1. What is flutter?

Flutter is an open-source UI toolkit or software development kit (SDK) developed by Google to build mobile, web, and desktop applications from a single codebase.

2. Advantages of Flutter or Features

- Cross-platform Development
- Hot Reloading and Hot Restart
- Similar to Native App performance
- Good Community Support
- Minimal Code
- Documentation

- Flexible UI

3. What are the types of widgets present in Flutter?

- **Stateless Widget:** A widget that does nothing is a Stateless Widget. In essence, they are static and don’t store any state. Thus, they don’t save values that may change. Icon, Text, and IconButton are some examples of Stateless widgets.
- **Stateful Widget:** A widget that does anything is a Stateful Widget. Stateful widgets are dynamic by nature, which means they can monitor changes and update the UI accordingly. Radio, Slider, CheckBox, and Image are some of the examples of Stateful widgets.

4. Talk about different build modes in Flutter.

- **Debug mode** is used to debug the application on the physical device, emulator, or simulator.
- **Profile mode** is used to analyze the performance of your app. Here, some extensions and tracing are enabled.
- **Release mode** is enabled to deploy your app. Here, assertions, service extensions, and debugging are disabled.

5. What is Tree shaking?

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.

6. What is pubspec.yaml file?

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.

7. What is Context?

Context is a handle to the location of the widget in the widget tree. It is a BuildContext instance that gets passed to the builder of a widget to know where it is inside the widget tree. It is nothing but a reference to the location of a Widget within the tree structure of all the built widgets.

8. Explain BuildContext

BuildContexts are used to identify or locate widgets in widget trees. Each widget has its BuildContext, i.e., one BuildContext per widget. We're using it to find references to other widgets and themes. In addition, you can utilize it to interact with widget parents and access widget data.

9. Explain Devtools

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.

10. What is the HTTP package?

The HTTP package is used in the Flutter project to provide a simple way to fetch data from the Internet.

11. How can we create HTTP requests in Flutter?

http.get('https://xyz.com/users/1');

It will return a Future <http.Response>.

12. What is the use of Navigation.push and Navigation.pop function?

The push method is used to add a route to the stack of routes managed by the navigator.

The pop method is used to remove the current route from the stack of routes managed by the navigator.

13. Explain the flutter widgets.

Each element on the Flutter app's screen is considered a widget. In Flutter, the UI is built using widgets.

14. What do you understand from hot reload and hot restart?

Hot reload means injecting the updated source code files into running Dart VM (Virtual Machine). The Hot reload process does not add only new classes but it also adds properties, fields, and methods to existing classes, and changes existing functions.

Hot restart works by resetting the app’s current state to the app’s initial state.

15. What do you understand from ‘State’? What is the use of the setState() method?

The state is the data that your application currently showing.

State refers to the information that you can read synchronously when a widget is built and it might change during the lifetime of the widget.

The ‘setState’ method is used to notify the changed state of an internal object.

16. Explain the lifecycle of a Stateful Widget.

- createState
- initState
- didChangeDependencies
- build
- didUpdateWidget
- setState
- deactivate
- dispose

17. What is the Container class in flutter?

The Container class provides the ability to create a widget with certain properties like padding, borders, height, width, border, etc.

18. What is the difference between SizedBox VS Container?

The Container in Flutter is a parent widget that can contain multiple child widgets and manage them efficiently through width, height, padding, background color, etc. If we have a widget that needs some background styling may be a color, shape, or size, we may wrap it in a container widget.

The SizedBox widget in Flutter is a box that comes in a specified size. Unlike Container, it does not allows us to set the color or decoration for the widget. We can only use it for sizing the widget passed as a child. It means it forces its child widget to have a specific width or height.

19. What are packages and plugins in Flutter?

A package is a group of similar types of classes, interfaces, and sub-packages. The packages and plugins help us to build the app without having to develop everything from packages.

