call

```
function playVideo(a, b)
{
    console.log(`name is ${this.name}`);
    console.log(`sum is ${a+b}`);
}

playVideo.call({ name : "Anubhav Gupta"}, 5, 6);

//name is Anubhav Gupta
//sum is 11
```

apply

```
function playVideo(a, b)
{
    console.log(`name is ${this.name}`);
    console.log(`sum is ${a+b}`);
}

playVideo.apply({ name : "Anubhav Gupta"}, [5, 6]);

//name is Anubhav Gupta
//sum is 11
```

bind

```
function playVideo(a, b)
{
    console.log(`name is ${this.name}`);
    console.log(`sum is ${a+b}`);
}

const play = playVideo.bind({ name : "Anubhav Gupta"}, 5, 6);
play();
//name is Anubhav Gupta
//sum is 11
```

```
const video =
{
    title: 'a',
    tags: ['a', 'b', 'c'],
    showTags: function()
    {
        this.tags.forEach(function(tag)
        {
            console.log(tag, this.title);
        });
    }
}

video.showTags();

//a undefined
//b undefined
//c undefined
```

Using Array Callback Parameters

The forEach method has optional parameters for the index and the array itself, which can be useful.

```
const video =
{
    title: 'a',
    tags: ['a', 'b', 'c'],
    showTags: function()
    {
        this.tags.forEach(function(tag)
        {
            console.log(tag, this.title);
        }, this);
    }
}

video.showTags();

//a a
//b a
//c a
```

## Caching this

Store the this context in a variable before entering the forEach loop.

```
const video =
{
    title: 'a',
    tags: ['a', 'b', 'c'],
    showTags: function()
    {
        const self = this;
        this.tags.forEach(function(tag)
        {
            console.log(tag, self.title);
        });
    }
}

video.showTags();

//a a
//b a
//c a
```

## Using Arrow Function

Arrow functions inherit the this context from their containing function (lexical scoping). So, using an arrow function as the callback will maintain the correct this context.

```
const video =
{
    title: 'a',
    tags: ['a', 'b', 'c'],
    showTags: function()
    {
        this.tags.forEach((tag)=>
        {
            console.log(tag, this.title);
        });
    }
}

video.showTags();

//a a
//b a
//c a
```

Using .bind(this)
This is what you've used in your code. It explicitly binds the outer this context to the inner callback function, making sure it refers to the video object.

```
const video =
{
    title: 'a',
    tags: ['a', 'b', 'c'],
    showTags: function()
    {
        this.tags.forEach(function(tag)
        {
            console.log(tag, this.title);
        }.bind(this));
    }
}

video.showTags();

//a a
//b a
//c a
```

apply() and call() methods are used to explicitly set the value of this within a function call. They are useful when you want to call a function with a specific this value, but they are not well-suited for changing the this context of inner functions within a nested function like in your forEach callback. You could potentially use .apply() or .call() for the outer function (showTags), but that wouldn't affect the inner function's this context within the forEach callback.