this

JavaScript 에는 `this`라는 키워드가 있다. this는 문맥에 따라 다양한 값을 가지는데, **this가 쓰이는 함수를 어떤 방식으로 실행하느냐에 따라 그 역할이 구별된다.** this의 값들은 크게 4가지 정도로 나눌 수 있다. 즉, this를 이용하는 함수를 4가지 방식 중에 어떤 방식으로 실행하느냐에 따라 this의 값이 결정된다는 뜻이다. 이러한 특성 때문에 this가 무엇을 지칭하는지 알기 위해서 우리는 this가 사용된 함수가 어디서 어떻게 실행되었는지를 찾아야만 한다.

1. 일반 함수 실행 방식 (Regular Function Call)

첫 번째는, 일반 함수 실행 방식으로 함수를 실행했을 때 this의 값은 글로벌 오브젝트를 가리킨다. 즉, 브라우저 상에서는 window 객체를 말한다. 일반 함수 실행 방식이란 아래 예제 코드처럼 우리가 함수를 선언한 후, 실행할 때 흔히 사용하는 방식을 말한다.

```
function foo() {
    console.log(this)
}

foo()
```

위 코드에서 foo() 와 같이 함수를 호출하는 방식을 일반 함수 실행 방식이라고 한다. 이 때, foo 함수 안에 있는 this는 글로벌 객체, 브라우저 상에서는 window 객체를 가리킨다.

```
let name = 'Julia'

function foo() {
    console.log(this.name) // 'Julia'
}
```

```
foo()
```

위 코드에서 name이란 변수는 전역 변수이기 때문에 전역 객체인 window에 속성으로 추가된다. 그렇기 때문에 this.name이 'Julia'가 되는 것이다.

```
var age = 100;

function foo() {
    var age = 99;
    bar(age)
}

function bar() {
    console.log(this.age)
}

foo()
```

위 코드에서 foo 함수 안에서 bar 함수가 실행되고, bar 함수는 this.age를 콘솔창에 출력한다. 이때, bar 함수는 foo 함수 내부에서 일반 함수 실행 방식으로 실행되고 있다. 그러므로 우리가 bar에 뭘 넘겨줬든, bar 함수 내부의 this.age는 window.age 이고, 이는 전역 변수로 선언된 age 변수의 값을 말한다. 그러므로 위 코드를 실행하면 100이 출력된다.

> 💡 Strict mode에서 일반 함수 실행 방식

Strict mode에서는 모든 코드들에 조금 더 엄격한 규칙들이 적용된다. strict mode를 사용하려면 스크립트 코드 맨 상단에 'use strict' 를 추가하면 된다.

보통 일반 함수 실행 방식에서 this는 window 객체를 가리킨다고 했다. 하지만 strict mode에서 this는 무조건 undefined 이다. 그렇기 때문에 아래 코드는 에러를 출력한다.

```
'use strict'

var name = 'Julia'

function foo() {
    console.log(this.name) // Error
}

foo()
```

2. 도트 표기법 (Dot Notation)

Dot Notation 이란 우리가 Object를 만들고 그 Object의 key와 value를 부여한 후 도트 (.) 로 값에 접근하는 방식을 말한다. 아래 예제를 보면 이해가 쉽다.

```
var age = 100;

var ken = {
    age: 35,
    foo: function () {
        console.log(this.age) // 35
    }
}

ken.foo()
```

ken 이라는 변수에 Object를 만들었다. 그리고 foo라는 key에 this.age를 출력하도록 함수를 만들었다. 이렇게 도트를 사용하여 객체 속성의 값에 접근하는 방식을 Dot Notation 이라고 한다.

그리고 이렇게 Dot Notation 으로 함수가 실행되면, **this는 그 도트 앞에 써있는 객체 자체를 가리킨다.**

```
function foo() {
    console.log(this.age)
}

var age = 100
var ken = {
    age: 36,
    foo: foo
}

var wan = {
    age: 32,
    foo: foo
}

ken.foo() // 36
wan.foo() // 32

var fn = ken.foo;
fn() // 100
```

ken.foo()와 wan.foo()는 모두 Dot Notation 방식으로 실행되었다. 하지만 fn = ken.foo 로 할당이 되었지만 실행이 fn() 으로 되었다. 이는 일반 실행 함수 방식이므로 이 때의 this는 Global Object를 가리키게 되고 전역 변수 age의 값인 100이 출력된다.

