1~2. 리액트 학습 준비

(리액트 소개, ES6+)

Prof. Seunghyun Park (sp@hansung.ac.kr)

Division of Computer Engineering

한성대학교 HANSUNG UNIVERSITY

학습 목표: 1. 리액트 소개 & 2. 리액트를 위한 자바스크립트

- React 소개
  - 참고: MVC 구조와 리액트
  - 개발환경 설정
    - node, npm, yarn (각자)
- ES2015+
  - 변수와 상수
  - 템플릿 문자열
  - 함수
  - 화살표 함수
  - 구조분해 할당
  - 객체 리터럴
  - 스프레드 연산자
  - 비동기 자바스크립트
  - 클래스

구조분해 할당

- 구조 분해*destructuring*

  속성

  : 객체 안의 필드, 배열을 구성하는 원소를

  변수에 쉽게 대입할 수 있도록 활용

  - 객체의 구조분해 할당

```
const { var_name } = object_name;
```

  - 배열의 구조분해 할당

```
const [ var_name ] = array_name;
```

추가로 확인해 볼 사항:
- bread, meat의 순서를 바꿔볼 것
- breat, meat 대신 다른 이름을 사용해 볼 것

```
/* ch02-04-01-destructuring.html */

// 객체 구조분해
var sandwich =  {
  bread: "더치-크런치",
  meat: "참치",
  cheese: "스위스",
  toppings: ["상추", "토마토", "머스타드"]
}

var {bread, meat} = sandwich
console.log(bread, meat)

bread = "마늘"
meat = "칠면조"
console.log(bread, meat)

console.log(sandwich.bread, sandwich.meat)
```

객체와 같은 이름의 변수에 값을 할당

변수이름 = 저장할 속성

변수에 새로운 값을 할당하고 출력

기존 객체 참조

```
더치-크런치 참치
마늘 칠면조
더치-크런치 참치
```

사본이었기 때문에 원본은 영향을 안받았다

객체의 구조분해 할당

```
/* ch02-04-02-destructuring.html */

// 객체를 인자로 받는 함수
const lordify = regularPerson => {
  console.log(`캔터베리의 ${regularPerson.firstname}`)
}

const regularPerson = {
  firstname: "현석",
  lastname: "오"
}

lordify(regularPerson)
```

캔터베리의 현석

```
/* ch02-04-03-d

// 객체 인자 구조분해
const lordify = ({firstname}) =>
  console.log(`캔터베리의 ${firstname}`)


const regularPerson = {
  firstname: "현석",
  lastname: "오"
}

lordify(regularPerson)
```

캔터베리의 현석

```
const { firstname } = regularPerson;
```
이 문장이 생략된 것으로 해석

괄호생략 금지

배열의 구조분해 할당 배열이라서 순서 중요

```
/* ch02-04-04-destructuring.html */

// 배열 구조분해
const [firstResort] = ["용평","평창","강촌"]
console.log(firstResort)
```

변수 한 자리

용평

리스트 매칭: 불필요한 변수는 ,를 사용해 생략

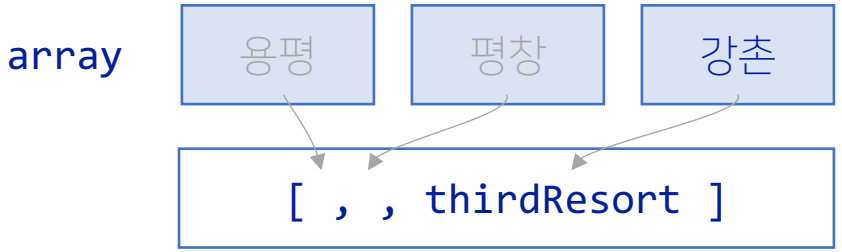
```
/* ch02-04-05-destructuring.html */

const array = ["용평","평창","강촌"];
const [,,thirdResort] = array;

// 배열 구조 분해(다른 예)
const [,,thirdResort] = ["용평","평창","강촌"]

console.log(thirdResort)
```

