Unit 2: HTML5, JQuery and Ajax

jquery: Callback

A callback function is executed after the current effect is 100% finished. JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors. To prevent this, you can create a callback function.

Syntax :

$(selector).hide(speed, callback);

Example : callback 1 .html

In JavaScript, statement lines are executed one by one. It might cause problems at times, as a certain effect might start running before the previous one finishes.

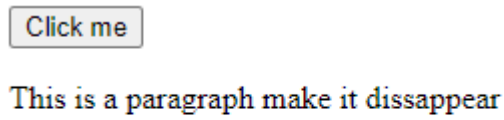
To prevent that, callback function jQuery comes in handy. It creates a queue of effects so they are run in a row.

CODE:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://code.jquery.com/jquery-3.4.1.min.js"
  integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
  crossorigin="anonymous"></script>
<script type="text/javascript" src="scripts.js"></script>
<link rel="stylesheet" href="styles.css">
</head>
<body>

<button>Click me</button>

<p>This is a paragraph make it dissappear</p>

</body>
</html>
```

```
$(document).ready(() => {
    $("button").click(() => {
        $("p").hide("slow", () => {
            alert("Congratulations it works");
        });
    });
});
```

Output:



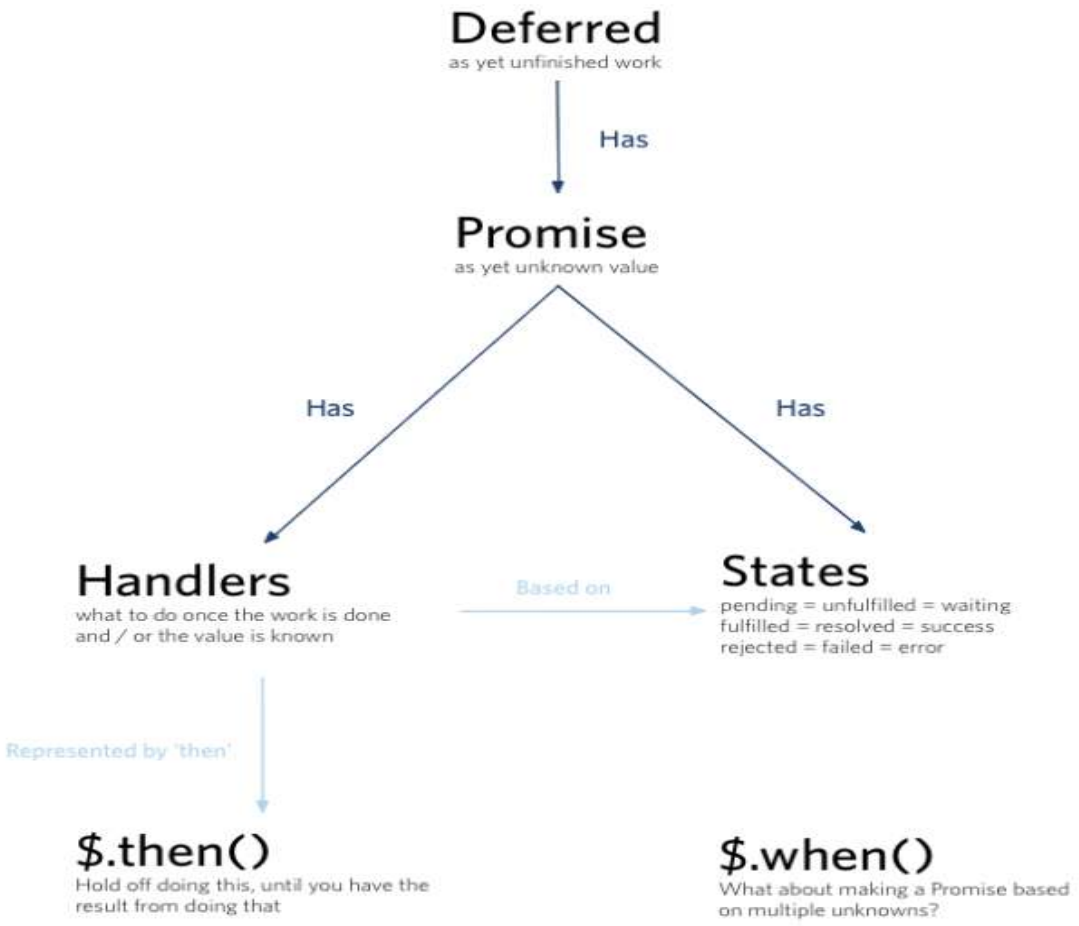
Once you click on Click ME button the texts disappears and alert box pops up:

Callback functions are simple and get the job done, but they become unmanageable as soon as you need to execute many asynchronous operations, either in parallel or in sequence. The situation where you have a lot of nested callbacks, or independent callbacks that have to be synchronized, is often referred to as the "callback hell."

jQuery helps you avoid all these browser issues but, depending on the version of the library you're using, it might do it in a slightly different manner. jQuery provides you with two objects, Deferred and Promise, that you can reliably use

in projects Lets c the concept of promises through the use of two objects: **Deferred** and **Promise**.

- A promise represents a value that is not yet known
- A deferred represents work that is not yet finished



In JavaScript, a promise is a request to resource from another website, the result of a long calculation either from your server or from a JavaScript function, or the response of a REST service (these are examples of promises), and you perform other tasks while you await the result. When the latter becomes available (the promise is resolved/fulfilled) or the request has failed (the

promise is failed/rejected), you act accordingly. Promises have been widely discussed and such discussions have resulted in two proposals:

- Promises/A
- Promises/A+.

A promise represents the eventual result of an asynchronous operation. The primary way of interacting with a promise is through its then method, which registers callbacks to receive either a promise's eventual value or the reason why the promise cannot be fulfilled.

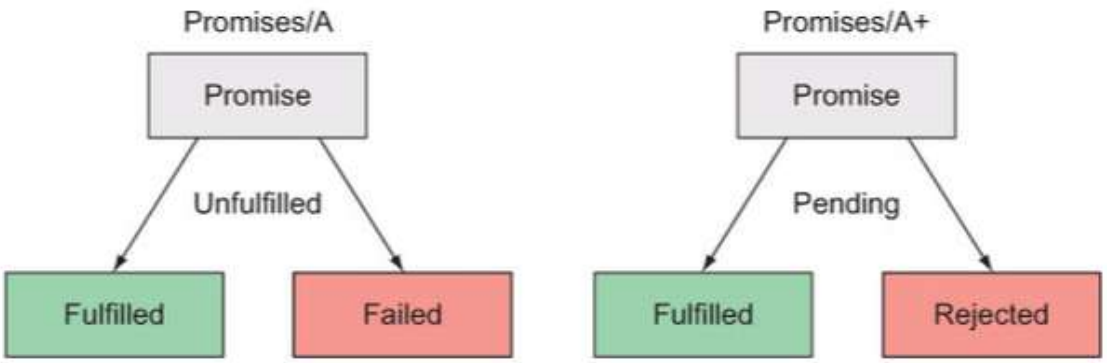
Promises/A+ specifications

The then() method described by the Promises/A+ proposal is the core of promises. The then() method accepts two functions: one to execute in the event that the promise is fulfilled and the other if the promise is rejected. When a promise is in one of these states, regardless of which one, it's settled. If a promise is neither fulfilled nor rejected (for example, if you're still waiting for the response of a server calculation), it's pending.

