



[Aula 5] Linguagem de Programação

Análise e Desenvolvimento de Sistemas

Matrizes e Subprogramação – Prof. Jean Zahn

jeanozahn@gmail.com

Matrizes

- ▶ Variável composta **multidimensional**

- ▶ É equivalente a um vetor, contudo permite a utilização de diversas dimensões acessadas via diferentes índices
- ▶ Pode ser pensada como um vetor onde cada célula é outro vetor, recursivamente
- ▶ Em diversas situações matrizes são necessárias para correlacionar informações



Exemplo motivacional

- Assumindo que **um aluno é avaliado com cinco notas**, seria necessário um vetor de cinco posições para guardar as notas de um aluno...

	0	1	2	3	4
notas	10.0	7.0	9.0	5.5	6.0



Exemplo motivacional

- Assumindo que **um aluno é avaliado com cinco notas**, seria necessário um vetor de cinco posições para guardar as notas de um aluno...

		notas				
		0	1	2	3	4
alunos	0	5.0	4.5	7.0	5.2	6.1
	1	2.1	6.5	8.0	7.0	6.7
	2	8.6	7.0	9.1	8.7	9.3

```
turma = [[5.0, 4.5, 7.0, 5.2, 6.1], [2.1, 6.5, 8.0, 7.0, 6.7],  
         [8.6, 7.0, 9.1, 8.7, 9.3]]
```

Exemplo motivacional

- Assumindo que **um aluno é avaliado com cinco notas**, seria necessário um vetor de cinco posições para guardar as notas de um aluno...

		notas				
		0	1	2	3	4
alunos	0	5.0	4.5	7.0	5.2	6.1
	1	2.1	6.5	8.0	7.0	6.7
	2	8.6	7.0	9.1	8.7	9.3


```
turma= [[5.0, 4.5, 7.0, 5.2, 6.1], [2.1, 6.5, 8.0, 7.0, 6.7],  
        [8.6, 7.0, 9.1, 8.7, 9.3]]
```

Exemplo motivacional

- ▶ Na verdade, na memória seria algo assim...

turma	0	0	5.0
		1	4.5
		2	7.0
		3	5.2
		4	6.1
		0	
...	2	0	
		1	7.0
		2	9.1
		3	8.7
		4	9.3

Acesso aos valores: [linha][coluna]

► Segunda nota do primeiro aluno

```
>>> turma[0][1]
```

```
4.5
```

► Quinta nota do terceiro aluno

```
>>> turma[2][4]
```

```
9.3
```

		notas				
		0	1	2	3	4
alunos	0	5.0	4.5	7.0	5.2	6.1
	1	2.1	6.5	8.0	7.0	6.7
	2	8.6	7.0	9.1	8.7	9.3

Calcular a média da turma

```
turma = [[5.0, 4.5, 7.0, 5.2, 6.1],  
[2.1, 6.5, 8.0, 7.0, 6.7], [8.6, 7.0,  
9.1, 8.7, 9.3]]  
#calcula a média  
media = 0  
#for para percorrer as linhas  
for i in range(3):  
    #for para percorrer as colunas  
    for j in range(5):  
        media = media + turma[i][j]  
media = media / 15  
print(media)
```



Preencher a matriz por leitura

```
turma = []  
for i in range(3):  
    # cria linha vazia  
    linha = []  
    for j in range(5):  
        #vai adicionando as notas na linha  
        linha.append(int(input('Digite a nota  
        [' + str(i) + ', ' + str(j) + ']:')))  
    #adiciona a linha na matriz turma  
    turma.append(linha)
```



Exemplo

- ▶ Programa que cria uma matriz **n x m** preenchida com zeros

```
n = int(input('Digite a dimensão n da matriz: '))
m = int(input('Digite a dimensão m da matriz: '))
matriz = []
for i in range(n):
    linha = []
    for j in range(m):
        linha.append(0)
    matriz.append(linha)
print(matriz)
```



Simplificando o exemplo

- ▶ Programa que cria uma matriz $n \times m$ preenchida com zeros

```
n = int(input('Digite a dimensão n da matriz: '))
m = int(input('Digite a dimensão m da matriz: '))
matriz = []
for i in range(n):
    matriz.append([0]*m)
print(matriz)
```



Imprimir em forma de matriz

- ▶ Programa que cria uma matriz **n x m** preenchida com zeros e a **imprime no formato de matriz**

```
n = int(input('Digite a dimensão n da matriz: '))
m = int(input('Digite a dimensão m da matriz: '))
matriz = []
for i in range(n):
    matriz.append([0]*m)
#imprimir em formato de matriz
for i in range(n):
    print(matriz[i])
```



Imprimir em forma de matriz

- ▶ Programa que cria uma matriz **n x m** preenchida com zeros e a **imprime no formato de matriz**

```
n = int(input('Digite a dimensão n da matriz: '))
m = int(input('Digite a dimensão m da matriz: '))
matriz = []
for i in range(n):
    matriz.append([0]*m)
#imprimir em formato de matriz
for i in range(n):
    print(matriz[i])
```

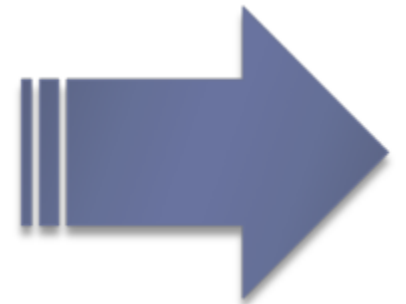
Resultado para matriz 2x3:

```
[0, 0, 0]
[0, 0, 0]
```



Exemplo Contar Pares

- ▶ Programa que lê uma matriz 3x3 digitada pelo usuário e conta quantos números pares existem na matriz, imprimindo na tela o resultado e a matriz.



Exemplo Contar Pares

```
matriz = []
for i in range(3):
    linha = []
    for j in range(3):
        linha.append(int(input('Digite o valor de [' + str(i) + ', ' + str(j) + ']:')))
    matriz.append(linha)

#contar pares
pares = 0
for i in range(3):
    for j in range(3):
        if matriz[i][j] % 2 == 0:
            pares = pares + 1

#imprimir em formato de matriz
for i in range(3):
    print(matriz[i])

#imprimir qtde de números pares
print('A matriz contém', pares, 'números pares')
```



Relembrando: for iterando sobre valores

- ▶ Um comando for também pode iterar sobre valores de uma lista

```
lista = [1, 2, 4, 5, 7, 8, 9]  
for i in lista:  
    print(i)
```



Variação do Exemplo Contar Pares

```
matriz = []
for i in range(3):
    linha = []
    for j in range(3):
        linha.append(int(input('Digite o valor de [' + str(i) + ', ' + str(j) + ']:')))
    matriz.append(linha)

#contar pares
pares = 0
for linha in matriz:
    for valor in linha:
        if valor % 2 == 0:
            pares = pares + 1

#imprimir em formato de matriz
for i in range(3):
    print(matriz[i])

#imprimir qtde de números pares
print('A matriz contém', pares, 'números pares')
```



