MC-102 — Aula 14
Funções II

Instituto de Computação – Unicamp

29 de Setembro de 2016

Roteiro

Variáveis locais e variáveis globais

- Uma variável é chamada local se ela é **criada ou alterada** dentro de uma função. Nesse caso, ela existe somente dentro daquela função, e após o término da execução da mesma a variável deixa de existir. **Variáveis parâmetros também são variáveis locais**.
- Uma variável é chamada global se ela for criada fora de qualquer função. Essa variável pode ser visível por todas as funções. Qualquer função pode alterá-la.

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()
```

Escopo de variáveis

- O escopo de uma variável determina de quais partes do código ela pode ser acessada, ou seja, de quais partes do código a variável é visível.
- 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.

Variáveis locais e globais

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

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


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

- Tanto **f1** quanto **f2** usam a variável **x** que é global pois foi criada fora das funções.
- A saída do programa será:

```
7
17
4
```

Variáveis locais e globais

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

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


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

- Neste outro exemplo **f1** cria uma variável local **x** com valor 10. O valor de **x** global permanece com 4.
- A função **f2** por outro lado continua acessando a variável global **x**.
- A saída do programa será:

```
13
17
4
```

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???

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** é alterado o valor de **x**, e pela regra de Python esta variável é portanto uma variável local. O erro ocorre pois está sendo usado uma variável local **x** antes dela ser criada!

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**.

Variáveis locais e globais

```
def f3(a):
    c=10
    print(a+x+c)
x=4
print(c)
```

- A variavél **c** foi criada dentro da função e ela só existe dentro desta. Ela é uma variável **local** da função **f3**.

Saída:

```
Traceback (most recent call last):
  File "teste.py", line 5, in <module>
    print(c)
NameError: name 'c' is not defined
```

Variáveis locais e globais

```
def f4(a):
    c=10
    print("c de f4 :",c)
    print(a+x+c)
x=4
c=-1
f4(1)
print("c global:", c)
```

- Neste caso existe uma variável **c** no programa principal e uma variável local **c** pertencente à função **f4**.
- Alteração no valor da variável local **c** dentro da função não modifica o valor da variável global **c**, a menos que esta seja declarada como **global**.

Saída:

```
c de f4 : 10
15
c global: -1
```

Variáveis locais e globais

```
def f4(a):
    global c
    c=10
    print("c de f4 :",c)
    print(a+x+c)
x=4
c=-1
f4(1)
print("c global:", c)
```

- Neste caso a variável **c** de dentro da função **f4** foi declarada como global.
- Portanto é alterado o conteúdo da variável **c** fora da função.

Saída:

```
c de f4 : 10
15
c global: 10
```

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 distintas de código.
- A legibilidade do seu código também 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.

Exemplo Utilizando Funções

- Em uma das aulas anteriores vimos como testar se um número em **candidato** é primo:

```
divisor = 2
eprimo=True
while divisor<=candidato/2 :
  if candidato % divisor == 0:
    eprimo=False
    break
  divisor+=1

if eprimo:
  print(candidato)
```

Exemplo Utilizando Funções

- Depois usamos este código para imprimir os *n* primeiros números primos:
- Veja no próximo slide.

Exemplo Utilizando Funções

```
n=int(input("Digite numero de primos a imprimir:"))
if n>=1:
    print("2")
    primosImpr=1
    candidato=3
    while primosImpr < n:
        divisor = 2
        eprimo=True
        while divisor <= candidato/2 :
            if candidato % divisor == 0:
                eprimo=False
                break
            divisor+=1

        if eprimo:
            print(candidato)
            primosImpr+=1

        candidato=candidato+2 #Testa proximo numero
```

Exemplo Utilizando Funções

- Podemos criar uma função que testa se um número é primo ou não (note que isto é exatamente um bloco logicamente bem definido).
- Depois fazemos chamadas para esta função.

Exemplo Maior

```
def ePrimo(candidato):
  divisor = 2
  eprimo=True
  while divisor <= candidato/2 :
    if  candidato % divisor == 0:
      eprimo=False
      break
    divisor+=1

  return eprimo
```

Exemplo Utilizando Funções

```
def main():
  n = int(input("Digite numero de primos a imprimir:"))
  if n >= 1:
    print("2")
    primosImpr = 1
    candidato = 3
    while primosImpr < n:
      if ePrimo(candidato):
        print(candidato)
        primosImpr+=1
      candidato=candidato+2

def ePrimo(candidato):
  divisor = 2
  eprimo=True
  while divisor <= candidato/2 :
    if  candidato % divisor == 0:
      eprimo=False
      break
    divisor+=1

  return eprimo

main()
```

Exemplo Utilizando Funções

- O código é mais claro quando utilizamos funções.
- Também é mais fácil fazer alterações.
- Exemplo: queremos otimizar o teste de primalidade, e para tanto não vamos testar todos os divisores $2, \ldots, (candidato/2)$.
  - Testar se número é par maior que 2 (não é primo).
  - Se for ímpar, testar apenas os divisores ímpares $3, 5, 7, \ldots$.
- O uso de funções facilita modificações no código. Neste caso altera-se apenas a função **ePrimo**.

Exemplo Maior

Função **ePrimo** é alterada para:

```
def ePrimo(candidato):
  if (candidato>2) and (candidato % 2 == 0) :
    return False #par > 2 não é primo

  divisor = 3
  eprimo=True
  while divisor <= candidato/2 :
    if candidato % divisor == 0:
      eprimo=False
      break
    divisor += 2 #testa apenas divisores ímpares

  return eprimo
```

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** recebe uma nova lista, e portanto um novo identificador.
- 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**.