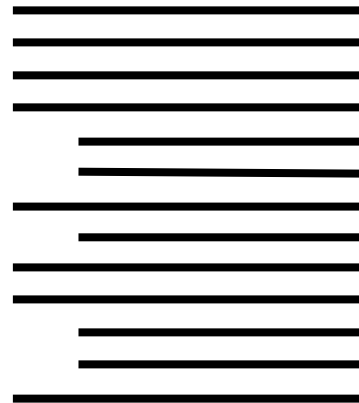


Lógica de Programação II

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

```
a = [1, 2, 3, 4, 5]
soma = 0
for i in range(len(a)):
    soma = soma + a[i]
media = soma/len(a)
print(media)
```

```
b = [10, 20, 30, 40]
soma = 0
for i in range(len(b)):
    soma = soma + b[i]
media = soma/len(b)
print(media)
```

Trecho se
repete 2
vezes

Exemplo 2

1. Ler um vetor A de 10 posições de inteiros
2. **Ordenar o vetor A**
3. Ler um vetor B de 10 posições de inteiros
4. **Ordenar o vetor B**
5. Multiplicar o vetor A pelo vetor B, e colocar o resultado num vetor C
6. **Ordenar o vetor C**

Operação de ordenação do vetor 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_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media
```

```
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```



Definição da função

Chamada da função

Chamada da função

Fluxo de Execução



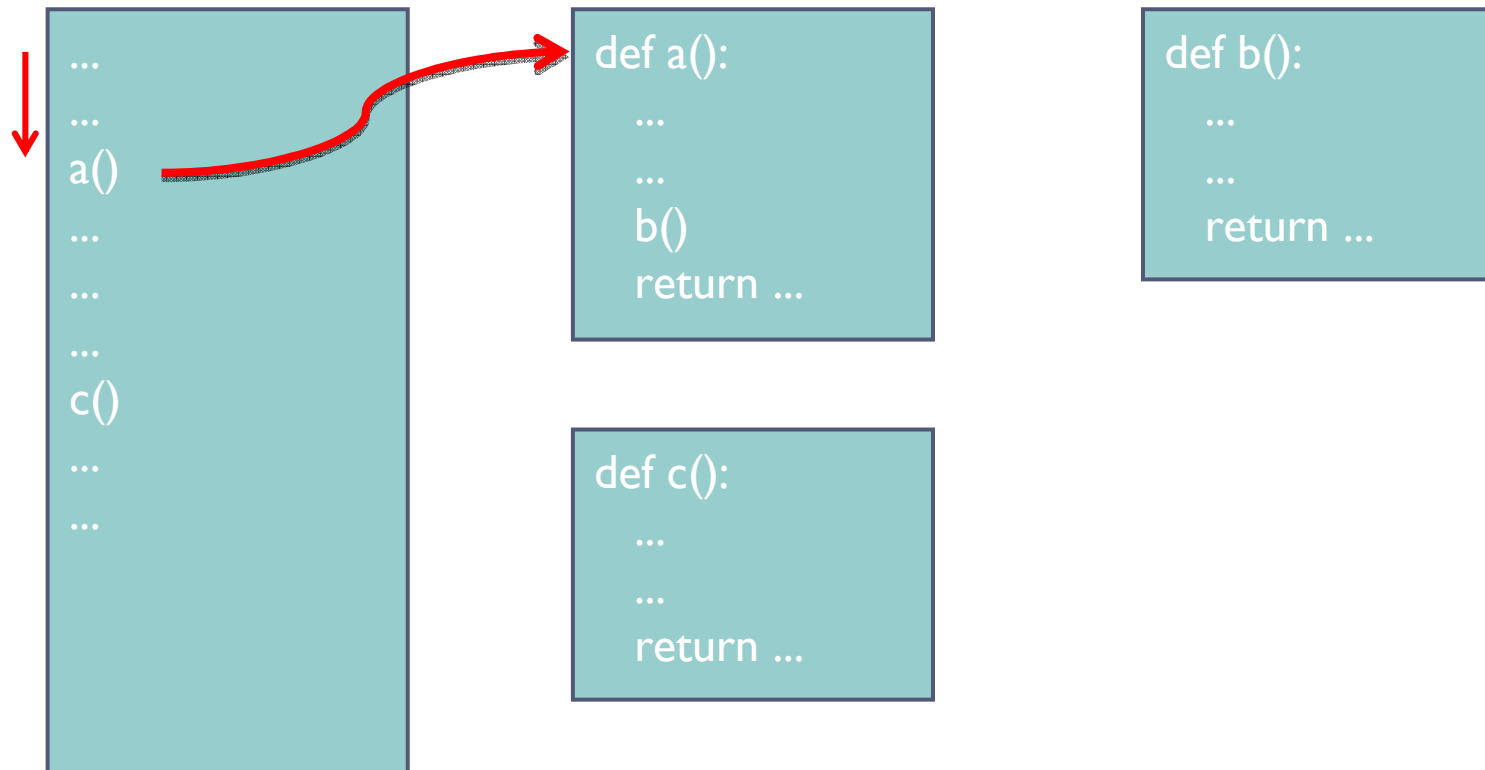
```
...  
...  
a()  
...  
...  
...  
c()  
...  
...
```

