```
Springboard
                                             JavaScript Promises
                                                                                                                              🎇 Springboard
     JavaScript Promises
                                             Download Demo Code
         « Back to Homepage
                                             Goals
Goals
                                              • Review the callback pattern for asynchronous code in JavaScript
 Goals
                                              • Define what a promise is
                                              • Use promises to manage asynchronous code
The Callback Pattern: A Review
 Asynchronous JavaScript
                                              • Compare and contrast promises with the callback pattern
 Async + AJAX
                                              • Explore the Promise function in detail
 Takeaways
Promises
                                             The Callback Pattern: A Review
 What's a Promise?
 A Mental Model for Promises
                                             Asynchronous JavaScript
Working with Promises
 Our First Promise
                                              JavaScript runs synchronously, but there's a way to handle async code: asynchronous callbacks.
 What exactly is a Promise?
  .then and .catch
                                              console.log("this prints first");
 More Promise Examples
 Promise Chaining
                                              setTimeout(function() {
                                                 console.log("this prints third, one second later");
 Promise Chaining Example
                                              }, 1000);
 Takeaways
Why do we have Promises?
                                              console.log("this prints second");
 Callback Hell
                                             This code does not run "out of order." However, the callback to setTimeout is not executed right away — it runs
 Pokemon Callback Hell
                                             after the timer expires.
 Pokemon with Promises
The built-in Promise function
                                             Async + AJAX
 More on Promise
                                             When working with timers, you'll need to manage asynchronous code.
 Promise.all
 Promise.all Example
                                             A common scenario for managing asynchronous code is dealing with AJAX.
 Promise.race
 Promise.race Example
                                              Let's try to pull some data with jQuery, which utilizes callbacks (we're just using this as an example since axios
                                             doesn't support callbacks):
 Promise.resolve
  Promise.reject
  Building our own Promises
                                              let planet;
 Making Promises: An Example
                                              $.getJSON("https://swapi.dev/api/planets/1/", response => {
 Asynchronous Function Pattern
                                                 planet = response;
 Takeaways
                                              });
                                              console.log(planet);
                                              • What is the value of planet?
                                              • Why is it undefined?!
                                              • The console.log was synchronous; it ran before the asynchronous callback.
                                             Let's fix it!
                                              let planet;
                                              $.getJSON.get("https://swapi.dev/api/planets/1/", response => {
                                                 planet = response;
                                                 console.log("done", planet);
                                              console.log("waiting");
                                              • Asynchronous callbacks run after the rest of the code
                                              • Once you are inside the callback, the code executes predictably as per usual, (unless there are more async
                                                callbacks in there)
                                             Takeaways
                                              • JS code is executed synchronously (in-order).
                                              • JS can use special asynchronous callbacks to delay execution of code.
                                              • Not all callbacks are async; you'll have to consult their docs to tell.
                                               Note: More on callbacks
                                               If you'd like to read more about JS callbacks: MDN Callback Function
                                             Promises
                                             What's a Promise?
                                             A promise is one-time guarantee of future value.
                                             A Mental Model for Promises
                                             Working with Promises
                                             Our First Promise
                                             demo/app-axios.js
                                              let url = "https://swapi.dev/api/planets/1/"
                                              let ourFirstPromise = axios.get(url);
                                              console.log(ourFirstPromise);
                                               // Promise {<pending>}
                                              • axios is a promise based library for making HTTP requests.
                                              • syntactically, it's similar to the jQuery AJAX methods, but doesn't use the callback pattern.
                                              What exactly is a Promise?
                                              • Promises in JavaScript are objects
                                              • They are native to the language as of ES2015
                                              • A promise can be in one of three states:
                                                 • Pending - It doesn't yet have a value
                                                 • Resolved - It has successfully obtained a value
                                                 • Rejected - It failed to obtain a value for some reason
                                              • The only way to access the resolved or rejected value is to chain a method on the end of the promise.
                                              .then and .catch
                                              • Promises provide a .then and a .catch, which both accept callbacks.
                                              • The callback to .then will run if the promise is resolved, and has access to the promise's resolved value.
                                              • The callback to .catch will run if the promise is rejected, and typically has access to some reason behind the
                                                rejection.
                                                Note: Thenables
                                               When reading about promises, you'll often see a related term, called a thenable. A thenable is simply any
                                               object or function that has a then method defined on it.
                                               By this definition, all promises are thenables, but not all thenables are promises! There are many more
                                               specifications that a promise needs to satisfy.
