Flutter Interview Questions

1. What is *Dart*?

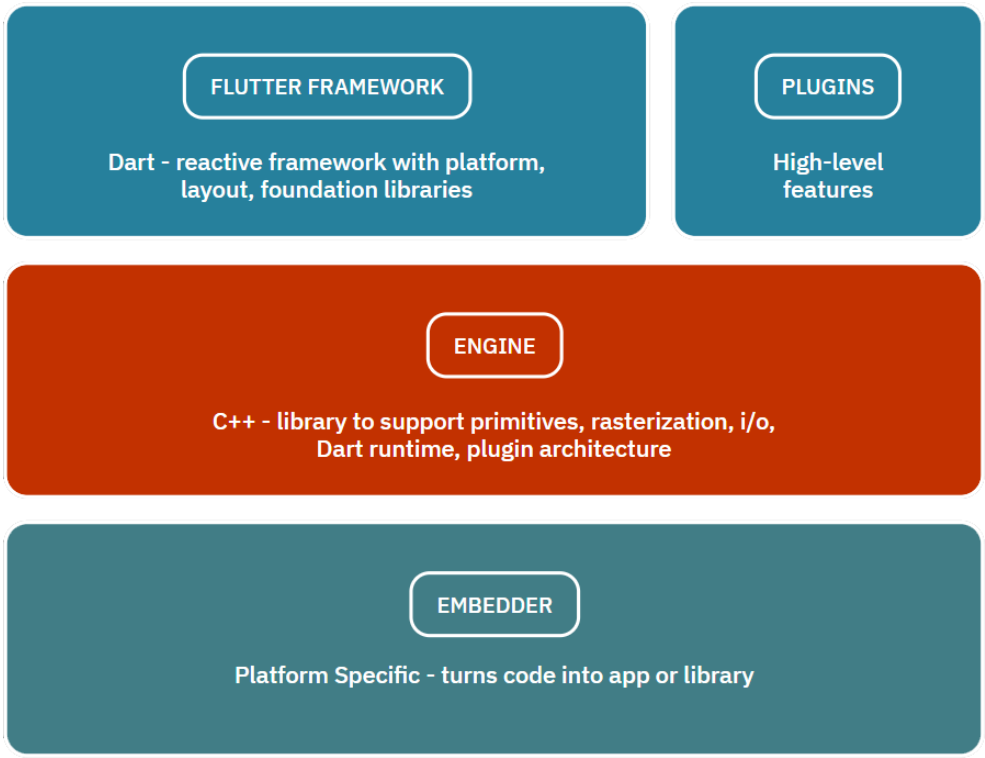
a. Dart is a general-purpose, object-oriented programming language with C-style syntax. It is open-source and developed by Google in 2011. The Dart language can be compiled both AOT (Ahead-of-Time) and JIT (Just-in-Time.

2. What is *Flutter*?

a. Flutter is a framework developed by Google that allows you to build natively compiled mobile applications with one programming language and a single codebase. Flutter is not a language; it is an SDK. Flutter apps use Dart programming language for creating an app. Flutter is mainly optimized for 2D mobile apps that can run on both Android and iOS platforms.

3. What is the *Architecture* of Flutter?

a. Flutter has a modular, layered architecture. This allows you to write your application logic once and have consistent behavior across platforms, even though the underlying engine code differs depending on the platform. The layered architecture also exposes different points for customization and overriding, as necessary.



4. **What are the different *Build modes* in Flutter?**

a. Build mode is chosen depending on where we are in the **development cycle:**

Debug mode sets up the app for debugging on the physical device, emulator, or simulator.

Profile mode is used to measure the **performance** of our applications. In this mode, some debugging ability is maintained to profile your app's performance. This mode is **disabled** on the emulator and simulator because they are not representative of real performance.

Release mode allows us to **optimize** the codes and generate them without any debug data in a fully optimized form. In this mode, many of the application's code will be entirely removed or rewritten. We use this mode when we are ready to release the app. It enables maximum optimization and minimal footprint size of the application.

