



# Promise

then · catch / async · await

정혜민



Promise

이런 then 그러 catch



Promise

비동기적  
처리

async

프로미스  
체이닝

reject

then

await

catch

# CONTENTS

## 01 | Promise 개념

동기적 처리와 비동기적 처리  
Promise란?  
Promise의 상태

## 02 | Promise 사용법

Promise 객체의 생성  
Promise의 후속 처리 메서드  
Promise의 예외 처리  
Promise의 체이닝

## 03 | Async · Await 맛보기

async · await 개념 및 사용법  
Promise와 async/await의 차이점

## 추가 | 교안 코드 분석

01

---

## Promise 개념

```
console.log(1);  
console.log(2);  
console.log(3);|
```

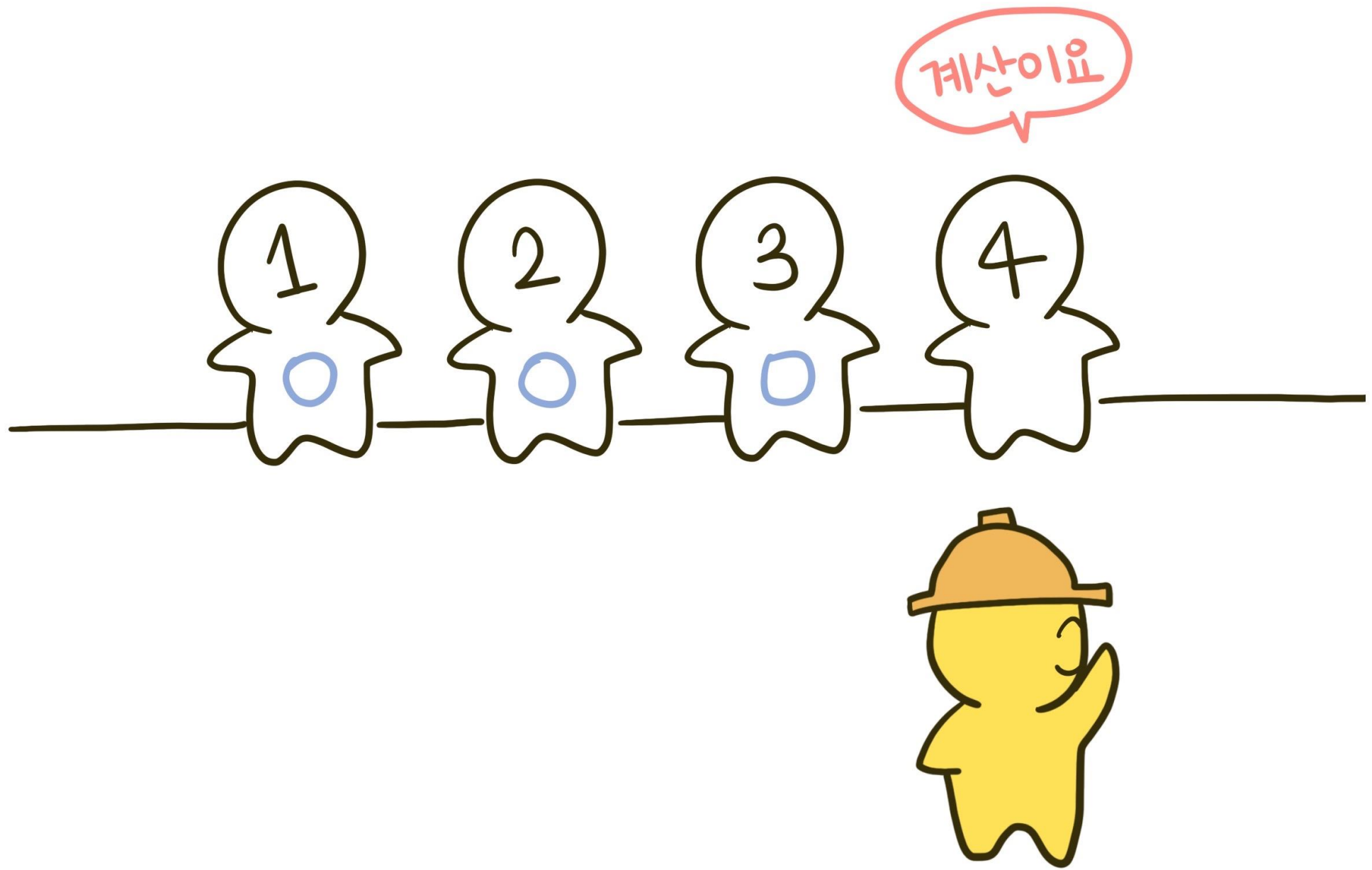
1 / 2 / 3

동기적 처리

```
console.log(1);  
setTimeout(function(){console.log(2)}, 1000);  
console.log(3);
```

