

**this 교안 풀이**

**박재영**

# this

함수가 자신이 속한 객체를 가리킬때 사용하는 변수

JavaScript에서의 this의 값은 대부분 함수를 호출한 방법(시점)에 의해 동적으로 결정된다

엄격 모드, 비엄격 모드에서도 일부 차이가 있으며 함수를 호출할 때마다 다를 수 있다

함수 호출 방식에 상관 없이 this를 설정할 수 있는 bind 메서드도 존재하고  
this바인딩이 없는 화살표 함수도 있다.

# this

this의 값은 대부분 함수를 호출한 방법(시점,문맥)에 의해 동적으로 결정된다.

엄격 모드

'use strict'

Global context

window

Function context

일반 함수 호출

중첩 함수

화살표 함수

메서드 & 프로토타입 메서드

생성자 함수

이벤트 처리기

콜백함수

bind,call,apply 메서드

접근자, 생성자

global context

# global context

가장 기본적으로 this에는 전역 객체가 연결 된다  
즉, this는 window를 가리키는게 기본이다.

```
console.log( this === window ) // true
```

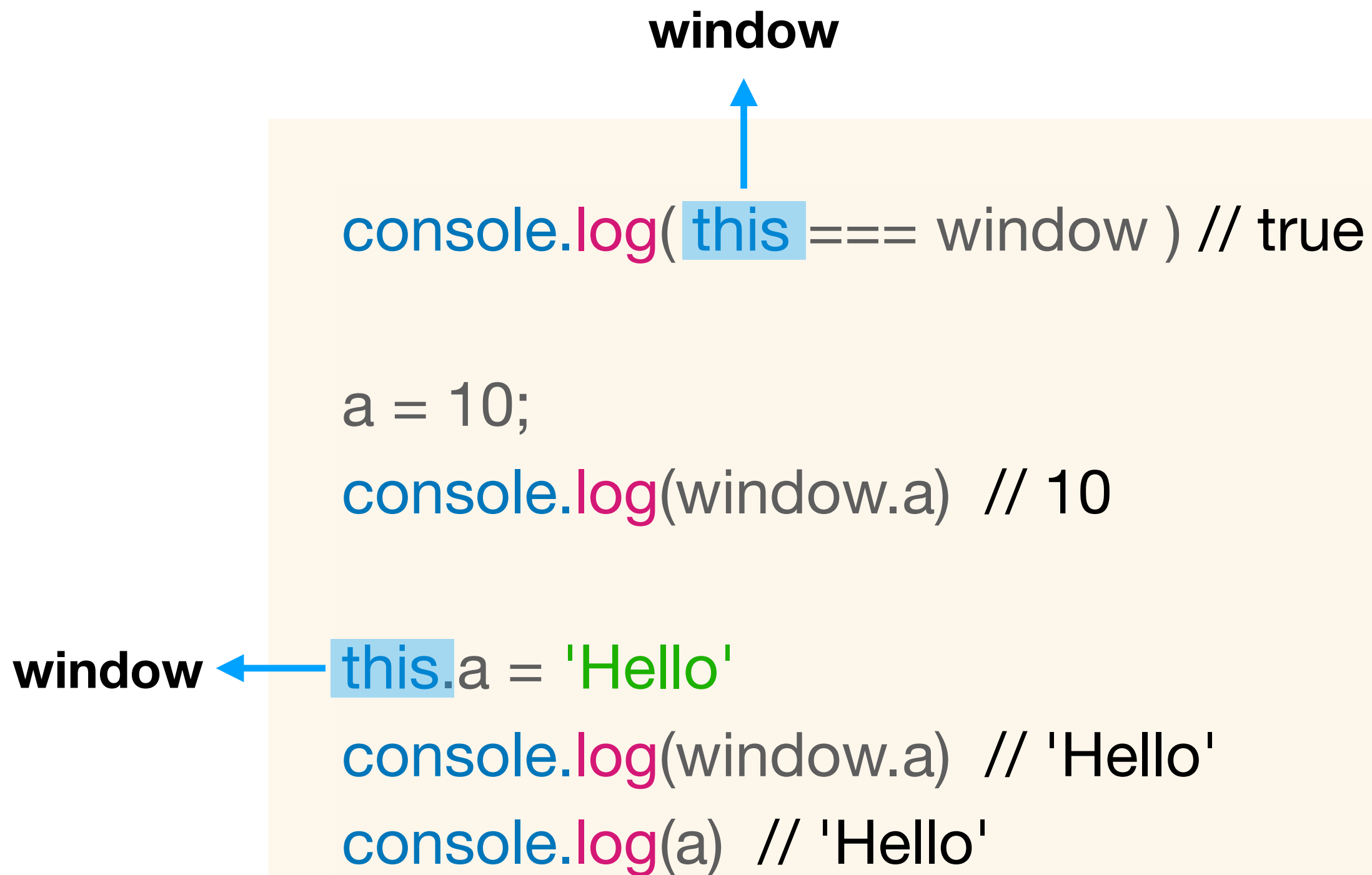
```
a = 10;  
console.log(window.a) // 10
```

```
this.a = 'Hello'  
console.log(window.a) // 'Hello'  
console.log(a) // 'Hello'
```

