INSTITUTO FEDERAL
Paraíba
Campus João Pessoa

Curso: Sistemas para Internet/Redes de Computadores
Disciplina: Estrutura de Dados
Período: 2º
Professor: Alex Sandro da Cunha Rêgo

Lista
8

Recursividade

python

STRINGS

1) Faça uma função recursiva chamada **recursiveLength()** que retorne a quantidade de caracteres de um string.

2) Faça uma função recursiva chamada **printstr()** que imprima na tela uma string (caractere a caractere).

3) Faça uma função recursiva chamada **invertString()** que retornte a sequência de caracteres de uma string passada como argumento na ordem inversa

4) Faça uma função recursiva chamada **printInverse()** que imprima uma string ao contrário

5) Faça uma função recursiva chamada **compareStr(char *str1, char *str2)** que retorne:
   0: str1 é igual a str2;
   1: str1 é maior que str2;
   -1: str1 é menor que str2;

6) Faça uma função recursiva chamada **ispalindrome()** que retorne verdadeiro caso uma string seja palíndrome, ou falso caso contrário. O protótipo da operação é definido por:

```
def ispalindrome(str)
```

VARIADOS

7) Escreva uma função recursiva que retorne a soma dos **n** primeiros números inteiros. Se **n = 3,** a soma seria igual a 1 + 2 + 3 = 6

8) Faça uma função recursiva chamada **menores_rec()** que receba como parâmetro um **list** de valores numéricos e um número inteiro **key.** A função deve retornar quantos elementos da lista possuem valor inferior a **key.** O protótipo da função é definido por:

```
def menores_rec( lista, key )
```

9) Faça uma função recursiva chamada **decToBin()** que receba um número inteiro na base decimal e imprima seu correspondente na base binária. O protótipo da função é definido por:

```
def decToBin( num )
```

10) Um material radioativo denominado invictus, quando em contato com o oxigênio, perde metade de sua massa a cada 50 segundos. Faça um função recursiva que receba uma quantidade de massa do invictus, em gramas, e exiba o tempo (em segundos) necessário para que sua massa se torne menor que 0,8 g. A função também deve retornar o o valor da massa final.

11) Faça uma função recursiva denominada **seqTermos1()** que calcule a soma dos **n** termos da série:

```
n =       1    2     3     4     5            n
série:    1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n
```

def seqTermos1(n) , onde **n** é o número de termos

```
def seqTermos1(n):
   if( n == 1 )
      return 1
```

12) Faça uma função recursiva denominada **seqTermos2()** que calcule a soma dos **n** termos da série:

$$\frac{2}{4}+\frac{5}{5}+\frac{10}{6}+\frac{17}{7}+\frac{26}{8}+\cdots+\frac{(n^2+1)}{(n+3)}$$

```
def seqTermos2( n);
```

13) Dado um vetor de números reais, escreva uma função recursiva para determinar a soma dos elementos do vetor.

14) Dado um vetor de números inteiro, escreva uma função recursiva para identificar o maior valor armazenado no vetor.

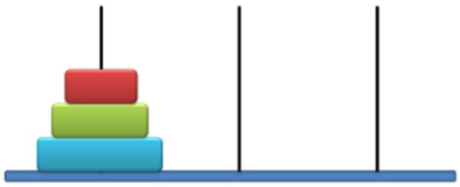
15) Dado um vetor de números inteiro, escreva uma função recursiva para verificar se um vetor está ordenado ou não.

16) O código abaixo imprime, de forma recursiva, o Triângulo de Pascal. De acordo com o programa, qual a sequência numérica a ser impressa para cada iteração do comando for inserido no programa principal?

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        line = [1]
        previous_line = pascal(n-1)
        for i in range(len(previous_line)-1):
            line.append(previous_line[i] + previous_line[i+1])
        line += [1]
    return line

# Aqui se inicia o programa principal
for i in range(1,6):
    print(pascal(i))
```

17) Torre de Hanói é um "quebra-cabeça" clássico com solução via recursividade que consiste em uma base contendo três pinos, em um dos quais são dispostos alguns discos uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo. O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma situação. O número de discos pode variar sendo que o mais simples contém apenas três.



É interessante observar que o número mínimo de "movimentos" para conseguir transferir todos os discos da primeira estaca à terceira é $\mathbf{2^n-1}$, sendo **n** o número de discos. Logo:

- Para solucionar um Hanói de 4 discos, são necessários 15 movimentos
- Para solucionar um Hanói de 7 discos, são necessários 127 movimentos
- Para solucionar um Hanói de 15 discos, são necessários 32.767 movimentos

Acompanhe a solução recursiva do problema da Torre de Hanói e exiba o que vai ser impresso na tela.

```
def moveTower(numDiscos,origem, destino, auxiliar):
    '''
     numDiscos: int - Quantidade de discos a movimentar.
     origem: identificador da torre de origem
     destino: identificador da torre destino.
     temp: identificador da torre auxiliar
    '''
    if numDiscos >= 1:
        moveTower(numDiscos-1,origem,auxiliar,destino)
        moveDisco(origem,destino)
        moveTower(numDiscos-1,auxiliar,destino,origem)

def moveDisco(origem,destino):
    print("Movendo disco da haste",origem,"para a haste",destino)

moveTower(3,"A","B","C")
```

18) Implemente uma função recursiva que converta um número da base decimal para qualquer uma das seguintes bases: binária (2), octal (8) e hexadecimal (16). A função deve obedecer ao seguinte protótipo:

```
def decToBase(num, base)
```

19) Faça uma função recursiva que retorne a soma de todos os elementos de um list de inteiros passado como argumento. O protótipo da função é definido por: