Questão 1: Na linguagem Dart, a compreensão dos conceitos de imutabilidade é crucial para o desenvolvimento de código seguro e eficiente. Variáveis declaradas com as palavras-chave final e const desempenham um papel importante na garantia de que certos valores não serão alterados durante a execução do programa.  
Explique a diferença entre as palavras-chave final e const em Dart. Em que situações é mais apropriado usar cada uma delas? Forneça exemplos de código para ilustrar suas explicações.

**O que é:** A palavra-chave final é usada para declarar uma variável cujo valor é atribuído uma única vez e não pode ser alterado após a atribuição inicial.

Ex: void main() {

final currentTime = DateTime.now(); // Atribuição dinâmica baseada no tempo atual

print(currentTime);

final list = [1, 2, 3];

list.add(4); // Modificação do conteúdo da lista é permitida, mas não da referência

print(list);

}

**O que é:** A palavra-chave const é usada para declarar variáveis cujo valor é constante e conhecido em tempo de compilação. Variáveis const são avaliadas e inicializadas no momento da compilação, o que pode levar a um código mais eficiente e a uma memória mais conservadora.

Ex: void main() {

const pi = 3.14159; // Valor constante conhecido em tempo de compilação

print(pi);

const numbers = [1, 2, 3];

// numbers.add(4); // Isso causará erro, pois 'numbers' é uma constante

const Map<String, int> ages = {

'Alice': 30,

'Bob': 25

};

print(ages);

}

Questão 2: As estruturas condicionais if/else são amplamente utilizadas na programação para permitir que o código tome decisões com base em condições específicas. No Dart, a sintaxe dessas estruturas é clara e direta, mas é importante saber como e quando utilizá-las de forma eficaz.  
Explique como a estrutura if/else funciona em Dart. Dê um exemplo de código onde essa estrutura seja utilizada para verificar se um número é positivo, negativo ou zero, e imprima uma mensagem correspondente para cada caso.

No Dart, a estrutura condicional if/else permite que você tome decisões baseadas em condições específicas. A sintaxe é clara e direta, e você pode usar essa estrutura para executar blocos de código diferentes dependendo do resultado de uma expressão booleana.

void main() {

int numero = -5; // Você pode mudar esse valor para testar diferentes casos

if (numero > 0) {

print('O número é positivo.');

} else if (numero < 0) {

print('O número é negativo.');

} else {

print('O número é zero.');

}

}

Questão 3: Em Dart, as classes e os objetos permitem a modelagem de conceitos do mundo real em código, utilizando a programação orientada a objetos. A herança e a composição são duas formas de reutilizar código e criar relações entre classes.  
Compare e contraste os conceitos de herança e composição em Dart. Em que situações cada um desses conceitos deve ser utilizado? Dê exemplos de código que demonstrem a aplicação de ambos.

Herança é um mecanismo que permite que uma classe (chamada de **classe derivada** ou **subclasse**) herde características (métodos e atributos) de outra classe (chamada de **classe base** ou **superclasse**).

Ex: class Animal {

void comer() {

print('O animal está comendo.');

}

}

class Gato extends Animal {

void miar() {

print('O gato está miando.');

}

}

void main() {

Gato meuGato = Gato();

meuGato.comer(); // Método herdado da classe Animal

meuGato.miar(); // Método específico da classe Gato

}

**Composição** é um padrão de design onde uma classe é composta por outras classes. Em vez de herdar de uma classe base, você inclui uma instância de outra classe dentro da sua classe.

Ex: class Motor {

void ligar() {

print('O motor está ligado.');

}

}

class Carro {

final Motor motor;

Carro(this.motor);

void ligarCarro() {

motor.ligar(); // Usando a composição para acessar a funcionalidade do Motor

print('O carro está ligado.');

}

}

void main() {

Motor meuMotor = Motor();

Carro meuCarro = Carro(meuMotor);

meuCarro.ligarCarro(); // Chama o método que usa a composição

}

Questão 4: A estrutura switch/case em Dart é usada para simplificar a seleção de um bloco de código a ser executado com base no valor de uma expressão. Ela pode ser uma alternativa mais organizada e eficiente ao uso de múltiplos if/else quando há muitas condições a serem avaliadas.  
Descreva como a estrutura switch/case funciona em Dart. Em que situações essa estrutura é mais indicada do que if/else? Crie um exemplo de código que use switch/case para avaliar o valor de uma variável que representa um dia da semana e imprima uma mensagem específica para cada dia.