Dimensões da matriz

- ▶ É possível usar `len()` para saber a dimensão de uma matriz
- ▶ Assumindo que todas as linhas possuem o mesmo número de elementos, pode-se fazer:
 - ▶ Número de linhas: `len(matriz)`
 - ▶ Número de colunas: `len(matriz[0])`

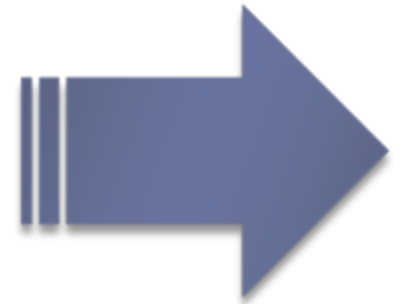
```
matriz = []
parar = False
while not (parar):
    linha = [0] * 10
    matriz.append(linha)
    x = input("Deseja parar? (S/N) ")
    if x == "S":
        parar = True
print("A matriz possui %d linhas" % len(matriz))
print("A matriz possui %d colunas" % len(matriz[0]))
```



Python permite misturar tipos em uma matriz

- ▶ Exemplo: programa que armazena os nomes e idades de 10 pessoas em uma matriz, e imprime o nome da pessoa mais nova

Ana	10
Lucas	15
Bia	13
Larissa	24
Leo	21
Bruno	32
Cássio	4
Jonas	8
Lauro	23
Mateus	18



Encontra a pessoa mais nova

```
m = []  
  
#preenche a matriz  
for i in range(10):  
    linha = []  
    linha.append(input('Digite o nome da pessoa ' + str(i) + ':'))  
    linha.append(int(input('Digite a idade de ' + linha[0] + ':')))  
    m.append(linha)
```

#procura a pessoa mais nova

```
menor = m[0][1]  
pos = 0  
for i in range(10):  
    if m[i][1] < menor:  
        menor = m[i][1]  
        pos = i
```

#imprime a matriz

```
for i in range(10):  
    print(m[i])  
print('A pessoa mais nova é', m[pos][0])
```

Ana	10
Lucas	15
Bia	13
Larissa	24
Leo	21
Bruno	32
Cássio	4
Jonas	8
Lauro	23
Mateus	18



Matrizes

- ▶ Uma matriz pode ter um número qualquer de dimensões! Basta usar um índice para cada dimensão.



Exemplo motivacional

- ▶ Ainda, assumindo que **um curso tem duas turmas**, seria necessária uma matriz tridimensional para guardar as notas de todos os alunos de todas as turmas do curso.

		notas					
		0	1	2	3	4	
alunos	0	4.2	5.1	6.0	5.4	5.1	turmas 1
	1	5.0	4.5	7.0	5.2	6.1	
	2	2.1	6.5	8.0	7.0	6.7	
	3	8.6	7.0	9.1	8.7	9.3	
		0	1	2	3	4	0

Atribuição

```
>>> m = [[5.0, 4.5, 7.0, 5.2,  
5.1], [2.1, 6.5, 8.0, 7.0, 6.7], [8.6, 7.0, 9.1, 8.7, 9.3]], [[  
4.2, 5.1, 6.0, 5.4, 5.1], [9.0, 8.0, 7.5, 8.1, 8.8], [2.3, 4.4,  
6.7, 6.6, 7.0]]]
```

Turma

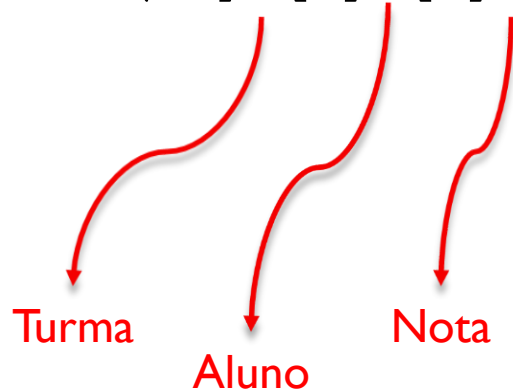
		notas						turmas
		0	1	2	3	4		
alunos	0	4.2	5.1	6.0	5.4	5.1	1	0
	1	5.0	4.5	7.0	5.2	6.1		
	2	2.1	6.5	8.0	7.0	6.7		
	3	8.6	7.0	9.1	8.7	9.3		

Atribuição

```
>>> m = [[[5.0, 4.5, 7.0, 5.2,  
5.1], [2.1, 6.5, 8.0, 7.0, 6.7], [8.6, 7.0, 9.1, 8.7, 9.3]], [[  
4.2, 5.1, 6.0, 5.4, 5.1], [9.0, 8.0, 7.5, 8.1, 8.8], [2.3, 4.4,  
6.7, 6.6, 7.0]]]
```

```
>>> print(m[0][1][0])
```

2.1



	notas						
	0	1	2	3	4		
alunos	0	4.2	5.1	6.0	5.4	5.1	
	1	5.0	4.5	7.0	5.2	6.1	
	2	2.1	6.5	8.0	7.0	6.7	
		8.6	7.0	9.1	8.7	9.3	
							turmas
							0

Exercícios

1. Faça um programa que leia uma matriz 3×3 de inteiros e multiplique os elementos da diagonal principal da matriz por um número k . Imprima a matriz na tela antes e depois da multiplicação.
2. Faça um programa que leia duas matrizes A e B 2×2 de inteiros e imprima a matriz C que é a soma das matrizes A e B .
3. Faça um programa que leia as dimensões de duas matrizes A e B , e depois leia as duas matrizes (os elementos devem ser inteiros). Se as matrizes forem de tamanhos compatíveis para multiplicação, multiplique as matrizes. Imprima as matrizes A , B e a matriz resultante da multiplicação.



Exercícios

4. Faça um programa que leia uma matriz 3x3 de inteiros e retorne a linha de maior soma. Imprima na tela a matriz, a linha de maior soma e a soma.
5. Faça um programa que leia a ordem de uma matriz quadrada A (até 100), posteriormente leia seus valores e escreva sua transposta AT , onde

$$AT[i][j] = A[j][i]$$

4. Uma pista de Kart permite 10 voltas para cada um de 6 corredores. Faça um programa que leia os nomes e os tempos (em segundos) de cada volta de cada corredor e guarde as informações em uma matriz. Ao final, o programa deve informar:
 - a) De quem foi a melhor volta da prova, e em que volta
 - b) Classificação final em ordem (1º. o campeão)
 - c) Qual foi a volta com a média mais rápida



Exercícios

7. Faça um programa que leia uma matriz 6×3 com números reais, calcule e mostre: (a) o maior elemento da matriz e sua respectiva posição (linha e coluna); (b) o menor elemento da matriz e sua respectiva posição.
8. Faça um programa que leia duas matrizes A e B de números inteiros e verifica se ambas são inversas (ou seja, se a multiplicação de A por B é a matriz identidade).
9. Faça um programa que leia uma matriz 3×3 que representa um tabuleiro de jogo da velha e indique qual posição deveria ser jogada para ganhar o jogo (se possível) ou ao menos evitar uma derrota.



Exercícios

10. Faça um programa que lê duas notas para cada aluno de duas turmas. Cada turma tem 3 alunos. Armazene os dados em uma matriz M. Cada aluno deve ter três notas (as duas digitadas e a média dessas duas). Calcule a média de cada turma e armazene em um vetor TURMA. Informe qual turma tem maior média, e quais alunos tiveram média maior que a média de sua turma.





