**EXERCÍCIOS SOBRE FUNÇÕES**

FONTE: https://www.weheartswift.com/functions/

1. **Mínimo**: Escreva uma função que recebe dois inteiros, a e b, e retorna o menor deles.

println(min(1, 2)) // 1

println(min(10, 5)) // 5

1. **Equação**: Considere uma equação do segundo grau na forma y = ax2 + bx + c. Escreva um programa que retorne as raízes da equação, ou seja, valores de x para os quais y = 0.

func raizes (a: Double, b: Double, c: Double) -> Double? {

// seu código aqui

}

1. **Números primos**: implemente uma função que recebe dois inteiros, a e b, e retorna TRUE se a é divisível por b. Implemente uma segunda função, contaDivisores, que usa a primeira para contar o número de divisores de um número recebido como parâmetro. Finalmente, escreva a função verificaPrimo para determinar se um número é primo com base no seu número de divisores.

func divide (a: Int, b: Int) -> Bool {

// seu código aqui

}

func contaDivisores(numero: Int) -> Int {

// seu código aqui

}

func verificaPrimo(numero: Int) -> Bool {

// seu código aqui

}

1. **Primeiros primos**: use a função verificaPrimo para imprimir os primeiros N números primos a partir de 2.

imprimePrimeirosPrimos(3)

// 2

// 3

// 5

imprimePrimeirosPrimos(10)

// 2

// 3

// 5

// 7

// 11

// 13

// 17

// 19

// 23

1. **Mais divisores**: Qual número inteiro, entre 1 e 10000, tem o maior número de divisores e quais são estes divisores? Escreva uma função que usa a função *contaDivisores* para chegar neste resultado. Retorne o resultado na forma de uma tupla.
2. **Soma**: escreva uma função que recebe um vetor de números inteiros e retorna a sua soma.

soma([1, 2, 3]) // 6

soma([1, 1, 1, 1, 1]) // 5

1. **Pares corretos**: escreva a função *verificaParênteses* que recebe uma string contendo sinais de abre e fecha parênteses, além de quaisquer outros caracteres. A função deve retornar TRUE se as aberturas de parênteses correspondem aos fechamentos (conforme os exemplos abaixo), ignorando os outros caracteres.

verificaParênteses("()") // true

verificaParênteses("((") // false

verificaParênteses("(())") // true

verificaParênteses("()()") // true

verificaParênteses("(()))") // false

verificaParênteses(")(") // false

1. **Filas**: uma fila é uma estrutura de dados que permite duas operações: **push**, que adiciona um valor no final da fila e **pop**, que remove um valor do início da fila. Implemente estas duas operações considerando um vetor de inteiros recebido como referência, conforme os exemplos abaixo.

// fila vazia

var queue: [Int] = []

// adicionando o primeiro elemento

push(1, &queue) // queue = [1]

// adicionando o segundo elemento

push(2, &queue) // queue = [1, 2]

// retirando o primeiro elemento

pop(&queue) // 1, queue = [2]

// adicionando o terceiro elemento

push(3, &queue) // queue = [2, 3]

// retirando o segundo elemento

pop(&queue) // 2, queue = [3]

// retirando o terceiro elemento

pop(&queue) // 3, queue = []

// retirando um elemento inexistente

pop(&queue) // retorna nil

1. **Pilha**: uma pilha é uma estrutura de dados que pode realizar as seguintes operações: **push**, que adiciona um elemento no topo da pilha, **top**, que retorna o elemento que está no topo da pilha, sem removê-lo, e **pop**, que remove e retorna o elemento do topo da pilha. Implemente as três operações conforme os exemplos abaixo.

var stack: [Int] = []

push(1, &stack) // stack = [1]

push(2, &stack) // stack = [1, 2]

pop(&stack) // 2, stack = [1]

push(3, &stack) // stack = [1, 3]

pop(&stack) // 3, stack = [1]

pop(&stack) // 1, stack = []

pop(&stack) // retorna nil

1. **Jogando dados**: Em um jogo de 7/11 (*seven-eleven*), o jogador lança um par de dados e soma seus resultados. Se a soma dos valores dos dois dados for 7 ou onze, o jogado ganha. Caso contrário, ele perde. Quantas vezes você precisa lançar os dados, em média, para ganhar no jogo? Escreva uma função que calcule este número a partir de uma simulação.
2. **Recursividade**: implemente uma função que calcule o fatorial de um número inteiro recebido como parâmetro.
3. **Algoritmo de Euclides**: implemente uma função com o algoritmo euclidiano para encontrar o maior divisor comum entre dois números. O algoritmo começa com dois números A e B e subtrai o menor do maior até que algum dos números atinja zero. Neste caso, o outro número será o máximo divisor comum. Para entender o algoritmo, considere os exemplos abaixo.

10 2

8 2

6 2

4 2

2 2

2 0 // 2 é o maior divisor comum entre 10 e 2

9 6

3 6

3 3

3 0 // 3 é o maior divisor comum entre 9 e 6

35 49

35 14

21 14

7 14

7 7

7 0 // 7 é o maior divisor comum entre 35 e 49