SCC 124 - Introdução à Programação para Engenharias

Funções

python

Professor: André C. P. L. F. de Carvalho, ICMC-USP
Pos-doutorando: Isvani Frias-Blanco
Monitor: Henrique Bonini de Britto Menezes

Aula de hoje

- Introdução
- Passagem de parâmetros
- Parâmetros X argumentos
- Retorno de valores
- Polimorfismo
- Argumentos com valor default

Introdução

- Programas Python são compostos por funções
- Código reutilizáveis
- Permitem nomear um bloco de comandos
  - Que podem ser executados em qualquer parte do programa
  - Qualquer número de vezes

Função

- Trecho de código para a realização de uma operação específica e especializada
- Ao final da execução, controle volta para o trecho que chamou a função
- Associada a um nome
  - O uso desse nome em um código é uma chamada de função
- Python tem várias funções pré-construídas

Definição de funções

- Funções em Python são definidas de acordo com a seguinte sintaxe:

```
def nome (lista_de_parâmetros):
    corpo_da_função
```

- Lista de parâmetros
  - Entradas da função
    - Funcionam como variáveis locais da função
  - Pode ser vazia

Exemplo

```
def area_quad (altura, largura):
  a = altura * largura
  print ("Area = ", a)
  return a
```

```
def volume_cubo (n):
  v = n * n * n
  print ("Volume = ", v)
  return (v)
```

```
def multi (x, y):
  return x*y
```

```
>>> area_quad (2,4)
>>> volume_cubo (5)
>>> multi (3, 5)
```

No modo script, a chamada a função multi não imprime resultado

Exemplo

```
def area_quad (altura, largura):
  a = altura * largura
  print ("Area = ", a)
  return a

def volume_cubo (n):
  v = n * n * n
  print ("Volume = ", v)
  return (v)

def multi (x, y):
  return x*y
```

```
>>> area_quad (2,4)
8
>>> volume_cubo (5)
125
>>> multi (3, 5)
15
```

No modo script, a chamada a função multi não imprime resultado

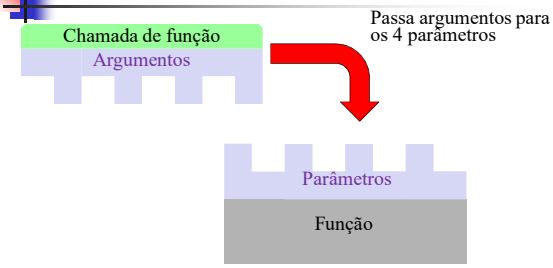
Resultado de funções

- Uma função pode retornar valores ao programa que a chamou
  - Ex.: *return expressão (opcional)*
    - Comando *return* termina execução da função
    - O valor de *expressão* é retornado como valor da função
      - Valores de qualquer tipo podem ser retornados
        - Funções em Python não têm tipo

Chamada de um função

- Pode enviar informações para a execução da função
  - Argumentos
- Argumentos = lista de expressões
  - Na chamada de uma função, lista aparece entre parênteses após o nome da função
    - Ex. *mdc (x, y) :*

Chamada de uma função



Chamada de um função

- Cada expressão na lista de argumentos é avaliada
  - Número de parâmetros deve ser maior ou igual ao número de argumentos
    - Se maior, e sem default, dá erro
- O valor de cada argumento é atribuído ao parâmetro correspondente na função
- O corpo da função é executado

Chamada de um função

- Parâmetros X argumentos
  - Parâmetros – dentro
    - Informados na declaração da função
    - Recebem valores dos argumentos
  - Argumentos – fora
    - Informados na chamada da função
    - Passam valores para os parâmetros

Passagem de dados

- Argumentos são passados por valor
  - Mesmo sendo variáveis
    - Quando função é chamada, seus parâmetros recebem valor das variáveis (arg.) passadas
    - Função chamada não pode mudar valor da variável usada em argumento da chamada
  - Parâmetros de uma função deixam de existir após a execução da função

Comando return

- Se não existir?
  - Função simula um procedimento
    - Existe em outras linguagens
  - Função possui, implicitamente, no final, o comando *Return None*
    - *None*
      - Palavra reservada
      - Pode ser usado para indicar que uma variável não tem valor

Comando return

- Se existir?
  - Se comando *return* possui uma expressão:
    - O valor da expressão é retornado para quem a chamou
    - Pode retornar mais de um valor
      - Valor de um tipo sequência
  - Se comando *return* não possui uma expressão:
    - Nenhum valor é retornado
    - retorna *None* (palavra reservada)

Módulo

- Programa Python escrito no editor do ambiente de desenv. (ex. PyCharm)
  - Módulo é um arquivo com as definições e comandos de um programa em Python
  - Código é perdido quando sai do interpretador
    - A menos que seja salvo para reutilizar depois
      - Todo o código escrito ou parte dele

Variáveis globais

- Declaradas no nível mais alto de um módulo
  - Ou em uma função precedidas da palavra global
- Válidas em todo o módulo onde são declaradas
- Atribuições a variáveis globais não são feitas dentro de uma função
  - A menos que seja usado o comando *global*
- Variáveis globais podem ser referenciadas em funções
  - Não usar no curso

