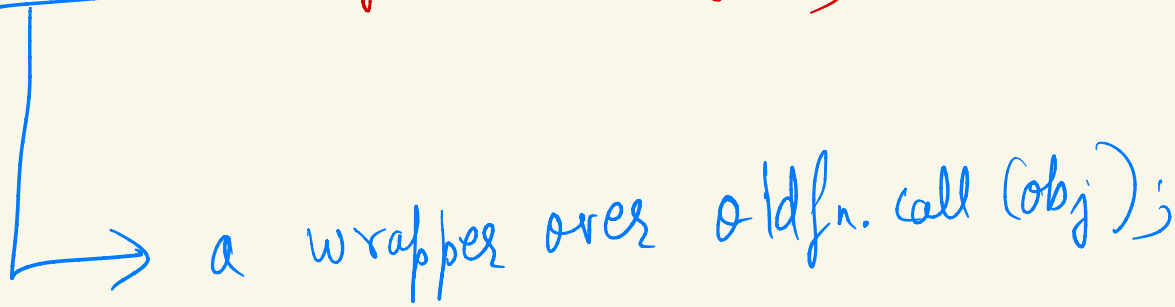
* const newFn = oldfn.bind(obj);



---

* newFn() ⟶ oldfn.call(obj);

* `const newFn = oldFn.bind(obj, arg1, arg2, arg3);`

↳ If it's a fn, it should be capable of taking arguments.

* old fn has 3 arg → arg1, arg2, arg3.

let's pass arg1 in old fn & arg2, arg3 in new fn

---

① case 1 arg1 → 10

```
const newFn = oldFn.bind(obj, 10);
```

→ newFn(arg2, arg3) → oldFn.call(obj, 10, arg2, arg3);

---

* whatever arg you are writing in bind method, it will be directly passed to call method, & subsequent arg it will take from newFn.

* whatever arg you are passing in bind method it is fixed for new Fn.

---

① newFn is same as old Fn. only diff is this keyword.

	① passing args in bind arg	② passing arg in new Fn
①	arg1	arg2, arg3
②	arg1, arg2	arg3

```
const newFn = oldFn.bind(obj, 10);
```

oldFn
arg1 arg2 arg3

* newFn ⇒ oldFn.call(obj, 10);

* newFn(100, 200, 300);

⇒ oldFn.call(obj, 10, 100, 200, 300);

* const newFn = oldFn.bind(obj, 10, 20);

* newFn(100, 200, 300);

⇒ oldFn.call(obj, 10, 20, 100, 200, 300);

---

* we have 2 chances of passing args while making newFn.

① bind

② newFn

* Const newFn = oldFn.bind(obj, fixedarg1, fixedarg2);

here we fixed arg1 & arg2 in old fn.

---

* newFn(nonfixedarg1);

⇒ oldFn.call(obj, arg1, arg2, nonfixedarg1);