

Perguntas da entrevista do Dart

Iniciante (40 perguntas)

1. O que é a linguagem de programação Dart?

Dart é uma linguagem de programação orientada a objetos, desenvolvida pelo Google. Ela é otimizada para a criação de aplicativos para múltiplas plataformas (web, mobile, desktop)

2. Quais são os principais recursos do Dart?

Orientação a Objetos (POO)
Segurança Nula (Null Safety)
Compilação AOT
Assincronicidade

3. Como você define uma variável no Dart?

Você define uma variável usando a palavra-chave var ou especificando explicitamente o tipo de dado (ex: String, int, double)

4. Qual é a diferença entre var, final e const no Dart?

var: Variável cujo valor pode ser alterado (mutável) após a inicialização.
final: Variável cujo valor só pode ser definido uma única vez (imutável) e deve ser definida no momento da execução (runtime).
const: Variável que deve ser uma constante de tempo de compilação. Seu valor precisa ser conhecido antes de o programa ser executado.

5. Qual é o propósito da função main() no Dart?

A função main() é o ponto de entrada (início) de qualquer programa Dart. O código dentro dela é a primeira coisa que é executada quando o programa é iniciado.

6. Como criar uma função no Dart?

Uma função é criada especificando um tipo de retorno (ou void se não retornar nada), um nome, e parênteses que podem conter parâmetros (argumentos).

7. Quais são os diferentes tipos de dados no Dart?

Dart possui tipos de dados básicos para representar diferentes tipos de informação:

Números (num): int, double

Texto (String):

Booleano (bool):

Lista (List):

Mapa (Map):

Conjunto (Set):

8. Qual é o tipo dinâmico no Dart?

O tipo dynamic é um tipo de dado especial que permite que a variável armazena qualquer tipo de valor (int, String, bool, etc.) e que o tipo do valor possa ser alterado em tempo de execução.

9. Como o Dart lida com a segurança de tipos?

Ele lida de forma estática e forte, o que significa que ele verifica os tipos das variáveis durante a compilação, antes mesmo de o programa rodar.

10. Qual é a diferença entre == e identical() no Dart?

== (Operador de Igualdade): Verifica se dois objetos são iguais em valor
identical(): Verifica se duas referências apontam para o mesmo objeto na memória (identidade).

11. Como escrever um loop for básico no Dart?

O loop for é usado para executar um bloco de código repetidamente, geralmente por um número fixo de vezes.

```
for (int i = 0; i < 5; i++) {  
  print('Contagem: $i');  
}
```

12. Como funciona a Lista no Dart? Você pode dar um exemplo de uso de uma lista?

Uma Lista (List) no Dart é uma coleção ordenada de objetos. É o equivalente a um array em outras linguagens. Cada item em uma lista tem um índice

(posição) que começa em zero (0).

13. O que é um mapa no Dart? Como ele é usado?

Um Mapa no Dart é uma coleção não ordenada de pares de chave-valor. É semelhante a um dicionário ou um objeto JSON. As chaves devem ser únicas e são usadas para buscar seus respectivos valores.

14. Como você declara e inicia um conjunto no Dart?

Usando um Conjunto (Set) no Dart que é uma coleção não ordenada de valores onde cada elemento é único. Você declara e inicializa um conjunto usando chaves {} ou o construtor Set().

15. Qual é a diferença entre uma lista e um conjunto no Dart?

A principal diferença reside em duas características:

Ordem:

Lista (List): Ordenada por índice (posição).

Conjunto (Set): Não ordenada

Unicidade:

Lista (List): Permite elementos duplicados.

Conjunto (Set): Não permite elementos duplicados

16. Como você lida com valores nulos no Dart?

No Dart lido com valores nulos através do recurso de **Segurança Nula (Null Safety)**.

17. O que são async e await no Dart?

async e await são palavras-chave usadas para escrever código assíncrono que se parece e se comporta como código síncrono (executado linha por linha).

async: É colocado antes da declaração de uma função para indicar que ela retornará um Future e pode conter a palavra-chave await.

await: É colocado antes de uma expressão que retorna um Future. Ele pausa a execução da função async até que o Future (a operação assíncrona) seja concluído e seu resultado esteja disponível.

18. Como você lida com exceções no Dart?

Exceções no Dart são tratadas usando o bloco try-catch-finally. Isso permite que o programa se recupere de erros inesperados sem travar completamente.

19. Você pode explicar o try, catch e finally block no Dart?

20. Qual é o propósito do Future no Dart?

Um Future no Dart é um objeto usado para representar o resultado de uma operação assíncrona que ainda não foi concluída. Ele é como uma promessa de que um valor estará disponível em algum momento no futuro.

21. Qual é a diferença entre Future e Stream no Dart?

Ambos, Future e Stream, lidam com a programação assíncrona, mas representam diferentes tipos de dados assíncronos

22. Como você define uma classe no Dart?

Uma classe é um modelo ou *blueprint* para criar objetos. Ela encapsula dados (variáveis/propriedades) e comportamento (funções/métodos). Onde você a define usando a palavra-chave class.

