When You Should Not Use Arrow Functions

An arrow function doesn't have its own this value and the arguments object. Therefore, you should not use it as an event handler, a method of an object literal, a prototype method, or when you have a function that uses the arguments object.

Event handlers

Suppose that you have the following input text field:

```
<input type="text" name="username" id="username" placeholder="Enter a username">
```

And you want to show a greeting message when users type their usernames. The following shows the element that will display the greeting message:

```
<div id="greeting"></div>
```

Once users type their usernames, you capture the current value of the input and update it to the element:

```
const greeting = document.querySelector('#greeting');
const username = document.querySelector('#username');

username.addEventListener('keyup', () => {
  greeting.textContent = 'Hello ' + this.value;
});
```

However, when you execute the code, you will get the following message regardless of whatever you type:

```
Hello undefined
```

It means that the this.value in the event handler always returns undefined.

As mentioned earlier, the arrow function doesn't have its own this value. It uses the this value of the enclosing lexical scope. In the above example, the this in arrow function references the global object.

In the web browser, the global object is window. The window object doesn't have the value property. Therefore, the JavaScript engine adds the value property to the window object and sets its values to undefined.

To fix this issue, you need to use a regular function instead.
The this value will be bound to the <input> element that triggers the event.

```
username.addEventListener('keyup', function() {
  greeting.textContent = 'Hello ' + this.value;
});
```

Object Methods

```
const person =
{
      firstName: 'John',
      lastName: 'Doe',
      getFullName: ()=> `${this.firstName} ${this.lastName}`
};

const name = person.getFullName();
console.log(name); //undefined undefined

const person =
{
      firstName: 'John',
      lastName: 'Doe',
      getFullName: function(){ return `${this.firstName} ${this.lastName}`
}
};

const name = person.getFullName();
console.log(name); //John Doe
```

Prototype Methods

```
function Counter(count = 0)
{
  this.count = count;
}

Counter.prototype.increment = ()=>
{
  this.count++;
  console.log(this.count);
}

const counter = new Counter(5);

counter.increment(); //NaN


Counter.prototype.increment = function()
{
  this.count++;
  console.log(this.count);
}

const counter = new Counter(5);

counter.increment(); //6
```

Functions that use the arguments object

```
const fx = (a, b)=>
{
  console.log(arguments[0]);
  console.log(arguments[1]);
}

fx(5, 6);

const fx = function(a, b)
{
  console.log(arguments[0]); //5
  console.log(arguments[1]); //6
}

fx(5, 6);
```