YOUR FIRST APP

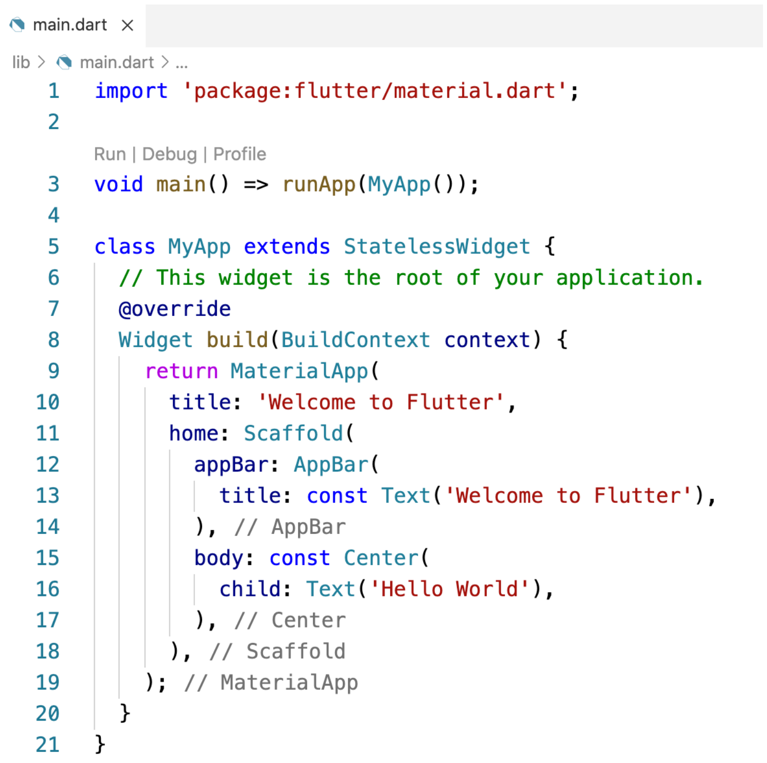
1 WHAT YOU'LL LEARN

- How to create a Flutter app
- How to use an external package
- How to add a Stateful widget
- How to create an infinite scrolling ListView

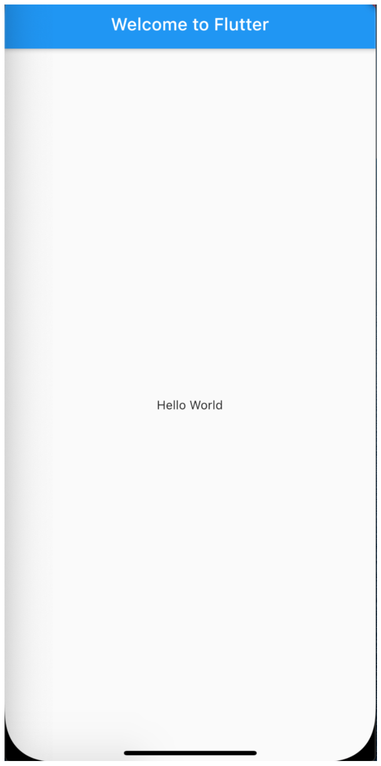
2 CREATE A FLUTTER APP

2.1 Create a Flutter app

- Use section Create the app in Lesson01 to create a Flutter app with name listapp.
- Replace the contents of **lib/main.dart**.
- Delete all of the code from lib/main.dart.
- Replace with the following code, which displays "Hello World" in the center of the screen.



```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ), // AppBar
        body: const Center(
          child: Text('Hello World'),
        ), // Center
      ), // Scaffold
    ); // MaterialApp
  }
}
```



- Run the app and see the result.

2.2 Observations

- This example creates a Material app. **Material** is a visual design language that is standard on mobile and the web. Flutter offers a rich set of Material widgets. It's a good idea to have a **uses-material-design: true** entry in the **flutter** section of your **pubspec.yaml** file. This will allow you to use more features of Material, such as their set of predefined **Icons**.

