React

javascript library

html template

html:5

Javascript

Everything in Js happens inside an execution context.

Global execution context

variable envnt | thread of execution

memory	Code component
key : value a: 10 fn: {....}	

→ line by line execution

• Js is synchronous single-threaded language.

What happens when you run javascript?

```
Var n = 2;
function square (num){
Var ans = num * num;
 return ans; }
Var square2 = square(n);
Var square4 = square (4);
```

Memory	Code
→ allocates spl value. n: undefined. square: {.....} square2: undefined square4: undefined.	memory | code

→ argument
n : 2

Parameter of function

Phase 2

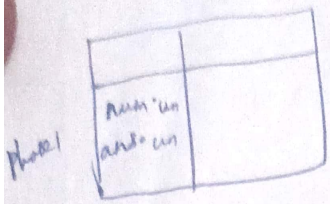
We invoke a function, brand new execution is created.

function name with parentheses,

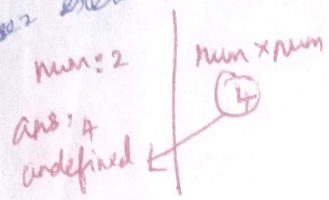
eg: Var square2 = square (n);

```
function Square (num) {
  var ans = num * num;
  return ans;
}
```

- allocating memory for num and ans,



Phase2 execution each line.

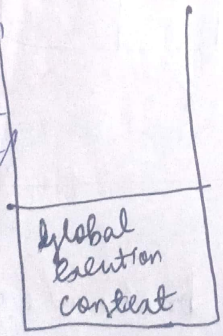


return the control of program to place where this fn was invoked, with this execution (stack) context is deleted from

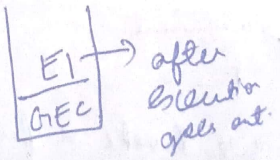
memory	Code
num: 4	46
ans: 16	

Call Stack - maintains order of execution of execution context

Program Control runtime machine / Execution context



Js program is run, Call Stack is populated with GEC.



- When there is a fn invocation all execution context is created and same process is followed.

When the function code is done executing the execution context is popped out, last global execution context also deleted.

function → new execution context created.

Console - debug code,
test ideas.

Code	where it shows
Console.log("Hello")	In the console.
document.body.innerHTML = "hello"	on the web page

Console.log() - only to developer not user.

Code 1:

```
Var x=7;
function getname(){
 console.log("Ezhil");
}
getname();
Console.log(x);
```

Ezhil
7

} function invocation

Before initialised, it is called

```
getName();
Console.log(x);
Var x=7;
function getName(){
Console.log("Namaste js");}
```

Namaste is undefined.

```
getname();
function getname(){
  };
```

Namaste js

```
console.log(getName);
function getName(){
console.log("Namaste js");
}
```

```
getname();
function getname(){
console.log("Namaste js");
}
```

Before the code is executed, memory is allocated to every variable.

JS ins through the program, allocates memory to variable before allocation.

- In JS before the code is executed, variables get initialised to undefined
- Arrow functions react as variable and get "undefined" during the memory creation phase while fns actually get run
- Hoisting. mechanism in JS where the variable declarations are moved to the top of scope before execution. possible to call a function before initialising it
- Variable declaration are scanned and are made undefined

function declaration scanned, and are available

Arrow function.

```
var getName = () => {
    console.log("Namaste");};
```

Not defined.

```
getName();
console.log(x);   -> error.
console.log(getName);
fn getName(){
    console;}
```

Not a function but a variable

```
getName();
console.log(getName);
Var getName = () => {
console.log("Namaste is"); (undefined)
}

Var getName2 = function() {
```

↓

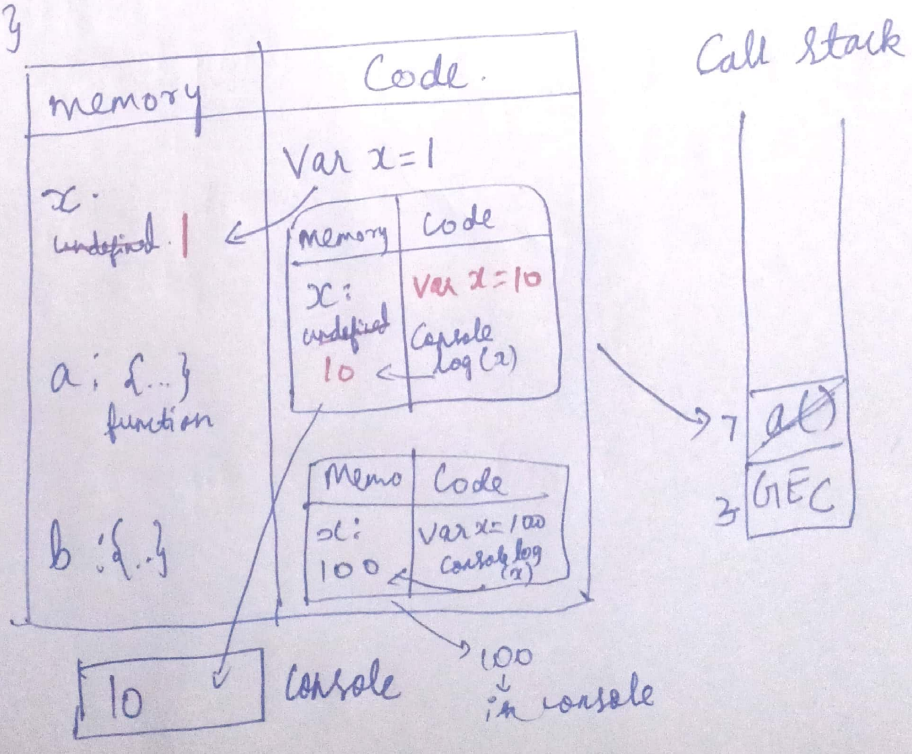
undefined

behaves like a variable only.

how functions works in js?

```
Var x=1;
a();
b();
Console.log(x);
function a(){
Var x=10;
console.log(x);}
function b(){
Var x=100;
console.log(x);
}
```

10
100
1



- first phase, Variables are assigned, undefined, while functions have their own code.
- Higher level of GEC on stack, higher its preference
- GEC created as fn, is invoked, destroyed as it execution is completed.
- Same Variable name but different scope of execution separates the Variable values.