# global context

가장 기본적으로 this에는 전역 객체가 연결 된다  
즉, this는 window를 가리키는게 기본이다.

**window**



```
console.log( this === window ) // true

a = 10;
console.log(window.a) // 10

window ← this.a = 'Hello'
console.log(window.a) // 'Hello'
console.log(a) // 'Hello'
```

# global context

추가로, 객체 리터럴은 기본적으로 this 스코프를 만들지는 않는다  
즉, 객체 내의 함수(메서드)만 this 스코프를 만들어내기 때문이다.

```
const obj = {  
  a : this → window  
};
```

```
console.log(obj.a === window)  
// true
```

없음 ←

```
const obj2 = {  
  this.a : 10  
};
```

```
// Uncaught SyntaxError:  
Unexpected token '.'
```

# global context

추가로, 객체 리터럴은 기본적으로 this 스코프를 만들지는 않는다  
즉, 객체 내의 함수(메서드)만 this 스코프를 만들어내기 때문이다.

```
const obj = {  
  a : this → window  
};
```

```
console.log(obj.a === window)  
// true
```

없음 ←

```
const obj2 = {  
  this.a : 10  
};
```

```
// Uncaught SyntaxError:  
Unexpected token '.'
```



**function context**

# function context

함수 내부에서 this는 함수를 호출한 방법에 의해 결정된다

1. 단순 호출
2. 객체의 메서드로서 호출
3. 중첩 함수
4. 화살표 함수

...

이 외에도 여러 방식에 따른 규칙이 있는데  
다 설명하기엔 너무 많아서 큰 틀만 준비해봤습니다

# 단순 호출

전역 함수는 일반 함수로 호출하면 함수 내부의 **this**에는 전역 객체가 연결된다

그리고 어떤 함수라도 일반 함수로 호출되면 this에 전역 객체가 연결된다. (중첩함수, 콜백함수)

```
function a()  
{  
  console.log( this )  
  return this;  
}  
  
console.log( a() === window )
```

# 단순 호출

전역 함수는 일반 함수로 호출하면 함수 내부의 **this**에는 전역 객체가 연결된다

그리고 어떤 함수라도 일반 함수로 호출되면 **this**에 전역 객체가 연결된다. (중첩함수, 콜백함수)

```
function a()  
{  
  console.log( this )  
  return this;  
}
```

일반함수로서의 호출

함수 내부의 **this**가  
전역 객체에 연결

```
console.log( a() === window )  
// window  
// true
```



```
function a()  
{  
  console.log( window )  
  return window;  
}
```

```
console.log( a() === window )  
// window  
// true
```