23. Qual é a diferença entre uma variável de instância e uma variável estática no Dart?

Variável de Instância (id): É como o nome de uma pessoa. Cada pessoa tem um nome diferente.

Variável Estática (totalCriado): É como a população total da sua cidade. É um número que pertence à cidade (a classe) e é o mesmo para todos que vivem nela. É usada para informações que são universais para todos os objetos daquela classe.

24. Qual é o propósito da palavra-chave this no Dart?

A palavra-chave this é usada dentro de uma classe para se referir à instância atual do objeto. Seu principal propósito é desambiguação e referência

25. O que são getters e setters no Dart?

Getters e Setters são métodos especiais que fornecem controle sobre como

as variáveis de instância de uma classe que são lidas e modificadas.

Getter é um método que lê ou calcula um valor, parecendo uma variável, permitindo que você execute lógica antes de retornar o valor.

Setter é um método que modifica o valor de uma propriedade, permitindo que você execute lógica (como validação) antes de permitir a alteração.

26. Como criar um construtor no Dart?

```
class Ponto {  
    int x, y;  
    Ponto(this.x, this.y);  
}
```

27. Qual é a diferença entre um construtor nomeado e um construtor padrão no Dart?

Construtor Nomeado: São construtores adicionais que permitem criar instâncias da classe de diferentes maneiras, com nomes descritivos. Eles são definidos como NomeDaClasse.nomeDoConstrutor().

Construtor Padrão: É o construtor principal da classe. Você só pode ter um construtor padrão, e ele tem o mesmo nome da classe. É invocado diretamente: Classe().

28. Como funciona a herança no Dart?

Herança é um princípio da POO que permite que uma nova classe herda as propriedades e o comportamento de uma classe existente. No Dart, a herança é implementada usando a palavra-chave extends.

29. Você pode explicar o conceito de mixins no Dart?

Mixins são uma forma de reutilizar código de classe em múltiplas hierarquias de classes, sem a necessidade de herança tradicional, mixin é uma classe tendo funcionalidades você pode "misturar" em outras classes usando a palavra-chave with.

30. Qual é o propósito da palavra-chave super no Dart?

A palavra-chave super é usada em uma subclasse (filha) para se referir à sua superclasse (pai). Ela tem dois usos principais:

1. **Chamar o Construtor do Pai:** Para garantir que a parte da superclasse do objeto seja inicializada corretamente.
2. **Acessar/Chamar Membros do Pai:** Para acessar métodos ou variáveis que foram reescritos na subclasse, mas que você ainda precisa da versão original.

31. O que é uma classe abstrata no Dart?

Uma classe abstrata é uma classe que não pode ser instanciada. Ela serve como um modelo para outras classes que a herdarão.

32. O que é uma interface no Dart? Como implementá-la?

33. Como você define um padrão singleton no Dart?

O padrão Singleton garante que uma classe tenha apenas uma única instância e fornece um ponto de acesso global a essa instância. No Dart, a forma mais comum de implementar um Singleton é usando, um construtor privado

34. Qual é a diferença entre is e as no Dart?

is (Verificação de Tipo): É um operador booleano que verifica se um objeto é de um determinado tipo. Retorna true ou false.

as (Conversão de Tipo/Casting): É um operador de conversão que tenta converter o objeto para um tipo específico.

35. Você pode explicar o conceito dos operadores is e is! no Dart?

is verifica se um objeto é de um determinado tipo.

is! é o operador de negação de tipo, que verifica se um objeto **não é** de um determinado tipo. É o equivalente a !(objeto is Tipo).

36. Como escrever uma instrução switch-case no Dart?

O switch é usado quando você tem muitas opções

```
var estado = 'MG';

switch (estado) {
    case 'SP':
        print('São Paulo');
    case 'RJ':
    case 'ES':
        print('Rio de Janeiro ou Espírito Santo');
    case 'MG':
        print('Minas Gerais');
    default:
        print('Outro estado');
}
```

37. Como você pode executar uma verificação de tipo no Dart?

A verificação de tipo é feita principalmente usando o operador **is**. Quando **objeto is Tipo** retorna true, o Dart executa um recurso chamado Type Promotion. Isso significa que, dentro do bloco if, a variável é tratada automaticamente como sendo daquele tipo, permitindo acesso seguro aos seus métodos e propriedades.

38. Qual é a diferença entre var e dynamic no Dart?

var: É um atalho para não ter que digitar o tipo, mas o tipo permanece fixo.
dynamic: É o tipo "tudo ou nada". O Dart desliga a maioria dos avisos de segurança para ele.

39. Qual é o significado da palavra-chave late no Dart?

A palavra-chave late é usada para atrasar a inicialização de uma variável. Ela tem dois usos principais no Dart

Inicialização Atrasada: Inicializa uma variável na primeira vez que ela for

usada, em vez de quando é declarada.

Variáveis Não-Nuláveis: Permite que você declare uma variável não-nula sem inicializá-la imediatamente.

40. Qual é a diferença entre um Stream e um Iterable no Dart?

Ambos lidam com sequências de dados, mas se diferenciam pelo modo de entrega e a forma de consum.

