



스터디 4주차 과제


1. 클래스

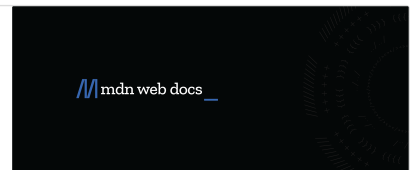
자바스크립트에서 클래스는 ES6에서 추가된 개념으로 기존 Java나 C++의 클래스를 대체한다.

자바스크립트의 클래스는 프로토타입을 이용해 만들어졌다. 다음 문서들을 참고하여 클래스를 나름대로 정리해보자.

Poimaweb
웹 프로그래밍 튜토리얼
 <https://poimaweb.com/es6-class>




Classes - JavaScript | MDN
Class는 객체를 생성하기 위한 템플릿입니다. 클래스는 데이터와 이를 조작하는 코드를 하나로 추상화합니다. 자바스크립트에서 클래스는 프로토타입을 이용해서 만들어졌지만 ES5의 클래스 의미와는 다른 문법과 의미를 가집니다.
 <https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Classes>



클래스
 <https://ko.javascript.info/classes>



 항상 남들에게 설명할 수 있을 정도로 정리하는 것이 제일 중요!

2. 프로미스

프로미스는 자바스크립트 비동기 처리에 사용되는 객체다.

자바스크립트의 비동기 처리란 '특정 코드의 실행이 완료될 때까지 기다리지 않고 다음 코드를 먼저 수행하는 자바스크립트의 특성'을 말한 이러한 프로미스는 주로 서버에서 데이터를 받아와 화면에 표시할 때 사용한다.

프로미스의 개념에 대해 정리하고, 사용법과 사용하는 이유에 대해 정리할것


자바스크립트 비동기 처리와 콜백 함수
(중급) 중급 자바스크립트 개발자가 되기 위한 자바스크립트 비동기 처리와 콜백 함수 이해하기. 콜백 지옥과 해결 방법 등
 <https://joshua1988.github.io/web-development/javascript/javascript-asynchronous-operation/>



3. async / await

async와 await은 프로미스를 좀 더 편리하게 사용하기 위한 문법이다.

사용법에 대해 정리해보고 다음 코드를 async와 await를 쓰는 형태로 바꿔볼 것

 GPT를 사용하지 않고 바꿔보는 것을 권장합니다!
여유가 난다면 에러처리도 한번 해보길 바랍니다!

```
const f1 = () => {  
  return new Promise((res, rej) => {
```

```

    setTimeout(() => {
      res("1번 함수 호출 완료");
    }, 1000);
  });
};

const f2 = (message) => {
  console.log(message)
  return new Promise((res, rej) => {
    setTimeout(() => {
      rej("2번 함수 호출 완료");
    }, 2000);
  });
};

const f3 = (message) => {
  console.log(message)
  return new Promise((res, rej) => {
    setTimeout(() => {
      res("3번 함수 호출 완료");
    }, 3000);
  });
};

f1()
  .then(res => f2(res))
  .then(res => f3(res))
  .then(res => console.log(res))
  .catch(console.log);

```

Quiz

01 - Callback Hell 🔥

제출할 폴더 이름 :	ex01/
제출할 파일 이름 :	recipe_callback.js

성결대학교에서 공부를 하고 있는 **천민우**는 최근 새로운 전자기기를 구입하기 위해 **안양1번가**를 전전하며 알바를 구하기 시작하였다.

그 결과 안양1번가의 명물 파배기 집에 알바로 들어가게 되었다!

민우는 파배기를 만드는 업무에 배정을 받게 되었는데, 파배기를 만들려면 다음 아래와 같은 작업을 진행하게 된다.

해당 작업은 아래와 같은 순서와 시간이 걸리게 된다.

1. 반죽 만들기 - 3초
2. 1차 발효 - 5초
3. 성형 하기 - 4.2초
4. 2차 발효 - 2초
5. 튀기기 - 5초

민우는 위와 같은 작업을 할 수 있는 **js** 코드를 구현하려고 한다, **천민우**를 도와주자!!!!

