

# 1~2. 리액트 학습 준비 (실습)

Prof. Seunghyun Park ([sp@hansung.ac.kr](mailto:sp@hansung.ac.kr))

Division of Computer Engineering

## 학습 목표: 2. 리액트를 위한 자바스크립트

---

- ES2015+
  - 함수
  - 화살표 함수
  - 비동기 자바스크립트

# 실습1: 함수

- 문자열을 매개변수로 전달받고, 해당 문자열을 console에 출력하는 함수
  - 문제1. 함수 선언문으로 함수를 정의하고, 함수 호출하기
    - 함수명: func1
  - 문제2. 함수 표현식으로 함수를 정의하고, 함수 호출하기
    - 함수명: func2
  - 문제3. 화살표 함수로 함수를 정의하고, 함수 호출하기
    - 함수명: func3

```
/* ch02-func1.js */  
  
function func1(str){ // 함수 선언문  
    console.log(str);  
}  
  
func1("Hello, func1");
```

Hello, func1

## 실습2: 화살표 함수 (1)

---

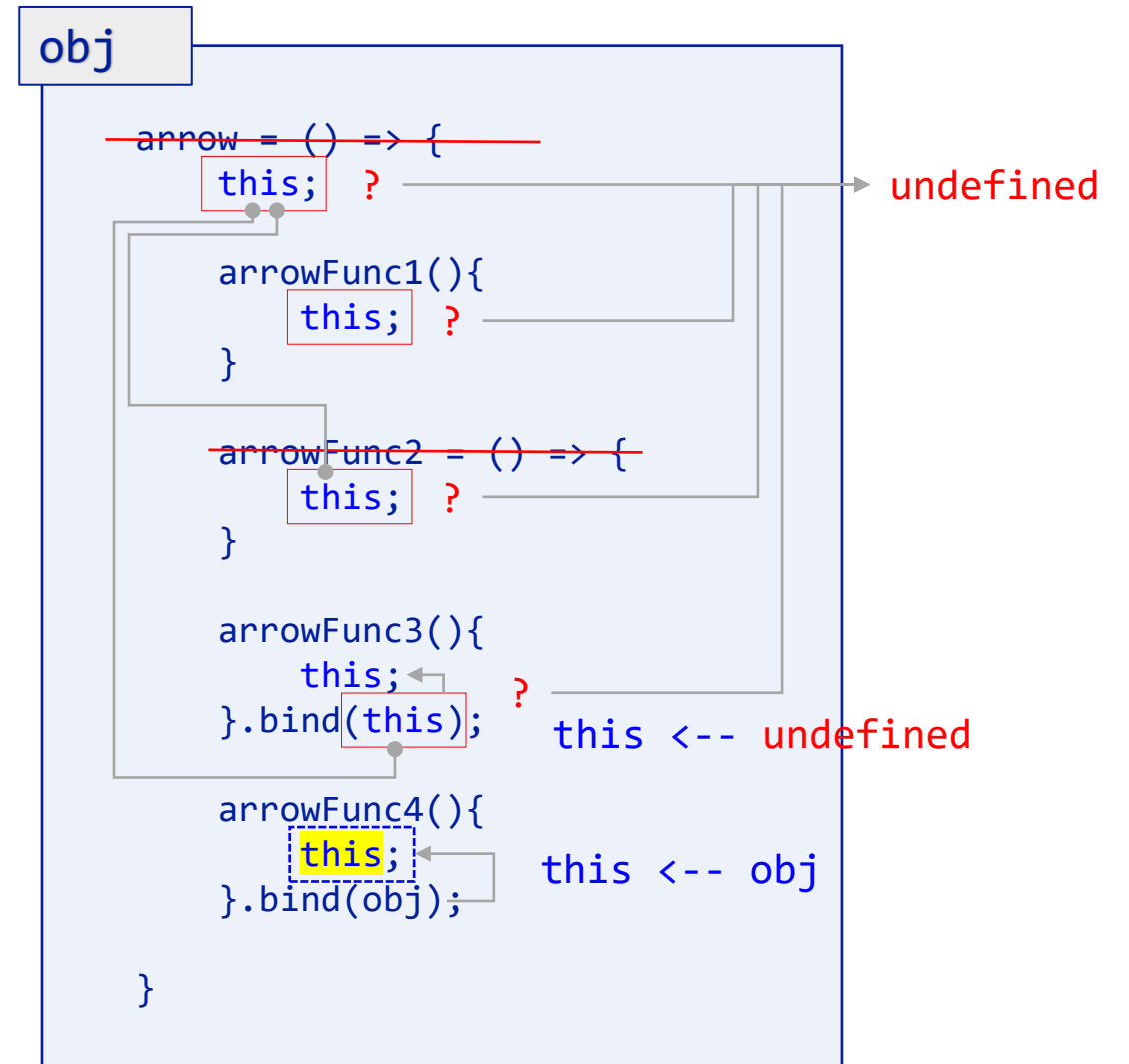
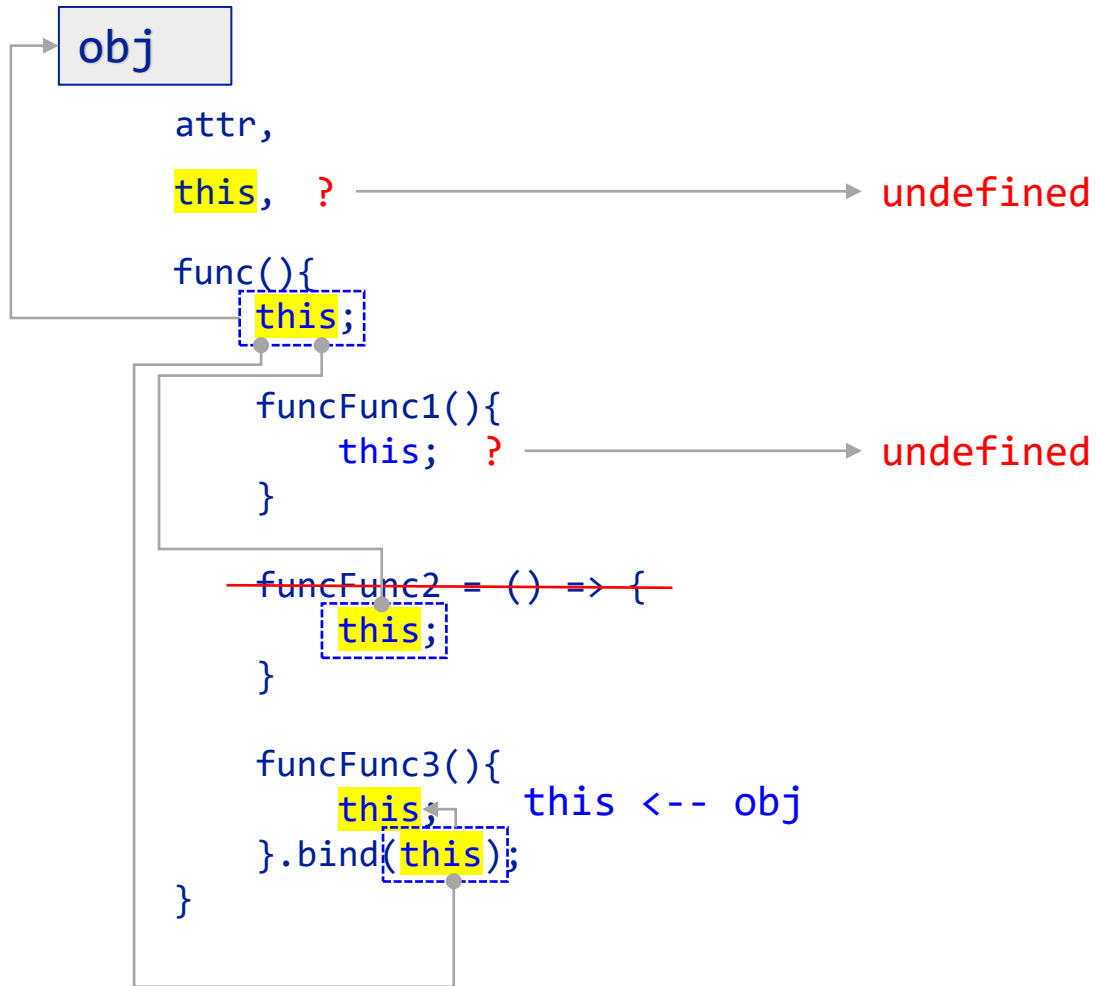
- 다음 조건에 따라 화살표 함수를 구현하고, 결과를 console에 출력
  - 문제1. 매개변수를 2개 전달 받아서, 이 값을 더한 값을 반환하는 화살표 함수 구현
    - 함수명: aFunc1
  - 문제2. 매개변수 없이 함수 내부에서 선언한 상수 2개의 값을 더한 값을 반환하는 화살표 함수
    - 함수명: aFunc2
  - 문제3a. 매개변수를 1개 전달 받아서, 이 값에 200을 더한 값을 반환하는 화살표 함수 구현
    - 함수명: aFunc3a
  - 문제3b. 문제3a에서 매개변수의 ( ) 를 생략
    - 함수명: aFunc3b