Subprogramação



O que vimos até agora

- ▶ Programas usam apenas sequência, repetição e decisão
- ▶ Capacidade de resolver diversos problemas, mas difícil de resolver problemas grandes
 - ▶ Em diversas situações, é necessário repetir o mesmo trecho de código em diversos pontos do programa



Exemplo 1

```
max = 4
```

```
soma = 0
```

```
for i in range(max) :
```

```
    soma = soma + i
```

```
print(soma)
```

```
soma = 0
```

```
for x in range (10, 50, 10) :
```

```
    soma = soma + x
```

```
print(soma)
```



Exemplo 1

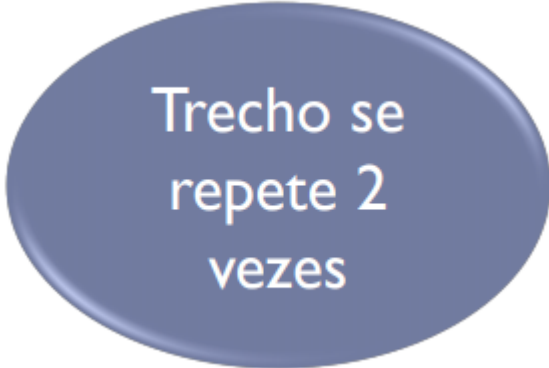
```
max = 4
```

```
soma = 0
```

```
for i in range(max):  
    soma = soma + i  
print(soma)
```

```
soma = 0
```

```
for x in range (10, 50, 10):  
    soma = soma + x  
print(soma)
```




Trecho se
repete 2
vezes



Exemplo 2

1. Ler dois valores X e Y
2. **Calcular a média de X e Y**
3. Ler dois valores A e B
4. **Calcular a média de A e B**
5. Multiplicar A por X e guardar o resultado em A
6. Multiplicar B por Y e guardar o resultado em B
7. **Calcular a média de A e B**



Operação de
cálculo de média
é repetida 3
vezes



Problemas desta “repetição”

- ▶ Programa muito grande, porque tem várias “partes repetidas”
- ▶ Erros ficam difíceis de corrigir (e se eu esquecer de corrigir o erro em uma das N repetições daquele trecho de código?)



Solução: subprogramação

- ▶ Definir o trecho de código que se repete como uma “função” que é chamada no programa
- ▶ A função é definida uma única vez, e chamada várias vezes dentro do programa



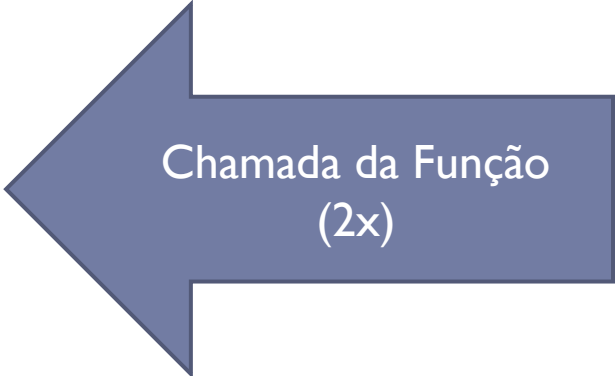
Voltando ao Exemplo 1

```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
  
    return soma
```

```
s = calcula_soma(0, 4, 1)  
print(s)  
print(calcula_soma(10, 50, 10))
```



