# Flutter Interview Question



I’m just trying to share my thoughts and understanding of flutter. I took all this content from Wikipedia, Stack Overflow and some other’s medium articles. Basically, I’m wrapping all this content in my own preferred order/format. I believe that this article will help you to get a basic idea of flutter and which may be used to recall your flutter knowledge while preparing for interviews.

**What is Flutter?**

Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, iOS, Linux, Mac, Windows and the web from a single codebase. Flutter apps are written in the Dart language.

What is Dart?

Dart is a client-optimized programming language for apps on multiple platforms. It is developed by Google and is used to build mobile, desktop, server, and web applications. Dart is an object-oriented, class-based, garbage-collected language with C-style syntax. Dart can compile to either native code or JavaScript.

Design-specific widgets

The Flutter framework contains two sets of widgets which conform to specific design languages.

* **Material Design** widgets implement Google’s design language of the same name.
* **Cupertino widgets** implement Apple’s iOS Human interface guidelines

Flutter/Dart naming conventions

* [UpperCamelCase](https://dart.dev/guides/language/effective-dart/style) - Classes, enum types, typedefs, extensions and type parameters.
* [lowerCamelCase](https://dart.dev/guides/language/effective-dart/style) - Identifiers, constants and methods.
* [lowercase\_with\_underscores](https://dart.dev/guides/language/effective-dart/style) - Libraries, packages, directories, and source files.

Flutter Doctor

flutter doctor

This command checks your environment and displays a report of the status of your Flutter installation. The output of this command shows the list of software you might need to install or further tasks to perform.

Flutter Clean

flutter clean

This command clears your project’s build cache.

What is pubspec.yaml file in flutter?

[pubspec.yaml](https://dart.dev/tools/pub/pubspec) is a file where you can provide all the dependencies and assets (plugins, Dart version, font, images) of your Flutter project. It’s also a place where you configure the project’s name and description.

Flutter pub commands

* **flutter pub get** - This command gets all the dependencies listed in the pubspec.yaml file in the current working directory, as well as their transitive dependencies. If the system cache doesn’t already contain the dependencies, pub get updates the cache, downloading dependencies if necessary. To map packages back to the system cache, this command creates a .packages file.
* **flutter pub upgrade** - This command gets the latest versions of all the dependencies listed in the pubspec.yaml file in the current working directory, as well as their transitive dependencies.

What are widgets?

Everything in Flutter is a widget, but what exactly is a widget? A widget is a user interface element.

How many types of widgets are in Flutter?

A widget is either stateful or stateless.

* StatelessWidget
* StatefulWidget

What is the difference between stateful and stateless widgets?

* **Stateless widgets** do not require mutable state, i.e., it is immutable. Stateless widgets have their state created only once — therefore, the build method is called only once as well. In simple words, Stateless widgets cannot change their state during the runtime of the app, which means the widgets cannot be redrawn while the app is in action. If you want to redraw the Stateless Widget, then you will need to create a new instance of the widget.
* **Stateful widgets** have a mutable state, i.e., they are mutable and can be drawn multiple times within its lifetime. They are the widgets which can change their state multiple times and can be redrawn on to the screen any number of times while the app is in action.

What does State mean? What is setState()?

The state is a collection of the variable values/properties of your widget. Anything that can change like a counter count, text, etc. can be part of State.

setState() is essentially a way of telling the app to refresh and rebuild the screen with the new values/properties.

State objects are created by the framework. In order to change your widget, you need to update the state object which can be done using setState() function available for Stateful widgets. setState() sets properties of state object which in turn triggers the update to the UI. But this technique has its own drawbacks.

What’s the lifecycle of a ‘StatefulWidget’?

* createState() — a method in a StatefulWidget that’s called immediately and should return a State object for the widget
* initState() — the first method called in the State object after the widget is created
* didChangeDependencies() — called immediately after initState() the first time the widget is built
* build() — very similar to the one from the StatelessWidget. It’s called right after didChangeDependencies(). It’s called every single time the UI needs to render and returns a widget’s tree.
* didUpdateWidget() — it’s called when the parent widget changes and needs to redraw its UI
* deactivate() — called before dispose(), when this object is removed from the tree
* dispose() — called when deallocating the whole widget and its state.

