

Descriptive Questions

Q1. Can we nest the Scaffold widget ? Why or why not?

Yes we can use nested scaffold widget in our flutter project. If we look at the inner class of scaffold widget we get Scaffold is a class in dart it accepts body in constructor and scaffold is also a type of widget so we can use another Scaffold here (means Nested Scaffold).

```
class Scaffold extends StatefulWidget {  
  const Scaffold({  
    Key? key,  
    this.appBar,  
    this.body,  
    this.floatingActionButton,  
    ...  
    ...  
  }) : super(key: key);  
}
```

So we can pass another scaffold in the body but this is not recommended because creating a nested Scaffold widget will decrease the performance of our app. Scaffold contains all the basic material design of our app like appBar, bottom navigation or drawer . Nested Scaffold simply means we are re-loading all the objects of the scaffold two times whenever we are loading our screen which is not recommended.

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

By Using Custom Widget in our app we can reduce the usage of code and if we want to modify any widget and want to achieve our design goal. Custom widget makes it easier to add the same functionality throughout the code which also saves development and test time of our app.

We can create a custom widget by using 4 ways.

- a.) Stateless Widget
- b.) Stateful Widget
- c.) RenderObjectWidget
- d.) Preferred Size Widget

Stateless Widget & Stateful widget:

Here I'm creating a custom TextField.

```
class CustomTextField extends StatelessWidget {

  String text = "";
  double fontSize = 0;

  CustomTextField({
    required this.text,
    required this.fontSize,
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return TextField(
      style: TextStyle(fontSize: fontSize,
        fontStyle: FontStyle.italic,
        color: Colors.black),
      decoration: InputDecoration(
        hintText: text,
        hintStyle: TextStyle(fontSize: fontSize,
          fontStyle: FontStyle.italic,
          color: Colors.black),
      ),
    );
  }
}
```

Render Object : The RenderObject is the piece of code responsible for computing the layout and painting it to the screen. Our custom class must be extended with RenderObjectWidget . **We have 3 types of RenderObjectWidget.**

- a.) **LeafRenderObjectWidget :** If our custom **widget have no children** then we will extend our class with LeafRenderObject
- b.) **SingleChildRenderWidget :** If our custom **widget has one and only one child** then our class will extend from SingleChildRenderWidget.
- c.) **MultiChildRenderObjectWidget :** If our custom **widget has two or more children** then we will extend our class from MultiChildRenderObjectWidget.

PreferredSize Widget : Preferred Size is a custom widget in flutter through which we can design our custom appBar in our app. We can also create tabs or more effective design for our appBar by creating a customChild for your appBar with the help of PreferredSizeWidget.

```
class MyWidget extends StatelessWidget implements PreferredSizeWidget
{
  @override
  Size get preferredSize => Size.fromHeight(kToolbarHeight);

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

Q3. How can I access platform (ios or Android) specific code from flutter?

In flutter we can access specific code by using platform channels. Platform channels provide an interface to call methods and pass the data between the code we have written in native. Basically we have 3 platform channels through which we access our platform specific code.

a.) Message Channel b.) Method Channel c.) Event Channel

Message Channel passes the data in the form of bytes so whenever we have a requirement to pass the data as bytes like we are passing the image so we will use Message Channel.

Method Channel passes the data from native to dart side but it doesn't support continuous stream of data.

EventChannel passes the data from native to dart side but it supports a continuous stream of data. This is useful if we want to send data every time a particular event occurs.

For eg. When the user turns on or off the location service.

Step 1: Open Android project folder in your IDE then create your java/kotlin native class. Create your function and channel in your native class.

Step 2: For Ios, Open ios module in Xcode and create your channel in AppDelegate.swift file. Now we can call our platform channels from the dart side.

I have implemented demo sample app on my Github profile:

Method Channel : <https://github.com/Anonymousgaurav/Flutter2Native>

Event Channel : https://github.com/Anonymousgaurav/light_sensor_event_channel

Q4. What is BuildContext? What is its importance?

BuildContext is an abstract class in flutter which is used to get the location of a widget. As we know in flutter we have stateful and stateless widget and both these classes extend widget. Each time a widget is placed in a tree, the widget is inflated into Element. So each element implements BuildContext and this is used to get the location of each widget. Each widget we create has its own build context.

Build context is important bcz without BuildContext we cannot get the location of our widget. And if we have any requirement to get the size of screen or widget or we have to show a snackbar or the best example is Navigator . Flutter doesn't understand what is the exact location where to start and where to go without context or Where to show the snackbar at which location.

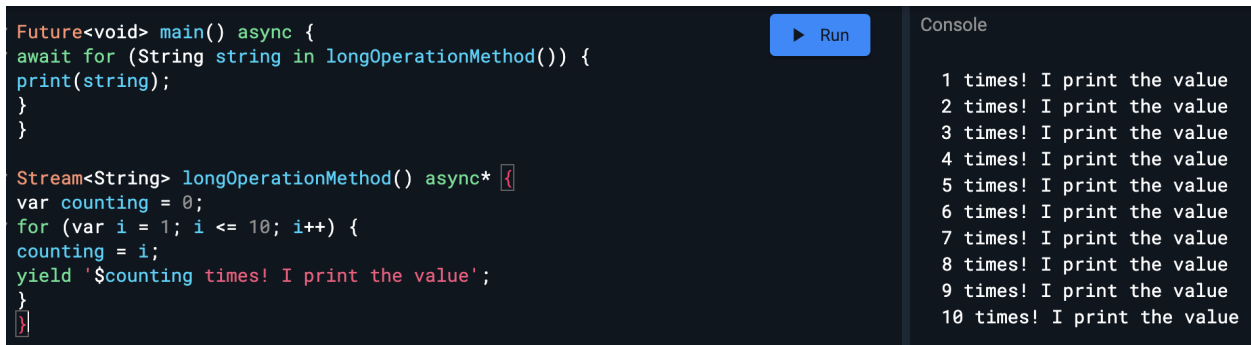
Flow => StatefulWidget extends Widget and Widget is Inflated in Element and Each Element implements BuildContext

Coding Questions

Q1.

