**1)What is Flutter?**Flutter is an open-source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.  
Flutter is not a language; it is an SDK. Flutter apps use the Dart programming language for creating an app. The first alpha version of Flutter was released in May 2017.

**2) What is Dart?**It is open-source and developed by Google in 2011. The purpose of Dart programming is to create frontend user interfaces for the web and mobile apps.  
Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development.

**3)What are the Flutter widgets?**Flutter widgets are built using a modern framework that takes inspiration from [React](https://reactjs.org/). The central idea is that you build your UI out of widgets. Widgets describe what their view should look like given their current configuration and state. When a widget’s state changes, the widget rebuilds its description, which the framework diffs against the previous description to determine the minimal changes needed in the underlying render tree to transition from one state to the next.

**4) What is the Difference Between Stateless and Stateful Widget in Flutter?**

*A stateless widget is useful when the part of the user interface you are describing does not depend on anything other than the configuration information and the BuildContext whereas a Stateful widget is useful when the part of the user interface you are describing can change dynamically.*

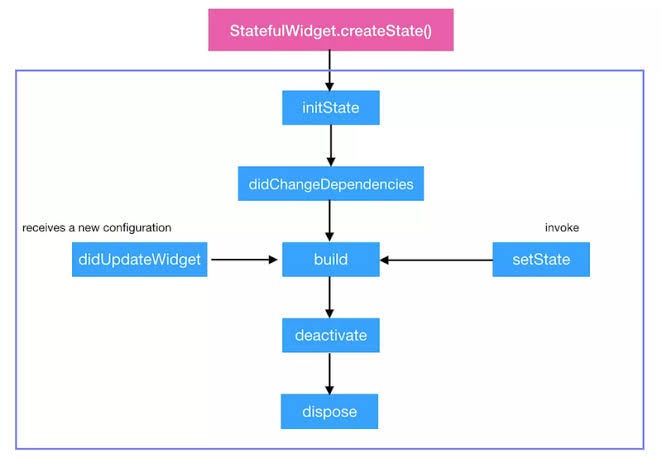
A Stateful widget is a mutable widget. That is the reason it can be drawn multiple times within its lifetime. It is referred to as dynamic because it can change the inner data during the widget’s lifetime. A widget that allows us to refresh the screen is called a Stateful widget. This widget has a createState() method, which returns a class that extends the Flutter State Class. Examples of Stateful widgets are Checkbox, Radio, Slider, InkWell, Form, and TextField.

A Stateless widget will never rebuild by itself but can do so from external events. The Stateless widget does not have any state information. It remains static throughout its lifecycle. Examples of the Stateless widget are Text, Row, Column, Container, etc. If the screen or widget contains static content, it should be a Stateless widget. However, if you want to change its content, it needs to be a Stateful widget.

**5) What is StatefulWidget LifeCycle?**

The lifecycle has the following simplified steps:

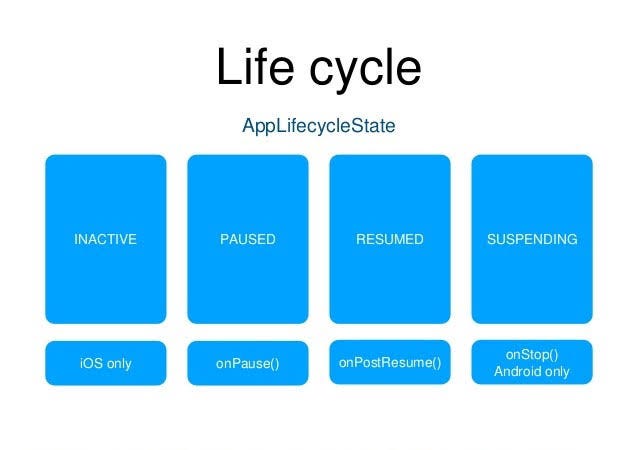
* **createState()**: When we build a new StatefulWidget, this one calls createState() right away and this override method must exist.
* **initState()**: It is the first method called after the Widget is created.This is our equivalent to onCreate() and viewDidLoad()
* **didChangeDependencies()** : This method is called immediately after initState() on the first time the widget is built.
* **build()** : This iscalled right after didChangeDependencies(). All the GUI is rendered here and will be called every single time the UI needs to be rendered.
* **didUpdateWidget()**: It’ll be called once the parent Widget did a change and needs to redraw the UI.
* **deactivate()**: Framework calls this method whenever it removes this State object from the tree
* **dispose()**: It is called when this object and its State are removed from the tree permanently and will never build again.



**6) What is AppLifecycleState?**

AppLifecycleState is as follows:

* **inactive** — The application is in an inactive state and is not receiving user input. (iOS only).
* **paused** — The application is not currently visible to the user, not responding to user input, and running in the background.
* **resumed** — The application is visible and responding to user input.
* **suspending** — The application will be suspended momentarily. (Android only).



**7) What is pubspec.yaml file?**It is responsible for handling importing images/fonts/third-party packages which you want to include in your project.  
**The pubspec. yaml file** is used to define the dependencies of your Flutter project. This metadata information is **written in the YAML language**. This file can have the following fields name, version, description, homepage, repository, documentation, dependencies, environment, and more about the **pubspec.yaml file.**

**8) what is the difference between a flutter package and the flutter plugin.**A “package” contains only Dart code.  
A “plugin” contains both Dart and Native code (kotlin/js/swift/…)

**Flutter plugins:***(Native-related developments).*Flutter plugin is the wrapper of the native code like android( Kotlin or java) and iOS(swift or objective c)  
Flutter can do anything that a native application can through the use of Platform Channels and Message Passing. Flutter instructs the native iOS/Android code to act and returns the result to Dart.

**Flutter packages or modules:**(*Make the development faster by using code from util libraries).*Flutter supports using shared packages contributed by other developers to the Flutter and Dart ecosystems. This allows for quickly building an app without having to develop everything from scratch.

*A package can use plugins if it wants to. It will still qualify as a package.*

**9) What is the difference between WidgetsApp and MaterialApp?**

**WidgetsApp**:- A convenience class that wraps several widgets that are commonly required for an application.  
One of the primary roles that WidgetsApp provides is binding the system back button to pop the Navigator or quitting the application.

**MaterialApp**:- A convenience widget that wraps several widgets that are commonly required for material design applications.  
It builds upon a WidgetsApp by adding material-design specific functionality, such as AnimatedTheme and GridPaper.

**10) What’s the difference between hot reload and hot restart?**

In simple words,  
Hot Reload is just updating the changes in your program.  
But Hot Restart will again remove your previous state and run the complete program.

**Hot Reload**

* Flutter hot reload features works with a combination of the Small r key on the command prompt or Terminal.
* The 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 than Hot restart.
* There is also a drawback 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**

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

**11) Can you nest a Scaffold? Why or why not?**

Yes, you can nest a Scaffold. That’s the beauty of Flutter. You control the entire UI.  
Scaffold is just a widget, so you can put it anywhere a widget might go. By nesting a, you can layer drawers, snack bars, and bottom sheets.

**12) Describe Some of the major features of Flutter.**

Some of the major features of using Flutter are,

**Fast Development**— With the use of a rich set of fully customizable widgets, you can build native interfaces in minutes with Flutter.

**Expressive and Flexible UI**— The layered architecture present with the Flutter enables you to fully customize your UI. This results in fast rendering and expressive designs.

