Otro concepto nuclear que está en el “core”, en el núcleo, en el corazón de Javascript y hay que entenderlo para trabajar bien con Javascript, es la programación asincrónica. La posibilidad de trabajar con una secuencia de código que no se detiene a esperar un resultado, así trabaja Javascript.
Otros lenguajes, por ejemplo, pueden decir, “okey, voy a buscar datos a un sitio” o, mejor dicho, el programa podría decir la instrucción “ve a buscar los datos a un sitio”, a una base de datos, eso no es instantáneo como leer de una, de un "array", porque esa información viene de fuera, viene de un servidor. Ustedes dirán “bueno, pero es muy rápido”, sí, probablemente sea muy rápido tal vez, a veces, va a durar menos de un segundo, es imperceptible, pero no es automático.
Hay lenguajes de programación que cuando decimos “ve a buscar información a un sitio”, se quedan esperando la respuesta, pero Javascript no se queda esperando la respuesta, Javascript nunca nos espera, por lo tanto, si nosotros decimos “ve a buscar la respuesta” y muestra la respuesta, como no se va a quedar esperando, la respuesta que nos muestre será incorrecta, porque en realidad, los datos Javascript todavía no los tienen, los está buscando, pero el código se sigue ejecutando y ustedes dirán “bueno, end game, fin del juego, no sirve”. No, obviamente que esto no es así. Hay maneras de decirle a Javascript “ve a buscar los datos y muéstrame dos cuando los tengas” y luego, si el programa sigue corriendo con otras cosas, con otras cuestiones, básicamente esto explicado, tal vez, de una manera muy primaria, pero es nuestra primera aproximación, así que permítanme la simplificación. De esto se trata de la programación asincrónica.

Para manejar el código asincrónico, Javascript utiliza “"promesa"s”. Supongan que tengo una función que dice “voy a buscar los datos” y luego muestra menos, bueno, Javascript la ejecuta y esa función no devuelve los datos, devuelve una "promesa" que dice “te daré los datos o te mostraré un error, pero tranquilo, porque cuando esto esté resuelto yo te voy a avisar” y, efectivamente, así es como funciona. ¿Cuesta entenderlo un poco?, sí. Si yo fuera un novato y estuviese escuchando esta explicación, necesitaría ejemplos. Acá tienen un ejemplo, creo que bien detallado.

Vamos a entender el concepto de "promesas", para luego ir a buscar información como haríamos en la vida real, con una “API” que se llama “fetch”.
Pero primero, para que entiendan cómo funciona "fetch" hagamos nuestro propio "fetch", vamos a construir uno para que ustedes entiendan cómo funcionan las "promesas", cómo trata Javascript el código asincrónico.
Seguimos teniendo aquí nuestro JSON con empleados, simulando que ésta es una base de datos o una API externa que nos entrega esta información. Sería ideal si pudiésemos construir una función “get data”, está, por ejemplo, que recibiese un recurso y luego, simplemente, bueno, lo convirtiera a formato Javascript válido, para finalmente retornarlo. Podríamos hacer aquí, por ejemplo, “const data, igual, JSON parse de recurso”, esto entrará por parámetro, ¿verdad? Es la manera en la que le diremos a esta función “escúchame, ve a buscar la información a tal sitio, ¿bien? y luego de hacer esto podría devolver, mostrar esa información”. Muy bien.
Y ahora podríamos llamar a la función "get data" e indicarle, permítanme jugar juntos, vamos a suponer que JSON, como hacíamos hasta ahora es, bueno, pues, la dirección externa

donde tiene que ir a buscar el recurso. Así que, digamosle, “trae los datos de JSON”, que suponemos que es una dirección, ¿verdad?, fuera de nuestra, de nuestro sistema, en algún servidor, en algún sitio del mundo, ¡boom!, listo, ejecuta la función, va a dónde le decimos y nos trae un arreglo con los empleados. Resuelto. Bueno, pues, tengo malas noticias, en la vida real esto no funciona así, porque esta función ejecuta un procedimiento asíncrono, es decir, este JSON, que en realidad no debería estar aquí, esto es simplemente para que aprendamos, está en un servidor, en otro sitio, por lo tanto, la respuesta no va a ser inmediata, puede demorar, puede demorar milisegundos o, inclusive, segundos. Entonces vamos a simular cómo es esto en la vida real, indicando aquí, por ejemplo, una, un "settimeout", es decir, ¿recuerdan ustedes se "settimeout"?, es para hacer un "delay", para poner una demora, programática en este caso, en la ejecución de este código. Supongamos que podría demorar un segundo, mil milisegundos, es lo que tarda en traer la información y convertirla, ¿okey? Vamos a ver qué ocurre ahora cuando la función intenta mostrarnos la "data", el recurso "data".

Bien, aquí tenemos un problema, en el sentido de que primero vamos a olvidado crear este recurso. Esto vamos a crearlo por aquí, vamos a crear una variable que se llame “data” que no contenga nada, no importa. Ahora sí, bien. Noten que dice "undefined". La línea 11, la línea 11 es el “console log”, pero ¿por qué es "undefined" si aquí tengo la información data y la estoy convirtiendo al formato de objeto Javascript?, ya que entro en formato JSON, bueno, esta, este procedimiento demora un segundo, mil milisegundos, pero Javascript no se queda aguardando el resultado, no se detiene. Ejecuta esta línea y continúa de inmediato con la siguiente, entonces este “console log” de “data”, aún no tiene este resultado es, simplemente, bueno, pues, una variable que no está inicializada y recuerden que Javascript inicializa las variables como "undefined", si es que nosotros no definimos un valor inicial. Por lo tanto, vemos que si no manejamos el código asincrónico, nunca podemos ir a buscar información JSON a ningún sitio, no vamos a poder obtenerla, ¿okey? Para esto, utilizaremos en la próxima clase la API "fetch", pero, ¿cómo trabaja "fetch"?, "fetch" es un procedimiento asincrónico, lo vamos a simular haciendo que esta función que hemos creado y que hasta ahora nos retornaba los datos, pero nos estaba retornando de una manera en la que no nos servía porque, efectivamente, no podíamos acceder a ellos, retorne una "promesa". Entonces, ahora, esta función "get data" va a retornar una nueva, utilizamos el operador “NEW” y con mayúsculas, porque es el nombre de una clase. Decimos "promise", entonces, ahora nuestra función, nunca nos va a retornar "undefined", nuestra función se va a ejecutar de manera tal que, cuando la ejecutemos, no nos va a dar los datos, nos va a decir “okey, esperá mi respuesta, porque yo te voy a dar los datos y si no los encuentro te voy a informar que no los encontré o que hubo un error, pero vas a obtener siempre una respuesta de mi parte”. Así es, groso modo, cómo funcionan las "promesas".
Las "promesas" reciben siempre un “callback” con dos, dos objetos. El objeto "resolve" y "reject", esto es convencional, ustedes pueden usar otros nombres, yo les recomiendo que usen estos pero, reitero, es convencional. Muy bien y aquí dentro vamos a poner la operación que hace nuestra función, es decir, va a buscar los datos, ¿bien?, va a buscar los datos algún sitio. Aquí podríamos poner la función la, inicializar la variable o declararla, mejor dicho. Entonces nuestra función ya no retorna a los datos, sino retorna una "promesa": “voy a hacer esto, voy a intentar hacerlo y te voy a informar”.
Muy bien. Ahora aquí dentro podríamos inclusive preguntar si hubo un error y vamos a inicializar aquí arriba, sólo para comprobar una variable que se llame “error” y que va a estar

