1. Explique a diferença entre as palavras-chave var, dynamic, final, e const em Dart. Em quais situações seria mais adequado utilizar cada uma?

R = **var**: Quando você quer que o Dart infira o tipo, mas a variável pode ser reatribuída.

**dynamic**: Quando você precisa de uma variável que pode ter tipos diferentes ao longo do tempo.

**final**: Quando você quer uma variável que não pode ser reatribuída, mas pode ser inicializada com um valor que não é necessariamente uma constante em tempo de compilação.

**const**: Quando você precisa de uma constante que é determinada em tempo de compilação e não pode ser alterada.

2) Com base no Princípio da Responsabilidade Única (Single Responsibility Principle) do SOLID, descreva um cenário onde uma classe viola esse princípio.

Como você refatoraria o código para que ele seguisse o SRP corretamente?

R= class User {

String username;

String password;

User(this.username, this.password);

}

class Authenticator {

User user;

Authenticator(this.user);

bool authenticate() {

// Lógica de autenticação

return user.username == 'admin' && user.password == '1234';

}

}

class Notifier {

User user;

Notifier(this.user);

void sendNotification(String message) {

// Lógica para enviar uma notificação

print('Notification to ${user.username}: $message');

}

}  
3) Descreva o que é o Princípio da Inversão de Dependências (Dependency Inversion Principle) e explique como ele pode ajudar na manutenção e flexibilidade de um sistema. Dê um exemplo prático de sua aplicação em Dart.

R = O Princípio da Inversão de Dependências (Dependency Inversion Principle - DIP) é um dos princípios SOLID que enfatiza a importância de dependências entre módulos. O DIP afirma que:

**Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.**

**Abstrações não devem depender de detalhes. Detalhes devem depender de abstrações.**

**Exemplo em Código Dart:**

**abstract class PaymentMethod {**

**void pay(String identifier, double amount);**

**}**

**class PayPal implements PaymentMethod {**

**@override**

**void pay(String email, double amount) {**

**print('Paying $amount using PayPal with email: $email');**

**}**

**}**

**class CreditCard implements PaymentMethod {**

**@override**

**void pay(String cardNumber, double amount) {**

**print('Paying $amount using Credit Card with number: $cardNumber');**

**}**

**}**

**class Checkout {**

**PaymentMethod paymentMethod;**

**Checkout(this.paymentMethod); // Dependência injetada via construtor**

**void checkout(String identifier, double amount) {**

**paymentMethod.pay(identifier, amount);**

**}**

**}**

4) Explique como funcionam as estruturas de controle condicionais (if, else, switch) em Dart. Qual é a principal diferença entre o uso do if/else e do switch?

**R=**

**Exemplo em código:**

**if (idade < 13) {**

**print('Você é uma criança.');**

**} else if (idade < 18) {**

**print('Você é um adolescente.');**

**} else {**

**print('Você é um adulto.');**

**}**

**String diaDaSemana = 'sábado';**

**switch (diaDaSemana) {**

**case 'segunda':**

**print('Dia de trabalho.');**

**break;**

**case 'sábado':**

**case 'domingo':**

**print('Final de semana.');**

**break;**

**default:**

**print('Dia inválido.');**

**}**

A estrutura if avalia uma condição booleana e, se a condição for verdadeira, executa um bloco de código. O bloco else pode ser usado para definir um código a ser executado quando a condição do if é falsa.

A estrutura switch permite que você teste uma variável contra múltiplos valores possíveis, cada um definido em um caso. É uma alternativa ao uso de múltiplos if e else if para comparar a mesma variável com diferentes valores.

5) O que são interfaces em programação orientada a objetos? Como Dart lida com a implementação de interfaces e qual é a diferença entre interfaces e herança de classes?

R=

Interfaces definem um conjunto de métodos que devem ser implementados, promovendo um comportamento consistente entre classes.

Herança permite a reutilização de código e a definição de uma hierarquia de classes, incluindo implementação e características compartilhadas.

Dart trata interfaces como qualquer classe, e você pode implementar várias interfaces, enquanto a herança é restrita a uma única superclasse.

6) Compare as coleções List e Set em Dart. Em que situações o uso de um Set é mais apropriado do que o uso de uma List?

R=

**Use List**: Quando a ordem é importante e quando você pode ter duplicatas.

**Use Set**: Quando a unicidade dos elementos é necessária, quando você precisa realizar operações de conjunto ou quando a ordem não importa e você precisa de verificações rápidas de existência.

Exemplo em código:

List<int> numeros = [1, 2, 3, 4, 4, 5]; // Duplicatas permitidas

print(numeros[2]); // Acessa o terceiro elemento: 3

Set<int> numerosUnicos = {1, 2, 3, 4, 4, 5}; // Duplicatas são removidas

print(numerosUnicos); // Saída: {1, 2, 3, 4, 5}

7) Explique o funcionamento da estrutura de repetição for em Dart. Dê um exemplo de um laço for que percorre uma lista e imprima os números pares.

R= A estrutura de repetição for em Dart é usada para executar um bloco de código várias vezes, geralmente com base em um contador ou em uma condição.

Exemplo em código:

void main() {

List<int> numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

for (int i = 0; i < numeros.length; i++) {

if (numeros[i] % 2 == 0) { // Verifica se o número é par

print(numeros[i]); // Imprime o número par

}

}

}

8) No contexto do Git, descreva o que acontece quando você executa o comando git pull. Quais os possíveis conflitos que podem surgir e como resolvê-los?

R= O comando git pull traz atualizações de um repositório remoto e tenta mesclá-las à sua branch local. Conflitos podem surgir quando há alterações conflitantes nos mesmos arquivos. Resolvendo conflitos envolve identificar os arquivos em conflito, editar o código para resolver as divergências, adicionar os arquivos resolvidos e finalizar o merge.

9) O que é o conceito de herança em programação orientada a objetos? Explique como a herança pode ser usada em Dart, mencionando vantagens e desvantagens.

R= A herança é um conceito fundamental em programação orientada a objetos que permite a criação de hierarquias de classes e a reutilização de código. Em Dart, a herança é implementada com a palavra-chave extends, permitindo que classes derivadas herdem e sobrescrevam métodos da superclasse. Embora traga vantagens significativas, como reutilização e organização do código, também apresenta desvantagens, como acoplamento e complexidade. É importante usar herança de maneira consciente para manter a clareza e a manutenção do código.

10) Explique como o compilador Just-In-Time (JIT) e Ahead-of-Time (AOT) do Dart diferem em termos de desempenho. Como o Flutter utiliza essas técnicas para otimizar aplicativos mobile?

R=

* **JIT**: Ideal para desenvolvimento, oferecendo rápida iteração e suporte a hot reload, mas com desempenho inferior em produção.
* **AOT**: Melhor para a execução em produção, proporcionando um desempenho mais rápido e otimizado, mas com tempos de compilação mais longos e binários maiores.

A combinação de JIT e AOT permite que o Flutter ofereça uma experiência de desenvolvimento eficiente e ao mesmo tempo garanta que os aplicativos finais sejam rápidos e responsivos.