What is the difference between hot reload, hot restart, and full restart?

Flutter’s hot reload feature helps you quickly and easily experiment, build UIs, add features, and fix bugs. Hot reload works by injecting updated source code files into the running Dart Virtual Machine (VM). After the VM updates classes with the new versions of fields and functions, the Flutter framework automatically rebuilds the widget tree, allowing you to quickly view the effects of your changes.

* **Hot reload** loads code changes into the VM and re-builds the widget tree, preserving the app state; it doesn’t rerun main() or initState(). Hot reload takes less time then Hot restart.
* **Hot restart** loads code changes into the VM, and restarts the Flutter app, losing the app state. Hot restart takes much higher time than Hot reload.
* **Full restart** restarts the iOS, Android, or web app. This takes longer because it also recompiles the Java / Kotlin / ObjC / Swift code. On the web, it also restarts the Dart Development Compiler.

What are the different ways to create navigation?

There are three ways to create navigation in Flutter

* **Direct navigation** with MaterialPageRoute

Route route = MaterialPageRoute(builder: (context) => NextScreen());  
Navigator.push(context, route);

* **Static navigation** with a map route — where you create a routes map in the MaterialApp and then push it by using its name

*Creating a routes map in the MaterialApp*

void main() {  
 runApp(  
 MaterialApp(  
 initialRoute: '/home',  
 routes: <String, WidgetBuilder>{  
 '/home': (context) => HomePage(),  
 '/second': (context) => SecondHome(),  
 },  
 )  
 );  
}

*Pushing it by using its name*

Navigator.pushNamed(context, '/second');

* **Dynamic navigation** which generates routes — i.e., implementing onGenerateRoute callback in the MaterialApp class

MaterialApp(  
 home: SecondPage(),  
 onGenerateRoute: (RouteSettings settings) {  
 switch (settings.name) {  
 case '/home':  
 return MaterialPageRoute(builder: (context)=> HomePage());  
 break;  
 case '/second':  
 return MaterialPageRoute(builder: (context)=> SecondPage());  
 break;  
 }  
 },  
),

Flutter’s build modes

The Flutter tooling supports three modes

* Use **debug mode** during development, when you want to use hot reload.
* Use **profile mode** when you want to analyze performance.
* Use **release mode** when you are ready to release your app.

Debug Mode

* Hot reload works only in debug mode.
* The emulator and simulator execute only in debug mode.
* Application performance can be janky in debug mode.
* Assertions are enabled.
* Service extensions are enabled.
* The compilation is optimized for fast development and run cycles (but not for execution speed, binary size, or deployment).
* Debugging is enabled, and tools supporting source-level debugging (such as DevTools) can connect to the process.

**Profile mode**

* In profile mode, some debugging ability is maintained — enough to profile your app’s performance.
* Profile mode is disabled on the emulator and simulator because their behaviour is not representative of real performance.
* On mobile, profile mode is similar to release mode.
* Some service extensions, such as the one that enables the performance overlay, are enabled.
* Tracing is enabled, and tools supporting source-level debugging (such as DevTools) can connect to the process.

Release Mode

* Use release mode for deploying the app, when you want maximum
* optimization and minimal footprint size.
* Assertions are disabled.
* Debugging information is stripped out.
* Debugging is disabled.
* The compilation is optimized for fast startup, fast execution, and small package sizes.
* Service extensions are disabled.

Parameter types in Dart Function

A function can have two types of parameters: *required* and *optional*.

* Dart’s optional parameters are *optional* in that the caller isn’t required to specify a value for the parameter when calling the function.
* Optional parameters can be either named or positional, but not both.
* Optional parameters can only be declared after any required parameters.
* Optional parameters can have a default value, which is used when a caller does not specify a value.
* You can get better clarity from [here](https://stackoverflow.com/questions/13264230/what-is-the-difference-between-named-and-positional-parameters-in-dart#:~:text=Dart%20has%20two%20types%20of%20optional%20parameters%3A%20named%20and%20positional.&text=Dart's%20optional%20parameters%20are%20optional,declared%20after%20any%20required%20parameters).

