

SUBPROGRAMAÇÃO

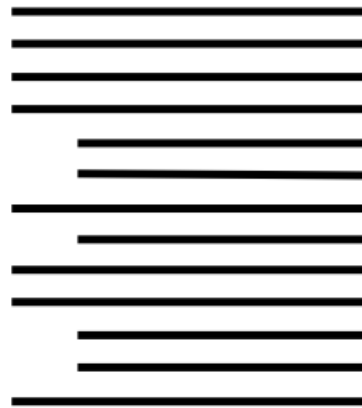
PROGRAMAÇÃO APLICADA A MATEMÁTICA

Meirylene Avelino

meirylenerea@id.uff.br

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)
```

Trecho se
repete 2
vezes

```
soma = 0
```


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

```
    soma = soma + x
```

```
print(soma)
```

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
```

Definição da
função

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

Chamada da
função (2x)

FLUXO DE EXECUÇÃO



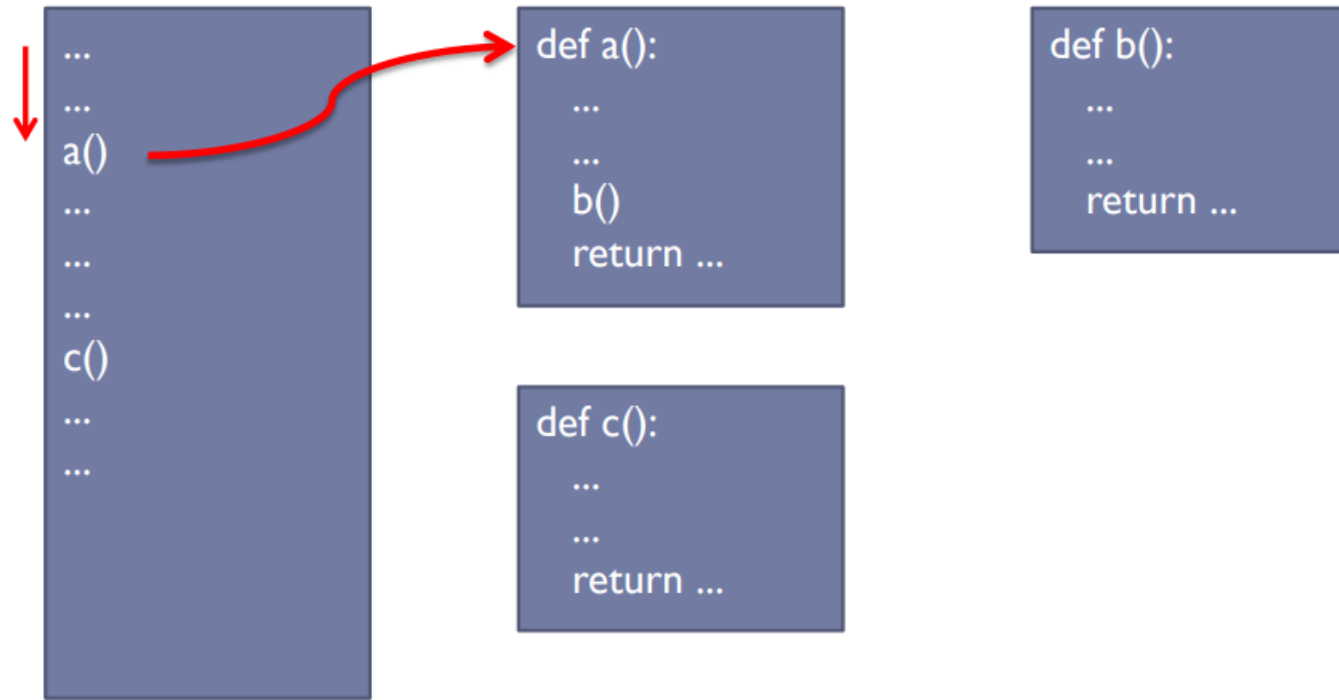
```
...  
...  
a()  
...  
...  
...  
c()  
...  
...
```

```
def a():  
    ...  
    ...  
    b()  
    return ...
```

