Promises

→ Promises are special JS objects that are also considered readability enhancers. They get immediately returned from a funcⁿ setup to return a promise.

→ They act as placeholders for the data we hope to get back from some future Task.

→ We also attach the functionality we want to defer until the future Task is Done. And promises automatically handles execution of this functionality.

→ So promises do two things, one inside JS & one outside JS.

1) It signs up the process required to run in the runtime & gives a placeholder in JS, which has a value property.

How to create a promise ??

How to consume a promise ??

Promises

$$\begin{tikzcd}
 & \text{how to create these objs} \\
\text{promises are special JS objects} \arrow[ur] \arrow[r] \arrow[dr] & \text{how to consume} \\
 & \text{properties}
\end{tikzcd}$$

How promises work behind the scene ??

The promise object we create has 4 major properties

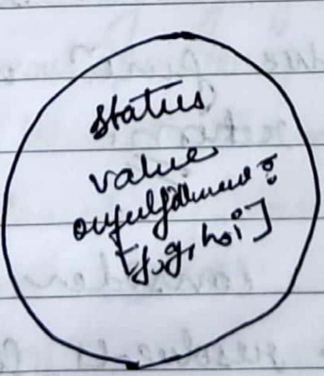
1) status/state
2) values
3) onfulfillment
4) onReject

x status/ → status shows current promise status state

1) pending state
2) fulfilled state → success
3) rejected state → error

* value → when status of the promise is pending, this value property is undefined. The moment promise is resolved status → fulfilled the value property is updated from undefined to the new value (this value we can consider as the returned value/ resolved value).
so the value property acts like a placeholder till the time promise finishes

* onfulfillment → This is an array, which contains functions that we attach to our promise object. (To a promise object we can attach some funcⁿ using .then() method). When the value property is updated from undefined, to the new value, JS gives chance to these attached funcⁿ one by one with the value property as their argument (If there is no piece of code in the call stack & global code left.)



```
for (i=0 ; i<10^10 ; i++)
  {

  }
```

classmate
Date
Page

Creating promise syntax
Promise constructor

```
new promise(function(resolve, reject) {
        //write here
 })
```

this constructor takes callback as argument

→ To create a promise call the promise constructor
→ The promise constructor takes a callback as an argument.
→ The callback passed inside constructor, expects 2 arguments resolve, reject → funcⁿ

→ Then inside the callback write your logic

→ If you want to return something on success, then call resolve funcⁿ with whatever value you want to return.

Q→ When do we consider a promise fulfilled??
→ If we call resolve() funcⁿ, we consider it fulfilled

→ we consider it rejected if we call reject()

creation of a promise object - is synchronous.

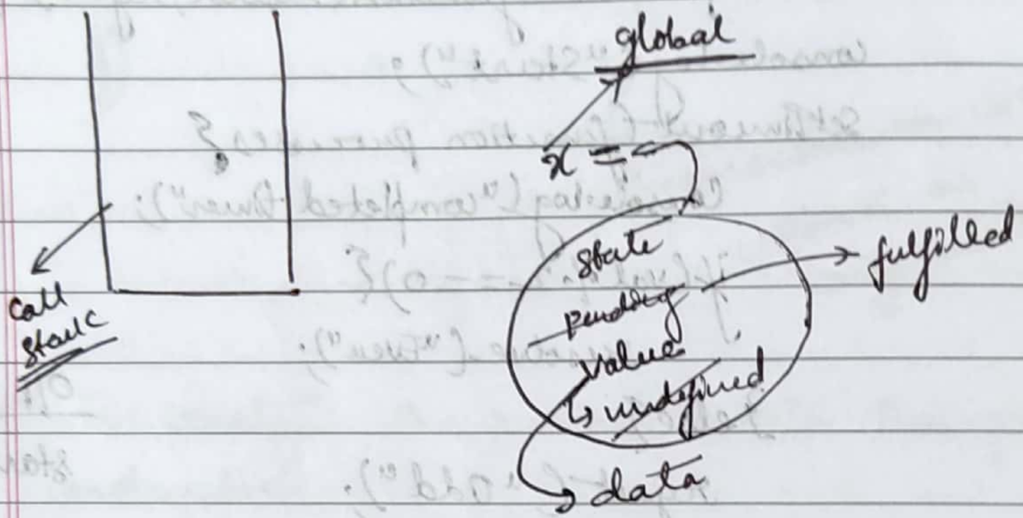
constructor
callback to constructor

```
function demo2(val){
    return new promise(function(resolve, reject) {
        console.log("Start");
        setTimeout(function process{
            Console.log("completed timer");
            if(val%2==0){
                resolve("Even");
            }else{
                reject("Odd");
            }
        },10000);
        Console.log("Somewhere");
    });
}
```

