

1~2. 리액트 학습 준비

(리액트 소개, ES6+)

Prof. Seunghyun Park (sp@hansung.ac.kr)

Division of Computer Engineering

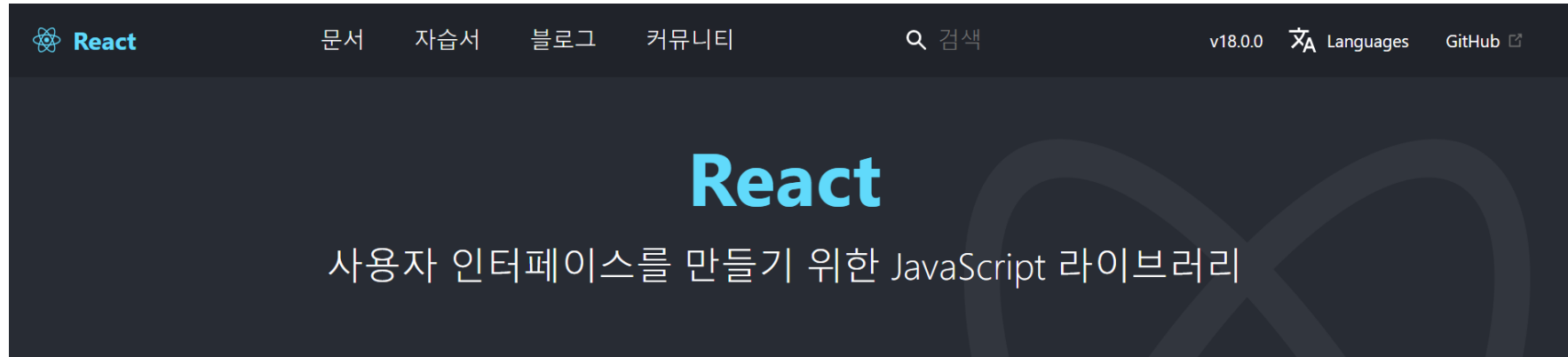
학습 목표: 1. 리액트 소개 & 2. 리액트를 위한 자바스크립트

- React 소개

- 참고: MVC 구조와 리액트
- 개발환경 설정
 - node, npm, yarn (각자)

- ES2015+

- 변수와 상수
- 템플릿 문자열
- 함수
- 화살표 함수
- 구조분해 할당
- 객체 리터럴
- 스프레드 연산자
- 비동기 자바스크립트
- 클래스



<https://ko.reactjs.org>

<https://react.dev>

- 사용자 인터페이스^{UI}를 만들기 위한 JavaScript 라이브러리

- Facebook (Meta)에서 만든 오픈소스 프로젝트

- MVC 애플리케이션의 View에 관련된 영역만 담당

- UI 라이브러리: 사용자 인터페이스를 구현하기 위해서 사용

- 컴포넌트 기반 라이브러리 (재사용성)