강촌



객체 리터럴

```
/* ch02-04-08-object-literal-enhancement.html */

// 예전 방식의 객체 리터럴
const name = "Julia Mancuso"
const sound = "go-fast"

var skier = {
  name: name,
  sound: sound,
  powderYell: function() {
    var yell = this.sound.toUpperCase()
    console.log(`${yell} ${yell} ${yell}!!!`)
  },
  speed: function(mph) {
    this.speed = mph
    console.log('속력(mph):', mph)
  }
}

skier.powderYell()
skier.speed(350)
console.log(JSON.stringify(skier))
```

프로퍼티 메소드

```
GO-FAST GO-FAST GO-FAST!!!
속력(mph): 350
{"name":"Julia Mancuso","sound":"go-fast","speed":350}
```

```
/* ch02-04-09-object-literal-enhancement.html */

// 객체 리터럴 개선
const name = "Julia Mancuso"
const sound = "go-fast"

const skier = {
  name,
  sound,
  powderYell() {
    const yell = this.sound.toUpperCase()
    console.log(`${yell} ${yell} ${yell}!!!`)
  },
  speed(mph) {
    this.speed = mph
    console.log('속력(mph):', mph)
  }
}

skier.powderYell()
skier.speed(350)
console.log(JSON.stringify(skier))
```

지움

속성과 할당할 변수가 이름이 같으면, 속성만 명시 가능

매서드 선언 시, function 키워드 생략 가능

스프레드 연산자 (..., 계속)

```
/* ch02-04-10-spread-operator.html */

// 스프레드 연산자
var peaks = ["대청봉", "중청봉", "소청봉"]
var canyons = ["천불동계곡", "가야동계곡"]
var seoraksan = [...peaks, ...canyons]

console.log(seoraksan.join(', '))
```

모든 원소를 나열

```
대청봉, 중청봉, 소청봉, 천불동계곡, 가야동계곡
```

```
/* ch02-04-11-spread-operator.html */

 // .reverse()가 peaks 배열을 변경함
var peaks = ["대청봉", "중청봉", "소청봉"]
var [last] = peaks.reverse()

console.log(last)
console.log(peaks.join(', '))
```

reverse(): 배열의 순서를 반전
※ 원본을 변형하고, 참조를 반환

join(): 배열의 모든 요소를 연결
※ 하나의 문자열로 생성하여 반환

```
소청봉
소청봉, 중청봉, 대청봉
```

```
/* ch02-04-12-spread-operator.html */

// peaks를 스프레드 연산자로 복사한 후 reverse 수행
var peaks = ["대청봉", "중청봉", "소청봉"]
var [last] = [...peaks].reverse()

console.log(last)
console.log(peaks.join(', '))
```

사본

peaks를 복사한 배열로 reverse()
※ 원본은 영향받지 않음

```
소청봉
대청봉, 중청봉, 소청봉
```

스프레드 연산자 (...)

```
/* ch02-04-14-spread-operator.html */

// 스프레드 연산자로 인자를 배열로 바꾸기
// 스프레드 연산자로 인자 중
function directions(...args) {
  var [start, ...remaining] = args
  var [finish, ...stops] = remaining.reverse()

  console.log(`${args.length} 도시를 운행합니다.`)
  console.log(`${start}에서 출발합니다.`)
  console.log(`목적지는 ${finish}입니다.`)
  console.log(`중간에 ${stops.length}군데 들립니다.`)
}
directions("서울","수원","천안","대전","대구","부산")
```

rest parameter: 나머지 원소들

remaining: [수원, 천안, 대전, 대구, 부산]

remaining: [부산, 대구, 대전, 천안, 수원]
finish: 부산, stops: [대구, 대전, 천안, 수원]

```
6 도시를 운행합니다.
서울에서 출발합니다.
목적지는 부산입니다.
중간에 4군데 들립니다.
```

```
/* ch02-04-15-spread-operator.html */

// 객체에 대한 스프레드 연산자
var morning = {
  breakfast: "미역국",
  lunch: "삼치구이와 보리밥"
}
var dinner = "스테이크 정식"
var backpackingMeals = {
  ...morning,
  dinner
}

console.log(backpackingMeals)
```

