The power of then - async operations

In the previous example we learned how to apply sync transformations to the result. But what if we need to do another async operation instead?

For example, what if we want to fetch the user from a database, then fetch all of his friends?

Callbacks

With callbacks we can nest the friend fetching operation and pass the original callback.

```
function getUserFriends(id, callback) {
    User.findOne({id: id}, function(err, user) {
        if (err) return callback(err);
        User.find({id: {$in: user.friends}}, callback);
    });
}
```

Promises

With promises, we can return the promise for the fetched friends from inside `.then`

```
function getUserFriends(id) {
    return User.findOne({id: id}).then(function(user) {
        return User.find({id: {$in: user.friends}});
    });
}
```

And we get a promise for the fetched friends outside of `.then`.

Notes

How come this works? If we return a promise from `.then()` won't we get a promise for a promise inside `.then()`?

The answer is no. When `.then()` sees that we have returned a promise, it will also try to "unpack" it to get to an actual value. As long as the unpacking results with another promise, `.then()` will continue to unpack them.

In the callback example, we must explicitly handle the error. Since we can't deal with that error there, we must call the passed callback to pass that error.

In the promise example, if at any point `.then` encounters a promise that resolved with an error, it will stop unpacking and propagate that error through the returned promise.