The Simple Rules to 'this' in Javascript

Rule 1 → If new keyword is used when calling function, this inside the fn is brand new object.

Eg →
```
function abc () {
    Console.log (this);      // { }
    this. value = 10;
    Console. log (this);     // {value:10}
}
new abc ();
```

Rule 2: If apply, call, or bind are used to call a fn, this inside function is object that is passed as arguement.

Eg: Function fn() {
console.log (this);
}
var obj = { value: 5 }
var boundFn = fn.bind(obj);

boundFn(); // { value: 5 }
fn.call(obj); // { value: 5 }

Rule 3 → If fn is called as a method then this is object that function is a property of.

Eg → var obj = { value: 5,
printThis: function() {
Console.log (this);
}; }
obj.printThis(); // { value: 5,
printThis: f }

Rule 4: If fn is invoked as free function invocation, meaning it was invoked without any conditions present above, this is global object. In browser it's window.

For eg- Function fn() {
    console.log(this);
}

// If called in browser;
fn(); // window { stop: f, open: f, ... }

Function that is not declared as a method automatically becomes a property of global object, window.

When we call fn(), it is interpreted as window.fn().

Rule 5: If multiple of above rules apply, the rule that is higher wins.

Rule 6: If function is ES6 arrow function, it ignores all rules above and receive the this value of its surrounding scope at time it's created.

To determine 'this', go one line above arrow function's creation and see what value of this is there

for eg -

```
const obj = {
    value: 'abb'
    createArrowFn: function() {
        return () => console.log(this);
    }
};

const arrowFn = obj.createArrowFn();
arrowFn();          // { value: 'abb', createArrowFn: f }
```