객체의 요소를 나열
단, 객체는 배열과 달리 순서는 중요하지 않음

morning 과 ...morning의 결과 비교해 볼 것

```
{ breakfast: '미역국', lunch: '삼치구이와 보리밥', dinner:
'스테이크 정식'}
```

비동기 자바스크립트 (계속)

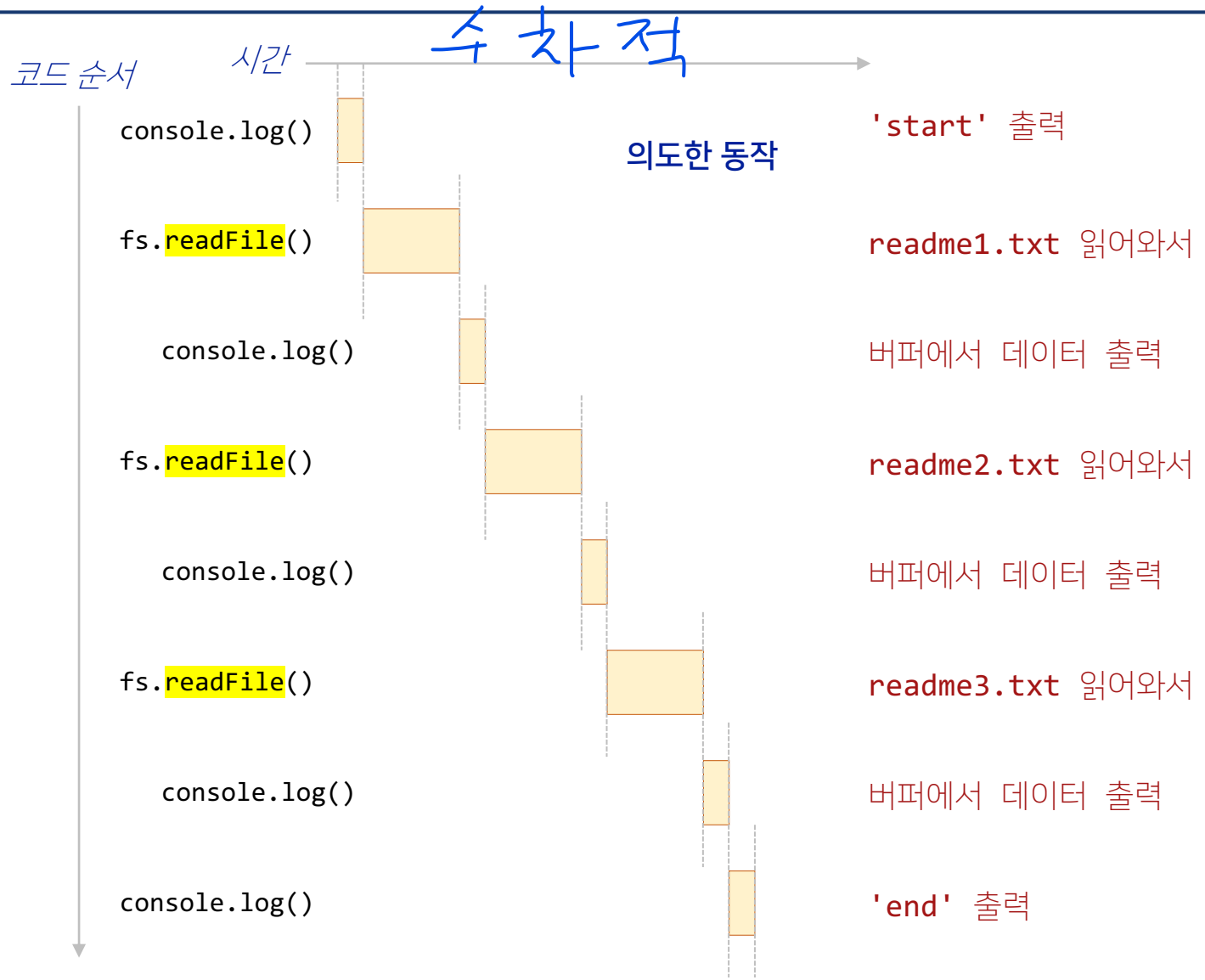
비동기 : 동시에 수행하지 않는다

예제) 3개의 텍스트 파일을 순서대로 읽어와 콘솔에 출력하는 프로그램 구현

출력 예:

```
start
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
3rd reading: This file contains sample text #3.
end
```

비동기 자바스크립트 (계속)




```
start
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
3rd reading: This file contains sample text #3.
end
```

비동기 자바스크립트 (계속): code1

실제로는 순서가 이상하다

```
/* ex-promise-01-async.js */

const fs = require('fs'); 모듈가져옴
console.log('start');