A estrutura switch/case permite que você verifique o valor de uma variável contra várias constantes e execute blocos de código diferentes dependendo da correspondência. A sintaxe básica é a seguinte:

switch (variável) {

case valor1:

// Código executado se variável == valor1

break;

case valor2:

// Código executado se variável == valor2

break;

// Você pode ter quantos casos precisar

default:

// Código executado se nenhum dos casos acima corresponder

}

### Situações em que switch/case é Mais Indicado do que if/else

* **Número de Condições:** Quando você tem várias condições que dependem do valor de uma única variável, o switch/case pode ser mais legível e organizado do que uma série de instruções if/else.
* **Valores Constantes:** O switch/case é ideal quando você está comparando valores constantes ou enumerados. Isso pode incluir dias da semana, meses do ano, opções de menu, etc.
* **Legibilidade:** Para muitas condições com a mesma variável, o switch/case melhora a clareza e a estrutura do código.

Exemplo de Código com switch/case:

void main() {

String diaDaSemana = 'quarta-feira'; // Você pode mudar esse valor para testar diferentes dias

switch (diaDaSemana) {

case 'segunda-feira':

print('Início da semana de trabalho!');

break;

case 'terça-feira':

print('Dia de reuniões.');

break;

case 'quarta-feira':

print('Dia de revisão de projetos.');

break;

case 'quinta-feira':

print('Preparação para o fim de semana.');

break;

case 'sexta-feira':

print('Finalmente sexta-feira!');

break;

case 'sábado':

print('Dia de relaxar.');

break;

case 'domingo':

print('Dia de descansar e se preparar para a semana.');

break;

default:

print('Dia da semana inválido.');

}

}

Questão 5: As coleções, como listas e mapas, são frequentemente utilizadas em Dart para armazenar e manipular conjuntos de dados. A manipulação eficiente dessas coleções é fundamental para o desempenho de uma aplicação.  
Explique a diferença entre uma lista (List) e um mapa (Map) em Dart. Em que situações cada um deve ser utilizado? Forneça exemplos de código que demonstrem a criação e manipulação de uma lista e de um mapa.

List:

*  **Descrição**: Uma List é uma coleção ordenada de elementos, onde cada elemento tem um índice numérico (começando do zero). Os elementos podem ser acessados e manipulados usando esses índices.
* **Acesso**: O acesso aos elementos é feito através do índice.
* **Ordenação**: Mantém a ordem dos elementos conforme foram inseridos.
* **Uso Típico**: Ideal para situações em que a ordem dos elementos é importante, e você precisa de acesso rápido a elementos pela posição.

**Ex**: void main() {

// Criação de uma lista

List<String> frutas = ['maçã', 'banana', 'laranja'];

// Acesso e impressão de um elemento específico

print(frutas[1]); // Imprime 'banana'

// Adicionar um novo elemento

frutas.add('uva');

print(frutas); // Imprime ['maçã', 'banana', 'laranja', 'uva']

// Remover um elemento

frutas.remove('banana');

print(frutas); // Imprime ['maçã', 'laranja', 'uva']

// Iterar sobre os elementos

for (var fruta in frutas) {

print(fruta);

}

}

 **Map**:

 **Descrição**: Um Map é uma coleção de pares chave-valor. Cada valor é associado a uma chave única, e os valores podem ser acessados e manipulados usando essas chaves.

 **Acesso**: O acesso aos valores é feito através das chaves.

 **Ordenação**: Não garante a ordem dos elementos. Em vez disso, os elementos são armazenados de acordo com a estrutura interna do mapa.

 **Uso Típico**: Ideal para situações em que você precisa associar valores a chaves únicas e acessar dados rapidamente através dessas chaves.

