Exercise 1

1. The 2 lines of code that make the list infinitely scrolling are:

```
if (index >= _suggestions.length) {
  _suggestions.addAll(generateWordPairs().take(10));
}
```

   If we remove these lines, then when we reach to a position in that is greater than the lists' size, thus we'll be out of range and receive and error message.

2. We may use the **ListView.separated** instead of **ListView.builder** this way:

```
final List<WordPair> entries = generateWordPairs().take(100).toList();
return ListView.separated(
  padding: const EdgeInsets.all(8),
  itemCount: 100,
  itemBuilder: (BuildContext context, int index) {
    return SizedBox(
      height: 50,
      child: Center(child: Text('${entries[index]}')),
    );  // SizedBox
  },
  separatorBuilder: (BuildContext context, int index) => const Divider(),
);  // ListView.separated
```

   I think, given the assumption that the list is finite, it's better to use the **ListView.separated**, since here the divider is part of the widget's properties (i.e., the separator), while with **ListView.builder**, we need to address the divider as a list item – even though logically, it is not.

   (That said, in an infinite list it's better to use the **ListView.builder** since the **itemCount** is not a required field, while it's required for the separated)

3. We need to call **setState()** in the **_buildRow** to be able to update the state of the widget when the user taps the heart icon. When the user clicks the heart, we need to re-render the screen to color the icon. The widget that contains the component that is interacted is the **_buildRow,** so that's where we'll call **setState()**.

Exercise 2

1. The purpose of MaterialApp widget is to wrap several widgets that are commonly required for material design applications. It also configures a Navigator for the app to search for routes. Properties for example:
    - *home* – The widget for the default route of the app (which is "/" for Navigator).
    - *darkTheme* – The ThemeData to use when a 'dark mode' is requested by the system.
    - *debugShowCheckedModeBanner* - Turns on a little "DEBUG" banner in debug mode to indicate that the app is in debug mode. This is on by default (in debug mode).
2. The key property is used to identify the Dismissible widgets that are under the same parent in the components tree. This property is required, that is, it can't be null. The reason for that is that Dismissibles are commonly used in lists and removed from the list when dismissed. Without keys, the default behavior is to sync widgets based on their index in the list, which means the item after the dismissed item would be synced with the state of the dismissed item. Using keys causes the widgets to sync according to their keys.