비동기프로그래밍 개념

비동기 프로그래밍을 쉽게 이해하기 위해 배달 서비스를 예로 들어볼게요. 배달 음식을 주문하면 요리가 완성되기 전까지 가만히 기다리지 않고, 그 사이에 다른 일도 할 수 있죠. 자바스크립트도 이런 식으로 비동기 작업을 처리합니다.

1. 동기 프로그래밍(Synchronous Programming)

- **동기 프로그래밍**은 **순차적으로 작업**이 진행됩니다. 하나의 작업이 끝나야 다음 작업이 시작됩니다.
- 예를 들어, 어떤 작업을 처리하는 데 시간이 20초 걸린다면, 그 작업이 끝난 후에야 다른 작업을 시작할 수 있습니다.
- 이런 방식은 **단순하고 직관적**이지만, 작업이 오래 걸릴 경우 전체 프로그램이 느려질 수 있습니다.

2. 비동기 프로그래밍(Asynchronous Programming)

- **비동기 프로그래밍**은 **여러 작업을 병행**해서 처리합니다. 즉, 하나의 작업이 완료되지 않아도 다른 작업을 동시에 시작할 수 있습니다.
- 자바스크립트에서 **비동기 작업**은 이벤트 루프를 통해 처리되며, 이 작업은 **병행성**을 가지고 있습니다. 병행성은 **여러 작업을 동시에 시작하지 않지만**, 한 작업이 완료될 때까지 다른 작업을 처리하는 것을 의미합니다.
- 예를 들어, 네트워크 요청이나 파일 읽기 같은 작업은 시간이 오래 걸리기 때문에, 이런 작업을 비동기적으로 처리하여 전체 프로그램의 흐름을 멈추지 않고 다른 작업을 계속할 수 있습니다.

3. 비동기 프로그래밍의 장점

- **성능**: 작업이 병렬로 실행되므로, 전체적인 작업 처리 시간이 단축됩니다.
- **비효율 감소**: 시간이 오래 걸리는 작업이 진행되는 동안에도 다른 작업이 진행될 수 있어 시스템 자원을 효율적으로 사용할 수 있습니다.

4. 비동기 프로그래밍의 단점

- **가독성**: 비동기 작업이 많아지면 코드가 복잡해지고 가독성이 떨어질 수 있습니다. 특히 **콜백 지옥**이 발생할 수 있습니다.
- **디버깅의 어려움**: 비동기 작업은 언제 완료될지 미리 알 수 없기 때문에, 에러가 발생했을 때 원인을 추적하기가 어렵습니다.
- **작업 완료 여부**: 비동기 작업은 **끝나는 시점**을 명확히 알기가 어렵습니다. 콜백 함수, `Promise`, `async/await` 같은 구조를 사용해 이를 관리해야 하지만, 작업이 완료되었는지 매번 확인해야 하는 번거로움이 있습니다.

5. 병행성과 동시성

- **병행성(Concurrency)**: 여러 작업을 **번갈아 가며** 실행하는 방식입니다. 자바스크립트의 비동기 프로그래밍은 단일 스레드 환경에서 병행성을 통해 여러 작업을 중단하지 않고 동시에 진행되는 것처럼 처리합니다.
- **동시성(Parallelism)**: 여러 작업이 **실제 동시에** 실행되는 방식입니다. 이는 멀티스레드 환경에서 이루어지며, 자바스크립트는 기본적으로 단일 스레드이기 때문에 비동기 작업은 병행성을 기반으로 합니다.

---

1. 콜백 함수

콜백 함수는 배달 음식을 주문하고, 음식이 도착하면 **알림을 주는 함수**라고 생각할 수 있어요. 음식을 기다리는 동안 다른 일을 할 수 있지만, 음식이 도착하면 그때 알려줘요.

코드 예시:

```
function orderFood(food, callback) {
  console.log(`${food} 주문을 시작해요.`);
  setTimeout(function() {
    console.log(`${food}가 배달되었어요!`);
    callback(); // 음식이 도착하면 콜백 함수 실행
  }, 3000); // 3초 후에 콜백 실행
}

orderFood('피자', function() {
  console.log('맛있게 먹어요!');
});
```

- `orderFood` 함수는 피자를 주문하고, 피자가 도착하면 **맛있게 먹는 작업**(콜백 함수)을 실행합니다.

2. 프로미스(Promise)

프로미스는 **배달이 올지 말지 약속**하는 것과 같아요. 음식이 도착하면 성공한 거고, 도착하지 않으면 실패한 거예요.

코드 예시:

```
const orderFood = new Promise((resolve, reject) => {
  const isDelivered = true; // 음식이 배달될지 여부를 확인하는 변수
  if (isDelivered) {
    resolve('음식이 배달되었어요!');
  } else {
    reject('배달에 실패했어요...');
  }
});

orderFood
  .then((message) => {
    console.log(message); // 배달 성공 시 실행
    console.log('맛있게 먹어요!');
  })
  .catch((error) => {
    console.log(error); // 배달 실패 시 실행
   });
```

- `resolve`는 음식이 성공적으로 배달되었을 때, `reject`는 배달 실패 시 사용해요. `.then()`에서 성공했을 때 메시지를 출력하고, `.catch()`에서는 실패 메시지를 출력해요.

3. `async`와 `await`

`async`와 `await`는 배달을 기다리는 동안 마치 **잠시 기다렸다가 음식이 오면 먹는 것**처럼 동작해요. 주문이 완료될 때까지 기다리면서, 그동안 다른 일을 할 수 있어요.

코드 예시:

```
function deliverFood() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve('음식이 배달되었어요!');
    }, 3000); // 3초 후에 음식 배달 완료
  });
}

async function orderAndEat() {
  console.log('음식을 주문했어요...');
  const message = await deliverFood(); // 배달이 올 때까지 기다림
  console.log(message); // 배달된 후 메시지 출력
  console.log('이제 음식을 먹어요!');
}

orderAndEat();
```

- `async` 함수는 음식이 도착할 때까지 기다리다가, `await` 키워드로 **음식이 도착한 후에** 다음 일을 처리합니다.

이렇게 배달 서비스를 비유로 해서 비동기 프로그래밍의 세 가지 방법, **콜백 함수**, **프로미스**, 그리고 **`async`와 `await`**을 이해할 수 있어요.