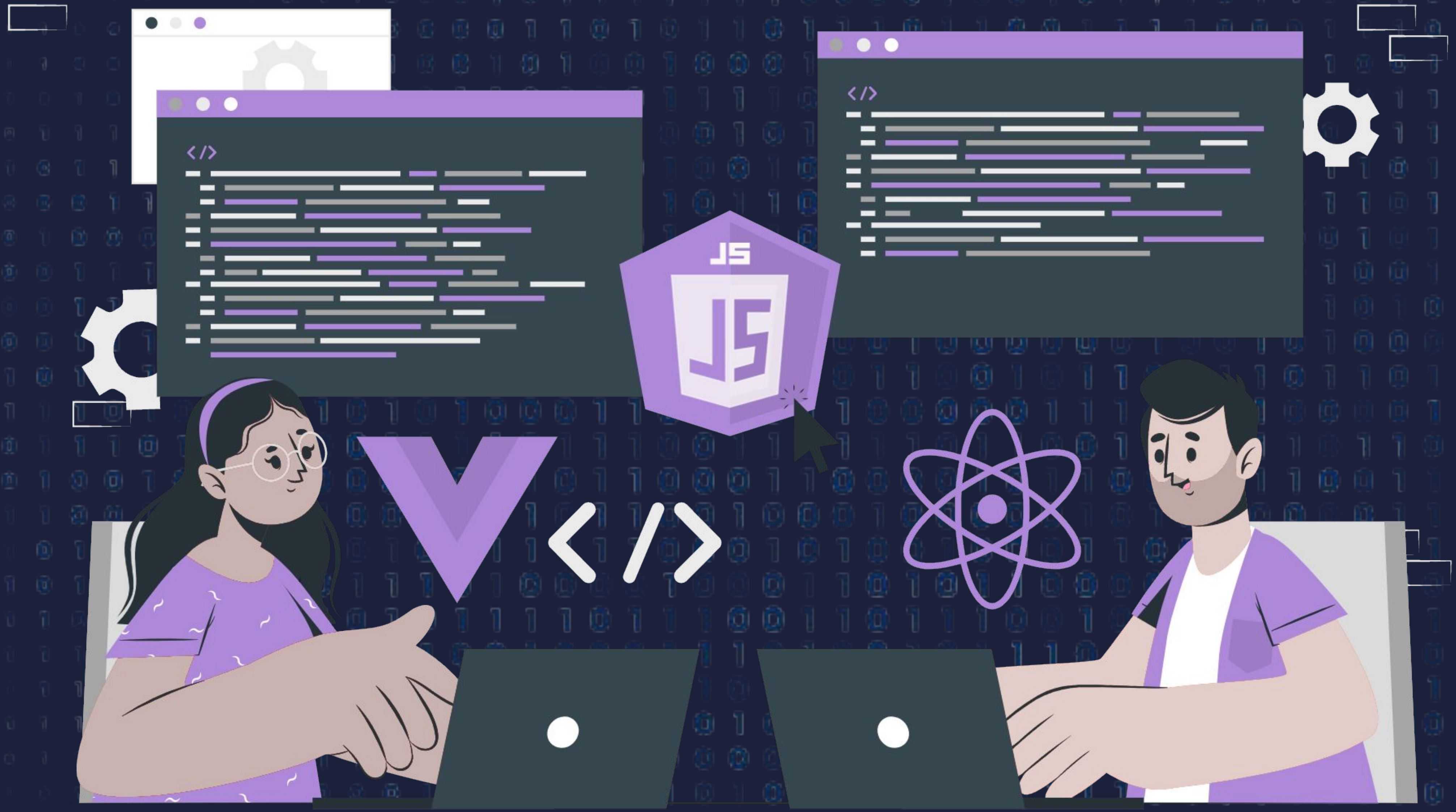
PW SKILLS

Promise Constructor



Lecture CheckList

1. Introduction to promises.
2. Importance of Javascript Promises.
3. Understanding promises.
4. Promise Lifecycle.
5. Promise constructor.
6. Consuming the Promise values.

Introduction to promises

Javascript is single-threaded, it can execute only one code statement at a time. Some pieces of code execute immediately and some take time. The operation of fetching data from the server is not instantaneous. In that case, the execution thread would be blocked, leading to a bad user experience due to slow loading. We can solve these problems through Promises.

A promise is an object that represents a value that may not be available yet but will be available at some point in the future. Promises are a way of handling asynchronous code, which means code that runs in the background while another code is executing. With promises, we can write code that waits for the completion of an asynchronous task before moving on to the next task.

