Recursividade

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 na mesma linha).

3) Faça uma função recursiva chamada **invertString()** que retorne 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)
```

DIVERSOS

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. Desenvolva 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 valor da massa final. O código com a solução utilizando a abordagem iterativa é ilustrado a seguir.

```
def invictusIterativo(massa):
    tempo = 0
    while( massa >= 0.8 ):
       tempo += 50
       massa = massa/2
    return massa,tempo
```

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

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

n = 1 2 3 4 5 ..... n

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

def **seqTermos2**(n); onde **n** é o número de termos

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.

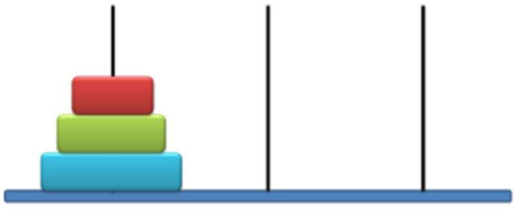
16) O código abaixo imprime, de forma recursiva, o Triângulo de Pascal. De acordo com o programa, qual a sequência numérica a ser impressa para cada iteração do comando for inserido no programa principal?

```
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)
        moveDisc(origem,destino)
        moveTower(numDiscos-1,auxiliar,destino,origem)

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

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

18) A função recursiva abaixo foi desenvolvida com o intuito de converter um número inteiro para seu correspondente em qualquer uma das seguintes bases: binária (2), octal (8) e hexadecimal (16). Neste exercício, examine o código e escreva a abstração do funcionamento desta função.

```
def decToBase(num, base):
   convertString = "0123456789ABCDEF"
   if num < base:
     return convertString[num]
   else:
      return decToBase(num//base,base) + convertString[num%base]

print(decToBase(8,8))
print(decToBase(8,2))
print(decToBase(10,16))
```

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:

```
def soma( lista )

Exemplo:
array = [ 5, 10, 20, 35, 40 ]
print( soma(array) )  # exibe 110
```

20) Escreva uma função recursiva que determina a soma de diferentes listas, conforme exemplo a seguir:

```
def somaListas( lista )

entrada = [ 1, 2, [3,4], [5,6] ]
print( somaListas(array) )  # exibe 21
```

21) Escreva uma função recursiva para calcular a soma dos inteiros positivos da série:

```
n + (n-2) + (n-4) + ... até (n – x <= 0)

somaSerie(6) = 12
somaSerie(10) = 30
```

22) O Máximo Divisor Comum (MDC) entre dois números naturais **u** e **v** é o maior número inteiro que divide ambos. Dois números naturais sempre têm divisores em comum. Para calcular o MDC, devemos fazer uma das seguintes operações: **decomposição em fatores primos** ou **decomposição sucessiva**. Neste exercício, vamos implementar o método da **decomposição sucessiva**, que é definida como segue:

$$\texttt{mdc}(u, v) \begin{cases} \texttt{mdc}(v,u) & , \text{se } u > v \\ u & , \text{se } v = 0 \\ \texttt{mdc}(v, u \bmod v) & , \text{se } v > 0 \end{cases}$$

Vamos tomar como exemplo a chamada **mdc(48,30)**

1º) dividimos o número maior pelo número menor;
48 / 30 = 1 (com resto 18)
2º) dividimos o divisor 30 (divisor da divisão anterior) por 18 (resto da divisão anterior), para obter o resto da divisão

30 / 18 = 1 (com resto 12)

3º) Segue a regra anterior sucessivamente, até que se alcance o resto 0 (zero)

18 / 12 = 1 (resto 6)
12 / 6 = 2 (resto zero - divisão exata)

4º) O divisor que corresponde à divisão exata é 6. Logo, este é o resultado do **mdc(48,30)**

Implemente a solução recursiva para o cálculo do **mdc**.