1~2. 리액트 학습 준비 (실습2)

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

Division of Computer Engineering

한성대학교 HANSUNG UNIVERSITY

학습 목표: 2. 리액트를 위한 자바스크립트

- ES2015+
  - 변수와 상수
  - 함수
  - 화살표 함수
  - 객체와 this
  - script type 확인
  - 비동기 자바스크립트 구현

실습5: 객체와 this (1)

- HTML 내부 script 영역 안에 다음의 문제를 순차적으로 구현

(1) 전역 scope에서 **this** 값 확인

(2) **obj** 객체 생성

- (3) **obj**의 속성 **obj_this**에 **'str1'** 값 할당
- (4) **obj**의 메서드 **func1()** 정의
- 메서드 **func1()**의 내부 구현
  - (5) **this**가 참조하는 값 콘솔에 출력
  - (6) 함수 표현식으로 (5)를 구현하는 함수 **funcFunc1()**를 구현하고 호출
  - (7) 화살표 함수로 (5)를 구현하는 함수 **funcFunc2()**를 구현하고 호출
  - (8) 함수 표현식으로 (5)를 구현하는 함수 **funcFunc3()**에, **obj**를 바인딩하고 호출
  - (9) **setTimeout()**의 콜백에서 **this 값 확인** (콜백은 함수 선언문, **timeout**은 **0**)
  - (10) **setTimeout()**의 콜백에서 **this 값 확인** (콜백은 화살표 함수, **timeout**은 **0**)

```
console this;

const obj = {

  obj_this: 'str1',

  func1(){
    console this;

    // 함수 표현식으로 funcFunc1() 구현
    const funcFunc1 = ...  // 함수 표현식

    // 화살표 함수로 funcFunc2() 구현
    const funcFunc2 = ...  // 화살표 함수

     // 함수 표현식으로 구현하고 obj 바인딩
    const funcFunc3 = ...  // 함수 표현식, bind()

    setTimeout(callback, 0);

    setTimeout(callback, 0);
  },
};
```

실습5: 객체와 this (2)

