

Coming  
soon

수업 준비중입니다  
잠시만 기다려주세요...

4주차 수업

# JavaScript - 심화

4주차

# 배열 내장 함수

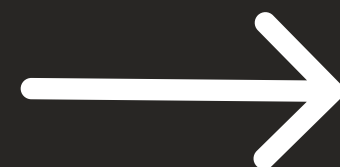
배열을 다룰 때 알고있으면 유용한 내장 함수

## forEach

기존에 우리가 배웠던 for문을 대체할 수 있음

```
const foods = [떡볶이, '치킨', '피자', '족발', '햄버거'];
```

```
for (let i = 0; i < foods.length; i++) {  
  console.log(foods[i]);  
}
```



```
foods.forEach(food => {  
  console.log(food);  
});
```

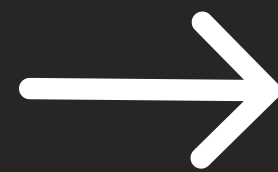
# 배열 내장 함수

배열을 다룰 때 알고있으면 유용한 내장 함수

## map

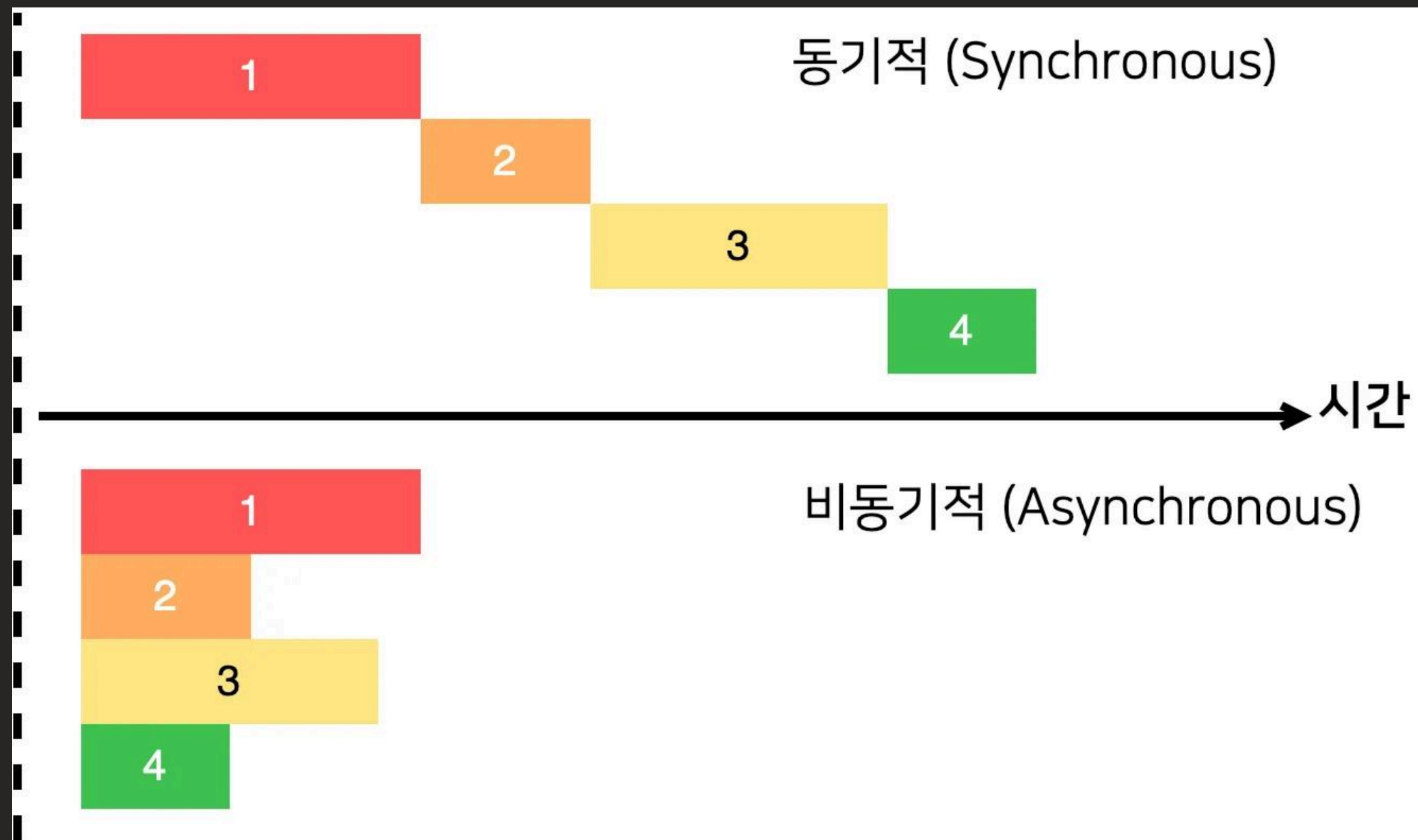
배열 안의 각 원소를 변환 할 때 사용되며, 이 과정에서 새로운 배열이 생성됨

```
const array = [1, 2, 3, 4, 5, 6, 7, 8];  
const squared = [];  
  
array.forEach(n => {  
  squared.push(n * n);  
});  
  
console.log(squared);
```



```
const square = n => n * n;  
const squared = array.map(square);  
  
console.log(squared);
```