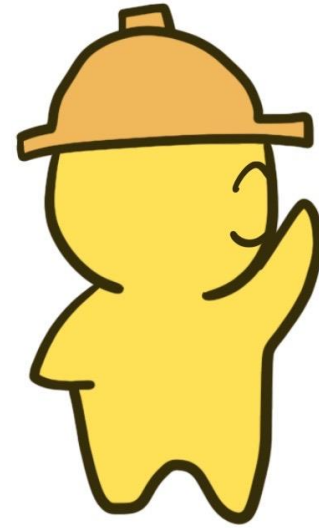
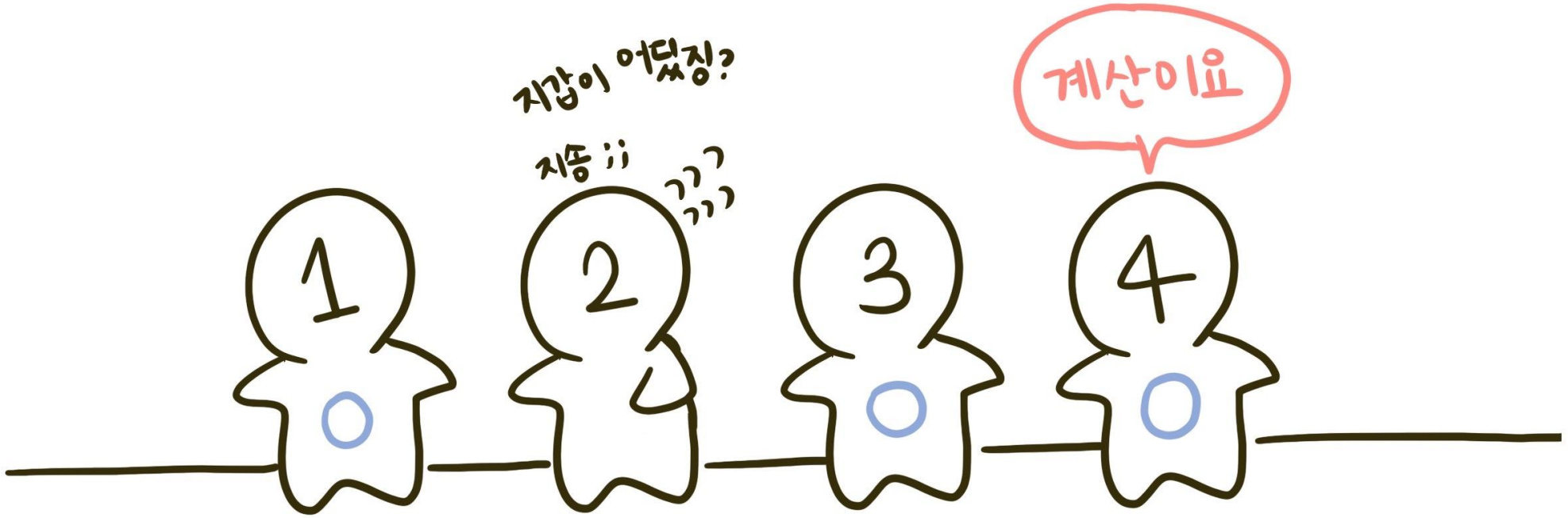
1 / 3 / 2

비동기적 처리



# 비동기적 처리





# Promise

### 비동기처리

자바스크립트의 비동기 처리란 '특정 코드의 실행이 완료될 때까지 기다리지 않고 다음 코드를 먼저 수행'하는 자바스크립트의 특성을 의미

### Promise

Promise는 자바스크립트 **비동기 처리에 사용되는 객체**  
비동기 처리를 위한 콜백 패턴의 콜백 헬, 에러처리 곤란 문제를 극복하기 위해 도입

