JS Promises

Abstract :

A promise is commonly defined as a proxy for a value that will eventually become available. Promises are one way to deal with asynchronous code, without getting stuck in callback hell. Promises have been part of the language for years (standardized and introduced in ES2015), and have recently become more integrated, with async and await in ES2017.

How promises work :

Once a promise has been called, it will start in a pending state. This means that the calling function continues executing, while the promise is pending until it resolves, giving the calling function whatever data was being requested.

The created promise will eventually end in a resolved state, or in a rejected state, calling the respective callback functions (passed to then and catch) upon finishing.

Which JS APIs use promises ?

In addition to your own code and libraries code, promises are used by standard modern Web APIs such as:

the Battery API

the Fetch API

Service Workers

Creating a promise :

```
const done = true;

const isItDoneYet = new Promise((resolve, reject) => {
  if (done) {
    const workDone = 'Here is the thing I built';
    resolve(workDone);
  } else {
    const why = 'Still working on something else';
    reject(why);
  }
});
```

Code Description :

As you can see, the promise checks the done global constant, and if that's true, the promise goes to a resolved state (since the resolve callback was called); otherwise, the reject callback is executed, putting the promise in a rejected state. (If none of these functions is called in the execution path, the promise will remain in a pending state)

Using resolve and reject, we can communicate back to the caller what the resulting promise state was, and what to do with it. In the above case we just returned a string, but it could be an object, or null as well. Because we've created the promise in the above snippet, it has already started executing. This is important to understand what's going on in the section Consuming a promise below.

Consuming a promise :

```
const isItDoneYet = new Promise(/* ... as above ... */);

// ...

const checkIfItsDone = () => {

  isItDoneYet

    .then(ok => {

      console.log(ok);

    })

    .catch(err => {

      console.error(err);

    });

};
```

Code Description :

Running checkIfItsDone() will specify functions to execute when the isItDoneYet promise resolves (in the then call) or rejects (in the catch call).