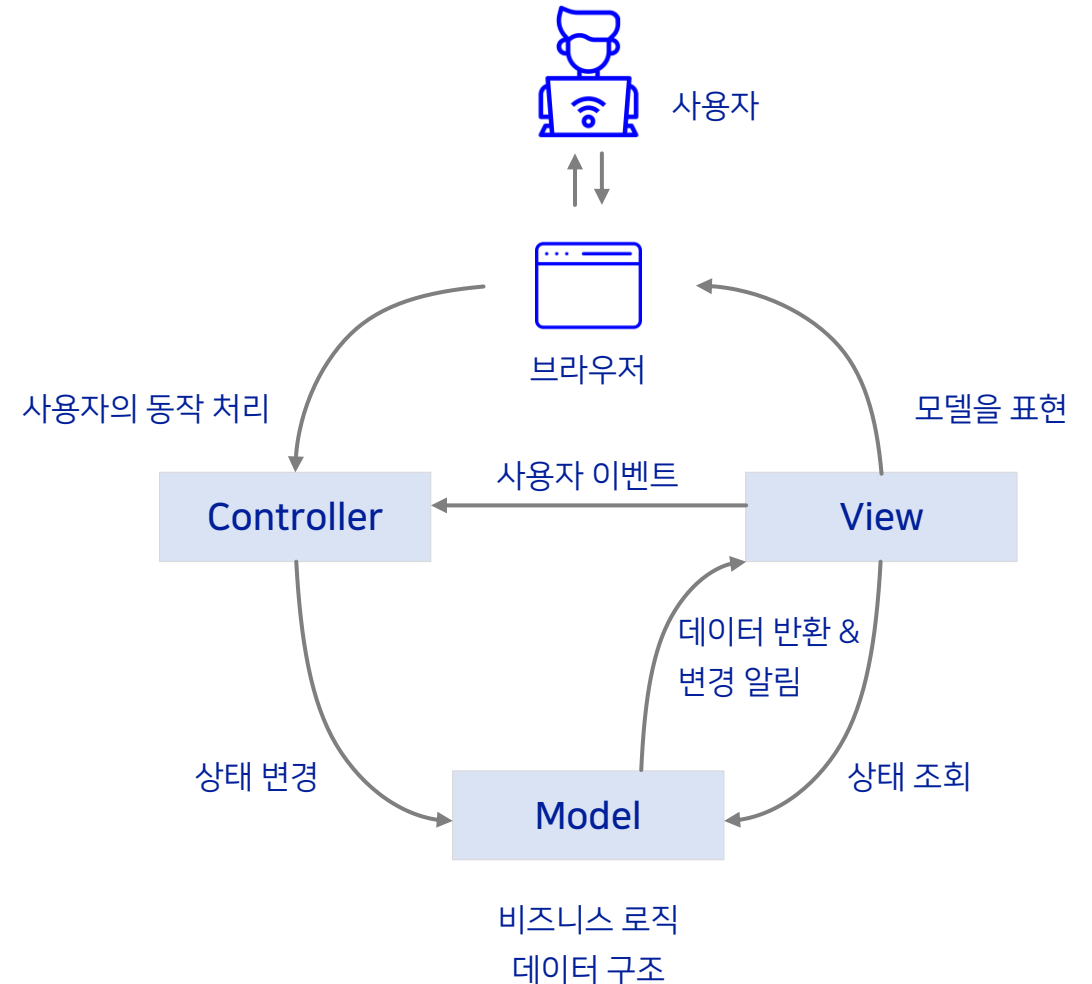
- Virtual DOM 사용

• MVC 구조

- 소프트웨어 디자인 패턴 중 하나
- 모델 + 뷰 + 컨트롤러로 구성
 - 모델: 데이터를 관리하는 영역
 - 뷰: 사용자에게 보이는 부분 (사용자 인터페이스 영역)
 - 컨트롤러: 모델 데이터를 조회, 수정
변경된 사항을 뷰에 반영

• React는 View에 관련된 영역만 처리

- 다른 프레임워크/라이브러리와 혼용 가능



- ES (ECMAScript)

※ ECMA (유럽 컴퓨터 제조회사 협회): European Computer Manufacturers Association

- ECMA-262 규격에 의해 정의된 범용 스크립트 언어 (by ECMA International)

- ECMA-262: 범용 목적의 스크립트 언어, JavaScript를 표준화하기 위해 개발

- JavaScript

- ECMAScript 사양을 준수하는 스크립트 언어

- ECMAScript 기반에 브라우저 처리 (BOM)와 도큐먼트 처리 (DOM) 추가: ECMAScript + BOM + DOM

- ES6

- ECMA-262의 여섯 번째 버전 (ES2015, ECMAScript 2015)

- 2015년 이후 매년 개정 (ES6, ES7, ES8, ...)

ECMAScript compatible table:

<http://kangax.github.io/compat-table/es2016plus/>



변수 선언: const

- 변수 선언: var

```
/* ch02-01-01-const.html */
```

```
// 변수 값은 변경 가능
```

```
var pizza = true
```

```
pizza = false
```

```
console.log(pizza)
```

```
false
```

- 상수 선언: const

```
/* ch02-01-02-const.html */
```

```
// ES6부터 상수 도입
```

```
const pizza = true
```

```
pizza = false
```

```
console.log(pizza)
```

```
Uncaught SyntaxError: Inline Babel script:  
"pizza" is read-only
```

변수 선언: **let** (계속) 스cope: 변수가 영향을 미치는 유효범위

- 변수 **var**의 scope

```
/* ch02-01-03-let.html */

// var 변수의 영역은 블록 안으로 제한되지 않음
var topic = "자바스크립트"

if (topic) {
  var topic = "리액트"
  console.log('블록', topic)
}
console.log('글로벌', topic)
```