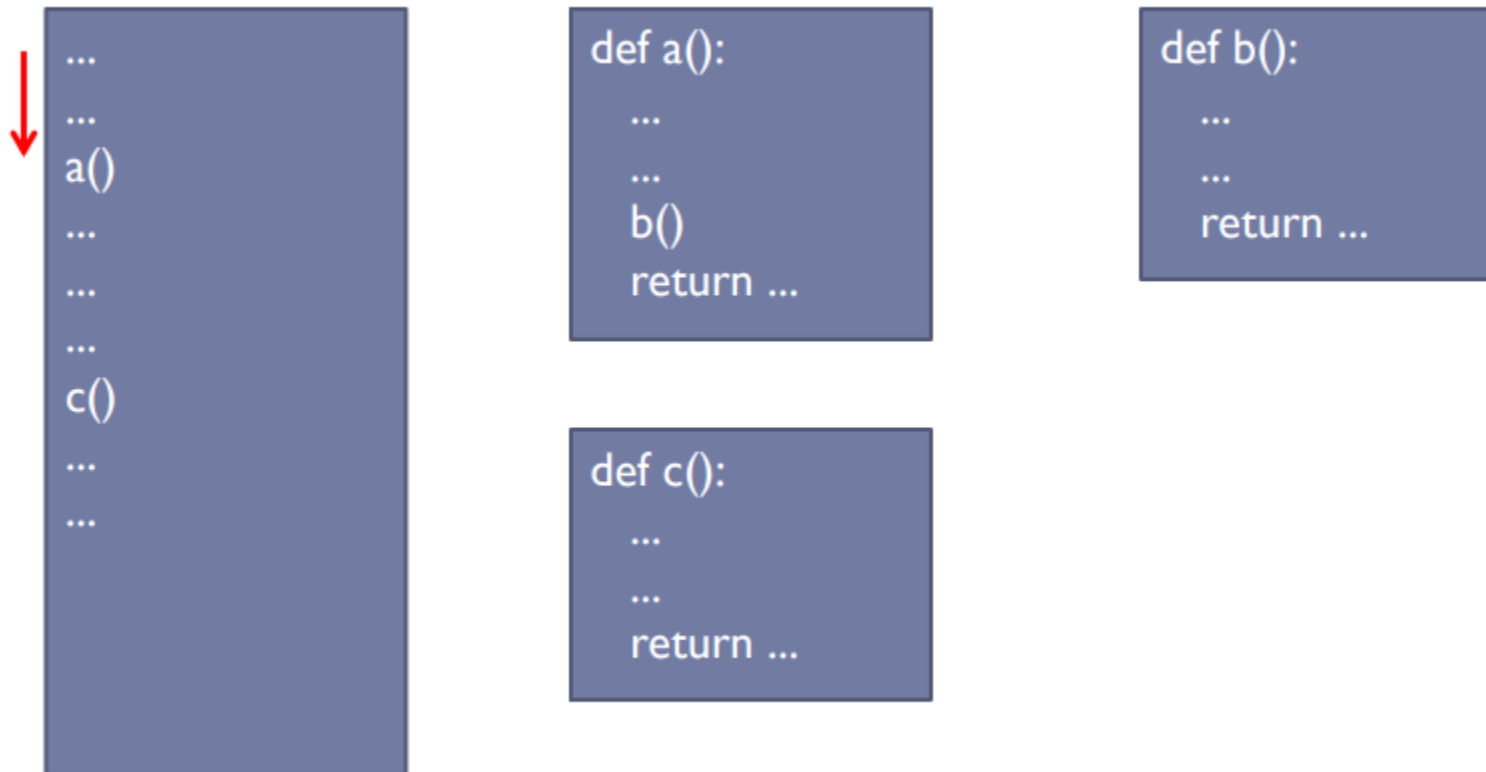
Definição da função



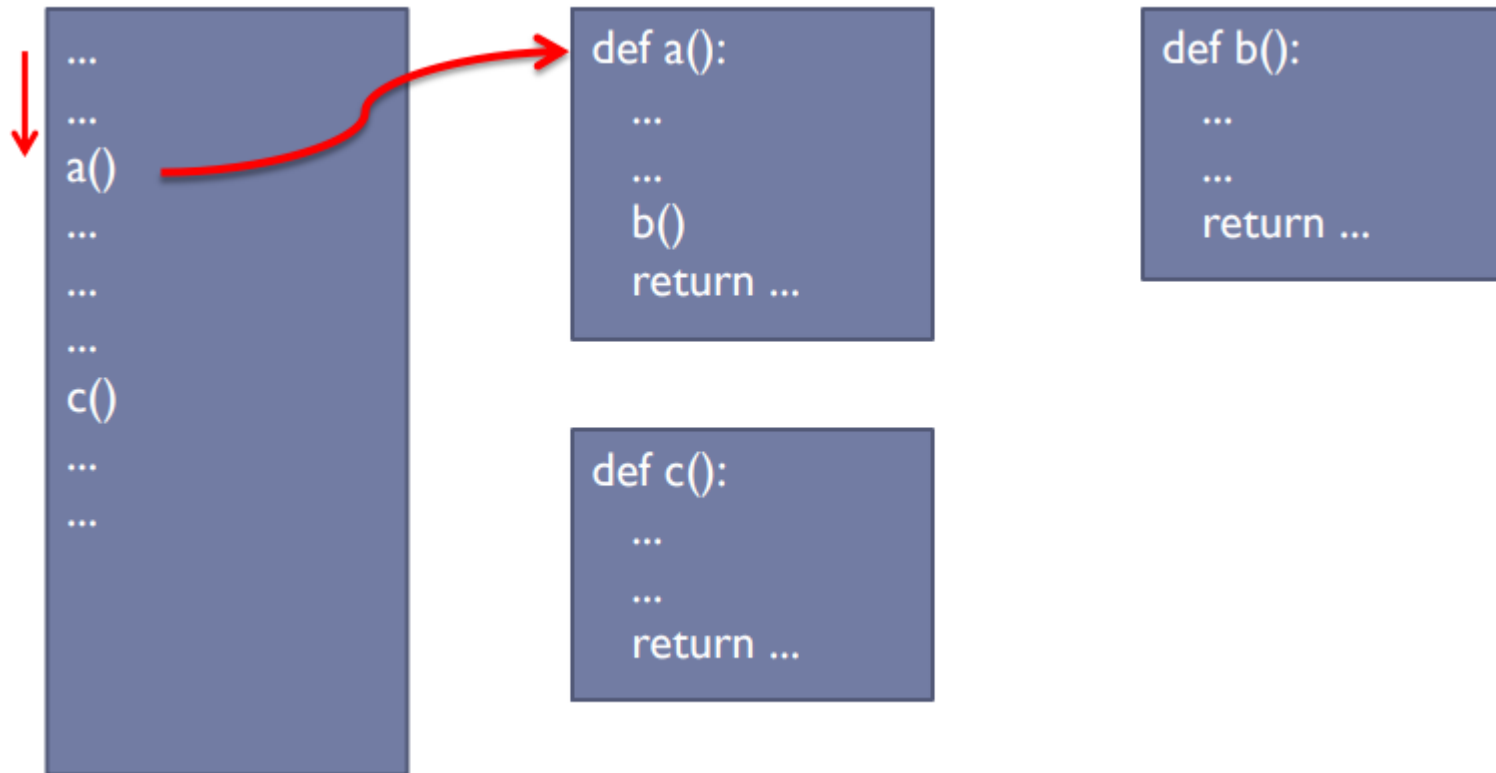
Chamada da Função
(2x)