fs.readFile('./readme1.txt', (err, data) => {
    if (err)
        console.error(err);
    else
        console.log('1st reading:', data.toString());
});

fs.readFile('./readme2.txt', (err, data) => {
    if (err)
        console.error(err);
    else
        console.log('2nd reading:', data.toString());
});

fs.readFile('./readme3.txt', (err, data) => {
    if (err)
        console.error(err);
    else
        console.log('3rd reading:', data.toString());
});
console.log('end');
```

asynchonized codes

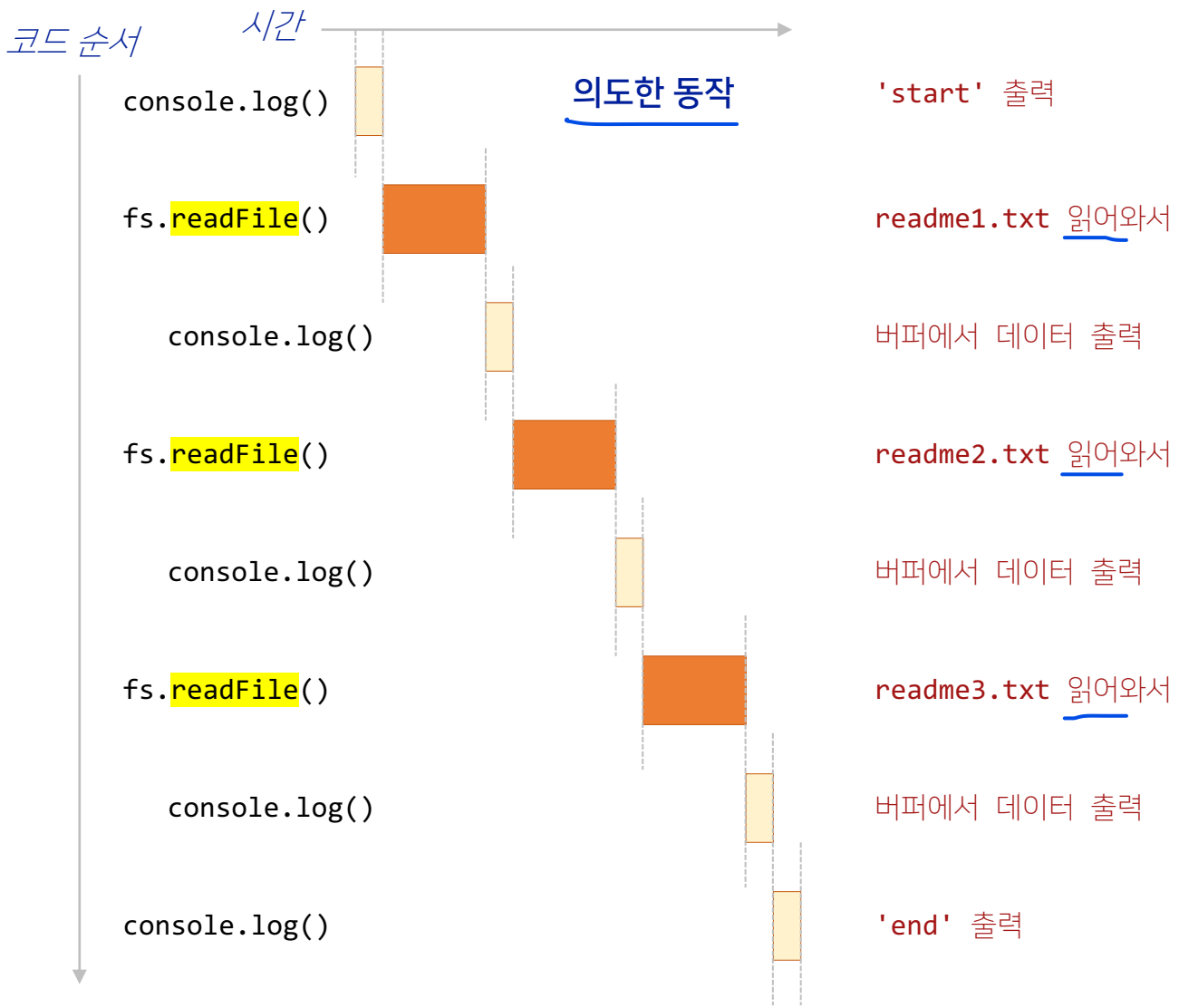
```
start
end
3rd reading: This file contains sample text #3.
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
```