O/P
Start
Somewhere
Completed timer

```
function fetchData(url){
    return new promise(function(resolve,reject){
        Console.log("going to start the download");
        setTimeout(function process(){
            let data="dummy downloaded data";
            Console.log("download completed");
            resolve(data);
        },10000);
        Console.log("Timer to mimic download started");
    });
}
Console.log("Starting the program");
console.log("we are expecting to mimic a downloader");
```

x = fetchData("www.google.com");
console.log ("New promise object created successfully, but downloading still going on");



Starting the program
we are expecting .... downloader going to start the download
Timer to mimic download started
New promise obj.... on
download completed

consuming a promise

The promise consumption is the main beauty, using which we will avoid inversion of control.

Whenever we call a funcⁿ, returning a promise, we will get a promise object which is like any JS object that we can store in a variable.

→ Now the question, will JS wait here??

let response = fetchData("www.datadrive.com");

stores the promise object

↳ function returns a promise object

Q→ Will JS wait here for promise to be resolved if it involves any async piece of code??

→ If creation of promise involves sync piece of code it will wait, otherwise not.

```
function fetchData(url){
    return new promise(function(resolve, reject){
        console.log("Started downloading from", url);
        // setTimeout(function processDownloading(){
        // let data = "Dummy data";
        // console.log("Download completed");
        // resolve(data);
        // }, 7000);
        for(let i=0; i<1000000000; i++) {}
        resolve("dummy data");
    });
}
```

this callback is having a long sync piece of code, so JS will have to wait for promise object creation. And just after the for loop, we also resolve the promise so we get a resolved promise

```
function fetchData(url) {
    return new promise (function (resolve, reject) {
        Console.log ("started downloading from", url);
        SetTimeout (function processDownloading () {
            let data = "dummy data";
            Console.log ("download completed");
            resolve(data);
        }, 7000);
    });
}
```

Promise object will get created easily as there is no blocking piece of code, but Initially It will be pending. As the fulfillment happens after a time of 7 sec.

Now Technically, when promise gets resolved, we have to execute some functions

→ We can use .then() function on the promise Object, to bind the functions we want to execute once we fulfill a promise.

The .then() funcⁿ takes funcⁿ as an argument that we want to execute after promise fulfills, and the argument function takes value property as parameter

```
function fetchData(url) {
    return new promise(function(resolve, reject) {
        Console.log ("Started downloading from", url);
        SetTimeout(function processDownloading() {
            let data = "Dummy data";
            Console.log ("Download completed");
            resolve (data);
            Console.log ("hello");

           //resolve ("Sanket"); // these lines
          //resolve (12345);        will not be
                                    executed
        }, 7000);
    });
}
```

state: pending → fulfilled
value: undefined → data
onfulfillment: [ ]

hello

```
downloadpromise = fetchdata ("www.google.com");
(x) = downloadpromise.then(function f(value) {
            Console.log(value);
            return "Sanket";
      })
```

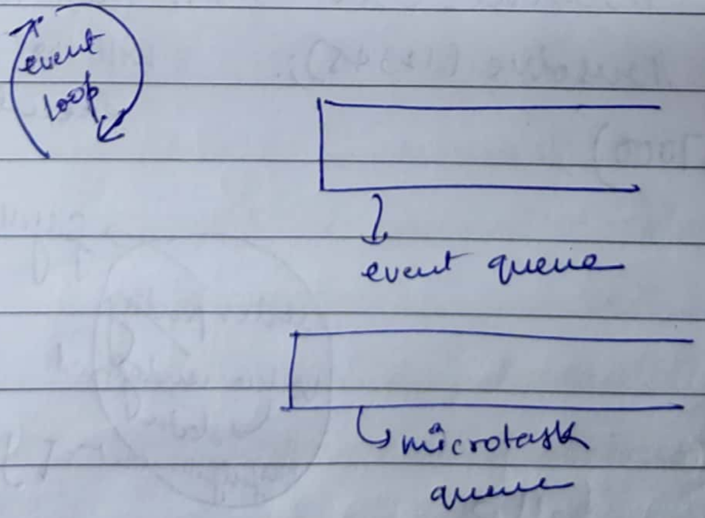
new promise

then .then function itself returns a new promise.

```
let downloadpromise = (fetchData) ("www.datadrive.com");
downloadpromise
.then(function processDownload (value) {
   console.log ("downloading done with following
                                       value", value);
   return value;
})
```