5. **What types of *Tests* can you perform in Flutter?**

a. ***Unit Tests:*** It tests a single function, method, or class. Its goal is to ensure the correctness of code under a variety of conditions. This testing is used for checking the validity of business logic.

Widget Tests: It tests a single widget. Its goal is to ensure that the widget's UI looks and interacts with other widgets as expected.

Integration Tests: It validates a complete app or a large part of the app. Its goal is to ensure that all the widgets and services work together as expected.

6. **What is the difference between *AOT* & *JIT* in Flutter?**

a. In development mode, Flutter is compiled ***just-in-time***. That is why we can do hot-reload/restart so fast. In release mode, your code is compiled **ahead-of-time**, to native code. It is for better performance, minimizes size and removes other stuff that are useful only in dev mode.

7. **How does Dart *AOT* work?**

a. Dart source code will be translated into assembly files, then assembly files will be compiled into binary code for different architectures by the assembler.

For mobile applications the source code is compiled for multiple processors ARM, ARM64, x64 and for both platforms - Android and iOS. This means there are multiple resulting binary files for each supported processor and platform combination.

8. **What's the difference between *Hot Reload* and *Hot Restart*?**

a. **Hot reload** maintains the app state while updating the UI almost instantaneously. **Hot restart**, by comparison, takes a little longer because it resets the app state to its initial conditions before updating the UI. Both are faster than doing a **Full restart**, which requires recompiling the app.

9. Why does a Flutter app usually takes *longer* to execute when running for the *first time*?

a. The first time you build a Flutter application, it takes much longer than usual since Flutter creates a device-specific IPA or APK file. XCode and Gradle are used in this process to build a file, which usually takes a lot of time.

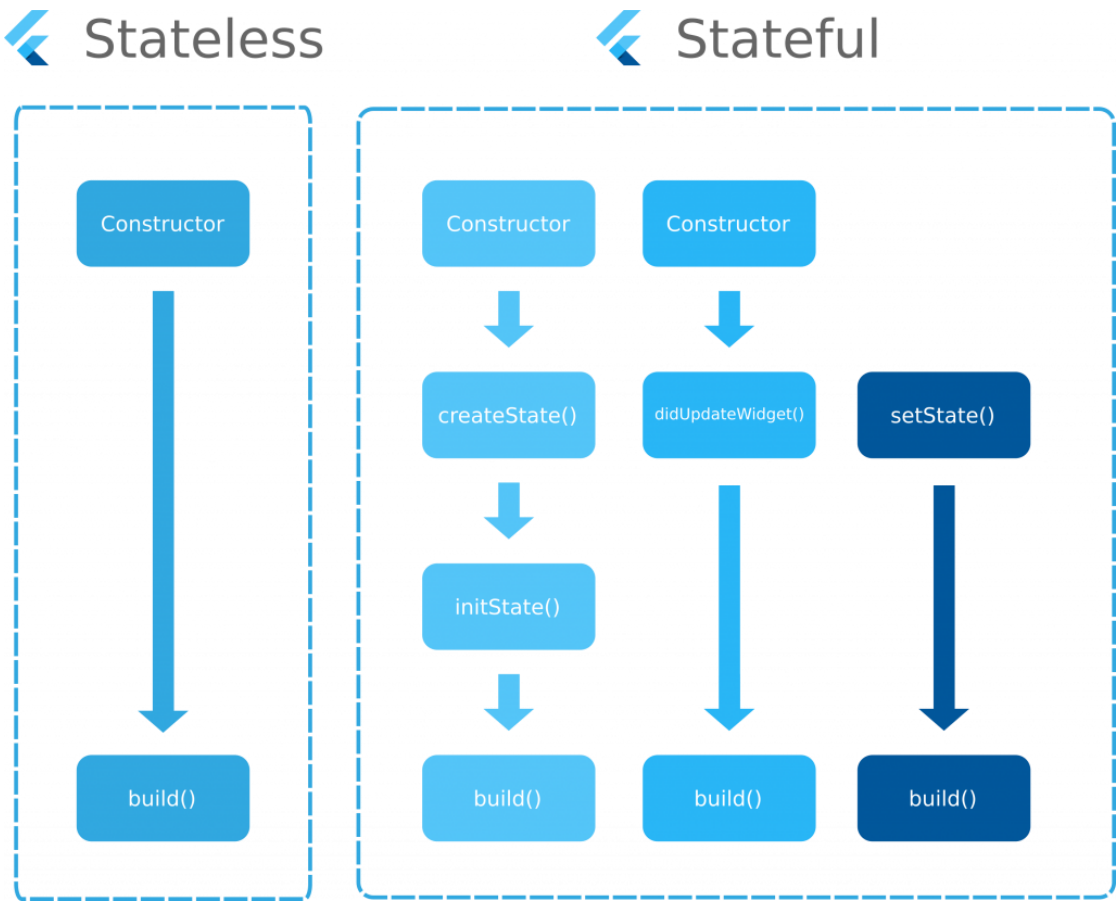
10. What is a *Widget*? and how many types of widgets are there in Flutter?

