Introdução ao Processamento de Dados
Turma 3 (2020.1)

Programação Modular

Gilson. A. O. P. Costa (IME/UERJ)

gilson.costa@ime.uerj.br

Programação Modular

- A **programação modular** é um conceito geral de programação.
- Envolve a **divisão de programas em módulos independentes**, cada qual contendo o código necessário para executar uma **tarefa específica**.
- A programação modular tem muitas vantagens em relação à programação monolítica.
- Os módulos podem ser considerados **caixas-pretas**, você não precisa se preocupar com sua implementação (**abstração**).
- Os módulos podem ser **reutilizados** em outros programas.
- Os módulos são mais fáceis de **testar e manter**.

Programação Modular

- Até agora nossos programas são compostos de uma **sequência de instruções**, podendo conter: atribuições; comandos de seleção; comandos de repetição; variáveis simples e compostas.
- Muitas vezes é preciso **repetir** num mesmo programa uma ou mais sequências de instruções que têm uma **função** específica.
- Por exemplo, para implementar a regra de Cramer temos que calcular quatro determinantes.

Programação Modular

- Além disso, há **trechos de código** que você poderia usar em vários programas, como: ordenação, busca, calcular se um número é primo, o determinante, etc.
- Até agora, para **reutilizar** estes trechos de código, você teria que copiar-e-colar em vários programas diferentes.
- Mas e se você encontrar um erro, ou uma maneira de tornar mais eficiente este código?

Funções

- O nível básico de modularização no Python é uma **função**.
- Uma **função** é um bloco de código organizado e reutilizável, criado para executar uma ação específica.
- Funções podem ser organizadas em **módulos** no Python: em bibliotecas de funções.

Funções

Vantagens da utilização de funções:

- Simplificação do código.
- Torna o programa mais legível e organizado.
- Permite a reutilização de código.
- Torna mais fácil a manutenção.

Funções

- A estrutura básica de uma função é:

```
def nome_funcao(argumento1, ..., argumentoN):
    instrucao1
    ...
    instrucaoN
    return saida
```

- Os **argumentos** (ou **parâmetros**) são as entradas da função, que são definidos quando a função é chamada pelo programa principal.
- Pode haver funções sem argumentos.

Funções

- A estrutura básica de uma função é:

```
def nome_funcao(argumento1, ..., argumentoN):
    instrucao1
    ...
    instrucaoN
    return saida
```

- A **saida** é o resultado da função: o valor que vai ser passado (retornado) para o programa principal pela função.
- Para se retornar mais que um valor, **saida** pode ser uma **lista**.
- Pode haver funções sem o comando *return*.

Funções

Exemplo: função que retorna a soma de dois números.

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

# programa principal
a = float(input('Entre com um número: '))
b = float(input('Entre com outro número: '))
print('A soma dos dois números é: ', soma(a, b))
```

Funções

Exemplo: função que avalia se um número é primo.

```
def primo(num):
    cont = 0
    for i in range(2, num):
        if (num%i) == 0:
            cont += 1
    if cont > 0:
        return False
    else:
        return True
# programa principal
x = int(input('Entre com um número: '))
if primo(x):
    print('O número é primo.')
else:
    print('O número não é primo.')
```

Funções

Exemplo: função que avalia se um número é primo (outra forma de fazer).

```
def primo(num):
    eh_primo = True
    for i in range(2, num):
        if (num%i) == 0:
            eh_primo = False
    return eh_primo

# programa principal
x = int(input('Entre com um número: '))
if primo(x):
    print('O número é primo.')
else:
    print('O número não é primo.')
```

Funções

Exemplo: função que ordena um vetor.

Ordenação por Seleção

```
N = int(input("Quanto alunos há na turma? "))
notas = [0.0]*N
for i in range(0,N):
    notas[i] = float(input("Entre com a nota #{}: ".format(i+1)))
print(notas)
for i in range(0,N-1):
    pos_min = i
    for j in range(i+1,N):          # procura o mínimo entre i+1 e N-1
        if notas[j]<notas[pos_min]:
            pos_min = j
    temp = notas[i]                 # troca as notas
    notas[i] = notas[pos_min]
    notas[pos_min] = temp
print(notas)
```

Ordenação por Seleção

```
N = int(input("Quanto alunos há na turma? "))
notas = [0.0]*N
for i in range(0,N):
    notas[i] = float(input("Entre com a nota #{}: ".format(i+1)))
print(notas)
for i in range(0,N-1):
    pos_min = i
    for j in range(i+1,N):          # procura o mínimo entre i+1 e N-1
        if notas[j]<notas[pos_min]:
            pos_min = j
    temp = notas[i]                 # troca as notas
    notas[i] = notas[pos_min]
    notas[pos_min] = temp
print(notas)
```

Funções

Exemplo: função que ordena um vetor.