- 각 작업은 **setTimeout** 함수를 이용하여 구현해야 한다.
- 매 단계별 **console.log**를 이용하여 단계명을 출력해야한다. (이때 **실패하였다는 내용도** 출력을 해주어야 한다.)
- 각 작업은 매번 아래의 함수를 호출하여 확률적으로 실패하도록 설계해야 한다.

```

function randomFail() {
  if (Math.random() < 0.2) throw "제작 실패..!(월급이 삭감되었다 ㅜㅜ)";
}

```

▼ 가이드라인 - 너무 어려워서 감이 안잡힌다면 펼쳐볼 것

```

function randomFail() {
  if (Math.random() < 0.2) throw "제작 실패..!(월급이 삭감되었다 ㅜㅜ)";
}

function makeDough() {
  return new Promise((resolve, reject) => {

```

```

    setTimeout(() => {
      try {
        console.log("반죽 만들기 시작!");
        randomFail();
        resolve();
        console.log("반죽 완성!");
      } catch (error) {
        console.log("반죽 만들기 실패...");
        reject(error);
      }
    }, 3000);
  });
}

function firstFermentation() { /* 위와 비슷한 형식으로 작성 */ }

function shape() { /* 위와 비슷한 형식으로 작성 */ }

function secondFermentation() { /* 위와 비슷한 형식으로 작성 */ }

function fry() { /* 위와 비슷한 형식으로 작성 */ }

// 파배기 제작
async function make() {
  try {
    await makeDough();
    //나머지 함수 실행
    //파배기 제작 완료 메시지 출력
  } catch (error) {
    //에러 메시지 출력
  }
}

make();

```

02 - 스파게티 스토리

제출할 폴더 이름 :	ex02
제출할 파일 이름 :	spaghetti.js

파배기 알바만으로 부족했던 **민우**는 시간을 쪼개 스파게티스토리에서 투잡을 뛰기로 결정했다. 다행히 작업 방식은 파배기랑 비슷해 이번0 스파게티를 만드는 작업과 소요시간은 다음과 같다.

1 A. 면 삶기 - 10초

- 2 B1. 브로콜리 대치기 - 1초
- B2. 마늘과 양파 볶기 - 2초
- B3. 베이컨과 햄 볶기 - 2초
- B4. 소스, 남은 야채 넣고 다같이 볶기 - 3초

3 C. 면까지 넣고 다 같이 볶기 - 3초

다만, 작업을 진행할 때 순차적으로 진행하면 만드는데 시간이 오래 걸릴 것 같아, **작업A와 B를 동시에** 하면서 A와 B 작업이 **모두 끝났을 때**

▼ 두개의 프로미스를 한번에 실행하는 법?

```

function firstTask() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("첫 번째 작업 완료");
      resolve(1);
    }, 2000);
  });
}

```

```

function secondTask() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("두 번째 작업 완료");
      resolve(2);
    }, 3000);
  });
}

async function runTasks() {
  try {
    const results = await Promise.all([firstTask(), secondTask()]);
    console.log("모든 작업 완료, 결과: ", results);
  } catch (error) {
    console.log("작업 실패: ", error);
  }
}

runTasks();

```

각각의 프로미스를 배열에 넣고 Promise.all 함수로 실행시켜주면 된다.

▼ 가이드라인

```

function boilNoodles() {
  console.log("면 삶기 시작");
  //면 삶기 작업 Promise를 리턴할 것.
}

function processB1() { /* 위와 비슷한 형태로 작성 */ }
function processB2() { /* ... */ }
function processB3() { /* ... */ }
function processB4() { /* ... */ }

async function processB() {
  //B프로세스 4개를 하나씩 처리
  console.log("B 작업 모두 완료");
}

function mixAllTogether() {
  console.log("면까지 넣고 다 같이 볶기 시작");
  //프로미스 리턴
}

async function makeSpaghetti() {
  //각각의 프로미스를 한번에 실행
  await mixAllTogether();
}

makeSpaghetti();

```