# Class 1

## Object Cloning

# Class 1

Agenda

→ Object Cloning

→ Various methods of Cloning

---

* let a = 10;
* let b = a;

} ⇒ I assigned using assignment operator

---

b = 100

* let obj1 = { a: 10 }

* let obj2 = obj1;

* obj2 is a copy of obj1

* obj2.a = 100;

---

⇒ console.log(obj1);

⇒ console.log(obj2);

obj1 → { a: 100 }

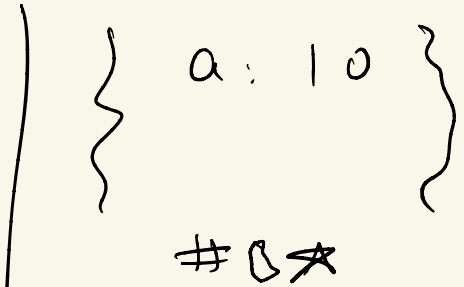
obj2 → { a: 100 }

let obj1 = #5★

★ let obj2 = obj1;

obj2 = #5★

★ this is known as shallow copy




when the cloned obj is having reference to old obj, this is known as Shallow copy.

⇒ If I am cloning obj1 to obj2 & then change any property in obj2, it should not reflect in obj1

* let obj1 = { a: 10 }

* let obj2 = { ...obj1 }

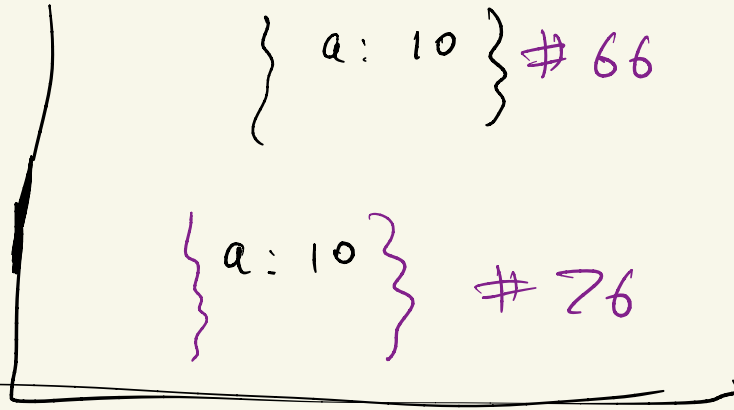
'...' → spread operator

It's used to spread properties

# how spread operator work ?

let obj1 = #66

let obj2 = #76




obj1.a = 100;

→ obj1 →

⇒ obj2 ⇒

Deep Clone → cloned obj should not have any ref of old obj

---

⇒ Cloning through spread opt is this deep copy / shallow copy.

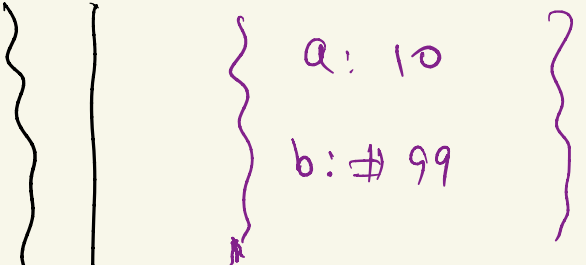
---

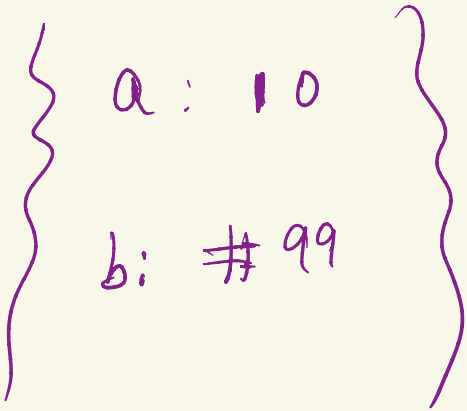
⇒ Spread operator helps in creating deep clone of any obj.

---

let obj1 = { a: 10, b: { c: 100 } }

let obj2 = { ... obj1 }




The new obj formed using '...' is having ref to old obj.

* deep copy

* Shallow copy

* partial deep copy / partial deep Clone

↓

Shallow copy.

* Now we need a proper deep clone method.

1) Stringify & parse ⇒ Easiest method

2) Structure Clone → inbuilt method → ✗

↳ our own structure clone

* JSON.stringify(obj) → It can manage n number of nesting

↳ It converts object into String

```
let obj1 = {
  a: 10,
  b: {
    c: 100,
  },
};
```

1

```
{"a":10,"b":{"c":100}}                    script.js:70
```

2

* JSON.parse(String version of obj)

↓

new obj

String version = JSON.stringify(obj);

let obj = { = }

JSON.stringify → gives string version of obj which has no ref to old obj

JSON.parse → gives obj version of any string if it's possible.