OOPS-1 Javascript.

Agenda:

* Execution Context ✓
  - GEC
  - FEC
* IQ: null vs undefined.
* object
* Normal function, function expression
* Method
* Arrow function
* this keyword
* call, apply, bind
* inheritance in JS (prototype property & its usecases.

* Execution Context:

It's an environment where JS code is executed.

Contains variables, functions, objects & the scope chain.

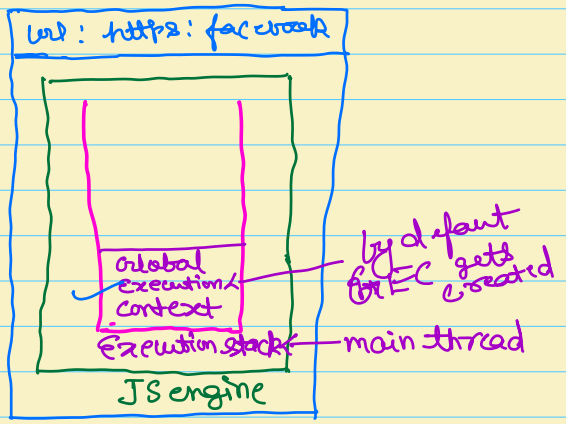
There are 2 types of ECs:

① GEC (Global Execution context)

② FEC (Function Execution context).

GEC:

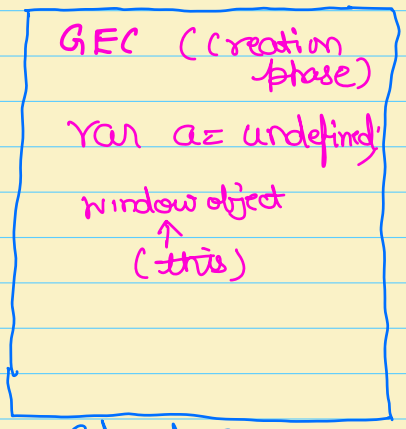
→ Whenever a JS engine receives a script file, it first creates global execution context (GEC) by default.



* For every JS file, there can be only one GEC.

If JS engine loads this JS code, GEC gets created in Execution stack.

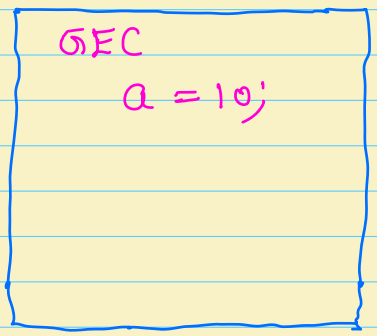
```
var a = 10;

function print() {

  var b = 20;

  console.log(a);
  console.log(a+b);

  var c = 20;

  if (c) {
    let d = 20;
    console.log(d);
  }
  console.log(d);
}

print();
```



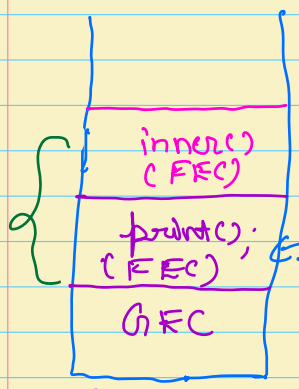
Global Execution Context

① **Creation phase:** It determines the values of variables & functions & set up the scope chain for the execution Context.

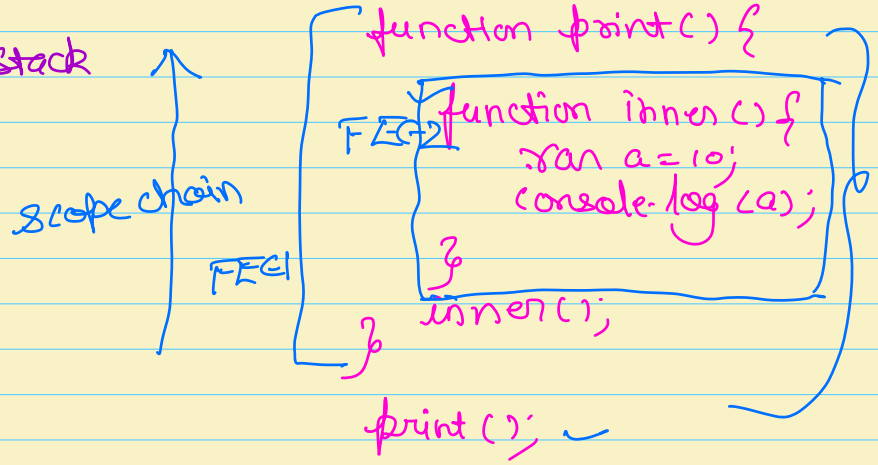
② **Execution phase:** In this phase, assignments of all variables are done and the code is finally executed.

