06.06

1. 실습

1.1 동기 vs 비동기

자바스크립트 엔진은 단 하나의 실행 컨텍스트 스택을 갖습니다.

⇒ 동시에 2개 이상의 함수를 동시에 실행할 수 없습니다.

1.1.1 동기

현재 실행 중인 태스크가 종료할 때까지 다음에 실행될 태스크가 대기하는 방식.

⇒ 태스크를 순서대로 처리한다는 장점이있지만, 앞의 태스크가 종료할 때까지 이후 태스크들이 블로킹되는 특징이 있음.

```
incrementSync: function () {
    const currentTime = Date.now();

    while (true) {
      const now = Date.now();
      if (now - currentTime > 3000) break;
    }

    this.count++;
  },
```

1.1.2 비동기

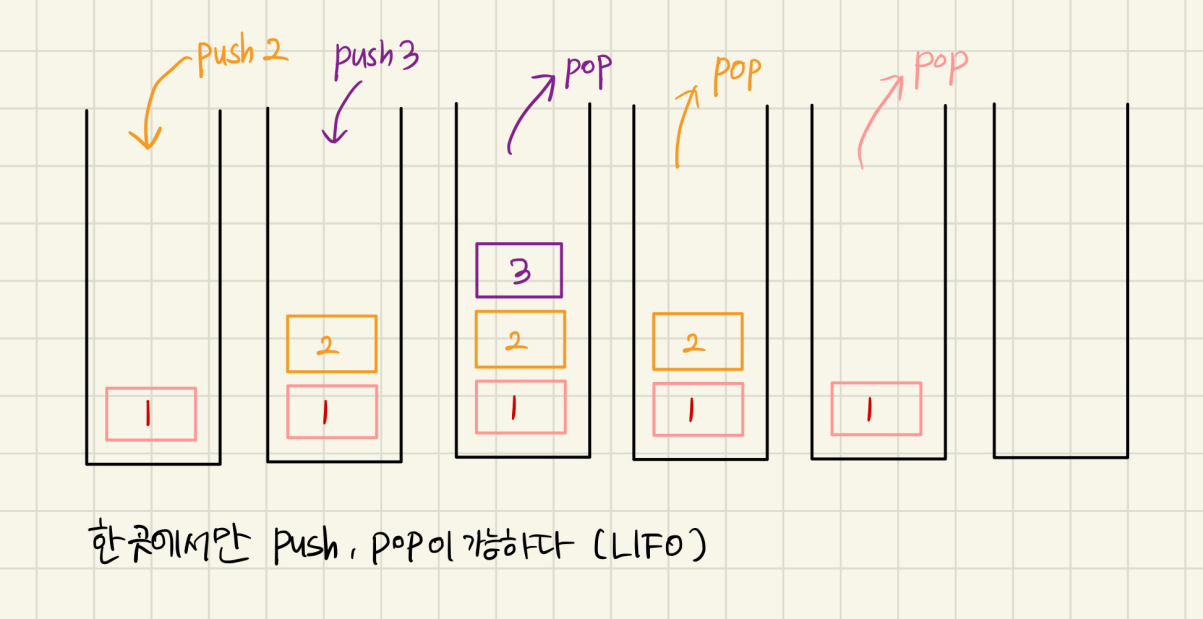
현재 실행 중인 태스크가 종료되지 않은 상태라 해도 다음 태스크를 곧바로 실행하는 방식

주로 콜백 패턴을 사용합니다.