**Native Performance**— The widgets present in the Flutter incorporate all the critical platform differences such as scrolling, navigation, icons, and more. It gives a full native performance on all platforms

**13) How do you reduce widget rebuild?**You rebuild widgets when the state changes. This is normal and desirable because it allows the user to see the state changes reflected in the UI. However, rebuilding parts of the UI that don’t need to change is wasteful.

There are several things you can do to reduce unnecessary widget rebuilding.

* The first is to refactor a large widget tree into smaller individual widgets, each with its build method.
* Whenever possible, use the const constructor, because this will tell Flutter that it doesn't need to rebuild the widget.
* Keep the subtree of a stateful widget as small as possible. If a stateful widget needs to have a widget subtree under it, create a custom widget for the stateful widget and give it a child parameter.

**14) What is BuildContext and how is it useful?**BuildContext is the widget's element in the Element tree — so every widget has its own BuildContext.

You usually use BuildContext to get a reference to the theme or 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.

For more information click [here](https://shirsh94.medium.com/what-is-buildcontext-in-flutter-af8890c14ee3).

**15) How do you talk to native code from within a Flutter app?**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.

You don’t use platform channels on the web, however, because they’re an unnecessary step.

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.

**16) What types of tests can you perform?**There are three main kinds of tests: **unit tests**, **widget tests,** and **integration tests**.  
**Unit tests** are all about checking the validity of your business logic.  
**Widget tests** are for making sure UI widgets have the components that you expect them.  
**Integration** tests check that your app is working as a whole.

For more information click [here](https://shirsh94.medium.com/testing-your-flutter-app-for-smooth-sailing-here-are-some-tips-291ab742826b).

**17) What are the pros and cons of different state management solutions?**While there are countless varieties, some of the more popular state management solutions include BLoC, ChangeNotifier with Provider, Redux, MobX, and RxDart. These are all appropriate for medium- to large-scale apps, if you’re only making a quick demo app, then a stateful widget is often enough.

Instead of listing the pros and cons of each state management option, it’s more useful to look at the situations where a certain class of solutions is a better fit. For example, for someone who’s overwhelmed with the sheer number of options, it’s important to choose a solution that’s easy to grasp, mentally. ChangeNotifier with Provider or MobX would be a good choice because it makes sense to directly call methods on the state class in response to events.

If you’re heavily reliant on streams, such as with a Firebase API, then it’s natural to choose a stream-based solution like BLoC or RxDart.

And if you need undo/redo functionality, then you’d want a solution like BLoC or Redux that handles immutable states well.

**18) What is Stream in Flutter?**A stream is a sequence of asynchronous events. It provides an asynchronous sequence of data. It is the same as a pipe where we put some value on one end, and if we have a listener on the other end, it will receive that value. We can keep multiple listeners in a stream, and all of those will receive the same value when put in the pipeline.

**19) How to create private variables in Dart?**In dart '**\_**' is used before the variable name to declare it as private. Unlike other programming languages, here private doesn't mean it is available only to the class it is in, private means it is **accessible in the file it is in** and **not accessible to other files**.

**20) What is the event loop, and what is its relationship to isolates?**In Flutter, the event loop is a central concept that helps manage the flow of control within an app. It is responsible for processing events and updating the app’s state in response to those events.

An isolate is a separate thread of execution that is isolated from the main thread of the app. Isolates are used in Flutter to allow concurrent execution of code, which can be useful for tasks that might take a long time to complete, such as network requests or computationally intensive operations.

The event loop is related to isolates in that it is responsible for coordinating the execution of code across multiple isolates, if they are used in the app. It does this by sending messages between isolates and scheduling the execution of code on the appropriate isolate.

Overall, the event loop plays a crucial role in the operation of a Flutter app, helping to manage the flow of control and coordinate the execution of code across multiple isolates.

**21) What is the tree shaking in Flutter?  
Tree shaking** is an optimization technique to remove the unused module in the bundle during the build process. It is a dead code elimination technique used to optimize the code.

**22) What are DevTools in Flutter?  
DevTools in Flutter** are a **set of tools** used for **performance management** and **debugging**. With these tools, you can inspect the UI layout, diagnose the UI performance issues, perform source-level debugging, view general log & diagnostics information, and more. This tool is still in preview release but you can test the alpha version of this tool by clicking the **“beaker”** icon in the upper-right corner of DevTools.

**23) What is The Flex widget in Flutter?**The Flex widget **allows you to control the axis along which the children are placed (horizontal or vertical)**. This is referred to as the main axis. If you know the main axis in advance, then consider using a Row (if it’s horizontal) or Column (if it’s vertical) instead, because that will be less verbose.

**24) What are the differences between expanded and flexible widgets?  
Flexible** is use to resize the widgets in *rows* and *columns*. It’s mainly used to adjust the space of the different child widgets while keeping the relation with their parent widgets.

Meanwhile, **Expanded** changes the constraints sent to the children of *rows* and *columns*; it helps to fill the available spaces there. Therefore, when you wrap your child in an Expanded widget it fills up the empty spaces.

**25) Material Vs Cupertino Widget?  
Cupertino**widgets are used to build an iOS-like app and MaterialApp is used to build an Android (Material) app.  
**Material** widgets implement the Material design language for iOS, Android, web, and desktop.  
**Cupertino** widgets implement the current iOS design language based on Apple’s Human Interface Guidelines.  
The **Material** Design language was created for any platform, not just Android. When you write a Material app in Flutter, it has the Material look and feels on all devices, even iOS. If you want your app to look like a standard iOS-styled app.  
you would use the Cupertino library You can technically run a Cupertino app on either Android or iOS, but (due to licensing issues) Cupertino won’t have the correct fonts on Android. For this reason, use an iOS-specific device when writing a Cupertino app.

**26) What is a Null Aware Operator?  
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.**A null-aware operator is a nice tool for making nullable types usable in Dart instead of throwing an error.**The **most common use of the Null aware operator** is when a developer wants to **parse JSON data from the server and after parsing JSON, the user can check whether the JSON is empty or not using the IF-Else condition.**

**27) What is Form, textfield and textFormField?**If you making a Form where you require to save, reset, or validate operations- use TextFormField. Else For Simple user input capture TextField is sufficient. TextFormField, which integrates with the Form widget.  
This is a convenience widget that wraps a TextField widget in a FormField. A Form ancestor is not required. The Form simply makes it easier to save, reset, or validate multiple fields at once. To use without a Form, pass a GlobalKey to the constructor and use GlobalKey.currentState to save or reset the form field.

**28) Difference between StreamBuilder and FutureBuilder?  
FutureBuilder** is used for one-time response, like taking an image from the Camera, getting data once from the native platform (like fetching device battery), getting file reference, making an HTTP request, etc.

On the other hand, **StreamBuilder** is used for fetching some data more than once, like listening for a location update, playing music, stopwatch, etc.

Both StreamBuilder and FutureBuilder have the same behavior: They listen to changes on their respective object. And trigger a new build when they are notified of a new value.  
Future is like Promise in JS or Task in c#. They are the representation of an asynchronous request. Futures have one and only one response. A common usage of Future is to handle HTTP calls. What you can listen to on a Future is its state. Whether it’s done, finished with success, or had an error. But that’s it.  
Stream on the other hand is like an async Iterator in JS. This can be assimilated into a value that can change over time. It usually is the representation of web sockets or events (such as clicks). By listening to a Stream you’ll get each new value and also if the Stream had an error or completed A Future can’t listen to a variable change. It’s a one-time response. Instead, you’ll need to use a Stream.

