

Agenda

- ① Host and Native objects
- ② behaviour of this
- ③ chaining
- ④ inheritance
- ⑤ (call, bind, apply) → (basic intro)

Native object and host object

Environment

host object

- ① Browser [window, document, local storage]
- ② Node.js [global, os, process]

JS

Native object

- ① [Date, JSON, Object, Array, ...]

only in GEC:

→ this is equal to window

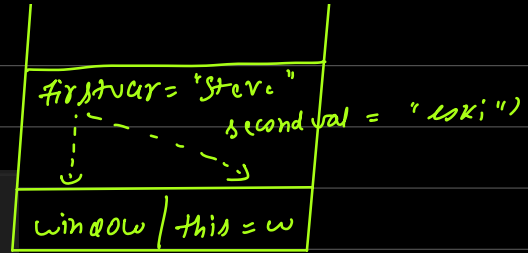
```

var firstVar = "steve";
let secondVal = "loki";

console.log("first: ", window.firstVar);
console.log("second: ", secondVal);
console.log("Hello from: ", this); // In GEC this = window.

```

gEC



```

let cap = {
  // property
  firstName: "Steve",
  // method
  sayHi: function () {
    console.log("Hi from ", this.firstName);
  }
}

```

EC

cap.sayHi()

→ cap.sayHi(); → method call

```
let sayHiAdd = cap.sayHi;
```

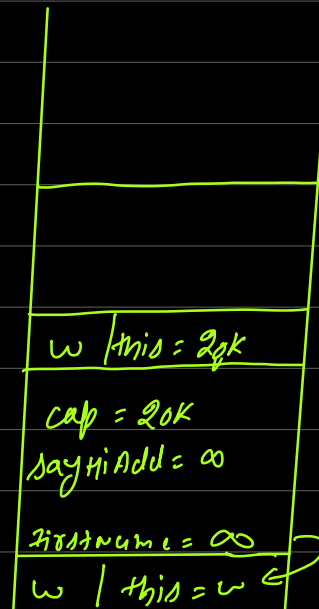
→ var firstName = "loki";

→ sayHiAdd();

→ Function call

window.sayHiAdd()

gEC

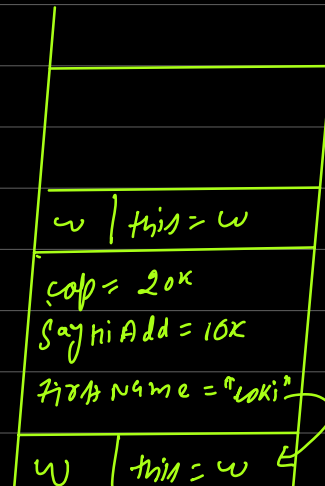


Heap



EC

gEC



```

let cap2 = {
  firstName: "Steve",
  sayHi: function (param) {
    console.log("47", this.firstName); ✓
    const iAmInner = function (param) { ✓
      console.log("49", this.firstName);
    }
    // EC by this kind of call -> window
    iAmInner(20);
  }
}

// // EC by this -> ?? -> cap
cap2.sayHi(10);

