Python – Modularização

Modularização

- Até agora nosso programa é uma sequência de comandos um após o outro, podendo ter:
  - Operadores
  - Comando de seleção
  - Comandos de repetição
  - Variáveis simples e compostas
- Porém, muitas vezes, para que o programa fique mais simples e fácil de ser implementado ele pode ser dividido em módulos
  - Módulos são trechos de código com uma função bem definida

Modularização

- As vantagens de se dividir um programa em módulos são as seguintes:
  - Simplificação do código
    - Torna o programa mais legível e organizado
  - Reutilização de código
    - Ao invés de reescrever o mesmo código várias vezes, utilizamos um módulo para executar a mesma função várias vezes
  - Facilidade de manutenção
    - A localização e correção de um erro fica mais fácil

Modularização

- Já foi mostrado que o Python é composto de vários módulos distintos que podem ser importados através do comando:
  - Import NOME_MODULO
  - Ex: import math
- Agora vamos ver como criar módulos em Python
- A principal diferença é que um módulo é definido com a palavra ***def***

Modularização

- Desse modo, usamos a palavra def, seguida do nome da função e os parâmetros de entrada, caso necessário
- Vamos ver o exemplo abaixo:

```
def soma (x, y):
    s = x + y
    return s
```

- O módulo soma recebe dois números como parâmetros e retorna o valor da soma dos parâmetros

Modularização

- A seguir o código python completo:

Modularização

- A seguir o código python completo:

```
def soma(x,y):
    s = x+y
    return s

a = input("N1:")
b = input("N2:")
print "soma:", soma(a,b)
```

Modularização

- A seguir o código python completo:

```
def soma(x,y):
    s = x+y
    return s

a = input("N1:")
b = input("N2:")
print "soma:", soma(a,b)
```

Toda vez que um módulo retorna um valor chamamos de FUNÇAO

Modularização

- A seguir o código python completo:

```
def soma(x,y):
    s = x+y
    return s

a = input("N1:")
b = input("N2:")
print "soma:", soma(a,b)
```

Toda vez que um módulo retorna um valor chamamos de FUNÇAO

Senão, chamamos de PROCEDIMENTO

Programa

- Fazer um módulo para receber dois valores inteiros e retornar a soma dos multiplos de 7 MENOS a soma dos multiplos de 13 ENTRE esses valores

Programa

- Fazer um módulo para receber dois valores inteiros e retornar a soma dos multiplos de 7 MENOS a soma dos multiplos de 13 ENTRE esses valores

```
def somaMult(x,y):
    soma = 0
    for i in range(x+1, y):
        if i % 7 == 0:
            soma += i
        if i % 13 == 0:
            soma -= i
    return soma

a = input("N1:")
b = input("N2:")
print "soma:", somaMult(a,b)
```

Programa

- Fazer um módulo para receber um vetor como parâmetro e retornar um outro vetor sem repetições

Programa

- Fazer um módulo para receber um vetor como parâmetro e retornar um outro vetor sem repetições

```
def vetSemRep(vet):
    vetAux = []
    for i in range(0,len(vet)):
        achou = False
        for j in range(0,len(vetAux)):
            if vet[i] == vetAux[j]:
                achou = True
        if not achou:
            vetAux.append(vet[i])
    return vetAux
```

Programa

- Fazer um módulo para receber um vetor como parâmetro e retornar um outro vetor sem repetições

```
def vetSemRep(vet):
    vetAux = []
    for i in range(0,len(vet)):
        achou = False
        for j in range(0,len(vetAux)):
            if vet[i] == vetAux[j]:
                achou = True
        if not achou:
            vetAux.append(vet[i])
    return vetAux
```

```
vet=[]
vetNovo=[]
for i in range(0,5):
    vet.append(int(input("elem:")))

vetNovo = vetSemRep(vet)
print "vet:", vet
print "vet novo:", vetNovo
```

Passagem de parâmetros

- Nas linguagens de programação mais tradicionais temos dois tipos de passagem de parâmetros:
  - Por valor
    - Passa uma cópia da variável passada parâmetro
  - Por referência
    - Passa o próprio endereço da variável
- Porém, em Python é diferente
  - As variáveis são passadas por “valor”
  - As listas são passadas por “referência”

Passagem de parâmetros

- Veja o exemplo abaixo:

```
    def vetSemRep(vet,x,y):
6       print "vet:", vet, "x:", x ,"y:", y
7       vet[1] = 99
8       x = 9
9       y = y + 1
10      print "vet:", vet, "x:", x ,"y:", y

1   vet=[1,2,3]
2   a = 0
3   b = 0
4   vetSemRep(vet,a,b)
5   print "vet:", vet, "x:", a ,"y:", b
```

vet: [1, 2, 3] x: 0 y: 0

vet: [1, 99, 3] x: 9 y: 1

Mudou Não Mudou

vet: [1, 99, 3] x: 0 y: 0

Passagem de parâmetros

- Desse modo, temos que usar o retorno da função quando quisermos retornar uma variável modificada

- As listas, por sua vez, retornam sempre alteradas

Programa

- Fazer um programa para ler uma frase e guardar as letras em um vetor e a quantidade de vezes que aparece em um outro vetor

Programa

- Fazer um programa para ler uma frase e guardar as letras em um vetor e a quantidade de vezes que aparece em um outro vetor

```
def busca(letra,v1):
    for i in range(0,len(v1)):
        if letra == v1[i]:
            return i
    return -1

def pegaLetra(frase,v1,v2):
    for i in range(0,len(frase)):
        if frase[i] != " ": #tirando os espacos vazios
            posicao = busca(frase[i],v1)
            if  posicao != -1:
                v2[posicao] += 1
            else:
                v1.append(frase[i])
                v2.append(1)
```