**29) What is the Future?**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.  
To perform asynchronous operations in Dart, you can use the Future class and the async and await keywords. A future (lower case “f”) is an instance of the Future (capitalized “F”) class.  
A future represents the result of an asynchronous operation and can have two states: uncompleted or completed.

**30)What is the Difference between synchronous operation and synchronous function also Difference between asynchronous operation and asynchronous function?**

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

**31) What is the difference between Method and function?**

A simple way to remember:

* **F**unction → **F**ree (Free means it can be anywhere, no need to be in an object or class)
* **M**ethod → **M**ember (A member of an object or class)

A **function** is a piece of code that is called by name. It can be passed data to operate on (i.e. the parameters) and can optionally return data (the return value). All data that is passed to a function is explicitly passed.

A **method** is a piece of code that is called by a name that is associated with an object. In most respects it is identical to a function except for two key differences:

1. A method is implicitly passed the object on which it was called.
2. A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class — the class is the definition, and the object is an instance of that data).

**32) What is mean by responsive and adaptive apps?  
Responsive:-** responsive app has had its layout tuned for the available screen size. Often this means (for example), re-laying out the UI if the user resizes the window, or changes the device’s orientation. This is especially necessary when the same app can run on a variety of devices, from a watch, phone, or tablet, to a laptop or desktop computer.

**Adaptive:-** Adapting the app to run on different device types, such as mobile and desktop, requires dealing with **mouse and keyboard input**, as well as **touch**input. It also means there are different expectations about the app’s visual density, how component selection works (cascading menus vs bottom sheets, for example), using platform-specific features (such as top-level windows), and more.

For more detail click [here](https://medium.com/@shirsh94/responsive-and-adaptive-apps-in-flutter-85171b922bc7).

**33) What is LayoutBuilder?**LayoutBuilder Widget is **similar to the Builder widget except that the framework calls the builder function at layout time and provides the parent widget’s constraints**. This is useful when the parent constrains the child’s size and doesn’t depend on the child’s intrinsic size.

**34) Difference between MediaQuery and LayoutBuilder?**

MediaQuery 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.  
LayoutBuilder helps to create a widget tree in the widget flutter which can depend on the size of the original widget.

*In simple words MediaQuery gives the****actual size****of the device while LayoutBuilder gives the****size of the enclosing parent****widget.*

**35) Difference between Double.infinity vs MediaQuery?**The difference can be summarized into:

* I want to be as big as my parent allows (double.infinity)
* I want it to be as big as the screen (MediaQuery).

Usually, you’ll want to use it double.infinity, but it's not always possible.  
Some Widgets allow their children to be as big as they want to be (Column, ListView, OverflowBox...).

In that situation using double.infinity creates a paradox:

* The parent allows any size
* The child wants the biggest size allowed by the parent

Using MediaQuery in these situations is bad though. You will rarely want to do that unless you're creating a widget similar to Scaffold.

**36) What is the use of AspectRatio?**You can use the widget to size the child to a specific aspect ratio. This widget first tries the largest width permitted by the layout constraints and then decides the height by applying the given aspect ratio to the width.

**37) What are dynamic, var, and final?  
dynamic:** can change the TYPE of the variable, & can change the VALUE of the variable later in code.  
**var:**can’t change TYPE of the variable, but can change VALUE of the variable later in code.  
**final:**can’t change TYPE of the variable, & can’t change VALUE of the variable later in code.

**38) Difference between Final vs const vs Static?**“**static**” means a member is available in the class itself instead of on instances of the class. That’s all it means, and it isn’t used for anything else. static modifies \*members\*. A static variable retains its values till the program finishes execution. Static members are referenced by the class name.

“**final**” means single-assignment: a final variable or field \*must\* have an initializer, and final is run constant. Once assigned a value, a final variable’s value cannot be changed. final modifies \*variables\*.

The **const** keyword in Dart behaves exactly like the final keyword. The only difference between final and const is that the const makes the variable constant from compile-time only. Using const on an object, makes the object’s entire deep state strictly fixed at compile time, and the object with this state will be considered frozen and completely immutable.

**39) What are Getter and setter methods?  
Getter and setter** methods are the class methods used to manipulate the data of the class fields. The getter is used to read or get the data of the class field whereas the setter is used to set the data of the class field to some variable.

**40) What is Factory constructors?**A factory constructor is a constructor that can be used when you don’t necessarily want a constructor to create a new instance of your class.

This might be useful if you hold instances of your class in memory and don’t want to create a new one each time (or if the operation of creating an instance is costly).  
Another use case is if you have certain logic in your constructor to initialize a final field that cannot be done in the initializer list.

class Car {  
 String make;  
 String model;  
 String yearMade;  
 bool hasABS;  
   
 factory Car.ford(String model, String yearMade, bool hasABS) {  
 return FordCar(model, yearMade, hasABS);  
 }  
}  
  
class FordCar extends Car {  
 FordCar(String model, String yearMade, bool hasABS):   
 super("Ford", model, yearMade, hasABS);  
   
}

41) **What is State?**A widget’s “state” is the information or data that it holds over time. It determines how the widget should be displayed on the screen. When a widget’s state changes, it is rebuilt to reflect the updated state. The result is a dynamic, responsive user interface. In simple words, State is the data that changes and updates the widgets on the app’s UI.

**42) Difference between Ephemeral state vs App state?  
Ephemeral State** -when your state variables are inside of the Stateful widget, it is known as an ephemeral state.

**App State** — When your state variables are outside of the Stateful widget, it's known as App state. (because that state is used by many widgets)  
For managing the app state, you have to use a state management solution (inherited widget or third-party library).

**43) Difference between dependencies vs dev\_dependencies packages?  
dependencies:**here we addalist of plugins that we have to include while deploying your App after the completion of your development stage.

**dev\_dependencies:**here we adda list of plugins that you want to try out at the development stage to test the apps at the development stage.  
For example: In the development stage, we use Mockito and test plugins and SDK to write the test cases and to test the complete behavior of the app. These kinds of plugins and SDKs we include in So in the release apps, there is no need of adding these plugins or SDK support.

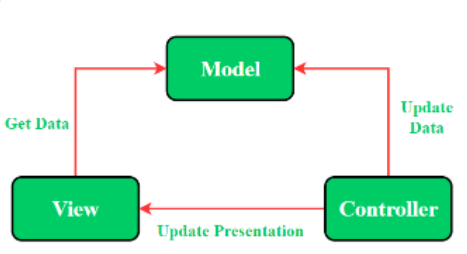
**44) What technology is Flutter built with?**Flutter is built with C, C++, Dart, and Skia (a 2D rendering engine). See this architecture diagram for a better picture of the main components. For a more detailed description of the layered architecture of Flutter, read the architectural overview.

**45) Difference between deactivate and dispose?  
Deactivte**: Called when this object is removed from the tree. deactivate is called when a widget may be dispose. But that is not guaranteed.  
**Dispose**: Called when this object is removed from the tree permanently. By understanding both sentences, you’ll see that deactivate will be called for widgets that are removed from the tree, temporarily or permanently, whereas dispose will only be called for widgets being removed permanently.