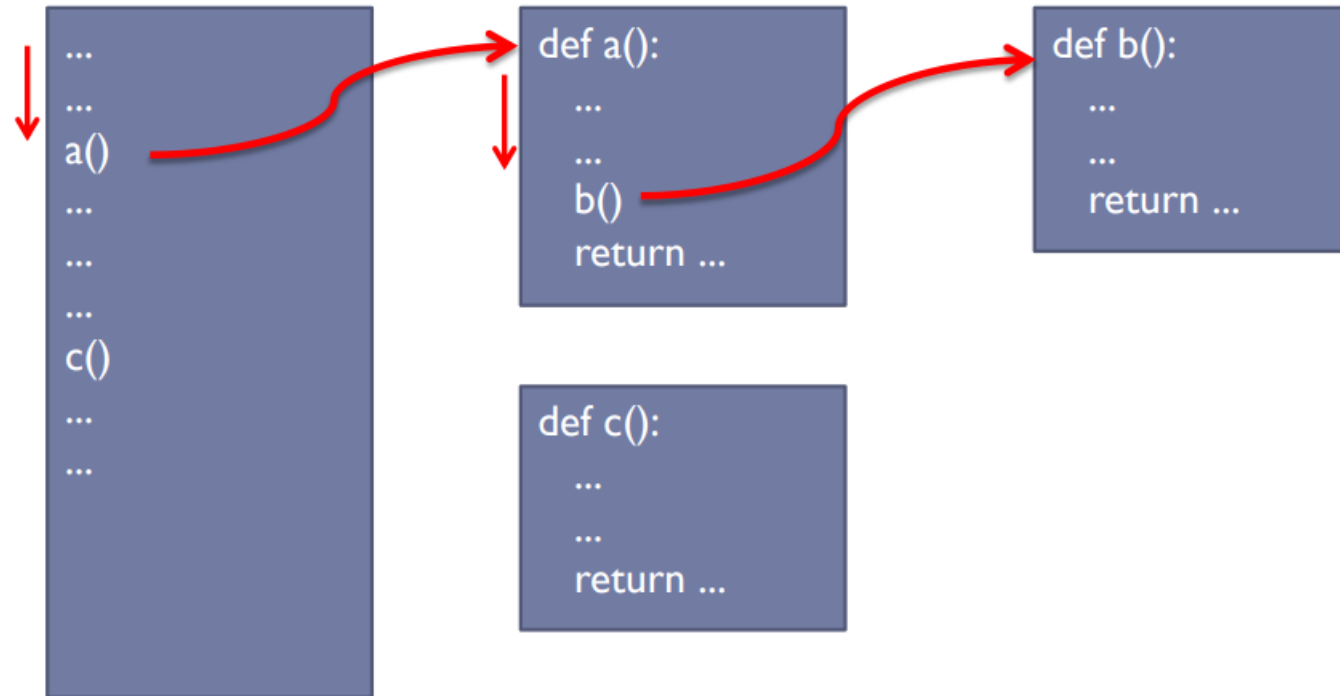
```
def b():  
    ...  
    ...  
    return ...
```

```
def c():  
    ...  
    ...  
    return ...
```