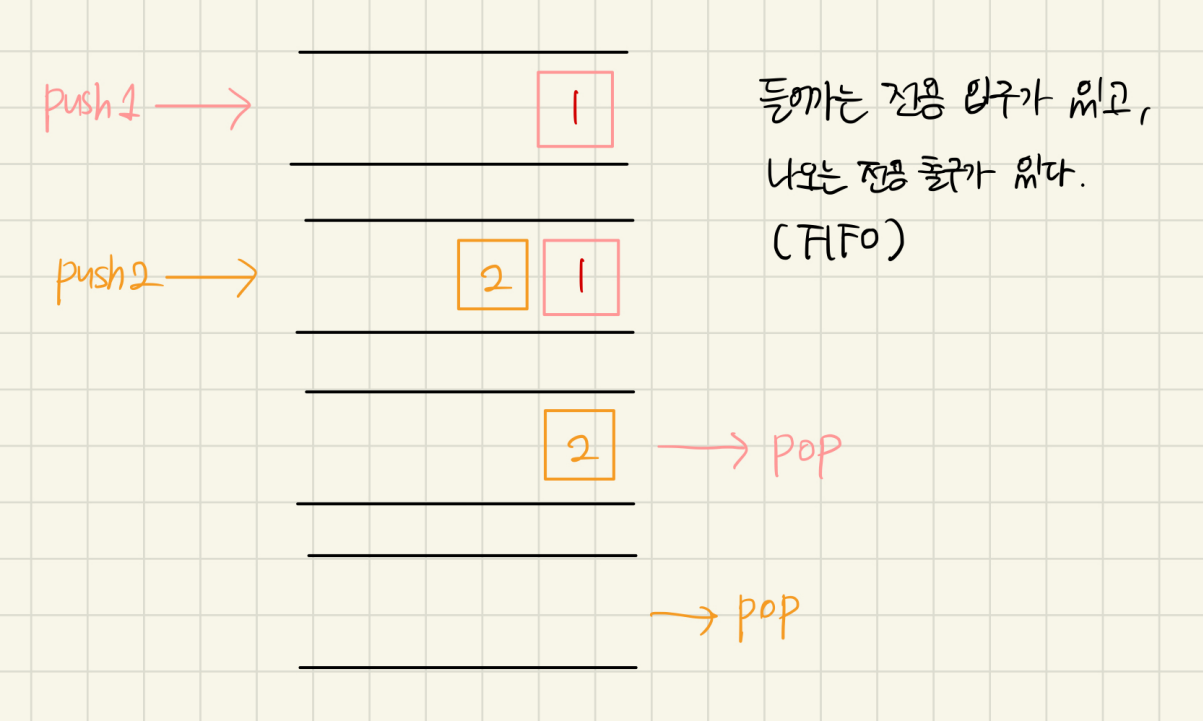
```
 incrementAsync: function (callback) {
    setTimeout(() => {
      this.count++;
      callback();
    }, 3000);
  },
```

1.2 스택과 큐

1.2.1 스택 LIFO(Last In Frist Out)



1.2.2 큐 FIFO (First In First Out)



1.3 실행 컨텍스트 키워드

1.3.1 콜 스택

소스코드 평가 과정에서 생성된 실행 컨텍스트가 추가되고 제거되는 스택 자료구조인 실행 컨텍스트 스택

1.3.2 힙

객체가 저장되는 메모리 공간, 콜 스택의 요소인 실행 컨텍스트는 힙에 저장된 객체를 참조합니다.

1.3.3 태스크 큐

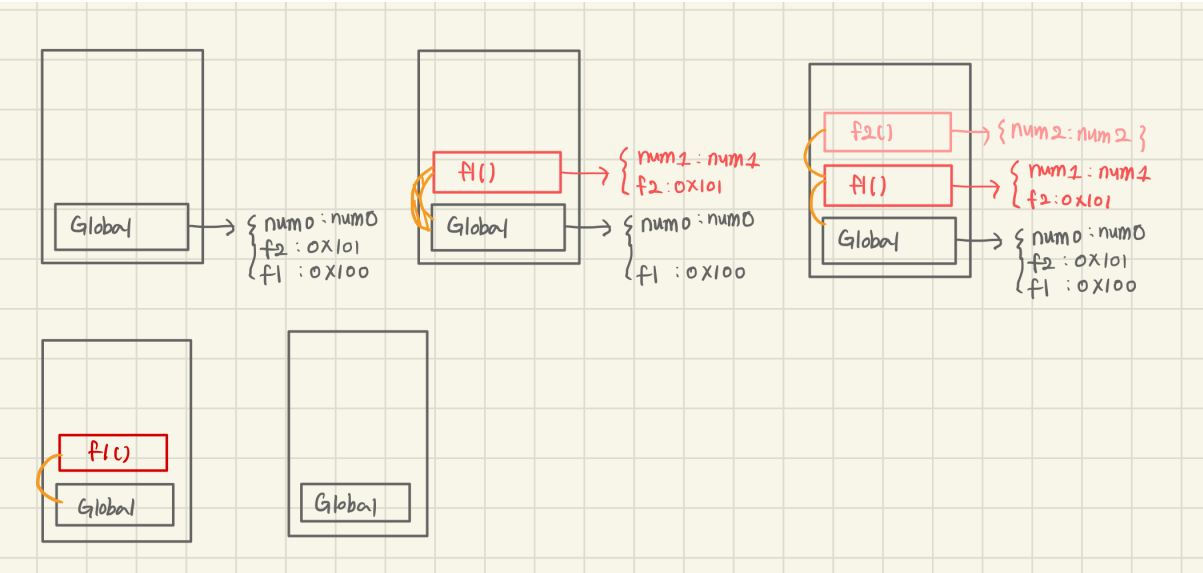
비동기 함수의 콜백 함수 또는 이벤트 핸들러가 일시적으로 보관되는 영역입니다.