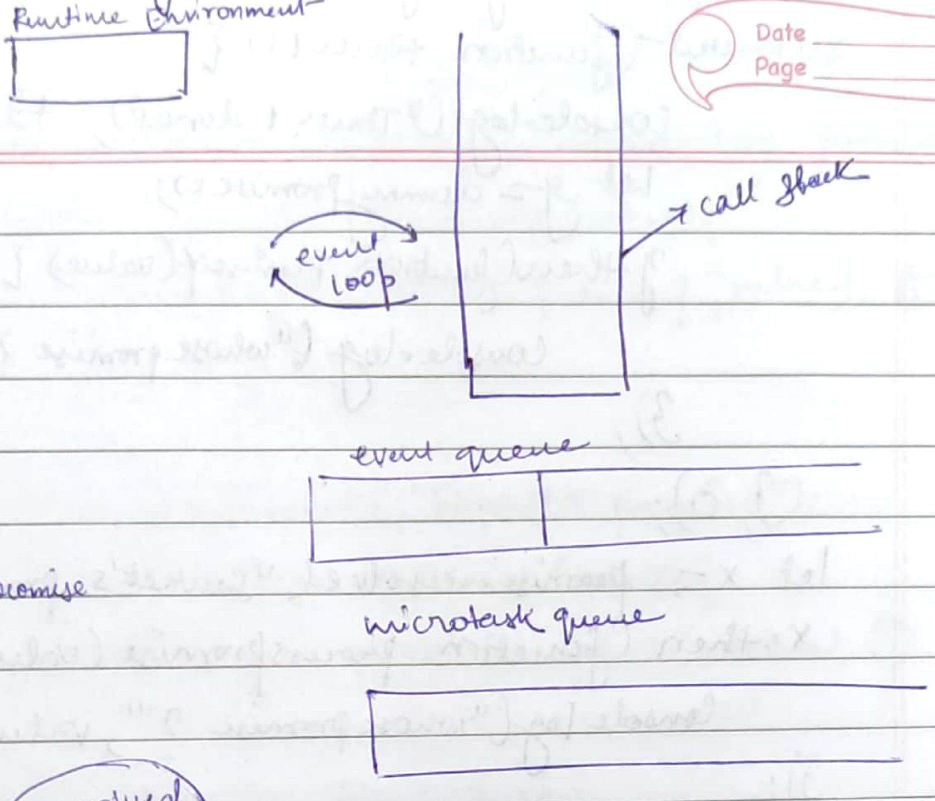
```
.then(function processWrite(value){
        return writeFile(value);
})
.then(function processUpload(value){
    return uploadData(value, "www.drive.google.com").
});
```

state
fulfilled



```
console.log("start of the file");
setTimeout(function timer1(){
        console.log("Timer1 done");
}, 0);
for (let i=0; i<1000000000; i++){
        // Something
}
let x = promise.resolve("Sanket's promise");
x.then(function processPromise(value){
    console.log("whose promise ?", value);
});
setTimeout(function timer2(){
            console.log("Timer 2 done");
}, 0);
```

Console.log("End of the file");