# 비동기 프로그래밍이란?



# 비동기 프로그래밍이란?

## setTimeout

setTimeout은 주어진 지연시간이 경과한 후 특정 코드나 함수를 실행하도록 예약하는데 사용



```
console.log(1);  
console.log(2);  
setTimeout(() => {console.log(3)}, 5000); // 5000은 5초와 같습니다.  
console.log(4);
```

# Callback



# Callback

## Callback 지옥

비동기 처리 로직을 위해 콜백함수를 연속해서 사용할 때 발생하는 문제

```
$.get('url', function(response) {  ///url에서 데이터를 가져옴
  parseValue(response, function(id) {  ///데이터를 파싱해서 id 추출
    auth(id, function(result) {  ///id를 이용해 인증 수행(auth)
      display(result, function(text) {  ///인증 결과를 이용해 최종 텍스트 생성
        console.log(text);  /// 최종 결과를 콘솔에 출력
      });
    });
  });
});
});
```

아직 코드를 완벽히  
이해할 필요는 없습니다!  
이런거구나~ 정도만 보기



# Callback

## Callback 지옥

비동기 처리 로직을 위해 콜백함수를 연속해서 사용할 때 발생하는 문제



```
function parseValueDone(id) {  
  auth(id, authDone); }  
  
function authDone(result) {  
  display(result, displayDone); }  
  
function displayDone(text) {  
  console.log(text); }  
  
$.get('url', function(response) {  
  parseValue(response, parseValueDone); }));
```

