2022/5/23

1. Promise

   - 콜백 안에 콜백 안에 콜백 ..... 인 콜백 지옥을 벗어나기 위해 고안된 문법

   - 콜백 함수를 다시 부르겠다고 약속하는 것

   - pending(대기 상태)에서 내용이 해결될 시와 거부될 시로 나뉘어서 처리됨

   - pending(대기 상태) -> resolve(해결) -> fulfilled(성공)의 과정(resolve(코드...))을 거치면 .then() 의 내용이 실행됨

   - pending(대기 상태) -> reject(거부) -> rejected(실패)의 과정(reject(코드...))을 거치면 .catch() 의 내용이 실행됨

   - resolve(), reject() 아무 것도 지정해 주지 않으면 영원히 pending 상태가 되어 버림

   - .finally()는 resolve이든 reject이든 상관없이 실행됨 (보통 프로미스의 가장 마지막에 연결해 줌)

   - promise chaining은 하나의 promise에 계속 then을 잇는 것 (이미 리턴된 promise에 then을 잇는 것은 promise chaining이 아님) -> promise chaining을 통해서 then의 결과를 또 다른 프로미스로 처리할 수 있음

2. async & await

- 분기를 나누기가 어려움, then이 너무 많아져서 가독성이 떨어짐 등 promise의 단점을 보완하기 위해 고안된 문법

- 콜백 함수 앞에 async를 붙이고, 실행이 끝날 때까지 기다릴 프로미스 앞에 await을 붙여줌 (await은 프로미스 앞에만 붙일 수 있고, 보통 함수 앞에 붙이면 아무런 효과가 없음)

- 최근 문법에서는 await 단독으로 사용하기도 하고, await이 async의 밖에도 존재할 수 있음

- try-catch문으로 에러를 잡음 (try{ await () =>
{...} }catch(error){ ... })

- await function1; await function2; ... 은 먼저 온 게 처리될 때까지 기다리므로 동기 처리, Promise.all(function1, function2, ...) 은 동시에 처리하므로 비동기 처리