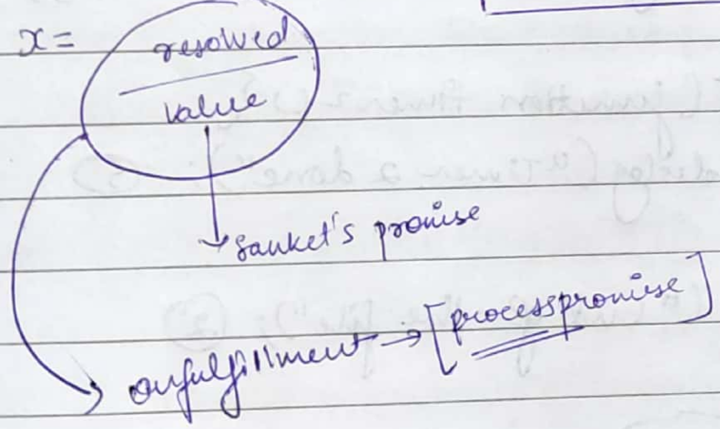


the start of file
end of ~~file~~ file

Whose promise? Sanket's promise

Timer 1 done

Timer 2 done



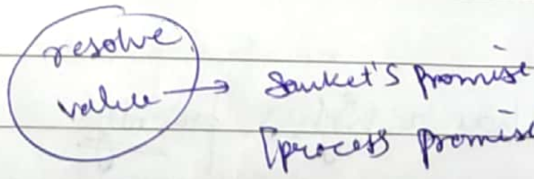
Microtask queue has a higher priority

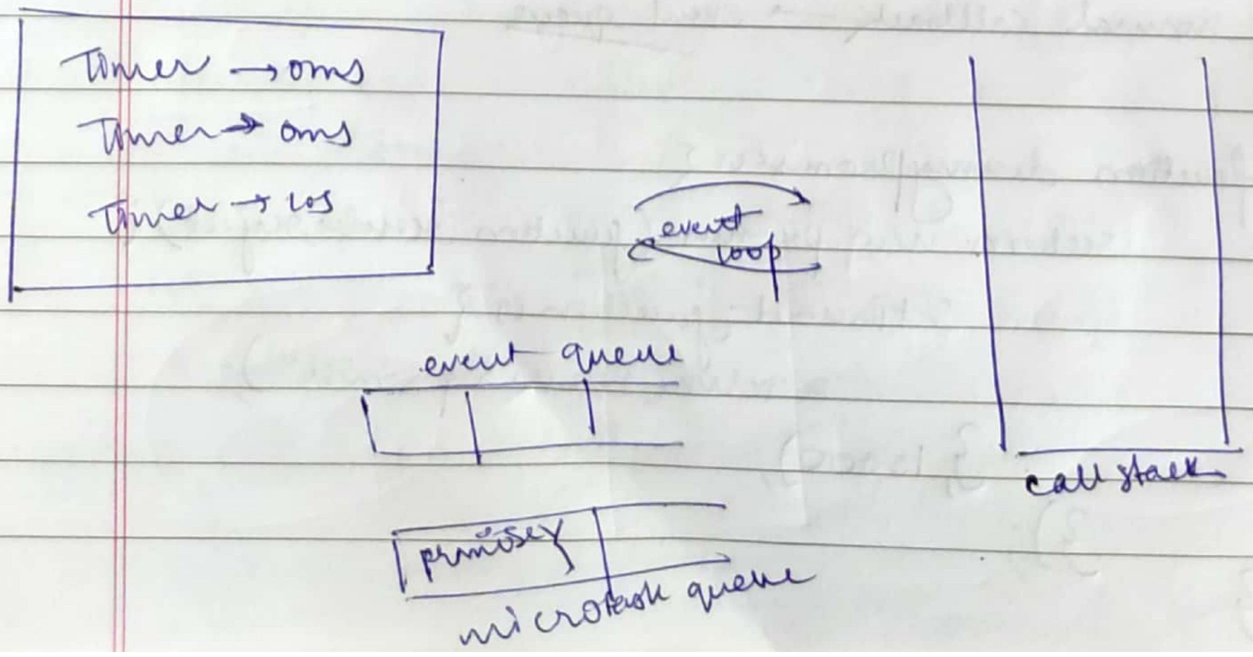
promise → callbacks → microtaskqueue

normal callback → event queue

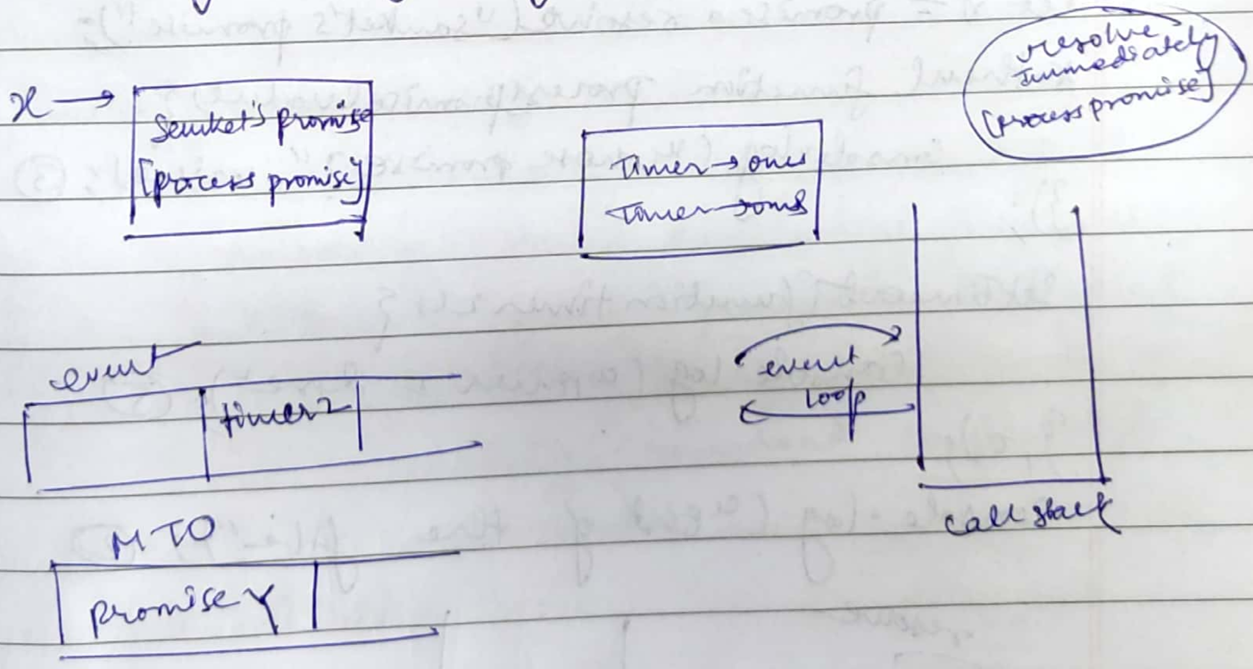
```
function dummyPromise() {
    return new promise(function (resolve, reject) {
            setTimeout(function() {
                resolve("Timer's promise");
            }, 10000);
        });
}
```

```
Console.log ("Start of the file"); — ①
SetTimeout (function timer1 () {
    Console.log ("Timer 1 done"); ④
    let y = dummyPromise();
    y.then(function promiseY (value) {
        Console.log ("whose promise ?", value); ⑥
    });
}, 0);
let x = promise.resolve( "Sanket's promise");
x.then (function processpromise (value){
    Console.log("whose promise ?", value); ③
});
SetTimeout (function timer2 () {
    console.log ("Timer 2 done"); ⑤
}, 0);
Console.log (" End of the file"); ②
```