- HTML 내부 script 영역 안에 다음의 문제를 순차적으로 구현

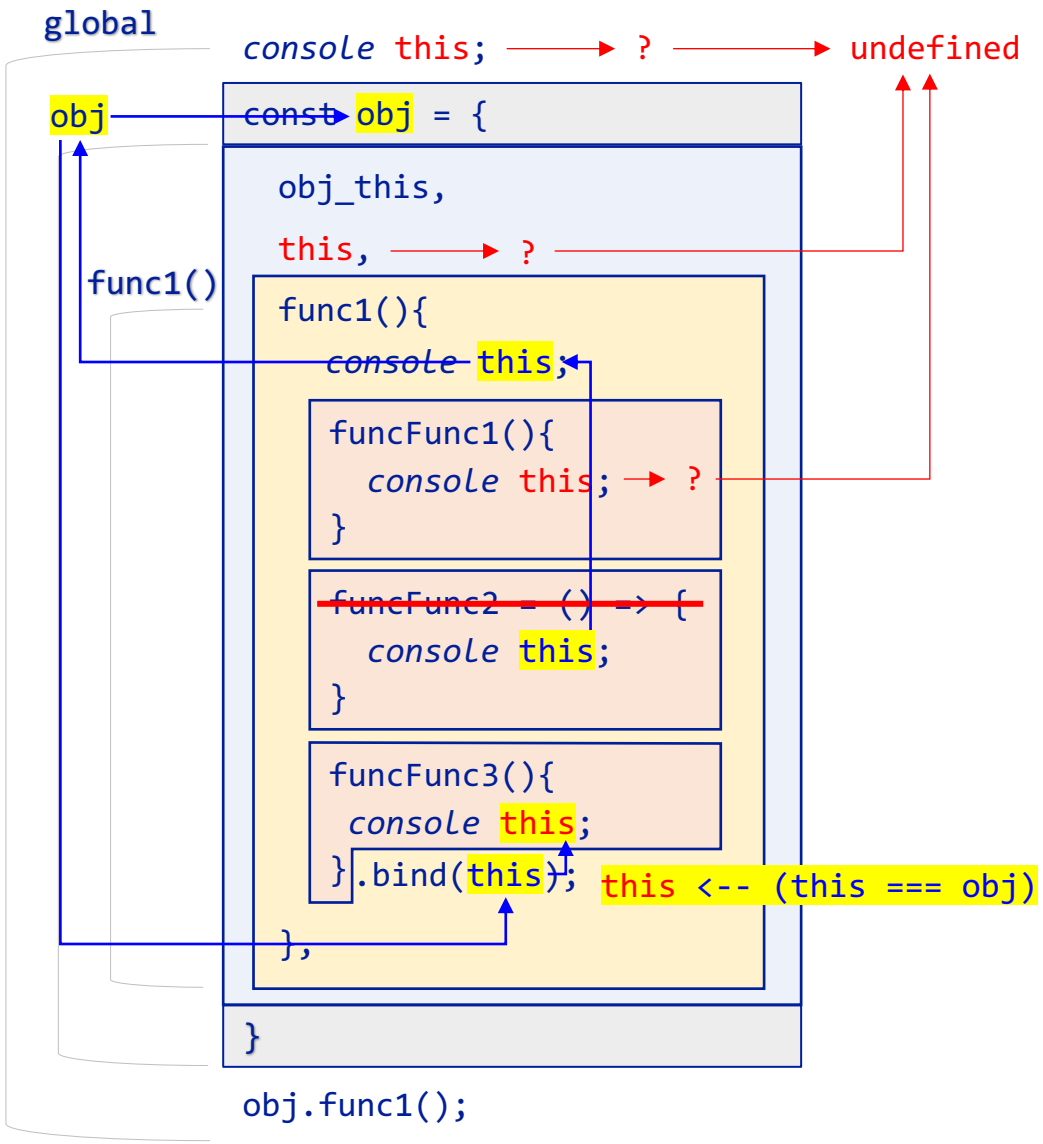
(1) 전역 scope에서 **this** 값 확인

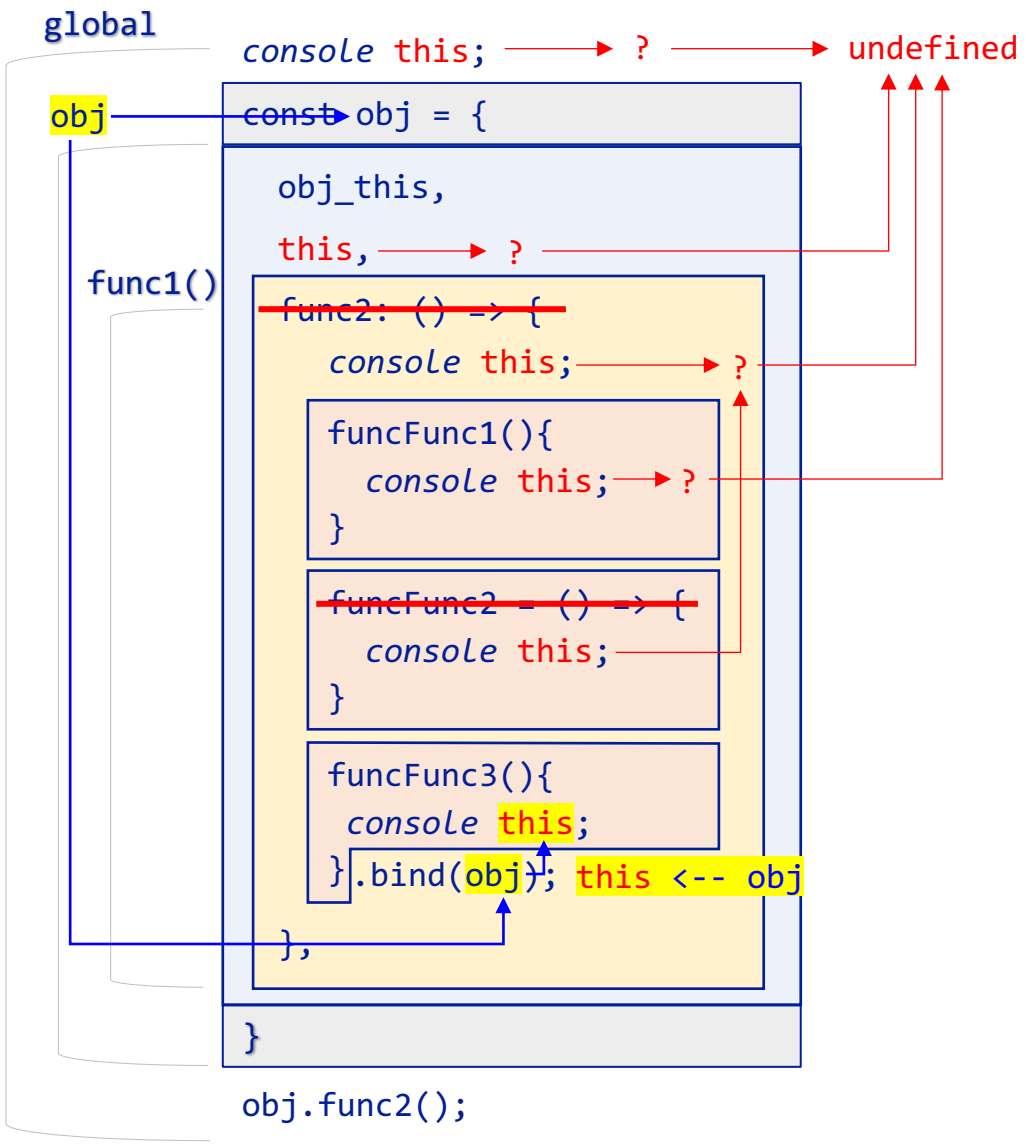
(2) **obj** 객체 생성

- (3) **obj**의 속성 **obj_this**에 **'str1'** 값 할당
- (4) **obj**의 메서드 **func2()**를 화살표 함수로 정의
- 메서드 **func1()**의 내부 구현
  - (5) **this**가 참조하는 값 콘솔에 출력
  - (6) 함수 표현식으로 (5)를 구현하는 함수 **funcFunc1()**를 구현하고 호출
  - (7) 화살표 함수로 (5)를 구현하는 함수 **funcFunc2()**를 구현하고 호출
  - (8) 함수 표현식으로 (5)를 구현하는 함수 **funcFunc3()**에, **obj**를 바인딩하고 호출
  - (9) **setTimeout()**의 콜백에서 **this 값 확인** (콜백은 함수 선언문, **timeout**은 **0**)
  - (10) **setTimeout()**의 콜백에서 **this 값 확인** (콜백은 화살표 함수, **timeout**은 **0**)

```
console this;

const obj = {

  obj_this: 'str1',

  func2: () => {
    console this;

    // 함수 표현식으로 funcFunc1() 구현
    const funcFunc1 = ...  // 함수 표현식

    // 화살표 함수로 funcFunc2() 구현
    const funcFunc2 = ...  // 화살표 함수

     // 함수 표현식으로 구현하고 obj 바인딩
    const funcFunc3 = ...  // 함수 표현식, bind()

    setTimeout(callback, 0);

    setTimeout(callback, 0);
  },
};
```

실습5: 객체와 this


global
console this; → ? → undefined
obj
const obj = {
obj_this,
this, → ?
func1()
func1(){
console this;
