사용.
end	스트림에서 데이터를 모두 읽었을 때 발생하는 이벤트. 데이터를 모두 읽었다는 사실을 인지하고 이후 작업이 필요할 때 사용
error	스트림에서 오류가 생겼을 때 발생하는 이벤트

이벤트 처리

.on('이벤트', 콜백)

on의 반환 값은 본인을 부른 스트림 객체

writable stream

이벤트

finish 더이상 쓸게 없을때

chapter03/sec05/stream-1.js

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

const rs = fs.createReadStream('./readMe.txt');

rs.on('data', (chunk) => {
    console.log('new chunk received:');
    console.log(chunk.length, chunk);
  })
  .on('end', () => {
    console.log('finished reading data');
  })
  .on('error', (err) => {
    console.error(`Error reading the file: ${err}`);
  });
```

콜백함수는 chunk 정보를 받는다

```
new chunk received:
65536 <Buffer 4e 6f 64 65 2e 6a 73 20 69 73 20 61 6e 20 6f 70 65 6e 2d 73 6f 75 72 63 65 2c 20 63 72 6f 73 73 2d 70 6c
61 74 66 6f 72 6d 20 4a 61 76 61 53 63 72 69 ... 65486 more bytes>
new chunk received:
65536 <Buffer 9c 20 eb b9 8c eb 93 9c eb 90 9c 20 4a 61 76 61 53 63 72 69 70 74 20 eb 9f b0 ed 83 80 ec 9e 84 ec 9e 85
eb 8b 88 eb 8b a4 2e 0d 0a 4e 6f 64 65 2e 6a ... 65486 more bytes>
new chunk received:
57036 <Buffer eb b9 8c eb 93 9c eb 90 9c 20 4a 61 76 61 53 63 72 69 70 74 20 eb 9f b0 ed 83 80 ec 9e 84 ec 9e 85 eb
8b 88 eb 8b a4 2e 0d 0a 4e 6f 64 65 2e 6a 73 20 ... 56986 more bytes>
finished reading data
```

라이터블 스트림 writable stream

데이터를 기록하는 스트림

fs.createWriteStream(경로, 내용[, 옵션])

- 경로
- 옵션
  - flags: 기본값 w
  - encoding: 인코딩 방식, 기본값 null
  - fd: 이미 열린 파일의 번호, 지정되면 파일 열기 생략, 기본값 null
  - mode: 사용자 접근 모드. 기본값 0o666
  - autoClose: 읽기가 끝난 후 파일을 자동으로 닫을 지 여부. 기본값 true
  - start: 쓰기 시작 위치 지정

스트림ㅇ의 크최대 크기 만큼만 읽을 수 있기에
최대 크기 이상의 것을 계속해서 쓰고 읽으려면 루프로 읽고 쓰기를 반복해야 한다

읽기 스트림의 결과를 바로 쓰기 스트림으로 연결도 할 수 있다.

2개의 스트림을 연결하는 파이프 – pipe

리눅스의 파이프 개념을 생각하면된다

- data 이벤트가 발생했을 때 따로 가져오기 기록하던 것을 한꺼번에 처리
- 이벤트 처리를 하지 않아도 됨
  - fs.readStream.pipe(writeStream [, 옵션])

두개의 만들어진 스트림을 연결하는 함수

읽기의 결과를 바로 쓰기의 내용으로

- 동작 방식
  - 리더블 스트림에서 데이터 읽기
  - 읽은 데이터를 라이터블 스트림으로 기록
  - 라이터블 스트림에 다 기록할 때까지 리더블 스트림에서 읽고 쓰기 반복
  - 리더블 스트림에서 더 이상 읽을 데이터가 없거나, 라이터블 스트림에 더 이상 쓸 데이터가 없으면 pipe 함수가 자동 종료

2개의 스트림을 연결하는 파이프 – pipe

- pipe 함수를 사용하지 않을 때

```
fs.readStream.on('data', (chunk) => {
  fs.writeStream.write(chunk)
});
```

- pipe 함수를 사용했을 때

```
fs.readStream.pipe(writeStream);
```

chapter03/sec05/pipe.js

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

const rs = fs.createReadStream('./readMe.txt', 'utf8');
const ws = fs.createWriteStream('./writeMe.txt');

rs.pipe(ws);
```

요청 url 들어갈 수 있음

응답 출력 url 이 될 수 있음

한번 es모듈 버전으로도 해보고

promise 기반의 함수버전ㄴ들로도 해봐라

파이프함수의 반환값은 writable

예) es6방식

```
import fs from 'node:fs/promises';

async function read(file) {

try {
const data = await fs.readFile(file,'utf-8');
console.log(data)
return data;
}
catch(e) {
console.error(e);
}

read('./example.txt');
```

commonjs promises방식

```
const fs=require('fs').promises;
```