20. Why is the Android and iOS folder in the Flutter project?

Android: This folder holds a complete Android project. It is used when you create the Flutter application for Android. When the Flutter code is compiled into the native code, it will get injected into this Android project, so that the result is a native Android application. **For Example**: When you are using the Android emulator, this Android project is used to build the Android app, which is further deployed to the Android Virtual Device.

iOS: This folder holds a complete Mac project. It is used when you build the Flutter application for iOS. It is similar to the Android folder, which is used when developing an app for Android. When the Flutter code is compiled into the native code, it will get injected into this iOS project, so that the result is a native iOS application.

21. What is Tween Animation?

In a tween animation, it is required to define the start and endpoint of animation. It means the animation begins with the start value, then goes through a series of intermediate values, and finally reached the end value. It also provides the timeline and curve, which defines the time and speed of the transition. The widget framework provides a calculation of how to transition from the start and endpoint.

22. What is Stream in Flutter?

A stream is like a pipe, you put a value on the one end and if there's a listener on the other end that listener will receive that value. A Stream can have multiple listeners and all of those listeners will receive the same value when it's put in the pipeline. The way you put values on a stream is by using a StreamController.

23. Explain the difference between "??" and "?" Operators.

expr1 ?? expr2

The “??” operator is used to evaluate and returns the value between two expressions based on the given condition.

condition ? expr1 : expr2

This operator first checks the condition, and if it is true, it will evaluate expr1 and returns its value (if the condition is matched). Otherwise, it evaluates and returns the value of expr2.

24. What are keys in Flutter, and when to use it?

- 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.

25. What is the difference between WidgetsApp and MaterialApp?

WidgetsApp is used for basic navigation. It includes many foundational widgets together with the widgets library that Flutter uses to create the UI of our app.

MaterialApp, along with the material library, is a layer that is built on the top of WidgetsApp and its library. It implements Material Design that provides a unified look and feels to our app on any platform.

26. What types of tests can you perform in Flutter?

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 our 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.

27. What are Null-aware operators?

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 ?? operator returns the first expression if and only if it is not null.
- The ??= operator in Dart is used when we want to assign a value if and only if it is not null.

- The ? operator is used when we want to make sure that we don't invoke a function of a null value. It will call a function if and only if the object is not null.

28. What is the use of Ticker in Flutter?

We use a ticker to tell how often our animation is refreshed in Flutter.

29. What is the use of Mixins?

Multiple inheritances are not supported by Dart. Thus, we need mixins to implement multiple inheritances in Flutter/Dart. The use of mixins makes it easy to write reusable class code in multiple class hierarchy levels.

30. What is state management?

Using it, states of various UI controls are centralized to handle data flow across an application. It can be a text field, radio button, checkbox, dropdown, toggle, form, and so on. In Flutter, state management can be categorized into two types as follows:

- **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:** This is different from the ephemeral state since it is a 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. By means of scoped_model, Flutter provides support for this state.

31. Explain Flutter Provider.

The provider package is an easy-to-use package which is basically a wrapper around the InheritedWidgets that makes it easier to use and manage. It provides a state management technique that is used for managing a piece of data around the app.

32. What is await in Flutter?

Until the async method is finished, await interrupts the process flow.

33. Can you explain the concepts of Isolate, Event Loop, and Future in Dart?

In Dart, an **Isolate** is akin to a separate execution thread with its own memory, ensuring that Dart remains free of shared-state concurrency issues. Each isolate has its own memory heap, ensuring that no isolate's state is accessible from any other.

The **Event Loop** is a mechanism that handles the execution of events or messages for a particular thread. It continually checks if there are tasks to execute and runs them in order.

Future represents a potential value or an error that will be available at some time in the future. It's a core part of Dart's asynchronous programming model, allowing developers to write non-blocking code for operations that might take time, like fetching data from a server.

34. How to Implementing Interfaces in Flutter ?

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

```
class country_info {

 String country_name(){}

 String flag_image_url(){}

}

class country  implements country_info  {
  String country_name() {
   return "Viet Nam";
 }
 String flag_image_url() {
   return "VN";
 }
}
```

35. What are the differences between JIT and AOT?

JIT (Just-In-Time) and AOT (Ahead-Of-Time) are two compilation approaches in Dart.