en “false” inicialmente. Es decir, asumimos que no hay ningún error, pero vamos a preguntar porque podría haber uno. Entonces, vamos a preguntar si hay un error y si hay un error en el procedimiento de la "promesa" vamos a rechazarla, vamos a indicar aquí, okey “reject” y podríamos indicar por qué la rechazamos. Bueno, vamos a rechazarla por el error. Simplemente vamos a indicar “error”, bien, que va a tener en este caso el valor de “true”, si es que hubiera un error. Por ahora no hay un error. También aquí podríamos indicar hubo un error etcétera no importa eso es secundario.
Ahora si no hubo un error, “else”. Bien vamos a retornar, vamos a resolver la "promesa" vamos a indicar aquí “resolve” y, ¿cómo vamos a resolverla, qué es lo que vamos a devolver? Bueno, precisamente esto vamos a resolverla con JSON parse. Bien, para ser más claros, más específicos, podríamos escribir un poco más, ya que estamos en una etapa de aprendizaje, no está mal hacerlo, porque tal vez así es más explícito. O sea si no hay un error, bien, tomamos el recurso, lo convertimos al formato Javascript, lo ponemos en un objeto “data” y una variable y lo retornamos. Bien, perfecto. Ya tenemos nuestra función que retorna ahora una "promesa". Podríamos manejar perfectamente

nuestro pedido asincrónico.
Ahora deberíamos ejecutar esta "promesa", de una manera levemente distinta a como estábamos haciéndolo, ya que es una un procedimiento asincrónica ¿verdad? Bueno, podríamos indicar aquí, bien, que vaya a buscar a este sitio nuestros datos y luego, al ser un método asincrónico que retorna una "promesa", tenemos que indicar punto “then”, podríamos escribirlo aquí de corrido ¿verdad? Por lo general lo escribimos abajo, para que sea más legible, pero es parte de la misma instrucción, okey. Entonces, ¿me vas a traer la información?, perfecto, entonces esa información que me has traído, que ya está, ya está convertida, ya viene parceada, simplemente mostrarme la API por la consola, son los "data". Bien, digo que la información viene parceada, porque lo incluimos en este método, si no lo hubiéramos hecho, deberíamos hacerlo aquí dentro. Bien, vamos a verlo por ahora así y luego podríamos concatenar el método “catch”, para el caso de que hubiera un error y si lo hubiera, noten que esta línea va creciendo y por eso es que preferimos escribirlo abajo, bueno, vamos a hacerlo en una sola línea, “error”, bueno, nos mostraría el error, “console log error”. Muy bien.
Esto ahora, si termina la instrucción, para que quede más prolijo de manera visual, podríamos indicarlo de este modo para separar, bien, cada método tenemos el punto “then” y el punto “catch”. Bien y efectivamente, miren lo que ocurre aquí, en nuestra consola, no sé si notaron que hay un "delay", pero ya no tenemos "undefined", defectivamente va y nos trae los datos. Vamos a verlo otra vez. Vamos a poner aquí una demora de tres segundos, para que sea más evidente. Guardamos, uno, dos, llegaron los datos. Okey. Pero ya no tenemos el "undefined" que teníamos, cuando se trataba de una función que no manejaba el procedimiento asincrónico. Iba a buscar la información pero no aguardaba al resultado. Ahora tenemos una "promesa", una función asincrónica que nos devuelve una "promesa" y cumple con su "promesa", no nos deja tirados en la banquina, digamos, con los neumáticos pinchados y, ¿qué pasa si hay un error? Vamos a simular lo diciendo que nuestro error está en “true”. Uno, dos , tres, bien y nos devuelve el error, no trae la información, pero tampoco se rompe nuestro programa aquí.
Para que esto fuera más claro podríamos decir, ¡oops!, bien, sería un poco más claro que hubo un error para nosotros, por lo menos aquí en esta etapa. Bien, así es como funciona una "promesa". A continuación vamos a utilizar la API “fetch”, que funciona con el formato de

"promesa", por eso quería que vieran esta sintaxis y se acostumbrasen a ella y vamos a traer información real de una API externa, cualquiera, vamos a elegir alguna de muchas que hay por ahí.