블록 리액트
글로벌 리액트

- 변수 **let**과 상수 **const**의 scope: 블록

```
/* ch02-01-04-let.html */
let은 스코프를 벗어나면 사라진다
// let을 쓰면 구문적 변수 영역 규칙을 적용할 수 있음
var topic = "자바스크립트"

if (topic) {
  let topic = "리액트"
  console.log('블록', topic)
}
console.log('글로벌', topic)
```


let과 const는 블록 스코프

블록 리액트
글로벌 자바스크립트

※ let과 const는 재선언과 호이스팅을 허용하지 않음

변수 var와 let의 scope 차이 (예)

```
<h1>박스를 클릭하세요</h1>
<div id="container"></div>
```

: i의 scope

```
/* ch02-01-05-let.html */
// 문제: 카운터 i 영역이 for 블록 안으로만 제한되지 않음
var div,
    container = document.getElementById('container')

for (var i=0; i<5; i++) {
  div = document.createElement('div')
  div.onclick = function() {
    alert('이것은 박스 #' + i + '입니다.')
  }
  container.appendChild(div)
} #5 #3 #5 #5 #5
```

```
/* ch02-01-06-let.html */
// Solution, lexical variable scope
var div,
    container = document.getElementById('container')

for (let i=0; i<5; i++) {
  div = document.createElement('div')
  div.onclick = function() {
    alert('이것은 박스 #' + i + '입니다.')
  }
  container.appendChild(div)
} #0 #1 #2 #3 #4
```

var 대신 let 사용

박스를 클릭하세요



```
<h1>박스를 클릭하세요</h1>
<div id="container"> flex
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

id가 container인 div 객체 하위에서
5번의 루프를 돌면서 새로운 div 객체를 생성
div 객체에 카운터 i를 활용하는 onclick() 이벤트 리스너 구현
카운터 i의 scope 확인

문자열 연결과 템플릿 문자열 (계속)

- 문자열을 연결하기 위해 사용하는 + 연산자는 가독성이 떨어짐
 - ES2015부터 ` (백틱) 사용 가능
 - ` ` 안에 ` 문자열과 \${계산식} ` 을 혼용



문자열 + 문자열

```
/* ch02-01-07-template-strings.html */
```

```
const lastName = "Oh"
const middleName = "현석"
const firstName = "Frank"
```

문자열 연결 연산자: +

```
console.log(lastName + ", " + firstName + " " + middleName)
```

귀찮음

Oh, Frank 현석

` \${expression} `

계산

```
/* ch02-01-08-template-strings.html */
```

```
const lastName = "Oh"
const middleName = "현석"
const firstName = "Frank"
```

템플릿 문자열

```
console.log(`${lastName}, ${firstName} ${middleName}`)
```

Oh, Frank 현석

템플릿 문자열 활용 예 (계속)

```
/* ch02-01-09-template-strings.html */
```

```
var lastName = "Oh"  
var middleName = "현석"  
var firstName = "Frank"  
var ticketAgent = "예술의 전당"  
var event = "서태지와 아이들"  
var qty = 2  
var price = 10
```

```
console.log(`  
  ${firstName} 님께,  
  ${event} 티켓 ${qty} 건을 구매해주셔서 감사합니다.
```

템플릿 문자열은 공백이나 줄바꿈을 유지

주문 상세 정보:

```
  ${lastName} ${firstName} ${middleName}  
  ${qty} x ${price} = ${qty*price} 공연: ${event}
```

공연 시작 30분 전까지 배부처에서 티켓을 수령하시기 바랍니다.
감사합니다.