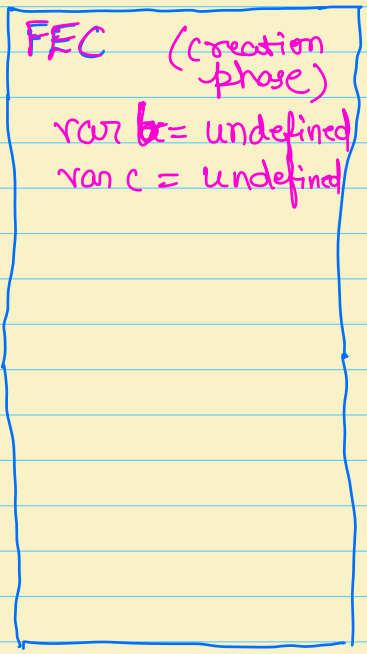


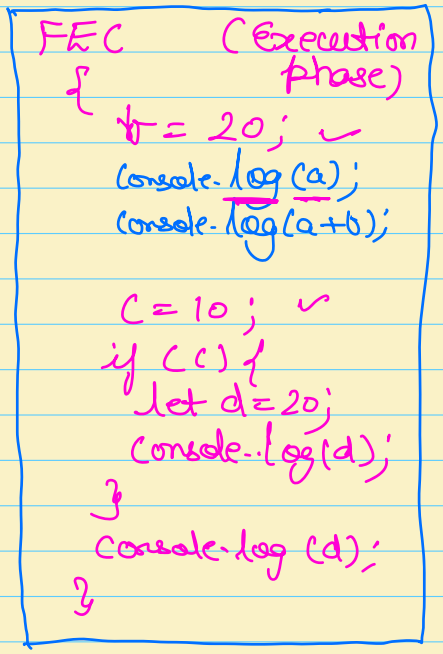
② FEC (Function Execution Context):

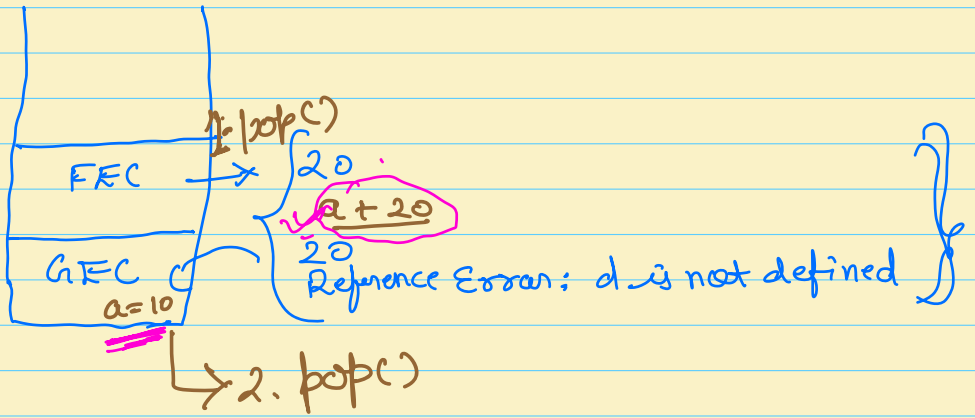


- FEC is created whenever a function is called.
- FEC can be more than one.









20

30

20

Reference error: d is not defined.

