const get DataAfterPause = throttleClickfunc(getWD,1000)

Interview questions on debouncing and throttling
Debouncing → flipkart search
Throttling → Twitter scroll bar
Implement throttle, debounce, throttle pollyfill, debounce pollyfill
Refer to github JS Topicwise

18/3/24 Javascript S2 - Ep.0 (Asynchronous JS)

Callback hell

Case study: We are using a e-commerce website, we had some api which has create cart, proceed to payment, order summary and update wallet fns, but they should be executed in order (after completion of prev fn)

Soln:
Callbacks are the functions which are called back after completion of certain parent function

Good part:
Asynchronous JS can be done with call backs
eg: var cart= ["shoes", "bag"];

```
    api createOrder ( cart, function() {
  api. proceedtoPayment ( function() {
      api ordersummary (function() {
      api.updatewallet ();
      }
    }
  }
}
```

Bad part / problematic part:
a) Callback hell:
Our code grows horizontally which is unmaintainable and unreadable. This situation is also known as pyramid of doom

b) Invension of control:

As we have given our written code to some random function and entrusting it to call back the function, It can have some bugs (the parent fn). Here we are losing control our code.

Hence promises come into picture.

Promises.

* Promises are used to handle async operations

Case Study: We have an e-commerce website, where we will be creating a cart and proceeds to payment.

Soln: Use promise to avoid call back hell and inv of control

a) In callback hell, our function is passed to the parent fn

```
api.createOrder(cart, function (orderId){
        proceedToPayment (orderId)
    });
```

What can happen?

createOrder fn may call cb fn twice/never call (or) we are definitely unsure. But what we sure is we want to call only once, therefore we will attach it to a promise

b) Promise makes the fn attached to it and calls when only the promise is resolved

```
Const promise = createOrder(cart) // returns promise
```

promise is basicaly an empty object initially but filled with data after resolving promise

```
promise.then (function (orderId) {
        proceedToPayment (orderId)
    });
```

Understanding promise object in browser

Eg: Use fetch to get github data for a profile. fetch returns the promise.

```
Const GITHUB_API = " api.github.com/pabhinav1999"
const userProm1 = fetch( GITHUB_API);
Console.log (userProm1);
```

As soon as you observe the userProm1 in chrome, it displays Promise {<pending>}; but if you expand the object it may say fulfilled. (This is one observation)

```
Promise {
  Prototype: promise
  State: fulfilled
  PromResult: Response
}
```

you can extract data from response through json

JS guarantees that promise can execute only once

Promise has 3 states: pending, fulfilled and rejected and they are immutable object.

Definitions

→ Promise obj is a placeholder for a certain period of time until we receive data from a asynchronous operation.

→ A Container for a future value.

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

19/3 Promise Chaining

Promise chaining helps in avoiding pyramid of Doom..

~~Eg. for promise chaining~~

~~createOrder(cart, function (order Id) {~~
~~proceedPayment{ ~~paymentInfo~~, fn orderSummary (~~
~~ord~~

Eg. for Promise Chaining

```
createOrder( cart, function(orderId) {
    proceedPayment( orderId,  function( paymentInfo) {
        orderSummary(paymentInfo, fn updateWallet() {
            updateWallet();
        }
    }
}) )
```

⇓ Converting to promise chaining

```
createOrder (cart). then ( function (orderId) {
    return  proceedPayment (orderId)
}). then ( ~~orderSummary~~ function ( paymentInfo) {
    return  orderSummary (paymentInfo)
}). then ( function () {
    return updateWalletBalance();
    }
```

Make sure to return, otherwise we will be losing data.

Creating a Promise and Error Handling

While chaining promise from one chain to another we can return data or promise

21/3 Advanced Promise Chaining / Advanced Error Handling

whenever there is a big promise chain, the chain at the last which has catch handles any chains error

* What to do if we want to proceed further if any of the step fails?

A: Then catch ~~should~~ be placed below then handles which will that error and any then below catch will process further.

Developer responsibility to see where your catch fits.

~~Promise APIs~~.