# 객체의 메서드로서 호출

메서드 내부의 **this**는 메서드를 호출한 객체에 연결된다.  
메서드를 소유한 객체가 아니라 메서드는 호출한 객체에 연결된다

```
let myObj = {  
  val1 : 100,  
  func1 : function () {  
    console.log( this.val1 );  
  }  
}  
  
myObj.func1()
```

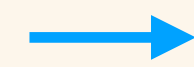
# 객체의 메서드로서 호출

메서드 내부의 **this**는 메서드를 호출한 객체에 연결된다.  
메서드를 소유한 객체가 아니라 메서드를 호출한 객체에 연결된다

```
let myObj = {  
  val1 : 100,  
  func1 : function () {  
    console.log( this.val1 );  
  }  
}
```

myObj.func1()

메서드로서 호출



메서드내부의 **this**가  
메서드를 소유한 객체에 연결



출력

100

# 객체의 메서드로서 호출

메서드 내부의 this는 메서드를 호출한 객체에 연결된다.  
메서드를 소유한 객체가 아니라 메서드를 호출한 객체에 연결된다

```
let myObj = {  
  val1 : 100,  
  func1 () {  
    console.log( this.val1 );  
  }  
}
```

```
let test = myObj.func1  
test()
```

# 객체의 메서드로서 호출

메서드 내부의 this는 메서드를 호출한 객체에 연결된다.  
메서드를 소유한 객체가 아니라 메서드를 호출한 객체에 연결된다

```
let myObj = {  
  val1 : 100,  
  func1 () {  
    console.log( this.val1 );  
  }  
}
```

myObj의 func1메서드를  
전역객체에 test프로퍼티 생성 (변수 선언)  
func1 복사 및 할당



```
let test = myObj.func1
```

```
test() → 전역 객체의 메서드로서 호출
```



출력

undefined



```
function sayName( ) {  
    console.log( this.name )  
}
```

```
var c = {  
    name : 'c',  
    'say' : sayName  
}
```

```
var b = {  
    name : 'b',  
    'c' : c  
}
```

```
var a = {  
    name : 'a',  
    'b' : b  
}
```

c.say( )

b.c.say( )

a.b.c.say( )

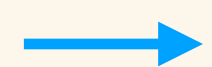
```
function sayName( ) {  
  console.log( this.name )  
}
```

```
var c = {  
  name : 'c',  
  'say' : sayName  
}
```

```
var b = {  
  name : 'b',  
  'c' : c  
}
```

```
var a = {  
  name : 'a',  
  'b' : b  
}
```

```
c.say()  
b.c.say()  
a.b.c.say()
```



모두 c 객체의 메서드로서 호출



출력

c  
c  
c

```
function sayName( ) {  
    console.log( this.name )  
}
```

```
var c = {  
    name : 'c',  
    'say' : sayName  
}
```

```
var b = {  
    name : 'b',  
    'say' : sayName  
}
```

```
var a = {  
    name : 'a',  
    'say' : sayName  
}
```

```
c.say( )
```

```
b.say( )
```

```
a.say( )
```

```
function sayName() {  
  console.log( this.name )  
}
```

```
var c = {  
  name : 'c',  
  'say' : sayName  
}
```

```
var b = {  
  name : 'b',  
  'say' : sayName  
}
```

```
var a = {  
  name : 'a',  
  'say' : sayName  
}
```

```
c.say()  
b.say()  
a.say()
```

→ 각 객체의 메서드로서 호출



출력

c  
b  
a

```
function attackBeam() {
```

```
  this.hp -= 20
```

```
}
```

```
function attackKnife() {
```

```
  this.hp -= 5
```

```
}
```

```
let jombie = {
```

```
  damaged: [attackBeam, attackKnife],
```

```
  hp: 10000,
```

```
}
```

```
jombie.damaged[0]()
```

```
jombie.damaged[1]()
```

```
function attackBeam() {  
  this.hp -= 20  
}  
function attackKnife() {  
  this.hp -= 5  
}
```

```
let jombie = {  
  damaged : [attackBeam, attackKnife],  
  hp : 10000,  
}
```

```
jombie.damaged[0]()  
jombie.damaged[1]()
```

