

Friday 10/11/01

JavaScript

Nanaste

20

Saturday (81-314)

8
9
10
11

→ Callback

Callback play a very important role in writing asynchronous code in javascript

21 Sunday (52-313)

Exp 1:-

```
console.log("Namaste")
```

setTimeout () => {

```
console.log('JavaScript')
```

3,5000)

FEB

M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	.	.	.

Week 8

President Day (USA)

```
console.log("Season 2")
```

Output:

Namaste

Season 2

JavaScript

Exp 2 :-

Ecomm website

Exp 2 :- Ecomm website

- 1. create order
- 2. Proceed to Payment
- 3. show order summary
- 4. update wallet

```
const cart = ["shoes", "pants", "Mushu"]
```

const cart = ["shoes", "pants", "kurti"] 4. update wall

23

(54-311) Tuesday

```
api.createOrder (cast, c) => {
```

api.proceedToPayment () => {

```
api: showOrderSummary ( ) => {
```

```
api.updateWallet()
```

3) γ

3)

M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

February

24

Wednesday (55-310)

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

7

8

9

10

11

12

1

2

3

4

5

6

01

Labour Day (Australia); Republic Day (Switzerland)

Once we have data inside the Promise object what will happen is above callback function that we attach to this promise object will be automatically called.

02

Tuesday (61-304)

```

11  const cart = ["Shoes", "pants", "shirts"]
12  → call back
13
14  createOrder (cart, (orderId) => {
15    proceedToPayment (orderId)
16  });

```

9																															
M	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	

2010

03

(62-303) Wednesday

Week 9

Hina Matsuri (Doll Festival) (Japan); Martyr's Day (Malawi)

→ Promises

```
const promise = createOrder (cart);  
promise.then (c => {  
  proceedToPayment (c.orderId)  
});
```

→ promises is very much better than callback⁶

→ In callback we passed function to create orders API and we were blindly trusting create orders API we were relying on it

→ In promises we are attaching a callback function to a promise object

→ In callback we passed function and now
create order API would have called it whenever
it wants to

→ In promises we have the control of our program with us. create order API will only do its job it will create an order and it will fill the promise object with the data the order ID whenever it wants to and as soon as this promise object is filled with that data it will automatically call our callback function.

→ promises give guarantee as soon as we have data inside promise it will call the function definitely 100 percent of the time it will call it just once and only once

T F S S M T W T F S S M T W T F S S M T W T F S
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

March

05

Friday (84-301)

Week 9



Promise Object

{

```
[[Prototype]]: Promise
[[PromiseState]]: "pending"
[[PromiseResult]]: undefined
```

}

→ promiseResult will store whatever data it returns whatever the data that fetch method will return will stored inside promise result.

06

Saturday PromiseState will tells you that what state that promises is initially the promise will be in pending state and once we have got the data back then promise state changes to fulfilled state

Eg :-

```
const GITHUB_API = "
const user = fetch(GITHUB_API)
console.log(user);
```

2010

08

Monday (67-298)

Week 10

International Womens Day (Uganda)

```
user.then(function (data) {
  console.log(data);
});
```

3);

→ These are three state PromiseState pending, Fulfilled and rejected

→ promise objects are immutable so whenever it is fulfilled or whenever we have data inside our promise object we can just pass it here and there in our code and we don't have to worry about that someone can mutate that data (change the data)




What is Promise

→ The Promise object is a placeholder which will be filled later with a value makes sense promise object is a placeholder for a certain period of time until we receive a value from a asynchronous operation.

10
Wednesday (69-296)

OR

→ A promise is an object representing the eventual completion or failure of an asynchronous operation.

1  Promise also resolve the
2
3 Problem of callback hell
4

5 Eg:- callback

```

1 createOrder (cart, function (orderId) {
2     proceedToPayment (orderId, function (paymentInfo) {
3         showOrderSummary (paymentInfo, function () {
4             updateWalletBalance();
5         });
6     });
7 }
8 );

```

$\xrightarrow{10}$ promise

```

11 createOrder (cost)
12   .then (function (orderId) {
1   return proceedToPayment (orderId);
2   })
3   .then (function (paymentInfo) {
4   return showOrderSummary (paymentInfo);
5   })
6   .then (function (paymentInfo) {
7   return updateWalletBalance (paymentInfo);
8   });

```

M A	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W
R	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

12
(71-294) Friday

Independence Day (Mauritius)

→ promise resolve the problem of callback hell with the help of Promise chaining.

➡ Creating the promise

```
const cast = ["Shoes", "pants", "Kuola"]
```

```
const promise = createOrder(cart)
```

```

        orderId
        promise.then(function() {
            // proceedToPayment (orderId)
        })
        3) console.log (orderId)
    }
}

```

// creating createOrder API (producer)
// createOrder will called below function

```
function createOrder(card) {
```

```
const pr = new Promise(function(resolve, reject) {
  // create codes
  // validate cast
  // codesId
  if (!validateCast(cast)) {
    const err = new Error("Cast is not valid")
    reject(err);
  }
})
```

F	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

15

Week 11

16

Gudi Padva (India)



M	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W
A	R						7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

2010

17

(76-289) **Wednesday**

Week 11

St. Patrick's Day (Switzerland, UK, USA)

⇒ Promise Chaining

T F S S M T W T F S S M T W T F S S M T W T F S

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

18

(77-288) Thursday