- **JIT** compilation happens at runtime, translating the code into machine language just before it's executed. This allows for features like hot-reloading in Flutter, where changes can be injected into a running application.
- **AOT**, on the other hand, compiles the code into machine language before the app is launched. This results in faster startup times and optimized performance, making it the preferred choice for production Flutter apps.

36. How do `final` and `const` differ in Dart?

In Dart, both `final` and `const` are used to declare variables that cannot be reassigned, but they serve different purposes:

- The value of a `final` variable is determined at **runtime**. It can be set once, typically derived from runtime computations, making it suitable for values that remain constant during execution but might change between runs.
- The value of a `const` variable is determined at **compile-time**, and it must also be immutable. This makes it ideal for values that are truly constant across all instances and runs, such as physical constants.

37. What access modifiers are available in Dart?

Dart offers a set of access modifiers to control the visibility of members: **public** (default, if no modifier is specified), **private** (indicated by a leading underscore `_`), and **protected** (not explicitly available but achieved through conventions). Dart's approach to privacy is library-based, meaning that private members are hidden within the same library file but can be accessed across classes within that file.

38. What are named parameters and optional parameters in Dart?

Named Parameters: In Dart, named parameters are specified by their name rather than their position during a function call. This enhances code readability, especially when a function has many parameters. They are defined within curly braces `{}` in the function declaration.

Optional Parameters: Dart supports two types of optional parameters: positional and named.

- **Positional:** These are wrapped in square brackets `[]` in the function definition. When invoking the function, you can skip these parameters.
- **Named:** These are combined with named parameters, allowing them to be omitted during a function call. They can either have a default value or be marked as nullable.

In summary, while named parameters enhance clarity in function calls, optional parameters provide flexibility in how functions are invoked.

39. What is the difference between a named constructor and a factory?

In Dart, a **named constructor** allows a class to define multiple ways to initialize, using different names for clarity. It directly creates a new instance of the class. On the other hand, a **factory** is a special kind of constructor that doesn’t always return a new instance. Instead, it can return an existing instance, an instance of a subtype, or even an instance of a completely different class. While named constructors offer varied initialization methods, factories provide greater control over the object creation process.

40. What are the four principles of Object-Oriented Programming (OOP)?

The four foundational principles of OOP are Encapsulation, Inheritance, Polymorphism, and Abstraction.

- **Encapsulation** bundles data and methods operating on that data within a single unit, ensuring data integrity.
- **Inheritance** allows a class to inherit properties and behaviors from another class.
- **Polymorphism** permits one interface to be used for a general class of actions.
- **Abstraction** hides complex implementations and exposes only the necessary functionalities.

41. Can you provide an overview of the SOLID principles?

SOLID is an acronym representing five design principles that ensure software is scalable, maintainable, and organized. They are:

1. **S**ingle Responsibility Principle: A class should have only one reason to change.
2. **O**pen/Closed Principle: Software entities should be open for extension but closed for modification.
3. **L**iskov Substitution Principle: Subtypes must be substitutable for their base types.
4. **I**nterface Segregation Principle: Clients should not be forced to depend on interfaces they don't use.
5. **D**ependency Inversion Principle: High-level modules should not depend on low-level ones; both should depend on abstractions

42. How do `Object`, `dynamic`, and `var` differ in Dart?

- `Object` is the root class for all Dart classes, allowing a variable to hold any type of value but requires explicit casting for most operations.
- `dynamic` is a type that bypasses static type checking, offering flexibility at the cost of forgoing some compile-time checks.
- `var` is a keyword used to declare a variable without specifying its type. Dart determines and fixes the variable's type based on its initial value at compile-time.

43. Can you define the cascade and spread operators in Dart?

The **cascade operator** (`..`) allows for performing a series of operations on a single object without breaking the chain. For instance:`var obj = Object()..method1()..method2();`The **spread operator** (`...`) is used to insert multiple elements from one collection into another. It's especially useful when constructing lists or other collections:`var list = [1, 2, ...otherList, 3];`

44. What are the different approaches to state management in Flutter, and how do they operate?

In Flutter, state management refers to the way developers handle the data used by the app to influence its behavior and appearance. It’s about maintaining and manipulating the state, or data, of a widget, and determining how the changes in state reflect in the UI.

