4C Forms

We're reading data from the user so now let's do two things with it. Let's make sure it is valid data and then let's save it to the server.

1. To prepare, create a *purchase* to POST to the server. In the class add this:

```
Map<String, dynamic> _purchase = {
  "seats": [1, 2, 3],
  "showing_id": 1,
};
```

This will represent the entire purchase that we'll submit to the server. Note that the seats for a showing are being hardcoded. We'll link them to the real purchase in a later lab.

2. Surround all of your fields with a Form() widget. It can be anywhere around the fields, so it may be convenient to put it around the Column. You can decide how you want to do it.

Saving the data

Remember that in order to save a Form, you need a key.

3. Create a GlobalKey<FormState> called *_key* and assign it to the Form. It's of type FormState so that it'll have the needed .validate() and .save() methods.

4. The TextFormFields are okay as they are, but the DropdownButtons will need to wrapped in a FormField so that they have onSaved and validate properties. Go ahead and wrap them now.

5. You already have a FAB for purchasing. Make its onPressed event call a _checkout method.

6. _checkout doesn't exist so let's create it. It should receive nothing and return a void.

7. In _checkout, you'll call the key.currentState's save() method.

Remember that this will fire the onSaved event handlers in every field inside the Form.

8. Since onSaved is being fired, we should write an onSaved() event for each field. Each onSaved will receive a value. Go ahead and add that to the _purchase Map. Here's an example for the firstName field:

```
onSaved: (val) => _purchase["firstName"] = val,
```

9. Write an onSaved for each field like the above.

10. Back in _checkout(), put this in after the key.currentState's save():

```
buyTickets(purchase: _purchase).then((res) {
  // Response will have an array of ticket numbers.
  print("success!");
}).catchError((err) {
  print("Error purchasing. ${err}");
});
```

This calls buyTickets which will reserve the seats for that showing and return some ticket numbers so that we can provide tickets to our customer.

11. Run and test. Run through the debugger and/or add some print statements to make sure you're sending some data to the server.

Validating the data

We're now getting data from the user and sending it to the API server. That's very cool. But if you put in nonsense data, it still submits. Shouldn't we make sure we send good data to the server? Let's validate the data as soon as the user enters it.

12. Put in validators. Pick one or two of these business requirements and implement them:
    - First name, last name, and credit card number are required.
    - Email OR cell are required
    - Credit card number looks like a credit card number.
    - Month and Year must be in the future.
    - CVV is three digits

    Hints:
    - Each Form Field can have its own validate().
    - A validate property must be a function that receives a val and returns ...
        - A null if everything is okay
        - A String with the error message if it isn't valid.
    - Regular expressions come in really handy when you want to make sure strings match a pattern. Dart handles them with the RegExp class.

13. Look in the _checkout function. Add this line to it:

```
if (!_key.currentState!.validate()) return;
```

14. Bonus!! If you want your validation to happen as the user is entering data, add this to your form:

```
autovalidateMode: AutovalidateMode.onUserInteraction,
```