Material: https://material.io/design

Material Icons: https://fonts.google.com/icons

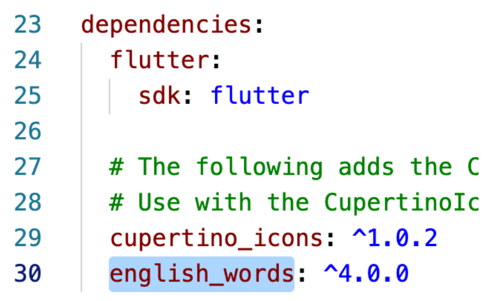
- The **main()** method uses arrow (=>) notation. Use arrow notation for one-line functions or methods.
- The app extends **StatelessWidget**, which makes the app itself a widget. In Flutter, almost everything is a widget, including alignment, padding, and layout.
- The **Scaffold** widget, from the Material library, provides a default **app bar**, and a **body property** that holds the widget tree for the home screen. The widget subtree can be quite complex.
- A widget's main job is to provide a build() method that describes how to display the widget in terms of other, lower level widgets.
- The body for this example consists of a Center widget containing a Text child widget. The Center widget aligns its widget subtree to the center of the screen

3 USE AN EXTERNAL PACKAGE

In this section, you'll start using an open-source package named **english_words**, which contains a few thousand of the most used English words plus some utility functions. https://pub.dev/packages/english_words

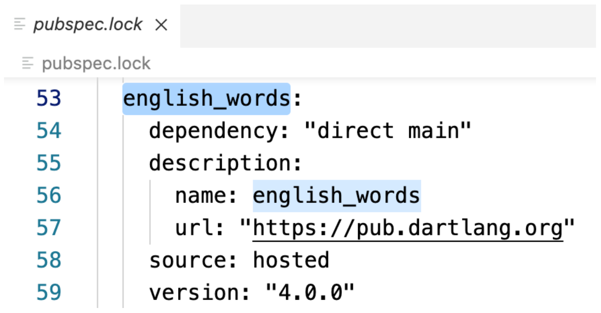
You can find the english_words package, as well as many other open source packages, on pub.dev. https://pub.dev

1. The **pubspec.yaml** file manages the assets and dependencies for a Flutter app. In **pubspec.yaml**, add **english_words** (3.1.5 or higher) to the dependencies list:



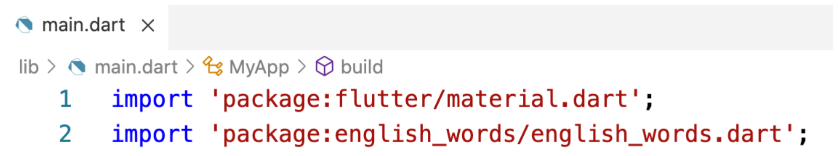
```
dependencies:
  flutter:
    sdk: flutter

  # The following adds the C
  # Use with the CupertinoIc
  cupertino_icons: ^1.0.2
  english_words: ^4.0.0
```

2. While viewing the **pubspec.yaml** click Get Packages to pull the package into your project. This will auto-generates the pubspec.lock file with a list of all packages pulled into the project and their version numbers.



```yaml
english_words:
  dependency: "direct main"
  description:
    name: english_words
    url: "https://pub.dartlang.org"
  source: hosted
  version: "4.0.0"
```

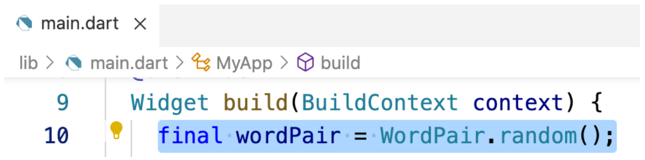
3. In **lib/main.dart**, import the new package:



```dart
import 'package:flutter/material.dart';
import 'package:english_words/english_words.dart';
```

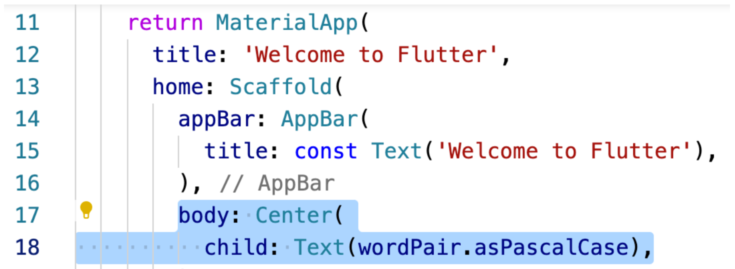
4. Use the **English words package** to generate the text instead of using the string “Hello World”:

+ add variable wordPair



```dart
  Widget build(BuildContext context) {
    final wordPair = WordPair.random();
```

+ replace ‘Hello World’ by wordPair.asPascalCase



```dart
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ), // AppBar
        body: Center(
          child: Text(wordPair.asPascalCase),
```

“Pascal case” (also known as “upper camel case”), means that each word in the string, including the first one, begins with an uppercase letter. So, “uppercamelcase” becomes “UpperCamelCase”.

5. If the app is running, hot reload to update the running app. Each time you click hot reload, or save the project, you should see a different word pair, chosen at random, in the running app. This is because the word pairing is generated inside the build method, which is run each time the MaterialApp requires rendering, or when toggling the Platform in Flutter Inspector.

4 ADD A STATEFUL WIDGET

Stateless widgets are immutable, meaning that their properties can’t change—all values are final.

Stateful widgets maintain state that might change during the lifetime of the widget. Implementing a stateful widget requires at least two classes: 1) a **StatefulWidget** class that creates an instance of 2) a **State** class. The **StatefulWidget** class is, itself, immutable and can be thrown away and regenerated, but the **State** class persists over the lifetime of the widget.

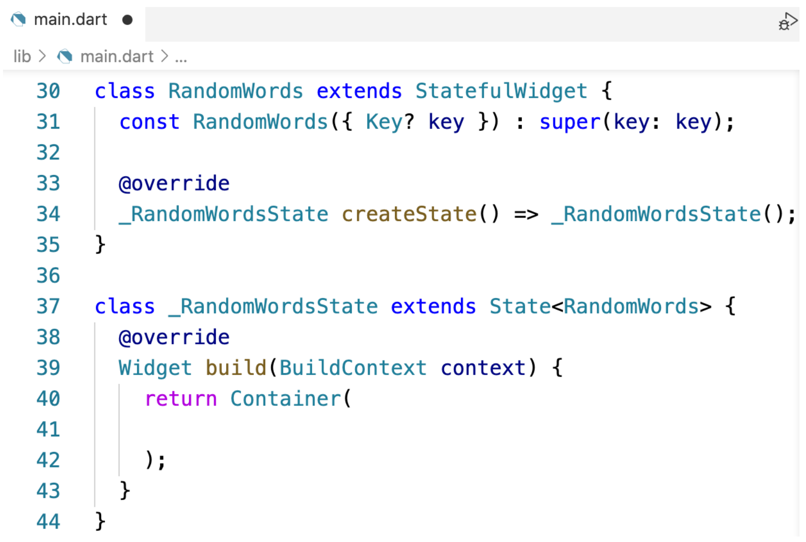
In this section, you’ll add a stateful widget, **RandomWords**, which creates its **State** class, **_RandomWordsState**. You’ll then use **RandomWords** as a child inside the existing **MyApp** stateless widget.

1. Create the boilerplate code for a stateful widget.
   In **lib/main.dart**, position your cursor after all of the code, enter **Return** a couple times to start on a fresh line. In your IDE, start typing **stful**. The editor asks if you want to create a **Stateful** widget. Press **Return** to accept. The boilerplate code for two classes appears, and the cursor is positioned for you to enter the name of your stateful widget.
2. Enter **RandomWords** as the name of your widget.
   The **RandomWords** widget does little else beside creating its **State** class.

Once you’ve entered **RandomWords** as the name of the stateful widget, the IDE automatically updates the accompanying **State** class, naming it **_RandomWordsState**. By default, the name of the **State** class is prefixed with an underbar. Prefixing an identifier with an underscore enforces privacy in the Dart language and is a recommended best practice for **State** objects.