**46) Difference Between MVC, MVP, and MVVM Architecture Pattern?**

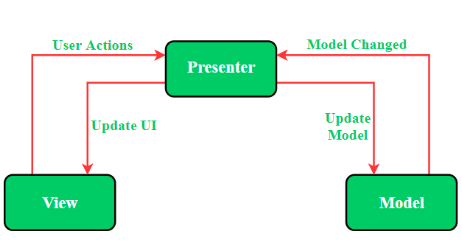
**The Model—View—Controller(MVC) Pattern**The MVC pattern suggests splitting the code into 3 components. While creating the class/file of the application, the developer must categorize it into one of the following three layers:

* **Model:** This component stores the application data. It does not know the interface. The model is responsible for handling the domain logic(real-world business rules) and communication with the database and network layers.
* **View:** It is the UI(User Interface) layer that holds components that are visible on the screen. Moreover, it provides the visualization of the data stored in the Model and offers interaction to the user.
* **Controller:** This component establishes the relationship between the View and the Model. It contains the core application logic and gets informed of the user’s response and updates the Model as per the need.



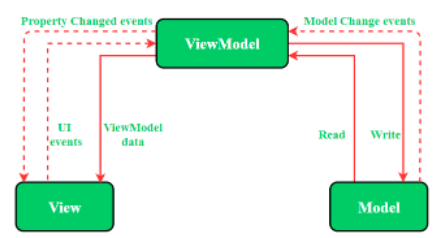
**The Model—View—Presenter(MVP) Pattern**MVP pattern overcomes the challenges of MVC and provides an easy way to structure the project codes. The reason why MVP is widely accepted is that it provides modularity, testability, and a more clean and maintainable codebase. It is composed of the following three components:

* **Model:** Layer for storing data. It is responsible for handling the domain logic(real-world business rules) and communication with the database and network layers.
* **View:** UI(User Interface) layer. It provides the visualization of the data and keeps a track of the user’s action to notify the Presenter.
* **Presenter:** Fetch the data from the model and applies the UI logic to decide what to display. It manages the state of the View and takes actions according to the user’s input notification from the View.



**The Model — View — ViewModel (MVVM) Pattern**MVVM pattern has some similarities with the MVP(Model — View — Presenter) design pattern as the Presenter role is played by the ViewModel. However, the drawbacks of the MVP pattern have been solved by MVVM. It suggests separating the data presentation logic(Views or UI) from the core business logic part of the application. The separate code layers of MVVM are:

* **Model:**This layer is responsible for the abstraction of the data sources. Model and ViewModel work together to get and save the data.
* **View:** The purpose of this layer is to inform the ViewModel about the user’s action. This layer observes the ViewModel and does not contain any kind of application logic.
* **ViewModel:** It exposes those data streams which are relevant to the View. Moreover, it serves as a link between the Model and the View.



**47) What is the difference between debug mode and profile mode?**in Simple words, we Use **debug mode** during development, when you want to use hot reload. and we Use **profile mode** when you want to analyze performance.  
**Debug includes debugging information in the compiled files (allowing easy debugging) while Release usually has optimizations enabled**. As far as conditional compilation goes, they each define different symbols that can be checked in your program, but they are language-specific macros.

**48) What is the difference between these operators ?? and ?.**

the ?? and ?. operators are used to handling null values and prevent null reference errors.

The ?? operator is known as the null coalescing operator. It is used to provide a default value when an expression evaluates to null.

In this example, the ?? operator checks if the value name is null. If it is null, then "Admin" is assigned to userName. If name is not null, then its value is assigned to userName.

The ?. an operator is known as the null-aware access operator. It is used to access properties or methods of an object that may be null, without throwing a null reference error. For example:

In this example, the ?. operator checks if the student object is null. If it is not null, then the sayHello() method is called on the student object. If it student is null, then nothing happens.

Overall, the ?? an operator is used to provide a default value when a variable is null, while the ?. operator is used to safely access properties or methods of an object that may be null.

**49) Differentiate between required vs optional vs named parameters in Dart.**

**Required Parameter**

The required parameter is a well know old-style parameter that we are all familiar with it  
***example****:*

findVolume(int length, int breath, int height) {  
 print('length = $length, breath = $breath, height = $height');  
}findVolume(10,20,30);

***output****:*

length = 10, breath = 20, height = 30

**Optional Positional Parameter**

the parameter will be disclosed with the square bracket **[ ]** & square bracketed parameters are optional.

***example:***

findVolume(int length, int breath, [int height]) {  
 print('length = $length, breath = $breath, height = $height');  
}findVolume(10,20,30);//valid  
findVolume(10,20);//also valid

***output:***

length = 10, breath = 20, height = 30  
length = 10, breath = 20, height = null // no value passed so height is null

**Named Parameters:**

These are parameters that can be passed in any order by passing the name of the parameter followed by the passed value. For example:

void sum({int num1, int num2});

This function is called this:

sum(num1: 12, num2: 24);

Also, named parameters can also have default values.

**50) Explain the different types of Streams?**There are two kinds of streams.

1. Single subscription streams:-  
The most common kind of stream. — It contains a *sequence of events* that are parts of a larger whole. Events need to be delivered in the correct order and without missing any of them. — This is the kind of stream you get when you read a file or receive a web request. — Such a stream can only be listened to once. Listening again later could mean missing out on initial events, and then the rest of the stream makes no sense. — When you start listening, the data will be fetched and provided in chunks.

2. Broadcast streams:-

* It is intended for individual messages that can be handled one at a time. This kind of stream can be used for mouse events in a browser, for example.
* You can start listening to such a stream at any time, and you get the events that are fired while you listen.
* More than one listener can listen at the same time, and you can listen again later after canceling a previous subscription.

**51) What are the keys in Flutter and when to use it?**In Flutter, a Key is a unique identifier for a widget that allows Flutter to identify a widget from its previous state. Keys are used to identify widgets when the widget tree changes, so that Flutter can preserve state for those widgets.

There are several types of keys in Flutter, including:

* GlobalKey: A global key is unique across the entire app, and is useful when you need to identify a widget from a different part of the app.
* UniqueKey: A unique key is unique within a widget's parent widget, and is useful when you need to identify a widget within a list of widgets.
* ValueKey: A value key is based on a value, such as a string or an integer, and is useful when you need to identify a widget within a list of widgets based on its value.

You should use keys in Flutter whenever you need to identify a widget and preserve its state. This is typically necessary when you are working with lists of widgets, or when you are building widgets that need to be able to update their state based on user interactions.

For example, if you have a list of items that can be reordered by the user, you can use keys to identify each item and preserve its state when it is reordered. Similarly, if you have a form with multiple input fields, you can use keys to identify each field and preserve its state as the user fills out the form.

It’s important to note that keys should only be used when necessary, as they can have a performance impact on your app. You should only use keys when you need to identify a widget and preserve its state, and you should choose the appropriate type of key based on your needs.

for more info click [here](https://shirsh94.medium.com/keys-in-flutter-concept-of-key-in-widgets-be1e5ebd3070).

**52) Explain async, await in Flutter?**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.

Example:-  
Future<void> test2() async {  
var a = await fetchData();  
}

**53) How does Dart AOT work?**The AOT-compiled code runs inside an efficient Dart runtime that enforces the sound Dart type system and manages memory using fast object allocation and a generational garbage collector.

**54) What are the similarities and differences between Future and Stream in Flutter?**Both futures and streams are parts of Dart. The main similarity between them is that they are both used for asynchronous programming.

also One of the main similarities is that both will return in the future(asynchronous). One difference is that Future only returns once. The stream will return again and again.

