INSTITUTO FEDERAL
Catarinense
Campus Blumenau

# Algoritmos

Bacharelado em Ciência da Computação
Prof. Paulo César Rodacki Gomes

## Lista de exercícios — Alg-08
## Exercícios sobre Recursividade em Python

Estes exercícios devem ser entregues no Google Classroom. Para cada um dos exercícios, crie um arquivo fonte Python com o respectivo nome de acordo com a seguinte regra: `SUASINICIAIS-Alg-08-Ex-num.py`. Por exemplo, se o professor resolvesse o exercício número 3, o nome do arquivo seria `PCRG-Alg-08-Ex-03.py`.

## Introdução

Uma função recursiva é uma função que chama a si mesma (inclusive em programação). Neste capítulo, você deve usar funções recursivas para resolver uma variedade de problemas. Os programas que você escrever na resolução dos exercícios abaixo o ajudarão a aprender a:

- Identificar o(s) caso(s) base para uma função recursiva
- Identificar o(s) caso(s) recursivo(s) para uma função recursiva
- Escrever uma função recursiva não trivial
- Usar uma função recursiva que você escreveu para resolver um problema

**Questões de conjuntos:**

1. **Fatorial**. Implementa uma função recursiva para calcular o fatorial de um número inteiro positivo. O fatorial é denotado pelo símbolo de exclamação "!" e é definido da seguinte forma: 1! = 1 e n! = n x (n-1)!, para n>1.

2. **Sequencia de Fibonacci.** A série de Fibonacci é uma sequencia de $F_n$ números inteiros no qual um termo é definido pela soma dos dois termos anteriores. Os primeiros termos $F_i$ da sequencia são 0, 1, 1, 2, 3, 5, 8, 13, etc. Portanto, o enésimo termo da sequencia é definido por $F_n = F_{n-1} + F_{n-2}$, sendo $F_0 = 0$ e $F_1 = 1$. Escreva uma função Python recursiva que recebe como parâmetro um valor inteiro n, e retorna o enésimo termo da sequencia de Fibonacci.

3. **Palíndromo**. Faça uma função Python **recursiva** que recebe uma string e retorne um valor lógico indicando se ela é ou não é um palíndromo. OBS: Um palíndromo é uma palavra ou frase, que é igual quando lida da esquerda para a direita ou da direita para a esquerda (Espaços em branco e sinais de pontuação devem ser descartados). Exemplo de palíndromo: "saudavel leva duas".

4. **Fibonacci com memorização de resultado**. Escreva uma nova versão da sua função recursiva do exercício 2 (Fibonacci) utilizando a técnica de memorização de resultado para melhorar desempenho e consumo de memória.

5. **Total de valores numéricos.** Escreva um programa que leia os valores numéricos do usuário até que uma linha em branco seja inserida. Exiba a soma total de valores inseridos pelo usuário (ou 0,0 se o primeiro valor inserido for uma linha em branco). Conclua esta tarefa usando recursão. Seu programa não pode usar nenhum laço de repetição.

> **Dica:** o corpo da sua função recursiva precisará ler um valor do usuário e, em seguida, determinar se deve ou não fazer uma chamada recursiva. Sua função não precisa de nenhum parâmetro, mas precisará retornar um resultado numérico.

6. **MDC - Máximo Divisor Comum**. Euclides foi um matemático grego que viveu há aproximadamente 2.300 anos. Seu algoritmo para calcular o MDC - máximo divisor comum de dois inteiros positivos, ***a*** e ***b***, é eficiente e recursivo. Está descrito abaixo:

```
MDC(a, b)
    if b == 0 then
        return a
    else
        c ← a%b
        return MDC(b,c)
    end
```

   Escreva um programa que implemente o algoritmo recursivo de Euclides e o use para determinar o máximo divisor comum de dois inteiros inseridos pelo usuário.