아직 코드를 완벽히  
이해할 필요는 없습니다!  
이런거구나~ 정도만 보기

# Callback

## Callback 지옥

비동기 처리 로직을 위해 콜백함수를 연속해서 사용할 때 발생하는 문제

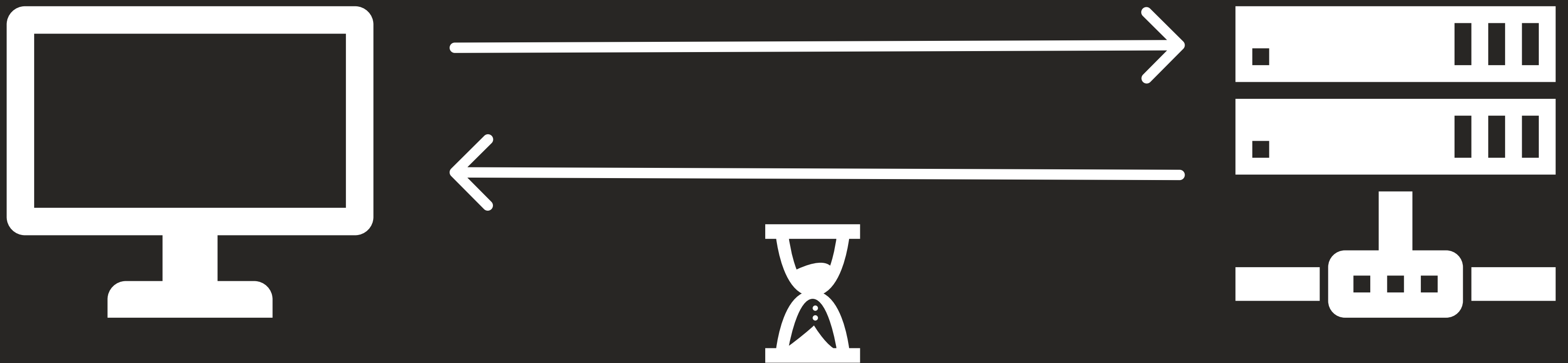
```
function parseValueDone(id) {  
  auth(id, authDone);  
}  
  
function authDone(result) {  
  display(result, displayDone);  
}  
  
function displayDone(text) {  
  console.log(text);  
}  
  
$.get('url', function(response) {  
  parseValue(response, parseValueDone);  
});
```

# Callback

아직 코드를 완벽히  
이해할 필요는 없습니다!  
이런거구나~ 정도만 보기

# Promise

비동기 작업을 더 편리하게 다루기 위한 객체



# Promise

비동기 작업을 더 편리하게 다루기 위한 객체

## Promise의 세가지 상태

### 1. pending(대기)

비동기 처리 로직이 아직 완료되지 않은 상태

### 2. fulfilled(이행)

비동기 처리가 완료되어 프로미스가 결과값을 반환해준 상태

### 3. rejected(실패)

비동기 처리가 실패하거나 오류가 발생한 상태

# Promise

비동기 작업을 더 편리하게 다루기 위한 객체

## Promise 사용법



```
new Promise()
```

`new Promise()` 메서드 호출 → 대기 상태



```
const condition = true;
const promise = new Promise((resolve, reject) => {
  // 보통 여기에 비동기 함수를 넣습니다!
  if (condition) {
    resolve('resolved'); // condition이 참
  } else {
    reject('rejected'); // condition이 거짓
  }
});
```

값이 참이면 ⇒ **resolve** 호출  
거짓이면 ⇒ **reject** 호출

# Promise

비동기 작업을 더 편리하게 다루기 위한 객체

## Promise 사용법

```

promise
  .then((res) => {
    console.log(res);
  })
  .catch((error) => {
    console.error(error);
  });
    
```