The main difference between them is:

* a future is used for a value that may be not available right now, may become available at some later moment, and then will not change. We want to react when the value becomes available (and e.g. display it)
* a stream is used when some value changes over time and we want to react to the changes (e.g. by displaying the current value)

**55) What’s the difference between async and async\* in Dart?**

In Dart, the async keyword is used to mark a function as asynchronous, meaning that it will return a Future object and can be awaited. The async\* keyword is used to mark a function as an asynchronous generator, meaning that it returns an object that implements the Stream interface and can be used to yield a sequence of values asynchronously.

Here’s an example of an asynchronous function:

Future<int> addAsyncData(int a, int b) async {  
 await Future.delayed(Duration(seconds: 1));  
 return a + b;

And here’s an example of an asynchronous generator function

Stream<int> countAsyncData(int item) async\* {  
 for (int i = 1; i <= item; i++) {  
 await Future.delayed(Duration(seconds: 1));  
 yield i;  
 }  
}

Asynchronous functions are useful for performing operations that may take some time to complete, such as making a network request or reading a file from disk. Asynchronous generators are useful for creating streams of data that can be consumed asynchronously.

You can await the results of an asynchronous function, but you cannot await the results of an asynchronous generator. Instead, you would typically use a for loop or the await for syntax to consume the values produced by the generator.

**56) Why does the first Flutter app build take so long?**

When you build the Flutter app the first time, it will take a longer time. It is because Flutter built the device-specific APK or IPA file. Thus, the Gradle and Xcode are used to build the file, taking a long time.

in simple words the first time you build a Flutter app, it can take a while because the Flutter framework needs to be compiled for the specific platform you are targeting (e.g., Android or iOS). This process involves downloading and installing any dependencies, as well as building the necessary native code for the platform. Additionally, if you are using a physical device to run the app, the app will need to be transferred to the device, which can also take some time.

**57) What is the difference between “main()” and “runApp()” functions in Flutter?**

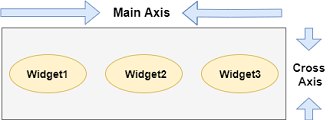
We can differentiate the main and runApp functions in Flutter as below:

* The main() function is responsible for starting the program. Without the main() function, we cannot write any program on Flutter.
* The runApp() function is responsible for returning the widgets that are attached to the screen as a root of the widget tree and will be rendered on the screen.

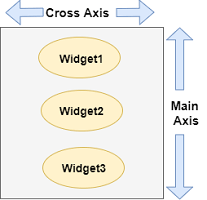
**58) When should you use mainAxisAlignment and crossAxisAlignment?**

We can use the crossAxisAlignment and mainAxisAlignment to control how row and column widgets align their children based on our choice.

The row’s cross-axis will run vertically, and the main axis will run horizontally. See the below visual representation to understand it more clearly.



The column’s cross-axis will run horizontally, and the main axis will run vertically. The below visual representation explains it more clearly.



For more detail click [here](https://medium.com/@shirsh94/flutter-crossaxisalignment-vs-mainaxisalignment-541f21623f88).

**59) What is the difference between SizedBox VS Container?**

`SizeBox` is a utility widget in Flutter that enforces a fixed size constraint on its child widget without any visual representation of its own. It allows you to specify the width and height of its child widget.

On the other hand, `Container` is a widget that combines various visual and layout properties. It allows you to specify the size, alignment, padding, margin, background color, and other styling options for its child widget. It provides more flexibility for controlling the appearance and layout of its child.

SizedBox:

* SizedBox is a widget used to enforce specific dimensions (width and height) on its child widget.
* It allows you to set the dimensions directly using properties like width and height.
* The primary purpose of SizedBox is to create a box of a specific size or to enforce size constraints on its child.
* It is useful when you want to ensure a widget has a specific size or create empty space of a particular size.

Container:

* Container is a versatile widget that provides various layout and styling options in addition to controlling the size of its child widget.
* It automatically sizes itself based on its child widget but also allows you to manually set its dimensions using properties like width and height.
* In addition to size control, Container offers options like padding, margin, alignment, and decoration.
* It is commonly used to build more complex UI layouts, where you need fine-grained control over the positioning and appearance of child widgets.

In summary, SizedBox is primarily used for setting explicit dimensions on its child, while Container offers more advanced capabilities for layout and styling, making it suitable for complex UI designs.

**60) Why is the build() method on State and not StatefulWidgets?**The main reason behind this is that the StatefulWidget uses a separate State class without building a method inside its body. It means all fields inside a Widget are immutable and includes all its sub-classes.

On the other hand, the StatelessWidget has its build and associated methods inside its body. It is due to the nature of StatelessWidget, which is rendered completely on the screen using the provided info. It also doesn’t allow any future changes in its State information.

The StatefulWidget allows us to change the State information during the app. Therefore, it is not suitable for storage in a build method to satisfy Widget class conditions where all fields are immutable. This is the main reason to introduce the State class. Here, we only need to override the createState() function to attach the defined State with the StatefulWidget, and then all expected changes happen in a separate class.

**61) Why do we need mixins in flutter?**Dart does not support multiple inheritances. Thus to implement the multiple inheritances in Flutter/Dart, we need mixins. Mixins provide a way to write the reusable class’s code in multiple class hierarchies.

**62) Why do we use a Ticker in Flutter?**Ticker in Flutter is a refresh rate of our animation. It is a class that sends a signal at a regular interval, i.e., around 60 times per second. We can understand it with our watch, which tics at regular intervals. At each tick, Ticker provides a callback method with the duration since the first ticks at each second, after it was started. Even if the tickers started at different times, they always synchronized automatically.

**63) When to use mixins and when to use interfaces in Dart?**The difference lies in the concept. If you understand this, you will use it in the right way.

In OOP, an interface is something that enforces the deriving class to implement a set list of public fields and methods.

But unlike other traditional programming languages like C# and JAVA, Dart does not have explicit interface types. Each class, by default, defines its interface composed of public fields and methods. **So, every class can act as an interface in Dart.**

*implements* keyword is to implement an interface. Also, a class can implement multiple interfaces.

1. In OOP, inheritance implies sharing of behavior between classes. We can not share features with an interface. So, when we implement a class, we can not share its behavior.

If you want to share the behavior across these two classes, you should use the *extends* keyword.

1. In OOP, a mixin is a class that contains methods for use by other classes. Unlike the interface and inheritance approach, a mixin doesn’t have to be the parent class of those other classes.

For more detail click [here](https://dart.dev/language/mixins).

So a mixin neither imposes usage restriction nor forces type restriction.

You will usually put common functions inside a mixin. Make use of the mixin by using the “*with”* keyword.

**64) What is the Container class in flutter?**The container class is a widget in Flutter which will contain other widgets. Container widget used for decorating its child widget. we can give properties like borders, padding, alignment, height, width, etc. The container class will only contain one child widget.

**65) What is the difference between adaptive and responsive design in Flutter?**

Adaptive design in flutter involves creating multiple layouts and styles that are optimized for specific screen sizes or device types. This means that the app will display a different layout depending on the device being used.

Responsive design in flutter involves designing a UI that automatically adjusts to different screen sizes and orientations. This means that the UI elements will resize and reposition themselves to fit the available screen space.

For more detail click [here](https://medium.com/@shirsh94/responsive-and-adaptive-apps-in-flutter-85171b922bc7).