Intermediário (40 questões)

1. Como executar programação assíncrona no Dart?

A programação assíncrona no Dart é executada principalmente usando três ferramentas:

2. Você pode explicar o ciclo de vida de um Future no Dart?

O ciclo de vida de um Future tem três estados principais:

Incompleto: O Future é criado e a operação assíncrona associada a ele foi iniciada, mas ainda não foi concluída.

Concluído com Valor: A operação assíncrona foi concluída com sucesso e o Future agora contém o resultado

Concluído com Erro: A operação assíncrona falhou, e o Future agora contém um objeto Error ou Exception.

3. O que é um Stream e como ele é usado no Dart?

Um Stream no Dart é uma sequência assíncrona de dados. Ele permite que você receba múltiplos valores ao longo do tempo.

listen: O método mais comum para consumir um *stream*. Você fornece callbacks para lidar com novos dados, erros e a conclusão do *stream*.

Operadores: Os *streams* podem ser transformados usando métodos como map, where.

4. Como criar um Stream no Dart? Dê um exemplo.

A maneira mais comum e flexível de criar um Stream é usando a classe **StreamController**.

```
import 'dart:async';
```

```
final controller = StreamController<int>();  
  
void main() {  
    controller.stream.listen((data) {  
        print('Recebido: $data');  
    });  
  
    controller.add(10);  
    controller.add(20);  
  
    controller.addError('Algo deu errado!');  
    controller.close();  
}
```

5. Como a palavra-chave await funciona no Dart?

Quando o `await` é encontrado antes de uma expressão `Future`, ele suspende a execução do código restante na função `async`. No entanto, ele não bloqueia a *thread* principal. Em vez disso, ele cede a execução de volta ao *Event Loop* do Dart, permitindo que outros códigos (incluindo a renderização da UI) continuem a ser executados. Assim que o `Future` for concluído e seu resultado estiver disponível, o *Event Loop* retoma a execução da função `async` logo após o `await`.

6. Qual é a diferença entre um `StreamController` e um `Stream` no Dart?

StreamController: É o componente usado para gerenciar e enviar dados para o *stream*. Ele tem métodos como `add()`, `addError()`, e `close()`. Ele é o lado do produtor.

Stream: É o objeto que os consumidores escutam. Ele representa o fluxo de dados em si e é usado apenas para consumo. Ele é o lado do consumidor.

7. Como você lida com o cancelamento de uma transmissão no Dart?

Quando você chama `stream.listen(...)`, o método retorna um objeto `StreamSubscription`. E para parar de ouvir o *stream* e liberar os recursos associados, você deve chamar o método `cancel()` nesse objeto.

8. O que é um widget `FutureBuilder` no Flutter (Dart)?

É um widget que facilita a manipulação do ciclo de vida assíncrono de um Future e a atualização da interface do usuário de acordo com o estado desse mesmo Future.

9. Você pode explicar `async*` e `yield` em Dart?

async*: É colocado antes da função para indicar que ela é um Gerador Assíncrono e que seu valor de retorno será um Stream.

yield: É usado para enviar um valor para o Stream. A função `async*` pausa no `yield` até que o próximo valor seja solicitado pelo listener do Stream.

10. Como o Dart oferece suporte à simultaneidade?

Por meio de Isolates e do Event Loop.

11. O que é o Isolate no Dart? Qual a diferença entre ele e um thread?

Um Isolate no Dart é uma unidade de execução que permite a execução de código em paralelo, semelhante a uma thread, mas com uma diferença, ele tem sua própria memória e não compartilha o espaço de memória com outros isolates onde cada Isolate opera de forma totalmente independente e isolada.

12. Como você cria exceções personalizadas no Dart?

Definindo uma nova classe que geralmente implementa a classe base `exception`.

13. Qual é a finalidade dos mixins no Dart? Dê um exemplo.

Sua finalidade é reutilizar o código de forma eficiente entre diferentes classes, sem a necessidade de uma herança tradicional

14. Como o Dart oferece suporte a conceitos de programação funcional?

Por meio de Funções como valores, Métodos funcionais, Imutabilidade, Funções puras.

15. Como você implementa fechamentos no Dart?

Um fechamento é uma função que lembra do ambiente onde foi criada. Ela consegue guardar informações e continuar usando depois, mesmo que o resto do código já tenha acabado.

```
Function multiplicador(int valor) {  
    return (int numero) => numero * valor;
```

```
}

void main() {
    var dobrar = multiplicador(2);
    var triplicar = multiplicador(3);
    print(dobrar(5));
    print(triplicar(5));
}
```

16. O que são funções de primeira classe no Dart?

Significa que as funções são tratadas como cidadãos de primeira classe, como Atribuição, Argumentos e Retorno

17. Como você lida com genéricos no Dart?

Genéricos permitem escrever código que pode funcionar com vários tipos de dados, mantendo a segurança de tipos. Eles são definidos usando o operador **<T>** em classes, interfaces e funções.

Ele meio que tem 2 propósitos, sendo **Reutilização**: Onde o mesmo código pode ser usado para Listas de int, Listas de String,etc.

Segurança: O compilador Dart garante que o código não tente misturar tipos onde não deveria.