promise 상태  
Pending →

전달 인자

실행되는 함수

resolve 함수

Fulfilled

비동기 작업의 **결과**를  
promise 객체에 전달

then 콜백함수

reject 함수

Rejected

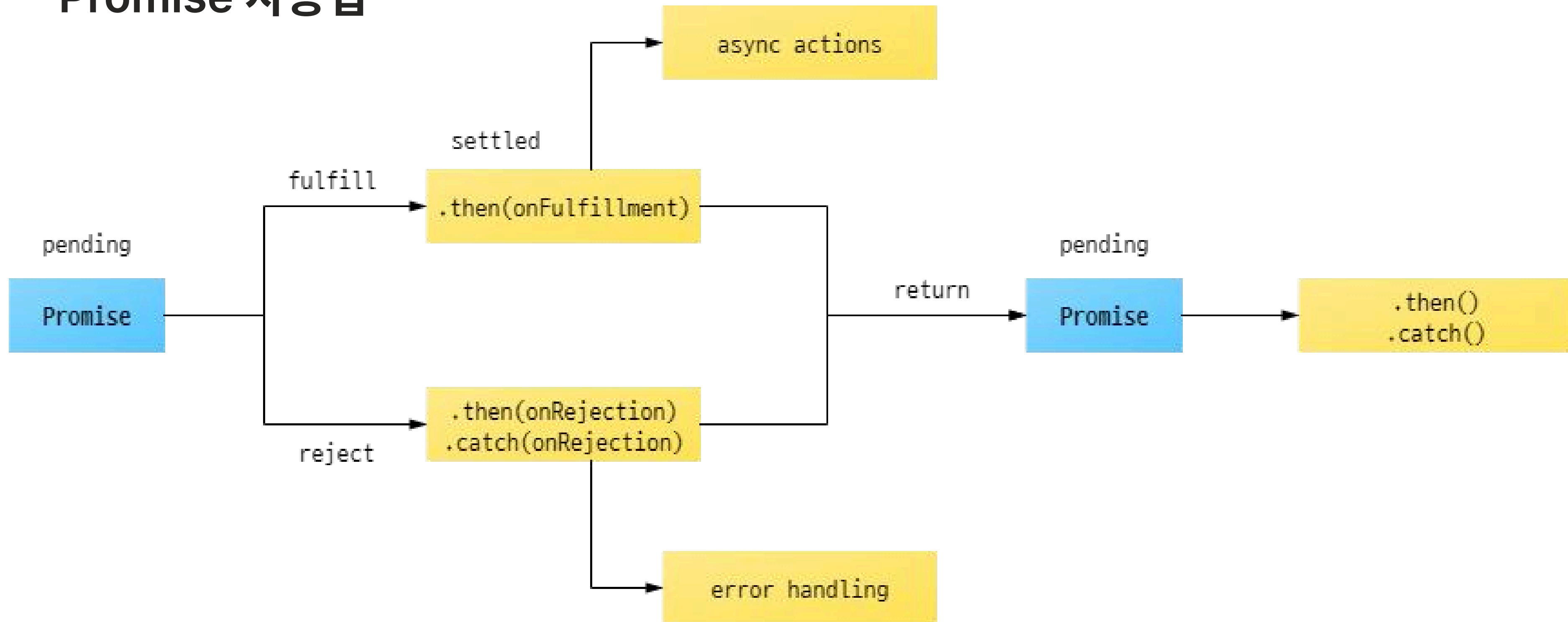
비동기 작업의 **오류**를  
promise 객체에 전달

catch 콜백함수

# Promise

비동기 작업을 더 편리하게 다루기 위한 객체

## Promise 사용법



# Promise

비동기 작업을 더 편리하게 다루기 위한 객체

## then() 지옥


```
const userInfo = {
  id: 'likelion@google.com',
  pw: '****' };

function parseValue() {
  return new Promise({
    // ...
  }); }

function auth() {
  return new Promise({
    // ...
  }); }

function display() {
  return new Promise({
    // ...
  }); }

getData(userInfo)
  .then(parseValue)
  .then(auth)
  .then(diaplay);
```

■ ■ ■ 

더 좋은 건 없을까??



# async/await

비동기 작업을 더 편리하게 다루기 위한 객체

## async/await 기본 문법

```
const 함수명 = async () => {  
  await 비동기_처리_메서드_명();  
};
```

→ 반드시 **promise 객체** 반환해야함

- await는 async 함수 안에서만 동작  
⇒ 즉 async 함수가 아닌데 await를 사용하면 문법 에러가 발생
- '변수 = await promise'인 형태에서는 프로미스가 resolve된 값이 변수에 저장,  
'변수 = await 값'인 경우 그 값이 변수에 저장

# async/await

비동기 작업을 더 편리하게 다루기 위한 객체

## Promise 사용법

```
const fetchItems = () => {  
  return new Promise((resolve, reject) => {  
    let items = [1, 2, 3];  
    resolve(items);  
  });  
};  
  
const logItems = async () => {  
  let resultItems = await fetchItems();  
  console.log(resultItems); // [1,2,3]  
};  
  
logItems();
```

# Try / Catch

이것만은 알고 가자!