**66) What are the extension methods in Dart? Why use it?**Extension methods, introduced in Dart 2.7, are a way to add functionality to existing libraries. You might use extension methods without even knowing it. For example, when you use code completion in an IDE, it suggests extension methods alongside regular methods.

For more information click [here](https://shirsh94.medium.com/extension-methods-in-flutter-dart-deep-dive-into-extension-methods-in-dart-9d4e2d361c86).

**67) In how many ways you can pass the parameters in Dart?  
Mainly on the three ways, you can pass parameters:**default, optional, and named.also, Dart functions allow positional parameters, named parameters, and optional positional and named parameters, or a combination of all of them. Positional parameters are the basic parameters, and they’re required.

**68) How to access property or method conditionally in Dart?**?. is known as a Conditional Property access operator used to guard access to a property or method of an object that might be null, we **put a question mark ( ? )** **before the dot ( .** ): This operator is widely used to prevent the access of the null object.

**69) Explain the Spread operator.**In Dart, Spread Operator (…) and Null-aware Spread Operator (…?) are used for inserting multiple elements in a collection like Lists, Maps, etc.

**Syntaxes:**

* **Spread operator**

...Data\_structureExample:-  
var a = [0,1,2,3,4];  
var b = [6,7,8,9];  
var c = [...a,5,...b];  
  
print(c); // prints: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

* Null-aware Spread operator

...?Data\_structureExmaple :-List<int> l1 = [1, 2, 3];List<int> nullList = null;List<int> result = [...l1, ...?nullList];print(result); // prints: [1, 2, 3]

70) Explain **Provider Pub/Library in Flutter.**Provider is a wrapper around InheritedWidget to make them easier to use and more reusable. this is probably the approach you should start with. The provider package is easy to understand and it doesn’t use much code. It also uses concepts that are applicable in every other approach. By using provider instead of manually writing InheritedWidget, you get: — state management pattern and separate business logic from UI — simplified allocation/disposal of resources — lazy-loading — a largely reduced boilerplate over making a new class every time devtools friendly — a common way to consume these InheritedWidgets increased scalability for classes with a listening mechanism that grows exponentially in complexity (such as ChangeNotifier, which is O(N²) for dispatching notifications).

**71) Explain Solid Principle.**

The SOLID principles are a set of guidelines that can help developers design software that is easy to maintain, extend, and scale. These principles were first introduced by Robert C. Martin in his book “Agile Software Development, Principles, Patterns, and Practices,” and they have since become an important part of the software development community.

The SOLID principles are:

1. Single Responsibility Principle (SRP): A class should have only one reason to change.
2. Open-Closed Principle (OCP): Software entities (classes, modules, etc.) should be open for extension but closed for modification.
3. Liskov Substitution Principle (LSP): Subtypes must be substitutable for their base types.
4. Interface Segregation Principle (ISP): Clients should not be forced to depend on interfaces they do not use.
5. Dependency Inversion Principle (DIP): High-level modules should not depend on low-level modules. Both should depend on abstractions.

Flutter is a framework for building cross-platform mobile applications, and these principles can be applied when designing and building apps with Flutter. Adhering to these principles can help you create a more maintainable, scalable, and flexible app.

for more detail checkout [here](https://medium.com/nerd-for-tech/solid-principles-in-a-flutter-32eaf7218476)

**72) Explain singleton class in flutter.**

In Flutter, a singleton class is a class that allows only a single instance of itself to be created, and provides a global point of access to that instance. Singleton classes are used to represent global state or resources that are shared by multiple parts of an application.

To create a singleton class in Flutter, you can use the singleton pattern, which involves creating a private constructor and a static method that returns the single instance of the class. Here's an example of a singleton class in Flutter:

class MySingletonClass {  
 static final MySingletonClass \_instance = MySingletonClass.\_internal();

factory MySingletonClass() {  
 return \_instance;  
 }

For more detail click [here](https://shirsh94.medium.com/singleton-flutter-design-patterns-f67e01ec8d19).

// Other methods and properties go here  
}

To use the singleton class, you would call the static factory method:

MySingletonClass().doSomething();

It’s important to note that in Flutter, it’s generally better to use dependency injection to manage shared state and resources, rather than relying on singletons. This can help to improve the testability and maintainability of your code.

**73) How to Secure your Flutter Application?**For securing flutter application, you can follow these steps,

1. Code obfuscation
2. Manage background snapshots
3. Stay up-to-date with your Flutter version
4. Use Flushing in-memory cache
5. Use local authentication
6. Use Secure Storage
7. Restrict network traffic
8. Use jail-breaking protection

For more information click [here](https://shirsh94.medium.com/8-tips-to-secure-your-flutter-mobile-apps-cc74deb37e1b).

**74) What is the difference between Provider vs. InheritedWidget?***Yes. The provider is indeed mostly features based on Inheritedwidgets.*

*If you want to make your own, then that’s fine. But you’ll quickly realize that, without a provider, you’ll have hundreds of useless repetitive lines.*

*The provider takes the logic of InheritedWidgets but reduces the boilerplate to the strict minimum.*

Provider is not a must, but should.

First of all, it’s promoted by Flutter Team and flexible enough to handle almost any state-management solution.

It might not be fair to say that InheritedWidget with dispose because Provider has too many different use cases and inherits some optimizations probably you won't find anywhere else.

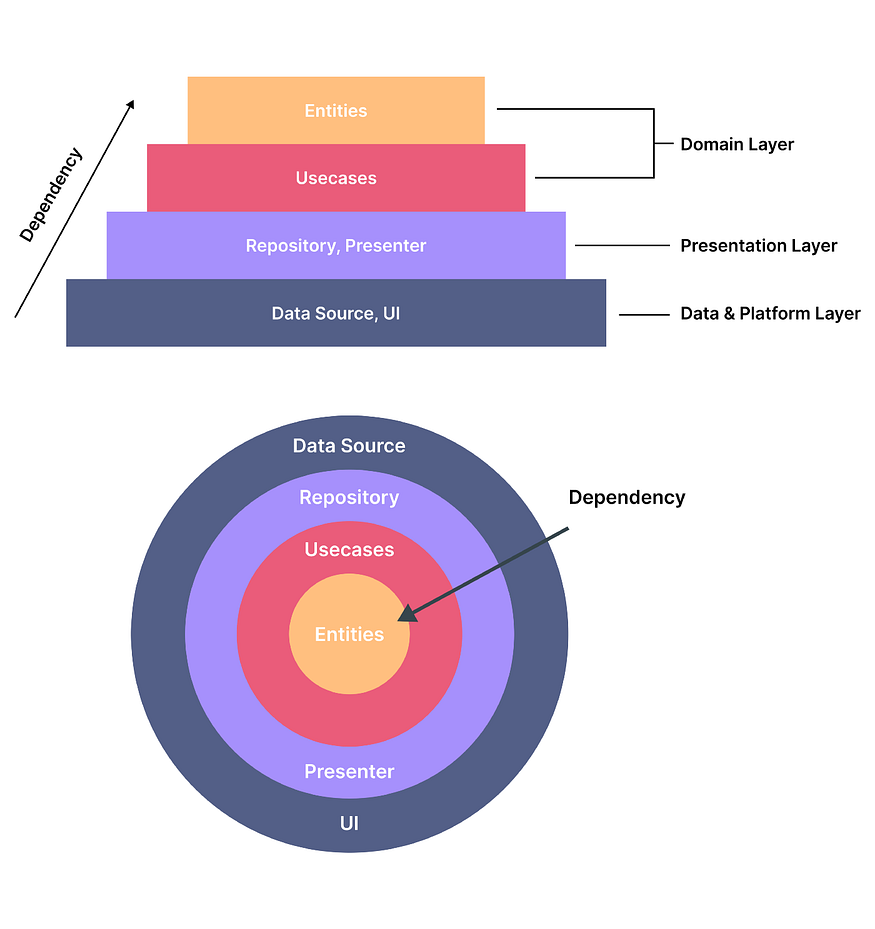
If you use InheritedWidget it in a large application, build methods always rebuild the whole build method. But with Provider you have Consumer widget which is can be very specific to control specific blocks of build the method, so you have more efficiency. Also listeners have less complexity than InheritedWidgets'(O(N) vs O(N²)).