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

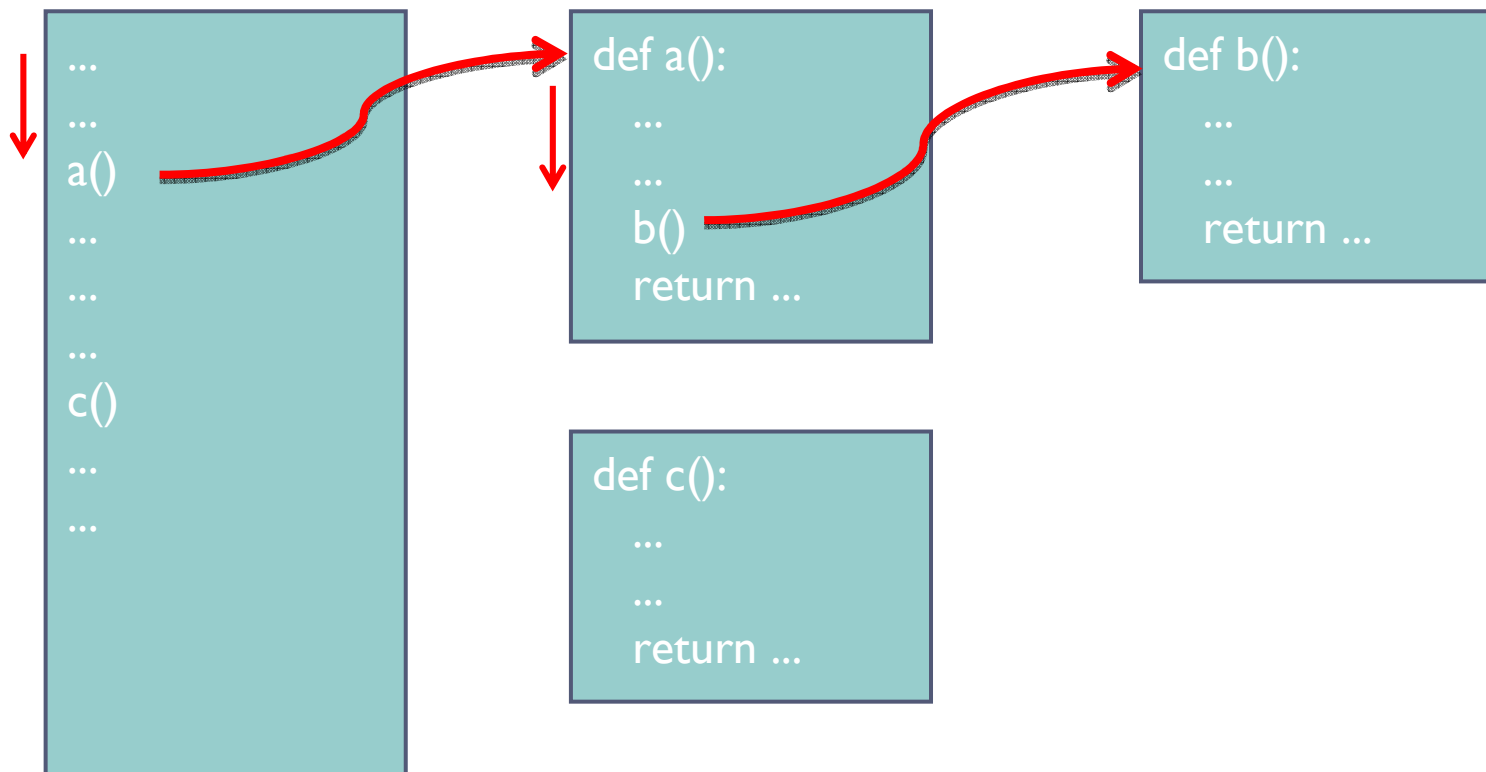
```
def b():  
    ...  
    ...  
    return ...
```