```
  ${ticketAgent} 드림  
`)
```

Frank 님께,
서태지와 아이들 티켓 **2** 건을 구매해주셔서 감사합니다.

주문 상세 정보:

Oh Frank 현석
2 x \$10 = \$20 공연: 서태지와 아이들

공연 시작 30분 전까지 배부처에서 티켓을 수령하시기 바랍니다.
감사합니다.

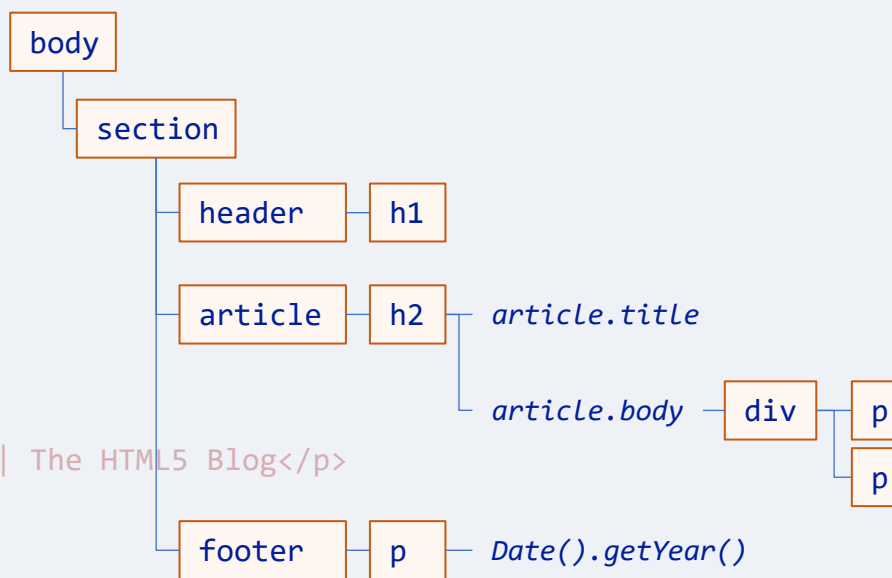
예술의 전당 드림

템플릿 문자열 활용 예

```
/* ch02-01-10-template-strings.html */
```

```
var article = {  
  title: "Template Strings",  
  body: `  
    <div>  
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor</p>  
      <p> laboris nisi ut aliquip ex ea commodo consequat.</p>  
    </div>  
  `  
}
```

```
document.body.innerHTML = `  
<section>  
  <header>  
    <h1>The HTML5 Blog</h1>  
  </header>  
  <article>  
    <h2>${article.title}</h2>  
    ${article.body}  
  </article>  
  <footer>  
    <p>copyright ${new Date().getFullYear()} | The HTML5 Blog</p>  
  </footer>  
</section>  
`
```



The HTML5 Blog

Template Strings

Lorem ipsum dolor sit amet, consectetur
laboris nisi ut aliquip ex ea commodo co
copyright 123 | The HTML5 Blog

함수: 선언과 호출, 함수 표현식, 호이스팅

```
/* ch02-02-00-1-function-declare.html */
```

```
function logCompliment(){  
    console.log('Good job!');  
}
```

함수 선언

```
logCompliment();
```

함수 호출

Good job!

```
/* ch02-02-00-2-function-declare.html */
```

```
const logCompliment = function() {  
    console.log('Good job!');  
}
```

익명함수를 이용한 함수 표현식
함수 객체를 변수나 상수에 할당

```
logCompliment();
```

함수 호출

Good job!

- 진체 코드를 꼭 훑어보고,
- **호이스팅**: 함수/변수 선언을 유효 scope의 최상단으로 이동

```
/* ch02-02-00-3-function-declare.html */
```

```
logCompliment();
```

함수 선언 전 호출: 가능

```
function logCompliment(){  
    console.log('Good job!');  
}
```

함수 선언

Good job!

함수 선언 전에도 호출 가능

```
/* ch02-02-00-4-function-declare.html */
```

```
logCompliment();
```

함수 선언 전 호출: 에러

```
const logCompliment = function() {  
    console.log('Good job!');  
}
```

함수 표현식

Uncaught TypeError

함수 표현식은 호이스팅 허용하지 않음

TypeError: logCompliment is not a function

교재: 함수를 선언하기 전 호출하여 오류가 발생하였다면, 함수를 선언으로 리팩터링 하라.

>> 함수 표현식 사용을 권장. 에러가 발생하였다면, 의도하지 않은 에러인지 먼저 검증할 것

함수: default parameters

```
/* ch02-02-01-default-parameters.html */
function logActivity(name="오성원", activity="테니스") {
    console.log( `${name}은 ${activity}를 좋아합니다.` )
}

logActivity()
```

별도로 매개변수를 전달하지 않아도
default parameters로 미리 정의된 매개변수 활용

매개변수 *parameters*를 전달하고,
값을 반환 *return* 하는 것은 기존 언어와 동일

오성원은 테니스를 좋아합니다.

