Programação Script

Funções (parte 2)

Aula 06

Prof. Felipe A. Louza

Roteiro

Roteiro

Funções Podem Invocar Funções

Vimos como definir funções na aula anterior:

```
1  def funcao1():
2    #cmd C
3    funcao2();
4    #cmd K
5
6    ...
7
8  def funcao2():
9    #cmd D
10   funcao3();
11   #cmd J
```

```
1  def funcao3():
2    #cmd E
3    funcao4()
4    #cmd I
5
6    ...
7
8  def funcao4():
9    #cmd F
10   #cmd G
11   #cmd H
```

```
1  def main():
2    # cmd A
3    # cmd B
4    funcao1()
5
6  main()
```

Variáveis locais e globais

Escopo de variáveis:

- Cada variável em **Python** está associada à um escopo, que define onde ela foi criada, e quem pode acessar a variável.

```
1 a = 5
2 b = a + 1
3
4 def fun(d):
5   c = 7 + a + d
6   print(c)
7
8 fun(1)
```

- Lembre-se que em **Python** uma variável é criada quando associamos um objeto a esta.

Variáveis locais e globais

Escopo global:

- Variáveis criadas no **bloco principal** do programa pertencem ao escopo global.

```
1 a = 5
2 b = a + 1
3
4 def fun(d):
5   c = 7 + a + d
6   print(c)
7
8 fun(1)
```

- Todas as funções tem acesso à elas.
- Ou seja, é possível modifica-las dentro de uma função.

Variáveis locais e globais

Escopo local:

- Variáveis criadas **dentro de funções** pertencem ao escopo local da função.

```
1 a = 5
2 b = a + 1
3
4 def fun(d):
5   c = 7 + a + d
6   print(c)
7
8 fun(1)
```

- Apenas a função que a criou tem acesso à ela.
- Variáveis atribuidas por parâmetros também são locais.

Variáveis locais e globais

Escopo local:

- Importante:
  - Após o término da execução da função a variável deixa de existir.

```
1 a = 5
2 b = a + 1
3
4 def fun(d):
5   c = 7 + a + d
6   print(c)
7
8 fun(1)
```

Variáveis locais e globais

Em resumo:

- As variáveis globais são visíveis por todas as funções.
- As variáveis locais são visíveis apenas na função onde foram criadas.

```
1  a = 5
2
3  def fun1():
4    b = 1
5    print(a)
6    print(b)
7
8  def fun2():
9    c = 3
10   print(a)
11   print(c)
12
13 print(a)
```

Variáveis locais e globais

Regras de escopo:

- Variáveis em escopos diferentes podem ter os mesmos nomes.

```
1  a = 5
2
3  def fun1():
4    a = 1
5    print(a)
6
7  def fun2():
8    a = 3
9    print(a)
10
11 print(a)
```

Variáveis locais e globais

Regras de escopo:

- Variáveis locais têm precedência sobre variáveis globais

```
1  a = 5
2  def fun1():
3    a = 1
4    print(a)
5
6  def fun2():
7    a = 3
8    print(a)
9
10 fun1()
11 fun2()
12 print(a)
```

- Dentro de cada função o que vale é a variável local.

Variáveis locais e globais

Regras de escopo:

- Caso não exista variável local com o identificador, utiliza-se a variável global.

```
1  a = 5
2  def fun1():
3    print(b)
4
5  def fun2():
6    print(b)
7
8  b = 7
9  fun1()
10 b = 9
11 fun2()
12 print(b)
```