→ null vs undefined:

	null	undefined
①	null means there is no value.	① undefined means the absence of variable itself.
②	null can be assigned to a variable which indicates that it has no value.	② a variable is declared only but no value assigned.
	eg: let data = null;	eg: let a;

③ typeof null returns "object".	③ typeof undefined returns "undefined".
④ null is converted to Zero (value) when we perform primitive operations.	④ undefined is converted to NaN, when we perform primitive operations.
eg: let data = null; data + 20 = 20. data + 0 = 0	eg: let x; x + 10 = NaN

→ Object:

- An object is a real world entity that exists physically.
- Eg: ashwani (person), car, mobile, pan card, tablet.
- What is not an object ?
  an object that you can't see.
  eg: ① feelings ② thoughts ③ air
- It's a non-primitive data type that allows to store multiple collections of data in key: value pair.

- eg:
```
Const person = {
    'last name': 'Agrawal',
    name : 'Rohant',
    age: 34,
    greetMsg: function(){
        console.log("Hello" + this.name);
    }
};
```
(method)

Every object has 2 things:
① characteristics (name, age, height etc)
② behaviour / action eg: (speak, earn, run)

```
console.log(person.name); Rohant.
console.log(person.age); 34
person.greetMsg(); // Hello Rohant.

console.log(person.last name); // last is not
console.log(person['last_name']; defined in
// Agrawal.  person object.
```

Assignment:

① Car
② pizza
③ pan card
④ book

→ What is function?

- a reusable piece of code that performs a specific task or calculates a value.

- eg: factorial of a number.

```
if (num == 0 || num == 1) {
    return 1;
} else {
```

```
    return num * fact(num-1);
}
```

JS implementation.

{ == V8 ===

```
function fact(num) {
    if (num === 0 || num === 1) {
        return 1;
    } else {
        return num * fact(num-1);
    }
}
fact(5); // 120.
```

5 * 4 * 3 * 2 * 1 = 120

Assignment

① sum(1)(3)(5); // 9

② sum(1)(3)(5)(8)(); // 17

→ what is function expression in JS?

- when you assign a function to a variable is known as FE

- eg:

```
const fact = function() {
    ---
    ---
    ---
}
```

→ what is method in JS?

- when you define a function inside an

object or in a Constructor function.
? (In OOPS-2)

→ What is arrow function?

- provides a shorter and streamlined way to write functions as compared to traditional function expressions.

- eg: ① factorial of a number.

```
Const fact = (num) => {
    ---
    ---
    ---
}
```

② multiplication of 2 numbers.

```
Const multiply = (x, y) => x * y;
```

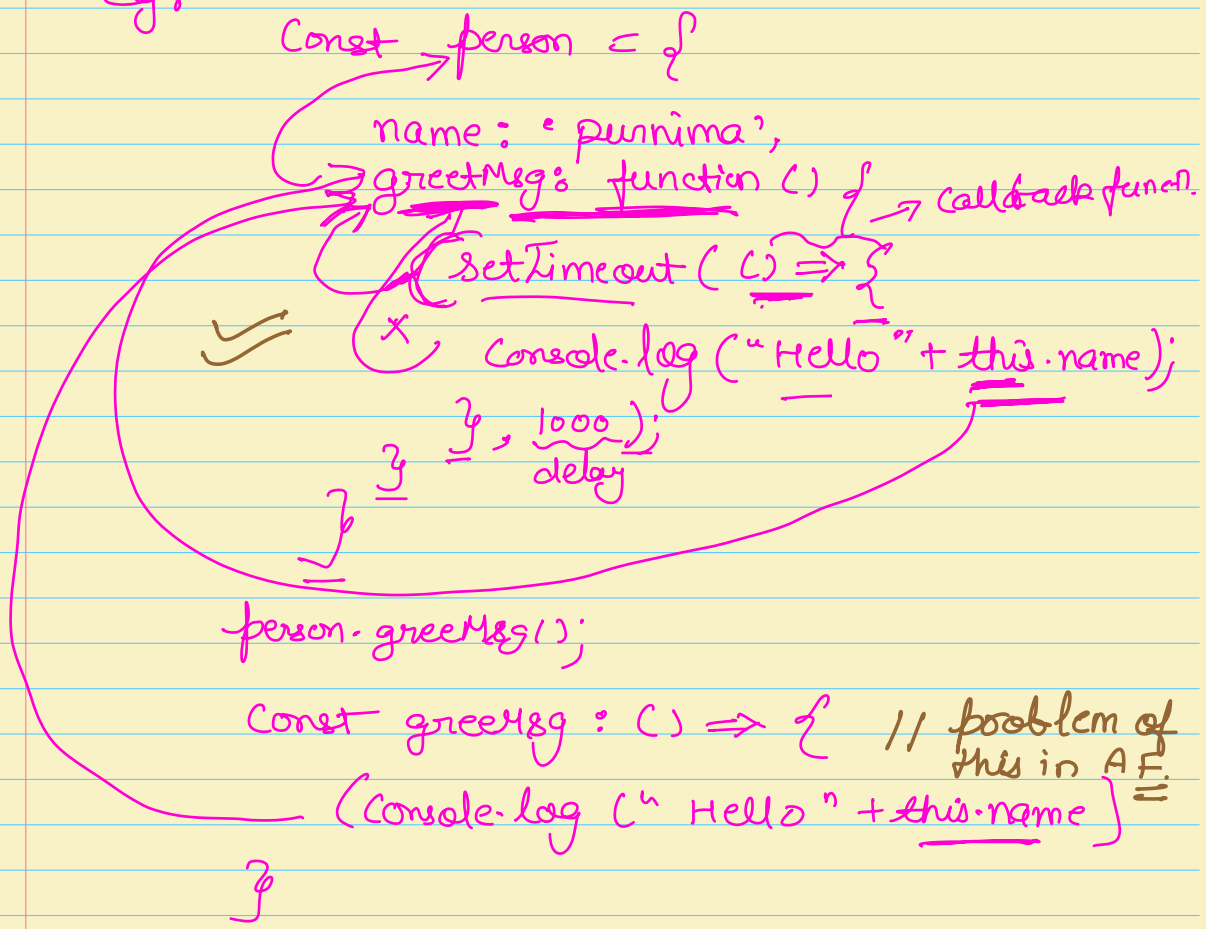
IQ: Can you find this in an arrow function?

