→ Closures

→ Some imp Promise func^n

→ iterators

→ generators

→ how async await internally works

→ how async await handles promises

Closures

parameter

vvi

```
function doSomething(task) {
    console.log("Initialising do something");
    setTimeout( () => {
        console.log("Timer done and task done", task);
    }, 3000);

    console.log("Exitting do something after intialising a timer");

}
```

```
undefined
```

```
doSomething("complete homework");
```

1 → how we are cbk to access 2nd.

2

call

call Stack

cb

Macrotask

cb

Callback

R.E

Timer → 3s →

Event Loop

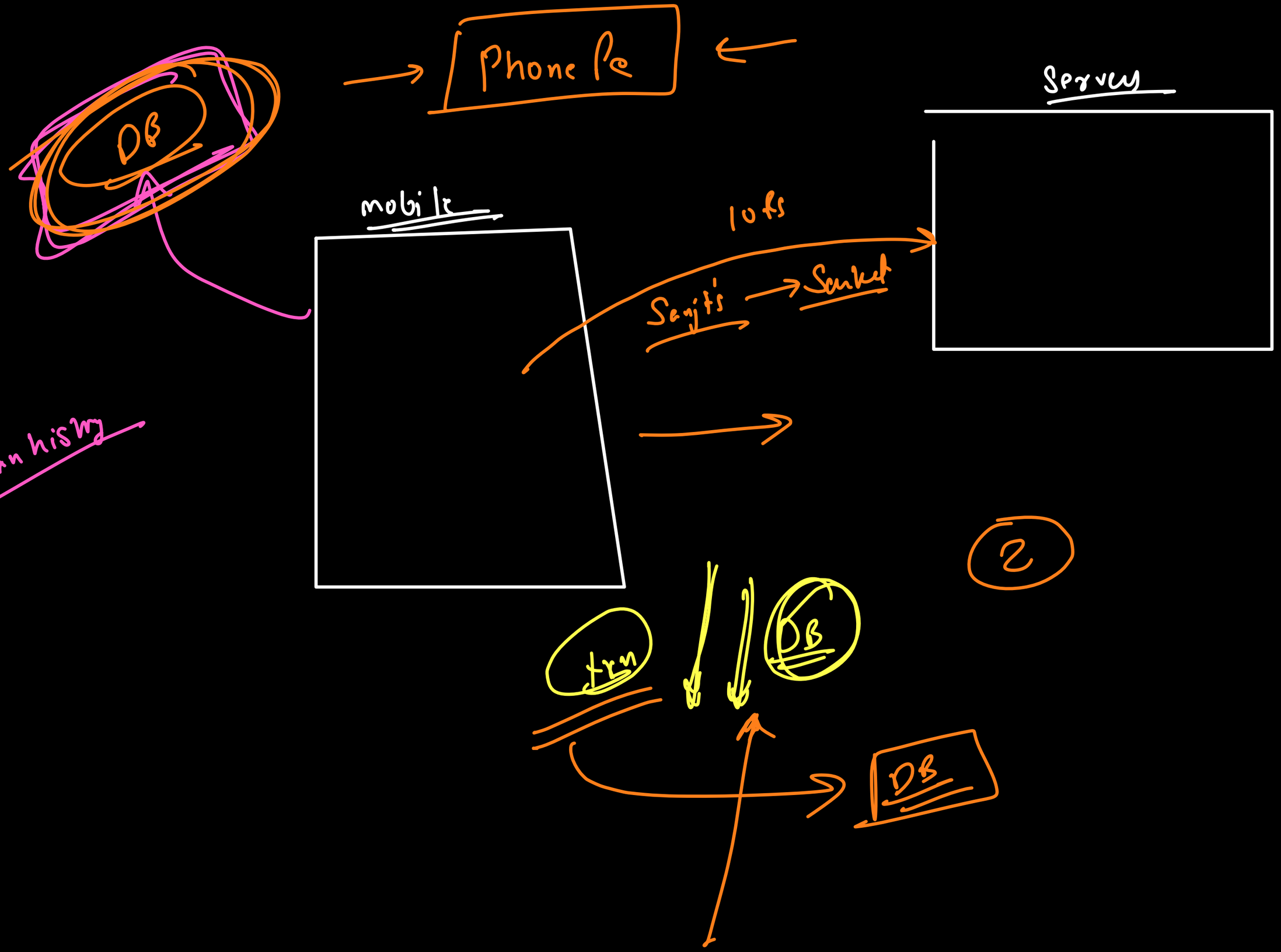
```
function outer() {
    let i = 0;

    function inner() {
        i += 1;
        return i;
    }

    return inner;

}
```