Organização de um Programa

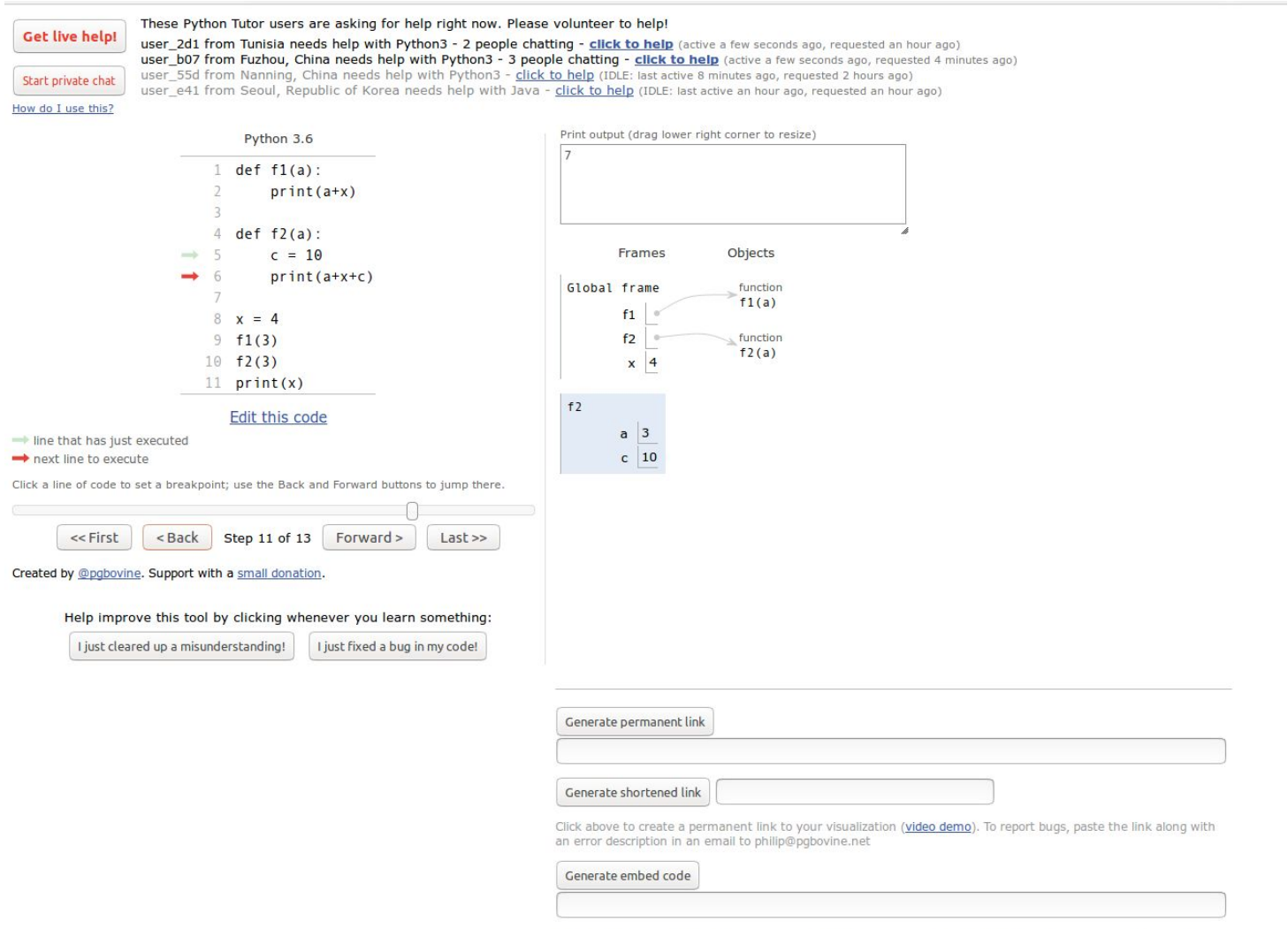
Em geral, um programa em **Python** é organizado da seguinte forma:

```
1  import bibliotecas
2
3  variáveis globais
4
5  def main():
6    variáveis locais
7    Comandos Iniciais
8
9  def fun1(Parâmetros):
10   variáveis locais
11   Comandos
12
13 def fun2(Parâmetros):
14   variáveis locais
15   Comandos
16
17 ...
18 main()
```

- Atenção: nenhum comando é definido fora das funções.

pythontutor.com

http://www.pythontutor.com/visualize.html#mode=edit



Roteiro

Exemplo 1

Quais são as variáveis locais e globais do programa abaixo?

```
1  x=4
2
3  def main():
4    f1(3)
5    f2(3)
6    print(x)
7
8  def f1(a):
9    print('f1', a+x)
10
11 def f2(a):
12   c=10
13   print('f2', a+x+c)
14
15 main()
```

- O que será impresso?

Exemplo 2

Quais são as variáveis locais e globais do programa abaixo?

```
1  a = 5
2  b = a + 1
3
4  def main():
5    fun(1)
6    print('a', a)
7    print('b', b)
8
9  def fun(a):
10   c = b + a
11   a = a + 1
12   print('fun c', c)
13   print('fun a', a)
14
15 main()
```

- O que será impresso?

Exemplo 3

Quais são as variáveis locais e globais do programa abaixo?

```
1  x=4
2
3  def main():
4    f1(3)
5    f2(3)
6    print(x)
7
8  def f1(a):
9    x = 10
10   print('f1',a+x)
11
12 def f2(a):
13   c=10
14   print('f2',a+x+c)
15
16 main()
```

- O que será impresso?

Roteiro

Variáveis locais e globais

Cuidado:

- Erro comum:

```
1  a = 5
2  def fun1():
3    a = 1
4    print(a)
5
6  def fun2():
7    print(a)
8    a = 3
9
10 fun1()
11 fun2() # este comando vai dar um erro
12 print(a)
```

- O que aconteceu??

