# MC-102 — Aula 13
# Funções II

Eduardo C. Xavier

Instituto de Computação – Unicamp

10 de Novembro de 2020

# Roteiro

# Escopo de Variáveis: Variáveis locais e variáveis globais

- Cada variável em Python está associada à um escopo, que define onde ela foi criada, e quem pode acessar a variável.
- Lembre-se que em Python uma variável é criada quando associamos um objeto a esta.

# Escopo de Variáveis: Variáveis locais e variáveis globais

- Ao executar o programa **python main.py**, é criado um escopo associado ao módulo **main.py**, que simplificadamente chamaremos de escopo global.
- Variáveis criadas no bloco principal de **main.py** pertencem ao escopo de **main.py** (ao escopo global).
- No exemplo abaixo as variáveis **a**, e **b** pertencem ao escopo global. O nome **fun** também pertence ao escopo global (funções são objetos, logo o nome **fun** está associado à função definida):

```
a = 5
b = a + 1

def fun(d):
        c = 7 + a + d
        print(c)

fun(1)
```

# Escopo de Variáveis: Variáveis locais e variáveis globais

- Quando uma função é executada, também será criado um escopo para esta, e qualquer variável que sofre uma associação (seja criada ou alterada) dentro da função pertencerá ao escopo desta função.
- No exemplo abaixo as variáveis **c** e **d** pertence ao escopo da função **fun**. Note que a variável global **a** é visível dentro de **fun**.

```
a = 5
b = a + 1

def fun(d):
        c = 7 + a + d
        print(c)

fun(1)
```

# Escopo de Variáveis: Variáveis locais e variáveis globais

- Uma variável é chamada **local** se ela é associada (criada) dentro de uma função. Ela só pode ser acessada de dentro daquela função, e durante a execução da função. Após o término da execução da função a variável deixa de existir. **Variáveis parâmetros também são variáveis locais**.
- Uma variável é chamada **global** se ela é associada (criada) no bloco principal fora de qualquer função. Essa variável é visível por todas as funções do módulo corrente.
- A regra de escopo em Python é bem simples:
  - As variáveis globais são visíveis por todas as funções.
  - As variáveis locais são visíveis apenas na função onde foram criadas.

# Organização de um Programa

- Em geral um programa é organizado da seguinte forma:

```
import bibliotecas

variáveis globais

def main():
  variáveis locais
  Comandos Iniciais

def fun1(Parâmetros):
  variáveis locais
  Comandos

def fun2(Parâmetros):
  variáveis locais
  Comandos

...
...
main()
```

# Exemplos: Variáveis locais e globais

- Quais são as variáveis locais e globais do programa abaixo? O que será impresso?

```
def f1(a):
    print('f1', a+x)

def f2(a):
    c=10
    print('f2', a+x+c)

x=4
f1(3)
f2(3)
print(x)
```

# Exemplos: Variáveis locais e globais

```
def f1(a):
    print('f1', a+x)

def f2(a):
    c=10
    print('f2', a+x+c)

x=4
f1(3)
f2(3)
print(x)
```

- Variáveis globais: Apenas **x**. Note que **f1** e **f2** podem acessar a variável **x**.
- Variáveis locais: (1) De **f1** é **a**; (2) De **f2** são **a** e **c**.
- A saída do programa será:

```
f1 7
f2 17
4
```

# Exemplos: Escopo de Variáveis: Variáveis locais e variáveis globais

- O que será impresso pelo programa? Quais são as variáveis locais e globais?

```
a = 5
b = a + 1

def fun(a):
        c = b + a
        a = a + 1
        print('fun c', c)
        print('fun a', a)

fun(1)

print('a', a)
print('b', b)
```

# Exemplos: Escopo de Variáveis: Variáveis locais e variáveis globais

- Variáveis globais: **a** e **b**.
- Variáveis locais: De **fun** são **a** e **c**.

```
a = 5
b = a + 1

def fun(a):
        c = b + a
        a = a + 1
        print('fun a', c)
        print('fun b', a)

fun(1)

print('a', a)
print('b', b)
```

- Será impresso:

```
fun 7
fun 2
a 5
b 6
```

# Exemplos: Variáveis locais e globais

- O que será impresso pelo programa? Quais são as variáveis locais e globais?

```
def f1(a):
    x = 10
    print('f1',a+x)

def f2(a):
    c=10
    print('f2',a+x+c)


x=4
f1(3)
f2(3)
print(x)
```

# Exemplos: Variáveis locais e globais

- Variáveis globais: apenas **x**.
- Variáveis locais: (1) De **f1** temos **x** e **a**; (2) De **f2** temos **a** e **c**.

```
def f1(a):
    x = 10
    print('f1',a+x)

def f2(a):
    c=10
    print('f2',a+x+c)


x=4
f1(3)
f2(3)
print(x)
```

- Será impresso:

```
f1 13
f2 17
4
```

# Exemplos: Variáveis locais e globais

Veja este outro exemplo:

```
def f1(a):
    print(a+x)

def f3(a):
    x=x+1
    print(a+x)

x=4
f1(3)
f3(3) # este comando vai dar um erro
```

A saída será:

```
7
Traceback (most recent call last):
  File "teste.py", line 10, in <module>
    f3(3) # este comando vai dar um erro
  File "teste.py", line 5, in f3
    x=x+1
UnboundLocalError: local variable 'x' referenced before assignment
```

O que aconteceu???