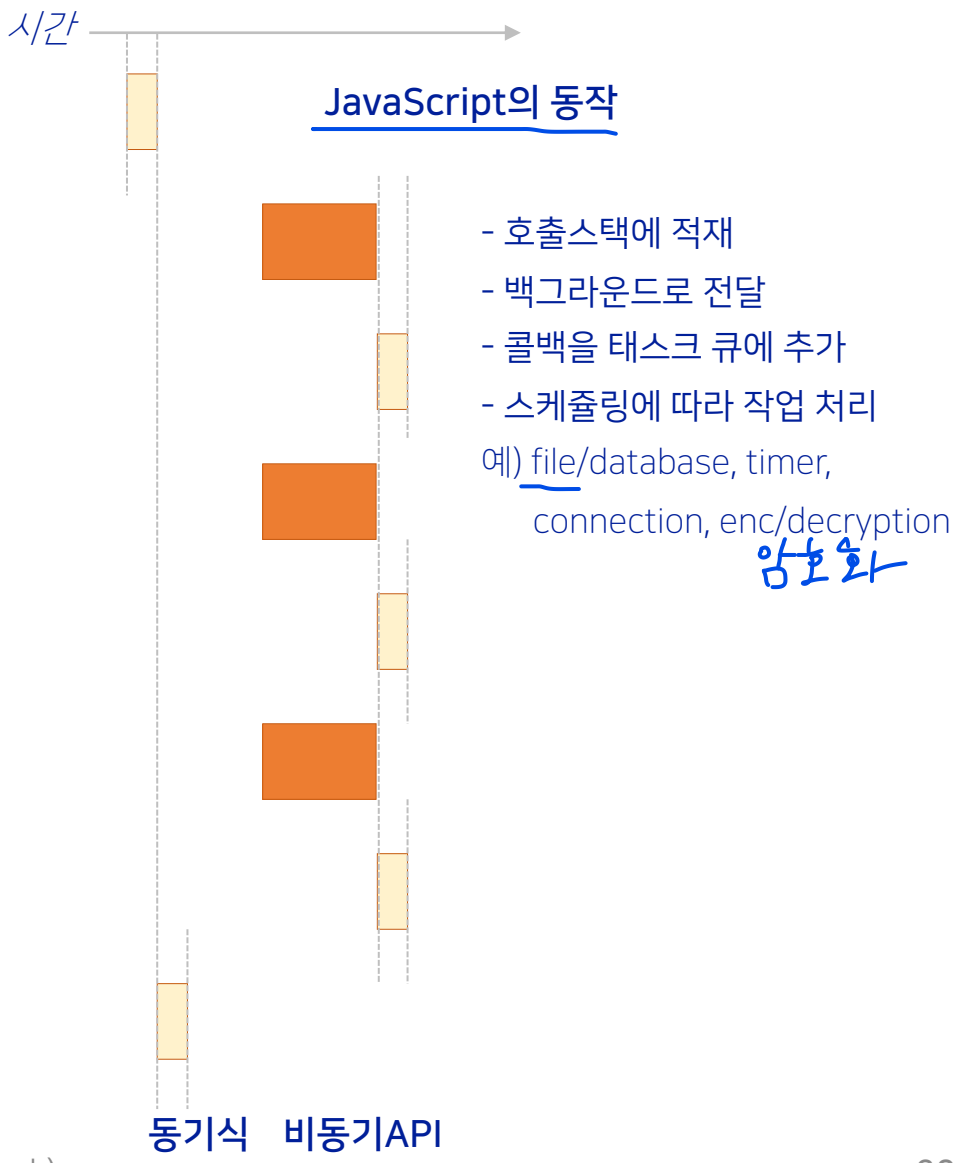
```
start
end
1st reading: This file contains sample text #1.
3rd reading: This file contains sample text #3.
2nd reading: This file contains sample text #2.
```

```
start
end
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
3rd reading: This file contains sample text #3.
```

```
start
end
2nd reading: This file contains sample text #2.
1st reading: This file contains sample text #1.
3rd reading: This file contains sample text #3.
```