Exemplos 1

```
x = 99
def fun (y):
    z = x + y
    return z

w = fun (1)
print (w)

Saida:
```

```
x = 99
def fun ():
    global x
    x = 20

fun ()
print (x)

Saida:
```

```
x = 99
def fun ():
    x = 20
    print (x)
    x = x+1

fun ()
print (x)

Saida:
```

Exemplos 1

```
x = 99
def fun (y):
    z = x + y
    return z

w = fun (1)
print (w)

Saida:
100
```

```
x = 99
def fun ():
    global x
    x = 20

fun ()
print (x)

Saida:
20
```

```
x = 99
def fun ():
    x = 20
    print (x)
    x = x+1

fun ()
print (x)

Saida:
20
99
```


Exemplos 2

```
y, z = 1,2
def fun ():
    global x
    x = z + y
fun ()
print (x)

Saida :
```

```
y = 2
z = 3
def fun ():
    global y, z
    x = y + z
    y = x + 1
fun ()
print (y)

Saida :
```


Exemplos 2

```
y, z = 1,2
def fun ():
    global x
    x = z + y
fun ()
print (x)

Saida :
3
```

```
y = 2
z = 3
def fun ():
    global y, z
    x = y + z
    y = x + 1
fun ()
print (y)

Saida :
6
```


Polimorfismo

- Tipo dos argumentos pode variar de uma chamada para outra
- Resultado da função depende dos tipos dos objetos que manipula

```
def multi (x, y):
    return x*y

>>> multi (3, 5)

>>> x = multi (3.4, 2)
>>> x

>>> multi ("qua", 3)
```


Polimorfismo

- Tipo dos argumentos pode variar de uma chamada para outra
- Resultado da função depende dos tipos dos objetos que manipula

```
def multi (x, y):
    return x*y

>>> multi (3, 5)
15
>>> x = multi (3.4, 2)
>>> x
6.8
>>> multi ("qua", 3)
"quaquaqua"
```


Valores de argumento default

- Função pode receber número variável de argumentos a cada chamada
  - Deve ser menor ou igual ao número de parâmetros usado em sua definição
    - Valor default é usado para os demais
  - Usado quando o programador não quer por o valor de todos os argumentos na chamada
    - Precisa colocar *nome-do-argumento = valor-default* na lista de parâmetros da função

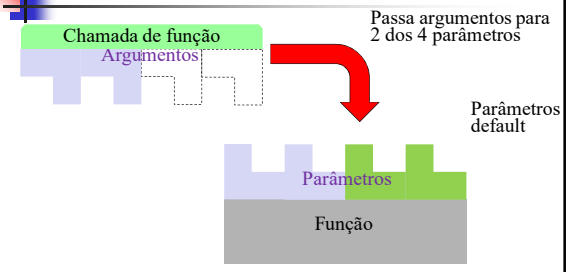
Exemplo

```
def fun (arg=5):
   print (arg)

fun ()

Saída:
5
```


Chamada de uma função



Valores de argumento default

- Só pode usar para os parâmetros no final da lista de parâmetros da função
  - Parâmetro com valor default não pode proceder parâmetro sem valor default
    - Atribuição de argumentos a parâmetros segue a ordem em que parâmetros aparecem na lista

Argumentos palavra chave

- Permite especificar nominalmente que parâmetros receberão valores
  - Usa o nome do parâmetro na lista ao invés da sua posição na lista
    - Não precisa se preocupar com a ordem dos argumentos na chamada
    - Permite atribuir valores apenas aos parâmetros desejados
      - Desde que os outros tenham valor *default*

Argumentos palavra chave

- Argumentos de funções podem ser passados em qualquer ordem

```
def calc_area (altura, largura = 3):
   area = altura * largura
   print ("Area total = ", area)
   return area

>>> x  = calc_area (4)
Area total = 12
>>> x = calc_area (largura=2, altura=7)
Area total = 14
```


Exemplo

```
def fun (a, b=5, c=10):
   print('a =', a, 'e b =', b, 'e c =', c)

fun(3, 7)
fun(25, c=24)
fun(c=50, a=100)

Saída:
a = 3 e b = 7 e c = 10
a = 25 e b = 5 e c = 24
a = 100 e b = 5 e c = 50
```

Parâmetros VarArgs

- Python permite definir função com número de parâmetros variável
  - Chamada à esta função pode ter qualquer número de argumentos
  - Colocar * antes do nome do parâmetro
    - Ex.: *def fun (*numeros)*
      - Valores ´ vão formar a tupla numeros
    - Para tipo dicionário usar **

Exemplo

```
def total(inicial=5, *numeros):
    contador = inicial
    for numero in numeros:
        contador += numero
    return contador

print(total(10, 1, 2, 3, ))
print(total())

Saída:
```


Exemplo

```
def total(inicial=5, *numeros):
    contador = inicial
    for numero in numeros:
        contador += numero
    return contador

print(total(10, 1, 2, 3, ))
print(total())

Saída:
16
5
```


Exercício

- Escrever função que calcula sequência de Finonacci

Conclusão

- Introdução
- Passagem de parâmetros
- Parâmetros X argumentos
- Retorno de valores
- Polimorfismo
- Argumentos
  - Valor *default*
  - Palavra chave
  - VarArgs

Perguntas