`this` in Javascript

> "함수 호출 방식"에 의해 결정되는 this

- 자바스크립트의 함수는 호출될 때 `this`를 암묵적으로 전달 받음
- java에서의 this와 python에서의 self는 인스턴스 자신을 가리킴(참조)
- 자바스크립트에서의 this keyword는 일반적인 프로그래밍 언어에서의 this와 조금 다르게 동작
- 자바스크립트는 해당 **함수 호출 방식**에 따라 this에 바인딩 되는 객체가 달라짐
- 즉, 함수를 선언할 때 this에 객체가 결정되는 것이 아니고, 함수를 호출할 때 **함수가 어떻게 호출 되었는지에 따라 동적으로 결정**됨

다양한 함수 호출 방식

1. 함수 호출

```
const app = function () {
  console.log(this)
}

app() // window
```

2. 메서드 호출

```
const myObj = {
    app: app,
    // app (ES6)
}

myObj.app() // myObj
```

3. 생성자 함수 호출

```
const myapp = new app() // myapp instance
```

함수 호출

> 전역객체는 모든 객체의 유일한 최상위 객체를 의미하며
> 일반적으로 Browser에서는 `window`

```
const greeting = function() {
    console.log(this)
}

greeting() // window
```

- 기본적인 함수 선언을 하고 호출한다면, 이 경우 전역에서 호출했으므로 전역 객체가 바인딩

메서드 호출

```
const you = {
  name: 'kim',
  greeting,
}

you.greeting() // {name: 'kim', greeting: f} : this는 해당 오브젝트(객체)
```

- 메서드로 선언하고 호출한다면, 객체의 메서드이므로 해당 객체가 바인딩

Arrow function

> 화살표 함수는 호출의 위치와 상관없이 상위 스코프를 가리킴 (Lexical scope this)
>
> **Lexical scope**
>
> - 함수를 어디서 호출하는지가 아니라 **어디에 선언**하였는지에 따라 결정
> - Static scope라고도하며 대부분의 프로그래밍 언어에서 따르는 방식

```
const arrowGreeting = () => {
  console.log(this)
}

const me = {
  name: 'me',
  arrowGreeting,
}

arrowGreeting() // window
me.arrowGreeting() // window
```

- 따라서, 메서드 선언을 화살표 함수로 하게된다면 해당 객체의 상위 스코프인 전역 객체 window가 바인딩 됨
- arrowGreeting() 함수는 애초에 전역에서 선언되었기 때문
- **메서드 선언은 function 키워드를 사용하자**

그렇다면 ES6에서 화살표 함수는 언제 활용하는 것이 좋을까?

`함수 내의 함수` **상황을 예시로 비교해보자**

```
// function 키워드
```

```
const num = {
  numbers: [1],
  print: function() {
    console.log(this) // {numbers: Array(1), print: f}
    console.log(this.numbers) // [1]
    this.numbers.forEach(function(num) {
      console.log(num) // 1
      console.log(this) // window
      })
    }
}

num.print()
```

```
// 화살표 함수

const num2 = {
  numbers: [1],
  print: function() {
    console.log(this) // {numbers: Array(1), print: f}
    console.log(this.numbers) // [1]
    this.numbers.forEach((num) => {
      console.log(num) // 1
      console.log(this) // {numbers: Array(1), print: f}
      })
    }
}

num2.print()
```

> forEach 문 콜백함수의 두번째 console.log() 주목

- print 메서드 내에 있는 forEach의 콜백함수에서의 상위 스코프는 num2 객체
- 즉, 해당위치에서 일반적으로 this는 num2 객체를 가리켜야하나 function 표현은 window를 가리킴
- 호출의 위치와 상관없이 상위 스코프를 가리키는 화살표함수는 this에 num2 객체가 바인딩 됨
- 따라서 함수 내의 함수 상황에서 위와 같이 화살표 함수를 쓰는 것이 좋다.

이벤트 리스너

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <button id="function">function</button>
  <button id="arrow">arrow function</button>

  <script>
```

```
    const functionButton = document.querySelector('#function')
    const arrowButton = document.querySelector('#arrow')

    functionButton.addEventListener('click', function(event) {
      console.log(this) // <button id="function">function</button>
    })

    arrowButton.addEventListener('click', event => {
      console.log(this) // Window
    })
  </script>
</body>
</html>
```

- addEventListener 에서의 콜백 함수는 특별하게 function 키워드의 경우에는 이벤트 리스너를 호출한 대상을(event.target) 뜻함
- 따라서, 호출한 대상을 원한다면 this 를 활용할 수 있음
- 다만, 화살표 함수의 경우 상위 스코프를 지칭하기 때문에 window 객체가 바인딩 됨
- **이벤트 리스너의 콜백 함수는 function 키워드를 사용하자**