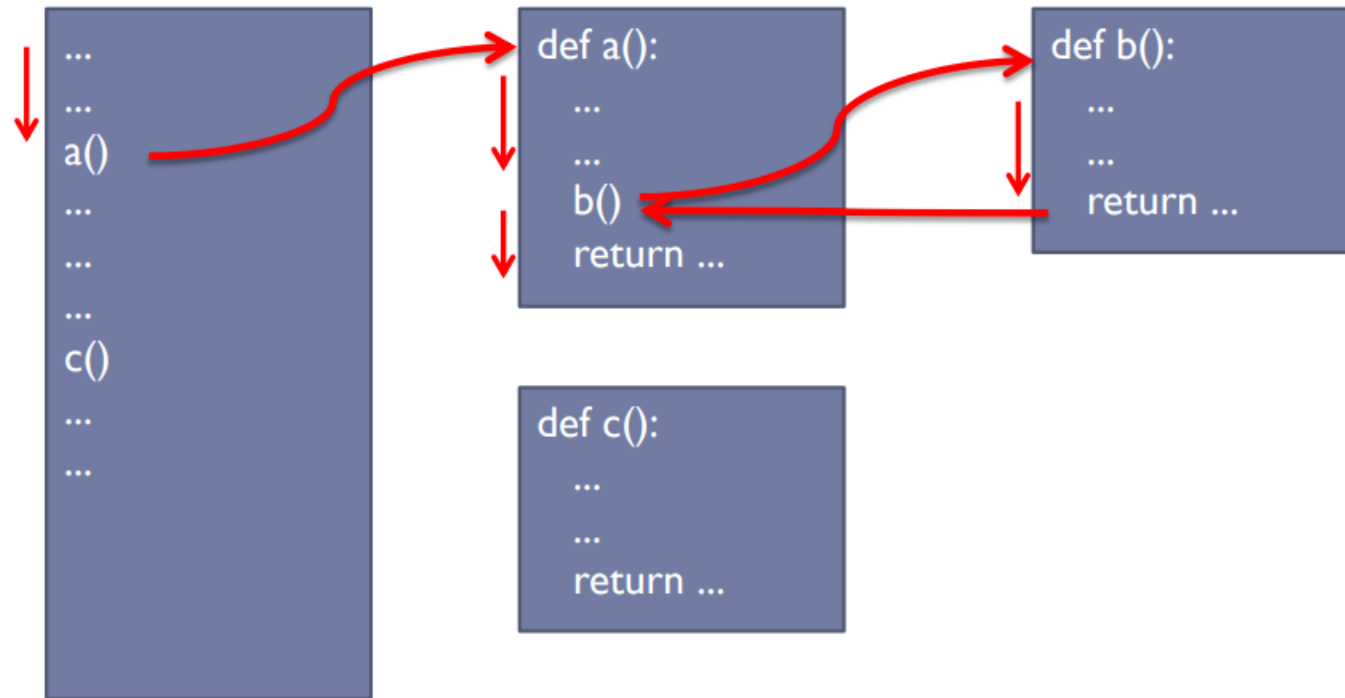
FLUXO DE EXECUÇÃO



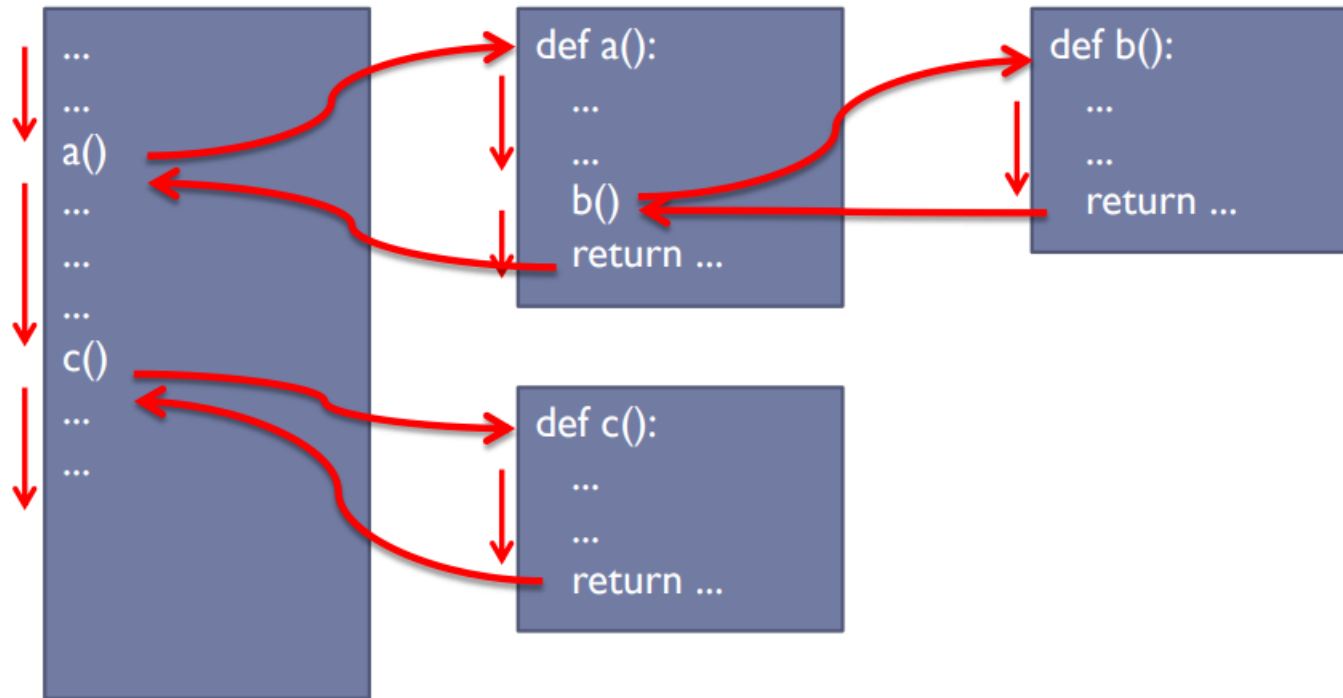
FLUXO DE EXECUÇÃO



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
```

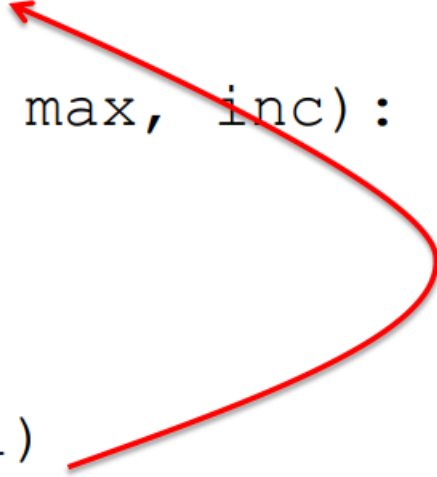
```
s = calcula_soma(0, 4, 1)  
↓ print(s)  
print(calcula_soma(10, 50, 10))
```

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



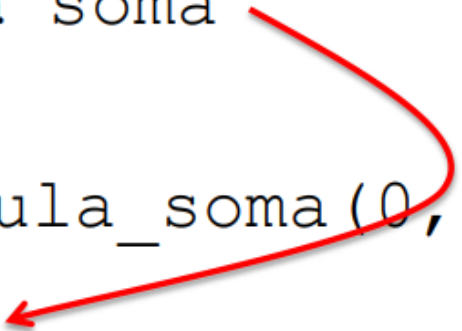
```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```

```
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

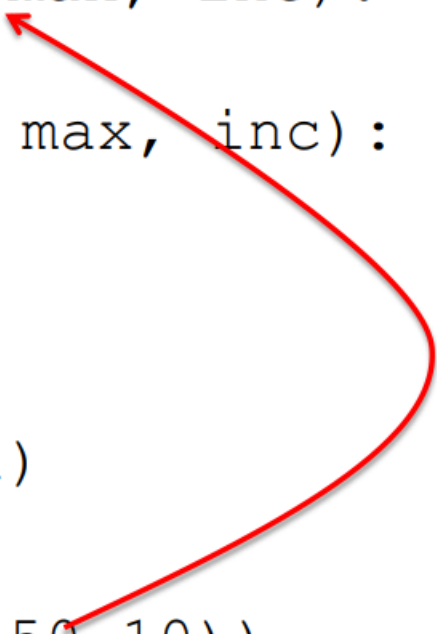
```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```

```
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



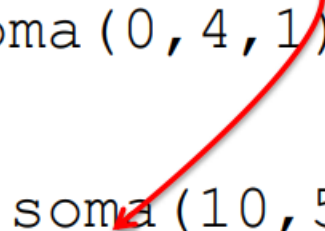
```
def calcula_soma(min, max, inc):  
    soma = 0  
    for i in range(min, max, inc):  
        soma = soma + i  
