Exercício 1: Variáveis e Tipos

1. Crie um novo projeto chamado tipos_basicos usando cargo.
2. Declare três variáveis com os seguintes tipos:

- Uma i32 chamada idade com o valor 25.
- Um f64 chamado peso com o valor 70.5.
- Uma bool chamada ativo com o valor true.

3. Imprima os valores dessas variáveis usando println!.
4. Adicione a anotação de tipo explicitamente em cada variável.
5. Modifique a variável idade para ser mutável e atribua a ela um novo valor.

Exercício 2: Escopo de Variáveis

1. Crie um novo projeto chamado escopo.
2. No main(), declare uma variável x com o valor 10.
3. Dentro de um bloco ({}), declare uma nova variável x com o valor 20 e imprima seu valor.
4. Fora do bloco, imprima o valor da variável x original.
5. Explique o que acontece com as variáveis dentro e fora do bloco.

Exercício 3: Shadows de Variáveis

1. Crie um novo projeto chamado shadowing.
2. No main(), declare uma variável chamada num com o valor 5.
3. Faça um shadowing da variável num para aumentar seu valor em 3.

4. Em seguida, faça outro shadowing para a variável num multiplicando seu valor por 2.

5. Imprima o valor final de num.

6. Tente usar mut em vez de shadowing e observe as diferenças.

Exercício 4: Constantes e Escopo de Módulo

1. Crie um novo projeto chamado constantes.

2. Declare duas constantes fora do main():

- PI com valor 3.1415 (f64)
- GRAVIDADE com valor 9.8 (f32)

3. No main(), declare uma variável raio com o valor 10.0 e calcule a área de um círculo usando a fórmula PI * raio * raio.

4. Imprima o resultado da área.

5. Tente alterar o valor de uma das constantes dentro da função main() e veja o erro gerado.

Exercício 5: Segurança de Memória com Variáveis Imutáveis

1. Crie um projeto chamado seguranca_memoria.

2. No main(), declare uma variável dados do tipo String com o valor "Segurança".

3. Passe essa variável para uma nova função chamada imprimir_dados que imprime o valor.

4. Tente usar dados novamente após a chamada da função. O que acontece? Corrija o erro usando uma referência (&).

Exercício 6: Funções Simples

1. Crie um novo projeto chamado funcoes_basicas.
2. Declare uma função chamada soma que recebe dois parâmetros do tipo i32 e retorna a soma deles.
3. No main(), chame a função soma com dois números e imprima o resultado.
4. Altere a função soma para que o tipo de retorno seja i64 e veja como isso afeta a chamada da função.

Exercício 7: Funções com Retorno Antecipado

1. Crie um projeto chamado retorno_antecipado.
2. No main(), declare uma função chamada divisao_segura que recebe dois i32 e retorna um Option<i32>. Se o divisor for zero, retorne None, caso contrário, retorne Some com o valor da divisão.
3. No main(), teste a função com diferentes valores e trate o retorno com match.

Exercício 8: Referências e Empréstimo

1. Crie um projeto chamado emprestimo_referencia.
2. No main(), declare uma função chamada calcular_dobro que recebe uma referência para um i32 e retorna o dobro do valor.
3. No main(), declare uma variável numero com valor 7 e passe-a para a função calcular_dobro.
4. Imprima o resultado, explicando como funciona o empréstimo de variáveis em Rust.

Exercício 9: Tipos Opcionais e Tratamento de Erros

1. Crie um projeto chamado opcionais.

2. No main(), declare uma função busca_numero que recebe uma referência para um vetor de i32 e um número para buscar.

3. A função deve retornar Option<usize> indicando a posição do número no vetor, ou None se o número não for encontrado.

4. Teste a função no main() com diferentes entradas e trate o Option com match.

Exercício 10: Mutabilidade e Segurança de Memória

1. Crie um projeto chamado mutabilidade.

2. No main(), declare uma variável lista como uma Vec<i32> e adicione alguns elementos.

3. Crie uma função chamada adicionar_numero que recebe uma referência mutável para o vetor e adiciona um número ao final.

4. Chame a função no main() e imprima o vetor antes e depois da modificação.

5. Experimente passar uma referência não mutável e veja o erro gerado.