1.3.4 이벤트 루프

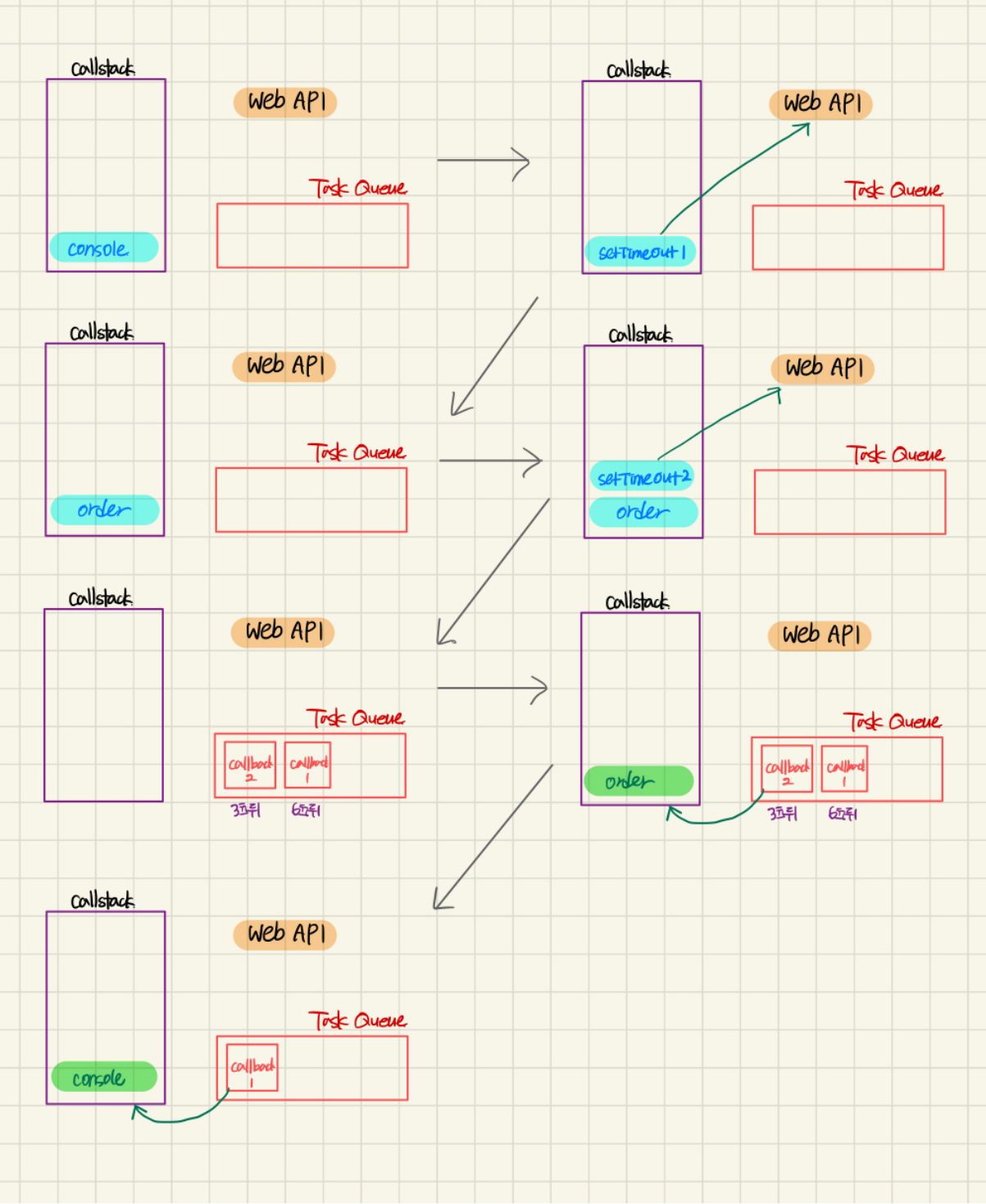
콜 스택에 현재 실행 중인 실행 컨텍스트가 있는지, 태스크 큐에 대기 중인 함수가 있는지 반복해서 확인합니다.