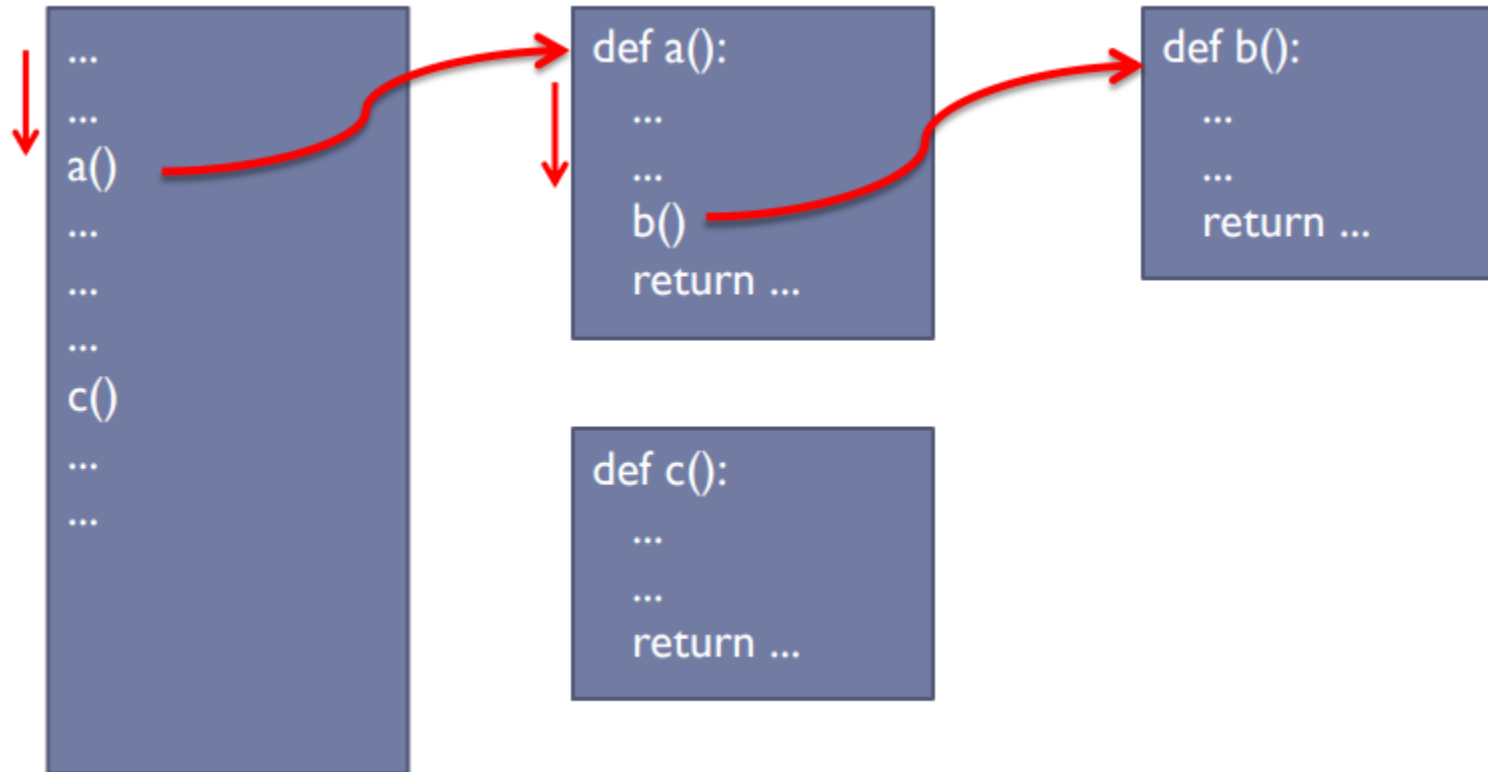
Fluxo de Execução



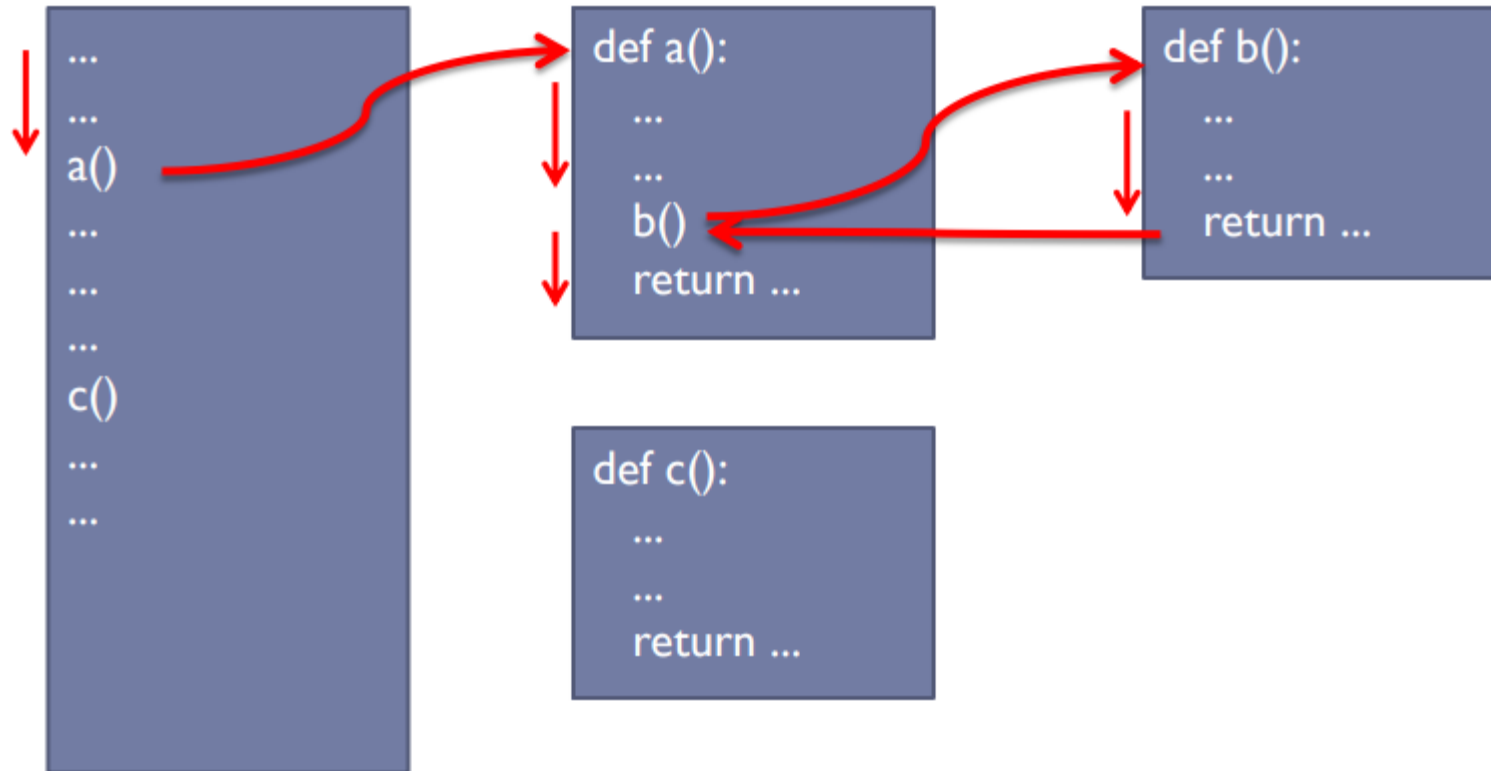
Fluxo de Execução



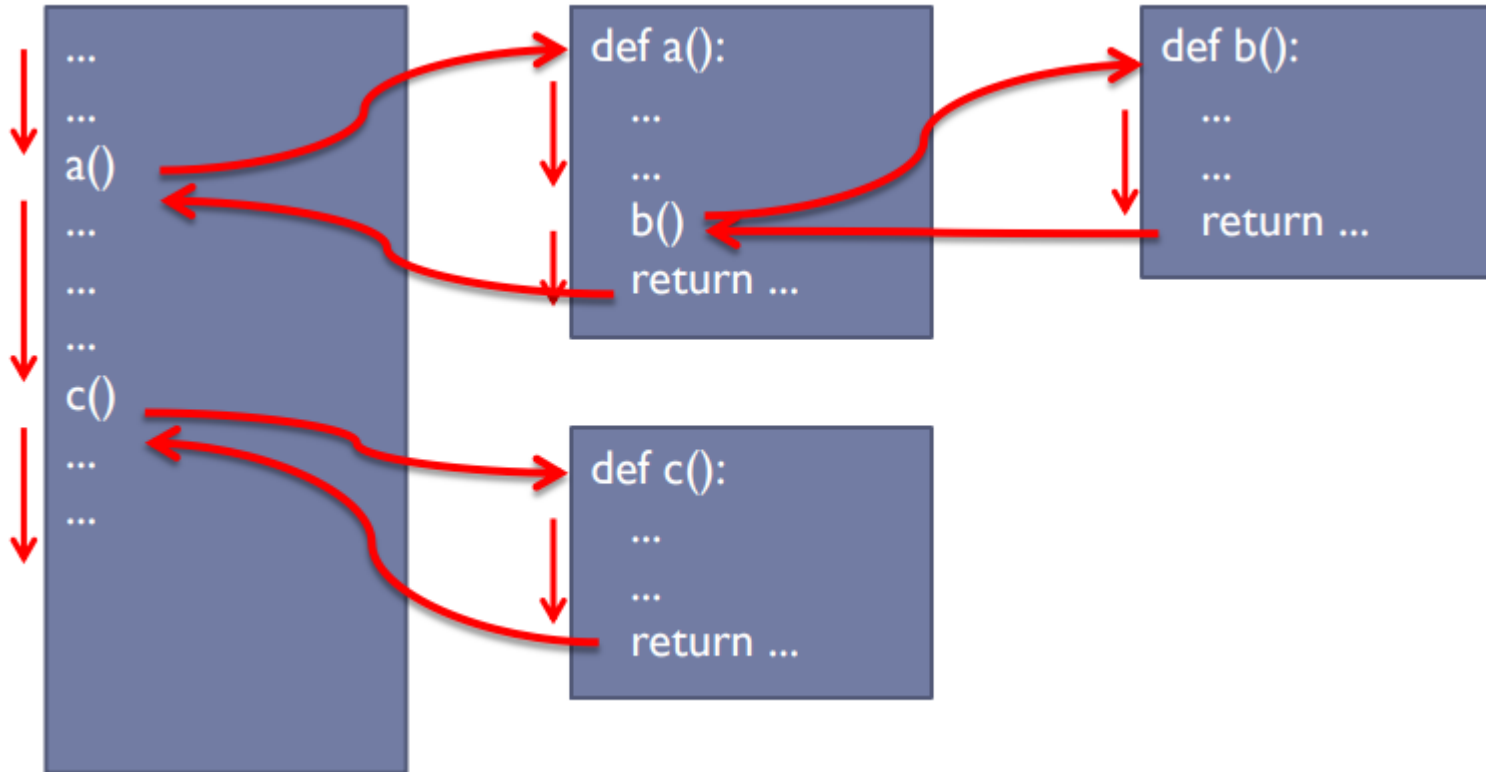
Fluxo de Execução



Fluxo de Execução



Fluxo de Execução



Fluxo de Execução

```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```

```
s = calcula_soma(0, 4, 1)  
print(s)  
print(calcula_soma(10, 50, 10))
```

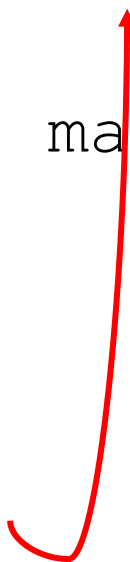


Execução
começa no
primeiro
comando que
está **fora de**
uma função




Fluxo de Execução

```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma  
  
s = calcula_soma(0, 4, 1)  
print(s)  
print(calcula_soma(10, 50, 10))
```



Fluxo de Execução



```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```

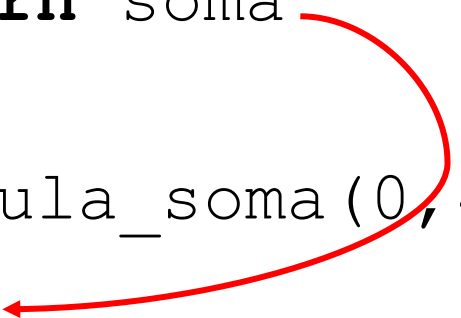
```
s = calcula_soma(0, 4, 1)  
print(s)  
print(calcula_soma(10, 50, 10))
```



Fluxo de Execução

```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```

```
s = calcula_soma(0, 4, 1)  
print(s)  
print(calcula_soma(10, 50, 10))
```



Fluxo de Execução

```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```

```
s = calcula_soma(0, 4, 1)
```


```
print(s)
```

```
↓ print(calcula_soma(10, 50, 10))
```




Fluxo de Execução

```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma  
  
s = calcula_soma(0, 4, 1)  
print(s)  
print(calcula_soma(10, 50, 10))
```



Fluxo de Execução



```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```

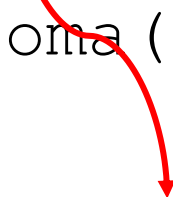
```
s = calcula_soma(0, 4, 1)  
print(s)  
print(calcula_soma(10, 50, 10))
```



Fluxo de Execução

```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```

```
s = calcula_soma(0, 4, 1)  
print(s)  
print(calcula_soma(10, 50, 10))
```



Declaração de Função

def nome_funcao (parametro, parametro, ..., parametro):
 <comandos>
 [return <variável ou valor>]

Exemplo:

```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```



Exemplo

```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    return tempo
```

```
def calcula_distancia(velocidade, tempo):  
    distancia = velocidade * tempo  
    return distancia
```

```
t = calcula_tempo(10, 5)  
print(t)  
d = calcula_distancia(5, 4)  
print(d)
```



Importante lembrar

- ▶ Um programa Python pode ter **0 ou mais** definições de função
- ▶ Uma função pode ser chamada **0 ou mais** vezes
- ▶ Uma função só é **executada** quando é **chamada**
- ▶ Duas chamadas de uma mesma função usando valores diferentes para os **parâmetros** da função podem produzir **resultados diferentes**



Escopo de Variáveis

- ▶ Variáveis podem ser locais ou globais
- ▶ Variáveis locais
 - ▶ Declaradas dentro de uma função