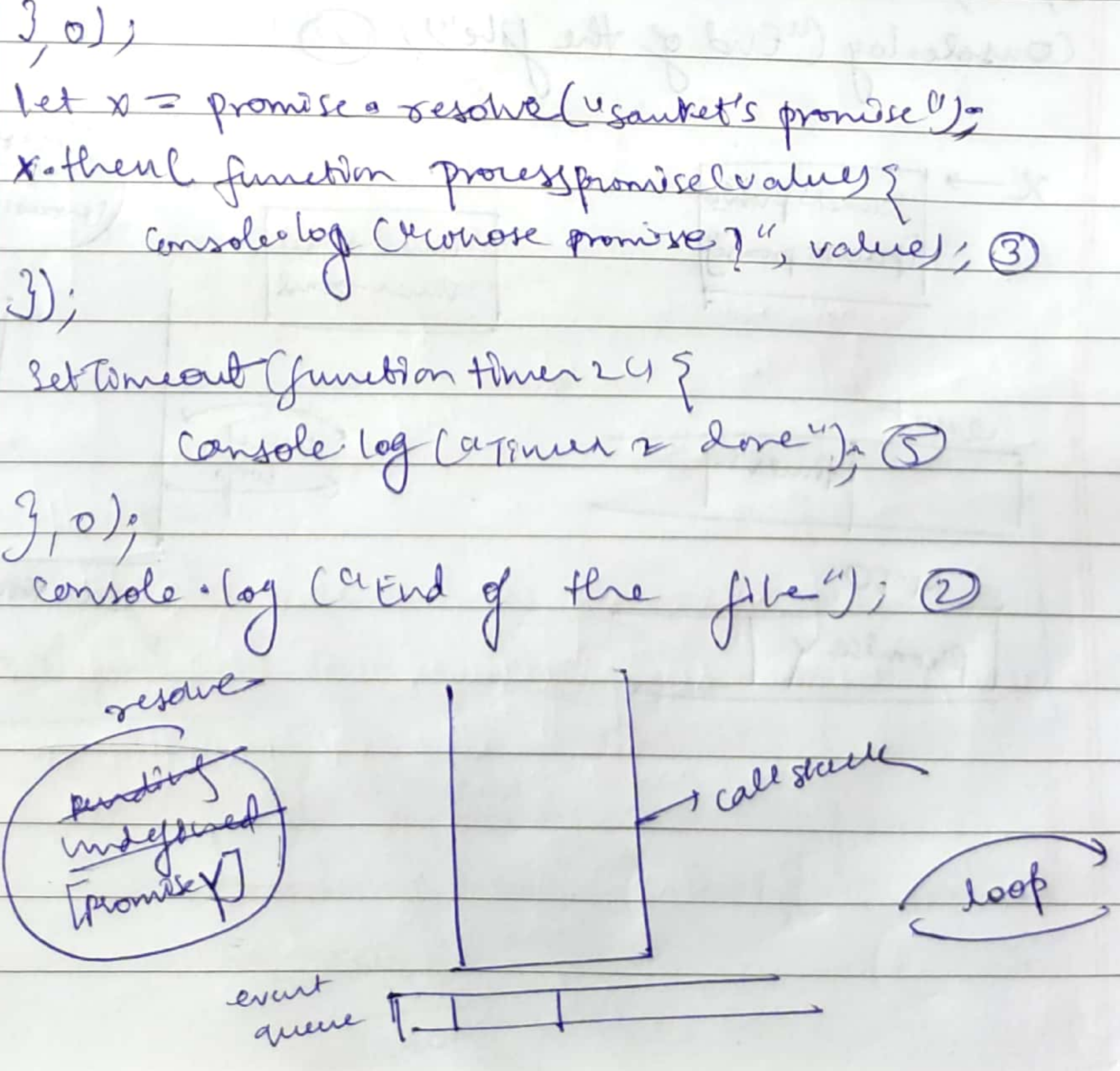




```
Console.log ("Start of the file"); ①
SetTimeout (function timer1() {
    console.log("Timer 1 done"); ④
    let y = promise.resolve ("Immediately promise");
    y.then (function promiseY (value){
        console.log ("whose promise ?", value); ⑤
    });
}, 0);
let x = Promise.resolve ("Sanket's promise");
x.then (function processPromise (value){
    Console.log ("whose promise ?", value); ③
});
Set Timeout (function timer2 () {
    console.log ("Timer 2 done"); ⑥
}, 0);
Console.log ("End of the file"); ②
```



```
function dummyPromise() {
    return new promise (function (resolve, reject) {
        setTimeout (function () {
            resolve ("Timer's promise");
        }, 0);
    });
}
console.log ("start of the file") → ①
setTimeout (function timer1 () {
    console.log ("timer1 done"); ④
    let y = dummypromise ();
    y.then (function promisey (value) {
        console.log ("whose promise ?", value); ⑥ Timer's process
    });
}, 0);
let x = promise.resolve ("sanket's promise");
x.then ( function processpromise (value) {
    console.log ("whose promise ?", value); ③
});
setTimeout (function timer2 () {
    console.log ("Timer 2 done"); ⑤
}, 0);
console.log ("End of the file"); ②
```



microtask queue |___|________

Timer → 0ms
Timer → 0ms
Timer → 0ms

x = resolve value → sainket's promise [process promise]

async & await

↳ we can declare a funcⁿ async
→ If you declare a funcⁿ async, it does the following →
① It allows the use of await keyword
② If you declare a funcⁿ async, it allows consumption of a promise using await
③ An async funcⁿ always converts your return value to a promise.

Ex
```
function fetchData(url) {
    return new promise (function (resolve, reject) {
        console.log ("started downloading from", url);
        setTimeout (function processDownloading() {
            let data = "dummy data";
            console.log ("download completed");
            resolve(data);
        }, 7000);
    });
}
```

```
async function processing() {
    console.log("enter processing");
    let value1 = await fetchData("www.youtube.com");
    console.log("youtube downloading done");
    let value2 = await fetchData("www.google.com");
    console.log("google downloading done");
    console.log("Exiting processing");
    return value1 + value2;
}
console.log("start");
setTimeout(function timer1() { console.log("timer1")}, 0);
console.log("after setting timer 1");
let x = processing();
x.then(function(value) {
    console.log("finally processing promise resolves with", value);
});
setTimeout(function timer2() { console.log("timer 2")}, 1000);
setTimeout(function timer3() { console.log("timer 3")}, 0);
console.log("End");
```

O/P

```
start
after seeing timer 1
Enter processing
started downloading from www.youtube.com
End
timer 1
timer 3
timer 2
```

```
Download completed
Youtube downloading done
Started downloading from www.google.com
Download completed
google downloading done
Editing Processing
finally Processing promise resolves with Dummy data Dummy data
```

→ Inside async funcⁿ this looks sync but over it's async