[**Flutter Top Questions & Answers**](http://interview.skyrites.com/2020/02/21/flutter-top-questions-answers/)

[](http://interview.skyrites.com/2020/02/21/flutter-top-questions-answers/)

**1. What is Flutter?**

Flutter is a open-source cross-platform mobile app development framework which, as Google defines it, “allows you to **build beautiful native apps on iOS and Android from a single codebase**”. First alpha version was released on May 2017; and after 1.5 years, Flutter got its first stable 1.0 release on December 4th, 2018.

In addition, to supporting for Android and iOS platforms, Flutter will be the main framework for developing applications for Google’s upcoming operating system, **Fuchsia**. It is currently under development and will possibly and gradually replace Android.

**2. What is Dart? And why?**

**Dart** is the object-oriented, garbage-collected programming language that you use to develop Flutter apps. It was also created by Google, but is open-source, so it has community inside and outside Google.

Other than its Google origins, Dart was chosen as the language of Flutter for the following reason: It’s one of very few languages that can be compiled both AOT (ahead-of-time*)* and JIT (just-in-time).

* **JIT compilation** is used during app development process since it enables hot reloads (about which I will talk more in next questions) and fast development cycles by compiling code at runtime on the fly.
* **AOT compilation** is used when you are done with development and ready for release. Code is then compiled AOT to native code, enabling fast startup and performant execution of the app.

Personally, what I’ve experienced about Dart is that it takes only couple of days to get used to it if you are a developer who is already familiar with Java or similar languages. So, if you are an Android developer for instance, the learning curve for this language should be very low for you.

With its clean but flexible syntax, Dart can be identified as a language that includes only the most wanted features of any high-level programming language exists.

**3. How is Flutter native?**

Flutter uses a graphics engine called **Skia** to perform all UI rendering on the application side. It means that it doesn’t depend on OEM widgets provided by the platform. It just needs a platform’s canvas to draw its own renderings. This ensures predictability and developer’s full control over the widgets and layout.

In addition to that, Flutter keeps its structure as a tree of widgets. By the way, **almost everything in Flutter is a widget**, which enables you to build your app in the structure of widgets inside widgets. This internal tree stucture allows Skia to render only the widgets that need to be updated, and retrieve unchanged or even moved widgets from cache.

**4. What is different from React Native?**

React Native utilizes a JavaScript bridge to convert its widgets to OEM widgets. And since it does this conversion (compare & update cycle) constantly, it creates a bottleneck and causes a drop in performance as a result.

While still using the advantages of reactive views, on the contrary, Flutter does not use such bridge to convert its own widgets to OEM widgets as shown in the answer of previous question. Besides fast & smooth UI performance and predictability, as a further advantage of this, what you see on an Android KitKat device will be the same as what you get on Android Pie. This compatibility is obvious because Flutter doesn’t use OEM widgets and isn’t affected by UI/UX changes between different Android version.

**5. No layout preview… is this a problem?**

Not at all!.. Flutter has a feature called **“hot reload”** which enables you to hot-swap code changes while app is running. It takes subsecond to see the effects of your changes on screen. It’s definitely fast, but also smart since it preserves app’s runtime state between hot reloads. As a result, you get something better and more useful than layout previews.

**6. What are Flutter widgets?**

A Flutter app is considered as a**tree of widgets**. In the widget tree given above, every node you see is a widget. Even to center a widget on screen, you wrap your widget with *Center*, which is also a widget itself. So, you build your entire UI by composing smaller widgets to create more complex widgets, even pages, and even hierarchy of pages. This is called **“composition over inheritance”** principle in object-oriented programming. On the contrary, in Android development for instance, you create your custom views by extending a View type of class.

There are two types of widgets in Flutter:

* **StatelessWidget** — immutable, meaning that its properties cannot change and should be final.
* **StatefulWidget** — maintains state that might change during lifetime of the widget. To change the state of a StatefulWidget, *setState()* should be called. Widget is then redrawn.

**7. How various are the widgets supported for both platforms?**

The widget set provided by Flutter is extensive, especially for **Material components**. Based on my hands-on experince, using complex components such as drawers, bottom navigation bars, or tab bars is much easier in Flutter compared to Android development. While you are wandering around the widgets of Flutter, you can encounter very interesting ones such as *UserAccountsDrawerHeader*. Yes, even this comes as a ready-to-use UI component with Flutter.

For **Cupertino (iOS-style) widgets**, I can’t say the same extensiveness applies to them. There are enough of them to cover most of the iOS widgets, but not as detailed as Material components. It’s actually not surprising when you consider that Flutter is a Google product. However, I cannot say that this is a serious issue because the widget set is so customizable that you can get whatever look you want by playing with parameters of widgets and placing almost any one of them inside any other.

***8)* What is an App state?**

* State that is not *ephemeral*, that you want to share across many parts of your app, and that you want to keep between user sessions, is what we call **application state** (sometimes also called *shared state*).
* Examples of application state:
  + User preferences
  + Login info
  + Notifications in a social networking app
  + The shopping cart in an e-commerce app
  + Read/unread state of articles in a news app

**9) What are the different build modes in Flutter?**

* The Flutter tooling supports three modes when compiling your app, and a headless mode for testing.
* You choose a compilation mode depending on where you are in the development cycle.
* The modes are:
  + Debug
  + Profile
  + Release

**10) Differentiate between Hot Restart and Hot Reload?**

**Hot Reload**

* Flutter hot reload features works with combination of **Small r key** on command prompt or Terminal.
* Hot reload feature quickly compile the newly added code in our file and sent the code to Dart Virtual Machine. After done updating the Code Dart Virtual Machine update the app *UI* with widgets.
* Hot Reload takes less time then Hot restart.
* There is also a draw back in Hot Reload, If you are using *States* in your application then Hot Reload preservers the *States* so they will not update on Hot Reload our set to their default values.

**Hot Restart**

* Hot restart is much different than hot reload.
* In Hot restart it destroys the preserves *State* value and set them to their default. So if you are using *States* value in your application then after every hot restart the developer gets fully compiled application and all the states will be set to their defaults.
* The app widget tree is completely rebuilt with new typed code.
* Hot Restart takes much higher time than Hot reload.