 - ▶ São visíveis somente dentro da função onde foram declaradas
 - ▶ São destruídas ao término da execução da função
- ▶ Variáveis globais
 - ▶ Declaradas fora de todas as funções
 - ▶ São visíveis por TODAS as funções do programa



Exemplo: variáveis locais

```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    return tempo
```

```
def calcula_distancia(velocidade, tempo):  
    distancia = velocidade * tempo  
    return distancia
```

```
t = calcula_tempo(10, 5)  
print(t)  
d = calcula_distancia(5, 4)  
print(d)
```



Exemplo: parâmetros também se comportam como variáveis locais

```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    return tempo
```

```
def calcula_distancia(velocidade, tempo):  
    distancia = velocidade * tempo  
    return distancia
```

```
t = calcula_tempo(10, 5)  
print(t)  
d = calcula_distancia(5, 4)  
print(d)
```



Exemplo: variáveis globais

```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    return tempo
```

```
def calcula_distancia(velocidade, tempo):  
    distancia = velocidade * tempo  
    return distancia
```

```
t = calcula_tempo(10, 5)  
print(t)  
d = calcula_distancia(5, 4)  
print(d)
```



Uso de Variáveis Globais x Variáveis Locais

- ▶ Cuidado com o uso de variáveis globais dentro de funções
 - ▶ Dificultam o entendimento do programa
 - ▶ Dificultam a correção de erros no programa
 - ▶ Se a variável pode ser usada por qualquer função do programa, encontrar um erro envolvendo o valor desta variável pode ser muito complexo
- ▶ Recomendação
 - ▶ Sempre que possível, usar variáveis **LOCAIS** nas funções e passar os valores necessários para a função como parâmetro



Escopo de Variáveis

```
def calcula_tempo(velocidade, distancia):
```

```
    tempo = distancia/velocidade
```

```
    return tempo
```



velocidade distancia tempo

```
def calcula_distancia(velocidade, tempo):
```

```
    distancia = velocidade * tempo
```

```
    return distancia
```



velocidade tempo distancia

```
v = 10
```

```
t = calcula_tempo(v, 5)
```

```
print(t)
```

```
d = calcula_distancia(v, t)
```

```
print(d)
```



v

t

d

Parâmetros

- ▶ Quando uma função é chamada, é necessário fornecer um valor para cada um de seus parâmetros
- ▶ Isso por ser feito informando o valor diretamente
 - ▶ `t = calcula_tempo(1, 2)`
- ▶ ou; Usando o valor de uma variável
 - ▶ `t = calcula_tempo(v, d)`



Passagem de Parâmetro

```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    return tempo
```

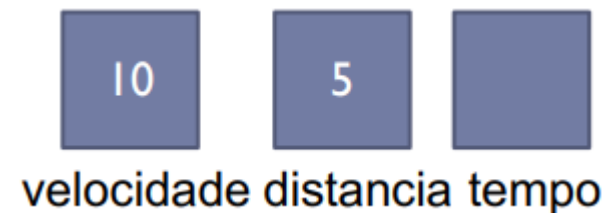
```
def calcula_distancia(velocidade, tempo):  
    distancia = velocidade * tempo  
    return distancia
```