비동기적 처리 / 비동기 처리 콜백함수에 대한 개념 참고

<https://joshua1988.github.io/web-development/javascript/javascript-asynchronous-operation/>

**pending · fulfilled · rejected**

## Promise의 상태

프로미스	
status	<b>pending</b>
result	<b>undefined</b>

resolve(value)

비동기 처리 수행

reject(error)

프로미스	
status	<b>"fulfilled"</b>
result	<b>value</b>

프로미스	
status	<b>"rejected"</b>
result	<b>error</b>

## Promise의 상태

프로미스의 상태 정보	의미	상태 변경 조건
<b>pending</b>	비동기 처리가 아직 <u>수행되지 않은 상태</u>	프로미스가 생성된 직후 기본 상태
<b>fulfilled</b>	비동기 처리가 <u>수행된 상태(성공)</u>	resolve 함수 호출
<b>rejected</b>	비동기 처리가 <u>수행된 상태(실패)</u>	reject 함수 호출

02

---

## Promise 사용법

## Promise 사용법

```
searchUser(userName, password) {  
  return new Promise((resolve, reject) => {  
    const xhr = new XMLHttpRequest();  
    xhr.open('GET', 'users.json');  
    xhr.onreadystatechange = () => {  
      if (xhr.readyState === 4 && xhr.status === 200) {  
        const result = JSON.parse(xhr.responseText).user.find((item) => {  
          return item.userName === userName && item.password === password  
        });  
        if (result) {  
          resolve(userName);  
          // onSuccess(userName);  
        } else {  
          reject('user not found');  
          // onError(new Error('user not found'));  
        }  
      }  
    }  
  })  
  xhr.send();  
})  
}
```



## Promise 사용법

```
searchUser(userName, password) {  
  return new Promise((resolve, reject) => {  
    const xhr = new XMLHttpRequest();  
    xhr.open('GET', 'users.json');  
    xhr.onreadystatechange = () => {  
      if (xhr.readyState === 4 && xhr.status === 200) {  
        const result = JSON.parse(xhr.responseText).user.find((item) => {  
          return item.userName === userName && item.password === password  
        });  
        if (result) {  
          resolve(userName);  
          // onSuccess(userName);  
        } else {  
          reject('user not found');  
          // onError(new Error('user not found'));  
        }  
      }  
    }  
    xhr.send();  
  })  
}
```

???

## Promise 사용법

```
searchUser(userName, password) {  
    return new Promise((resolve, reject) => {  
        const xhr = new XMLHttpRequest();  
        xhr.open('GET', 'users.json');  
        xhr.onreadystatechange = () => {  
            if (xhr.readyState === 4 && xhr.status === 200) {  
                const result = JSON.parse(xhr.responseText).user.find((item) => {  
                    return item.userName === userName && item.password === password  
                });  
                if (result) {  
                    resolve(userName);  
                    // onSuccess(userName);  
                } else {  
                    reject('user not found');  
                    // onError(new Error('user not found'));  
                }  
            }  
        }  
        xhr.send();  
    })  
}
```

```
const 프로미스 = new Promise((resolve, reject) => {  
  if (/*비동기 처리 성공*/){  
    resolve('result');  
  } else {/*비동기 처리 실패*/  
    reject('failure reason');  
  }  
});
```

## Promise 객체의 생성

```
const 프로미스 = new Promise((resolve, reject) => {  
  if (/*비동기 처리 성공*/){  
    resolve('result');  
  } else { /*비동기 처리 실패*/  
    reject('failure reason');  
  }  
});
```

new Promise() 메서드를 호출하면 대기(pending) 상태

## Promise 객체의 생성

```
const 프로미스 = new Promise((resolve, reject) => {  
  if (/*비동기 처리 성공*/){  
    resolve('result');  
  } else { /*비동기 처리 실패*/  
    reject('failure reason');  
  }  
});
```

**콜백 함수**를  
인수로 전달받음

Promise의 생성자 함수는 비동기 처리를 수행할 때 콜백 함수를 인수로 전달받음

## Promise 객체의 생성

```
const 프로미스 = new Promise((resolve, reject) => {  
  if (/*비동기 처리 성공*/){  
    resolve('result');  
  } else {/*비동기 처리 실패*/  
    reject('failure reason');  
  }  
})
```