```
/* ch02-02-02-default-parameters.html */
var defaultPerson = {
    name: {
        first: "성원",
        last: "오"
    },
    favActivity: "테니스"
}

function logActivity(p=defaultPerson) {
    console.log( `${p.name.first}은 ${p.favActivity}를
    좋아합니다.` )
}

logActivity()
```

성원은 테니스를 좋아합니다.

화살표 함수 (계속)

- 함수 표현식을 좀 더 간결하게 활용 (function, return 키워드 등 생략 가능)

```
const func_name = (param1, param2, ..., paramN) => { func_body };
```

```
/* ch02-03-01-arrows.html */
```

```
// 일반 함수
```

```
var lordify = function(firstname) {  
    return `켄터베리의 ${firstname}`  
}
```

```
console.log( lordify("오성원") )  
console.log( lordify("오정원") )
```

```
/* ch02-03-03-arrows.html */
```

```
// 일반 함수
```

```
var lordify = function(firstName, land) {  
    return `${land}의 ${firstName}`  
}
```

```
console.log( lordify("오성원", "브리즈번") )  
console.log( lordify("오정원", "시드니") )
```

```
/* ch02-03-02-arrows.html */
```

```
// 화살표 함수
```

```
var lordify = firstname => `켄터베리의 ${firstname}`
```

함수 객체 이름

매개변수

함수 정의 (반환 값)

```
console.log( lordify("오성원") )  
console.log( lordify("오정원") )
```

매개변수가 2개 이상인 경우 괄호() 사용
1개인 경우 생략 가능

```
/* ch02-03-04-arrows.html */
```

```
// 화살표 함수
```

```
var lordify = (firstName, land) => `${land}의 ${firstName}`
```

반환 값을 포함하여 함수의 body를 1줄로 표현할 수 있는 경우
return 키워드 생략 가능

```
console.log( lordify("오성원", "브리즈번") )  
console.log( lordify("오정원", "시드니") )
```

1줄 짜리 함수의 body는 {} 도 생략 가능

화살표 함수 필요한 파라미터보다 적게 전달한다면?

```
/* ch02-03-05-arrows.html */

// 일반 함수
var lordify = function(firstName, land) {
  if (!firstName) {
    throw new Error('lordify에 이름을 넘겨야 합니다.')
  }
  if (!land) {
    throw new Error('영주에게는 영지가 있어야 합니다.')
  }
  return `${land}의 ${firstName}`
}

console.log( lordify("이계영", "멜버른") )
console.log( lordify("오현석") )
```

멜버른의 이계영

Uncaught Error: 영주에게는 영지가 있어야 합니다.

```
/* ch02-03-06-arrows.html */

// 화살표 함수
var lordify = (firstName, land) => {
  if (!firstName) {
    throw new Error('lordify에 이름을 넘겨야 합니다.')
  }
  if (!land) {
    throw new Error('영주에게는 영지가 있어야 합니다.')
  }
  return `${land}의 ${firstName}`
}

console.log( lordify() )
```

화살표 함수의 일반적인 형태

Uncaught Error: lordify에 이름을 넘겨야 합니다.

에러를 던졌으나 받아서 처리하는 코드는 누락

- > 프로그램의 비정상 종료를 의미
- > 에러처리 코드도 함께 추가하는 것을 권장
- > 종료되는 코드 이후에 디버깅 코드를 추가하여 확인

화살표 함수와 this (계속)

ch02-03-07, ch02-03-08 코드에 테스트 코드 추가

```
/* ch02-03-07-arrows-test1.html */
console.log(this); // undefined

// 함수와 bind를 사용
const gangwon = {
  resorts: ["용평", "평창", "강촌", "강릉", "홍천"],
  print: function(delay=1000) {
    console.log(this); // gangwon을 참조
    setTimeout(function() {
      console.log(this); // Window
      console.log(this.resorts.join(","))
    }, delay)
  }
}

// 객체의 메서드 내부에 또 함수가 포함됨
// . 메서드 내부의 this는 객체를 참조
// . 메서드 내부의 콜백함수에서 this는 Window 객체 참조

gangwon.print()
```

```
undefined
{resorts: Array(5), print:f}
Window {window: Window, self: Window, ...}
Uncaught TypeError:
Cannot read properties of undefined (reading: 'join')
```

```
/* ch02-03-08-arrows-test2.html */
console.log(this); // undefined

// 함수와 bind를 사용
const gangwon = {
  resorts: ["용평", "평창", "강촌", "강릉", "홍천"],
  print: function(delay=1000) {
    console.log(this); // gangwon
    setTimeout(function() {
      console.log(this); // gangwon
      console.log(this.resorts.join(","))
    }.bind(this), delay) // this를 gangwon에 바인딩
  }
}