- No, arrow function can't find 'this'.
- Arrow functions don't have their own 'this' context.

Instead, they inherit the 'this' value

from the enclosing scope. (the context in which arrow function is defined).

eg:



→ 'this' keyword:

- this keyword refers to the context in which a function is called.
- It is a special identifier that allows a function to access & operate on the properties and methods of the

object that called it.

① Global scope: outside of any function, this refers to the global object (window object)

```
Console.log (this === window); // true.
```

ID: What is the difference b/w == & ===?
Which one has to use in JS?

→ == do only value comparision.
=== do value comparision & also check type as well.

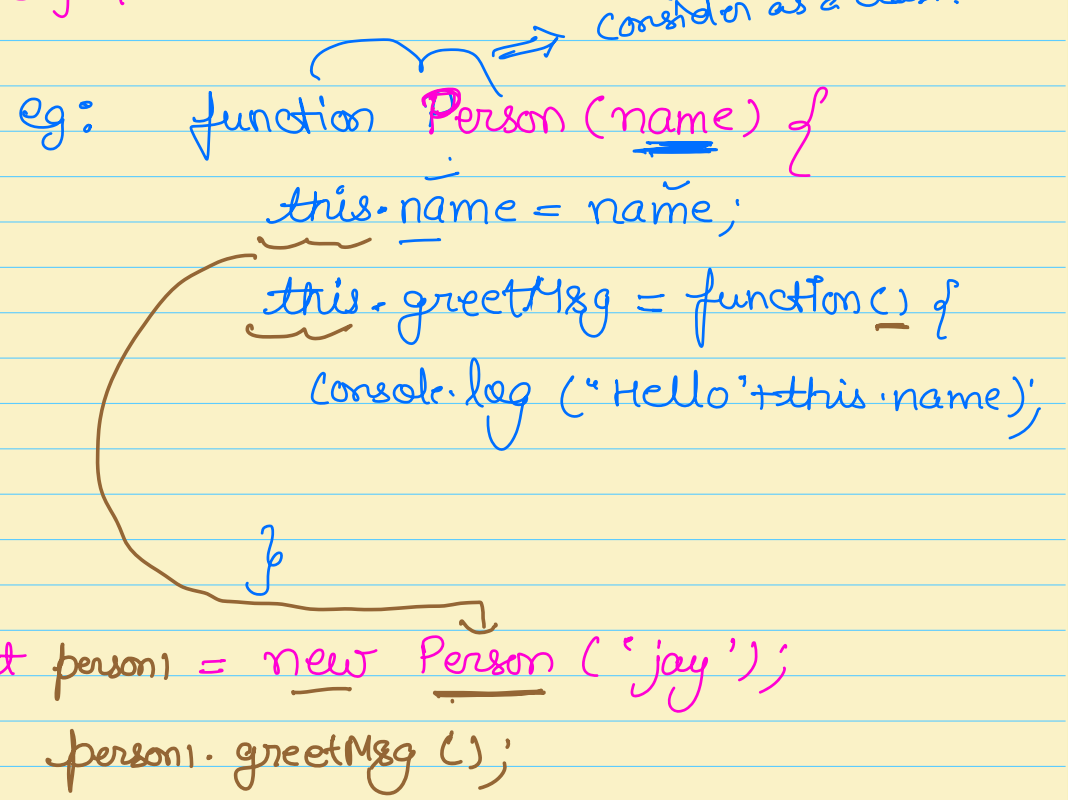
→ In JS program, always try to == operator.