**resolve, reject 함수를**  
인수로 전달 받음

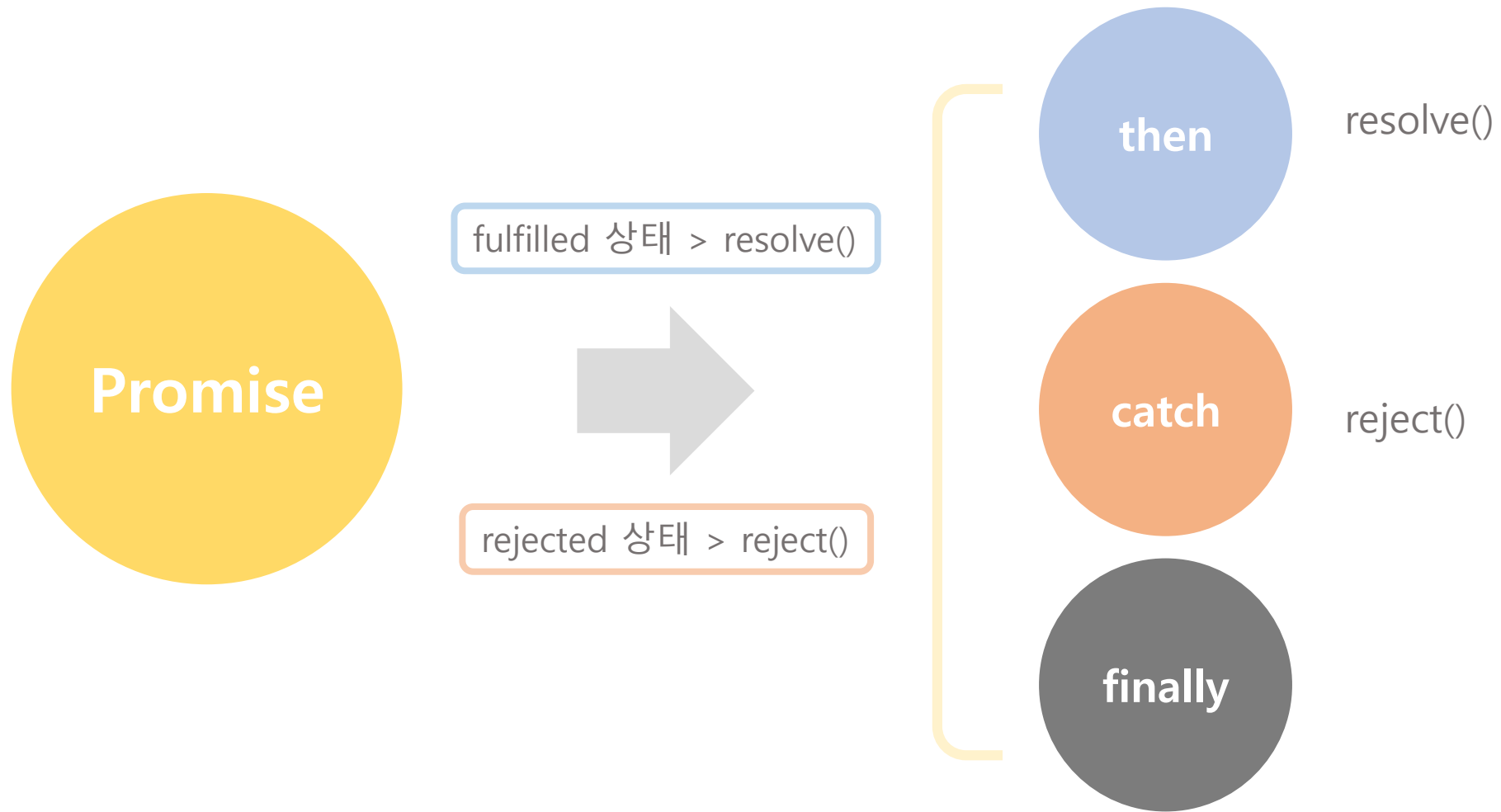
콜백 함수는 resolve와 reject 함수를 인수로 전달 받음

## Promise 객체의 생성

```
searchUser(userName, password) {  
  return new Promise((resolve, reject) => {  
    const xhr = new XMLHttpRequest();  
    xhr.open('GET', 'users.json');  
    xhr.onreadystatechange = () => {  
      if (xhr.readyState === 4 && xhr.status === 200) {  
        const result = JSON.parse(xhr.responseText).user.find((item) => {  
          return item.userName === userName && item.password === password  
        });  
        if (result) {  
          resolve(userName);  
          // onSuccess(userName);  
        } else {  
          reject('user not found');  
          // onError(new Error('user not found'));  
        }  
      }  
    }  
    xhr.send();  
  })  
}
```

