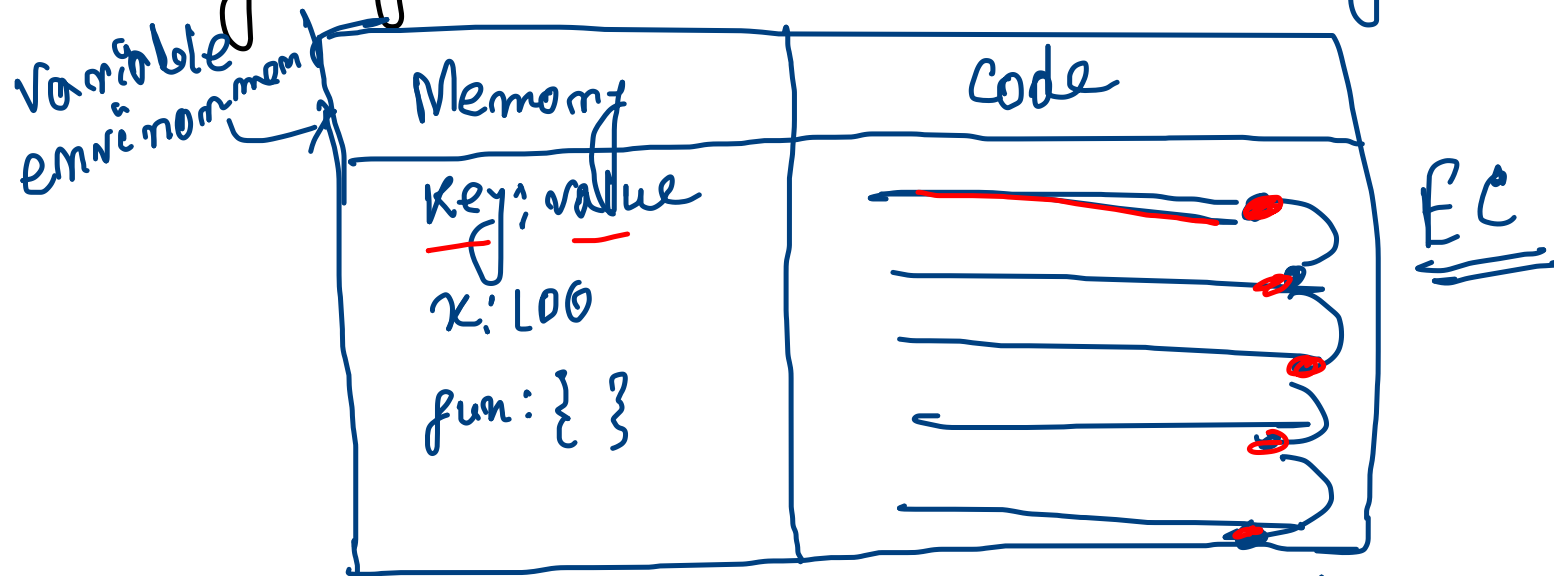


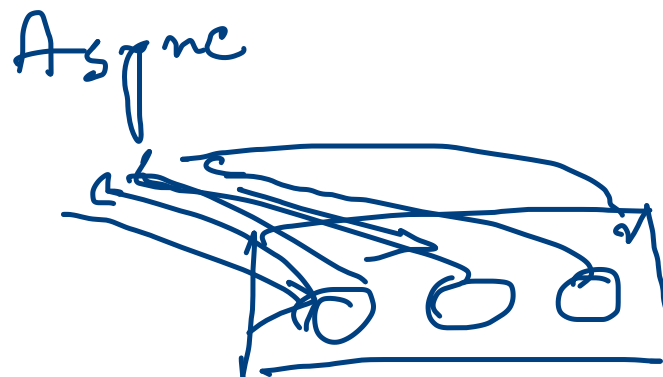
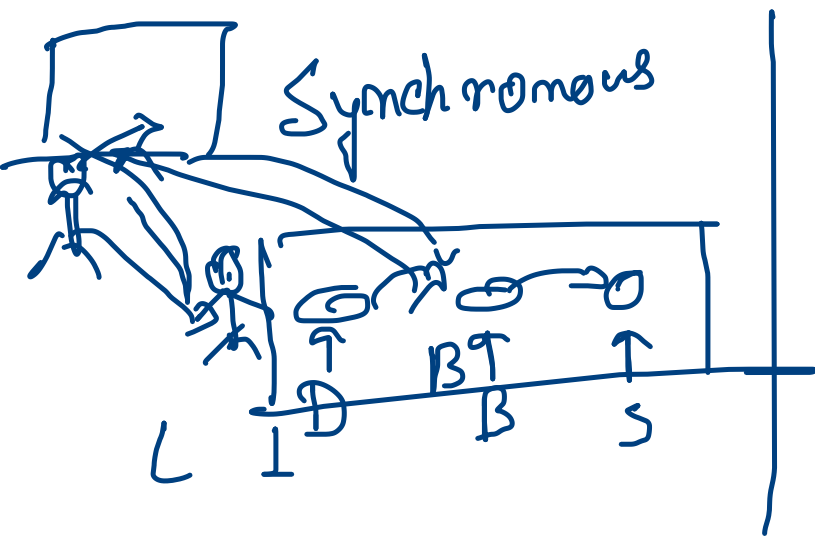
Execution Context?