비동기 자바스크립트 (계속): code1의 동작 분석





비동기 자바스크립트 (계속): code2

```
/* ex-promise-02-callback.js */

const fs = require('fs');
console.log(`start`);

fs.readFile('./readme1.txt', (err, data) => {
    if (err)
        console.error(err);
    else {
        console.log('1st reading:', data.toString());
        fs.readFile('./readme2.txt', (err, data) => {
            if (err)
                console.error(err);
            else {
                console.log('2nd reading:', data.toString());
                fs.readFile('./readme3.txt', (err, data) => {
                    if (err)
                        console.error(err);
                    else
                        console.log('3rd reading:', data.toString());
                    console.log(`end`);
                });
            }
        });
    }
});
```

callback 활용 해결법

callback 안에 callback
코드의 가독성이 떨어짐

콜백 지옥

비동기지만 현재 작업이 끝나야 다음으로 넘어갈 수 있다

```
start
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
3rd reading: This file contains sample text #3.
end
```

성공, 그러나 가독성 떨어짐

비동기 자바스크립트 (계속): code3

해결법 2. 다음 페이지의 promise를 먼저 알자

```
/* ex-promise-03-promise.js */

const fs = require('fs');  promise with promise chaining

const promise = new Promise( (resolve, reject) => {
    console.log('start');
    fs.readFile('./readme1.txt', (err, data) => {
        if (err)
            reject(err);
        else
            resolve(data);
    })
})

promise
.then(data => {
    console.log('1st reading:', data.toString());
    return new Promise( (resolve, reject) => {
        fs.readFile('./readme2.txt', (err, data) => {
            if (err)
                reject(err);
            else
                resolve(data);
        });
    });
})
.then(data => {
    console.log('2nd reading:', data.toString());
    return new Promise( (resolve, reject) => {
        fs.readFile('./readme3.txt', (err, data) => {
            if (err)
                reject(err);
            else
                resolve(data);
        });
    });
})
.then(data => console.log('3rd reading:', data.toString()))
.catch(err => console.error(err.message))
.finally(()=> console.log('end'));
```