- **InheritedWidget** is a foundational class in Flutter, which is particularly useful for small to medium-sized projects. It simplifies the transfer of data down the widget tree, eliminating the need for numerous constructor arguments, making the code cleaner and more manageable.

- The **Provider Package** built atop InheritedWidget, is suitable for medium to large-sized projects, offering a range of features for handling state, including dependency injection, and it encapsulates common patterns of using InheritedWidget, making it more user-friendly.
- **Bloc Pattern** is ideal for managing the state in large, complex projects. It promotes a clear separation between the user interface and business logic, making the components of the application easier to debug and test.
- The **Redux Pattern**, originally developed for JavaScript applications, maintains all the application's state information in a single entity called the store. It provides a single source of truth, making it easier to conceptualize the state of the application, but might be overkill for simpler state needs.
- **MobX** is another approach, best suited for developers who prefer working with reactive programming paradigms. It provides a reactive state that automatically updates the UI when the state changes, making state management seamless and efficient.

The choice of approach should align with the project's complexity and specific requirements, ensuring smooth development and optimal app performance.

45. What are the differences between Bloc and Cubit?

In Flutter, **Bloc** and **Cubit** are distinct **state management solutions** with unique mechanisms, catering to different levels of complexity.

- Bloc, ideal for more complex scenarios, employs a **reactive programming model**, using **streams** and requiring the definition of **events** and **states** to manage state transitions meticulously. It's particularly useful when multiple states and transitions are involved, necessitating a detailed and structured approach.

- On the other hand, **Cubit** is simpler and more direct, eliminating the need for event definitions and allowing state changes through simple function calls. This makes Cubit suitable for situations where simplicity and rapid development are crucial.

In essence, while Bloc offers structured solutions for intricate scenarios, Cubit is optimal for simpler, more straightforward state management needs.

46. What are the differences between BlocBuilder, BlocListener, and BlocConsumer?

In the Bloc library, **BlocBuilder**, **BlocListener**, and **BlocConsumer** serve distinct purposes, each contributing to efficient state management in Flutter applications.

1. **BlocBuilder** is a widget that rebuilds its UI at every state change in the Bloc. It's primarily used for UI rendering and is ideal when the UI needs to be redrawn in response to state changes. For instance:`BlocBuilder<MyBloc, MyState>(` `builder: (context, state) {return Text('$state');},)`
2. **BlocListener** is a widget that does not rebuild the UI but instead reacts to state changes, making it suitable for performing actions like navigation or showing a dialog. For example:`BlocListener<MyBloc, MyState>(listener: (context, state) {},` `child: Container(),)`
3. **BlocConsumer** combines the functionalities of both BlocBuilder and BlocListener. It rebuilds its UI and performs actions in response to state changes, allowing developers to handle both within the same widget.

Here's a simple usage:`BlocConsumer<MyBloc, MyState>(` `listener: (context, state) {},` `builder: (context, state) {},)`

In summary, BlocBuilder is for UI rendering, BlocListener is for reacting to state changes without UI rebuilding, and BlocConsumer is a hybrid, catering to both UI and action-based reactions to state changes.

47. Can you provide an overview of SQLite in Flutter?

In Flutter, **SQLite** is utilized for local data persistence, serving as a self-contained, serverless SQL database engine, ideal for mobile devices. It is particularly beneficial for apps requiring offline functionality, enabling data access and modification without internet connectivity, with synchronization occurring once online.

Developers often use the `sqflite` package to integrate SQLite, leveraging its high-level APIs for various database operations like CRUD (Create, Read, Update, Delete), utilizing familiar SQL queries for interaction.

SQLite stores all its data in a single file on the device, ensuring data persistence across app launches. It is especially suitable for managing local, structured, and moderately-sized datasets, such as user preferences or game scores, providing a balanced solution for simple, efficient local data management in Flutter apps.

48. 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.

49. 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.

50. 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.