A promise represents the eventual value returned from the single completion of an operation. A promise may be in one of the three states, unfulfilled, fulfilled, and failed. The promise may only move from unfulfilled to fulfilled, or unfulfilled to failed.

Promises/A specifications

As per below Fig, the terminology for the Promise object's definition is a bit different. The Promises/A proposal defines the unfulfilled, fulfilled, and failed states, whereas the Promises/A+ proposal uses the pending, fulfilled, and rejected states.



These proposals outline the behavior of promises and not the implementation, so libraries implementing promises have a then() method in common but may differ for other methods exposed.

The Deferred and Promise objects

The Deferred object was introduced in jQuery 1.5 as a chainable utility used to register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function. This object can be used for many asynchronous operations, like Ajax requests and animations, but also with JavaScript timing functions. Together with the Promise object, it represents the jQuery implementation of promises.

The Promise object is created starting from a Deferred object or a jQuery object and possesses a subset of the methods of the Deferred object (always(), done(), fail(), state(), and then()). Deferred objects are typically used if you write your own function that deals with asynchronous callbacks. So, function is the producer of the value and you want to prevent users from changing the state of the Deferred.

The Deferred methods

In jQuery, a Deferred object is created by calling the $.Deferred() constructor. The syntax of this function is as follows.

Method syntax: $.Deferred

$.Deferred([beforeStart])

A constructor function that returns a chainable utility object with methods to register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function. It accepts an optional function to execute before the constructor returns.

Parameters

beforeStart (Function) A function that's called before the constructor returns. The function accepts a Deferred object used as the context (this) of the function.

Returns

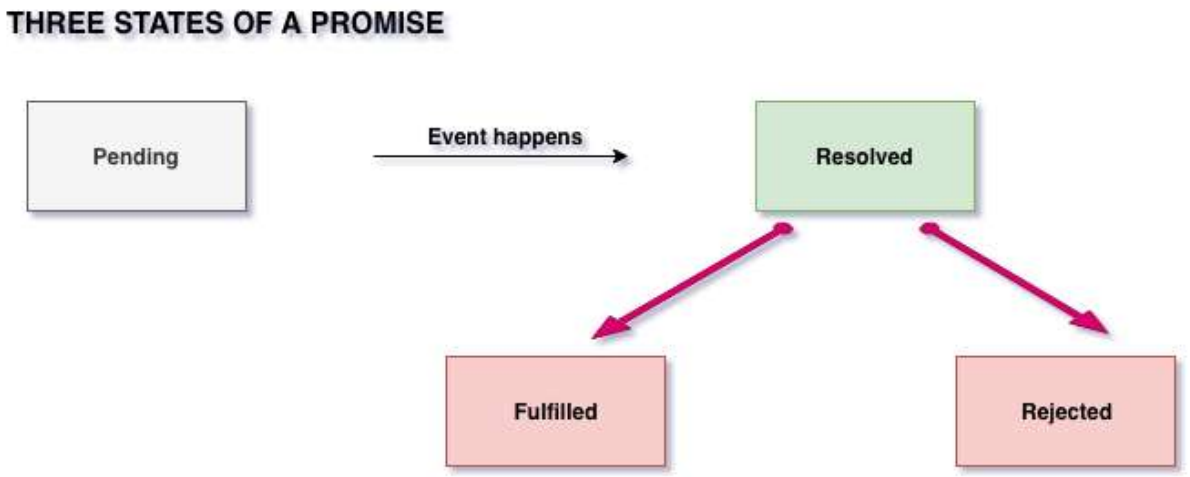
The Deferred object.

Example:

$.Deferred().html('Promises are great!');

In jQuery, a promise can be resolved (the promise is successful), rejected (an error occurred), or pending (the promise is neither resolved nor rejected). These states can be reached in two ways. The first is determined by code that you or another developer has written and with an explicit call to methods like deferred.resolve(),deferred.resolveWith(),deferred.reject(),ordeferred.rejectWith (). These methods, as we'll discuss in a few moments, allow you to either resolve or reject a promise. The second is determined by the success or the failure of a jQuery function—for example, $.ajax()—so you don't have to call any of the previously mentioned methods yourself.

A promise is an object that represents the return value or the thrown exception that the function may eventually provide. In other words, a promise represents a value that is not yet known. A promise is an asynchronous value. The core idea behind promises is that a promise represents the result of an asynchronous operation. A Promise has 3 possible states – Pending – Fulfilled – Rejected.



Example: Simple Functional Transform