- object method: When a function is called as a method of an object, this refers to the object itself.

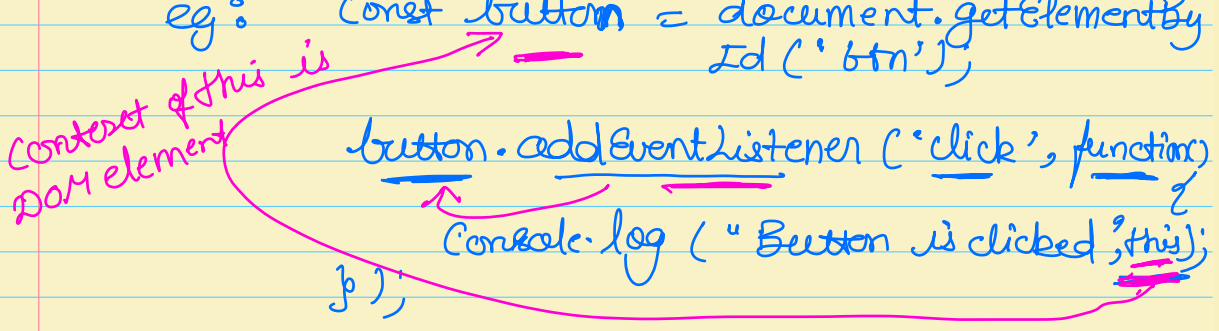
eg:
```
Const person = {
    name: 'Amit',
    greetMsg: function() {
        console.log ('Hello' + this.name);
    }
}
person.greetMsg(); // Hello, Amit.
```

here this is referring?
this is refers to person object.

③ Construction function: When a function is used as a Constructor function using new keyword, this refers to the newly created Object.



④ Event handlers: In event handlers or callbacks, this refers to the DOM element on which your event occured.



⑤ Explicit function binding:

- you can explicitly bind this to a specific object using methods (call(), apply() & bind()).

eg:

```
const person1 = {
    name: 'paresh',
    greet: function () {
        console.log('Hello,' this.name);
    }
};

const person2 = {
    name: 'Abhinav'
};

person1.greet(); // Hello, paresh.

person1.greet.call(person2); // Hello, Abhinav.
```

here this refers to person2 object of call() method.

→ Call, apply & bind

① call(): used to invoke a function with a specified this value and individual arguments.

Syntax: function.call(thisArg, arg1, arg2....argn);

eg: above.

② apply(): this similar to call(), but it accepts arguments as an array.

Syntax: function.apply(thisArg, [arg1, arg2, ... argn]);

```
function greet(msg) {
    console.log(msg + this.name);
}

const person = { name: 'Ashish' }
const args = ['Hi'];

greet.apply(person, args); // Hi Ashish
```

→ bind() : Creates a new function with a specified this value and pre-set arguments without immediate execution.

Syntax: function.bind(thisArg, arg1, arg2, ... argn);

eg:
```
const person = {
    name : 'Ashwani',
    greet : function() {
        console.log("Hello" + this.name);
    }
}

const sayHello = person.greet.bind(person);
sayHello(); // Hello, Ashwani;

const newPerson = {
    name : 'Ashu',
    greet : sayHello
};

newPerson.greet(); // Hello, Ashwani
```