7. **Conversão decimal → binário iterativa.** Escreva uma função que converte um número decimal (base 10) em binário (base 2). A função deve receber como parâmetro o número inteiro decimal (*q)* e, em seguida, deve realizar a conversão usando o algoritmo de divisão mostrado abaixo. Quando o algoritmo for concluído, o resultado contém a representação binária do número, que deve ser retornada pela função como uma string.

```
DecBinIterativo(q)
    result ← ""
    repeat
        r ← q%2
        result ← str(r) + result
        q ← q//2
    until q == 0;
    return result
```

8. **Conversão decimal → binário recursiva.** No exercício anterior, você escreveu um programa que usava um laço de repetição para converter um número decimal para sua representação binária. Neste exercício, você executará a mesma tarefa usando recursividade.

   Escreva uma função recursiva que converta um número decimal não negativo em binário. Trate 0 e 1 como casos básicos que retornam uma string contendo o dígito apropriado. Para todos os outros inteiros positivos, n, você deve calcular o próximo dígito usando o operador de resto e, em seguida, fazer uma chamada recursiva para calcular os dígitos de n // 2. Finalmente, você deve concatenar o resultado da chamada recursiva (que será um string) e o próximo dígito (que você precisará converter em uma string) e retornar essa string como o resultado da função.

   Escreva um programa principal que use sua função recursiva para converter um inteiro não negativo inserido pelo usuário de decimal para binário. Seu programa deve exibir uma mensagem de erro apropriada se o usuário inserir um valor negativo.

9. **Raiz quadrada recursiva**. No exercício 9 da lista 4 você explorou como a iteração pode ser usada para calcular a raiz quadrada de um número. Naquele exercício, uma melhor aproximação da raiz quadrada foi gerada com cada iteração adicional de um laço de repetição. Neste exercício, você deve usar a mesma estratégia de aproximação, mas você usará recursão ao invés de iteração.

   Crie uma função de raiz quadrada com dois parâmetros. O primeiro parâmetro, ***n***, é o número para o qual a raiz quadrada está sendo calculada. O segundo parâmetro, ***estimativa***, é a estimativa atual para a raiz quadrada. O parâmetro de estimativa deve ter um valor padrão de 1,0. Não forneça um valor padrão para o primeiro parâmetro.

Sua função de raiz quadrada será recursiva. O caso básico ocorre quando o valor absoluto da diferença entre ***estimativa*²** (estimativa ao quadrado) e ***n*** é menor ou igual a $10^{-12}$. Neste caso, sua função deve retornar o valor de ***estimativa*** porque está próximo o suficiente da raiz quadrada de **n**. Caso contrário, sua função deve retornar o resultado da chamada a si própria passando **n** como primeiro parâmetro e $\frac{estimativa + \frac{n}{estimativa}}{2}$ como o segundo parâmetro.

Escreva um programa principal que demonstre sua função calculando a raiz quadrada de vários valores diferentes. Ao chamar sua função de raiz quadrada a partir do programa principal, você deve passar apenas um parâmetro para ela, de modo que o valor padrão para estimativa seja usado.

> **Dica:** pesquise sobre funções com argumentos padrão (*default arguments*) em Python.

10. **Decodificação run-lenght**. A codificação run-length é uma técnica simples de compressão de dados que pode ser eficaz quando valores repetidos ocorrem em posições adjacentes dentro de uma lista. Compressão é obtida substituindo grupos de valores repetidos por uma cópia do valor, seguido pelo número de vezes que o valor deve ser repetido. Por exemplo, a lista `["A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "B", "B", "B", "B", "A", "A", "A", "A", "A", "A", "B"]` seria comprimida como `["A", 12, "B", 4, "A", 6, "B", 1]`. A descompressão é realizada replicando cada valor da lista o número de vezes indicado.

    Escreva uma função recursiva que descompacte uma lista codificada run-lenght. Sua função recursiva deve ter uma lista compactada em run-lenght como seu único parâmetro. Ela deve retornar a lista descompactada como seu único resultado. Crie um programa principal que exibe uma lista codificada em run-lenght e o resultado da decodificação.

11. **Codificação run-lenght.** Escreva uma função recursiva que implemente a técnica de compressão run-lenght descrita no exercício anterior. Sua função deve receber uma lista ou uma string como seu único parâmetro. Ela deve retornar a lista compactada em run-lenght como seu único resultado. Inclua um programa principal que leia uma string do usuário, a compacte e exiba o resultado codificado em run-lenght.

> **Dica:** talvez você precise implementar um laço de repetição dentro de sua função recursiva.