자바스크립트 비동기

- 자바스크립트에서의 비동기(asynchronous) 처리
  특정 코드의 실행이 완료될 때까지 기다리지 않고, 다음 코드를 계속해서 실행하는 방식을 말합니다. 이는 자바스크립트가 단일 스레드(single-threaded) 기반의 언어라는 점에서 중요한 개념입니다.

- 비동기 처리의 필요성
  자바스크립트는 웹 브라우저에서 사용되며, 사용자 인터페이스와 상호작용하는데 있어서 빠른 반응성이 필요합니다. 만약 모든 작업이 순차적으로(동기적으로) 처리된다면, 한 작업이 완료될 때까지 UI는 멈춘 상태가 될 수 있습니다. 예를 들어, 네트워크 요청이나 파일 읽기와 같은 시간이 오래 걸리는 작업을 수행할 때, 이를 비동기적으로 처리하지 않으면 애플리케이션이 해당 작업의 완료를 기다리면서 다른 작업을 수행할 수 없게 됩니다.

  비동기 처리 방식을 통해 자바스크립트는 시간이 오래 걸리는 작업들을 효율적으로 관리하며, 동시에 UI의 응답성을 유지할 수 있습니다.

- 비동기 처리 방법
  자바스크립트에서는 다음과 같은 방법으로 비동기 처리를 할 수 있습니다.

방식	설명
콜백 함수(Callback Functions)	비동기 작업이 완료되었을 때 호출되는 함수로, setTimeout, setInterval, 파일 읽기, 네트워크 요청 등에 주로 사용됩니다.
프로미스(Promise)	비동기 작업의 성공 또는 실패와 관련된 값을 나타내는 객체로, 비동기 작업을 구조화하고 관리하는 데 사용됩니다.
async/await	프로미스를 사용하는 더 간편한 방법으로, 비동기 코드를 동기적으로 작성할 수 있도록 합니다.

Promise의 Method 종류

`Promise.all()` 실행 가능한 모든 프로미스 반환
`Promise.then()` 반환된 프로미스 체이닝
`Promise.reject()` 프로미스 then 객체를 거부
`Promise.resolve()` 프로미스 then 객체를 반환
`Promise.catch()` 프로미스 에러 반환
`Promise.finally()` 마지막 프로미스 체이닝

```
// Callback function
function showTime() {
  console.log("3초가 지났습니다.");
}

setTimeout(showTime, 3000); // 3초 후에 showTime 콜백 함수를 실행
```

```
// Promises
function readFileAsync(path) {
  return new Promise((resolve, reject) => {
    fs.readFile(path, 'utf-8', (err, data) => {
      if (err) {
        reject(err);   // 에러 발생 시
      } else {
        resolve(data); // 성공적으로 읽어온 경우
      }
    });
  });
}

readFileAsync('example.txt')
  .then(data => console.log(data))
  .catch(err => console.error(err));


// async await
async function readAndShowFile() {
  try {
    const data = await readFileAsync('example.txt');
    console.log(data);
  } catch (err) {
    console.error(err);
  }
}

readAndShowFile();
```

콜백 함수(Callback Functions)

콜백 함수를 통해 시간에 따라 함수 호출을 스케줄링할 수 있습니다.

함수	설명
`setTimeout`	`setTimeout` 함수는 지정된 시간이 지난 후에 함수를 한 번 실행합니다. 예를 들어, 5초 후에 메시지를 출력하는 코드는 다음과 같습니다.
`setInterval`	`setInterval` 함수는 지정된 간격마다 함수를 반복해서 실행합니다. 예를 들어, 매 2초마다 현재 시간을 출력하는 코드는 아래와 같습니다.
`clearTimeout`과 `clearInterval`	`setTimeout`과 `setInterval`의 결과를 각각 `clearTimeout`과 `clearInterval`을 사용하여 취소할 수 있습니다. 예를 들어, `setTimeout`을 취소하는 방법은 다음과 같습니다.

```
// Callback function
// setTimeout
```

```
function showTime() {
  console.log("3초가 지났습니다.");
}
setTimeout(showTime, 3000); // 3초 후에 showTime 콜백 함수를 실행

// setInterval
function showCurrentTime() {
  let currentTime = new Date();
  console.log(currentTime.toLocaleTimeString()); // 현재 시간을 출력
}
setInterval(showCurrentTime, 2000); // 2000ms(2초) 간격으로 showCurrentTime 함
수 반복 호출

// setTimeout을 취소
let timerId = setTimeout(() => console.log("이 메시지는 보이지 않습니다"), 1000);
clearTimeout(timerId);

// 5초 후에 setInterval을 취소
let intervalId = setInterval(() => console.log("이 메시지는 매 2초마다 출력됩니
다"), 2000);
setTimeout(() => clearInterval(intervalId), 5000);
```