The problem is since Flutter was intended to be a UI framework at first, the default state management solutions are also UI-oriented.

Lastly, since you’ll need different state-management patterns for different projects, one package-for-all scenario is invaluable IMO.

**75) Explain clean architecture.**

Clean architecture is a software design philosophy that separates the elements of a design into ring levels. An important goal of clean architecture is to provide developers with a way to organise code in such a way that it encapsulates the [business logic](https://www.techtarget.com/whatis/definition/business-logic) but keeps it separate from the delivery mechanism.



**76) Define some famous companies who using Flutter?**

**Some companies that use Flutter are**

* realtor.com,
* Tencent,
* the new york times,
* square,
* google,
* eBay,
* Sonos,
* BMW,
* Emaar

For more detail click [here](https://medium.com/nerd-for-tech/top-10-applications-made-by-flutter-81ec4ecfba05).

**77) How would you execute code only in debug mode?**

First to import this,

import 'package:flutter/foundation.dart' as Foundation;

then you can use kReleaseMode like

if(Foundation.kReleaseMode){ // is Release Mode ??  
 print('release mode');  
} else {  
 print('debug mode');  
}

So, by this condition you can execute code **only in debug mode.**

**78) What is Iterable collections in Dart?**An Iterable is **a collection of elements that can be accessed sequentially**. In Dart, an Iterable is an abstract class, meaning that you can’t instantiate it directly. However, you can create a new Iterable by creating a new List or Set .

**79) What is Concurrency in Dart?**Concurrency is **the execution of several instruction sequences at the same time**. It involves performing more than one task simultaneously. Dart uses Isolates as a tool for doing work in parallel.

**80) What is Typedef in Dart?**A **typedef**, or a function-type alias, helps to define pointers to executable code within memory. Simply put, a **typedef** can be used as a pointer that references a function.

typedef ManyOperation(int firstNo , int secondNo);   
//function signature   
  
Add(int firstNo,int second){   
 print("Add result is ${firstNo+second}");   
}   
Subtract(int firstNo,int second){   
 print("Subtract result is ${firstNo-second}");   
}  
Divide(int firstNo,int second){   
 print("Divide result is ${firstNo/second}");   
}   
Calculator(int a, int b, ManyOperation oper){   
 print("Inside calculator");   
 oper(a,b);   
} void main(){   
 ManyOperation oper = Add;   
 oper(10,20);   
 oper = Subtract;   
 oper(30,20);   
 oper = Divide;   
 oper(50,5);   
}

**81) What is Generics in Dart?**Dart Generics are **the same as the Dart collections, which are used to store the homogenous data**. As we discussed in the Dart features, it is an optionally typed language. By default, Dart Collections are the heterogeneous type. In other words, a single Dart collection can hold the values of several data types.

void main() {   
 List <String> logTypes = new List <String>();   
 logTypes.add("WARNING");   
 logTypes.add("ERROR");   
 logTypes.add("INFO");   
   
 // iterating across list   
 for (String type in logTypes) {   
 print(type);   
 }   
}

**82) What is Runes in Dart.**A rune can be defined as **an integer used to represent any Unicode code point**. As a Dart string is a simple sequence of UTF-16 code units, 32-bit Unicode values in a string are represented using a special syntax. The String class in the dart:core library gives ways to access runes.  
Example:-

import 'dart:core';   
void main(){   
 f1();   
}   
f1() {   
 String x = 'Runes';   
 print(x.codeUnits); // [82, 117, 110, 101, 115]  
}

**83) What is HTML DOM in Dart?**Every webpage can be considered an object and it exists inside a browser window. We can access the webpage using the web browser and it needed to be connected to the internet. The DOM is the acronym for the **Document object model**. A Document object denotes the HTML document that is displayed in that window.

**84) How many Data Types Support in Dart?**These are the type of values that can be represented and manipulated in a programming language.

The Dart language supports the following types−

* Numbers
* Strings
* Booleans
* Lists
* Maps

**85) What is Symbol in Dart?**Symbols in Dart are opaque, dynamic string names used in reflecting out metadata from a library. Simply put, symbols are a way to store the relationship between a human-readable string and a string that is optimized to be used by computers.

Reflection is a mechanism to get metadata of a type at runtime like the number of methods in a class, the number of constructors it has or the number of parameters in a function. You can even invoke a method of the type which is loaded at runtime.

In Dart reflection, specific classes are available in the **dart: mirrors** package. This library works in both web applications and command-line applications.

import 'dart:mirrors';   
void main(){   
 Symbol lib = new Symbol("foo\_lib");   
 String name\_of\_lib = MirrorSystem.getName(lib);   
   
 print(lib); //Symbol("foo\_lib")  
 print(name\_of\_lib); //foo\_lib  
}

**86) What are Dart Constants?**

Dart Constant is defined as an immutable object, which means it can’t be changed or modified during the execution of the program. Once we initialize the value to the constant variable, it cannot be reassigned later.

**Defining/Initializing Constant in Dart**

The Dart constant can be defined in the following two ways.

* Using the final keyword
* Using the const keyword

**87) What are Dart Callable Classes?**Dart provides the facility to call class instances like a function. To make callable class, we need to implement a **call()** method in it. Let’s understand the following example -

class Addition {  
 int call(int a, int b) => a + b;  
}void main() {  
 Addition addition = Addition();  
 var result = addition(1, 5);  
 print(result); // 6  
}

**88) What is Super Constructor in Dart?**In dart, the subclass can inherit all the variables and methods of the parent class, with the use of **extends** keyword but it can’t inherit the constructor of the parent class. To do so we make use of the **super constructor** in the dart. There are two ways to call a super constructor:

1. Implicitly
2. Explicitly

**Implicit super:** In this case, the parent class is called implicitly, when there is object creation of child class. Here we don’t make use of the super constructor but when the child class constructor is invoked then it calls the default parent class constructor.

**Explicit super:** If the parent constructor is default then we call it as followed in implicit super, but if it takes parameters then the superclass is invoked as shown in the syntax mentioned above.

**89) What is Anonymous Function in Dart?**An **anonymous** function in ***Dart*** is like a named function but they do not have names associated with it. An **anonymous** function can have zero or more parameters with optional type annotations. An **anonymous** function consists of self-contained blocks of code that can be passed around in our code as a function parameter.

* In Dart, most of the functions are named functions we can also create a nameless function known as an anonymous function, lambda, or closure.
* In Dart we can assign an anonymous function to constants or variables, later we can access or retrieve the value of closure based on our requirements:

**90) What is FFI in Dart?**Dart’s FFI (foreign function interface) allows Flutter apps to make use of existing native libraries that expose a C API. Dart supports FFI on Android, iOS, Windows, macOS, and Linux. For the web, Dart supports JavaScript interop.

