Promise

> 💡 **Promise 간단 개념:**
>
> **비동기 작업**을 다루는 **'객체'**이다.
>
> **아직 결과를 모르는 비동기 작업의 결과를 위한 약속**입니다.

Promise는 3개의 상태를 가지고 있습니다.

- Pending(대기중) → 아직 작업이 완료되지 않은 상태
- Fulfilled(성공) → 작업이 성공된 상태(resolve() 호출)
- Rejected(실패) → 작업이 실패된 상태(reject() 호출)

여기서 중요한 점은 Promise는 각각 3개의 상태를 가지며
성공, 실패라는 2개의 결과값을 가지고 resolve() 혹은 reject()를 반환하여 값을 전달합니다.

여기서 비동기 작업에서 콜백 함수가 아닌 Promise가 자주 쓰이는 이유가 나오게 되는데

1. **콜백함수는 return을 통해서 값을 여러개 혹은 직접 넘기기 어렵다**
    - 일반 함수는 return으로 한 가지 값만 반환할 수 있지만(객체 제외), Promise는 성공 or 실패에 따른 다양한 값을 구조적으로 여러개를 전달할 수 있다.
    - 콜백함수를 여러개 쓰게되면 코드의 가독성이 복잡해지고 콜백 지옥에 빠지게 될 수 있기 때문에 후에 문제가 생길 수 있다.
2. **성공(resolve), 실패(reject)처럼 각각의 상황에서 처리가 명확하게 가능하다.**
3. **.then(), .catch(), .finally() 체이닝으로 가독성이 좋고 유지보수가 쉬워서 쉬운 코드 작성이 가능하다.**

.then(), .catch(), .finally()

위에서 설명한 Promise의 메서드로 각각 순서대로 성공, 실패, 항상시 실행되는 콜백을 등록하게 된다.
예를 들어서, Promise가 성공하게 되면 .then()에 전달한 함수가 호출되고, 그 결과를 다음 .then()또 전달할 수 있는 기능도 있다.

특징

- 여러 개 연속으로 붙일 수 있다 → **체이닝 가능**
- 각 단계에서 처리 후 값을 넘겨서 다음 단계로 전달이 가능하다

1. then()
    - 성공시 실행되는 함수를 이행
    - Promise가 성공되면 .then()에 전달된 함수가 호출됩니다
      (그 결과를 다음 .then()으로 전달도 가능하다(체이닝))
2. .catch()
    - 실패시 실행되는 함수를 이행
    - Promise가 실패되면 .then()에 전달된 함수를 호출합니다
      (.then() 중간에 발생된 에러도 .catch()로 잡을 수 있다)
3. .finally()
    - 성공, 실패 관계없이 항상 함수를 이행
    - 주로 마지막에 “작업 종료”를 호출

Async, Await

Promise를 더 간결하고 직관적으로 읽거나 사용할 수 있게 해주는 문법

특징

- async는 항상 Promise를 반환한다
- await은 항상 Promise를 기다리고 있다
  (Promise가 실행되고 성공과 실패를 판가름 할 때 까지 대기)
- await은 async를 선언한 함수 안에서만 사용이 가능하다

```
async function myFunc() {
  return "Hello";
}

myFunc().then(res => console.log(res)); // 출력: Hello
```

위와 같은 코드를 봤을 때
Promise를 안썼음에도 불구하고 .then()함수를 사용
함수 안에서 return한 값은 자동으로 resolve()가 된다

```
async function delayLog() {
  console.log("1초 대기 중...");
  await new Promise(resolve => setTimeout(resolve, 1000));
  console.log("1초 후 실행됨!");
}
```

위와 같은 코드를 봤을 때
await은 Promise가 처리될 때까지 다음 줄을 기다린다
즉, 코드 실행을 잠시 멈췄다가 결과가 오면 동기적으로 실행

BUT!!

await을 통해서 코드 실행을 멈췄다고 해서 전체 코드가 멈추는 것이 아니다
await안에 있는 함수의 코드 실행이 멈추는 것이고 전체 코드는 이미 실행되고 있다

```
async function asyncFunc(){
    await new Promise(resolve => setTimeout(resolve, 1000));
    console.log("A");
}

asyncFunc();
console.log("B");
```

이런 코드가 있었을 때 동기적으로 실행된다면

1. asyncFunc()함수 실행

2. setTimeout으로 1초를 기다린 뒤에 “A”출력
3. 함수 실행 아래 코드인 “B”출력
4. 최종적으로 “A”, “B”가 출력되겠지만

await을 사용하게 된다면

1. asyncFunc()함수 실행
2. setTimeout으로 1초를 기다리게 됨
3. 그 사이에 “B”출력
4. 1초뒤에 “A”출력
5. 최종적으로 “B”, “A”가 출력됨