3. 명백한 바인딩 / call, bind, apply

명백한 바인딩, 즉 this의 역할을 우리가 직접 명확하게 지정해준다는 뜻이다. 이는 function.prototype.call, function.prototype.bind, function.prototype.apply 와 같은 메서드를 사용하여 할 수 있다.

```
var age = 100

function foo() {
    console.log(this.age)
}

var ken = {
    age: 35,
    log: foo
}

foo.call(ken, 1, 2, 3)
foo.apply(ken, [1, 2, 3])
```

위 코드에서 우리는 foo 함수에 call 메서드를 사용하여 실행하였는데, 인자로 ken, 1, 2, 3을 주었다. 이 인자들 중에서 가장 첫 번째로 쓴 ken이 바로 this의 값으로 지정된다. 1, 2, 3은 this의 값과는 상관없이 순서대로 foo 함수가 된다. 그러므로 위 코드에서 this.age는 ken.age가 되어 35가 출력된다.

apply 또한 같은 역할을 한다. 단, apply는 무조건 배열을 인자로 넣어야 한다.

```
var age = 100

function foo(city, name) {
    console.log(this.age + '살' + this.name + this.city)
}

var ken = {
    age: 35,
    log: foo
```

```
}

foo.call(ken, 'Seoul', 'Jehyeok') // 35살 Jehyeok Seoul
foo.apply(ken, ['Seoul', 'Jehyeok']) // 35살 Jehyeok Seoul
```

4. new 키워드를 사용한 함수 실행

우리는 함수를 foo()와 같은 형태로 일반 실행할 수도 있지만 new 키워드를 사용해 생성자 함수로 만들어 사용할 수도 있다. 이 경우에 this는 빈 객체가 된다.

```
function Person() {
    console.log(this)
}

new Person()
```

이때, this는 빈 객체를 가리키며 위의 생성자 함수는 this라는 빈 객체를 return 한다. (return 문이 없어도 빈 객체를 return 하는 것이 생성자 함수의 특징 중 하나이다.)

```
function Foo() {
    console.log(this.age) // undefined
    this.age = 100;
    console.log(this.age) // 100
}

new Foo()
```

100이 출력되고, return 문이 없어도 return 은 되기 때문에 Foo 함수는 { age: 100 } 이라는 객체를 리턴한다.

```
function Person() {
    this.name = 'ken'
    console.log(this)
}

var ken = new Person()
console.log(ken) // { name: "ken" }
```

```
function foo() {
    this.age = 100;
    return 3;
}

var a = new foo()
console.log(a) // { age: 100 }
```

생성자 함수는 특이하게도 return 문이 있어도 이를 무시하고 this객체를 리턴하는 특징이 있다.

```
function foo() {
    this.age = 100
    return { haha: 23232 }
}

var a = new foo()
console.log(a) // { haha: 23232 }
```

하지만 생성자 함수도 리턴되는 대상이 객체라면 this 객체 대신에 해당 객체가 리턴된다.

```
const obj = { name: "Tom" };

const say = function(city) {
    console.log(`${this.name}, ${city}`);
}

const boundSay = say.bind(obj);
boundSay("Seoul");
```

bind 함수는 함수를 바로 실행하지 않고 새로운 함수를 리턴한다.

💡 Quiz

아래 코드에서 alert가 출력될 것인가??

```
function programmer() {
  this.isSmart = false;
  this.upgrade = function (version) {
    this.isSmart = !!version;
    work();
  }
}
function work() {
  it (this.isSmart) {
    window.alert('I can do my work! I am smart!');
  }
}
```

```
var programmer = new programmer();
programmer.upgrade(1.1);
```

programmer 생성자 함수의 upgrade 함수에서 work가 실행된다. 이때 work의 실행 방식을 자세히 보면... work() 로 일반 실행 방식이다. 그렇기 때문에 전역 isSmart는 없으므로 alert 문이 실행되지 않는다!