18. Quais são as diferenças entre uma Lista e uma Fila no Dart?

Lista:

- a. É como uma prateleira onde você pode acessar qualquer posição.
- b. Você pode adicionar, remover ou pegar um item em qualquer lugar (início, meio ou fim).
- c. Usa índices (posição 0, 1, 2, ...).

Fila:

- d. É como uma fila de pessoas no mercado, quem entra primeiro, sai primeiro.
- e. Você só adiciona no fim e remove do início.

- f. Não usa índices como uma lista.
19. Como o Dart lida com coleções como List, Set e Map em termos de imutabilidade?
- O Dart oferece suporte à imutabilidade de coleções de duas maneiras:
- Tempo de Compilação:** Usando a palavra-chave `const` para criar coleções que são imutáveis desde o momento da compilação. Se você tentar modificar uma lista `const`, o Dart lançará um erro em tempo de execução.
- Tempo de Execução:** Coleções criadas com `final` (ou `var`) são referências imutáveis, mas seus conteúdos podem ser mutáveis. Para atingir a imutabilidade, a abordagem correta é criando uma nova coleção em vez de modificar a existente.
20. Como a biblioteca `dart:io` interage com arquivos no Dart?
- Elá fornece APIs para interagir com o sistema de arquivos, sockets HTTP, e outros aspectos de E/S.
21. Como você executa operações de E/S de arquivos no Dart?
- Através da biblioteca `dart:io`, utilizando a classe `File` para ler, escrever, criar e excluir arquivos. As operações de E/S são executadas de duas maneiras:
- Assíncrona: Usando Future para métodos como `readAsString()`, `writeAsString()`, ou `openRead()`
- Síncrona: Usando métodos terminados em Sync `readAsStringSync()`, `writeAsStringSync()`.
22. Você pode explicar o conceito de `typedef` no Dart?
- O `typedef` é usado para criar um apelido para uma função ou um tipo de função complexo. `typedef` também pode ser usada para dar apelidos a qualquer tipo. Melhorando a legibilidade e permitindo reutilizar a assinatura da função.
23. Como você define uma constante que é calculada em tempo de execução no Dart?

Usando a palavra-chave final, pois o valor não precisa ser conhecido na compilação, mas deve ser calculado e atribuído apenas uma vez.

24. Você pode explicar como a injeção de dependência funciona no Dart?

Ela é um padrão de design onde um objeto recebe outras classes que ele precisa para funcionar de uma fonte externa, em vez de criá-las por conta própria.

25. Qual é o significado de dart:async no Dart?

Ela é a fundação da programação assíncrona no Dart. Ela contém todas as classes e funções essenciais que lidam com operações que ocorrem ao longo do tempo.

26. Como você gerencia o estado no Flutter usando o Dart?

27. Como você implementa classes mixin no Dart?

- a. Definindo usando mixin NomeDoMixin { ... }
- b. Aplicando o mixin à classe usando a palavra-chave with.
- c. Usando restrições como on NomeDaClasse para restringir o uso do mixin apenas a classes que herdam de NomeDaClasse.

28. Como async e await funcionam quando aninhados?

Quando async e await são aninhados, eles se comportam como uma cascata de promessas (Futures) que garantem que o código seja executado em ordem lógica, mesmo que envolva múltiplas camadas assíncronas.

29. Você pode explicar como os Futures são encadeados no Dart?

Encadeamento de Futures é o processo de executar uma série de operações assíncronas em sequência, onde o resultado de uma operação é usado como entrada para a próxima.

30. Como usar Future.delayed() no Dart? Dê um exemplo.

É usado principalmente para simular atrasos (em testes ou exemplos de código) ou para agendar uma tarefa para o futuro.

```
void main() {  
  print('Iniciando...');  
  Future.delayed(Duration(seconds: 3), () {  
    print('Passaram-se 3 segundos!');  
  });  
  print('Esperando...');  
}
```

31. Como você pode executar solicitações HTTP no Dart?

São executadas usando o pacote oficial http (disponível em pub.dev) ou um pacote de terceiros mais robusto como o Dio.

32. Qual é a diferença entre os pacotes http e dio no Flutter/Dart?

HTTP é mais simples, rápido e faz o básico bem feito.

DIO já é mais avançado tendo recursos como os **Interceptores**, que permite automatizar tarefas cruciais, como adicionar um token de autenticação a cada cabeçalho de requisição.

33. O que são isolados no Dart e como eles ajudam no paralelismo?

Isolados são unidades de execução independentes no Dart, semelhantes a processos em sistemas operacionais, que permitem paralelismo real usando múltiplos núcleos de CPU.

Eles ajudam no paralelismo porque cada Isolate tem seu próprio Event Loop e sua própria memória, garantindo que o código em um Isolate possa ser executado em um núcleo de CPU diferente do Isolate principal

34. Para que serve o pacote dart:ffi no Dart?

O pacote dart:ffi é a biblioteca que permite a integração entre o código Dart e bibliotecas escritas em C.

35. Como otimizar um programa Dart para desempenho?

Evitando alocação desnecessária, usar const e final, escolha correta de coleções.