    return soma
```

```
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
```

10

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

```
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

0.5

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**?

PASSAGEM DE PARÂMETRO POR VALOR

- 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

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

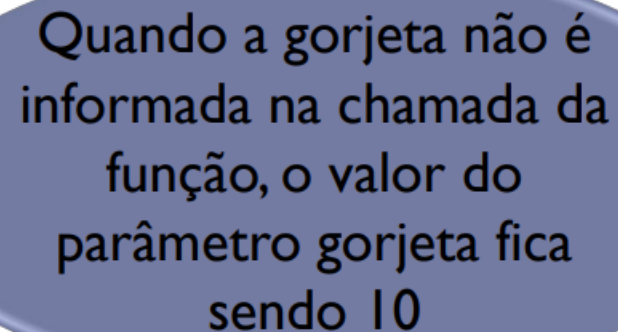
- Nem toda função precisa ter parâmetro
- Nesse caso, ao definir a função, deve-se abrir e fechar parênteses, sem informar nenhum parâmetro
- O mesmo deve acontecer na chamada da função

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)
```

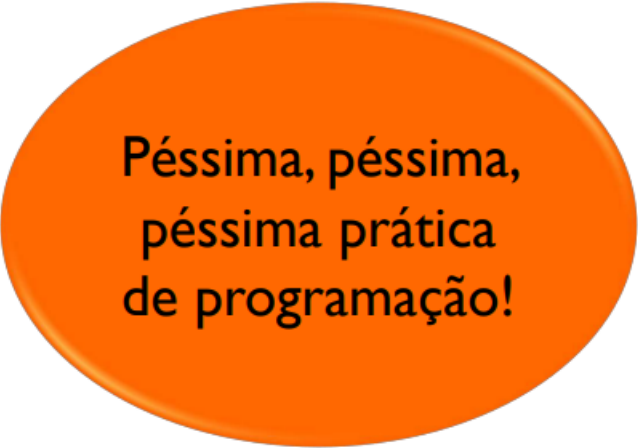


Péssima prática
de programação!

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)
```

