1] How we Can Create a Promise ?.

2] How Can We Consume a Promise ?

Promise (→ generally it means suppose for a example we give the Promise that we will do lot of hard work and I promise that I will make the extra effort, so the person will take the promise., but but promises Can break so there might be promises Can be fullfilled and promises Cannot be fullfilled.).

Promise

Can be fullfilled — Cannot be fullfilled

1] How to Create a Promise ?

→ Promise are NATIVE to Javascript as because we Can see about the promise in ECMA docs

So Creation of a Promise object is SYNCHRONOUS in nature., ONLY Creation is synchronous.

There are Three States in promises

i) Pending ↓ When we Create a new Promise object this is a default state it represents WORK IN PROGRESS

ii) fullfilled ↓ If the operation is Completed successfully then only this state is achieved

iii) Rejected ↓ If the operation is not successfull it will achieve the Rejected state.

new Promise.(f) → this Constructor excepts the Callbacks

new Promise (function ( ) {
    }) → this callback function that we are passing takes two parameters

new Promise (function (resolve, reject ) {
// inside this function we can write over time Consuming tasks
    }) ;

resolve, reject these two are actually the functions only

* If the function resolve() is called inside then the Promise go to the fullfilled state
* If the functions reject() is calleded inside then the promise will go to the rejected state.

OR

* If we don't Call resolve() OR reject() the Promise fill forever stay in pending state

this was all about STATE property

Now Let's talk about VALUE property.

* Till the time the state is at the pending state, the Value will be UNDEFINED (pending state → Value: Undefined)

* the moment we go to the fullfilled state or Rejected state then this Value will change

suppose if we are calling the resolve function with same Value x, and that value can be string, number, boolean, null, undefined doesn't matter what, so whatever the argument we call the resolve function with after moving to the fullfill state the value of property get's updated with the argument of resolve

Same applies to Reject function too

For Example :

```
function getRandomInt (max) {
        return Math.floor (Math.random() * max)
}

function createPromiseWithLoop () {
return new Promise (function executor (resolve, reject) {
    for (let i=0; i<1000000; i++) { }
    let num = getRandomInt (10)
    if (num % 2 == = 0) {
        resolve (num)  // if the random number is even
                          we fulfil
    }
else {
        reject (num)  // if the random number is odd
                         we reject
    }
})
}

Let x = createPromiseWithLoop()
    Console.log (x)
```

for loop

currently in above example the for loop is running
So after the running of for loop at the moment
we will get the o/p.
if the random number was even → Promise { 4 }

if the random number was Odd → promise { <rejected> 3 }

and here it initially won't print anything because the for loop is taking time, once the for loop execution is done we will get the o/p, so we can say whenever we create a promise they are synchronous in nature so let us better understand with one more example

Let's try with setTimeOut instead of Loop

Example 2 :-

```
function getRandomInt(max) {
return Math.floor(Math.random() * max)
}

function CreatePromiseWithTimeOut() {
return new Promise(function executor(resolve, reject){
setTimeout(function() {
let num = getRandomInt(10);
if(num%2==0) {
resolve(num)
}
else {
reject(num)
}
}, 10000)
}
```

10 secs

```
Let y = Create Promise With Time Out ()
console.log (y)
```

o/p → Promise {<pending>}

As we can see this time it immediately gave us pending state, as we had written return new Promise so the new Promise is expected to return, then there is Call back function from that function will be immediately called, and this function says SetTimeout, the moment Javascript sees SetTimeout it will go to the runtime and say hey runtime. Start the timer of 10 secs, and thats it after that it immediately prints Promise {<pending>} because Promise is SYNCHRONOUS and NATIVE to Javascript and ~~after the 10 secs gets~~ Completed and is last example the for loop was the blocking Piece of code ~~so it would~~ and as we know for loop is Native to JavaScript so after the Completion of for loop we Would get the o/p.

Here after Completion of 10 secs, the setTimeOut() is Completed so the Callback of setTimeOut() will be Called. and it creates a random number and then that number if it was even output would be Promise {4} if it was odd output would be Promise {<rejected> 3}

Example 3 :-

function createPromi

suppose in previous example we had written
resolve(num) and if the num was even we would get the o/p promise { 4 }

What if we pass multiple arguments into resolve for example resolve(num, 10, 20). and o/p remains the same if we get num as even o/p will be Promise{8} hence those arguments won't affect or we can say they are not considered SAME with reject too

Example 3 :

What happens if we don't write resolve() and reject()?

```
function CreatePromiswithTimeOut () {

return new Promise (function Executor(resolve, reject)
{

setTimeout (function () {
    Console.log("fullfilling")
    return num
})

setTimeout (function () {
    Let num = getRandom(10).          → gives the random number have not written the function of getRandom refer it for prev examples
    if (num%2 == 0) {
    Console.log("fullfilling").
        return num
        }
    else {
        console.log("rejecting").
        return num
        }.
    }, 10000 )
})
}
```

```
let y = createPromiseWithTimeOut();
console.log(y)
```

o/p → starting it will ~~print~~ print promise? <pending> }
after completion of 10 secs, it prints fullfilling

o/p → fullfilling, and the state of the Promise will be still in Pending, because the State of Promise will only change if we call resolve() or reject() function., if you don't call it will never change the state.

Example 4 → What if we write something after resolve() and reject().

```
setTimeout(function() {
  let num = getRandom(10)
  if (num%2 ==0) {
    resolve(num).
    console.log("completed")
  }
  else {
    reject(num)
    console.log("rejected")
},10000)
```

o/p → if the num was even the o/p would be

```
Promise { 6 }
completed.
```

> so whatever we write below resolve, that also will be executed

Suppose if the num was odd the o/p would be

promise { <rejected> 3 }
rejected

Nothing but we can say it works as Synchronous Code.

Example 9 → What if we do resolve() or reject() two times

```
setTimeOut (function () {
  let num = getRandomInt (10)
  if (num % 2 == 0) {
    ~~console.log~~
    resolve (num)
    console.log ("completed resolving")
    resolve (10)
    console.log ("resolving again")
  }
  else {
    reject (num)
    console.log ("completed rejecting")
    reject (10)
  } console.log ("resolving again")
}, 10000)
}
```

let y = createPromiseWithTimeout();
console.log(y);

o/p → Initially we will be having Promise {<pending>}
after 10 secs

o/p → Completed resolving
resolving again.

BUT Let's see what is in the Promise.

in the promise we have
Promise {<fulfilled>: 6}.
Which was from the random generated num. It is not
→ 10 (which was of one more resolve)

Note → That means ~~Once~~ Once we Resolve(). or Reject(), the it can. NEVER be updated.