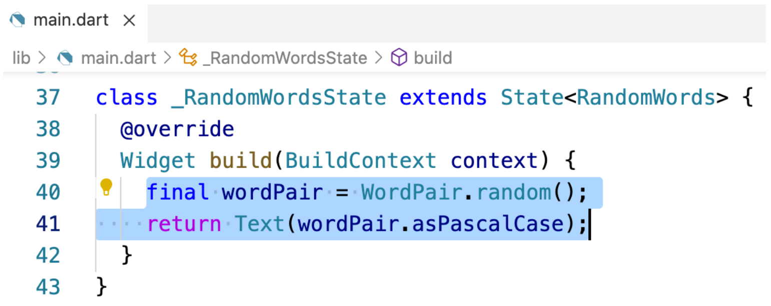
The IDE also automatically updates the state class to extend **State<RandomWords>**, indicating that you’re using a generic **State** class specialized for use with **RandomWords**. Most of the app’s logic resides here—it maintains the state for the **RandomWords** widget. This class saves the list of generated word pairs, which grows infinitely as the user scrolls and, in the next section, favorites word pairs as the user adds or removes them from the list by toggling the heart icon.

https://api.flutter.dev/flutter/widgets/State-class.html



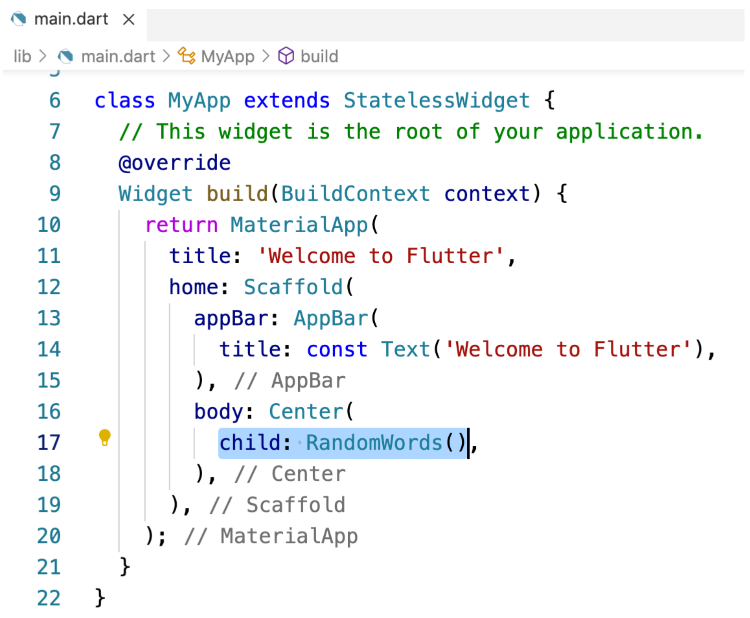
```dart
class RandomWords extends StatefulWidget {
  const RandomWords({ Key? key }) : super(key: key);

  @override
  _RandomWordsState createState() => _RandomWordsState();
}

class _RandomWordsState extends State<RandomWords> {
  @override
  Widget build(BuildContext context) {
    return Container(

    );
  }
}
```

3. Update the **build()** method in **_RandomWordsState:**



```dart
class _RandomWordsState extends State<RandomWords> {
  @override
  Widget build(BuildContext context) {
    final wordPair = WordPair.random();
    return Text(wordPair.asPascalCase);
  }
}
```

4. Remove the word generation code from **MyApp** by making the changes shown below.



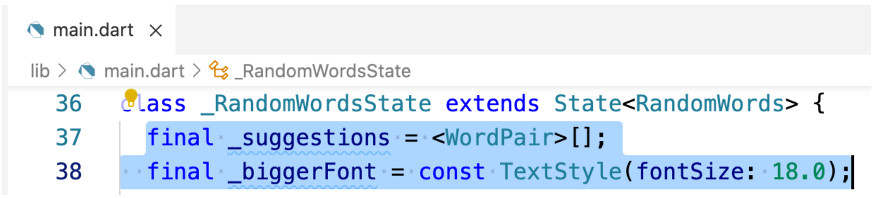
```dart
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ), // AppBar
        body: Center(
          child: RandomWords(),
        ), // Center
      ), // Scaffold
    ); // MaterialApp
  }
}
```

5. Restart the app. The app should behave as before, displaying a word pairing each time you hot reload or save the app.

5 CREATE SCROLLING LISTVIEW

In this section, you'll expand **_RandomWordsState** to generate and display a list of word pairings. As the user scrolls the list (displayed in a **ListView** widget) grows infinitely. **ListView's builder** factory constructor allows you to build a list view lazily, on demand.

