Remember

this keyword value depends on how the function is called

i.e

```
"use strict";
function x() {
    console.log(this);
}
x();
```

→ undefined

```
window.x();
```

→ window. as x is called by window object.

```
const student = {
    name : "Akshay",
    printName : function() {
        console.log(this.name);
    },
};
```

↳ this refers to student as we call it using

```
student.printName();
```

so, depend upon how it called. as here called by student.

Use of call : (Method sharing) / function borrow

```
const student2 = {
    name : "Deepika",
};
```

student2.printName() ✗ student2 have no method

But can do do by another way (using call).

```
student.printName.call(student2); // value of this is now student2. as it is now called by student2
```

We can use call in also in this way

We don't need to write it in one object, we can write it globally

i.e. let printFullName = function () {
    console.log(this.firstname + this.lastName)
}

Here this is global as if we call it will directly
i.e. printFullName();

But
let name = {
    firstName : "Akshay",
    lastName : "Saini"
};

let name2 = {
    firstName : "Sachin",
    secondName : "Tendulkar"
}

we want this to be name of name2

so printFullName.call(name);
↓
Akshay saini
printFullName.call(name2);
↓
Sachin Tendulkar

If the functions have same parameter, how then do we pass the argument. By seperating with comma, after object name

i.e. printFullName.call(name, args1, args2, ...);

Apply Method

Only difference between call() & apply() is the way we pass the arguments.

Here we pass args in a Array list, list.

printFullName.apply(name, [args1, args2, ...]);

Bind Method

Bind method binds the method with the object & returns the copy of object.

i.e. ~~let printMyName = printMyName~~

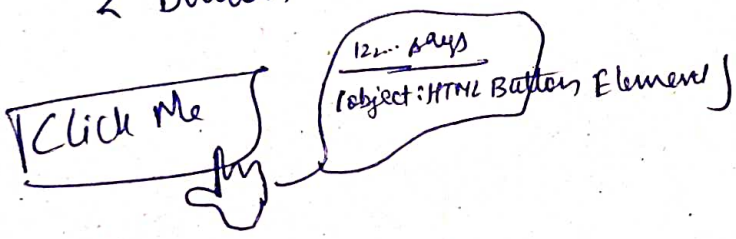
i.e. let printMyName = printFullName.bind(name, *args1, args2...)

console.log(printMyName());

---

this with Dom → reference to HTML Element

<button onclick="alert(this)">click Me</button>



this.tagName
↓
BUTTON