만약 콜 스택이 비어있고, 태스크 큐에 대기 중인 함수가 있다면 이벤트 루프는 순차적으로 태스크 큐에 대기 중인 함수를 콜 스택으로 이동시킵니다.

1.4 동기 실행 컨텍스트



1.5 비동기 실행 컨텍스트



1.6 Quiz (한 번 혼자 생각해보세요!)

1.6.1 출력 순서가 어떻게 될까요?

```
console.log(1);
console.log(2);
setTimeout(() => {
  console.log(3);
},3000)
console.log(4);
```

1.6.2 출력 순서가 어떻게 될까요?

```
console.log(1);
console.log(2);
setTimeout(() => {
  console.log(3);
},0)
console.log(4);
```

1.6.3 출력 순서가 어떻게 될까요?

```
function a() {
  setTimeout(() => {
    console.log('1');
  },0);
  console.log('2');
}

setTimeout(() => {
  console.log('3');
  a();
},0);

setTimeout(() => {
  a();
  console.log('4')
},0);


//실행 컨텍스트
//태스크 큐
//백그라운드
//콘솔
```

1.7 이전 시간에 작성한 타이머 코드

```
const Counter = {
  count: 0,

  getCount: function () {
    return this.count;
  },

  resetCount: function () {
    this.count = 0;
  },

  incrementSync: function () {
    const currentTime = Date.now();

    while (true) {
      const now = Date.now();
      if (now - currentTime > 3000) break;
    }

    this.count++;
  },

  incrementAsync: function (callback) {
    setTimeout(() => {
      this.count++;
      callback();
    }, 3000);
  },
};

export default Counter;
```

1.8 실습 풀이

▼ 동기 비동기 타이머 다르게 구현해보기

```
let count = 0;

function getCount() {
  return count;
}

function resetCount() {
  count = 0;
}

function incrementSync() {
  const currentTime = Date.now();
  while (true) {
    if (Date.now() - currentTime > 3000) break;
  }
```

```
  count++;
}

function incrementAsync(callback) {
  setTimeout(() => {
    count++;
    callback();
  }, 3000);
}

module.exports = { getCount, resetCount, incrementSync, incrementAsync };
```