Ex: void main() {

// Criação de um mapa

Map<String, int> idades = {

'Alice': 30,

'Bob': 25,

'Carlos': 28

};

// Acesso e impressão de um valor específico usando a chave

print(idades['Bob']); // Imprime 25

// Adicionar um novo par chave-valor

idades['Diana'] = 32;

print(idades); // Imprime {Alice: 30, Bob: 25, Carlos: 28, Diana: 32}

// Remover um par chave-valor

idades.remove('Carlos');

print(idades); // Imprime {Alice: 30, Bob: 25, Diana: 32}

// Iterar sobre os pares chave-valor

idades.forEach((nome, idade) {

print('$nome tem $idade anos');

});

}

Questão 6: Estruturas de repetição são fundamentais para automatizar tarefas repetitivas na programação. Em Dart, as estruturas for, while e do/while permitem iterar sobre coleções ou executar blocos de código repetidamente até que uma condição específica seja satisfeita.  
Explique as diferenças entre as estruturas de repetição for, while, e do/while em Dart. Em que situações cada uma delas é mais adequada? Dê exemplos de código que utilizem cada uma dessas estruturas para iterar sobre uma lista de números e imprimir apenas aqueles que são maiores que 50.

A estrutura for é usada quando você sabe de antemão quantas vezes deseja repetir um bloco de código. É útil para iteração sobre uma faixa de valores ou para acessar elementos de uma coleção usando índices.

Ex: void main() {

List<int> numeros = [45, 67, 23, 89, 12, 55];

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

if (numeros[i] > 50) {

print(numeros[i]);

}

}

}

A estrutura while executa um bloco de código enquanto a condição especificada for verdadeira. É usada quando você não sabe de antemão quantas iterações serão necessárias e deseja que o loop continue até que uma condição seja satisfeita.

Ex: void main() {

List<int> numeros = [45, 67, 23, 89, 12, 55];

int i = 0;

while (i < numeros.length) {

if (numeros[i] > 50) {

print(numeros[i]);

}

i++;

}

}

A estrutura do/while é semelhante ao while, mas a diferença é que o bloco de código é executado pelo menos uma vez antes de verificar a condição. É útil quando você deseja garantir que o bloco de código seja executado pelo menos uma vez.

EX: void main() {

List<int> numeros = [45, 67, 23, 89, 12, 55];

int i = 0;

do {

if (numeros[i] > 50) {

print(numeros[i]);

}

i++;

} while (i < numeros.length);

}

Questão 7: Dado um array de inteiros em Dart, escreva um algoritmo que calcule e imprima a soma de todos os números ímpares contidos nesse array. Utilize a estrutura de repetição for para iterar pelos elementos do array.  
void main() {  
  List<int> numeros = [10, 15, 20, 25, 30, 35, 40, 45, 50, 55];  
  // sua implementação aqui  
}

void main() {

List<int> numeros = [10, 15, 20, 25, 30, 35, 40, 45, 50, 55];

int somaImpares = 0; // Variável para armazenar a soma dos números ímpares

// Iterando sobre cada elemento do array usando a estrutura de repetição for

for (int numero in numeros) {

if (numero % 2 != 0) { // Verifica se o número é ímpar

somaImpares += numero; // Adiciona o número ímpar à soma

}

}

// Imprime a soma dos números ímpares

print('A soma dos números ímpares é: $somaImpares');

}

Questão 8: Em Dart, a verificação de palíndromos é uma tarefa comum em muitos problemas de programação. Um palíndromo é uma palavra, número ou qualquer outra sequência de caracteres que lida da mesma forma de trás para frente.  
Escreva um algoritmo em Dart que verifique se uma string dada é um palíndromo. O algoritmo deve ignorar espaços e diferenças entre letras maiúsculas e minúsculas.  
  
void main() {  
  String palavra = "Apos a sopa";  
  // sua implementação aqui  
}

void main() {

String palavra = "Apos a sopa";

// Função para verificar se uma string é um palíndromo

bool ehPalindromo(String str) {

// Remove espaços e converte para minúsculas

String normalizada = str.replaceAll(' ', '').toLowerCase();

// Cria a versão invertida da string normalizada

String invertida = normalizada.split('').reversed.join('');

// Verifica se a string normalizada é igual à sua inversa

return normalizada == invertida;

}

// Verifica se a palavra é um palíndromo e imprime o resultado

if (ehPalindromo(palavra)) {

print('A string "$palavra" é um palíndromo.');

} else {

print('A string "$palavra" não é um palíndromo.');

}

}