객체 내의  
배열의 메서드로서  
호출



jombie.damaged 배열에 hp 프로퍼티 생성

```
jombie.damaged.hp -= 20  // undefined - 20 → NaN  
jombie.damaged.hp -= 5   // NaN - 20
```

```
jombie = {  
  damaged = [attackBeam, attackKnife, NaN ],  
  hp : 10000,  
}
```

# 중첩함수

중첩 함수를 일반 함수로 호출하면 함수 내부의 **this**에는 전역 객체가 연결된다

메서드 내에서 정의한 중첩 함수도 일반 함수로 호출되면 중첩 함수 내부의 **this**에는 전역 객체가 연결된다

```
function a() {  
  console.log( this )  
  function b() {  
    console.log( this )  
    function c() {  
      console.log( this )  
    }  
    c()  
  }  
  b()  
}  
a()
```

# 중첩함수

중첩 함수를 일반 함수로 호출하면 함수 내부의 **this**에는 전역 객체가 연결된다

메서드 내에서 정의한 중첩 함수도 일반 함수로 호출되면 중첩 함수 내부의 **this**에는 전역 객체가 연결된다

```
function a() {  
  console.log( this )  
  function b() {  
    console.log( this )  
    function c() {  
      console.log( this )  
    }  
  }  
}
```

```
c()  
}  
b()
```

→ 중첩함수로서 호출

함수 내부의 **this**가  
전역 객체에 연결

```
}
```

```
a()
```

→ 일반함수로서 호출

함수 내부의 **this**가  
전역 객체에 연결



출력

window  
window  
window



# 중첩함수

중첩 함수를 일반 함수로 호출하면 함수 내부의 this에는 전역 객체가 연결된다

메서드 내에서 정의한 중첩 함수도 일반 함수로 호출되면 중첩 함수 내부의 this에는 전역 객체가 연결된다

```
let person = {  
  name : 'hojun',  
  a() {  
    console.log( this )  
    function b() {  
      console.log( this )  
      function c() {  
        console.log( this )  
      }  
      c()  
    }  
    b()  
  }  
}  
person.a()
```

# 중첩함수

중첩 함수를 일반 함수로 호출하면 함수 내부의 `this`에는 전역 객체가 연결된다

메서드 내에서 정의한 중첩 함수도 일반 함수로 호출되면 중첩 함수 내부의 `this`에는 전역 객체가 연결된다

```
let person = {  
  name : 'hojun',  
  a() {  
    console.log( this.name )  
    function b() {  
      console.log( this.name )  
      function c() {  
        console.log( this.name )  
      }  
      c()  
    }  
    b()  
  }  
}
```

→ 중첩함수로서 호출

함수 내부의 `this`가  
전역 객체에 연결

`person.a()`

→ 메서드로서 호출

메서드의 `this`가  
메서드를 소유한 객체에 연결

출력

hojun

"

"

// window에는 기본적으로 name프로퍼티가  
빈 문자열로 존재한다

# 화살표 함수

화살표 함수는 기본적으로 this 바인딩이 존재하지 않는다  
그래서 스코프체인을 따라 상위 스코프에서 this를 검색하여 사용한다

```
let a = () => {  
  console.log( this )  
  let b = () => {  
    console.log( this )  
    let c = () => {  
      console.log( this )  
    }  
    c()  
  }  
  b()  
}  
a()
```