성공하면 then으로 이동시키고 또 Promise를 만드는 반복

실패 무조건 수행

```
start
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
3rd reading: This file contains sample text #3.
end
```

promise

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Promise

- Promise
  - 비동기 방식으로 실행하지만,
    아직 결과를 반환하지 않은 객체
    - 성공: resolve(value)는 .then으로 연결
    - 실패: rejected(value)는 .catch로 연결
  - 미래의 어떤 시점에 결과를 제공하겠다는 약속
    - 비동기 연산 종료 후 결과와 실패에 대한 처리기로 활용

```
hello
world
success
default
```

```
/* ex-promise-00.js */
```

asynchronised code with promise

```
const condition = true;
console.log('hello');

const promise = new Promise((resolve, reject) => {
    if (condition)
        resolve('success');
    else
        reject('fail');
});

console.log('world');

promise
.then(message => {
    console.log(message);
})
.catch(err => {
    console.error(err);
})
.finally(() => {
    console.log('default');
});
```

fulfilled from resolve case

rejected from reject case

default case

성공여부 상관없이 실행

비동기 자바스크립트 (계속): code4

fs 모듈에도 promise 가 있다

```
/* ex-promise-04-promise.js */

fs.promises with promise chaining

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

console.log('start');

fsPromises.readFile('./readme1.txt')
.then(data => {
    console.log('1st reading:', data.toString());
    return fsPromises.readFile('./readme2.txt');
})
.then(data => {
    console.log('2nd reading:', data.toString());
    return fsPromises.readFile('./readme3.txt');
})
.then(data => console.log('3rd reading:', data.toString()))
.catch(err => console.error(err.message))
.finally(() => console.log('end'))
```

계속 return과 then

fs.readFile 과 다름, 콜백함수가 없음

fsPromise에 Promise 객체가 포함되어있다

```
start
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
3rd reading: This file contains sample text #3.
end
```

비동기 자바스크립트 (계속): code5

async & await

```
/* ex-promise-05-async-await.js */

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

console.log('start');
(async () => {
    try {
        let data = await fsPromises.readFile('./readme1.txt');
        console.log('1st reading:', data.toString());

        data = await fsPromises.readFile('./readme2.txt');
        console.log('2nd reading:', data.toString());

        data = await fsPromises.readFile('./readme3.txt');
        console.log('3rd reading:', data.toString());
    }
    catch(err){
        console.error(err.message);
    }
    finally{
        console.log('end');
    }
})();
```

await 쓰려면 반드시 async 필요

오래 걸리는 파일 읽기에 await을 붙여서, 앞작업이 끝날때까지 대기

```
start
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
3rd reading: This file contains sample text #3.
end
```

동기식 코드: code6

```
/* ex-promise-06-sync.js */

const fs = require('fs');
console.log('start');

try {
    let data = fs.readFileSync('./readme1.txt');
    console.log('1st reading:', data.toString());

    data = fs.readFileSync('./readme2.txt');
    console.log('2nd reading:', data.toString());

    data = fs.readFileSync('./readme3.txt');
    console.log('3rd reading:', data.toString());
}
catch(err){
    console.error(err.message);
}
finally{
    console.log('end');
}
```

synchonized codes

무조건 순서대로 수행,
오래 걸리는 게 앞에 있을 경우
비효율적이게 될 수 있음

```
start
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
3rd reading: This file contains sample text #3.
end
```

비동기 자바스크립트: promise, fetch

fetch는 Promise 객체를 반환

```
/* ch02-05-01-promises.html */

const getFakeMembers = count => new Promise((resolves, rejects) => {
  const api = `https://api.randomuser.me/?nat=US&results=${count}`
  const request = new XMLHttpRequest()
  request.open('GET', api)
  request.onload = () =>
    (request.status === 200) ?
      resolves(JSON.parse(request.response).results) :
      reject(Error(request.statusText))
  request.onerror = (err) => rejects(err)
  request.send()
})

getFakeMembers(5).then(
  members => console.log(members),
  err => console.error(
      new Error("randomuser.me에서 멤버를 가져올 수 없습니다."))
)
```