```
v = 10  
t = calcula_tempo(v, 5)  
print(t)  
d = calcula_distancia(v, t)  
print(d)
```



Passagem de Parâmetro

```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    return tempo
```



```
def calcula_distancia(velocidade, tempo):  
    distancia = velocidade * tempo  
    return distancia
```

```
v = 10  
t = calcula_tempo(v, 5)  
print(t)  
d = calcula_distancia(v, t)  
print(d)
```



Passagem de Parâmetro

```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    return tempo
```

10

5

0.5

velocidade distancia tempo

```
def calcula_distancia(velocidade, tempo):  
    distancia = velocidade * tempo  
    return distancia
```

v = 10

t = calcula_tempo(v, 5)

print(t)

d = calcula_distancia(v, t)

print(d)

10

v

t

d

Passagem de Parâmetro por Valor

- ▶ Python usa passagem de parâmetro por valor
 - ▶ Faz cópia do valor da variável original para o parâmetro da função
 - ▶ Variável original fica preservada das alterações feitas dentro da função
- ▶ Existem exceções que veremos mais tarde




Exemplo

```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    velocidade = 0  
    return tempo
```

```
def calcula_distancia(velocidade, tempo):  
    distancia = velocidade * tempo  
    return distancia
```

```
v = 10  
t = calcula_tempo(v, 5)  
print(v)  
print(t)  
d = calcula_distancia(v, t)  
print(d)
```



O valor impresso
por **print(v)**
será **10** ou **0**?



Retorno das funções

- ▶ Função que retorna um valor deve usar **return**
 - ▶ Assim que o comando **return** é executado, a função termina
- ▶ Uma função pode não retornar nenhum valor
 - ▶ Nesse caso, basta **não usar o comando return**
 - ▶ Assim a função termina quando sua última linha de código for executada



Exemplo de função sem retorno

```
def imprime_asterisco(qtd):  
    for i in range(qtd):  
        print('*****')
```

```
imprime_asterisco(2)  
print('PROGRAMAR EH LEGAL')  
imprime_asterisco(2)
```



Chamada de função

- ▶ Se a função retorna um valor, pode-se atribuir seu resultado a uma variável

```
m = maior(v)
```

- ▶ Se a função não retorna um valor (não tem **return**), não se deve atribuir seu resultado a uma variável (se for feito, variável ficará com valor **None**)

```
imprime_asterisco(3)
```



Função sem parâmetro

- ▶ Nem toda função precisa ter parâmetro
- ▶ Nesse caso, ao definir a função, deve-se abrir e fechar parênteses, sem informar nenhum parâmetro
- ▶ O mesmo deve acontecer na chamada da função



Exemplo

```
def menu() :  
    print('*****')  
    print('1 - Somar')  
    print('2 - Subtrair')  
    print('3 - Multiplicar')  
    print('4 - Dividir')  
    print('*****')
```

menu()

```
opcao = eval(input('Digite a opção desejada: '))  
#tratar opção do usuário  
...
```



Parâmetros default

- ▶ Em alguns casos, pode-se definir um valor *default* para um parâmetro. Caso ele não seja passado na chamada, o valor *default* será assumido.
- ▶ Exemplo: uma função para calcular a gorjeta de uma conta tem como parâmetros o valor da conta e o percentual da gorjeta. No entanto, na grande maioria dos restaurantes, a gorjeta é 10%. Podemos então colocar 10% como valor *default* para o parâmetro `percentual_gorjeta`



Exemplo da gorjeta

```
def calcular_gorjeta(valor, percentual=10):  
    return valor * percentual/100
```

```
gorjeta = calcular_gorjeta(400)
```

```
print('O valor da gorjeta de 10% de uma conta de R$  
400 eh', gorjeta)
```

```
gorjeta = calcular_gorjeta(400, 5)
```

```
print('O valor da gorjeta de 5% de uma conta de R$ 400  
eh', gorjeta)
```

Quando a gorjeta não é informada na chamada da função, o valor do parâmetro gorjeta fica sendo 10



Uso de Variáveis Globais


- ▶ Variáveis globais podem ser acessadas dentro de uma função
- ▶ Se for necessário alterá-las, é necessário declarar essa intenção escrevendo, no início da função, o comando **global <nome da variável>**



Exemplo: variáveis globais **acessadas** na função

```
def maior():  
    if a > b:  
        return a  
    else:  
        return b
```

```
a = 1  
b = 2  
m = maior()  
print(m)
```



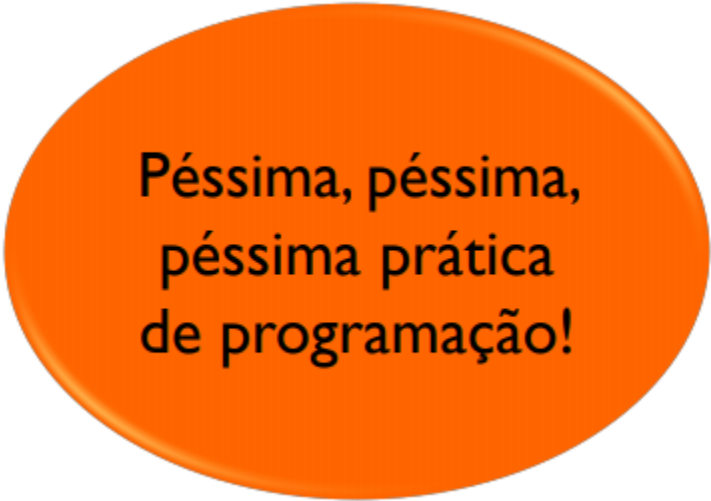
Péssima prática
de programação!



Exemplo: variável global **modificada** na função

```
def maior():  
    global m  
    if a > b:  
        m = a  
    else:  
        m = b
```