a. Widgets are high level objects used to describe any part of an application. It can be but is not limited to UI elements, layout (alignment, padding, ...), data (theme, configurations, …). In flutter everything is a widget, almost everything you'll code will be inside a Widget. There are **2** types of widgets in Flutter, **Stateless** & **Stateful** widgets.

11. What is the difference between *Stateless* & *Stateful* widgets?

a. `StatelessWidget` is an immutable class that acts as a blueprint for some part of the UI layout. You use it when the widget doesn't change while displaying and, therefore, has no State.

`StatefulWidget` is also immutable, but it's coupled with a ***State Object*** that allows you to rebuild the widget with new values whenever calling `setState()`.



12. What is the difference between *WidgetsApp* and *MaterialApp*?

a. `WidgetsApp` provides basic navigation. Together with the ***widgets library***, it includes many of the foundational widgets that Flutter uses.

   `MaterialApp` and the corresponding ***material library*** is a layer built on top of WidgetsApp and the widgets library. It implements *Material Design*, which gives the app a unified look and feel on any platform or device. You can also use `CupertinoApp` to make iOS users feel at home, or you can even build your own set of custom widgets to fit your brand.

13. Can you *nest* a Scaffold? Why or why not?

a. Yes, you can nest a Scaffold. Scaffold is just a widget, so you can put it anywhere a widget might go. By nesting a Scaffold, you can layer drawers, snack bars and bottom sheets.

14. What is *async* & *await* in Flutter?

a. Until the async method is finished, await interrupts the process flow. Await generally means: Wait here until this function is finished so that you can get its return value. It can only be used with async.

15. What is the difference between a *Future* and a *Stream*?

a. A Stream is a combination of Futures. Future has only one response, but Stream could have any number of Responses.

16. What is a *Stream*?

a. It is a sequence of asynchronous data. We can create a stream using async generator (`async*`) and listen to it using `listen()`. There are 2 types of streams, **Single Subscription** and **Broadcast** streams.

17. What are the different *types* of *Streams*?

a. ***Single Subscription Streams*:** These streams deliver events sequentially. They are considered as sequences contained within a larger whole. These streams are used when the order in which events are received matters, such as reading a file. There can be only one listener throughout the sequence, and without a listener, the event won't be triggered.

   ***Broadcast Streams*:** These streams deliver events to their subscribers. Upon subscribing to events, subscribers are immediately able to start listening to them. These are versatile streams that allow several listeners to listen simultaneously. Furthermore, one can listen again even after canceling a previous subscription.