```
var author = getAuthors();

var authorName = author.name;

// becomes

var authorPromise = getAuthors().then(function (author) {

return author.name; });
```

Method syntax: promise

promise([type][, target])

Returns a dynamically generated Promise object that's resolved once all actions of a certain type bound to the collection, queued or not, have finished. By default, type is fx, which means the returned Promise is resolved when all animations of the selected elements have completed.

Parameters

type (String) The type of queue that has to be observed. The default value is fx, which represents the default queue for the effects.

target (Object) The object onto which the promise methods have to be attached.

Returns

A Promise object.

Example:

```
$('p')

  .promise()
```

.then(function(value) { console.log(value); });

Deffered Vs Promise

The main difference between callbacks and promises is that with callbacks you tell the executing function what to do when the asynchronous task completes, whereas with promises the executing function returns a special object to you (the promise) and then you tell the promise what to do when the asynchronous task completes.

Deffered	Promise
You cannot use $.Promise();	new $.Deferred(); OR $.Deferred()
A **deferred** object is an object than can create a promise and change its state to resolved or rejected. Deferreds are typically used if you write your own function and want to provide a promise to the calling code. You are the producer of the value.	A **promise** is, as the name says, a promise about a future value. You can attach callbacks to it to get that value. The promise was "given" to you and you are the **receiver** of the future value. You cannot modify the state of the promise. Only the code that *created* the promise can change its state.
You can call use resolve or reject etc.	The Promise exposes only the Deferred methods needed to attach additional handlers or determine the state (then, done, fail, always, pipe, progress, and state), but not ones that change the state (resolve, reject, notify, resolveWith, rejectWith, and notifyWith).