## 제일 흔하게 사용되는 코드 형태

```
const fetchData = async () => {  
  try {  
    const response = await  
      axios.get('https://api.example.com/data');  
    ///url의 api로 get 요청을 보낸 후 await를 사용하여 응답을 받을 때까지 기다림  
    const data = response.data;  
    ///실제 API에서 받아온 데이터를 추출 후 data로 지정  
    console.log(data);  
  } catch (error) {    ///API 요청이 실패하면 어떤 에러인지 출력  
    console.error('Error:', error);  
  }  
}  
  
fetchData();
```

# Event 다루기

지긋지긋한 비동기를 벗어나

## 이벤트란?

이벤트는 웹 브라우저와 사용자 사이에 상호작용이 발생하는 특정 시점을 의미합니다.

자바스크립트에서 이벤트가 발생하면, 이벤트 종류에 따라 특정 작업을 수행하거나 미리 등록한 함수를 호출하는 등의 조작을 지정할 수 있습니다.

BUTTON



페이지 이동

# Event 처리기

## 이벤트 처리기란?

이벤트가 발생했을 때 이에 반응하여 수행할 동작이 작성된 함수에 연결되도록 하는 처리기.  
즉, 해당 이벤트가 발생할 때 대응하여 동작하는 **콜백함수**를 말합니다.

**이벤트 핸들러**

**vs**

**이벤트 리스너**

# Event 처리기

eventHandler

addEventListener()

형태

속성

메서드

중복 등록

불가능

가능

새 이벤트가 기존 것을 덮음

새 이벤트가 추가 됨

제거 여부

불가능

가능

removeEventListener() 이용

기능 확장성

제한적

더 유연하고 확장 가능

# Event 다루기

가장 많이 사용하는 중요 이벤트

## 이벤트 핸들러 속성

마우스 이벤트	onclick	마우스로 클릭하면 발생
	onmouseover	마우스 포인터를 올리면 발생
	onwheel	마우스 휠을 움직이면 발생
키보드 이벤트	onkeypress	키보드 버튼을 누르고 있는 동안 발생
	onkeydown	키보드 버튼을 누른 순간 발생
	onkeyup	키보드 버튼을 눌렀다가 떼는 순간 발생
포커스 이벤트	onfocus	요소에 포커스가 되면 발생
	onchange	요소의 값이 바뀌었을 때 발생
폼 이벤트	onselect	텍스트 필드 등의 텍스트를 선택했을 때 발생
	onsubmit	폼이 전송 될 때 발생

# Event 다루기

가장 많이 사용하는 중요 이벤트

## 이벤트 핸들러 - 인라인 방식

```
<button onclick="clickEvent()">버튼</button>
```

```
<script>  
function clickEvent(){  
    alert("click");  
}  
</script>
```



# Event 다루기

가장 많이 사용하는 중요 이벤트

## 이벤트 핸들러 - 비인라인 방식

```
let btn = document.getElementById('myBtn');  
/// id="myBtn"을 가진 버튼 요소를 JavaScript에서 가져와 btn 변수에 저장.  
  
btn.onclick = function() {  
  ///onclick에 익명 함수를 할당하여 클릭 이벤트를 처리.  
    alert('clicked!');  
}
```

# Event 다루기

## 이벤트 리스너



```
let btn = document.getElementById('myBtn');  
/// id="myBtn"을 가진 버튼 요소를 JavaScript에서 가져와 btn 변수에 저장.  
  
btn.addEventListener('click', function() {  
  alert('clicked!');  
});
```

# 과제

## 숫자 맞추기 게임

1부터 100 사이의 숫자를 맞춰보세요!

확인

남은 기회: 9

50 ▲ UP!

## 숫자 맞추기 게임

1부터 100 사이의 숫자를 맞춰보세요!

확인

남은 기회: 10

## 숫자 맞추기 게임

1부터 100 사이의 숫자를 맞춰보세요!

확인

남은 기회: 4

정답입니다! 56