18. What is `BuildContext` and how is it useful?

a. It is actually the widget's element in the Element tree, so every widget has its own **`BuildContext`.** You usually use it to get a reference to the theme or to another widget. For example, if you want to show a material dialog, you need a reference to the scaffold. You can get it with `Scaffold.of(context)`, where `context` is the build context. `of()` searches up the tree until it finds the nearest scaffold.

19. What is the difference between `main()` and `runApp()`?

a. `main()` is needed for every Dart program since it's the entry point for the app. But in Flutter apps, you should also call `runApp()` to load the framework & attach the given widget to screen.

20. What is `dispose()` method in flutter?

a. It is used to release the memory allocated to variables when state object is removed from the tree permanently. After the framework calls dispose, the State object is considered unmounted, and the mounted property is set to false. It is an error to call `setState()` at this point. This stage of the lifecycle is terminal: there is no way to remount a ***State object*** that has been disposed.

21. What are the different *types* of *Image* widgets in flutter?

a. `Image()`**:** for obtaining an image from an ***ImageProvider***.
`Image.asset()`**:** for obtaining an image from an ***AssetBundle*** using a key.
`Image.network()`**:** for obtaining an image from a ***URL***.
`Image.file()`**:** for obtaining an image from a ***File***.
`Image.memory()`**:** for obtaining an image from a ***Uint8List***.

22. What is the difference between *Required* and *Optional* parameters?

a. ***Required*** parameters are the arguments essential for the function to complete its code block.

While ***Optional*** parameters doesn't have to be specified by the caller while calling the function. Optional parameters can only be declared after required parameters. It can be ***Positional [ ]*** or ***Named { }***, it can also have a default value, which is used when a caller does not specify a value.

23. What is *MediaQuery*?

a. It provides a higher-level view of the current app's screen size and can also give more detailed information about the device and its layout preferences.

24. Why do we need *Mixins*?

a. Dart does not support ***multiple inheritances***. Thus, to implement multiple inheritances in Flutter/Dart, we need mixins. ***Mixins*** provide a way to write the reusable class's code in multiple class hierarchies.

25. What are *Keys* in Flutter, and when to use it?

a. Keys in Flutter are used as an identifier for **Widgets**, **Elements** and **SemanticsNodes**. We can use it when a new widget tries to update an existing element; then, its key should be the same as the current widget key associated with the element. Keys should not be different amongst the Elements within the same parent.

   The subclasses of Key must be a ***GlobalKey*** or ***LocalKey***.

   Keys are useful when we try to manipulate (such as adding, removing, or reordering) a collection of widgets of the same type that hold some state.

26. What is an *event loop*, and how is it related to *isolates*?

a. Dart code runs on a single thread called an isolate. Separate isolates don't share any memory and they only communicate through messages sent over ports.

   Every isolate has an event loop, which schedules asynchronous tasks to run. The tasks can be on one of two different queues: the ***microtask queue*** or the ***event queue***. Microtasks always run first, but they are mainly internal tasks that the developer doesn't need to worry about. Calling a future puts the task on the event queue when the future completes.

   A lot of new Dart programmers think async methods run on a separate thread. Although that may be true for I/O operations that the system handles, it isn't the case for your own code. That's why if you have an expensive computation, you need to run it on a separate isolate.

27. What is *Tween animation*?

a. The shortened version of in-between animation is tween animation. The start and endpoints of an animation must be specified in tween animation. Using this method, the animation can begin at the beginning and can progress through a series of values until it reaches the endpoint. Transition speed and duration are also determined by using the tween animation.

28. What is the use of *Ticker* in Flutter?

a. We use a ticker to tell how often our animation is refreshed in Flutter. Signals are sent at a constant frequency, such as 60 times per second, using this type of signal-sending class. We understand it better with our watch, which ticks constantly. For each tick, a callback method is provided that has the time since the first tick at each second since it was started. The tickers are synchronized immediately, even if they begin at different times.

29. What is meant by *Null-aware* operators?

a. Null-aware operators allow you to make computations based on whether or not a value is **null**. Dart provides some useful information to handle the null values.