36. Como você lida com o gerenciamento de memória e coleta de lixo no Dart?

O Dart gerencia memória e coleta de lixo automaticamente.

37. Qual é o significado de typedef no Dart?

O typedef é um mecanismo no Dart para criar apelidos de tipo

38. Como você implementa um padrão de fábrica no Dart?

O Padrão de Fábrica é implementado no Dart usando o factory constructor. Ele é um construtor que não é obrigado a criar uma nova instância da sua classe. Em vez disso, ele pode retornar uma instância existente ou retornar uma instância de uma subclasse.

39. Como você usa o StreamBuilder no Flutter para manipular fluxos de dados?

O StreamBuilder serve para atualizar a interface automaticamente sempre que novos dados chegam de uma fonte contínua, usado em chat, sensor, ou banco de dados.

40. Você pode explicar o conceito do loop de eventos do Dart?

Ele é o mecanismo de concorrência do Dart que permite que a única *thread* de execução gerencie as operações assíncronas de forma não-bloqueante, monitorando duas filas principais e a *thread* principal.

Experiente (40 perguntas)

1. Como a VM Dart difere da VM JavaScript (no contexto do Flutter)?

A principal diferença é que a **VM Dart** é construída para ter um modelo de execução mais previsível, focado em manter a interface do usuário fluida, enquanto a **VM JavaScript** é mais flexível, mas historicamente teve mais problemas com bloqueio da *thread* principal.

2. Você pode explicar os detalhes internos do processo de coleta de lixo do Dart?

O coletores de lixo do Dart usa um método chamado Coleta Geracional para gerenciar a memória de forma eficiente, baseando-se no princípio de que a

maioria dos objetos morre jovem.

3. Como você usa ffi (Foreign Function Interface) no Dart para chamar código nativo?
 - a. Primeiro defino as funções em C que desejo chamar no Dart, especificando a assinatura exata usando ponteiros.
 - b. Assim o Dart carrega a biblioteca nativa usando `DynamicLibrary.open()`.
 - c. O Dart "mapeia" a função nativa para uma função Dart usando `lookupFunction`.
 - d. E por fim o Dart chama a função mapeada.
4. Quais são algumas considerações de desempenho ao trabalhar com Dart em aplicativos de nível de produção?

Para garantir que o aplicativo final seja o mais rápido possível:

- a. Compilar o código para o modo (AOT), que é o mais rápido.
 - b. Minimizar a criação de lixo. Menos lixo significa que o Coletor de Lixo trabalha menos e o aplicativo fica mais rápido.
 - c. Se você tem uma tarefa que leva mais de um segundo, use o `Isolate`.
5. Como funciona a compilação just-in-time (JIT) e ahead-of-time (AOT) do Dart?

JIT:

O código Dart é compilado enquanto o app está rodando.
É usado durante o desenvolvimento.

AOT:

O código Dart é compilado antes de rodar, gerando código nativo otimizado.

6. Qual é a diferença entre `StreamSubscription` e `StreamController` no Dart?

O `StreamController` é quem envia os dados, e o `StreamSubscription` é o o que permite de você receber os dados.

7. Como você gerencia a simultaneidade no Dart, especialmente em aplicativos Flutter?

8. O que é a biblioteca dart:ffi e como ela pode ser usada para programação em nível de sistema?

Ela é a biblioteca que permite que o Dart se conecte com código de nível de sistema, que é o código escrito em C que interage diretamente com o hardware ou o sistema operacional.

Ela pode ser usadas para chamar APIs de baixo nível do sistema operacional que não têm uma interface Dart pronta ou usada para integração com bibliotecas de alta performance onde a velocidade do código C é essencial.

9. Você pode explicar como o Dart lida com multithreading com Isolate?

Usando isolates, que são threads independentes com memória própria onde cada isolate roda seu próprio código e não compartilha variáveis com os outros. Assim eles se comunicam por mensagens usando SendPort e ReceivePort. Assim, onde o Dart consegue rodar tarefas em paralelo sem travar a interface do app Flutter.

10. Como você otimiza o desempenho do aplicativo Flutter usando o modelo de simultaneidade do Dart?

A otimização se resume a nunca bloquear o *Event Loop* da *thread* principal. Garantindo que todo o código demorado seja delegado para tarefas assíncronas e se uma função demora, ela deve retornar um Future.

11. Como você cria e usa Isolate personalizado no Dart?

Usando Isolates personalizados para mover uma tarefa que consome muita CPU para um núcleo de processador separado, garantindo que o aplicativo não congele.

12. Como você interage com o SQLite no Dart e quais pacotes você usa?

No Dart, o banco de dados SQLite é usado para armazenar dados localmente, ótimo para apps que precisam salvar informações mesmo sem internet. Utilizando o pacote sqflite

13. Como o pacote Provider funciona para gerenciamento de estado no Dart/Flutter?

O Provider gerencia o estado no Flutter permitindo compartilhar dados entre widgets e reagir automaticamente a mudanças, sem precisar passar variáveis

manualmente.