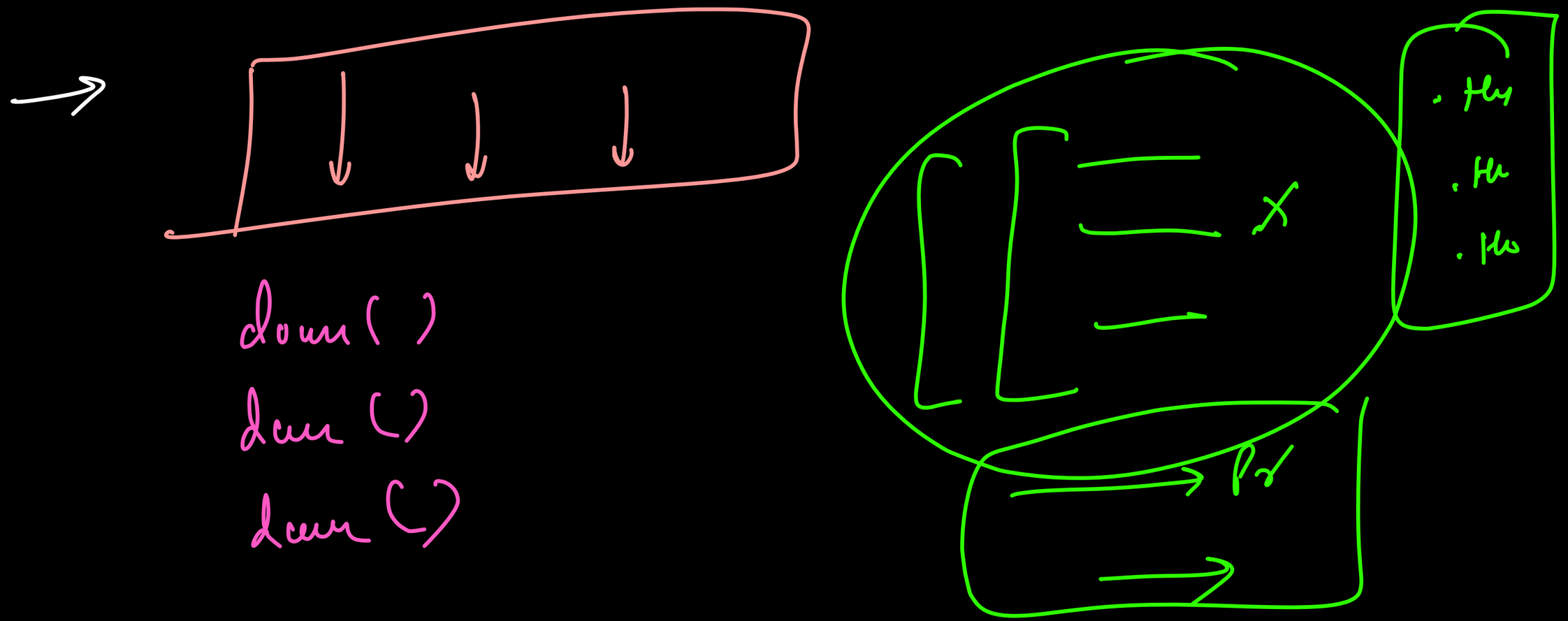
Closure is the mechanism using which a func^ remembers variables which might not be present in its local scope.

In closure, we close over variable instead of snapshotting the variable.

```
x = outer()
x();
```

inner





Mom
↓
3 brothers

a — 10m → grocery

b — 30m → HW

c — 1hr → drive

Imperative

declarative

Iterator

↳ declarative way of fetching data

Imperative

array

[1, 2, 3, 4, 5]

C++

for (i=0 ; i< arr.length ; i++) {

console.log (arr[i])

}

3

value

done :

next :