                                               Here's a simple example of a thenable that isn't a promise:
                                                 let notAPromise = {
                                                   fruit: "apple",
                                                  veggie: "carrot",
                                                  then: () => {
                                                     console.log("I'm just a random object with a then method.");
                                                 };
                                                notAPromise.then();
                                                // "I'm just a random object with a then method."
                                             More Promise Examples
                                             demo/app-axios.js
                                              let validURL = "https://swapi.dev/api/people/1/";
                                              let futureResolvedPromise = axios.get(validURL);
                                               futureResolvedPromise
                                                 .then(data => console.log(data))
                                                 .catch(err => console.log(err));
                                             demo/app-axios.js
                                              let invalidURL = "https://swapi.dev/api/tacos/1/";
                                              let futureRejectedPromise = axios.get(invalidURL);
                                              futureRejectedPromise
                                                 .then(data => console.log(data))
                                                 .catch(err => console.log(err));
                                             Promise Chaining
                                              • When you call .then on a promise, you can return new promise in the callback!
                                              • This means you can chain multiple asynchronous operations together with several .then calls.
                                              • When using this pattern, you only need one .catch at the end. You don't have to catch every promise
                                                individually.
                                             Promise Chaining Example
                                             demo/app-axios.js
                                              let baseURL = "https://pokeapi.co/api/v2/pokemon";
                                               axios
                                                 .get(`${baseURL}/1/`)
                                                 .then(p1 => {
                                                   console.log(`The first pokemon is ${p1.data.name}`);
                                                   return axios.get(`${baseURL}/2/`);
                                                 .then(p2 => {
                                                   console.log(`The second pokemon is ${p2.data.name}`);
                                                   return axios.get(`${baseURL}/3/`);
                                                 .then(p3 => {
                                                   console.log(`The third pokemon is ${p3.data.name}`);
                                                 .catch(err => {
                                                   console.log(`Oops, there was a problem :( ${err}`);
                                             Takeaways
                                              • A promise represents a pending value (a guarantee that there will either be a resolved or rejected value)
                                              • Standard promises all have a .then() method, which takes a callback of the resolved value, and this can be
                                                chained.
                                              • Standard promises all also have a .catch() method, which takes a callback of the rejected value, and this
                                                can only be listed once at the end of a .then chain.
                                              • Axios and jQuerys's AJAX methods both use promises.
                                             Why do we have Promises?
                                             Callback Hell
                                                  Pokemon Callback Hell
                                             demo/app-jquery.js
                                              let baseURL = "https://pokeapi.co/api/v2/pokemon";
                                              $.ajax(`${baseURL}/1/`, {
                                                 success: p1 => {
                                                   console.log(`The first pokemon is ${p1.name}`);
                                                   $.ajax(`${baseURL}/2/`, {
                                                     success: p2 => {
                                                       console.log(`The second pokemon is ${p2.name}`);
                                                       $.ajax(`${baseURL}/3/`, {
                                                         success: p3 => {
                                                            console.log(`The third pokemon is ${p3.name}`);
                                                         error: err => console.log(err)
                                                       });
                                                     error: err => console.log(err)
                                                  });
                                                 error: err => console.log(err)
                                              });
                                             Pokemon with Promises
                                             demo/app-axios.js
                                              // promise chaining with pokemon api
                                              let baseURL = "https://pokeapi.co/api/v2/pokemon";
                                               axios
                                                 .get(`${baseURL}/1/`)
                                                 .then(p1 => {
                                                   console.log(`The first pokemon is ${p1.data.name}`);
                                                   return axios.get(`${baseURL}/2/`);
                                                 .then(p2 => {
                                                   console.log(`The second pokemon is ${p2.data.name}`);
                                                   return axios.get(`${baseURL}/3/`);
                                                 .then(p3 => {
                                                   console.log(`The third pokemon is ${p3.data.name}`);
                                                 .catch(err => {
                                                   console.log(`Oops, there was a problem :( ${err}`);
                                                 });
                                             The built-in Promise function
                                             More on Promise
                                              • Promise was added as a global variable in ES2015
                                              • You can create your own promises using Promise
                                              • There are also several helper methods that live on Promise, including:

    Promise.all

    Promise.race

    Promise.resolve

    Promise.reject

                                                Note: Not all Promise methods are created equal