```

EC

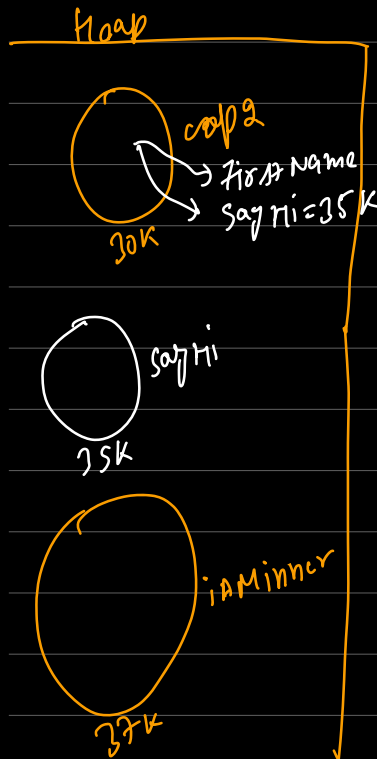
w | this = w

EC

w | this = 30K

cap2 = 30K

w | this = w



console

47 Steve
49 Loki

gEC

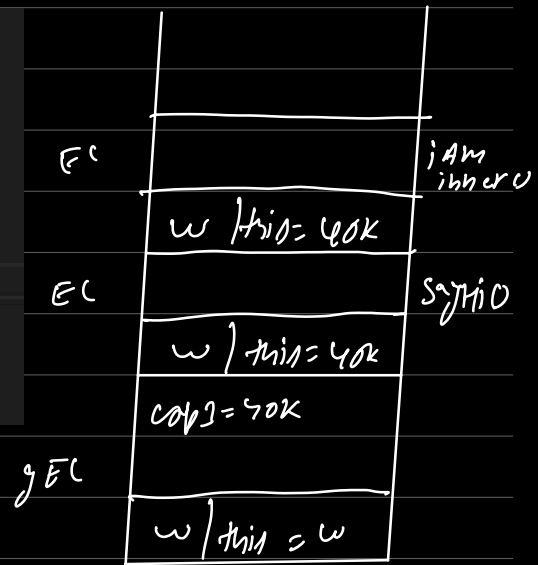
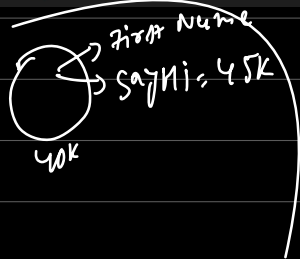
(*) object $\xrightarrow{\text{this} = w}$ simple-function

(*) object $\xrightarrow{\text{this} = \text{scope}}$ Arrow-function

```

let cap3 = {
  firstName: "Steve",
  sayHi: function () {
    console.log("53", this.firstName);
    const iAmInner = () => {
      console.log("55", this.firstName);
    }
    iAmInner();
  }
}
cap3.sayHi();

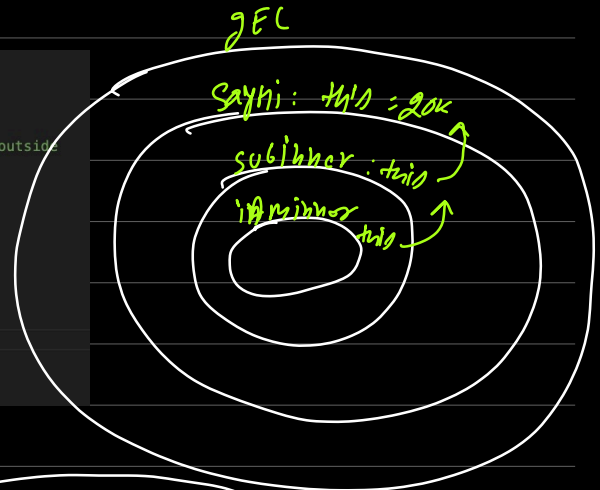
```



```

let cap4 = {
  firstName: "Steve",
  sayHi: function () {
    console.log("91", this.firstName);
    // arrow -> does not have it's own this. I am going to cheat it from outside
    const subInner = () => {
      console.log("94", this.firstName);
      const iAmInner = () => {
        console.log("95", this.firstName);
      }
      iAmInner();
    }
    subInner();
  }
}
cap4.sayHi();

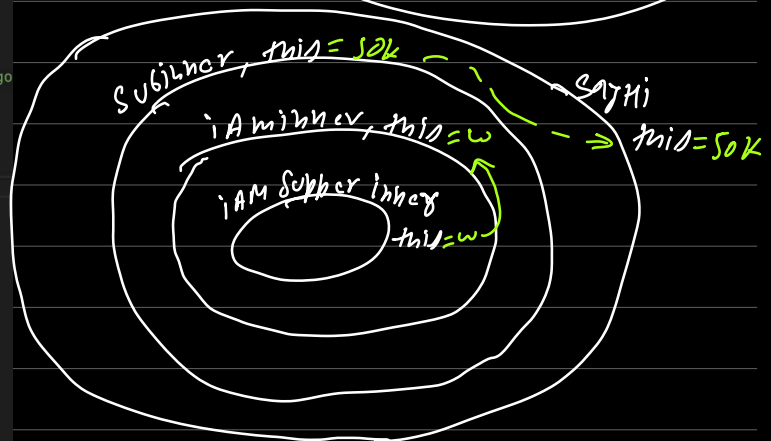
```



```


let cap5 = {
  firstName: "Steve",
  sayHi: function () {
    console.log("109", this.firstName);
    // arrow -> does not have it's own this. I am going to cheat it from outside
    const subInner = () => {
      console.log("94", this.firstName);
      const iAmInner = function () {
        console.log("114", this.firstName);
      }
      const iAmSuperInner = () => {
        console.log("117", this.firstName);
      }
      iAmSuperInner();
    }
    iAmInner();
  }
}
cap5.sayHi();