Everything in JS happens inside an execution context.
Thread of execution



JS is a synchronous, single threaded language.

It execute one command at a time



```

var a = 5
function sum(num) {
  var res = num + num;
  return res;
}

```

```

var S1 = sum(a);
var S2 = sum(2);

```

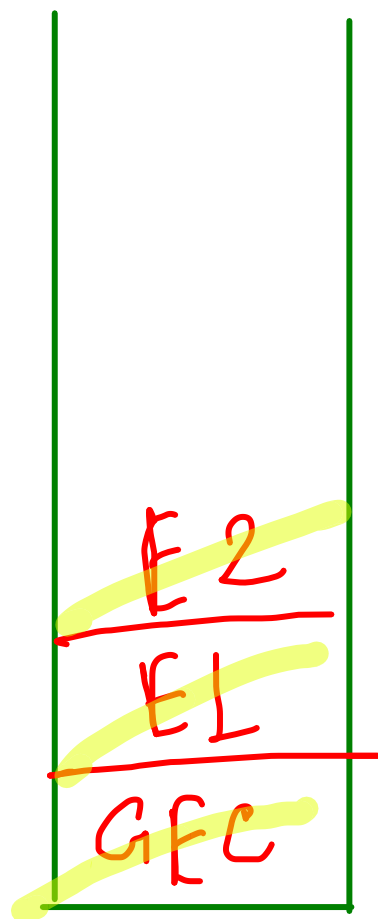
for func
total code

GEC Global execution context

Memory	Code
 a: undefined sum: { } S1: undefined S2: undefined 	 var a = 5; num = 5 num + num 5 + 5 return res
 num: 2 res: 4 	 num = 2 num + num 2 + 2 return res

- Memory Creation
- Code execution

Call Stack



- Execution Context Stack
- Program Stack
- Control Stack
- Runtime Stack
- Machine Stack

Callstack maintains the order
of execution context

Hoisting

```
index.html JS index.js X
JS index.js > demo
1 demo(); ✓
2 console.log(a);
3 var a=10;
4 function demo(){
5   | console.log("Hello World");
6 }
```

Hello World
undefined

Memory	Code
<u>a: undefined</u> <u>demo: {}</u>	<u>demo()</u> <u>call()</u>

```
JS index.js > demo
1 demo();
2 console.log(a);
3 var a=10;
4 function demo(){
5   console.log("Hello World");
6 }
7 demo();
8 console.log(a);
```

~~Output~~

```
Hello World
undefined
Hello World
10
```

Memory	Code
<div>10 a: undefined demo: {}</div>	<div><div>demo()</div><div>console.log(a);</div><div>a = 10</div><div>demo()</div><div>console.log(a);</div></div>

→ Hello world
under

Hello world

→ 10

```
JS index1.js > ...
1  demo();
2  console.log(a);
3  var a=10;
4  var demo={()=>{
5      console.log("hello World");
6  }}
7  demo();
8  console.log(a);
```