**then · catch · finally**



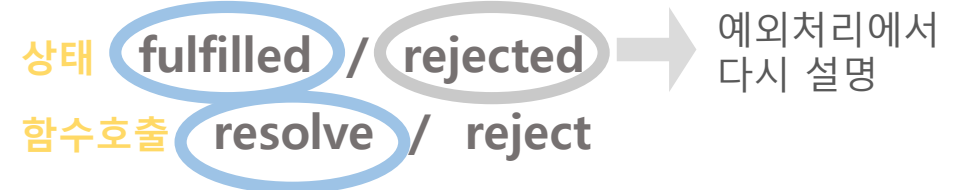


프로미스의 비동기 처리 상태가 변화하면 **후속 처리 메서드**에 인수로 전달한 콜백 함수가 선택적으로 호출됨

```
프로미스.then(function () {  
}).catch(function () {  
}).finally(function () {  
})
```

### Promise.prototype.then

프로미스가 ?? 경우 실행할 코드



```
프로미스.then(function () {  
}).catch(function () {  
}).finally(function () {  
})
```

### Promise.prototype.catch

프로미스가 ?? 경우 실행할 코드



상태 fulfilled / **rejected**

함수호출 resolve / **reject**

```
프로미스.then(function () {  
}).catch(function () {  
}).finally(function () {  
})
```

### Promise.prototype.finally

프로미스가 ?? 경우 실행할 코드



상태 fulfilled / rejected

## Promise의 후속 처리 메서드

```
const 프로미스 = new Promise((resolve, reject) => {  
  if (/*비동기 처리 성공*/){  
    resolve('result');  
  } else {/*비동기 처리 실패*/  
    reject('failure reason');  
  }  
});
```

resolve



```
프로미스.then(function () {  
  console.log('성공했음')  
}).catch(function () {  
  console.log('실패했음')  
})
```

then 안에 있는 것 실행

비동기 처리가 성공했을 때  
호출되는 성공 처리 콜백 함수

- ▶ 성공했음
- ▶ Promise

## Promise의 후속 처리 메서드

```
const 프로미스 = new Promise((resolve, reject) => {  
  if (/*비동기 처리 성공*/){  
    resolve('result');  
  } else {/*비동기 처리 실패*/  
    reject('failure reason');  
  }  
});
```

```
프로미스.then(function () {  
  console.log('성공했음')  
}).catch(function () {  
  console.log('실패했음')  
});
```

reject



catch 안에 있는 것 실행

비동기 처리가 실패했을 때  
호출되는 실패 처리 콜백 함수

- ▶ 실패했음
- ▶ Promise

## Promise의 후속 처리 메서드

```
const 프로미스 = new Promise((성공, 실패) => {  
  if (/*비동기 처리 성공*/){  
    성공('result');  
  } else {/*비동기 처리 실패*/  
    실패('failure reason');  
  }  
});  
  
프로미스.then((성공결과) => {  
  console.log(성공결과)  
}).catch((실패결과) => {  
  console.log(실패결과)  
})
```

성공() > result

(성공결과) 파라미터에 성공의 결과 값인 result를 인자로 받음

Then의 첫 번째 콜백 함수의 인자 값으로 성공의 결과 값을 받음

- ▶ result
- ▶ Promise

## Promise의 후속 처리 메서드

```
const 프로미스 = new Promise(function (성공, 실패) {  
  const 어려운연산 = 1 + 1;  
  성공(어려운연산);  
});
```

1+1 연산이 끝나면  
성공(어려운연산) 판정을  
내려주세요

```
프로미스.then(function (성공결과) {  
  console.log(성공결과)  
}).catch(function () {  
  console.log('실패했어요')  
})
```

