

Q1: Can we nest the Scaffold widget? Why or Why not?

Answer : The answer is simply yes, because as in the flutter all the entities are widget, so the Scaffold is also a widget itself and we can create the widget tree with the help of several widgets.

Q2: What are the different ways we can create a custom widget

Answer : There are basically two types of widgets over there. Statefull and Stateless widgets. By using these we can create our own widgets and also can apply custom themes as well.

Q3: How can I access platform(iOS or Android) specific code from Flutter?

Answer : If we are talking about the platform specific design than we can you the following import :

import 'dart:io' show Platform

Or else if we are talking about the logical codes then we have to use method channels for that and have to implement those code natively in **android** and **iOS** separately and call those methods from the flutter program.

Q4: What is BuildContext? What is its importance?

Answer : Every widget in the flutter has its own **BuildContext**. A **BuildContext** is nothing but a reference to the location of the widget in the widget tree. It is used to get reference to the theme or another widget.

For example if we want to show the dialog then we have to use the reference of our scaffold. The code of these is simply **Scaffold.of(context)**

Q5: Refactor the code below so that the children will wrap to the next line when the display width is small for them to fit.



```
class LongStringWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(children: [
      Chip(label: Text('I')),
      Chip(label: Text('am')),
      Chip(label: Text('looking')),
      Chip(label: Text('for')),
      Chip(label: Text('a')),
      Chip(label: Text('job')),
      Chip(label: Text('and')),
      Chip(label: Text('I')),
      Chip(label: Text('need')),
      Chip(label: Text('a')),
      Chip(label: Text('job')),
    ]);
  }
}
```

Answer :

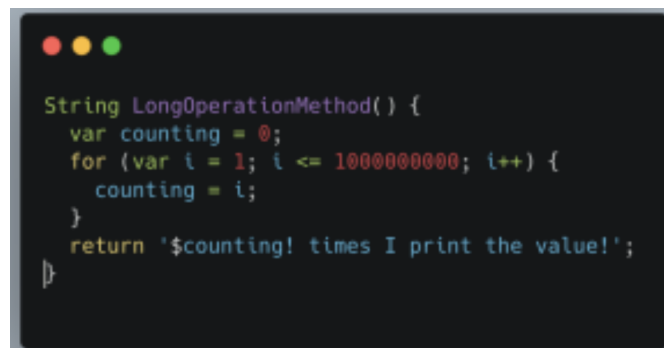
```
class LongStringWidget extends StatelessWidget {
  LongStringWidget({Key? key}) : super(key: key);

  List<String> sampleString =
    ["I", "am", "looking", "for", "a", "job", "and", "I", "need", "a", "job"];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Row(
        children: sampleString.map((e) => Expanded(child: Chip(label:
Text(e)))).toList(),
      ),
    );
  }
}
```

Also we can replace **Row** with **Wrap** widget as well so there will be no rendering issues as well.

Q6: Identify the problem in the following code block and correct it.



```
String LongOperationMethod() {
  var counting = 0;
  for (var i = 1; i <= 10000000000; i++) {
    counting = i;
  }
  return '$counting! times I print the value!';
}
```

Answer : The method will work fine, but there is a long running task in there so it will cause the UI to be stuck. We have two solutions, we can make the method **async** or we can use a separate **isolate** for that as well. By Async the method will run on same isolate only.

Using Async :

```
Future<String> LongOperationMethod() async {
  var counting = 0;
  await Future(() {
    for (var i = 1; i <= 10000000000; i++) {
      counting = i;
    }
  });
  return '$counting! time I print the value';
}
```

Using Isolate :

```
Future<String> DoTheOperartion() async {
  return await compute(LongOperationMethod, 10000000000);
}
```

```
String LongOperationMethod(int countTo) {
  var counting = 0;
  for (var i = 1; i <= countTo; i++) {
    counting = i;
  }
  return '$counting! time I print the value';
}
```

Q7: In the below code, list1 declared with *var*, list2 with *final* and list3 with *const*. What is the difference between these lists? Will the last two lines compile?



```
var list1 = ['I', '❤️', 'Flutter'];

final list2 = list1;
list2[2] = 'Dart'; // Will this line compile?

const list3 = list1; // Will this line compile?
```

Answer :

The first line `var list1 = ['I', '❤️', 'Flutter'];` (Inferred data type)
is equivalent to `List<String> list1 = ['I', '❤️', 'Flutter'];` (explicitly declared)

With `final` and `const`, we can not reassign a new value after the initial assignment. `final` values are assigned once at runtime and a `const` variable value has to be either known at compile time, or hard coded before you run your app.

The third line will compile. We are not reassigning the `list2` list itself, but changing the value of an item in the third index position. Lists are mutable by default in Dart.

The fourth line will not compile because the value of `list1` isn't assigned until runtime.