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

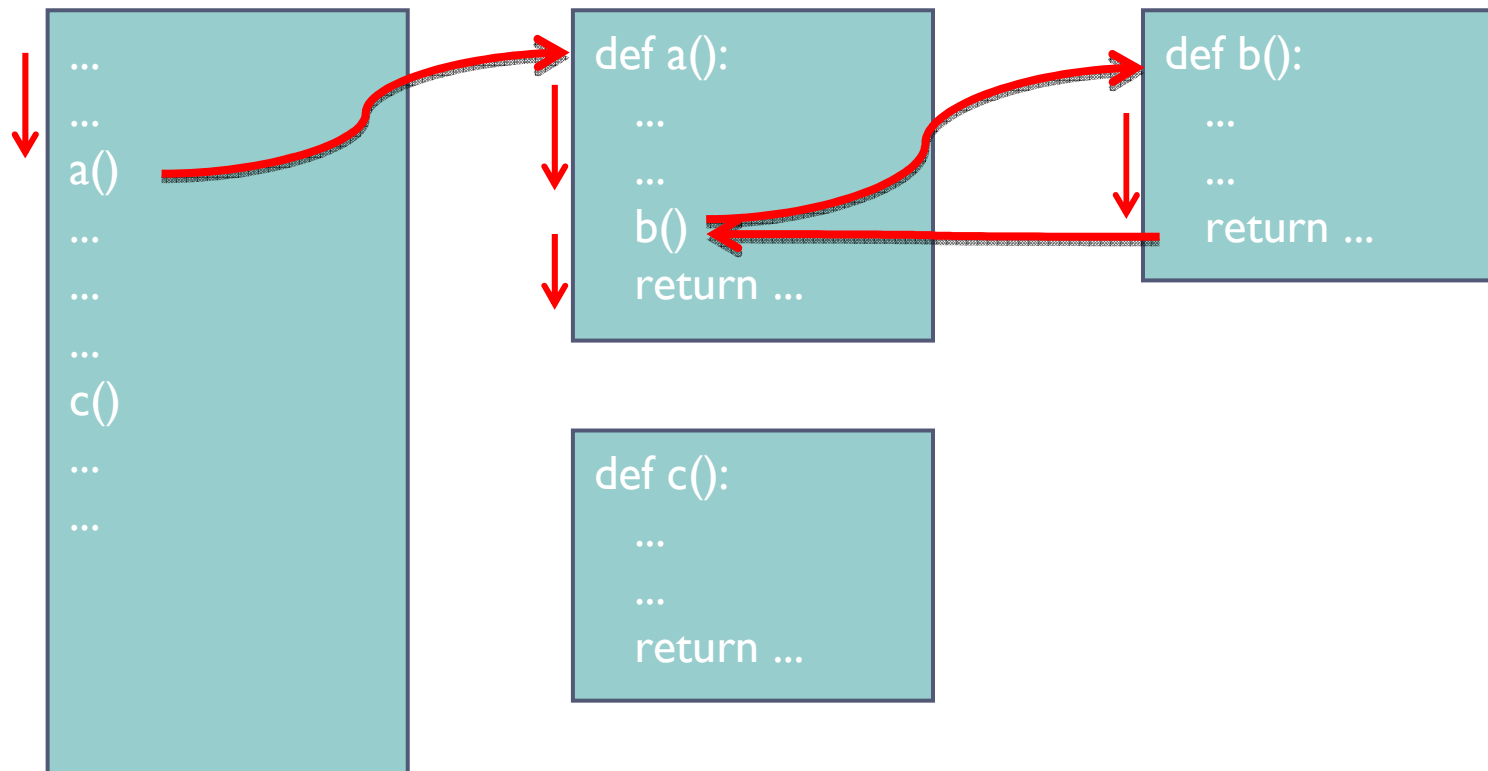

Fluxo de Execução



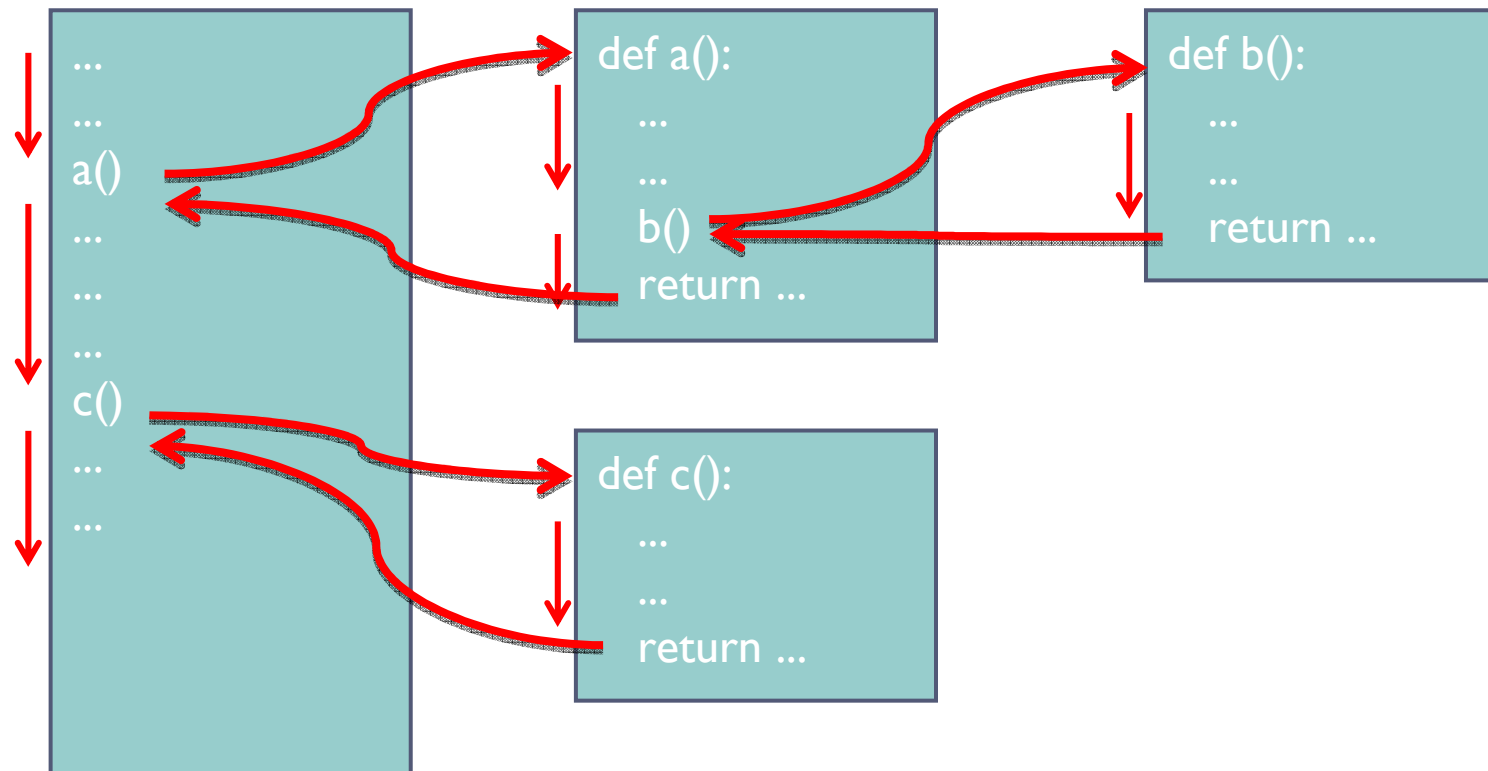
Fluxo de Execução



Fluxo de Execução



Fluxo de Execução



Fluxo de Execução

```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media
```

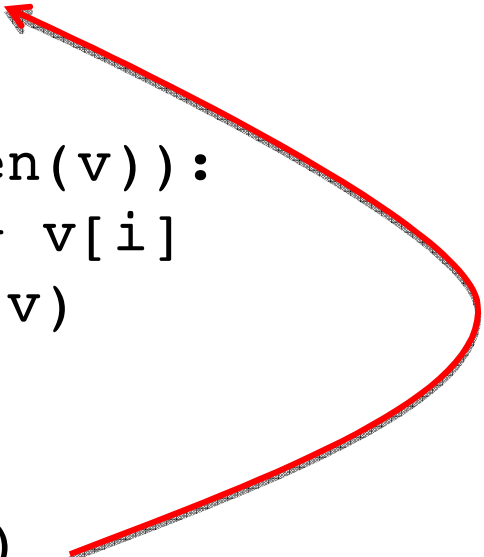
```
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```




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

Fluxo de Execução