성공()의 결과 값이 then 콜백함수의  
성공결과에 송~하고 들어감

- ▶ 2
- ▶ Promise



## 프로미스의 에러 처리

then() 의 두 번째 인자로 에러를 처리하는 방법

```
프로미스.then(function(){}, function(err){console.log(err)});
```

catch() 이용

```
프로미스.then()  
.catch(function(err){  
  console.log(err)});
```

추천

프로미스

```
.then(()=>{  
  })  
.then(()=>{  
  })  
.then(()=>{  
  })  
.catch(()=>{  
  })
```



## Promise의 체이닝

```
const 프로미스 = new Promise((resolve, reject) => {  
  const 어려운연산=1+1;  
  resolve(어려운연산);  
});  
  
프로미스.then(function (결과1) {  
  console.log(결과1)  
  return 결과1 + 10;  
}).then(function (결과2) {  
  console.log(결과2)  
  return 결과2 + 10;  
}).then(function (결과3) {  
  console.log(결과3)  
});
```

성공()  
> 2

결과1 = 2  
→ 2  
→ 12

결과2 = 12  
→ 12  
→ 22

결과3 = 22  
→ 22

- ▶ 2
- ▶ 12
- ▶ 22
- ▶ Promise

**Promise 객체를 사용하면 비동기적인 작업을 동기적인 함수처럼 처리할 수 있다?**

## Promise Quiz

```
> const 프로미스 = new Promise(function(resolve, reject){  
  setTimeout(function() {  
    resolve(1);  
  }, 2000);  
})
```

```
프로미스.then(function () {  
  console.log('성공')  
});
```

```
console.log(1);  
console.log(프로미스);  
console.log(3);
```

순서???

## Promise Quiz


```
> const 프로미스 = new Promise(function(resolve, reject){
  setTimeout(function() {
    resolve(1);
  }, 2000);
})

프로미스.then(function () {
  console.log('성공')
});

console.log(1);
console.log(프로미스);
console.log(3);
```

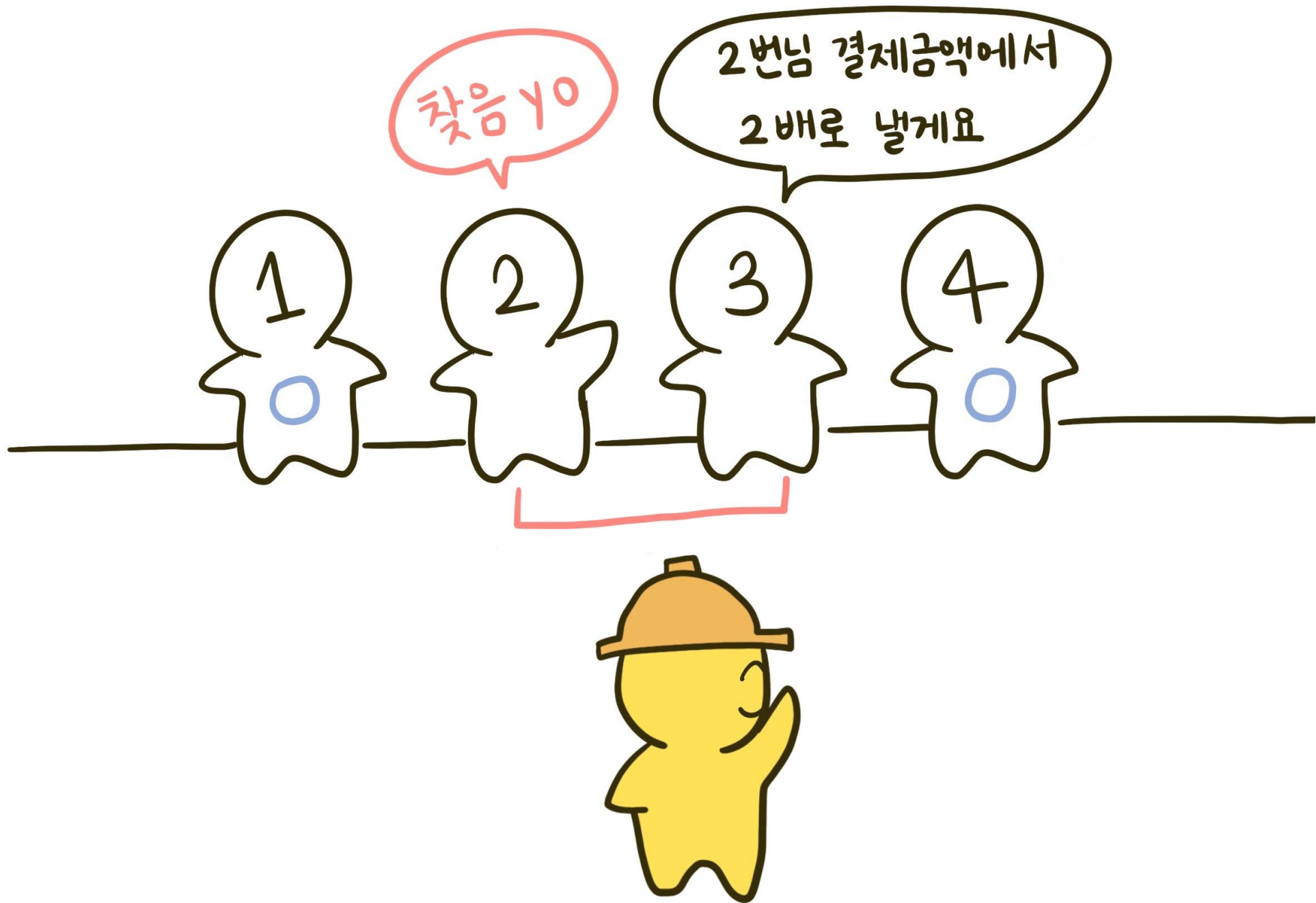
```
> const 프로미스 = new Promise(function(resolve, reject){
  setTimeout(function() {
    resolve(1);
  }, 2000);
})

프로미스.then(function () {
  console.log('성공')
});
```



```
console.log(1);
console.log(프로미스);
console.log(3);
```