**Future class in dart**

A Future is used to represent a potential value, or error, that will be available at some time in the future.

Receivers of a Future can register callbacks that handle the value or error once it is available.

What is a future?

A future is an instance of the Future class. A future represents the result of an asynchronous operation. It can have two states: *uncompleted*or *completed*.

* **Uncompleted** — When you call an asynchronous function, it returns an uncompleted future. That future is waiting for the function’s asynchronous operation to finish or to throw an error.
* **Completed**— If the asynchronous operation succeeds, the future completes with a value. Otherwise, it completes with an error.

Synchronous programming

In synchronous programming, the code will execute line by line.

* *Synchronous operation* — A synchronous operation blocks other operations from executing until it completes.
* *Synchronous function* — A synchronous function only performs synchronous operations.

Asynchronous programming

Asynchronous operations let your program complete work while waiting for another operation to finish.

* *Asynchronous operation*— Once initiated, an asynchronous operation allows other operations to execute before it completes.
* *Asynchronous function* — An asynchronous function performs at least one asynchronous operation and can also perform synchronous operations.

An async function runs synchronously until the first await keyword. This means that within an async function body, all synchronous code before the first await keyword executes immediately.

**What is async and await?**

The async and await keywords provide a declarative way to define asynchronous functions and use their results.

* **async**— You can use the async keyword before a function’s body to mark it as asynchronous.
* **await**— You can use the await keyword to get the completed result of an asynchronous expression. The await keyword only works within an async function.

asnyc vs async\*?

* **async** gives you a Future — You add the async keyword to a function that does some work that might take a long time. It returns the result wrapped in a Future.
* **async\*** gives you a Stream - You add the async\* keyword to make a function that returns a bunch of future values one at a time. The results are wrapped in a Stream.

Null-aware Operators in Dart

Null-aware operators in dart allow you to make computations based on whether or not a value is null. It’s shorthand for longer expressions. You can get better clarity from [here](https://medium.com/@thinkdigitalsoftware/null-aware-operators-in-dart-53ffb8ae80bb).

* **??**

//assign y to x, unless y is null. Otherwise, assign z.  
x = y ?? z

* **??=**

//assign y to x if x is null.  
x ??= y

* **?.**

//call foo() only if x is not null.  
x?.foo()

Spread operator

you can use the spread operator (...) to insert all the values of a list into another list.

var list = [1, 2, 3];  
var list2 = [0, ...list];  
assert(list2.length == 4);

**Null-aware spread operator**

If the expression to the right of the spread operator might be null, you can avoid exceptions by using a null-aware spread operator (...?).

var list;  
var list2 = [0, ...?list];  
assert(list2.length == 1);

**Collection if**

you can use to build collections using conditionals.

var nav = [  
 'Home',  
 'Furniture',  
 'Plants',  
 if (promoActive) 'Outlet'  
];

**Collection for**

you can use to build collections using repetition.

var listOfInts = [1, 2, 3];  
var listOfStrings = [  
 '#0',  
 for (var i in listOfInts) '#$i'  
];

dynamic — var — final

* **dynamic**— can change the type of the variable and can change the value of the variable.
* **var**— can’t change the type of the variable, but can change the value of the variable.
* **final**— can’t change the type of the variable and can’t change the value of the variable later in code.

dynamic v = 123;// v is of type int.  
v = 456; // changing value of v from 123 to 456.  
v = 'abc'; // changing type of v from int to String.var v = 123; // v is of type int.  
v = 456; // changing value of v from 123 to 456.  
v = 'abc'; // ERROR: can't change type of v from int to String.final v = 123; // v is of type int.  
v = 456; // ERROR: can't change value of v from 123 to 456.  
v = 'abc'; // ERROR: can't change type of v from int to String.

Advantages of flutter

* Cross-platform Development
* Faster Development
* Good Community
* Live and Hot Reloading
* Native Performance
* Provides Native Look and Feel

Thanks…

I’m looking for a job change

I am a Software Engineer having experience in mobile app development with expertise in Java, Android and Flutter. Interested people can contact me through [LinkedIn](https://www.linkedin.com/in/kathir-i/).