1. Add a **_suggestions** list to the **_RandomWordsState** class for saving suggested word pairings. Also, add a **_biggerFont** variable for making the font size larger.



main.dart ×

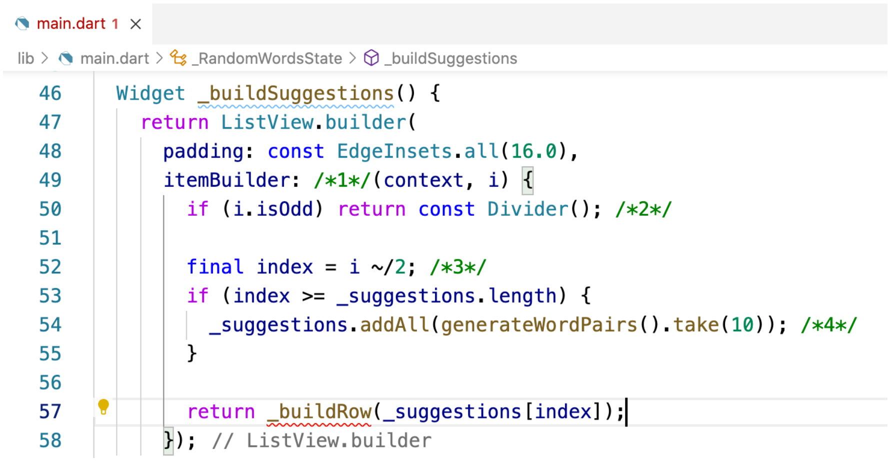
lib > main.dart > _RandomWordsState

```
class _RandomWordsState extends State<RandomWords> {
  final _suggestions = <WordPair>[];
  final _biggerFont = const TextStyle(fontSize: 18.0);
```

Next, you'll add a **_buildSuggestions()** function to the **_RandomWordsState** class. This method builds the **ListView** that displays the suggested word pairing.

The **ListView** class provides a builder property, **itemBuilder**, that's a factory builder and callback function specified as an anonymous function. Two parameters are passed to the function—the **BuildContext**, and the row iterator, **i**. The iterator begins at 0 and increments each time the function is called. It increments twice for every suggested word pairing: once for the ListTile, and once for the Divider. This model allows the suggested list to continue growing as the user scrolls.

2. Add a **_buildSuggestions()** function to the **_RandomWordsState** class:



main.dart 1 ×

lib > main.dart > _RandomWordsState > _buildSuggestions

```
Widget _buildSuggestions() {
  return ListView.builder(
    padding: const EdgeInsets.all(16.0),
    itemBuilder: /*1*/(context, i) {
      if (i.isOdd) return const Divider(); /*2*/

      final index = i ~/2; /*3*/
      if (index >= _suggestions.length) {
        _suggestions.addAll(generateWordPairs().take(10)); /*4*/
      }

      return _buildRow(_suggestions[index]);
    }); // ListView.builder
```

/*1*/ The **itemBuilder** callback is called once per suggested word pairing, and places each suggestion into a **ListTile** row. For even rows, the function adds a **ListTile** row for the word pairing. For odd rows, the function adds a **Divider** widget to visually separate the entries. Note that the divider might be difficult to see on smaller devices.

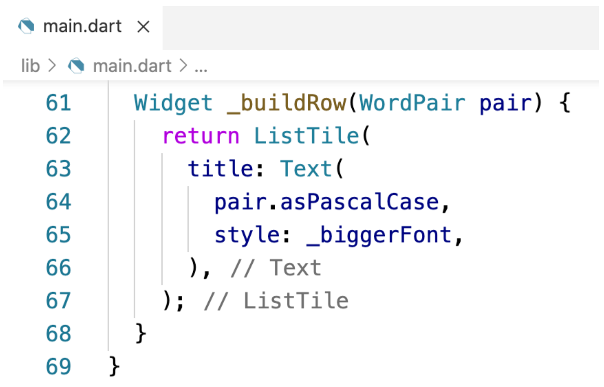
/*2*/ Add a one-pixel-high divider widget before each row in the **ListView**.

