1. **What is a Callback?**

Simply put: A callback is a function that is to be executed after another function has finished executing — hence the name ‘call back’.

More complexly put: In JavaScript, functions are objects. Because of this, functions can take functions as arguments, and can be returned by other functions. Functions that do this are called higher-order functions. Any function that is passed as an argument is called a callback function.

**Why do we need Callbacks?**

For one very important reason — JavaScript is an event driven language. This means that instead of waiting for a response before moving on, JavaScript will keep executing while listening for other events. Let’s look at a basic example:

function first(){

console.log(1);

}

function second(){

console.log(2);

}

first();

second();

As you would expect, the function first is executed first, and the function second is executed second — logging the following to the console:

// 1

// 2

All good so far.

But what if function first contains some sort of code that can’t be executed immediately? For example, an API request where we have to send the request then wait for a response? To simulate this action, were going to use setTimeout which is a JavaScript function that calls a function after a set amount of time. We’ll delay our function for 500 milliseconds to simulate an API request. Our new code will look like this:

function first(){

// Simulate a code delay

setTimeout( function(){

console.log(1);

}, 500 );

}

function second(){

console.log(2);

}

first();

second();

It’s not important that you understand how setTimeout() works right now. All that matters is that you see we’ve moved our console.log(1); inside of our 500 millisecond delay. So what happens now when we invoke our functions?

first();

second();

// 2

// 1

Even though we invoked the first() function first, we logged out the result of that function after the second() function.

It’s not that JavaScript didn’t execute our functions in the order we wanted it to, it’s instead that JavaScript didn’t wait for a response from first() before moving on to execute second().

So why show you this? Because you can’t just call one function after another and hope they execute in the right order. Callbacks are a way to make sure certain code doesn’t execute until other code has already finished execution.

Create a callback

function doHomework(subject, callback) {

alert(`Starting my ${subject} homework.`);

callback();

}

function alertFinished(){

alert('Finished my homework');

}

doHomework('math', alertFinished);

**2.**

**What is a Promise?**

A promise is an object that may produce a single value some time in the future: either a resolved value, or a reason that it’s not resolved (e.g., a network error occurred). A promise may be in one of 3 possible states: fulfilled, rejected, or pending.

**How Promises Work**

A promise is an object which can be returned synchronously from an asynchronous function. It will be in one of 3 possible states:

Fulfilled: onFulfilled() will be called (e.g., resolve() was called)

Rejected: onRejected() will be called (e.g., reject() was called)

Pending: not yet fulfilled or rejected

A promise is settled if it’s not pending (it has been resolved or rejected). Sometimes people use resolved and settled to mean the same thing: not pending.

**3.**

**What is Future in flutter?**