## 실습2: 화살표 함수 (2)

---

- 다음 조건에 따라 화살표 함수를 구현하고, 결과를 console에 출력
  - 문제4a. 문제3a (매개변수 1개 전달 받아서, 이 값에 200을 더한 값을 반환하는 화살표 함수)의 함수 정의를 1줄로 구현
    - 함수명: aFunc4a
  - 문제4b. 문제4a의 함수 정의에서 { }와 return 구문 생략하여 구현
    - 함수명: aFunc4b
  - 문제5. 문제4b의 화살표 함수의 호이스팅 여부 확인
    - 함수명: aFunc5

# 객체와 this



# 실습3: 객체와 this (1)

- HTML 내부 script 영역 안에 다음의 문제를 순차적으로 구현

1) 전역 scope에서 **this** 값 확인

2) **obj1** 객체 생성

- 속성 **obj\_this**에 **this** 할당
- 메서드1 정의: 함수 선언문 **func1()** { ... }

3) **func1()** 내부 구현

- ① **this**와 **this.obj\_this** 값을 출력해서 비교
- 함수 표현식으로 **funcFunc1()**에서 **this** 값 확인
- 화살표 함수로 **funcFunc2()**에 ①을 구현하고 호출
- 함수 표현식으로 **funcFunc3()**에 ①을 구현하고, **this**를 바인딩 후 호출
- **setTimeout()**의 **callback1**에서 ② **this** 값 확인, **callback1**은 함수 선언문
- **setTimeout()**의 **callback2**에서 ② 구현, **callback2**는 화살표 함수

```
print this;
```

```
obj1 = { ... }
```

```
obj_this <- this
```

```
func1(){
```

```
  print this;
```

```
  print obj_this;
```

```
  // 함수 표현식으로 funcFunc1() 구현  
  funcFunc1(); // 함수 표현식
```

```
  // 화살표 함수로 funcFunc2() 구현  
  funcFunc2(); // 화살표 함수
```

```
  // 함수 표현식으로 구현하고 this 바인딩  
  funcFunc3(); // 함수 표현식 + bind()
```

```
  setTimeout(callback1, 0);
```

```
  setTimeout(callback2, 0);
```

```
}
```

