JavaScript Asynchronous programming

Definition
SetTimeout
Ajax
Promises
Async/Await

Asynchronous programming

- Generally programs are executed line by line . So only when line 1 is complete , will execution move to line 2. Thus line 1's execution blocks line 2's execution.

- General programs are synchronous in nature .

- Asynchronous programming means code will be executed at a later "time" or "event". Subsequent code's execution doesn't have to wait .

- Ordering in restaurant with token system is a real-life example of asynchronous execution. It is non-blocking.

- Example: setTimeout function, Ajax requests

Asynchronous programming example - setTimeout

- Functions running in parallel with other functions are called asynchronous

```
Console.log(1);
function myFunction() {
  console.log(2);
}

setTimeout(myFunction, 3000);

console.log(3);
```

- myFunction in above example is a callback function

Definition - Callback

- A callback is a function passed as an argument to another function

- This technique allows a function to call another function

- A callback function can run after another function has finished

Blocking code

- We add a click event listener to a button so that when clicked, it runs a time-consuming operation (calculates 10 million dates then logs the final one to the console) and then adds a paragraph to the DOM: https://mdn.github.io/learning-area/javascript/asynchronous/introducing/simple-sync.html

- https://mdn.github.io/learning-area/javascript/asynchronous/introducing/simple-sync-ui-blocking.html

- We block user interactivity with the rendering of the UI. The first operation blocks the second one until it has finished running.

- In this block, the lines are executed one after the other:

Async programming example - Ajax

- https://www.w3schools.com/js/tryit.asp?filename=tryjs_callback7

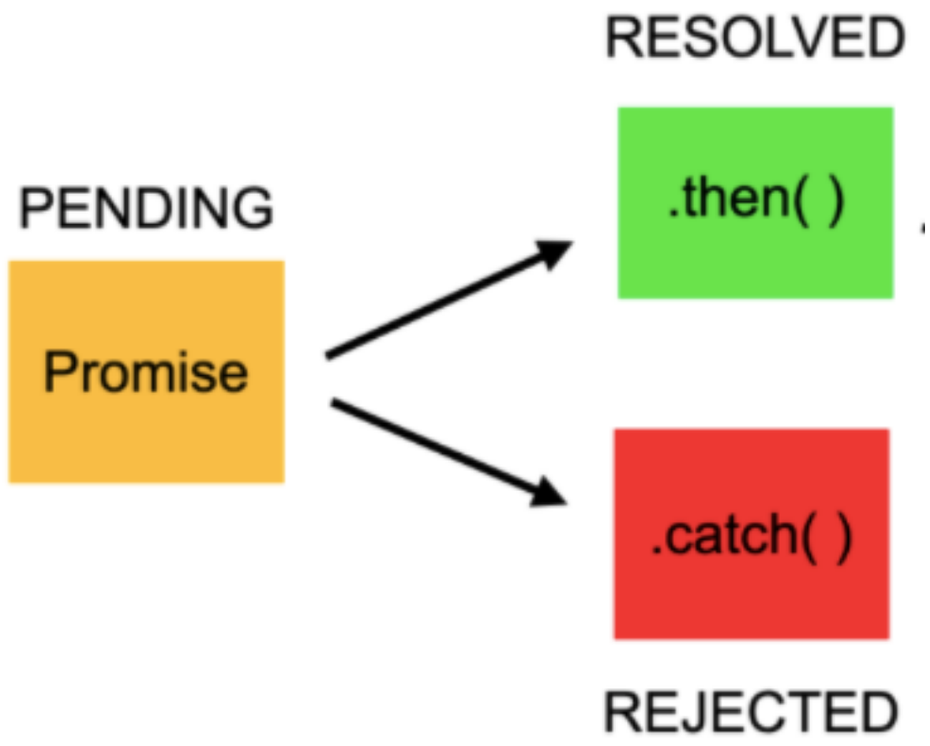
```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function getFile(myCallback) {
  let req = new XMLHttpRequest();
  req.open('GET', "mycar.html");
  req.onload = function() {
    if (req.status == 200) {
      myCallback(this.responseText);
    } else {
      myCallback("Error: " + req.status);
    }
  }
  req.send();
}

getFile(myDisplayer);
```

Promises

- **`Promise`** object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.



Promises

- A Javascript promise can be fulfilled, rejected, or pending
- While a Promise object is "pending" (working), the result is undefined.
- When a Promise object is "fulfilled", the result is a value.
- When a Promise object is "rejected", the result is an error object.
- Example: https://www.w3schools.com/js/tryit.asp?filename=tryjs_promise2

Promises

- Convert callback to promise

```
setTimeout(function() { myFunction(“hello
world!!!"); }, 3000);

function myFunction(value) {
  document.getElementById("demo").innerHTML = value;
}
```

Async/Await

- "async and await make promises easier to write” - syntactic sugar

- **async** makes a function return a Promise

- **await** makes a function wait for a Promise. You don’t need the `.then()` syntax