# Exemplos: Variáveis locais e globais

Veja este outro exemplo:

```
def f1(a):
    print(a+x)

def f3(a):
    x=x+1
    print(a+x)

x=4
f1(3)
f3(3) # este comando vai dar um erro
```

- Na função **f3** associamos **x** a um novo objeto, e pela regra de Python esta variável é portanto local. O erro ocorre pois está sendo usado uma variável local **x** antes dela ser criada (o **x** do lado direito da atribuição)!

# Exemplos: Variáveis locais e globais

- Para que **f1** use **x** global devemos especificar isto utilizando o comando **global**.

```
def f1(a):
    print(a+x)

def f3(a):
    global x
    x=x+1
    print(a+x)

x=4
f1(3)
f3(3) # sem erro
print(x)
```

- A saída neste exemplo será:

```
7
8
5
```

- Note que o valor de **x** global foi alterado pela função **f3**.

# Exemplos: Variáveis locais e globais

- O que será impresso pelo programa? Quais são as variáveis locais e globais?

```
def f4(a):
    global c
    c = 10 + a
    print('f4 c', c)
    print('f4 soma', a+x+c)

x = 4
c = -1
f4(1)
print("c", c)
```

# Variáveis locais e globais

- Variáveis globais: **x** e **c**.
- Variáveis locais de **f4**: apenas **a**.

```
def f4(a):
    global c
    c = 10 + a
    print("f4 c",c)
    print(a+x+c)

x = 4
c = -1
f4(1)
print("c", c)
```

- Será impresso:

```
f4 c 11
f4 soma 16
c 11
```

# Variáveis locais e variáveis globais

- O uso de variáveis globais deve ser evitado pois é uma causa comum de erros:
  - Partes distintas e funções distintas podem alterar a variável global, causando uma grande interdependência entre estas partes.
- A legibilidade do seu código piora com o uso de variáveis globais:
  - Ao ler uma função que usa uma variável global é difícil inferir seu valor inicial e portanto qual o resultado da função sobre a variável global.

# Variáveis locais e variáveis globais

Um importante princípio no desenvolvimento de programas:

- **Separation of Concern:** separar o programa em seções isoladas, onde cada seção resolve um sub-problema específico. No contexto de mc102, o trabalho feito por uma seção é acessado por chamadas de funções.

# Variáveis locais e variáveis globais

Email do Jeff Bezos para funcionários da Amazon sobre modularização (*Separation of Concern*) em 2002:

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols – doesn't matter. Bezos doesn't care.
- All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- Anyone who doesn't do this will be fired.
- Thank you; have a nice day!

# Listas em funções

```
def f1(a):
    a.append(3)

a = [1,2]
f1(a)
print(a)
```

- Neste caso mesmo havendo uma variável local **a** de **f1** e uma global **a**, o conteúdo de **a** global é alterado.
- O que aconteceu???

Saída:

```
[1, 2, 3]
```

# Listas em funções

```
def f1(a):
    a.append(3)

a = [1,2]
f1(a)
print(a)
```

- Lembre-se que **a** local de **f1** recebe o identificador da lista de **a** global. Como uma lista é mutável, o seu conteúdo é alterado.

Saída:

```
[1, 2, 3]
```

# Listas em funções

```
def f1(a):
    a = [10,10]

a = [1,2]
f1(a)
print(a)
```

- O que será impresso neste caso???.

# Listas em funções

```
def f1(a):
    a = [10,10]

a = [1,2]
f1(a)
print(a)
```

- Neste caso a variável **a** local de **f1** é associada com uma nova lista.
- Logo a variável **a** global não é alterada.

Saída:

```
[1, 2]
```

# Listas em funções

```
def f1():
    global a
    a = [10,10]

a = [1,2]
f1()
print(a)
```

- O que será impresso???

# Listas em funções

```
def f1():
    global a
    a = [10,10]

a = [1,2]
f1()
print(a)
```

- Neste caso **a** de **f1** é global e portanto corresponde a mesma variável fora da função.
- A variável **a** tem seu identificador alterado dentro da função para uma nova lista com conteúdo [10, 10].

Saída:

```
[10, 10]
```

# Exercício

- Escreva uma função em Python para computar a raiz quadrada de um número positivo. Use a idéia abaixo, baseada no método de aproximações sucessivas de Newton. A função deverá retornar o valor da vigésima aproximação.

Seja Y um número, sua raiz quadrada é raiz da equação

$$f(x) = x^2 - Y.$$

A primeira aproximação é $x_1 = Y/2$. A $(n+1)$-ésima aproximação é

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

# Exercício

- Escreva uma função em Python que recebe como parâmetros duas listas representando matrizes e computa a soma destas. O protótipo da função deve ser:

```
def somaMat(mat1, mat2)
```

- Você pode obter o número de linhas de **mat1** com **len(mat1)** e o número de colunas com **len(mat1[0])**.

# Exercício

- Escreva uma função em Python que recebe como parâmetros duas listas representando matrizes e computa a multiplicação destas. O protótipo da função deve ser:

```
void multiplicaMat(mat1, mat2)
```

- Você pode obter o número de linhas de **mat1** com **len(mat1)** e o número de colunas com **len(mat1[0])**. Se os tamanhos das matrizes forem incompatíveis a função deve devolver **None**.