Introduction to promises

A promise is an object that represents a value that may not be available yet but will be available at some point in the future. Promises are a way of handling asynchronous code, which means code that runs in the background while another code is executing. With promises, we can write code that waits for the completion of an asynchronous task before moving on to the next task.

Importance of Javascript Promises

Here are some of the reasons why promises are important.

- Promises are an effective way to handle asynchronous code in Javascript. With promises, we can write code that waits for the completion of an asynchronous task before moving on to the next task. This leads to more efficient and responsive code, as well as a better user experience.

- In the previous lecture, we looked into callback hell, where multiple nested callbacks make code hard to read and debug. Promises provide a cleaner and more manageable way to handle asynchronous code than traditional callback functions.

- Promises can be chained together to handle multiple asynchronous tasks in a more readable and manageable way. This can lead to more efficient and maintainable code, making it easier to debug and improve over time.

Importance of Javascript Promises

- Promises come with an inbuilt error-handling mechanism, we can handle both expected and unexpected errors in a consistent way. This makes it easier to identify and debug errors in your code.
- Promises are widely used. This means that developers can use promises in their code regardless of the libraries or frameworks they are using.

Understanding promises

The most effective way to understand javascript promises is by relating them to a real-life promise.

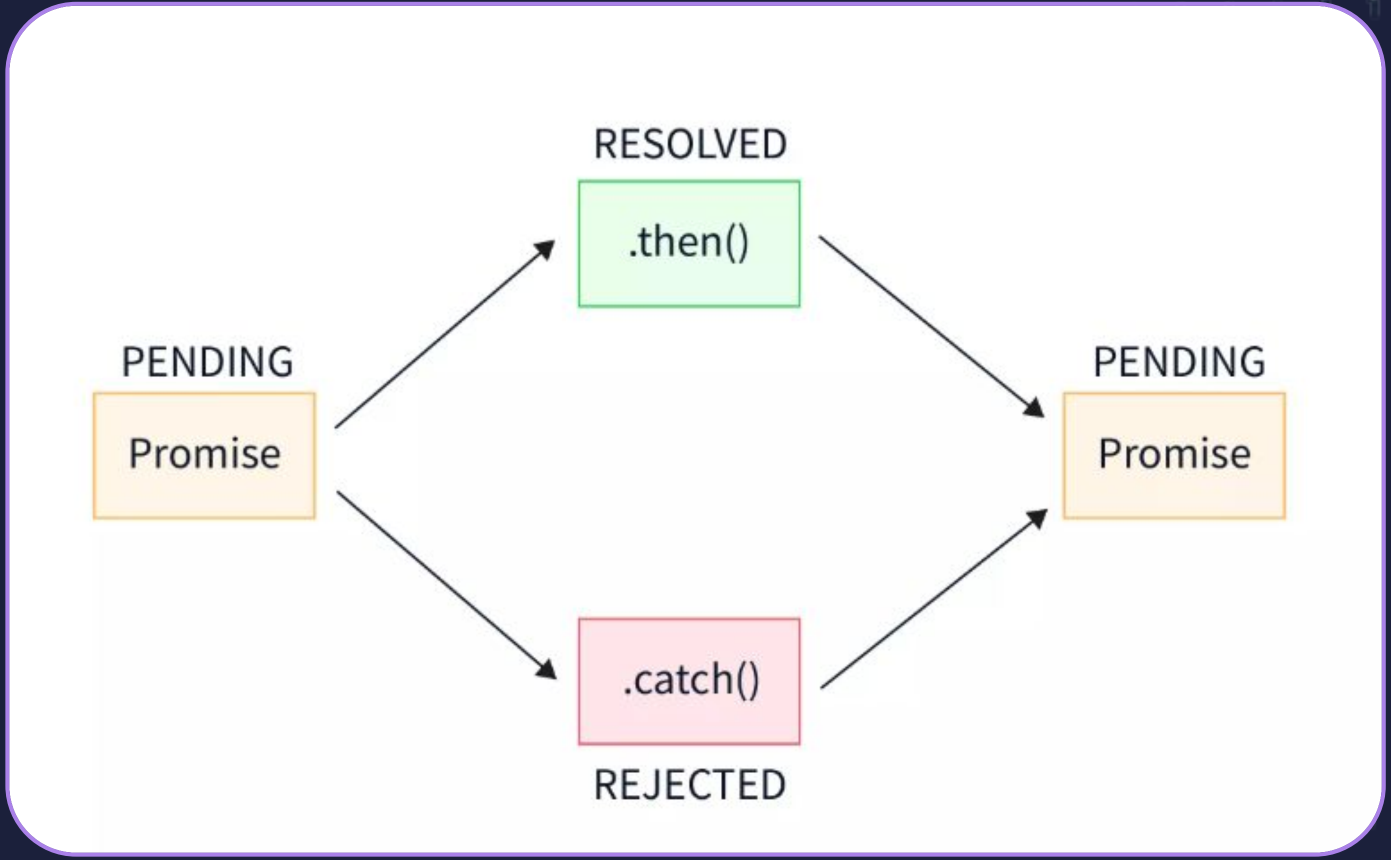
If you take a promise from your friend that he will get you chocolate, then here are the possible conditions.