2. 실습

2.1 실습 풀이

▼ 콜백함수 구현하기

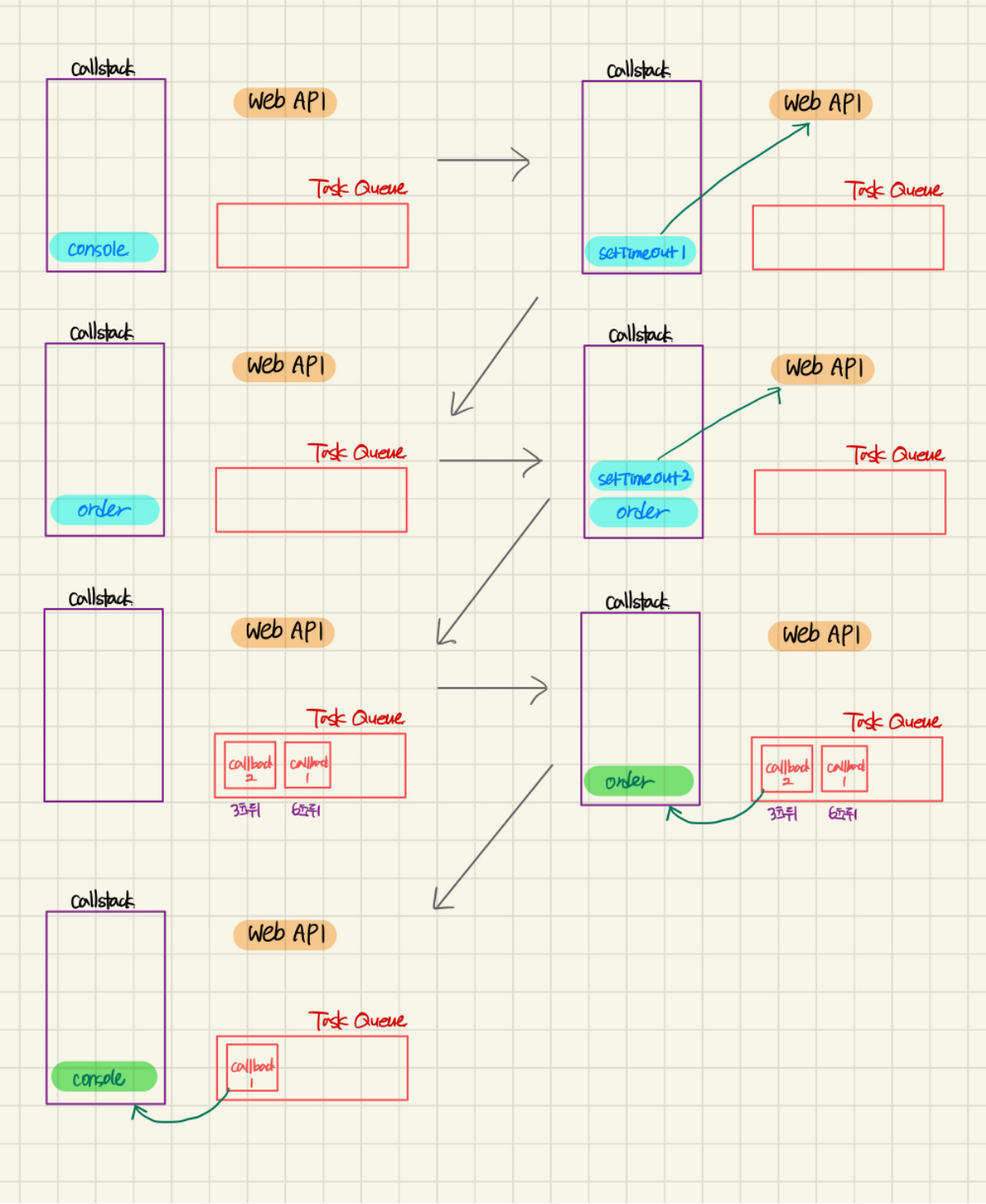
```
//1. 지시사항 1에 따라 아래에 코드를 작성하세요.
console.log('아메리카노 나왔습니다.');
// 아래 함수를 이용해 코드를 작성하세요.

setTimeout(() => {
  console.log('카라멜 프라푸치노 나왔습니다.');
}, 6000);

//2. 지시사항 2에 따라 아래에 코드를 작성하세요.
function order(callback) {
  setTimeout(() => {
    console.log('카라멜 프라푸치노 나왔습니다.');
    callback();
  }, 3000);
  // 이 곳에 코드를 작성하세요.
}

order(function () {
  console.log('아메리카노 나왔습니다.');
});
```

2.2 실습 문제 그림으로 보기


Callstack
Web API
Task Queue
console
setTimeout 1
order
setTimeout 2
callback 2
callback 1
3번째
5번째

3. 실습

3.1 콜백 지옥

```
setTimeout(() => {
    console.log("1번");
    setTimeout(() => {
        console.log("2번");
        setTimeout(() => {
            console.log("3번");
        }, 1000);
    }, 1000);
}, 1000);
```

3.1.1 프로미스 사용

```
function log(n){
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(`${n}번`)
    },1000);
  })
}

log(1).then((result) => {
  console.log(result);
  return log(2);
}).then((result) => {
  console.log(result);
  return log(3);
}).then((result) => {
  console.log(result);
})
```

3.2 프로미스 구조

Promise 생성자 함수는 비동기 처리를 수행할 콜백 함수를 인수로 받습니다.

콜백함수는 `resolve`, `reject` 함수를 인수로 받습니다.

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

```
  }
})
```

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

`fullfilled`, `rejected`상태를 `settled` 상태라고합니다.

프로미스의 상태는 resolve, reject 함수를 호출하는 것으로 결정됩니다.

3.3 then/catch/finally

3.3.1 then

메서드는 두 개의 콜백 함수를 인수로 전달 받는다.