/*3*/ The expression i ~/ 2 divides i by 2 and returns an integer result. For example: 1, 2, 3, 4, 5 becomes 0, 1, 1, 2, 2. This calculates the actual number of word pairings in the **ListView**, minus the divider widgets.

/*4*/ If you've reached the end of the available word pairings, then generate 10 more and add them to the suggestions list.

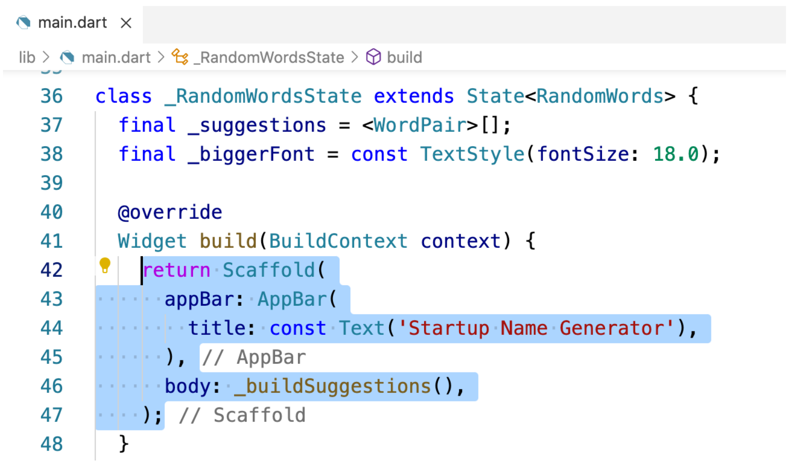
The **_buildSuggestions()** function calls **_buildRow()** once per word pair. This function displays each new pair in a **ListTile**, which allows you to make the rows more attractive in the next step.

3. Add a **_buildRow()** function to **_RandomWordsState**:



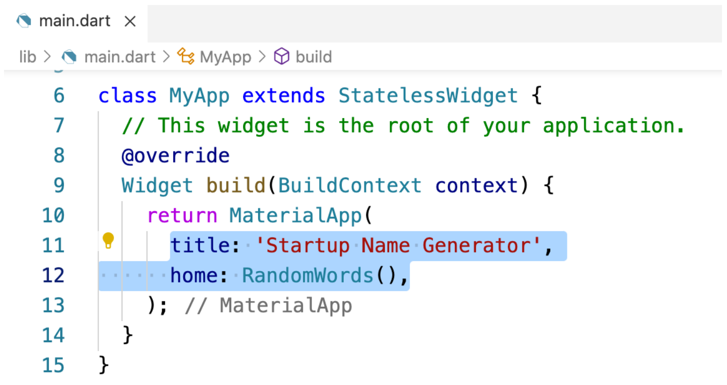
```
  Widget _buildRow(WordPair pair) {
    return ListTile(
      title: Text(
        pair.asPascalCase,
        style: _biggerFont,
      ), // Text
    ); // ListTile
  }
}
```

4. In the **_RandomWordsState** class, update the **build()** method to use **_buildSuggestions()**, rather than directly calling the word generation library. (**Scaffold** implements the basic **Material Design** visual layout.) Replace the method body with the highlighted code:



```
class _RandomWordsState extends State<RandomWords> {
  final _suggestions = <WordPair>[];
  final _biggerFont = const TextStyle(fontSize: 18.0);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Startup Name Generator'),
      ), // AppBar
      body: _buildSuggestions(),
    ); // Scaffold
  }
```

5. In the **MyApp** class, update the **build()** method by changing the title, and changing the home to be a **RandomWords** widget:



main.dart

lib > main.dart > MyApp > build

```dart
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Startup Name Generator',
      home: RandomWords(),
    ); // MaterialApp
  }
}
```

6. Restart the app. You should see a list of word pairings no matter how far you scroll.