* A Promise object represents a value that may not be available yet, but will be resolved at some point in the future.
* A promise represents the eventual result of an asynchronous operation. It is a placeholder into which the successful result value or reason for failure will materialize.
* It allows us to write asynchronous code in a more synchronous fashion.
* For example, if you use the promise API to make an asynchronous call to a remote web service you will create a Promise object which represents the data that will be returned by the web service in future.
* The concept here is the actual data is not available yet. It will become available when the request completes and a response comes back from the web service. In the meantime the Promise object acts like a proxy to the actual data. Furthermore, you can attach callbacks to the Promise object which will be called once the actual data is available.

var promise = new Promise(function(resolve, reject) {

//asynchronous code goes here

});

* We start by instantiating a new Promise object and passing it a callback function. The callback takes two arguments, resolve and reject, which are both functions. All your asynchronous code goes inside that callback. If everything is successful, the promise is fulfilled by calling resolve(). In case of an error, reject() is called with an Error object. This indicates that the promise is rejected.
* When we instantiate a Promise object we get a proxy to the data that will be available in future

A promise can be in either of below 3 states:

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

**Chaining**

* It is sometimes desirable to chain promises together. For instance, you might have multiple asynchronous operations to be performed. When one operation gives you data, you will start doing some other operation on that piece of data and so on. Promises can be chained together as demonstrated in the following example.

function getPromise(url) {

// return a Promise here

// send an async request to the url as a part of promise

// after getting the result, resolve the promise with it

}

var promise = getPromise('some url here');

promise.then(function(result) {

//we have our result here

return getPromise(result); //return a promise here again

}).then(function(result) {

//handle the final result

});

* The tricky part is that when you return a simple value inside then(), the next then()is called with that return value. But if you return a promise inside then(), the next then() waits on it and gets called when that promise is settled.