3

func^n

value → which value we are pointing at

next → fn using which we can get the v
iterator object's {value, done} pairs

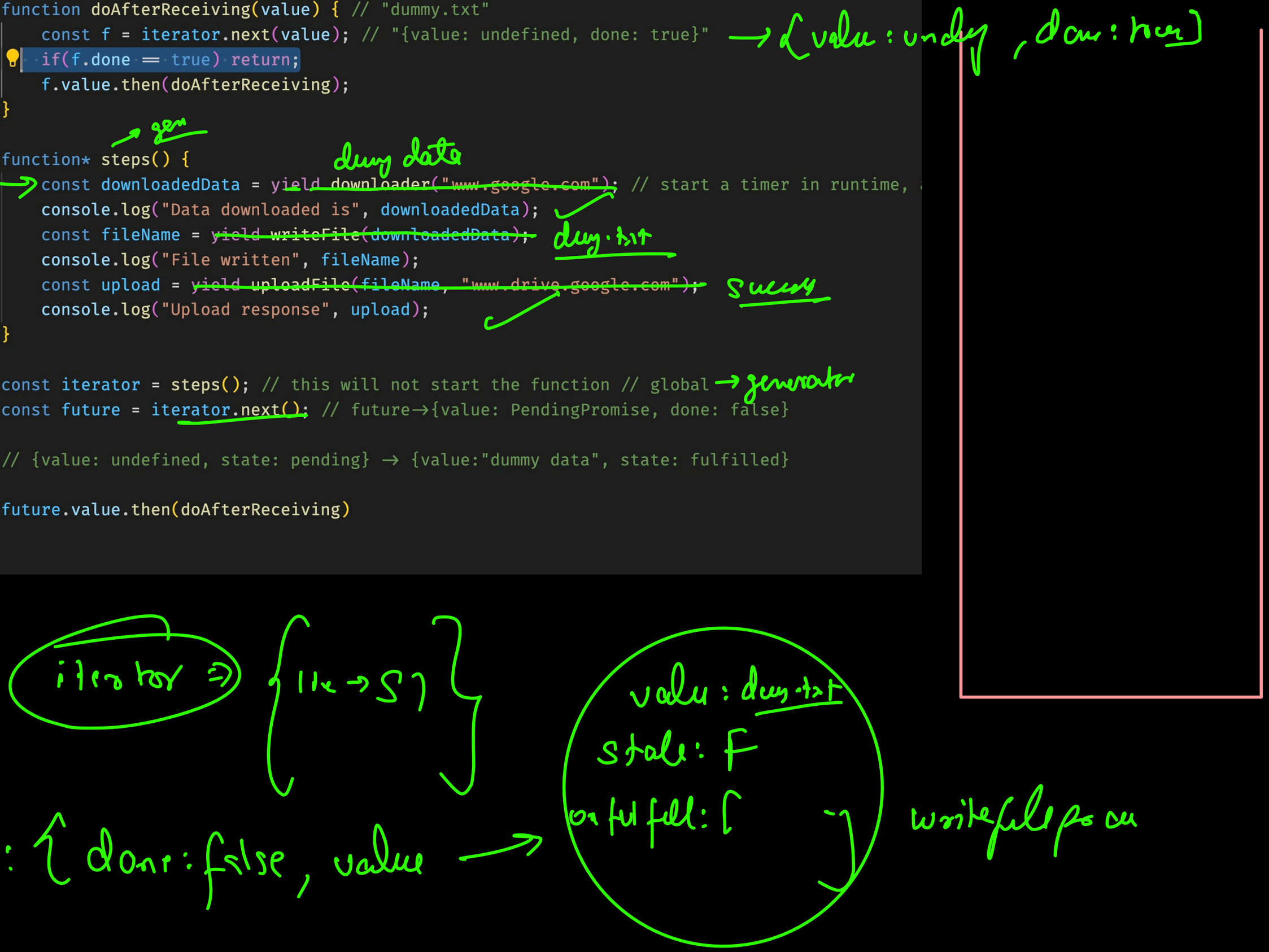
done → tells if more values can be fetched
or not

yield → it will immediately bring you out of a generator func^n

→ whatever value you yield will is given to the value parameter of the value object

→ if there are more yields left in the generator, done property is false.

```
40  function doAfterReceiving(value) { // "dummy.txt"
41      const f = iterator.next(value); // "{value: undefined, done: true}"
42      if(f.done == true) return;
43      f.value.then(doAfterReceiving);
44  }
45
46  function* steps() {
47      const downloadedData = yield downloader("www.google.com"); // start a timer in runtime,
48      console.log("Data downloaded is", downloadedData);
49      const fileName = yield writeFile(downloadedData);
50      console.log("File written", fileName);
51      const upload = yield uploadFile(fileName, "www.drive.google.com");
52      console.log("Upload response", upload);
53  }
54
55  const iterator = steps(); // this will not start the function // global
56  const future = iterator.next(); // future→{value: PendingPromise, done: false}
57
58  // {value: undefined, state: pending} → {value:"dummy data", state: fulfilled}
59
60  future.value.then(doAfterReceiving)
61
62
```



f : {done: false; value →

value : success

State : f

onfulfill: [catogory
reco]

upload process }

3S