Python: Funções

Claudio Esperança

Definindo funções

- Em Python, sub-programas têm o nome de ***funções***
- Formato geral:

  `def` *nome* (*arg*, *arg*, ... *arg*):

  comando

  . . .

  comando
- Onde:
  - *nome* é o nome da função
  - *args* são especificações de argumentos da função
    - Uma função pode ter 0, 1 ou mais argumentos
  - *comandos* contêm as instruções a ser executadas quando a função é invocada

Resultado de funções

- Uma função tipicamente computa um ou mais valores
- Para indicar o valor a ser devolvido como o resultado da função, usa-se o comando `return`, que tem o formato
  `return` *expressão*
  - onde a *expressão* é opcional *e* designa o valor a ser retornado
- Ao encontrar o comando `return`, a função termina imediatamente e o controle do programa volta ao ponto onde a função foi chamada
- Se uma função chega a seu fim sem nenhum valor de retorno ter sido especificado, o valor de retorno é `None`

Exemplo

```
>>> def f():
      return

>>> print f()
None
>>> def f():
      return "Oi"

>>> print f()
Oi
>>> def f(nome):
      return "Oi, "+nome+"!"

>>> print f("Joao”)
Oi, Joao!
```

Variáveis locais e globais

- Variáveis definidas em funções são *locais*, isto é, só podem ser usadas nas funções em que foram definidas
- Variáveis definidas fora de funções são conhecidas como variáveis globais
  - É possível no código de uma função ler o conteúdo de uma variável global
  - Para alterar uma variável global, ela precisa ser declarada no corpo da função usando o comando `global`

Exemplo

```
>>> def f():
       print a

>>> a = 1
>>> f()
1
>>> def f():
       a = 5

>>> f()
>>> print a
1
>>> def f():
       global a
       a = 5

>>> f()
>>> print a
5
```

Argumentos de funções

- Argumentos (ou parâmetros) são como variáveis que recebem seus valores iniciais do chamador
- Essas variáveis, assim como outras definidas dentro da função são ditas *locais*, isto é, só existem no lugar onde foram definidas
  - Ao retornar ao ponto de chamada, as variáveis locais são descartadas
- Se uma função define *n* argumentos, a sua chamada deve incluir valores para todos eles
  - Exceção: argumentos com valores default

Exemplo

```
>>> def f(x):
      return x*x

>>> print f(10)
100
>>> print x
....
NameError: name 'x' is not defined
>>> print f()
....
TypeError: f() takes exactly 1 argument (0
  given)
```

Argumentos *default*

- É possível dar valores *default* a argumentos
  - Se o chamador não especificar valores para esses argumentos, os defaults são usados
- Formato:
  `def` *nome* (*arg1*=default1, ..., *argN*=*defaultN*)
- Se apenas alguns argumentos têm default, esses devem ser os *últimos*
  - Se não fosse assim, haveria ambigüidade na passagem de argumentos

Exemplo

```
>>> def f(nome,saudacao="Oi",pontuacao="!!"):
      return saudacao+","+nome+pontuacao

>>> print f("Joao”)
Oi,Joao!!
>>> print f("Joao","Parabens”)
Parabens,Joao!!
>>> print f("Joao","Ah","...”)
Ah,Joao...
```

Passando argumentos com nomes

- É possível passar os argumentos sem empregar a ordem de definição desde que se nomeie cada valor passado com o nome do argumento correspondente
- Ex.:

```
>>> def f(nome,saudacao="Oi",pontuacao="!!"):
      return saudacao+","+nome+pontuacao

>>> print f(saudacao="Valeu",nome="Joao”)
Valeu,Joao!!
```

Documentando Funções

- Ao invés de usar comentários para descrever o que uma função, é mais vantajoso usar *docstrings*
  - Uma constante string escrita logo após o cabeçalho da função (comando def)
  - Permite o acesso à documentação a partir do interpretador, usando a notação *função* . `__doc__`

```
>>> def fat(n):
        "Retorna o fatorial de n.”
        for i in range(n-1,1,-1): n*=i
        return n
...
>>> fat(4)
24
>>> print fat.__doc__
Retorna o fatorial de n.
```

Lista de parâmetros variável

- Se o último argumento de uma definição de função começa com *, todos os valores passados, a partir daquele, são postos numa tupla
- Ex.:

```
>>> def imprime(nome,*atributos):
        print (nome,atributos)
...
>>> imprime ('a',1,2,'b’)
a (1, 2, 'b’)
>>> def media(*valores):
        total=0.0
        for x in valores: total+=x
        return total/len(valores)
...
>>> media (1,2,3,4)
2.5
```