```
m = 0  
a = 1  
b = 2  
maior()  
print(m)
```

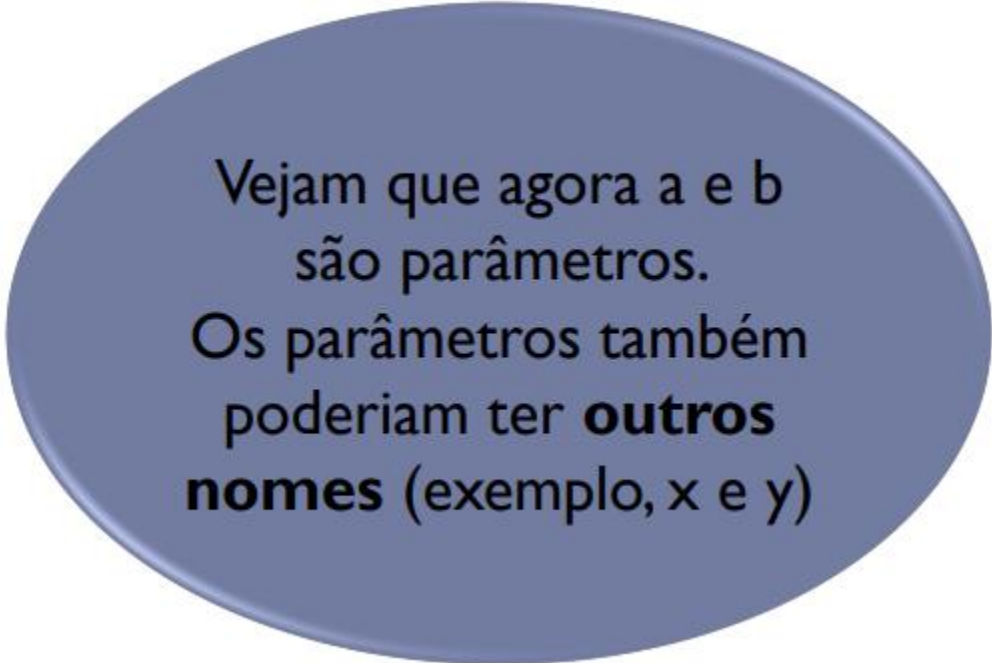


Péssima, péssima,
péssima prática
de programação!



Sem uso de variáveis globais: muito mais elegante!

```
def maior(a, b):  
    if a > b:  
        return a  
    else:  
        return b  
  
a = 1  
b = 2  
m = maior(a, b)  
print(m)
```



Vejam que agora a e b
são parâmetros.
Os parâmetros também
poderiam ter **outros**
nomes (exemplo, x e y)



Colocar funções em arquivo separado

- ▶ Em alguns casos, pode ser necessário colocar todas as funções em um arquivo separado
- ▶ Nesse caso, basta definir todas as funções num arquivo .py (por exemplo funcoes.py).
- ▶ Quando precisar usar as funções em um determinado programa, basta fazer **import <nome do arquivo que contém as funções>**
- ▶ Ao chamar a função, colocar o nome do arquivo na frente



Exemplo

Arquivo utilidades.py

```
def soma(a, b):  
    soma = a + b  
    return soma  
  
def media(a, b):  
    return (a + b) / 2
```

Arquivo teste.py

```
import utilidades  
  
x = 2  
y = 3  
print(utilidades.soma(x, y))  
print(utilidades.media(x, y))
```



Vantagens

- ▶ **Economia de código**
 - ▶ Quanto mais repetição, mais economia
- ▶ **Facilidade na correção de defeitos**
 - ▶ Corrigir o defeito em um único local
- ▶ **Legibilidade do código**
 - ▶ Podemos dar nomes mais intuitivos a blocos de código
 - ▶ É como se criássemos nossos próprios comandos
- ▶ **Melhor tratamento de complexidade**
 - ▶ Estratégia de “dividir para conquistar” nos permite lidar melhor com a complexidade de programas grandes
 - ▶ Abordagem *top-down* ajuda a pensar!

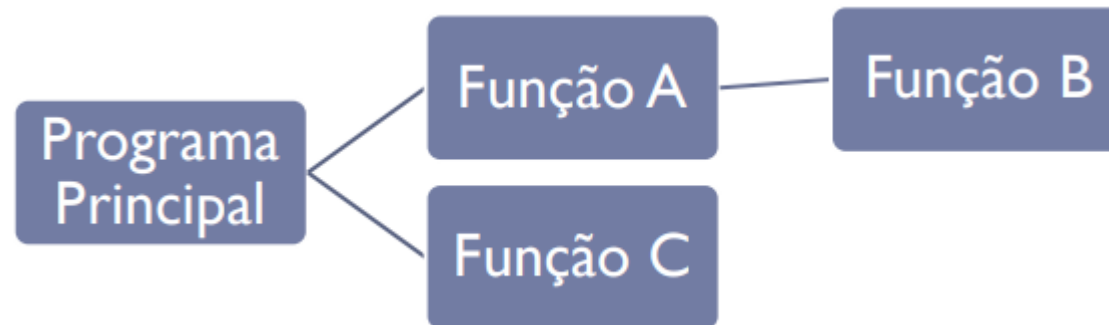


Dividir para conquistar

- ▶ Antes: um programa gigante



- ▶ Depois: vários programas menores



Exercícios (usar funções sempre que possível)

- ▶ Faça um programa que contém uma função que recebe um número inteiro que corresponde a um mês do ano e retorna o nome desse mês. Por exemplo, se o mês informado como parâmetro for 1 a função deverá retornar "janeiro", se o mês enviado como parâmetro for 2 a função deverá retornar "fevereiro" e assim por diante.



Exercícios (usar funções sempre que possível)

2. Faça uma função que informe o status do aluno a partir da sua média de acordo com a tabela a seguir:
 - ▶ Nota acima de 6 → “Aprovado”
 - ▶ Nota entre 4 e 6 → “Verificação Suplementar”
 - ▶ Nota abaixo de 4 → “Reprovado”

3. Faça uma função para verificar se um ano é bissexto ou não. Utilize a seguinte regra:
 - ▶ Um ano bissexto é divisível por 4, mas não por 100, ou então se é divisível por 400.
 - ▶ Exemplo: 1988 é bissexto pois é divisível por 4 e não é por 100; 2000 é bissexto porque é divisível por 400.

 - ▶ A função deve retornar True caso o ano seja bissexto, e False caso contrário.



Exercícios (usar funções sempre que possível)

4. Faça um programa que, dado uma figura geométrica que pode ser uma circunferência, triângulo ou retângulo, calcule a área e o perímetro da figura
 - ▶ O programa deve primeiro perguntar qual o tipo da figura:
 - ▶ (1) circunferência
 - ▶ (2) triângulo
 - ▶ (3) retângulo
 - ▶ Dependendo do tipo de figura, ler o (1) tamanho do raio da circunferência; (2) tamanho de cada um dos lados do triângulo; (3) tamanho dos dois lados retângulo



Exercícios (usar funções sempre que possível)

5. O professor deseja dividir uma turma com N alunos em dois grupos: um com M alunos e outro com $(N-M)$ alunos. Faça o programa que lê o valor de N e M e informa o número de combinações possíveis.
 - ▶ Número de combinações é igual a $N!/(M! * (N-M)!)$



Exercícios (usar funções sempre que possível)

6. Faça uma calculadora que forneça as seguintes opções para o usuário, usando funções sempre que necessário. Cada opção deve usar como operando um número lido do teclado e o valor atual da memória. Por exemplo, se o estado atual da memória é 5, e o usuário escolhe somar, ele deve informar um novo número (por exemplo, 3). Após a conclusão da soma, o novo estado da memória passa a ser 8.

Estado da memória: 0

Opções:

- (1) Somar
- (2) Subtrair
- (3) Multiplicar
- (4) Dividir
- (5) Limpar memória
- (6) Sair do programa

Qual opção você deseja?



Referências

- ▶ Alguns slides de Leonardo Murta e Aline Paes – Instituto de Computação – Universidade Federal Fluminense.





[Aula 5] Linguagem de Programação

Análise e Desenvolvimento de Sistemas

Matrizes e Subprogramação – Prof. Jean Zahn

jeanozahn@gmail.com