**91) Explain Threading in Dart?**Dart/Flutter is single-threaded and not possible to share global variables. As each isolate has its own memory, space, and everything.  
To make it work like multi-threaded you have to use isolates and the communication will be used through ports by sending messages to one another. If you do not want to use Future you can use isolates. Flutter is written using Dart and Dart is a single-threaded language then Flutter apps are single-threaded. This means that a Flutter app can only do one thing at a time.

**91) Difference between Equatable vs Freezed?**Equatable and Freezed are both Dart packages that aid in simplifying code related to object equality and immutability, but they serve different purposes and offer varying levels of functionality.

Equatable is a lightweight solution primarily focused on providing a standardized way to implement equality comparisons in Dart. It simplifies the process by allowing developers to specify which properties should be considered for equality, thereby reducing boilerplate code for overriding the == operator and hashCode method.

On the other hand, Freezed is a more comprehensive code generation package that not only handles equality comparisons but also creates immutable data classes. It provides a range of features including deep equality checks, support for union types (sealed classes), and automatic generation of utility methods such as copyWith, toString, and JSON serialization/deserialization. Therefore, while Equatable is best suited for simple classes where equality comparison is the main concern, Freezed is ideal for more complex data models requiring immutability and a richer set of functionalities.

in simple words Equatable is “An abstract class that helps to implement equality without needing to explicitly override == and hashCode.”. Freezed is “Code generation for immutable classes that has a simple syntax/API without compromising on the features.”

they both override == and hashCode, but after that, they are very different. Equatable is a mixin to give a consistent, easy == and hashCode (although not necessarily the most efficient). Freezed is a build system that can use annotations to make immutable classes.

**92) Explain FittedBox Widget?**The FittedBox widget in Flutter is a single-child layout widget that helps maintain the app’s UI’s robustness across different screen sizes. It scales and positions its child widget within itself according to its fit property. The FittedBox widget uses the aspect ratio of the child widget to determine how to best fit and fill within the target box.

**93) What is Lazy Loading?  
Lazy loading** is a design pattern commonly used in computer programming and mostly in web design and development to defer the initialization of an object until the point at which it is needed. It can contribute to efficiency in the program’s operation if properly and appropriately used.

**94) How to check for types in Dart? Or What is sound typing in Dart?**

Use .runtimeType to get the type:

void main() {var data\_types = ["string", 123, 12.031, [], {} ];`  
   
for(var i=0; i<data\_types.length; i++){print(data\_types[i].runtimeType);if (data\_types[i].runtimeType == String){  
 print("-it's a String");}else if (data\_types[i].runtimeType == int){  
 print("-it's a Int");  
   
 }else if (data\_types[i].runtimeType == [].runtimeType){  
 print("-it's a List/Array");}else if (data\_types[i].runtimeType == {}.runtimeType){  
 print("-it's a Map/Object/Dict");}else {  
 print("\n>> See this type is not their .\n");  
 }   
}}

***Soundness*** is about ensuring your program can’t get into certain invalid states. A sound *type system* means you can never get into a state where an expression evaluates to a value that doesn’t match the expression’s static type. For example, if an expression’s static type is String, at runtime, you are guaranteed to only get a string when you evaluate it.

**95) What is BLOC Pattern?**It's a state management system for Flutter recommended by Google developers. It helps in managing state and make access to data from a central place in your project. MVP and MVVM are some good examples. Only the thing that will change is: that BLOC will be replaced with ViewModel in MVVM.

The best part about this pattern is that you won’t need to import any plugins or learn any custom syntax. Flutter already comes with everything you need.

**96) Describe some Performance best practices in code.**- Avoid repetitive and costly work in build() methods since build() can be invoked frequently when ancestor Widgets rebuild. — Avoid overly large single Widgets with a large build() function.  
Split them into different Widgets based on encapsulation but also on how they change: — When setState() is called on a State, all descendent widgets rebuild. Therefore, localize the setState() call to the part of the subtree whose UI actually needs to change.  
Avoid calling setState() high up in the tree if the change is contained to a small part of the tree.  
Avoid using the Opacity widget, and particularly avoid it in an animation. Use AnimatedOpacity or FadeInImage instead. For more information, see Performance considerations for opacity animation. When using an AnimatedBuilder, avoid putting a subtree in the builder function that builds widgets that don’t depend on the animation. This subtree is rebuilt for every tick of the animation. Instead, build that part of the subtree once and pass it as a child to the AnimatedBuilder. For more information, see Performance optimizations.  
Avoid clipping in an animation. If possible, pre-clip the image before animating it.  
Avoid using constructors with a concrete List of children (such as Column() or ListView()) if most of the children are not visible on screen to avoid the build cost.  
Don’t split your widgets into methods Use const widgets where possible Avoid rebuilding all the widgets repetitively — (by change notifier) Use itemExtent in ListView for long Lists Avoid using the Opacity, particularly in an animation <https://blog.codemagic.io/how-to-improve-the-performance-of-your-flutter-app./> <https://flutter.dev/docs/perf/rendering/best-practices>

**97) How to hide Android StatusBar in Flutter?**You can use SystemChrome.setEnabledSystemUIOverlays([]) to hide and SystemChrome.setEnabledSystemUIOverlays(SystemUiOverlay.values) to bring it back again.

**98) How do you detect the host platform from Dart code?**

a simple way for web and app both, we can detect the host platform

import 'dart:io' show Platform;  
import 'package:flutter/foundation.dart' show kIsWeb;  
var platformName = '';  
if (kIsWeb) {  
 platformName = "Web";  
} else {  
 if (Platform.isAndroid) {  
 platformName = "Android";  
 } else if (Platform.isIOS) {  
 platformName = "IOS";  
 } else if (Platform.isFuchsia) {  
 platformName = "Fuchsia";  
 } else if (Platform.isLinux) {  
 platformName = "Linux";  
 } else if (Platform.isMacOS) {  
 platformName = "MacOS";  
 } else if (Platform.isWindows) {  
 platformName = "Windows";  
 }  
}  
print("platformName :- "+platformName.toString());

**99) How to get .apk and .ipa files from flutter?**For apk (Android) you need to run the command :

flutter build apk --release

If you want to split the apks per abi (Split Apk) then run

flutter build apk --target-platform android-arm,android-arm64,android-x64 --split-per-abi

For ipa (iOS) you need to run the command :

flutter build ios --release

From the console

P.S. --release is optional as it is by default if you need debug build, just replace --release with --debug

you can find the released APK or IPA files form

build/app/outputs/flutter-apk/app-release.apk

**100) What does the ‘yield’ keyword do in flutter?**yield adds a value to the output stream of the surrounding async\* function. It's like return, but doesn't terminate the function.

See <https://dart.dev/guides/language/language-tour#generators>

Stream asynchronousNaturalsTo(n) async\* {  
 int k = 0;  
 while (k < n) yield k++;  
}

When the yield statement executes, it adds the result of evaluating its expression to the stream. It doesn’t necessarily suspend (though in the current implementations it does).

*So that’s all the questions that I think it’s enough for understanding flutter knowledge.because if you are able to get its core concept, you can answer similar types of questions as well.  
I took it from many websites as some research if you find out any wrong info or miscorrected also if you think any major questions that I missed to include in there, please write in the comment below.*