```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media  
  
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```



Fluxo de Execução



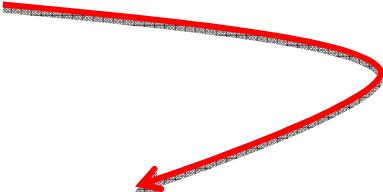
```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media
```

```
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```

Fluxo de Execução

```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media
```

```
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```

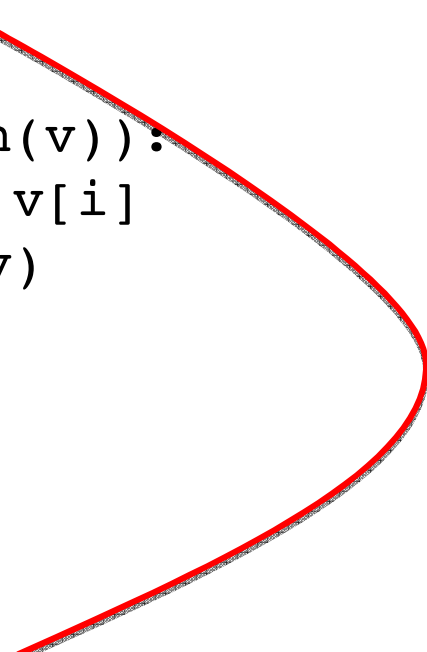


Fluxo de Execução


```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media  
  
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
↓  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```

Fluxo de Execução

```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media  
  
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```



Fluxo de Execução



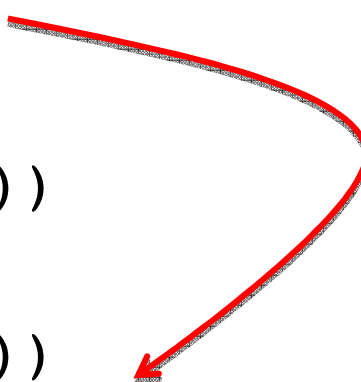
```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media
```

```
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```

Fluxo de Execução

