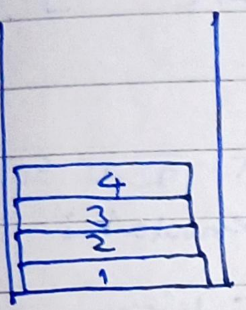
JS-11

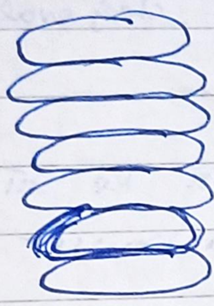
- **Call Stack.**

→ function hello(){
        console.log("hello");
  }
  hello();

} Calling.

→ **Stack** Last in first out. (LIFO)





→ deletion takes place from top. pehle ye vala out.

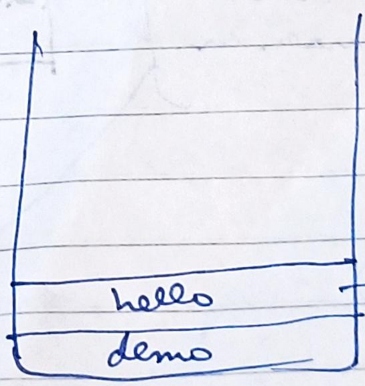
→ Addition also occurs at top.

- hello(){

  }

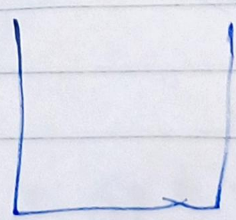
  demo(){
      hello()
  }

  demo(); (1)



→ जैसे ही hello ko call

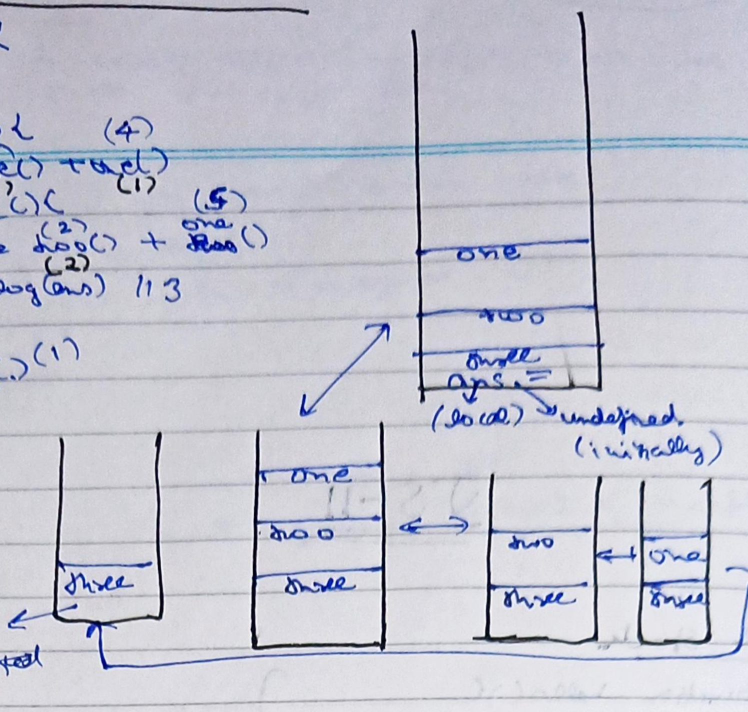
// जैसे ही hello ki call puri vo vapis aayegi kyu ki ye pura ho chuka ho next
// जैसे ही demo bhi complete.
// Stack खाली हो जाएगा।



• Visualising the call stack

```
function one(){
        return 1
}
function two(){                 (4)
        return one() + one()
}         (1)    (3)     (1)
function three(){                (5)
     let ans = two() + one()
        (2)
     console.log(ans)   // 3
}

⇒      three() (1)
```



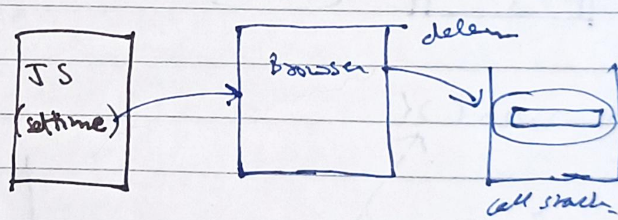
» we can also visualise call stack through debugging

⇒ JS is single threaded. [ek time pe code ki ek cheej evaluate]

→ callbacks helps in resolving this

→ setTimeout (JS settimeout से दो काम लेके आता उसी पर रहो Bcoz waiting का काम JS nhi browser krwata hai

```
> setTimeout(function(){
        console.log( ...
  },2000);
  setTimeout(function
          console.log...
  },2000);
```



> Synchronous

```
let a = 25
console.log(a)
let b = 10
console.log(b)
console.log(a+b)
```

Asynchronous
Using callbacks, settimeout, setinterval

Callback Hell (Due to asynchronous nature)

```
fun" changeColor( color, delay, nextColorChange) {
        setTimeout(() => {
            h1.style.color = color;
            if (nextColorChange) nextColorChange();
        }, delay);
}

changeColor("red", 1000, () => {
        changeColor("orange", 1000)
});
```

// callback nesting = callback hell.

Promises → object

> The promise object represents the eventual completion (or failure) of an asynchronous operation & its resulting value.

> यानी तो वो चीज़ complete होगी या incomplete same है JS as we see one thing then it will be complete or incomplete/fail as we go ahead.

- resolve (success) and reject (failure)

```
function saveToDb(data) {
    return new Promise((resolve, reject) => {
        let internetSpeed = Math.floor(Math.random()*10)+1;
        if (internetSpeed > 4) {
            resolve("success: data was saved")
        } else {
            reject("failure: weak connection");
        }
    });
}

saveToDb("apna college");
```

- Promises (then and catch)

let request = saveToDb("apna") // upper vala jo code likha bad likhna

request.then(() => {
    console.log("promise resolved")
    console.log(request)
})
.catch(() => {
    console.log("promise was rejected")
    console.log(request)

* Promises (Improved)

> single catch is enough to get error message if not saved.

> saveToDb("apna college")
    .then(() => {
        console.log("data1 saved");
        return saveToDb("helloworld");
    })
    .then(() => {
        console.log("data2 saved")
    })
    .catch(() => {
        console.log("promise rejected");
    });

// yr vala & isse upr vale call back hell में ज्यादा अच्छा बात है।

* Result and Errors

.then(result){
    console.log(request) // जो upper func में resolve में आयेगा hoga
.catch(error){
    console.log(error) // जो upper. reject में store hoga.
}

* Refractoring old code

// Pending promises हैं जो settimeout ने इसको 6 sec. दे दिया। तो जब तक 6 sec. nhi hue & humne request promise ko print karvana chahe to pending after 6sec. complete.