Variáveis locais e globais

Mensagem de erro:

- `UnboundLocalError: local variable 'a' referenced before assignment`

```
1  a = 5
2  def fun1():
3    a = 1
4    print(a)
5
6  def fun2():
7    print(a)
8    a = 3
9
10 fun1()
11 fun2() # este comando vai dar um erro
12 print(a)
```

  - Na função `fun2()`, o comando `a = 3`, faz com que a função entenda `a` como uma **nova** variável local.

Variáveis locais e globais

Cuidado:

- Outro erro:

```
1  a = 5
2  def fun1():
3    a = 1
4    print(a)
5
6  def fun2():
7    a = a + 3
8    print(a)
9
10 fun1()
11 fun2() # este comando vai dar um erro
12 print(a)
```

- E agora, o que aconteceu??

Variáveis locais e globais

Mensagem de erro:

- `UnboundLocalError: local variable 'a' referenced before assignment`

```
1  a = 5
2  def fun1():
3    a = 1
4    print(a)
5
6  def fun2():
7    a = a + 3
8    print(a)
9
10 fun1()
11 fun2() # este comando vai dar um erro
12 print(a)
```

  - Ocorre um erro pois a variável a está sendo usada antes de ser criada (o a do lado direito da atribuição).

O comando `global`

Podemos utilizar ou criar uma variável global dentro de uma função com o comando `global`.

```
1  a = 5
2  def fun1():
3    a = 1
4    print(a)
5
6  def fun2():
7    global a
8    a = a + 3
9    print(a)
10
11 fun1()
12 fun2() # sem erro
13 print(a)
```

- Agora, que o valor de `a` global foi alterado pela função `fun2`.

Exemplo 4

Quais são as variáveis locais e globais do programa abaixo?

```
1  x = 4
2  c = -1
3
4  def main():
5    f4(1)
6    print("c", c)
7
8  def f4(a):
9    global c
10   c = 10 + a
11   print("f4 c",c)
12   print(a+x+c)
13
14 main()
```

- O que será impresso?

Exemplo 5

Podemos definir uma variável global dentro de uma função

```
1  x = 4
2
3  def main():
4    f4(1)
5    print("c", c)
6
7  def f4(a):
8    global c
9    c = 10 + a
10   print("f4 c",c)
11   print(a+x+c)
12
13 main()
```

- O que será impresso?

Exemplo 5

Ainda assim, é preciso ter atenção para não usar uma variável antes dela ser criada.

```
1  x = 4
2
3  def main():
4    f4(1)
5    print("c", c)
6
7  def f4(a):
8    global c
9    c = c + a
10   print("f4 c",c)
11   print(a+x+c)
12
13 main()
```

- `NameError: name 'c' is not defined`

Variáveis locais e variáveis globais

Importante:

- 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 no 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 qual o resultado da função sobre a variável global.

Roteiro

Retorno de valores

Em **Python** funções podem retornar mais de um valor.

```
1  def main():
2    a = int(input("Digite o primeiro valor: "))
3    b = int(input("Digite o segundo valor: "))
4    a, b = inverte(a, b)
5    print("Primeiro =", a)
6    print("Segundo  =", b)
7
8  def inverte(a, b):
9    return b, a
10
11 main()
```

- O que será impresso pelo programa?

Retorno de valores

Na verdade, o **Python** retorna um objeto do tipo tupla.

```
1  def main():
2    x = 10
3    y = 3
4    res = func(x, y)
5    print(res)
6    print(type(res))
7
8  def func(x, y):
9    return x+y, x-y, x*y, x/y, x%y, "hello"
10
11 main()
```

- O que será impresso pelo programa?

Funções Aninhadas

Em **Python** é possível definir funções dentro de funções.

- Situação onde isso faz sentido:
  - A função interna é usada apenas pela função externa.

```
1  def fun1(a, b=10):
2
3    def fun2(a):
4      if a < 0:
5        a = -1 * a
6      return a
7
8    a = fun2(a)
9    i = 0
10   while(i < b):
11     print(a**i)
12     i = i+1
13
14 fun1(2, 5)
15 print('----')
16 fun1(-2,5)
```

  - O que será impresso pelo programa? Qual é o papel de `fun2()`??

Funções Aninhadas

Podemos utilizar ou criar uma variável do escopo acima da função aninhada com o comando `nonlocal`.

```
1  def fun1(a, b=10):
2
3    def fun2():
4      nonlocal a
5      if a < 0:
6        a = -1 * a
7
8    fun2()
9    i = 0
10   while(i < b):
11     print(a**i)
12     i = i+1
13
14 fun1(2, 5)
15 print('----')
16 fun1(-2,5)
```

- Agora, o valor de `a` em `fun1()` foi alterado pela função `fun2()`.