```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media
```

```
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```



Declaração de Função

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

Exemplo:

```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media
```

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
 - Passam a existir no início da execução da função
 - 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
```

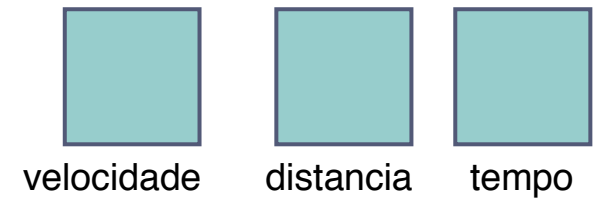
```
t = calcula_tempo(10, 5)  
print(t)  
d = calcula_distancia(5, 4)  
print(d)
```

Uso de Variáveis Globais x Variáveis Locais

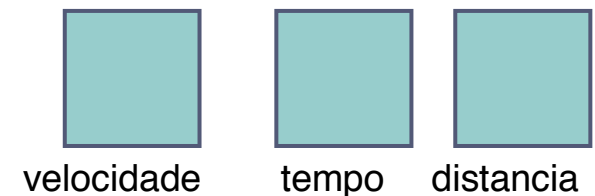
- Cuidado com variáveis globais
 - Dificultam o entendimento do programa
 - Dificultam a correção de erros no programa
 - Se a variável pode ser usada por qualquer função do programa, encontrar um erro envolvendo o valor desta variável pode ser muito complexo
- Recomendação
 - Sempre que possível, usar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

Escopo de Variáveis

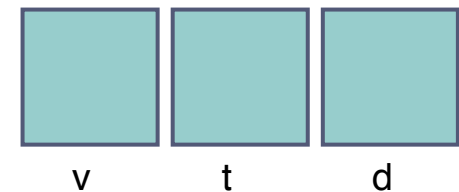
```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    return tempo
```



```
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)
```



Parâmetros

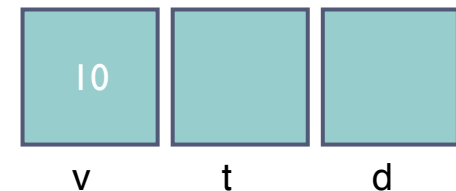
- Quando uma função é chamada, é necessário fornecer um valor para cada um de seus parâmetros
- Isso por ser feito informando o valor diretamente
 - `t = calcula_tempo(1, 2)`
- ou; Usando o valor de uma variável
 - `t = calcula_tempo(v, d)`

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)
```



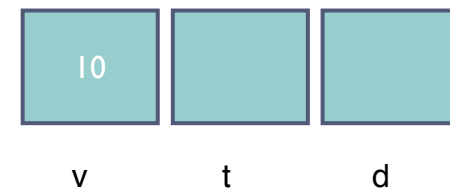
Passagem de Parâmetro

```
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

velocidade

5

distancia

0.5

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

- O valor da variável usada na chamada é **copiado** para a variável que representa o parâmetro na função
- **Alterações no valor parâmetro não são refletidas na variável correspondente àquele parâmetro no programa principal**

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

- 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
- Exceção: **vetores** (ou objetos) funcionam de forma diferente, pois o que é copiado é o **endereço** do vetor, e portanto qualquer alteração é refletida no programa principal → passagem de parâmetro por referência

Exemplo

```
def exclui_vetor(vetor,pos):  
    temp = []  
    for i in range(len(vetor)):  
        if i != pos:  
            temp.append(vetor[i])  
    vetor = temp  
    return len(vetor)  
v = [5, 4, 3, 2, 1]  
print(v)  
pos = 2  
m = exclui_vetor(v,pos)  
print(m)  
print(v)
```



O que será
impresso na tela?

Tipos de passagem de Parâmetro

- **Por valor:** o valor da variável na chamada é copiado para a variável da função.
 - Alterações não são refletidas na variável original
- **Por referência:** é como se o mesmo “escaninho” fosse usado.
 - Alterações são refletidas na variável original

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**
 - Nesse caso 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 pode atribuir seu resultado a uma variável

`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: '))
```