Programa

- Fazer um programa para ler uma frase e guardar as letras em um vetor e a quantidade de vezes que aparece em um outro vetor

```
def busca(letra,v1):
    for i in range(0,len(v1)):
        if letra == v1[i]:
            return i
    return -1

def pegaLetra(frase,v1,v2):
    for i in range(0,len(frase)):
        if frase[i] != " ": #tirando os espacos vazios
            posicao = busca(frase[i],v1)
            if  posicao != -1:
                v2[posicao] += 1
            else:
                v1.append(frase[i])
                v2.append(1)
```

```
frase = raw_input("frase:")
vetLetra=[]
vetQtd = []
pegaLetra(frase,vetLetra,vetQtd)
print vetLetra
print vetQtd
```

Módulos externos

- Podemos criar módulos para serem importados
- Para isso, basta criar um ARQUIVO seguido do "***.py***" e dentro desse arquivo declarar módulos

moduloSoma.py

```
def soma(x,y):
    return x + y
```

testaModulo.py

```
import moduloSoma

a = input("N1: ")
b = input("N2: ")
print moduloSoma.soma(a,b)
```

Constantes

- As constantes podem ser definidas como variáveis que NUNCA MUDAM DE VALOR
  - São usadas para definir valores que não se modificam DURANTE a execução, mas podem ser modificados em uma NOVA execução

- Ex: definir uma constante QTD = 10, para guardar a quantidade de elementos
  - Repare que a qualquer nova execução podemos modificar o valor de QTD
  - Porém, durante a execução não podemos modificar QTD
  - Em PYTHON NÃO EXISTE CONSTANTES

Programa

- Fazer um módulo para receber duas matrizes 5x5 como parâmetro retornar um vetor com a soma das LNHAS da matriz 1 MENOS a soma das COLUNAS da matriz 2

```
def lerMat(m):
    for i in range(0,LIN):
        m.append(0)
        m[i] = []
        for j in range(0,COL):
            m[i].append(int(input("elem:")))
    return m
def somaLinCol(m1,m2):
    vet=[]
    for i in range(0,LIN):
        somaCol = 0
        somaLin = 0
        for j in range(0,COL):
            somaLin += m1[i][j]
            somaCol += m2[j][i]
        vet.app end(somaLin - somaCol)
    return vet
```

```
LIN = 5
COL = 5
mat1 = []
mat2 = []
mat1 = lerMat(mat1)
mat2 = lerMat(mat2)
print mat1
print mat2
print somaLinCol(mat1,mat2)
```

Programa

- Fazer um módulo para receber um vetor de números inteiros como parâmetro e retornar se ele está ordenado em ordem NÃO DECRESCENTE ou não.
- Ex: 0 1 1 2 3 8 15

```
def lerVet(v):
    for i in range(0,TAM):
        v.append(input("elem:"))
    return v


def estaOrdenado(v):
    for i in range(1,TAM):
        if v[i-1] > v[i]:
            return False
    return True
```

```
TAM = 10
vet = []
vet = lerVet(vet)
if estaOrdenado(vet):
    print "ORDENADO"
else:
    print "NÃO ORDENADO"
```

Programa

- Fazer um módulo para receber uma String de LETRAS como parâmetro e retornar essa cadeia CRIPTOGRAFADA
- A regra de criptografia é a seguinte:
  - Se o número na tabela ASCII for PAR DIMINUI 1 do valor ASCII
  - Senão, SOMA 1
- É para retornar o valor da CADEIA

```
def senha(palavra):
    aux = ''
    for i in range(0,len(palavra)):
        valorAscii = ord(palavra[i])
        if valorAscii % 2 == 0:
            valorAscii -= 1
        else:
            valorAscii += 1
        valorChar = chr(valorAscii)
        aux = aux + valorChar
    return aux
```

```
pal = raw_input("palavra:")
print "senha:", senha(pal)
```

Programa

- Faça o módulo para DESCRIPTOGRAFAR

```
def senha(palavra):
    aux = ''
    for i in range(0,len(palavra)):
        valorAscii = ord(palavra[i])
        if valorAscii % 2 == 0:
            valorAscii -= 1
        else:
            valorAscii += 1
        valorChar = chr(valorAscii)
        aux = aux + valorChar
    return aux
```

É O MESMO CÓDIGO

Programa

- Fazer um programa para ler uma frase e guardar as palavras e a quantidade de vezes que ela aparece
- Ex: o que nao e o que nao pode ser que
- Resposta:
  - o, que, nao, e, pode, ser
  - 2, 3, 2, 1, 1, 1

```
frase = raw_input("frase:")
vetPalavra=[]
vetQtd = []
pegaPalavras(frase,vetPalavra,vetQtd)
print vetPalavra
print vetQtd
```

```
def busca(palavra,v1):
    for i in range(0,len(v1)):
        if palavra == v1[i]:
            return i
    return -1

def pegaPalavras(frase,v1,v2):
    auxPal = ""
    for i in range(0,len(frase)):
        if frase[i] == " ": #FORMANDO PALAVRAS
            if auxPal != "":
                posicao = busca(auxPal,v1)
                if  posicao != -1:
                    v2[posicao] += 1
                else:
                    v1.append(auxPal)
                    v2.append(1)
                auxPal = ""
        else:
            auxPal = auxPal + frase[i]
```