```



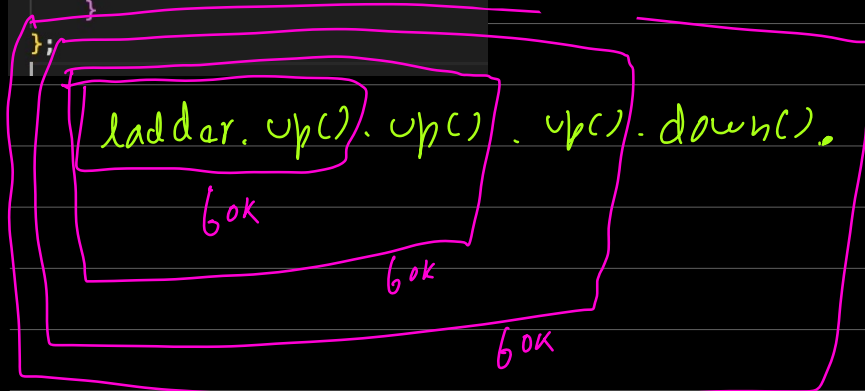
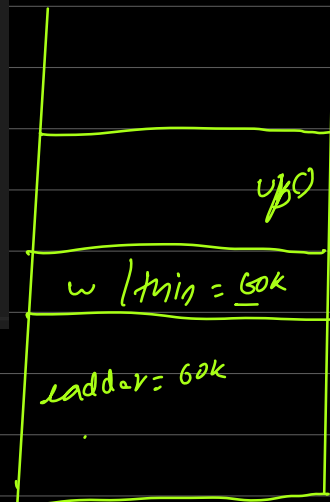
Non-strict (this)

Strict (this)

gbc	window	window
function		undefined
method	current object	current object
Arrow call	outer scope object	outer scope object.

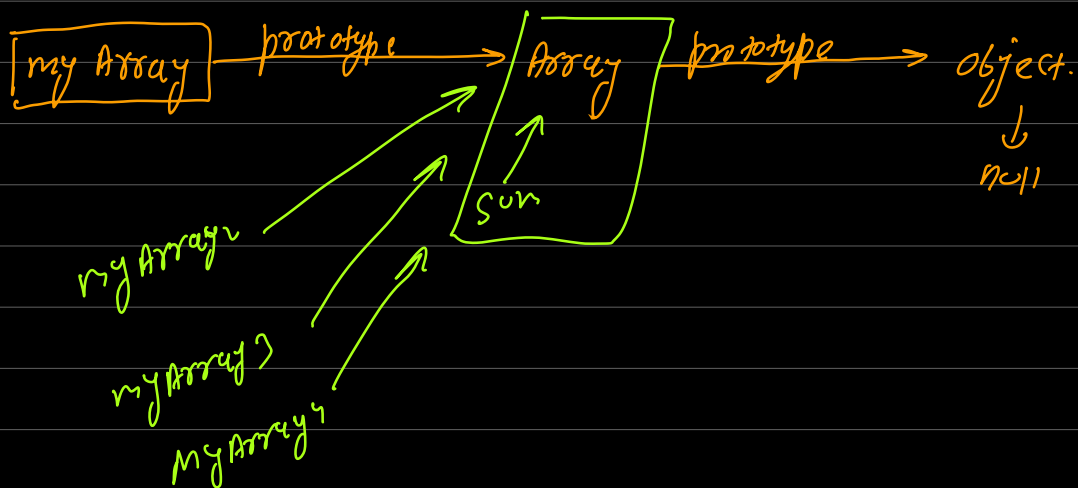
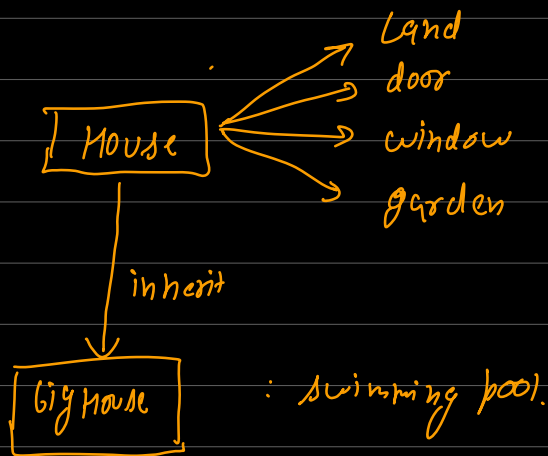
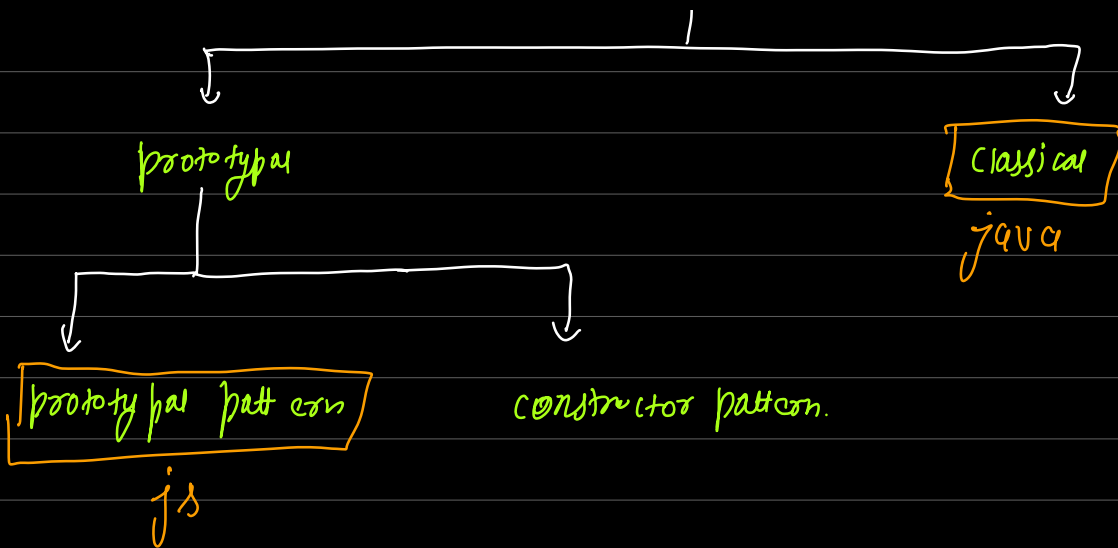
```
let ladder = {
  step: 0,
  up() {
    this.step++;
  },
  down() {
    this.step--;
  },
  showStep: function () {
    console.log(this.step);
  }
};
```

```
// goK.fun() ladder
ladder.up(); // step: 1
// up fn on ladder
ladder.up(); // step: 2
// goK.fun()
ladder.up(); // step: 3
ladder.down(); // step: 2
ladder.showStep(); // 2
```