// 객체의 메서드 내부에 포함된 함수의 this는 Window 객체 참조
// 객체의 메서드 내부의 this (gangwon 객체)로 바인딩하여 오류 방지

gangwon.print()
```

```
undefined
{resorts: Array(5), print:f}
{resorts: Array(5), print:f}
용평, 평창, 강촌, 강릉, 홍천
```

콜백함수 밖에 있는 함수 (bind)의 this는
동일한 스코프, 즉 bind(gangwon)

화살표 함수와 this (계속)

화살표 함수의 스코프의 특징은, 화살표 함수가 포함된 내부에 국한되지 않고, 스코프를 무시한다

```
/* ch02-03-09-arrows.html */
```

```
// 화살표 함수를 사용함
```

```
const gangwon = {  
  resorts: ["용평", "평창", "강촌", "강릉", "홍천"],  
  print: function(delay=1000) {  
    console.log(this); // this: gangwon  
    setTimeout(  
      () => console.log(this.resorts.join(", ")),  
      delay  
    )  
  }  
}
```

화살표 함수의 scope를 무시하고, this는 동일한 객체 참조

```
gangwon.print()
```

```
{resorts: Array(5), print: f}
```

```
용평, 평창, 강촌, 강릉, 홍천
```

```
/* ch02-03-10-arrows.html */
```

```
console.log(this); // undefined
```

```
// 객체의 메서드를 화살표 함수로 사용
```

```
const gangwon = {  
  resorts: ["용평", "평창", "강촌", "강릉", "홍천"],  
  print: (delay=1000) => {  
    console.log(this); // undefined  
    setTimeout(() => {  
      console.log(this); // undefined  
      console.log(this.resorts.join(", "))  
    }, delay)  
  }  
}
```

객체의 메서드를 화살표 함수로 사용
scope 무시

매개변수를 화살표 함수로 사용, scope 무시

```
gangwon.print()
```

```
undefined
```

```
undefined
```

```
undefined
```

```
Uncaught TypeError:
```

```
Cannot read properties of undefined (reading 'resorts')
```

화살표 함수와 this (계속)

```
/* ch02-03-11-arrows.html */
```

```
console.log(this)           // undefined
```

```
// 언제 화살표 함수를 사용해야 할지 알고 사용할 것
```

```
const gangwon = {  
  resorts: ["용평", "평창", "강촌", "강릉", "홍천"],  
  print: function(delay=1000) {  
    console.log(this)      // gangwon  
    setTimeout(() => {  
      console.log(this)    // gangwon  
    }, delay)  
  }  
}
```

화살표 함수의 scope를 무시하고, this는 동일한 객체 참조

```
gangwon.print();
```

```
undefined
```

```
{resorts: Array(5), print: f}
```

```
{resorts: Array(5), print: f}
```

참조: 화살표 함수와 this (계속)

```
/* ch02-03-12-arrows-test.html */
```

```
const obj = {  
  obj_attr: 'obj_attr',  
  
  func: function () {  
    console.log('obj.func(): ');  
    console.log(this);  
  },  
  
  const funcFunc1 = function () {  
    console.log('obj.func(funcFunc1()): w/o binding');  
    console.log(this);  
  };  
  funcFunc1();  
  
  const funcFunc2 = () => {  
    console.log('obj.func(funcFunc2()): arrow funcFunc2()');  
    console.log(this);  
  };  
  funcFunc2();  
  
