**Why does the bottom method defined with arrow syntax remains bound to the obj? And the top one doesn't?**

**[Making bound methods with class fields](https://javascript.info/class" \l "making-bound-methods-with-class-fields" \t "_blank)**  
As demonstrated in the chapter [Function binding](https://javascript.info/bind) functions in JavaScript have a dynamic this. It depends on the context of the call.  
So if an object method is passed around and called in another context, this won’t be a reference to its object any more.  
For instance, this code will show undefined:

class Button {  
 constructor(value) {  
 this.value = value;  
 } click() {  
 alert(this.value);  
 }  
}let button = new Button("hello");setTimeout(button.click, 1000); // undefined

The problem is called "losing this".  
There are two approaches to fixing it, as discussed in the chapter [Function binding](https://javascript.info/bind):

1. Pass a wrapper-function, such as setTimeout(() => button.click(), 1000).
2. Bind the method to object, e.g. in the constructor.

Class fields provide another, quite elegant syntax:

class Button {  
 constructor(value) {  
 this.value = value;  
 }  
 click = () => {  
 alert(this.value);  
 }  
}let button = new Button("hello");setTimeout(button.click, 1000); // hello

The class field click = () => {...} is created on a per-object basis, there’s a separate function for each Button object, with this inside it referencing that object. We can pass button.click around anywhere, and the value of this will always be correct.  
That’s especially useful in browser environment, for event listeners.