Promises

→ What is a Promise?

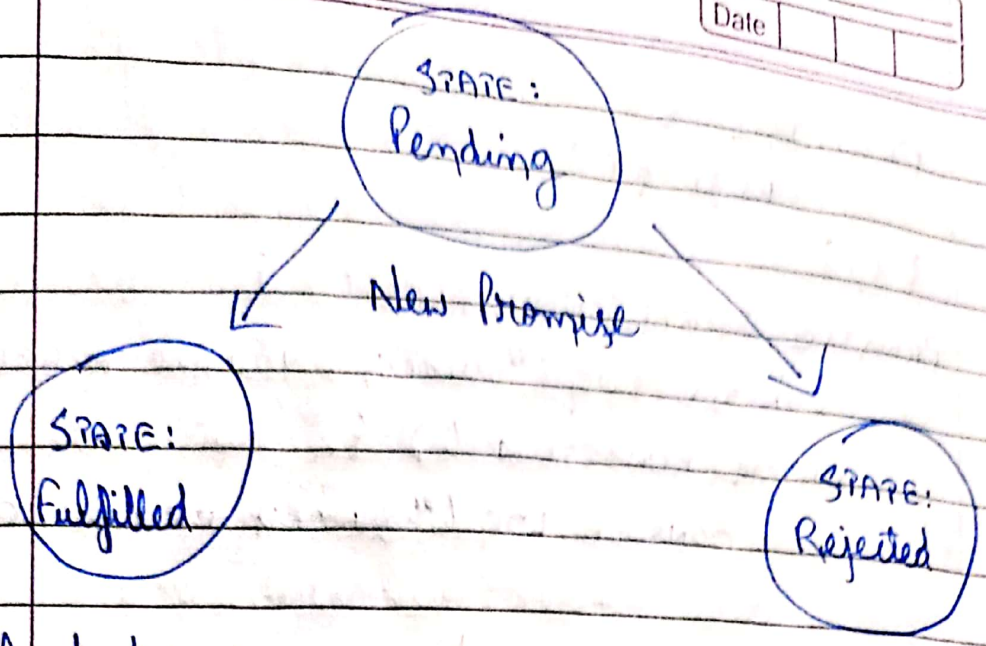
→ A promise in Javascript is similar to a promise we do in real life. When we make a promise, it is a guarantee that in future, we are going to do something. A promise has two possible outcomes: it will be kept when the time comes, or it will not. Similarly, in Javascript, when we define a promise, either it will be resolved when the time comes, or it will get rejected. According to MDN, "the Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value."

→ A promise has three states:

→ Pending: It is the initial state.

→ Fulfilled: It means promise was successful

→ Rejected: It means promise was unsuccessful.



Resolved promise ... Rejected promise

→ The ~~con~~ constructor syntax for a promise object is:

```
LET MYPROMISE = NEW PROMISE(FUNCTION (RESOLVE,
    REJECT) {
// code here
});
```

When ~~the~~ new Promise is created, the function passed into it runs automatically. It contains the producing code which produces the result. Its arguments resolve and reject. Ex:

```
VAR PROMISE = NE PROMISE (FUNCTION (RESOLVE,
  REJECT) {
CONST X = "GEEKS";
CONST Y = "GEEKS";
IF (X === Y) {
    RESOLVE();
}
```

```
} else {
    reject();
}});
promise.then(function() {
    console.log("Success, You are a geek");}).
    catch(function() {
        console.log("Error occurred");});
```

→ A promise is created using a constructor that takes a call back function with two arguments resolve and reject in line 1. If the task is successful. (x === y), the promise is resolved. If the task is unsuccessful (x is not eq. to y), then the promise is rejected. Then then() method is called if the promise is resolved, and the catch() method is called when the promise is rejected or if an error occurred during the code execution. Promises are the ideal choice for asynchronous programming. Promises can handle multiple asynchronous operations easily and are better at error handling as compared to callbacks and events.

→ Benefits of Promises:

1) It improves the code readability.
2) It is better in the handling of asynchronous

operations.

3) It has a better flow of control definition in asynchronous logic.

4) It is better in error handling.