JavaScript is single threaded, meaning that two bits of script cannot run at the same time; they have to run one after another. In browsers, JavaScript shares a thread with other stuff and activity in one of these things delays the others. Events and callbacks get around this but unfortunately it is possible that the events happened before we started listening for them, so we need to work around that using good programming practices.

Promises presents a solution with better naming, simplifying the process. A promise can only succeed or fail once. It cannot succeed or fail twice; neither can it switch from success to failure or vice versa. If a promise has succeeded or failed and you later add a success/failure callback, the correct callback will be called, even though the event took place earlier.

A promise can be:

* fulfilled - The action relating to the promise succeeded
* rejected - The action relating to the promise failed
* pending - Hasn't fulfilled or rejected yet
* settled - Has fulfilled or rejected



The promise constructor takes one argument, a callback with two parameters, resolve and reject. Does something within the callback, perhaps async, and then calls resolve if everything worked, otherwise call reject. Like throw, it is customary but not required to reject with an Error object. The benefit of Error objects is they capture a stack trace, making debugging tools more helpful.



then() takes two arguments, a callback for a success case, and another for the failure case. Both are optional, so you can add a callback for the success or failure case only. JavaScript promises will treat anything with a then() method as promise-like.

AJAX stands for Asynchronous JavaScript And XML. AJ represents that code is executed asynchronously, and XML distributes data over the internet through browsers. The term AJAX is outdated, as we don’t often use XML anymore. The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. The Promise returned from fetch() will not reject on HTTP error status and will only reject on network failure or if anything prevented the request from completing.

* fetch(): fetches resources asynchronously across the network
* clone(): creates a clone of the response
* redirect(): creates a new response but with a different URL
* arrayBuffer(): returns a promise that resolves with an array buffer
* formData(): returns a promise that resolves with a form data object
* blob(): resolves with a blob
* text(): resolves with a string
* json(): resolves the promise with JSON

[.PPW](file:///\\Naeast.ad.jpmorganchase.com\home\amerxbus\XBUS\nacdc1vdihome12\R666050\jpmDesk\Desktop\My%20Stuff\Proficiency;%20JavaScript\Presentation.pptx)

