Dry exercise

Answer the following questions:

1. Identify the 2 lines of code that make the list **infinitely scrolling**. What happens if we remove these lines, and scroll to the end of the list? Assume `_suggestions` starts with a non-empty list of 10 word-pairs. You can initialize `_suggestions` with `generateWordPairs().take(10).toList()`.
   Hint: run the changed code (complete restart, instead of hot reload, as the initialization of a state's global variable has changed) to see if you're right.
   Do not submit the changed code!

Solution:

The 2 lines that make the list infinitely scrolling are:

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

When i removed those two line and initialize _suggestions with the given value in the question, i get a list with 10 items, and after that an error, that 10 is not in the range of 0..9, and that is make sense because the indexes start from 0 to 9, and there is no place that we are changing the index (to see that we don't get Range Error)

2. Our list is separated with **Divider** widgets. Visit the **ListView** widget documentation here: https://api.flutter.dev/flutter/widgets/ListView-class.html; Give a different method to construct such a list with dividers. Which way do you think is better, and why?
   You may assume for this question only that the list is finite and contains 100 items from the start.

Solution:

More Generic way to make a list with a fixed number of items is to use
ListView.separated
I think this way is better because it's also more Readable, which is very important to us so other developers can understand our code.
Also, it provides more control over the appearance, compared to the previous

method, we can specify a custom separatorBuilder that returns a widget for each separator, allowing us to customize the divider's appearance, size, color and more.

3. `ListView.builder`'s `itemBuilder` contains a call to `setState()` inside the `onTap()` handler. Why do we need it there?

Solution:

setState() is used to update the state of the app and rebuild the widget tree when the user taps on a ListTile.
I will try to explain it further:
The ListView.builder's itemBuilder creates a ListTile for each suggestion and renders them on the screen.
Each ListTile has an onTap() handler that updates the state of _saved set and rebuilds the widget tree using setState().
Without setState(), the changes made to _saved set would not be reflected on the UI, and the app would not show the updated state of _saved set.
And therefore, we can understand that we need this there so we can see changes in each time we pressed the icon that appear in each row.

Dry exercise

Answer the following questions:

1. What is the purpose of the **MaterialApp** widget? Provide examples of 3 of its properties followed by a short explanation.

Solution:

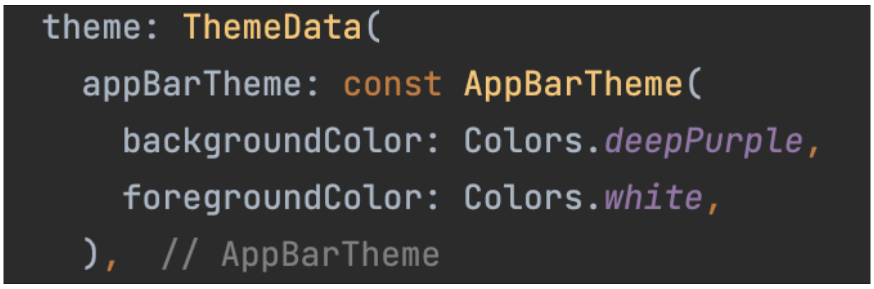
The MaterialApp widget is a fundamental widget that provides a lot of essential functionality for creating visually appealing and interactive mobile applications with Material Design.
It serves as the top-level widget for a Flutter application and is used to specify various app-wide attributes and settings such as app title, theme, routes, and more.

Examples of use:

1. **Title** - This property sets the title of the app which appears in the device's task switcher and notification center.
   It is also used by the AppBar widget when the automaticallyImplyLeading property is set to true.

   Example:
   ```
   title: 'Startup Name Generator'
   ```

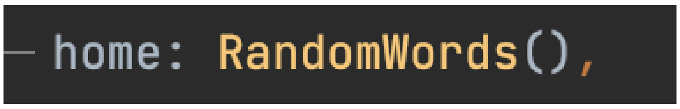
2. **Theme** - This property allows us to customize the app's look and feel by defining the colors, typography, and other visual properties.

   Example:
   

```
theme: ThemeData(
  appBarTheme: const AppBarTheme(
    backgroundColor: Colors.deepPurple,
    foregroundColor: Colors.white,
  ),  // AppBarTheme
```

3. **Home -** This property sets the screen that the app will start from.
   The widget for the default route of the app.

   Example:



```
home: RandomWords(),
```

2. The **Dismissible** widget has a key property. What does it mean and why is it required?

Solution:

The key property of a Dismissible widget is used to uniquely identify the widget within the parent widget tree.
When a widget is dismissed, it is removed from the widget tree, and if it needs to be added back to the tree (for example, if it was dismissed by accident), the key is used to recreate the widget with the same state as before it was dismissed.
In other words, the key property is required to maintain the state of the dismissed widget.
If the key property is not set or is not unique, Flutter will not be able to recreate the widget with its previous state, leading to unexpected behavior.