# 화살표 함수

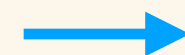
화살표 함수는 기본적으로 this 바인딩이 존재하지 않는다  
그래서 스코프체인을 따라 상위 스코프에서 this를 검색하여 사용한다

```
let a = () => {  
  console.log( this )  
  let b = () => {  
    console.log( this )  
    let c = () => {  
      console.log( this )  
    }  
  }  
}
```

```
c()  
}  
b()  
}  
a()
```



화살표 함수  
호출



상위 스코프에서  
this를 검색해서 사용



출력

window  
window  
window

# 화살표 함수

화살표 함수는 기본적으로 this 바인딩이 존재하지 않는다  
그래서 스코프체인을 따라 상위 스코프에서 this를 검색하여 사용한다

```
let person = {  
  name : 'hojun',  
  a() {  
    console.log( this.name )  
    let b = () => {  
      console.log( this.name )  
      let c = () => {  
        console.log( this.name )  
      }  
      c()  
    }  
    b()  
  }  
}  
person.a()
```

# 화살표 함수

화살표 함수는 기본적으로 this 바인딩이 존재하지 않는다  
그래서 스코프체인을 따라 상위 스코프에서 this를 검색하여 사용한다

```
let person = {  
  name : 'hojun',  
  a() {  
    console.log( this.name )  
    let b = () => {  
      console.log( this.name )  
      let c = () => {  
        console.log( this.name )  
      }  
      c()  
    }  
    b()  
  }  
}
```

c()  
}  
b()  
}

화살표 함수  
호출

상위 스코프에서  
this를 검색해서 사용

person.a()

메서드로서 호출

메서드의 this가  
메서드를 소유한 객체에 연결

출력

hojun  
hojun  
hojun

끝

reference : mdn, deepdive, 강의교안

## 추가) 콜백함수 예제

```
var age = 0;
function Person() {
  this.age = 0;

  setTimeout( function growUp() {
    this.age++;
  }, 1000);
}
var p = new Person()
```

코드가 실행되면 어떤 변화가 일어날지 생각해주세요

### 부연설명

콜백함수는 this에 대한 추가적인 처리를 해주지 않는 이상  
일반 함수 호출과 동일하게 처리한다

생성자 함수의 this는 자신이 생성할 인스턴스를 가리킨다



## 추가) 생성자 함수 예제

```
var age = 0;  
function a() {  
  this.b = function() {  
    console.log( this )  
  }  
}  
var obj = new a()  
obj.b()
```

출력이 어떻게 나올지 생각해주세요

### 부연설명

생성자 함수의 this는 자신이 생성할 인스턴스를 가리킨다

## 추가) 생성자 함수 예제

```
var age = 0;
function a() {
  this.b = function() {
    let f = function() {
      console.log( this )
    }
    f()
    console.log( this )
  }
}
var obj = new a()
obj.b()
```

출력이 어떻게 나올지 생각해주세요

### 부연설명

생성자 함수의 this는 자신이 생성할 인스턴스를 가리킨다

중첩 함수를 일반 함수로 호출하면  
함수 내부의 this에는 전역 객체가 연결된다

## 추가) 생성자 함수 예제

```
var age = 0;  
function a() {  
  this.b = function() {  
    (function() {  
      console.log( this )  
    })()  
    console.log( this )  
  }  
}  
var obj = new a()  
obj.b()
```

출력이 어떻게 나올지 생각해주세요

### 부연설명

생성자 함수의 this는 자신이 생성할 인스턴스를 가리킨다

중첩 함수를 일반 함수로 호출하면  
함수 내부의 this에는 전역 객체가 연결된다

## 추가) 생성자 함수 예제

```
var age = 0;
function a() {
  this.b = function() {
    let f = () => {
      console.log( this )
    }
    f()
    console.log( this )
  }
}
var obj = new a()
obj.b()
```

출력이 어떻게 나올지 생각해주세요

### 부연설명

생성자 함수의 this는 자신이 생성할 인스턴스를 가리킨다

화살표 함수의 this는 상위 스코프에서 검색된다