funcFunc1(){
console this; → ?
}
funcFunc2 = () => {
console this;
}
funcFunc3(){
console this;
}.bind(this);
this <-- (this === obj)
},
}
obj.func1();


global
console this; → ? → undefined
obj
const obj = {
obj_this,
this, → ?
func1()
func2: () => {
console this; → ?
funcFunc1(){
console this; → ?
}
funcFunc2 = () => {
console this;
}
funcFunc3(){
console this;
}.bind(obj);
this <-- obj
},
}
obj.func2();

실습5: 객체와 this (3)

- 실습 5-1과 5-2의 코드에서 script type을 변경하여 확인

(1) `<script type="module">`

(2) `<script type="text/babel">`

(3) `<script type="text/javascript">`

(4) `<script>`

실습6: script type 확인

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>What It Means to Be Functional</title>
  <script src="https://unpkg.com/babel-
standalone@6.15.0/babel.min.js"></script>
</head>
<body>
<h1>Functional JavaScript</h1>

<script type="module">
/* 06-type-module.html */

a = 5;
console.log(a);

console.log(this);

</script>
</body>
</html>
```

```
<script type="module">
<script type="text/babel">
```

```
<script type="text/javascript">
<script>
```

```
/* ch03-02.js */

a = 5;
console.log(a);

console.log(this);
```

```
'use strict'
```