14. Quais são os casos de uso comuns da classe Stream no Dart?

- a. Dados em Tempo Real
- b. *Eventos de Interface do Usuário*
- c. *Gerenciamento de Estado*
- d. Leitura Contínua

15. Você pode descrever como implementaria um widget personalizado no Flutter usando Dart?

Criaria uma classe que herda de StatelessWidget, definiria como ela deve aparecer no método build() e assim usaria ela normalmente em outros lugares da sua interface.

16. Como otimizar o gerenciamento de memória em aplicativos Dart e Flutter?

Cancelaria recursos assíncronos e utilizaria imutabilidade, variáveis estáticas e DevTools.

17. Você pode explicar o conceito de dart:html e como o Dart interage com a web?

O dart html é uma biblioteca interna do Dart usada quando o código roda no navegador. Ela permite que você interaja diretamente com a página da web, com o DOM, eventos do navegador, cookies, e requisições HTTP.

18. O que é um StreamTransformer no Dart e como usá-lo?

Um StreamTransformer é uma classe que permite modificar ou filtrar os dados que fluem por um Stream de forma reutilizável. Pegando um *stream* de entrada e o transformando em um *stream* de saída.

Você o usa aplicando ao *stream* de entrada usando o método .transform().

19. Como o Dart lida com a inferência de tipos e quando você deve declarar tipos explicitamente?

O Dart usa inferência de tipos para descobrir automaticamente o tipo de uma variável com base no valor que você atribui a ela. Mas você também pode declarar o tipo explicitamente quando quiser deixar o código mais claro ou evitar confusões.

Pode declarar quando você quer deixar o código mais legível.
Quando precisa de mais controle sobre o tipo, como listas ou funções.
Quando não há valor inicial.

20. Como escrever um teste unitário para código assíncrono no Dart?

Use a palavra-chave `async` na sua função `test()`, depois use `await` para esperar a conclusão de `Futures` ou outros códigos assíncronos dentro do seu teste.

21. Você pode explicar como funciona a segurança nula do Dart e como lidar com tipos anuláveis?

É um sistema que visa eliminar o erro de objeto nulo antes de o código rodar. Por padrão, variáveis não podem ser null. Ou seja, se você declara uma variável assim: `int numero = 5;` ela nunca poderá conter null

Se quiser permitir que uma variável possa ser nula, você precisa declarar explicitamente com `?` após o tipo, por ex: `int? numero = null;`

22. Como o Dart gerencia a compilação e a implantação de aplicativos móveis (Flutter)?

O Dart usa a compilação **AOT para aplicativos móveis**.

- a. **Compilação AOT:** Quando você executa `flutter build apk` ou `flutter build ios`, o código Dart é traduzido diretamente para o código de máquina nativo.
- b. **Implantação :** O código de máquina é empacotado junto com o motor de renderização do Flutter em um arquivo binário.

23. Como o Dart lida com fechamentos e escopo lexical?

- a. Se a função é um fechamento ela "fecha" as variáveis que estavam no escopo onde ela foi criada, mesmo que a função original já tenha terminado de rodar.
- b. Escopo Lexical significa que o escopo de uma variável é determinado pela localização do código no momento da escrita, e não pelo local onde a função é chamada.

24. O que são métodos de extensão no Dart e quando você deve usá-los?

Métodos de Extensão permitem que você adicione novas funcionalidades a uma classe já existente. Assim melhorar a legibilidade, podendo adicionar métodos úteis a tipos do Dart como converter uma String em DateTime de

forma fácil.

25. Como você implementa Injeção de Dependência (DI) no Dart?
26. Você pode explicar como usar dart:async para tratamento de erros em fluxos e futuros?

27. Como você abordaria a depuração de gargalos de desempenho em aplicativos Dart?

Para achar e corrigir gargalos de desempenho em Dart, eu usaria o DevTools para medir o que está lento, analisando reconstruções desnecessárias, movendo tarefas pesadas para fora da UI.

28. Você pode explicar o ciclo de vida Stream e Future e como lidar com cenários assíncronos complexos no Dart?
 - a. **Future**: executa uma tarefa assíncrona uma vez e retorna um resultado no futuro.
 - b. **Stream**: envia vários resultados ao longo do tempo.
 - c. Em cenários complexos, combina múltiplos Futures com Future.wait() e Streams com pacotes como rx.dart.
29. Quais são as melhores práticas para lidar com erros em funções assíncronas do Dart?

Usar **try/catch** com **await**, tratar **onError** em **Streams** e fornecer mensagens claras, assim, o código assíncrono fica seguro, estável e fácil de depurar.

30. Como você otimizaria o código Dart para aplicativos de larga escala?
31. Você pode descrever uma ocasião em que teve que depurar um problema complexo no Dart?
32. Como você gerencia e versiona suas dependências do Dart usando pubspec.yaml?

Eu gerencio as dependências do projeto por meio do arquivo pubspec.yaml, onde defino e versiono todos os pacotes utilizados. Utilizo comandos como flutter pub get para instalar e flutter pub upgrade para manter as versões atualizadas, assim garantindo compatibilidade e estabilidade no projeto.