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 sempre 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)
```

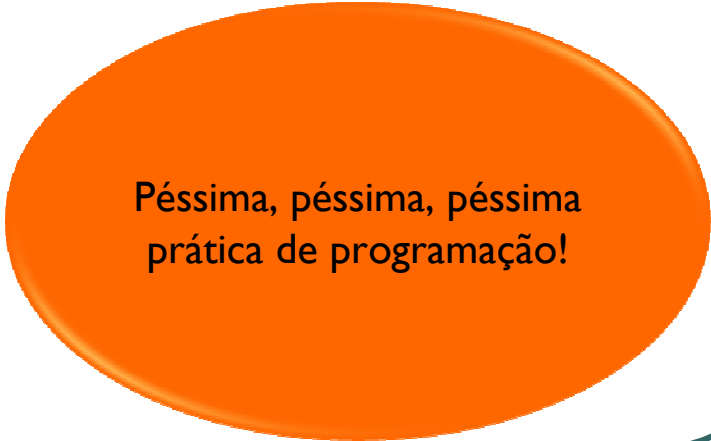


Péssima prática
de programação!

Exemplo: variável global modificada na função

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

```
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

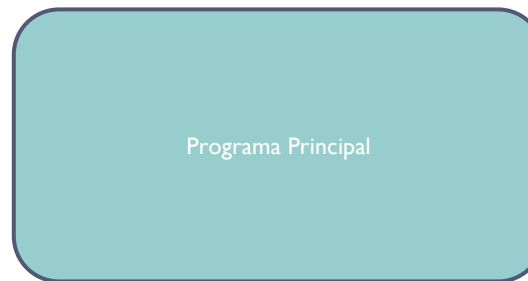
```
def soma(v):  
    soma = 0  
    for i in range(len(v)):  
        soma += v[i]  
    return soma  
  
def media(v):  
    return soma(v)/len(v)
```

Arquivo teste.py

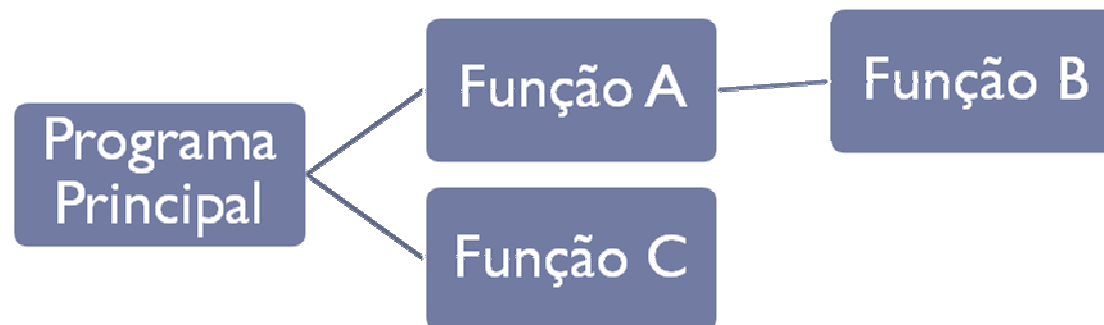
```
import utilidades  
  
v = [1, 3, 5, 7, 9]  
print(utilidades.soma(v))  
print(utilidades.media(v))
```

Dividir para conquistar

- Antes: um programa gigante



- Depois: vários programas menores



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!

Exercícios

Refaça a questão 2 da avaliação individual, usando uma função para calcular o total de faltas do campeonato, outra para calcular o time que fez mais faltas, e uma terceira para calcular o time que fez menos faltas. Antes de chamar essas funções, o programa deve permitir que o usuário adicione mais jogos ao campeonato.

Referências

- Slides baseados no curso de Programação de Computadores I da Prof. Vanessa Braganholo