**this**

A function's this keyword behaves a little differently in JavaScript compared to other languages. It also has some differences between strict mode and non-strict mode.

In most cases, the value of this is determined by how a function is called (runtime binding). It can't be set by assignment during execution, and it may be different each time the function is called. The bind() method can set the value of a function's this regardless of how it's called, and arrow functions don't provide their own this binding (it retains the this value of the enclosing lexical context).

This requires a context .

This is a reference word.

Until the function is called ‘this’ doesn’t point to anything.

We can manually provide context(explicite).

**Ex** :  
 const test = {

prop: 42,

func: function() {

return this.prop;

},

};

console.log(test.func());

**Syntax :  
  
 this**

**This have the followings 3 methods,**

* **Call**
* **Apply**
* **Bind**

Note that in non–strict mode, with **call** and **apply**, if the value passed as this is not an object, an attempt will be made to convert it to an object. Values null and undefined become the global object. Primitives like 7 or 'foo' will be converted to an Object using the related constructor, so the primitive number 7 is converted to an object as if by new Number(7) and the string 'foo' to an object as if by new String('foo'), e.g.

**The bind() method:**

ECMAScript 5 introduced **Function.prototype.bind().** Calling **f.bind(someObject)** creates a new function with the same body and scope as f, but where this occurs in the original function, in the new function it is permanently bound to the first argument of bind, regardless of how the function is being used.

Ex :

function f() {

return this.a;

}

const g = f.bind({ a: 'azerty' });

console.log(g()); // azerty

const h = g.bind({ a: 'yoo' }); // bind only works once!

console.log(h()); // azerty

const o = { a: 37, f, g, h };

console.log(o.a, o.f(), o.g(), o.h()); // 37,37, azerty, azerty