```
return Scaffold(  
  body: Wrap(  
    children: const [  
      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")),  
    ],  
  )); // Wrap, Scaffold  
}
```

Q2.



The screenshot shows a code editor with the following Dart code:

```
Future<void> main() async {  
  await for (String string in longOperationMethod()) {  
    print(string);  
  }  
}  
  
Stream<String> longOperationMethod() async* {  
  var counting = 0;  
  for (var i = 1; i <= 10; i++) {  
    counting = i;  
    yield '$counting times! I print the value';  
  }  
}
```

A "Run" button is visible next to the code. To the right, the console output shows 10 lines of text:

```
1 times! I print the value  
2 times! I print the value  
3 times! I print the value  
4 times! I print the value  
5 times! I print the value  
6 times! I print the value  
7 times! I print the value  
8 times! I print the value  
9 times! I print the value  
10 times! I print the value
```

Q3.

Variables with var keyword means we have no surety of data type. Like if we are getting the data from a remote server and we have no idea whether the data is a type of int or string ... so we will use var in this case.

Elements of the final list are immutable. Immutable means When we declare the object with the final keyword then it means we can't change the instance of that object. We will assign the value to final object at the runtime otherwise we will get the runtime error

Const : const is a keyword to make a value immutable means if we declare any value with const keyword then we cannot change the value in future. We will assign the value of const during compile time otherwise we will get a compile time error.



The screenshot shows a code editor with the following Dart code:

```
var list1 = ['I', '♥', 'Flutter'];  
  
final list2 = list1;  
list2[2] = 'Dart'; // Will this line compile?  
  
const list3 = list1; // Will this line compile?
```

So,

Line 1 :

```
var list1 = ["I", "😊", "Flutter"]
```

This line will be compiled successfully becuz we have no suritey of our data in the list so use of var is ok here.

Line 2&3:

```
final list2 = list1;  
List2[2] = "Dart";
```

Here code will compile successfully .we are just using the final keyword in list2 and replacing the data of list2. If we observe we are just replacing the element in the same instance means we are not modifying our instance of list2.

Let me prove this one:

```
▼ void main() {  
  var list1 = ["i", "😊", "Gaurav" ];  
  var list2 = list1;  
  print(list2.hashCode);  
  list2[2] = "Flutter";  
  print(list2.hashCode);  
}
```

Run

Console

990007869
990007869

We are printing the value of hashcode before assigning the value in list2 and after assigning the value in list2. So we are getting the same hashcode so we are not modifying the instance of list2 . i.e, line 2 will compile successfully.

Line 3:

```
const list3 = list1;
```

This line will not compile because list1 is of var type and we are assigning var type to const. We can only assign the const values in list1. For e.g: const list3 = list1; this will not throw error if const list1 = ["1",2,3];