1	<a href="#">VM1255:12</a>
▶ Promise {<pending>}	<a href="#">VM1255:13</a>
3	<a href="#">VM1255:14</a>
undefined	
성공	<a href="#">VM1255:8</a>



03

---

**Async · Await 맛보기**



## Async

함수 선언 앞에 사용

```
async function 더하기() {  
  let 프로미스 = new Promise(function (성공, 실패) {  
    let 어려운연산 = 1 + 1;  
    성공(어려운연산);  
  });  
  
  프로미스.then(function (결과) {  
    console.log(결과)  
  });  
}  
더하기();
```

▶ 2  
▶ Promise

Async 를 function 앞에 붙이면 함수가 **Promise 역할 가능**  
함수 실행 후에 Promise 객체 남음

## Async

함수 선언 앞에 사용

```
async function 더하기() {  
  let 프로미스 = new Promise(function (성공, 실패) {  
    let 어려운연산 = 1 + 1;  
    성공(어려운연산);  
  });  
  
  let 결과 = await 프로미스;  
  console.log(결과);  
}  
더하기();
```

## Await

then 대신 사용가능

▶ 2  
▶ Promise

Async function 안에서 사용하는  
Await

프로미스 해결까지 기다림

## Async

함수 선언 앞에 사용

```
async function 더하기() {  
  let 프로미스 = new Promise(function (성공, 실패) {  
    let 어려운연산 = 1 + 1;  
    성공(어려운연산);  
  });  
  
  let 결과 = await 프로미스;  
  console.log(결과);  
}  
더하기();
```

Try...catch 추가로 공부해주세요

await  
then 대신 사용가능

▶ 2  
▶ Promise

Async function 안에서 사용하는  
Await

프로미스 해결까지 기다림

## async

Async 를 function 앞에 붙이면 함수가  
**Promise 역할 가능**  
함수 실행 후에 Promise 객체 남음

## await

Then 대신 사용 가능하며  
async function 안에서 사용

**프로미스 해결까지 기다림**

추가

# 교안 코드 분석

▶ 원하시는 분이 있다면 언제든지 열려있습니다!

**Q & A**

# Reference

## | 서적

이웅모, 『모던 자바스크립트 Deep Dive』, 위키북스

## | 인터넷자료

captain pangyo, "자바스크립트 Promise 쉽게 이해하기",  
2018, <https://joshua1988.github.io/web-development/javascript/promise-for-beginners/>

## | 기타

코딩애플

멋쟁이사자처럼 5기 교안,  
<https://likelion.notion.site/14-AJAX-438d88e05e7b4bd1b444899c34ffc4e7>

감사합니다