**JavaScript Promises1**

[**https://www.w3schools.com/js/js\_promise.asp**](https://www.w3schools.com/js/js_promise.asp)

**https://www.geeksforgeeks.org/javascript-promises/**

*"I Promise a Result!"*

"Producing code" is code that can take some time

"Consuming code" is code that must wait for the result

A Promise is a JavaScript object that links producing code and consuming code

**JavaScript Promise Object**

A JavaScript Promise object contains both the producing code and calls to the consuming code:

**Promise Syntax**

let myPromise = new Promise(function(myResolve, myReject) {  
// "Producing Code" (May take some time)  
  
  myResolve(); // when successful  
  myReject();  // when error  
});  
  
// "Consuming Code" (Must wait for a fulfilled Promise)  
myPromise.then(  
  function(value) { /\* code if successful \*/ },  
  function(error) { /\* code if some error \*/ }  
);

When the producing code obtains the result, it should call one of the two callbacks:

|  |  |
| --- | --- |
| **Result** | **Call** |
| Success | myResolve(result value) |
| Error | myReject(error object) |

**Promise Object Properties**

A JavaScript Promise object can be:

* Pending
* Fulfilled
* Rejected

The Promise object supports two properties: **state** and **result**.

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.

|  |  |
| --- | --- |
| **myPromise.state** | **myPromise.result** |
| "pending" | undefined |
| "fulfilled" | a result value |
| "rejected" | an error object |

You cannot access the Promise properties **state** and **result**.

You must use a Promise method to handle promises.

**Promise How To**

Here is how to use a Promise:

myPromise.then(  
  function(value) { /\* code if successful \*/ },  
  function(error) { /\* code if some error \*/ }  
);

Promise.then() takes two arguments, a callback for success and another for failure.

Both are optional, so you can add a callback for success or failure only.

**Example**

function myDisplayer(some) {  
  document.getElementById("demo").innerHTML = some;  
}  
  
let myPromise = new Promise(function(myResolve, myReject) {  
  let x = 0;  
  
// The producing code (this may take some time)  
  
  if (x == 0) {  
    myResolve("OK");  
  } else {  
    myReject("Error");  
  }  
});  
  
myPromise.then(  
  function(value) {myDisplayer(value);},  
  function(error) {myDisplayer(error);}  
);

ADVERTISEMENT

**JavaScript Promise Examples**

To demonstrate the use of promises, we will use the callback examples from the previous chapter:

* Waiting for a Timeout
* Waiting for a File

**Waiting for a Timeout**

**Example Using Callback**

setTimeout(function() { myFunction("I love You !!!"); }, 3000);  
  
function myFunction(value) {  
  document.getElementById("demo").innerHTML = value;  
}

**Example Using Promise**

let myPromise = new Promise(function(myResolve, myReject) {  
  setTimeout(function() { myResolve("I love You !!"); }, 3000);  
});  
  
myPromise.then(function(value) {  
  document.getElementById("demo").innerHTML = value;  
});

**Waiting for a file**

**Example using Callback**

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

**Example using Promise**

let myPromise = new Promise(function(myResolve, myReject) {  
  let req = new XMLHttpRequest();  
  req.open('GET', "mycar.htm");  
  req.onload = function() {  
    if (req.status == 200) {  
      myResolve(req.response);  
    } else {  
      myReject("File not Found");  
    }  
  };  
  req.send();  
});  
  
myPromise.then(  
  function(value) {myDisplayer(value);},  
  function(error) {myDisplayer(error);}