33. Como você lida com internacionalização e localização no Flutter/Dart?

Com Internacionalização (i18n) para preparar o app para vários idiomas e **Localização (l10n)** para traduzir e adaptar o conteúdo (textos, datas, moedas) para cada idioma.

34. Você pode explicar o conceito de rastreamentos de pilha assíncronos no Dart e como lidar com eles?

Um rastreamento de pilha mostra a sequência de funções síncronas que levaram a um erro. Já o rastreamento de pilha é assíncrono e mais complexo, pois mostra a conexão lógica das funções assíncronas que se estenderam no tempo.

35. Como você configura sistemas de compilação personalizados no Dart?

Configurando sistemas de compilação personalizados usando o `build_runner` e arquivos `build.yaml`, definindo como o código deve ser gerado ou transformado

36. Qual é a função de `Future.wait()` no Dart e como ele funciona?

A função `Future.wait()` é usada para executar múltiplos `Futures` em paralelo e esperar que todos eles sejam concluídos

Ele receberia uma lista de `Futures`, logo depois o Dart iniciaria todos os `Futures` imediatamente, assim o `Future.wait()` retorna um `Future` único que é concluído com uma lista de resultados

37. Você pode explicar como integrar bibliotecas nativas de terceiros com o Dart em um aplicativo Flutter?

A integração de bibliotecas nativas é feita usando o Platform Channels, onde você define um **canal de comunicação** nomeado no código Dart e no código nativo.

Assim, o código Dart envia uma mensagem (chamada de método) através do canal. Então o código nativo (Kotlin/Swift) escuta o canal, executando a função nativa e enviando o resultado de volta para o Dart.

38. Como você lidaria com vazamentos de memória em aplicativos Dart/Flutter?

Resolveria encontrando e quebrando referências longas a objetos que deveriam ter morrido.

39. Como o loop de eventos do Dart funciona na prática e como ele pode ser otimizado para aplicativos de alto desempenho?

O Dart roda em um único thread principal, chamado Isolate principal, que tem um event loop, onde ele gerencia tarefas assíncronas e fila de eventos para que o app nunca trave, mesmo executando várias operações ao mesmo tempo.

Para otimizar, a melhor opção é evitar tarefas pesadas na thread principal e dividir tarefas grandes em partes menores.

40. Como você implementa animações complexas no Flutter usando Dart?

Para animações mais avançadas, o ideal é combinar múltiplos controladores, usando CurvedAnimation e o pacote flutter_sequence_animation.

1. Por que o Flutter é preferido em relação a outras ferramentas de desenvolvimento de aplicativos móveis?

Porque ele permite criar aplicativos nativos para Android, iOS, Web e Desktop com um único código, tendo alta performance que agiliza o desenvolvimento.

2. O que são pacotes e plugins no Flutter?

Pacotes: adicionam funcionalidades puramente em Dart (ex: manipular datas).

Plugins: têm código nativo Android e iOS e permite acessar recursos do dispositivo.

3. Quais são as limitações do Flutter?

Algumas integrações nativas exigiriam código adicional em Kotlin/Swift.

Bibliotecas novas às vezes não têm suporte oficial.

4. Por que geralmente leva muito tempo para desenvolver um aplicativo Flutter?

Criar interfaces complexas manualmente levam tempo além disso, cada build compilação AOT é pesado e leva mais tempo para gerar versões de produção.

5. O que são chaves e como usá-las?

As Keys identificam widgets de forma única, sendo importantes quando o Flutter reconstrói a tela, pois ajudam a preservar o estado dos widgets

6. Quais são os diferentes tipos de fluxos no Dart?

Single-subscription Stream: envia dados apenas para um ouvinte.

Broadcast Stream: pode enviar dados para vários ouvintes ao mesmo tempo.

7. O que é o arquivo pubspec.yaml?

É o arquivo de configuração do projeto Flutter/Dart

8. O que é Flutter?

O Flutter é um framework criado pelo Google, onde ele permite criar aplicativos nativos multiplataforma usando apenas a linguagem Dart.

9. O Flutter é desenvolvido por qual empresa?

Google

10. Qual foi a primeira versão do Flutter que rodou em um sistema operacional Android?

Flutter Alpha v0.0.6

11. Você pode nos contar os quatro elementos principais do Flutter?

1. **Widgets:** blocos de construção da interface.
2. **Rendering:** desenha os widgets na tela.
3. **Framework:** fornece as ferramentas e classes para o app funcionar.
4. **Engine:** usa o **Skia** para renderizar tudo de forma rápida e nativa

12. Quando foi lançada a primeira versão do Flutter?

Maio de 2017

13. Flutter é escrito em qual idioma?

Dart.

14. O que significa a sigla SDK?

Software Development Kit

15. Você pode citar alguns dos melhores editores para desenvolvimento do Flutter?

Visual Studio Code, Android Studio e IntelliJ IDEA

16. Qual operador é usado para avaliar e retornar valores entre duas expressões?

O operador ternário (? :)

17. Você pode nos dizer quantos tipos de widgets existem no Flutter?

Dois tipos principais:

1. **StatelessWidget:** Não muda (fixo).
2. **StatefulWidget:** Muda com o tempo (dinâmico).