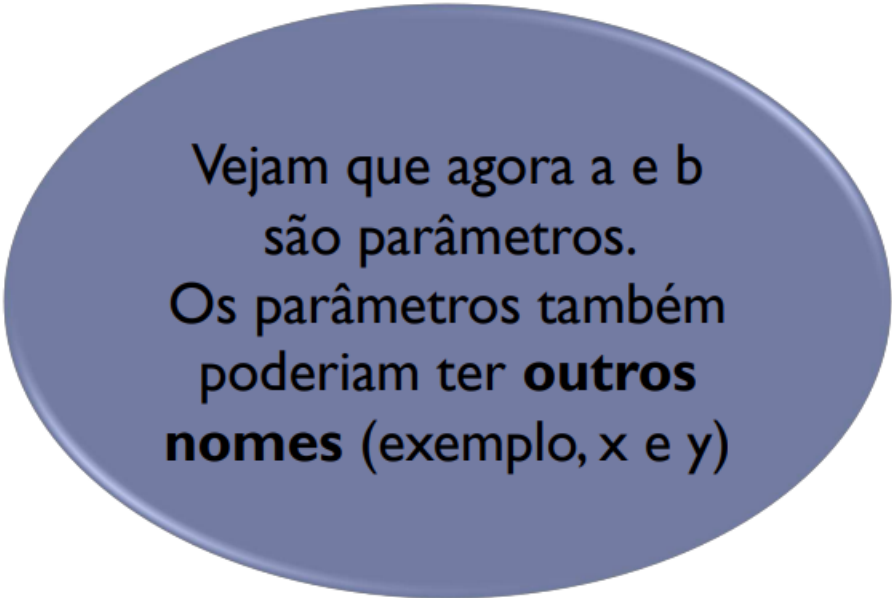


Péssima, péssima,
péssima prática
de programação!

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
- 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

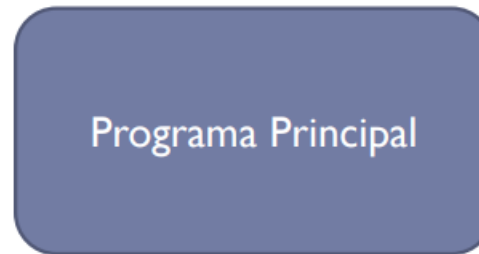
```
import utilidades  
  
x = 2  
y = 3  
print(utilidades.soma(x, y))  
print(utilidades.media(x, y))
```

VANTAGENS

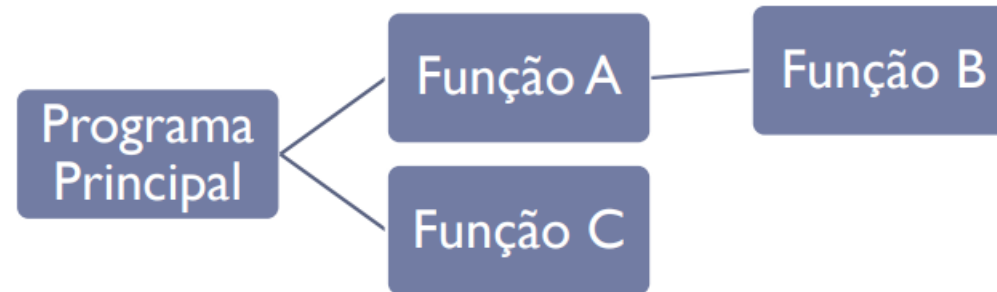
- Economia de código
 - Quanto mais repetição, mais economia
- Facilidade na correção de defeitos
 - Corrigir o defeito em um único local
- Legibilidade do código
 - Podemos dar nomes mais intuitivos a blocos de código
 - É como se criássemos nossos próprios comandos
- Melhor tratamento de complexidade
 - 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



REFERÊNCIAS

- Slides baseados no curso da Vanessa Murta

VETORES (LISTAS E TUPLAS)

PROGRAMAÇÃO APLICADA A MATEMÁTICA

Meirylene Avelino

meirylenerea@id.uff.br