ch02-05-01-1 코드와 비교

URL

URL에 자원을 요청

정상인 경우, 결과를 json으로 받아서

results 속성의 데이터 추출

members에 5개의 원소를 가진 배열을 반환

```
(5) [{…}, {…}, {…}, {…}, {…}]
 0: {gender: 'female', name: {…}, location: {…}, ...', login: {…}, …}
 1: {gender: 'female', name: {…}, location: {…}, ...', login: {…}, …}
 2: {gender: 'male', name: {…}, location: {…}, ...', login: {…}, …}
 3: {gender: 'female', name: {…}, location: {…}, ...', login: {…}, …}
 4: {gender: 'female', name: {…}, location: {…}, ...', login: {…}, …}
 length: 5
 [[Prototype]]: Array(0)
```

```
/* ch02-05-02-promises.html */

fetch("https://api.randomuser.me/?nat=US&results=10")
  .then(res => res.json())
  .then(json => json.results)
  .then(console.log)
  .catch(console.error)
```

URL로부터 데이터를 받음

response를 json 형태로 변환

객체의 results 속성 반환

console에 데이터 출력

에러시 console 에러 출력

request.response

배열

```
{
  "results":[
    { "gender":"female","name":{...}, ...},"nat":"US" },
    { "gender":"male","name":{...}, ...},"nat":"US" },
    ... ,
    { "gender":"female","name":{...}, ...},"nat":"US" }
  ],
  "info":{
    "seed":"df27615f97661362",
    "results":10,
    "page":1,
    "version":"1.4"
  }
}
```

비동기 자바스크립트: async/await

```
/* ch02-05-03-promises.html */

const getFakeMembers = async () => {
    try {
        let response = await fetch("https://api.randomuser.me/?nat=US&results=10");
        let { results } = await response.json();
        console.log(results);
    }
    catch (error) {
        console.error(error);
    }
};

getFakeMembers();
```

async/await 구문: 앞선 promise 구문이 끝날 때까지 기다림

URL로부터 데이터를 받고 대입

response를 json 형태로 변환
> 객체의 구조분해 할당

```
/* ch02-05-02-promises.html */

fetch("https://api.randomuser.me/?nat=US&results=10")
  .then(res => res.json())
  .then(json => json.results)
  .then(console.log)
  .catch(console.error)
```

원소를 10개 갖는 배열

```
(10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
 0: {gender: 'female', name: {…}, location: {…}, ...', login: {…}, …}
 1: {gender: 'female', name: {…}, location: {…}, ...', login: {…}, …}
 ...
 9: {gender: 'female', name: {…}, location: {…}, ...', login: {…}, …}
 length: 10
 [[Prototype]]: Array(0)
```

클래스

```
/* ch02-06-01-classes.html */

// 이전 방식으로 생성자를 만들고 프로토타입 설정하기
function Vacation(destination, length) {
  this.destination = destination
  this.length = length
}

Vacation.prototype.print = function() {
  console.log(this.destination + "는 " +
              this.length + "일 걸립니다.")
}

var trip = new Vacation("마우이", 7)
trip.print()
```

프로퍼티 정의

메서드 정의

프로토타입을 통해 print 메서드 상속

인스턴스 생성

```
마우이는 7일 걸립니다.
```

```
/* ch02-06-02-classes.html */

// 클래스를 사용해 정의하는 새로운 방식
class Vacation {
  constructor(destination, length) {
    this.destination = destination
    this.length = length
  }
  print() {
    console.log(`${this.destination}는
                ${this.length}일 걸립니다.`)
  }
}

const trip = new Vacation("칠레 산티아고", 9)
trip.print()
```

프로퍼티 정의

메서드 정의

인스턴스 생성

```
칠레 산티아고는 9일 걸립니다.
```

정리: 1. 리액트 소개 & 2. 리액트를 위한 자바스크립트

- React 개념
  - JavaScript 라이브러리
  - UI 구현을 위한 View 담당
- ES2015+
  - 변수와 상수
  - 템플릿 문자열
- ES2015+ (계속)
  - 함수: 선언과 호출, 함수표현식, 호이스팅, default parameters
  - 화살표 함수: 사용법과 this의 참조 범위
  - 구조분해 할당: 변수, 객체
  - 객체 리터럴
  - 스프레드 연산자
  - 비동기 자바스크립트: promise, fetch, async/await
  - 클래스