1. You are waiting for your friend to come. This means the promise is in a PENDING state.
2. Your friend bought you a chocolate. This means the promise is in a FULFILLED state.
3. Due to any reason, your friend failed to get a chocolate, maybe the shop was closed. This means the promise goes to the REJECTED state.

Understanding promises

In the same way, if you are performing any asynchronous task that might get some resources, it is a promise. At first, the promise remains in the PENDING state because no response has been received. After some time, when the response is received, there are two possible cases:

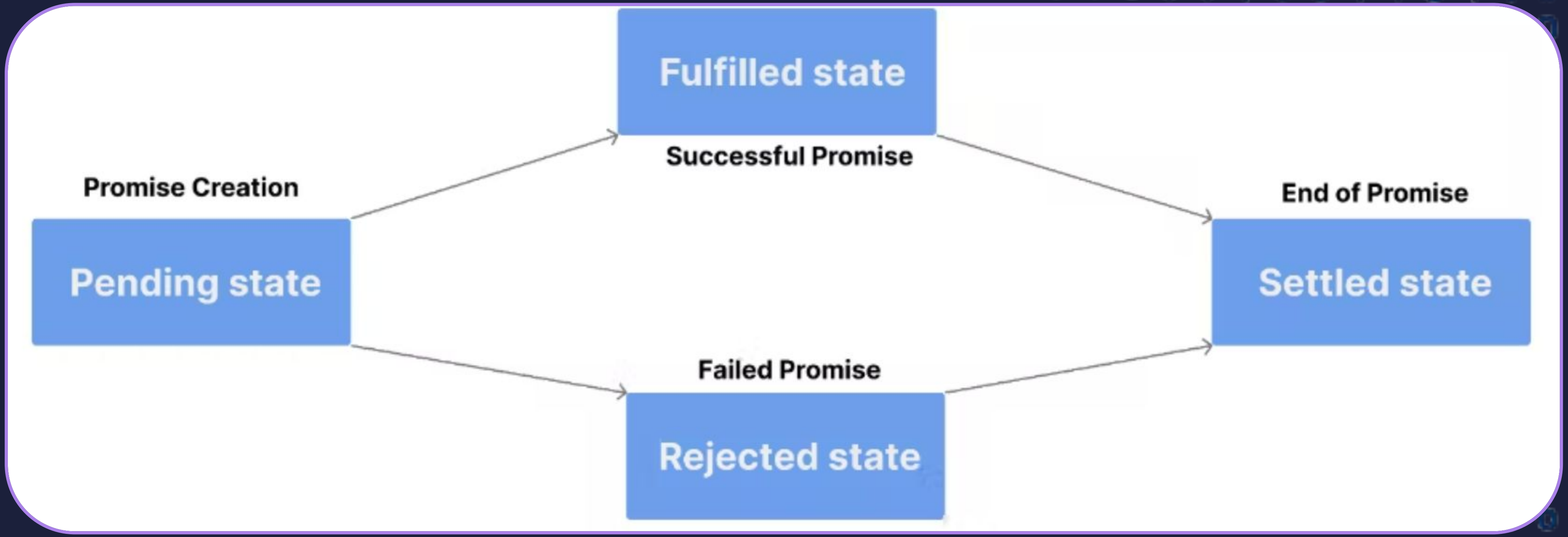
1. We got a response i.e., the promise goes to the FULFILLED state.
2. We got an error and no response the promise got REJECTED.



Promise Lifecycle

The lifecycle of promises consists of 4 stages.

1. Pending.
2. Resolved.
3. Rejected.
4. Settled.



Promise constructor

A promise constructor is used to create a new Promise.

The Promise constructor takes a function as its argument, which in turn takes two parameters, resolve and reject. These parameters are functions that are used to set the state of the Promise.

Consuming the Promise values

Promises will for sure reach one state or the other of the promise lifecycle. Consuming a promise simply means taking the value gotten from the promise (the resolved or rejected value) to process another operation.

We have three methods to consume the promise values.

1. .then().
2. .catch().
3. .finally().

SKILLS

THANK YOU