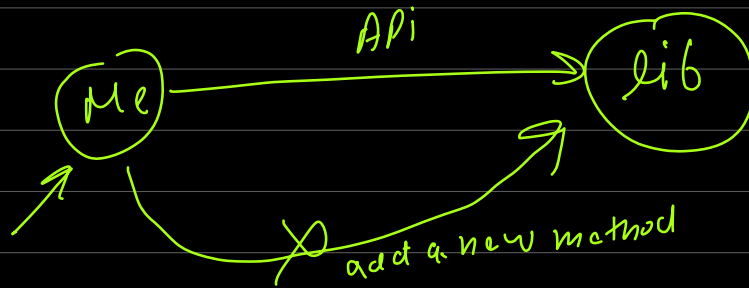


val = ladder.up()
val.up()

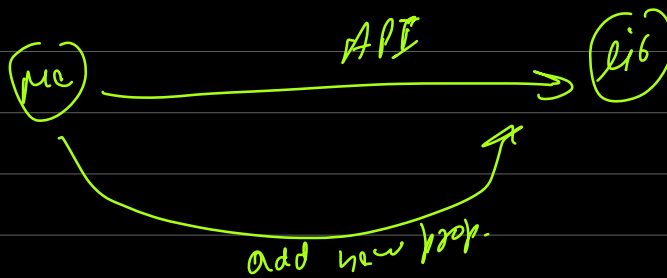
Inheritance



(java/c++)



js

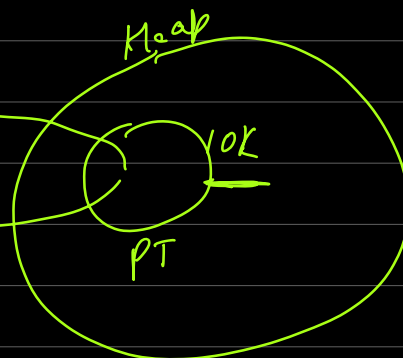


```
import {
```

```
  name :
```

```
  team :
```

```
}
```



Temporal Dead zone



Type	Redeclare	Reassign	Scope	T D Z
var	✓	✓	function/w	X
let	X	✓	block	✓
const	X	X	block	✓

fruit = ~~apple~~