Function as a variable

Memory	Code
a: undefined demo: undefined <u> </u> <u> </u>	<div>demo()</div>

```

2 console.log(a);
3 var a=10;
4 var demo={()=>{
5   console.log("hello World");
6 }}
7 demo();
8 console.log(a);

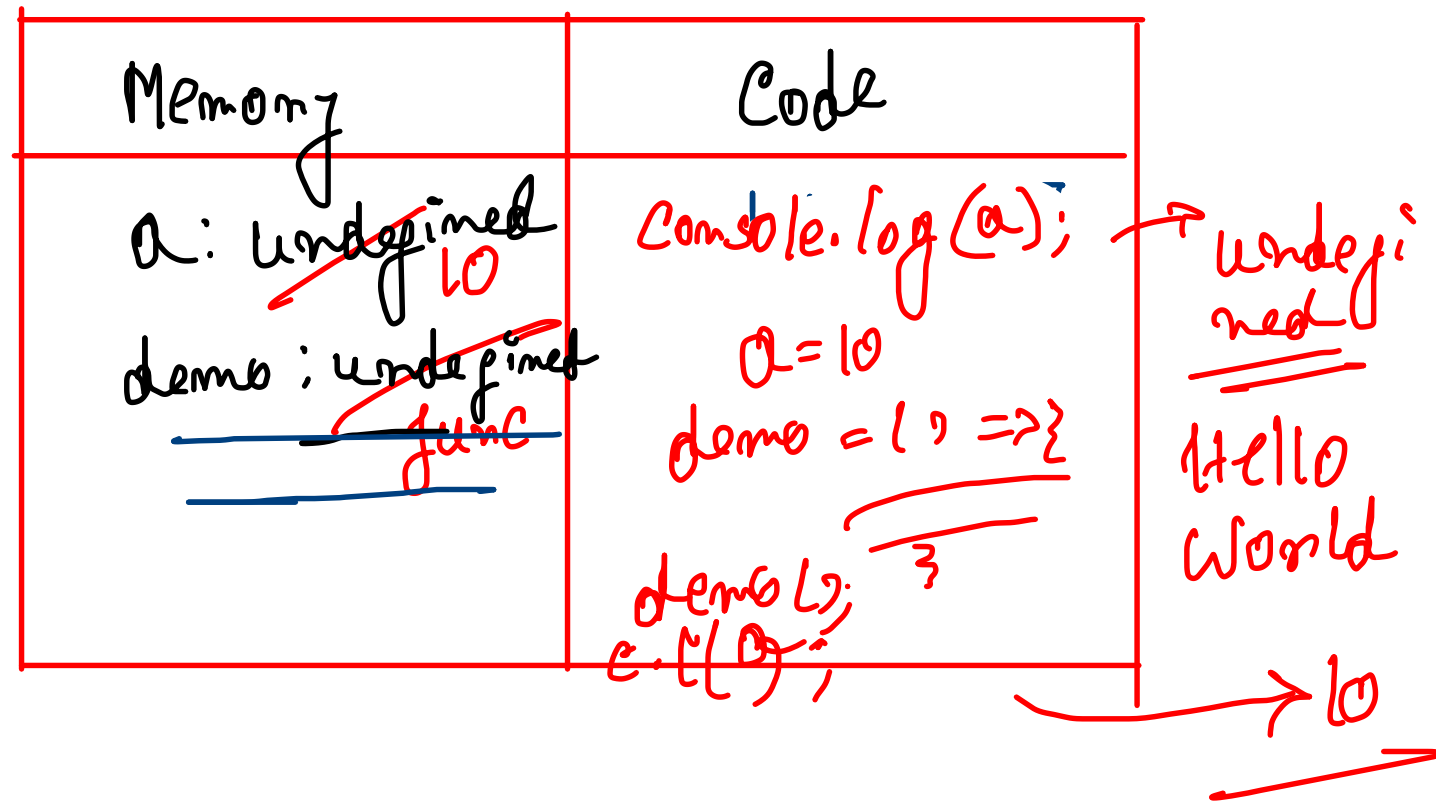
```

before a response

```

undefined
hello world
10

```



```

1  demo();
2  console.log(demo);
3  console.log(a);
4  var a=10;
5  function demo(){
6      console.log("Hello World");
7  }
8  demo();
9  console.log(a);
10 console.log(demo);

```

```

Hello World      index.js:6
f demo(){        index.js:2
  console.log("Hello World");
}
undefined        index.js:3
Hello World      index.js:6
10               index.js:9
f demo(){        index.js:10
  console.log("Hello World");
}

```

Memory	Code
<p>10</p> <p>a; undefined</p> <p><u>demo: {</u></p> <p><u>Total Function</u></p>	<p><u>demo();</u></p> <p><u>console.log(demo);</u></p>

Function

```
JS Func.js > ...
1  var a = 10;
2  demo();
3  demo1();
4  console.log(a);
5  function demo(){
6    var a = 20;
7    console.log(a);
8  }
9
10 function demo1(){
11   var a = 30;
12   console.log(a);
13 }
```

20
30
10

Memory

~~a: undefined 10~~

~~demo: { } -- }~~

~~demo1: { } -- }~~

Code

~~var a = 10;~~

Memory	Code
a: unde fined 20	var a = 20 console.log(a);

~~demo1~~

Memory	Code
a: unde fined 30	var a = 30 console.log(a);

~~demo1~~

~~demo()~~

~~GC~~