                                               In general, you'll typically find that Promise.all is by far the most useful method on the Promise function.
                                               There are definitely use-cases for Promise.race, Promise.resolve, and Promise.reject, but they are more rare.
                                               For now, you should focus your attention on getting comfortable with Promise.all, as we won't really
                                               encounter these other methods until we've gotten farther in the Node curriculum.
                                             Promise.all
                                              • Promise.all accepts an array of promises and returns a new promise
                                              • This new promise will resolve when every promise in the array resolves, and will be rejected if any promise in
                                                the array is rejected
                                              • Promise.all is extremely useful whenever you want to send out several independent requests in parallel.
                                             Promise.all Example
                                             demo/app-axios.js
                                              let fourPokemonPromises = [];
                                              for (let i = 1; i < 5; i++) {
                                                fourPokemonPromises.push(
                                                   axios.get(`https://pokeapi.co/api/v2/pokemon/${i}/`)
                                              Promise.all(fourPokemonPromises)
                                                 .then(pokemonArr => (
                                                   pokemonArr.forEach(p => console.log(p.name))
                                                 .catch(err => console.log(err));
                                             Promise.race
                                              • Promise.race accepts an array of promises and returns a new promise
                                              • This new promise will resolve or reject as soon as one promise in the array resolves or rejects
                                             Promise.race Example
                                             demo/app-axios.js
                                              let fourPokemonRace = [];
                                              for (let i = 1; i < 5; i++) {
                                                fourPokemonRace.push(
                                                   axios.get(`https://pokeapi.co/api/v2/pokemon/${i}/`)
                                              Promise.race(fourPokemonRace)
                                                 .then(pokemon => console.log(`${pokemon.name} won!`))
                                                 .catch(err => console.log(err));
                                             Promise.resolve
                                             Promise.resolve accepts a value and returns a promise which has immediately resolved to the value passed in.
                                             demo/promises.js
                                              let resolvedValue = "hello!";
                                              let p1 = Promise.resolve(resolvedValue);
                                              p1; // Promise {<resolved>: "hello!"}
                                              p1 === value; // false
                                             Promise.reject
                                             Promise.reject accepts a value and returns a promise which has immediately rejected to the value passed in.
                                             demo/promises.js
                                              let rejectedValue = "sorry :(";
                                              let p2 = Promise.reject(rejectedValue);
                                              p2; // Promise {<rejected>: "sorry :("}
                                              p2 === value; // false
                                              Building our own Promises
                                              • You can use Promise with the new keyword to make your own promises
                                              • Unfortunately, the syntax here takes some getting used to
                                              • Promise accepts a single function (call it fn) as an argument
                                                 • fn accepts two functions as arguments, resolve and reject
                                                 • Pass resolve a value for the promise to resolve to that value
                                                 • Pass reject a value for the promise to reject to that value
                                             Making Promises: An Example
                                             demo/promises.js
                                              let mockAjaxRequest = new Promise(function(resolve, reject) {
                                                 let probSuccess = 0.5;
                                                let requestTime = 1000;
                                                 // We mock a network request using a setTimeout.
                                                // The request takes requestTime milliseconds.
                                                 // Afterwords, the promise is either resolved with data
                                                 // or rejected with a timeout message,
                                                 // based on whether randomNum is less than probSuccess.
                                                 setTimeout(function() {
                                                   let randomNum = Math.random();
                                                   if (randomNum < probSuccess) {</pre>
                                                     let data = "here's your data!";
                                                     resolve(data);
                                                   } else {
                                                     reject("Sorry, your request failed.");
                                                }, requestTime);
                                              });
                                               mockAjaxRequest
                                                 .then(data => console.log(data))
                                                 .catch(err => console.log(err));
                                             Asynchronous Function Pattern
                                             demo/promises.js
                                               function myAsyncFunction() {
                                                 // return a new Promise
                                                 return new Promise((resolve, reject) => {
                                                     DO ASYNC STUFF HERE
                                                   // if it succeeds, call the resolve callback
                                                   resolve(/* success value*/);
                                                   // if it fails, call the reject callback
                                                   reject(/* fail value*/);
                                                });
```

Takeaways

• A promise is a one time guarantee of future value

• There are methods on **Promise** that can be used to help work with promises

• You can make your own promises using *Promise* with *new* keyword, but the syntax isn't intuitive and takes

• **Promise** is part of JavaScript as of ES2015

practice to get used to