Lecture 23 | `Promises` as a concept to work on Puppeteer JS (Web Automation)

General understanding of promises

> ⚡ Promises in otherwords is a commitment to a future value, which is either a success (fullfil) or a failure (reject) supposedly.

- A step above callback functions
- Promises are a way to handle asynchronous code

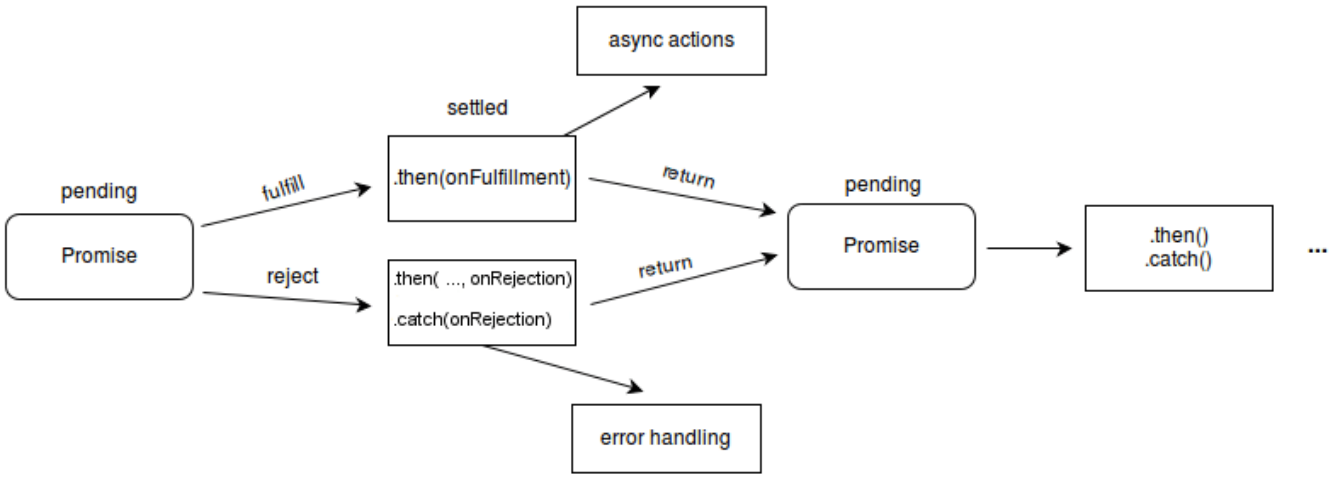
Story Time 🎉

- Lets take a scenario to get Burger from Burger King 🍔 🔥
- Step 1 : First we need to order from Burger King 🍔
- Step 2 : Then we get a token from their end (Pending Stage) 🤔
- Step 3 : Then we either get the burger from Burger King 🍔 (fulfilled) or we get an error/not get the burger 🤔 (rejected) (Settled Stage)

Likewise Promises in JavaScript work no diffrent

- Stages/States in `Promises`

Stages/States	Description
Pending	The initial state of a promise. The promise is pending, and has not yet resolved or rejected.
Fulfilled	The promise has resolved.
Rejected	The promise has rejected.
Settled	The promise has resolved or rejected.



Practical deepdive to Promises and coding practices

- Understanding promises in JavaScript handling async code

```
const fs = require("fs");
const log = console.log;

log("Before");

// fs.readFile("file1.txt", function (err, data) {
//   if (err) {
//     log(err);
//   }
//   log("File 1: " + data.toString());
// });

let promise = fs.promises.readFile("file1.txt"); // handles the callback
function mechanism as default (got the token here ⚡)
// log("Promise: " + promise); // Promise { <pending> }
// The two dedicated methords : then and catch, are used to fulfill the
promise
// in order to fulfill the promise, we need to call the then() method
// in order to handle rejection, we need to call the catch() method

promise
  .then(function (data) {
    log("File 1: " + data.toString()); // handles the fulfillment of the
promise
  })
  .catch(function (err) {
    log(err); // handles the rejection of the promise
  });

log("Promise: " + promise); // promise settled

log("After");
```

> Note : Promises as an optimisation over callback functions, since there is something known as callback hell (callback hell is a nightmare) and it is not easy to debug. 😱

`setTimeout()` : Helps to execute a function after a certain time

- The global `setTimeout()` method sets a timer which executes a function or specified piece of code once the timer expires.

Syntax

```
var timeoutID = setTimeout(function[, delay, arg1, arg2, ...]);
var timeoutID = setTimeout(function[, delay]);
var timeoutID = setTimeout(code[, delay]);
```

> Example Code

```
setTimeout(() => {
  console.log("this is the first message");
}, 5000);
setTimeout(() => {
  console.log("this is the second message");
}, 3000);
setTimeout(() => {
  console.log("this is the third message");
}, 1000);

// Output:

// this is the third message
// this is the second message
// this is the first message
```

Promises READ HERE

> Construction of promise ✅

```
const log = console.log;

// syntax to construct a promise
// let myPromise = new Promise(function (resolve, reject) {});

let promise = new Promise(function (resolve, reject) {
  const x = "Milind";
  const y = "Milind";
  if (x == y) {
    resolve("Same");
  } else {
    reject("Not Same");
  }
});

promise
  .then(function (data) {
    log(data);
  })
  .catch(function (err) {
    log(err);
  });

log(promise);
```

> Example Code : (Reading files)

```
const fs = require("fs");
const { clearScreenDown } = require("readline");
const log = console.log;

let f1p = fs.promises.readFile("f1.txt");
let f2p = fs.promises.readFile("f2.txt");
let f3p = fs.promises.readFile("f3.txt");

function cb(data) {
  log("File data :" + data);
}

f1p.then(cb);
f2p.then(cb);
f3p.then(cb);

// micro task queue : it is a queue of micro tasks that are executed in
the order they are added to the queue.
```

- Visualise Callback queue, Micro task queue, Call Stack, Node APIs and Event Loop HERE 🙌🏻

Making JavaScript to read files serially explicitly using callbacks

```
const fs = require("fs");

console.log("Before");

// read files serially :

fs.readFile("f1.txt", cb1);

function cb1(err, data) {
  if (err) {
    console.log(err);
  } else {
    console.log("File data :" + data);
    fs.readFile("f2.txt", cb2);
  }
}

function cb2(err, data) {
  if (err) {
    console.log(err);
  } else {
    console.log("File data :" + data);
    fs.readFile("f3.txt", cb3);
  }
}

function cb3(err, data) {
```

```
    if (err) {
      console.log(err);
    } else {
      console.log("File data :" + data);
    }
  }

  console.log("After");
```

> Output :

```
> node readingFilesSerailly.js
Before
After
File data :I am f1.txt
File data :I am f2.txt
File data :I am f3.txt
```

- Home work : read files serially using promises

> Hint 😊 : use .then() method to chain the promises individually to read the files