18. Você pode nomear o comando usado para compilar o modo de lançamento?

```
flutter run --release
```

19. Nomeie os diferentes tipos de modos de construção no Flutter.

Debug: Usado no desenvolvimento.

Profile: Mede desempenho e uso de recursos.

Release: Versão final otimizada para produção.

20. Qual widget no Flutter é uma caixa que vem com um tamanho diferente?

SizedBox.

21. Suponha que você precise representar um comportamento do mundo real no Flutter. Qual animação você usaria?

Animação implícita, pois ela simula comportamentos naturais, como suavidade e movimento fluido.

22. Cite alguns aplicativos populares que usam o Flutter.

Google Ads, Alibaba, BMW, eBay Motors, Nubank.

23. Qual pasta é usada para escrever aplicativos Android no Flutter?

A pasta android

24. Qual é a utilidade da função Await?

O await pausa a execução até que um Future termine, sem travar o resto do programa.

25. Você pode nos dizer qual função compila e atualiza o aplicativo?

setState()

26. Você consegue dizer qual função é responsável por iniciar o programa?

main().

27. Qual é o propósito da classe controladora de animação no Flutter?

Controlar o tempo, velocidade e o ciclo de uma animação, onde ela define quanto tempo a animação dura e como ela se repete ou reverte.

28. Como você pode testar um único widget?

Usando o pacote `flutter_test` e o método `testWidgets()`

29. É possível usar o `WidgetsApp` para navegação básica?

Sim, ele permite navegação e rotas simples, mas sem temas ou estilos prontos.

30. Qual widget nos permite atualizar a tela?

`StatefulWidget`, ele tem o método `setState()`, que permite atualizar a interface quando o estado muda.

31. Você pode citar alguns exemplos de widgets sem estado?

`Text`, `Icon`, `Image`, `Column`, `Container`

32. Explique o termo “Tree shaking” no Flutter.

É o processo de remover código não utilizado durante a compilação

33. O que é usado para importar pacotes para seu projeto?

A palavra-chave `import`.

34. Qual é a utilidade do `Navigation.push` no Flutter?

Ele abre uma nova tela (rota).

35. Por que o pacote `HTTP` é usado no Flutter?

Para fazer requisições de rede e enviar e receber dados de APIs.

36. Explique o modo de perfil no Flutter?

O Profile Mode é usado para analisar o desempenho do app.

37. Qual é o papel do `BuildContext` no Flutter?

Ele conecta o widget à árvore de widgets do app, assim permitindo acessar temas, navegação, heranças e dados.

38. Você pode nos dizer qual classe é responsável por implementar a estrutura básica de layout visual do Material Design para um aplicativo no Flutter?

Scaffold.

39. Você pode nos dizer qual será o tipo de retorno padrão, se você não especificar o tipo de retorno para uma função?

O tipo dynamic ou void, se não retornar nada.

40. Qual é o nome de um construtor para uma classe no Flutter?

Ele tem o mesmo nome da classe.

41. Qual é o núcleo do mecanismo de layout do Flutter?

É o mecanismo de renderização com a biblioteca gráfica Skia, que permite que o Flutter desenhe pixels diretamente e controle o layout da interface.

42. Qual widget pode ser usado para exibir um após o outro em formato de rolagem?

Um exemplo é o widget ListView

43. O que é um layout no Flutter?

É a forma como widgets são organizados e posicionados na tela, seria o formato visual dos componentes.

44. Você pode nos dizer como podemos testar um único widget no Flutter?

Usando testWidgets() no pacote flutter_test, lá você monta o widget isoladamente, usa await tester.pumpWidget(...), e depois verifica comportamentos ou visualizações.

45. Qual é a versão mais recente do Flutter?

A versão mais recente é 3.35

46. O Flutter é um framework front-end ou back-end?

É um framework front-end.

47. Qual é a função do futuro em Dart?

Um Future representa uma operação assíncrona que vai ser completa uma vez no

futuro, retornando um valor ou um erro.

48. Qual é a função do construtor image.network() no Flutter?

Ele cria um widget que carrega e exibe uma imagem da internet

49. Você pode nos contar o que sabe sobre Rune In Dart?

50. O Flutter é gratuito?

Sim

51. Por que usamos a palavra-chave const no Flutter?

Porque ela cria widgets imutáveis em tempo de compilação.

52. Quais são alguns operadores comumente usados no Dart?

+, -, *, /, %, ==, !=, >, <

53. Por que usamos um ticker no Flutter?

Porque ele mantém o AnimationController sincronizado com o tempo da tela.

54. Como você pode reduzir o tempo de execução do código?

Evitando loops desnecessários, usando async/await corretamente, dividindo tarefas pesadas em Isolates e reutilizando widgets com const

55. O que é Dart?

É uma linguagem de programação usada para criar aplicações web, mobile e desktop.

56. Qual processo você usará para reduzir a reconstrução do widget?

Usar widgets const, Provider ou ValueNotifier para atualizar apenas partes necessárias.

57. Defina widget espaçador?

É um widget usado para criar espaço flexível entre elementos dentro de um layout.