  const funcFunc3 = function () {  
    console.log('obj.func(funcFunc3().bind(this)):');  
    console.log(this);  
  }.bind(this);  
  funcFunc3();  
},
```

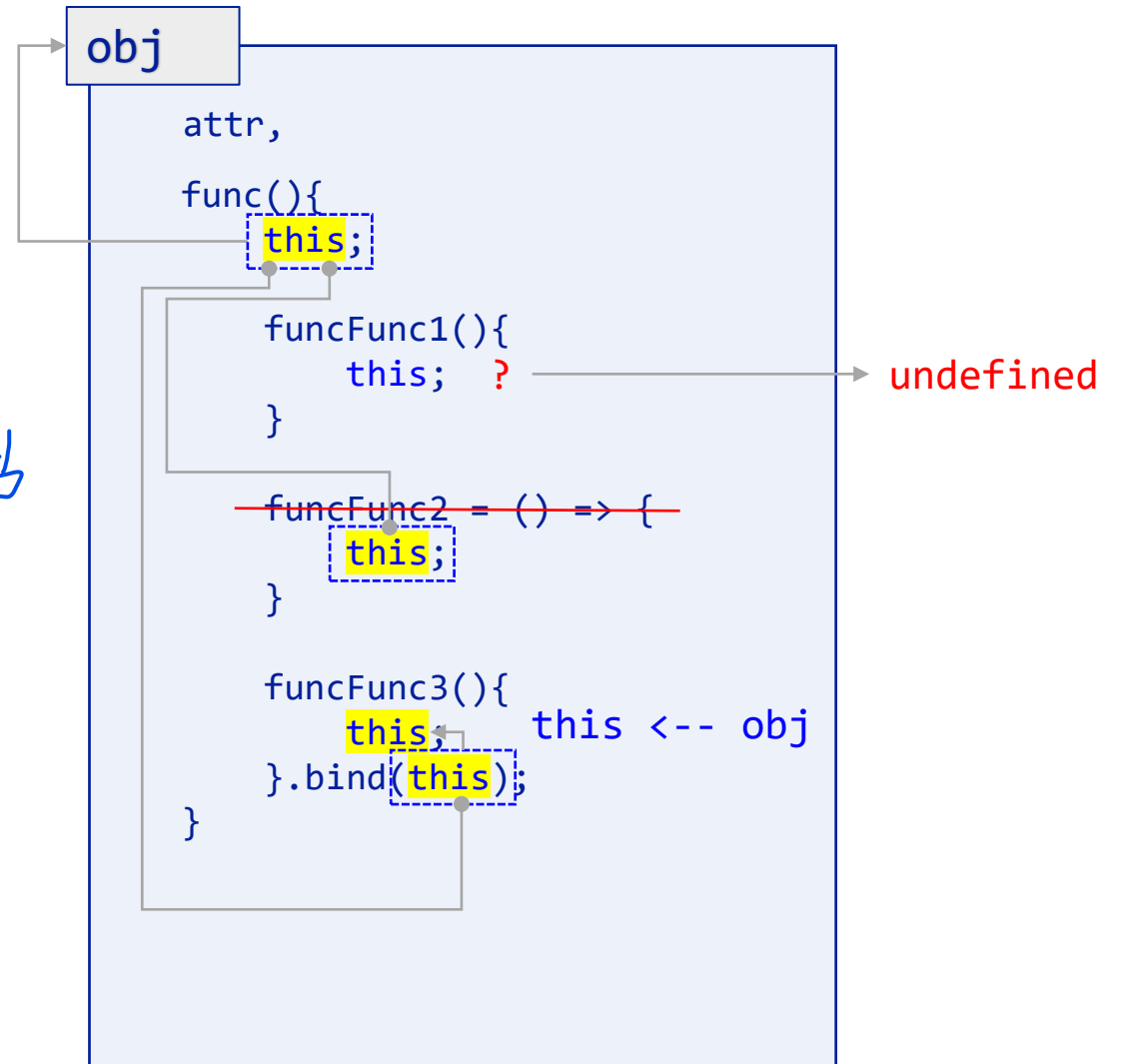
// obj //

// undefined

콜백함수 아님

// obj

// bind(this <-- obj) //



참조: 화살표 함수와 this

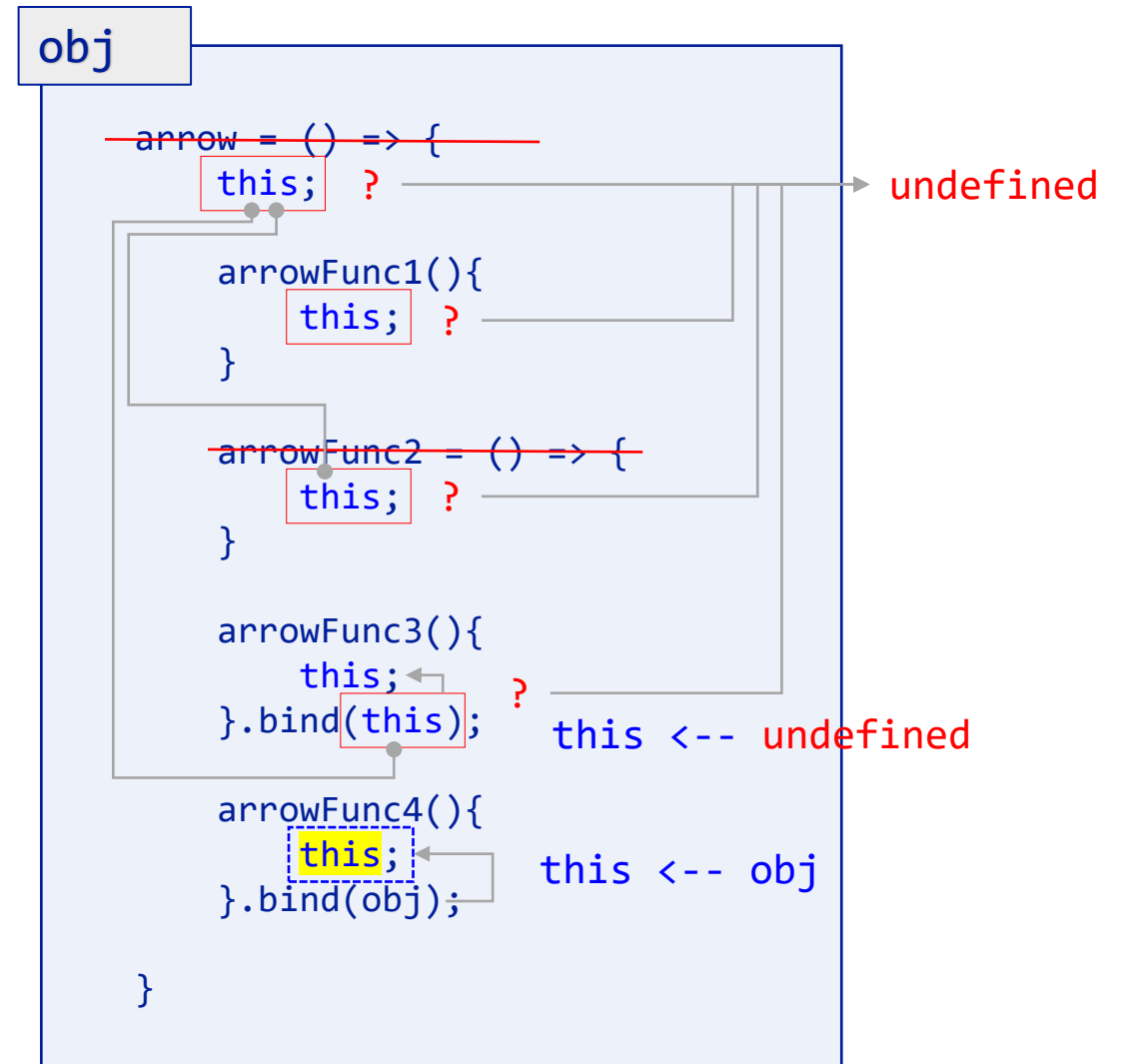
```
arrow: () => { /* ch02-03-12-arrows-test.html */
  console.log('obj.arrow(): ');
  console.log(this); // undefined