- **The "??=" assignment operator:** It assigns a value to a variable only if it is null.
- **The "??" null-aware operator:** Expression 1 is checked first for nullness, and if it is not null, its value is returned; otherwise, expression 2 is evaluated, and its value is returned.
- **The "?." Safe Navigation Operator:** It is also known as the Elvis operator. It is possible to use the ?. operator when calling a method/getter on an object, as long as the object isn't null (otherwise, the method will return null).

30. What is state management?

a. ***Ephemeral State*:** Ephemeral state is also called UI state or local state, and it pertains to a particular widget. In other words, it is a state that is contained within the specific widget. By means of `StatefulWidget`, Flutter provides support for this state.

***App State*:** It is the state that we intend to share across different parts of the app and which we want to maintain between sessions. These types of states can thus be used globally.

31. What is the difference between *Packages* and *Plugins*?

a. A **Package** contains only Dart code, while a **Plugin** contains both Dart & Native code, and it's used to access native features. Usually on `pub.dev`, both packages and plugins are referred to as packages and only while creating a new package is the distinction clearly mentioned.

32. When is it appropriate to use *packages*, *plugins*, or *third-party* dependencies?

a. Packages and plugins are great for saving you time and work. There's no need to solve a complex problem yourself when someone has already done it, especially if the solution is highly rated.

On the other hand, there's also a danger of being too reliant on third party packages. They can break, have bugs, or even be abandoned. When you need to switch to a new package down the road, you might have to make huge changes to your code.

33. How do you talk to *Native* code from within a Flutter app?

a. Normally you don't need to talk to native code because the Flutter framework or third-party plugins handle it. However, if you do find yourself needing to get special access to the underlying platform, you can use platform channels. One type of platform channel is a ***Method channel***. Data is serialized on the Dart side and then sent to the native side. You can write native code to interact with the platform before sending a serialized message back. That message might be written in Java or Kotlin on Android or Objective-C or Swift on iOS. The second type of platform channel is the ***Event channel***, which you use to send a stream of data from the native platform back to Flutter. This is useful for monitoring sensor data.

34. How does *Navigation* work in Flutter?

a. Flutter has an **imperative** routing mechanism, the ***Navigator*** widget, and a more idiomatic **declarative** routing mechanism, the ***Router*** widget. The two systems can be used together (indeed, the declarative system is built using the imperative system).

   Typically, small applications are served well by just using the Navigator API, via the **MaterialApp** constructor's `MaterialApp.routes` property.

   More elaborate applications are usually better served by the Router API, via the `MaterialApp.router` constructor. This requires some more up-front work to describe how to parse deep links for your application and how to map the application state to the set of active pages but is more expressive in the long run.

35. How to *prevent* unnecessary widget *rebuilds*?

a. 1) Refactor a large widget tree into smaller individual widgets, each with its own build method.

   2) When possible, use `const` constructor to tell Flutter that it doesn't need to rebuild the widget.

   3) Keep the subtree of a stateful widget as small as possible. If a stateful widget needs to have a subtree under it, create a custom widget for the stateful widget and give it a child parameter.

36. What is the `pubspec.yaml` file?

a. It's a file that is included when you create a Flutter project and is located at the top of the project tree. It makes sure that the next time you build the project, you will get the same package version. Additionally, you can set constraints for the app. During working with the Flutter project, this configuration file of the project will be required a lot. This specification is written in YAML, which can be read by humans. ***It includes the following*:**

   - General project settings, like name of the project, version, description, etc.
   - Dependencies within a project.
   - The assets of the project (e.g., images, audio, etc.)

37. What are different types of *Dependencies*?

a. ***dependencies*:** these are packages that anyone using your package will also need.

   ***dev_dependencies*:** they differ from regular dependencies in that dev dependencies of packages you depend on are ignored. They are only used for tests, examples, tools, etc.