Roteiro

Laços Encaixados

Para resolver alguns problemas, é necessário implementar um laço dentro de outro laço.

- Estes são conhecidos como laços encaixados.

```
1 i = 1
2 while(i < 5):
3   j = 1
4   while(j < 4):
5     print(i, j)
6     j += 1
7   i += 1
```

- O que será impresso por este programa?

Laços Encaixados

```
1 i = 1
2 while(i < 5):
3   j = 1
4   while(j < 4):
5     print(i, j)
6     j += 1
7   i += 1
```

- Fixado um valor para **i** no primeiro laço, começa-se o segundo laço, que varia o valor de **j** entre 1 e 3.
- No final deste segundo laço voltamos para o primeiro laço onde a variável **i** assumirá seu próximo valor.
  - Fixado este valor de **i** começa-se novamente o segundo laço.

Laços Encaixados

```
1 i = 1
2 while(i < 5):
3   j = 1
4   while(j < 4):
5     print(i, j)
6     j += 1
7   i += 1
```

- Será impresso:

```
1  1 1
2  1 2
3  1 3
4  2 1
5  2 2
6  2 3
7  ...
8  4 1
9  4 2
10 4 3
```

Exemplo 1

Escreva uma função que receba um número n e imprima n linhas na tela com o seguinte formato:

```
1 >>> fun1(5)
```

```
1 * * * * *
2 * * * * *
3 * * * * *
4 * * * * *
5 * * * * *
```

Exemplo 1

```
1  def fun1(a):
2    i = 0
3    while(i<a):
4      j=0
5      while(j<a):
6        print("*", end="")
7        j+=1
8      print()
9      i+=1
10   return None
```

Exemplo 2

Escreva uma função que receba um número n e imprima n linhas na tela com o seguinte formato:

```
>>> fun2(5)
```

```
+ * * * *
* + * * *
* * + * *
* * * + *
* * * * +
```

Exemplo 2

```
1  def fun2(a):
2    i = 0
3    while(i<a):
4      j=0
5      while(j<a):
6        if(i==j): print("+", end="")
7        else: print("*", end="")
8        j+=1
9      print()
10     i+=1
11   return None
```

Exemplo 3

Escreva uma função que receba um número *n* e imprima *n* linhas na tela com o seguinte formato:

```
>>> fun3(5)
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Exemplo 3

```
1  def fun3(a):
2    i = 0
3    while(i<a):
4      j=0
5      while(j<=i):
6        print(j+1, end=" ")
7        j+=1
8      print()
9      i+=1
10   return None
```

Exemplo 4

Escreva uma função que recebe como entradas dois números inteiros correspondentes à largura e à altura de um retângulo.
O programa deve imprimir o retângulo informado com caracteres '#' na saída.

```
>>> fun4(10, 3)
```

```
##########
##########
##########
```

Vamos assumir que `largura` e `altura` são maiores do que 2.

Exemplo 4

```
1  def main():
2    l = int(input("digite a largura: "))
3    a = int(input("digite a altura: "))
4    fun4(l, a)
5
6  def fun4(largura, altura):
7    i=0
8    j=0
9    while(i<altura):
10     while(j<largura):
11       print("#", end="")
12       j=j+1
13     print()
14     i=i+1
15     j=0
16   return None
17
18 main()
```

Exemplo 5

Refaça o exemplo anterior imprimindo os retângulos sem preenchimento, conforme exemplo abaixo:

```
>>> fun5(10, 3)
```

```
##########
#        #
#        #
##########
```

Exemplo 5

```
1  def main():
2    l = int(input("digite a largura: "))
3    a = int(input("digite a altura: "))
4    fun5(l, a)
5
6  def fun5(largura, altura):
7    i=0
8    j=0
9
10   #imprimindo a primeira linha
11   while(j<largura):
12     print("#", end="")
13     j=j+1
14
15   j=0
16   print()
17   ...
```

Exemplo 5

```
1   ...
2   #imprimindo as linhas do meio
3   while(i<altura-2):
4     print("#", end="") #primeira coluna
5     j=0
6     while(j<largura-2):
7       print(" ", end="")
8       j=j+1
9     print("#", end="") #última coluna
10    print()
11    i=i+1
12
13  #imprimindo a última linha
14  j = 0
15  while(j<largura):
16    print("#", end="")
17    j=j+1
18
19  print()
20  return None
21
22 main()
```

Fim

Dúvidas?

Leitura complementar

Leitura complementar:

1. https://panda.ime.usp.br/pensepy/static/pensepy/05-Funcoes/funcoes.html
2. https://www.w3schools.com/python/python_functions.asp

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Eduardo C. Xavier, da Universidade Estadual de Campinas.