- fulfilled 상태 (resolve 함수 호출 상태)
- rejected 상태 (reject 함수가 호출된 상태)

```
//function example() {
//    return new Promise((_, reject) => reject(new Error("rejected")));
//}

function example() {
    return new Promise((resolve) => resolve("fulfilled"));
}

example().then(
    (value) => console.log(value),
    (error) => console.error(error)
);
```

3.3.2 catch

catch 메서드의 콜백 함수는 프로미스가 rejected 상태인 경우만 호출한다.

```
function example() {
    return new Promise((_, reject) => reject(new Error("rejected")));
```

```
}

example().catch((error) => console.error(error));
```

3.3.3 finally

finally 메서드의 콜백 함수는 프로미스의 fulfilled, rejected와 상관없이 무조건 한 번 호출된다.

```
function example() {
    return new Promise((_, reject) => reject(new Error("rejected")));
}

example().finally(() => console.log('finally'))
```

3.3.4 프로미스 체이닝

`then`, `catch`, `finally` 메서드는 프로미스를 반환하므로 연속적으로 호출할 수 있다.

3.5 프로미스 에러처리

then에서 처리하기

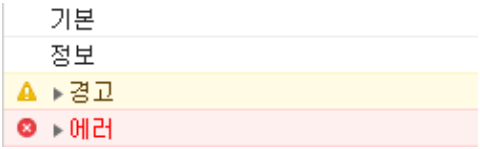
```
promiseGet(url)
.then((result) => console.log(result), (error) => console.error(error))
```

then catch로 처리하기

```
promiseGet(url)
.then((result) => console.log(result))
.catch((error) => console.error(error))
```

console.log vs console.error

```
console.log('기본');
console.info('정보');
console.warn('경고');
console.error('에러');
```



3.6 프로미스 체이닝

```
getDataPromise('10').then((data) => {
  return getDataPromise(data)
}).then((data) => {
  return getDataPromise(data)
}).then((data) => {
  console.log(data)
}).catch((err) => { //catch: an error handler for all of our promises in the promise chaining
  console.log(err)
})
```

후속 처리 메서드	콜백 함수 인수	후속 처리 메서드 반환 값
then	getDataPromise함수가 반환한 프로미스가 resolve한 값	콜백 함수가 반환한 프로미스
then	첫 번째 then 메서드가 반환한 프로미스가 resolve한 값	콜백 함수가 반환한 프로미스
then	첫 번째 then 메서드가 반환한 프로미스가 resolve한 값	콜백 함수가 반환한 값 (undefined)를 resolve한 프로미스
catch	getDataPromise 함수 혹은 앞선 후속 처리 메서드가 반환한 프로미스가 reject한 값	콜백 함수가 반환한 값 (undefined)를 resolve한 프로미스

3.6 실습 풀이

▼ 콜백 대신 Promise 써보기

```
const userLeft = false;
const userEnteredChat = false;
```

```
const posts = [
  { title: 'Post 1', body: '첫번째 게시글입니다.' },
  { title: 'Post 2', body: '두번째 게시글입니다.' },
];

const canvas = document.getElementById('myCanvas');
// 2d모드의 그리기 객체를 통해 canvas에 그림을 그릴 수 있다.
const ctx = canvas.getContext('2d');
// 이미지 객체 생성
const img = new Image();
// 이미지 소스 설정
img.src = './image/m1.jpg';
// 이미지 로드이벤트- 콜백함수 등록

function usePromise() {
  return new Promise((resolve, reject) => {
    if (userLeft) {
      //사용자가 웹 페이지를 떠났을 때
      reject({
        name: 'user left',
        message: ':(',
      });
      document.write('user left:(');
    } else if (userEnteredChat) {
      resolve({
        name: 'the user has entered the chat',
        message: 'entertain the user with memes',
      });
      img.onload = function () {
        ctx.drawImage(img, 100, 100);
      };
      document.write('entertain the user with memes');
    } else {
      resolve('subscribe to see more memes');
      setTimeout(() => {
        let output = '';
        posts.forEach((post, index) => {
          output = output + `<li>${post.title}<br> 내용: ${post.body} </li> `;
        });
        document.body.innerHTML = output;
      }, 1000);
    }
  });
}

usePromise()
  .then(message => {
    console.log('success:' + message);
  })
  .catch(error => {
    console.log(error.name + '' + error.message);
  });
```