```
def ordena(vetor):
    novo_vetor = vetor.copy()
    for i in range(0,len(novo_vetor)-1):
        pos_min = i
        for j in range(i+1, len(novo_vetor)):
            if novo_vetor[j]<novo_vetor[pos_min]:
                pos_min = j
        temp = novo_vetor[i]
        novo_vetor[i] = novo_vetor[pos_min]
        novo_vetor[pos_min] = temp
    return novo_vetor
...
```

```
...
# programa principal
n = int(input("Número de elementos: "))
vet = [0.0]*n
for i in range(0,n):
    vet[i] = float(input("Elemento no.{}: ".format(i+1)))
vet_ordenado = ordena(vet)
print("Vetor original: ", vet)
print("Vetor odenado: ", vet_ordenado)
```

Passagem de Parâmetros

- Em diversas linguagens de programação pode-se definir explicitamente se um parâmetro é passado à função como um **valor** ou como uma **referência**.
- Passagem por valor: é criada uma **cópia independente da variável/parâmetro** dentro da função.
- Passagem por referência: a função recebe apenas a **referência da variável/parâmetro** (endereço na memória).

Passagem de Parâmetros

- No Python a regra é fixa e **depende do tipo da variável** que é passada como parâmetro.
- Se a variável é de um **tipo simples**, e.g., *int*, *float*, *string*, ela é passada **por valor**.
- Se a variável é de um **tipo composto**, e.g., lista ou dicionário, ela é passada **por referência**.

Passagem de Parâmetros

Exemplo:

```
        def teste(vet, x, y):
 5          print("vet =", vet, " x =", x ," y =", y)          vet = [1, 2, 3]  x = 0  y = 0
 6          vet[2] = 4
            x = x + 1
 7          y = y + 1
            print("vet =", vet, " x =", x ," y =", y)          vet = [1, 2, 4]  x = 1  y = 1
 8
        vetor = [1, 2, 3]
 9      a = 0
        b = 0
        teste(vetor, a, b)                                        Mudou!      Não mudou!
 1      print("vetor =", vetor, " a =", a ," b =", b)          vetor = [1, 2, 4]  a = 0  b = 0
 2
 3
```

Funções

Exemplo: função que ordena um vetor (já mostrada).

```
def ordena(vetor):
    novo_vetor = vetor.copy()
    for i in range(0,len(novo_vetor)-1):
        pos_min = i
        for j in range(i+1, len(novo_vetor)):
            if novo_vetor[j]<novo_vetor[pos_min]:
                pos_min = j
        temp = novo_vetor[i]
        novo_vetor[i] = novo_vetor[pos_min]
        novo_vetor[pos_min] = temp
    return novo_vetor
...
```

```
...
# programa principal
n = int(input("Número de elementos: "))
vet = [0.0]*n
for i in range(0,n):
    vet[i] = float(input("Elemento no.{}: ".format(i+1)))
vet_ordenado = ordena(vet)
print("Vetor original: ", vet)
print("Vetor odenado: ", vet_ordenado)
```

Funções

Exemplo: função que ordena um vetor (sem criar uma cópia na função).

```
def ordena(vetor):
    for i in range(0,len(vetor)-1):
        pos_min = i
        for j in range(i+1, len(vetor)):
            if vetor[j]<vetor[pos_min]:
                pos_min = j
        temp = vetor[i]
        vetor[i] = vetor[pos_min]
        vetor[pos_min] = temp
...
```

```
...
# programa principal
n = int(input("Número de elementos: "))
vet = [0.0]*n
for i in range(0,n):
    vet[i] = float(input("Elemento no.{}: ".format(i+1)))
print("Vetor original: ", vet)
ordena(vet)
print("Vetor odenado: ", vet)
```

Funções

Exemplo: programa para ler uma frase e guardar as letras em uma lista e a quantidade de vezes que aparecem na frase em outra lista.