# 실습4: 객체와 this (2)

- HTML 내부 script 영역 안에 다음의 문제를 순차적으로 구현

1) 전역 scope에서 **this** 값 확인

2) **obj2** 객체 생성

- 속성 **obj\_this**에 **this** 할당
- 메서드2: 화살표 함수로 정의 **func2: () => { ... }**

3) **func2()** 내부 구현

- ① **this**와 **this.obj\_this** 값을 출력해서 비교
- 함수 표현식으로 **aFunc1()**에서 **this** 값 확인
- 화살표 함수로 **aFunc2()**에서 **this** 값 확인
- 함수 표현식으로 **aFunc3()**에 **this**를 바인딩하고, **this** 값 확인
- 함수 표현식으로 **aFunc4()**에 **obj2**를 바인딩하고, ①을 구현하여 확인

```
print this;
```

```
obj2 = { ... }
```

```
obj_this <- this
```

```
func2: () => {
```

```
  print this;
```

```
  print obj_this;
```

```
  // 함수 표현식으로 funcFunc1() 구현
```

```
  aFunc1(); // 함수 표현식
```

```
  // 화살표 함수로 funcFunc2() 구현
```

```
  aFunc2(); // 화살표 함수
```

```
  // 함수 표현식으로 구현하고 this 바인딩
```

```
  aFunc3(); // 함수 표현식 + bind(this)
```

```
  // 함수 표현식으로 구현하고 obj2 바인딩
```

```
  aFunc4(); // 함수 표현식 + bind(obj2)
```

```
}
```



# 실습5: 비동기 자바스크립트 구현

예제) 3개의 텍스트 파일을 순서대로 읽어와 콘솔에 출력하는 프로그램 구현

출력 예:

```
start
1st reading: This file contains sample text #1.
2nd reading: This file contains sample text #2.
3rd reading: This file contains sample text #3.
end
```



readme1.txt



readme2.txt



readme3.txt

```
const fs = require('fs');

fs.readFile('./readme1.txt', (err, data) => {
  if (err) {
    console.error(err);
  }
  else {
    console.log('1st reading', data.toString());
  }
});
```

# 실습5: 비동기 자바스크립트 구현 (1~2)

- 문제1: 3개의 비동기 함수를 연속해서 호출

```
// code1
console.log('start');
fs.readFile(filename1, callback);
fs.readFile(filename2, callback);
fs.readFile(filename3, callback);
console.log('end');
```

- 문제2: callback 함수 내부에서 비동기 함수를 호출

```
// code2
console.log('start');
fs.readFile(filename1, () => {
  console.log(data1);
  fs.readFile(filename2, () => {
    console.log(data2);
    fs.readFile(filename3, () => {
      console.log(data3);
      console.log('end');
    });
  });
});
```

# 실습5: 비동기 자바스크립트 구현 (3)

- 문제3: Promise 객체 활용

```
// code3
const promise = new Promise( (resolve, reject) => {
  console.log('start');
  fs.readFile(filename1, (err, data1) => {
    if (!err) resolve(data1);
  })
});

promise
  .then(data1 => {
    console.log(data1);
    return new Promise( (resolve, reject) => {
      fs.readFile(filename2, (err, data2) => {
        if (!err) resolve(data2);
      })
    })
  })
  .then(data2 => {
    console.log(data2);
    return new Promise( (resolve, reject) => {
      fs.readFile(filename3, (err, data3) => {
        if (!err) resolve(data3);
      })
    })
  })
  .then(data3 => console.log(data3))
  .catch(err => console.err())
  .finally(() => console.log('end'))
```

```
.then(data2 => {
  console.log(data2);
  return new Promise( (resolve, reject) => {
    fs.readFile(filename3, (err, data3) => {
      if (!err) resolve(data3);
    })
  })
})
.then(data3 => console.log(data3))
.catch(err => console.err())
.finally(() => console.log('end'))
```

# 실습5: 비동기 자바스크립트 구현 (4~5)

- 문제4: fs.promises 모듈 활용

```
// code4
console.log('start');
fsPromises.readFile(filename1)
  .then(data1 => {
    console.log(data1);
    return fsPromises.readFile(filename2)
  })
  .then(data2 => {
    console.log(data2);
    return fsPromises.readFile(filename3);
  })
  .then(data3 => console.log(data3))
  .catch(err => console.error())
  .finally(() => console.log('end'))
```

- 문제5: async/await 구문 활용

```
// code5
console.log('start');
(async () => {
  try {
    let data1 = await fsPromises.readFile(filename1);
    console.log(data1);
    let data2 = await fsPromises.readFile(filename2);
    console.log(data2);
    let data3 = await fsPromises.readFile(filename3);
    console.log(data3);
  }
  catch(err){
    console.error();
  }
  finally{
    console.log('end');
  }
})();
```