4. 실습

4.1 실습 풀이

▼ Promise로 타이틀 바꾸기

```
//posts 변수를 수정하지 마세요.
const posts = [
  { title: 'Post 1', body: '첫번째 게시글입니다.' },
  { title: 'Post 2', body: '두번째 게시글입니다.' },
  { title: 'Post 3', body: '세번째 게시글입니다.' },
  { title: 'Post 4', body: '네번째 게시글입니다.' },
  { title: 'Post 5', body: '다섯번째 게시글입니다.' },
];

function getPosts() {
  setTimeout(() => {
    posts.forEach(post => {
      document.body.innerHTML += `<li>${post.title}<br> 내용: ${post.body} </li>`;
    });
  }, 1000);
}

function createPost(post) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      posts.push(post);
      const error = false;
      if (!error) resolve();
      else reject();
    }, 2000);
  });
}

createPost({ title: 'Post N', body: 'N번째 게시글입니다.' })
  .then(result => {
    getPosts();
  })
  .catch(err => {
    console.error(err);
  });
```

5. 실습

5.1 실습 풀이

▼ Promise 사용하는 함수 만들기 3

```
// 지시사항에 따라 코드를 작성합니다.

function resolveAfterNSeconds(n) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(`${n}초가 지났습니다!`);
    }, n * 1000);
  });
}

// 아래의 숫자를 자유롭게 변경해 가며 실행해 보세요.
// 물론 그대로 두어도 됩니다. 채점과는 무관합니다.
const inputA = 1;

// 실행 혹은 제출을 위한 코드입니다. 지우거나 수정하지 말아주세요.
module.exports = { inputs: [inputA], func: resolveAfterNSeconds };
```

6. 실습

6.1 fetch

HTTP 요청 전송 기능을 제공하는 클라이언트 사이드 web api

6.1.1 fetch 함수 구조

- URL
- HTTP 요청 메서드
- HTTP 요청 헤더
- 페이로드

연습 툴

https://jsonplaceholder.typicode.com/

6.2 GET 요청

데이터를 가지고 옵니다.

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
  .then((response) => response.json())
```

```
  .then((data) => console.log(data));
```

6.3 POST 요청

새로운 데이터를 추가합니다.

```
fetch("https://jsonplaceholder.typicode.com/posts", {
    method: "POST",
    headers: {
        "Content-Type": "application/json",
    },
    body: JSON.stringify({
        title: "Test",
        body: "I am testing!",
        userId: 1,
    }),
})
    .then((response) => response.json())
    .then((data) => console.log(data));
```

6.4 PATCH 요청

수정하고자 하는 데이터의 일부분을 업데이트 해줍니다.

```
fetch("https://jsonplaceholder.typicode.com/posts/1", {
    method: "PATCH",
    headers: {
        "Content-Type": "application/json",
    },
    body: JSON.stringify({
        userId: 2,
    }),
})
    .then((response) => response.json())
    .then((data) => console.log(data));
```

6.5 DELETE 요청

데이터를 삭제합니다.

```
fetch("https://jsonplaceholder.typicode.com/posts/1", {
  method: "DELETE",
```

```
})
  .then((response) => response.json())
  .then((data) => console.log(data))
```

6.6 오류처리

```
fetch("https://jsonplaceholder.typicode.com/post/100")
    .then((response) => {
        if (response.ok) return response.json();
        else throw new Error(`${response.status} 요류가 발생했습니다.`);
    })
    .then((response) => {
        console.log(response);
    })
    .catch((error) => {
        console.error(error);
    });
```

6.1 실습 풀이

▼ fetch 사용하여 json 데이터 가져오기 2

```
// 자유롭게 코드를 작성하여, 예시 화면이 구현되도록 해 보세요.
const selectElem = document.querySelector('#selectEmployeeCode');
const buttonElem = document.querySelector('#buttonSubmit');
const emailElem = document.querySelector('#employeeEmail');

buttonElem.addEventListener('click', showEmail);

function showEmail(e) {
  const selectedEmployeeCode = selectElem.value;

  fetch('employees.json')
    .then(response => response.json())
    .then(data => {
      const foundData = data.Employees.find(
        employee => employee.employeeCode === selectedEmployeeCode
      );

      emailElem.innerHTML = foundData.emailAddress;
    });

  e.preventDefault();
}
```