Promises

→ Why need of Promises?
There are two problems that exists in callbacks i.e
① Inversion of control (biggest problem)
② Callback hell
Now, to resolve all of this, we have 'promises'

→ Let's see, what is 'Promises'?
Promises are nothing, they are 'readability enhancers.'
Readability → How readable our code is and promises enhances the readibility of our code.

Apart from this, Promises also solves the problem of 'inversion of control' in callbacks.

"In Javascript, promises are special type of objects that get returned immediately when we call them."

Promises are objects, nothing more nothing less.

The moment we call a new promise, they get immediately returned.

{ We will talk about immediately return later }

→ "Promises act as a 'placeholder' for the data we hope to get back sometime in future."

Let's assume, we have a function 'fetch'

```
x = fetch("http://www.xyz.com")
```

↳ assum 'fetch' is written using promise, then it will immediately return a promise object which will acts as a placeholder (i.e x)

This 'fetch()' function will download the content of the url. Now downloading is a time consuming task. We can't expect this 'fetch()' immediately complete. And we know that Javascript is **synchronous** in nature.

It means JS will not wait for this 'fetch()' function and move forward because 'fetch' is not a native Javascript call.

Now, lets assume 'fetch' is written using promises, then when javascript will come to the call the 'fetch' function, it will just going to immediately return a promise object (i.e x) and this promise obj. is going to act as a placeholder.

Page No.
Date / /

Immediately return

Example

```
x = fetch("http://www.xyz.com")
```

Placeholder

placeholder is kind of a temporary variable for the result

→ "In these promise objects we can attach the functionality, we want to execute once the future task is done."

Means, when the future task of promise is done once, then please execute my algorithm (It can be your logic or any functionality that you want to do).

→ "Once the future task is done, promises will automatically execute the attached functionality."

This above line is giving the feeling of something - something 'callbacks'.

"Promises returns an object but callbacks are not"

* For understanding promises, we have to understand two things
  1. How we can create a promise??
  2. How we can consume a promise??

Date / /

Let's understand the basic meaning of promise.

Promise

↳ It is kind of like an aggrement or commitment

↳ Mutual aggreement between two entities

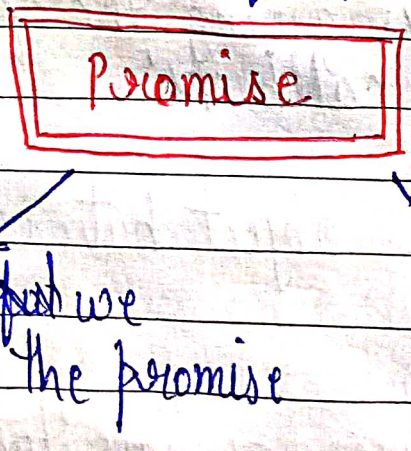
- For example:—

I am giving you a promise that I will do something for you.

But but but..... Promises can be break also 😔

So it means, there can be a situation where promise can be fullfilled

Or there can be a situation where promise is not fullfilled.



→ How to create a promise?

↳ creation of a promise object is synchronous in nature.

If there is some other functionality is clubed with promise object that will also synchronouse, it is not necessary. means functionality can be asynchronous in nature.

For example:-

From previous 'fetch' function -

Let's say the 'fetch' function returns a promise, so this 'fetch' function will create a 'promise' and that promise is downloading the data from 'xyz.com', Now downloading part will takes its time, its an asynchronous process because its a runtime feature. But the creation of the promise object that is returning the placeholder immediately.

Immediately return

```
x = fetch("http://www.xyz.com")
```

placeholder

This placeholder object return immediately because this is native to javascript and if anything native to javascript, it is synchronous in nature.

So just the creation of the promise object is synchronous in nature. {checkout official documentation}

→ Any promise object is one of three mutually exclusive states :

1. fulfilled
2. rejected and
3. pending

} Three states of Javascript promises.

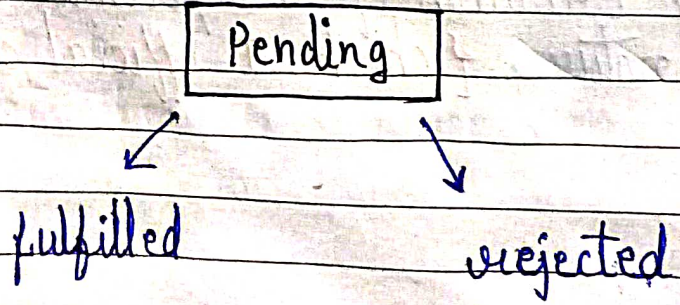
1. **Pending** :→ When we create a new promise object this is the default state
   It represents work in progress.

The moment we create a promise object, the state of a promise object by default is 'Pending'

2. **Fulfilled** :→ If the operation is completed successfully, in that case the state of the property of the promise object will migrate from 'Pending' to 'fulfilled'.

3. **rejected** :→ If operation was not successfull, then it will go in a 'rejected' state.

{ It means, the state property can migrate from 'Pending' to either 'fulfilled' or 'rejected' }



→ How to create a new promise object.

There is a keyword 'new' then we write 'Promise', this is technically a constructor of the JS object, promise object

Constructor is a special function using which a new object is created.

Now this 'Promise' expect a function, a callback function.

Example

constructor

callback function

keyword

```
new Promise ( function ( ) {

})
```

→ The callback function we are passing takes two parameters

1. resolve
2. reject

} parameter naming can be change, it can be a & b.

Example:-

```
new Promise ( function ( resolve, reject ) {
    // time consuming task
})
```

Inside this callback function we can write our time-consuming task. This time consuming task can be a big for loop or a timer or may be doconloading some data and even non-consuming task can also be added.

Date / /

→ Let's talk about 'resolve' and 'reject'
'resolve' and 'reject' both are of them are functions, kind like of inbuilt functions.

If we call this 'resolve' function anytime, the moment resolve function is called, the 'Promise' will go from 'Pending' state to 'resolve' state

Pending ⟶ resolve state

And if any time, we call 'reject' function instead of 'resolve' function, then promise will go immediately in the 'rejected' state.

Pending ⟶ rejected state

And if we don't call anyone, the promise will stay forever in 'Pending' state.

Note:

↳ Apart from this state property, our promise has also a 'value' property. Till the time the state is 'pending' the 'value' is undefined

Pending ⟶ undefined (value)

The moment, we go to ~~full fills~~ fulfill our reject, then this value can change.

Date / /

So this 'resolve()' function, if we call it with some value 'x' that value can be string, number, boolean, null, undefined, doesn't matter. The moment we call the 'resolve()' function, the promise moves to a 'fulfilled' state, the 'value' gets updated with the argument of 'resolve()' function.

Example:

```
new Promise( function (resolve, reject ) {
    // inside the function we can write
       any time consuming task.
    resolve (x);
})
```

x → value

" With whatever argument we call 'resolve' or 'reject' with gets assigned to the value property "

→ If we pass multiple values in 'reject' or 'resolve' function, it will going to set always the first value, it will take only first value.

Example:-

```
resolve (10, 20, 30);
```

multiple values.

value will be '10' because it is the first value.

Remember " Once you changed the state of promise object, it can never be updated. "