```
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)):
        posicao = busca(frase[i],v1)
        if posicao == -1:
            v1.append(frase[i])
            v2.append(1)
        else:
            v2[posicao] += 1
...
```

```
...
# programa principal
frase = input("Entre com a frase: ")
vetLetra = []
vetQtd = []
pegaLetra(frase, vetLetra, vetQtd)
print(vetLetra)
print(vetQtd)
```

Funções

Exemplo: programa para multiplicar uma matriz por um vetor.

Multiplicação de Matrizes

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

Multiplicação de matriz por vetor (matriz coluna):

```
# inserir o código para ler uma matriz A
# inserir o código para ler um vetor B
if len(A[0]) != len(B):
    print('Não é possível multiplicar! Dimensões não casam.')
else:
    C = [[0 for i in range(len(B[0]))] for j in range(len(A))]
    for i in range(len(A)):
        soma = 0
        for j in range(len(B)):
            soma += A[i][j]*B[j][0]
        C[i][0] = soma
# inserir o código para escrever a matriz A
# inserir o código para escrever o vetor B
# inserir o código para escrever o vetor C = B * A
```

Multiplicação de Matrizes

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

Multiplicação de matriz por vetor (matriz coluna):

```
# inserir o código para ler uma matriz A
# inserir o código para ler um vetor B
if len(A[0]) != len(B):
    print('Não é possível multiplicar! Dimensões não casam.')
else:
    C = [[0 for i in range(len(B[0]))] for j in range(len(A))]
    for i in range(len(A)):
        soma = 0
        for j in range(len(B)):
            soma += A[i][j]*B[j][0]
        C[i][0] = soma
# inserir o código para escrever a matriz A
# inserir o código para escrever o vetor B
# inserir o código para escrever o vetor C = B * A
```

Multiplicação de Matrizes

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

Multiplicação de matriz por vetor (matriz coluna):

```
def multMatVet(A, B):
    if len(A[0]) != len(B):
        print('Não é possível multiplicar! Dimensões não casam.')
    else:
        C = [[0 for i in range(len(B[0]))] for j in range(len(A))]
        for i in range(len(A)):
            soma = 0
            for j in range(len(B)):
                soma += A[i][j]*B[j][0]
            C[i][0] = soma
        return C
```

Funções

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

```
def leMat(M, lin, col):
    for i in range(0,lin):
        M.append([])
        for j in range(0,col):
            texto = '['+str(i)+']['+str(j)+']:'
            M[i].append(int(input(texto)))
```

Funções

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

```
def leMat(M, lin, col):
    for i in range(0,lin):
        M.append([])
        for j in range(0,col):
            texto = '['+str(i)+']['+str(j)+']:'
            M[i].append(int(input(texto)))
def escreveMat(M):
    for i in range(len(M)):
        for j in range(len(M[i])):
            print('{: ^3}'.format(M[i][j]), end =" ")
        print()
```

Funções

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

```
def leMat(M, lin, col):
    for i in range(0,lin):
        M.append([])
        for j in range(0,col):
            texto = '['+str(i)+']['+str(j)+']:'
            M[i].append(int(input(texto)))
def escreveMat(M):
    for i in range(len(M)):
        for j in range(len(M[i])):
            print('{: ^3}'.format(M[i][j]), end =" ")
        print()
def multMatVet(A, B):
    C = [[0 for i in range(len(B[0]))] for j in range(len(A))]
    for i in range(len(A)):
        soma = 0
        for j in range(len(B)):
            soma += A[i][j]*B[j][0]
        C[i][0] = soma
    return C
```

Funções

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

```
def leMat(M, lin, col):
    for i in range(0,lin):
        M.append([])
        for j in range(0,col):
            texto = '['+str(i)+']['+str(j)+']:'
            M[i].append(int(input(texto)))
def escreveMat(M):
    for i in range(len(M)):
        for j in range(len(M[i])):
            print('{: ^3}'.format(M[i][j]), end =" ")
        print()
def multMatVet(A, B):
    C = [[0 for i in range(len(B[0]))] for j in range(len(A))]
    for i in range(len(A)):
        soma = 0
        for j in range(len(B)):
            soma += A[i][j]*B[j][0]
        C[i][0] = soma
    return C
...
```

```
...
# programa principal
A=[]
B=[]
print('Elementos da matriz A:')
leMat(A,3,3)
print('Elementos do vetor B:')
leMat(B,3,1)
C = multMatVet(A, B)
print('Matriz A:')
escreveMat(A)
print('Vetor B:')
escreveMat(B)
print('Matriz A*B:')
escreveMat(C)
```

Escopo

- Escopo é a **região do programa** onde as **variáveis são acessíveis** (visíveis).
- O escopo de uma variável pode ser **local** ou **global**.
- O escopo local é **interno** a uma função.
- Quando uma variável é **criada dentro da função** ela tem escopo local a esta função: ela não é visível fora da função.
- As **variáveis globais** podem ser acessadas em qualquer parte do programa: inclusive dentro das funções.
- Variáveis globais são aquelas **criadas no programa principal**.

Escopo

Exemplo:

```
def soma(a, b):
    s = a + b
    return s

# programa principal
a = 10
b = 20
x = soma(a, b)
print('soma:', x)
print('soma:', s)    # Vai dar erro: variável local à função soma!
```

Escopo

Exemplo:

```
def soma():
    s = a + b          # Sem problemas: a e b são variáveis globais.
    return s

# programa principal
a = 10
b = 20
x = soma()
print('soma:', x)
print('soma:', s)    # Vai dar erro: variável local à função soma!
```

Escopo

Exemplo: definindo uma **variável global** dentro de uma função (evite!).