  const arrowFunc1 = function () {
    console.log('obj.arrow(arrowFunc1()): ');
    console.log(this); // undefined
  };
  arrowFunc1();

  const arrowFunc2 = () => {
    console.log('obj.arrow(arrowFunc2()): arrow arrowFunc2()');
    console.log(this); // undefined
  };
  arrowFunc2();

  const arrowFunc3 = function () {
    console.log('obj.arrow(arrowFunc3().bind(this)): ');
    console.log(this); // undefined
  }.bind(this); // bind(this <-- undefined)
  arrowFunc3();

  const arrowFunc4 = function () {
    console.log('obj.arrow(arrowFunc4().bind(obj)): ');
    console.log(this); // obj
  }.bind(obj); // bind(this <-- obj)
  arrowFunc4();
},
obj.func();
obj.arrow();
```



정리: 1. 리액트 소개 & 2. 리액트를 위한 자바스크립트

- React 개념

- JavaScript 라이브러리
- UI 구현을 위한 View 담당

- ES2015+

- 변수와 상수
- 템플릿 문자열

- ES2015+ (계속)

- 함수: 선언과 호출, 함수표현식, 호이스팅, default parameters
- 화살표 함수: 사용법과 this의 참조 범위
- 구조분해 할당: 변수, 객체
- 객체 리터럴
- 스프레드 연산자
- 비동기 자바스크립트: promise, fetch, async/await
- 클래스