```
def soma():
    global s        # Não é uma boa prática de programação!
    s = a + b
    return s
# programa principal
a = 10
b = 20
x = soma()
print('soma:', x)
print('soma:', s)    # Não vai dar erro: s agora é global.
```

Módulos

- Podemos criar módulos em Python, com funções a serem usadas em outros programas.
- Tais módulos podem ser importados como qualquer outro módulo externo já instalado no seu computador.
- Um módulo é como qualquer programa em Python: arquivo com extensão “.py”.

Módulos

Exemplo:

soma.py

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

testa_soma.py

```
import soma

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

Função Principal

- Geralmente módulos contém somente funções: não têm programa principal.
- Mas você pode criar um programa que tenha um programa principal e também funcione como um módulo externo.
- Para importar este programa/módulo em outro programa, e não executar seu programa principal, pode-se usar uma **função principal** que vamos chamar de **main()**.

Função Principal

- O Pyhton contém diversas **variáveis implícitas**, que são atualizadas durante a execução de programas.
- A variável implícita **__name__** indica se o código que está sendo executado foi importado de um módulo externo, ou faz parte do programa que importou aquele módulo.
- No código do **programa que importou** o módulo, o valor da variável **__name__** é **"__main__"**.
- No código do **módulo importado**, o valor da variável **__name__** é o **nome do módulo**.

Módulos

Exemplo:

soma.py

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

print("__name__ =", __name__)
```

testa_soma.py

```
import soma

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

print("__name__ =", __name__)
```

Módulos

Exemplo: função principal.

soma.py

```
def soma(x,y):
    return x + y
def main():
    print("__name__ =", __name__)

If __name__=="__main__":
    main()
```

testa_soma.py

```
import soma

a = int(input("N1: “))
b = int(input("N2: "))
print(soma.soma(a,b))

print("__name__ =", __name__)
```

Função Principal

Uma **boa prática** de programação em Python é sempre ter uma função principal, e.g., **main()**, nos programas.

Função Principal

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

```
def leMat(M, lin, col):
    for i in range(0,lin):
        M.append([])
        for j in range(0,col):
            texto = '['+str(i)+']['+str(j)+']:'
            M[i].append(int(input(texto)))
def escreveMat(M):
    for i in range(len(M)):
        for j in range(len(M[i])):
            print('{: ^3}'.format(M[i][j]), end =" ")
        print()
def multMatVet(A, B):
    C = [[0 for i in range(len(B[0]))] for j in range(len(A))]
    for i in range(len(A)):
        soma = 0
        for j in range(len(B)):
            soma += A[i][j]*B[j][0]
        C[i][0] = soma
    return C
...
```

```
...
# sem função principal
A=[]
B=[]
print('Elementos da matriz A:')
leMat(A,3,3)
print('Elementos do vetor B:')
leMat(B,3,1)
C = multMatVet(A, B)
print('Matriz A:')
escreveMat(A)
print('Vetor B:')
escreveMat(B)
print('Matriz A*B:')
escreveMat(C)
```

Função Principal

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

```
def leMat(M, lin, col):
    for i in range(0,lin):
        M.append([])
        for j in range(0,col):
            texto = '['+str(i)+']['+str(j)+']:'
            M[i].append(int(input(texto)))
def escreveMat(M):
    for i in range(len(M)):
        for j in range(len(M[i])):
            print('{: ^3}'.format(M[i][j]), end =" ")
        print()
def multMatVet(A, B):
    C = [[0 for i in range(len(B[0]))] for j in range(len(A))]
    for i in range(len(A)):
        soma = 0
        for j in range(len(B)):
            soma += A[i][j]*B[j][0]
        C[i][0] = soma
    return C
...
```

```
...
def main():   # função principal
    A=[]
    B=[]
    print('Elementos da matriz A:')
    leMat(A,3,3)
    print('Elementos do vetor B:')
    leMat(B,3,1)
    C = multMatVet(A, B)
    print('Matriz A:')
    escreveMat(A)
    print('Vetor B:')
    escreveMat(B)
    print('Matriz A*B:')
    escreveMat(C)

If __name__=="__main__":
    main()
```

Introdução ao Processamento de Dados
Turma 3 (2020.1)

Programação Modular

Gilson. A. O. P. Costa (IME/UERJ)

gilson.costa@ime.uerj.br