Programação I

Licenciatura em Engenharia Informática

2015–2016

Vitor Beires Nogueira

Escola de Ciências e Tecnologia
Universidade de Évora

Chamadas de funções

Definição

Uma função é uma *sequência* de instruções com *nome* que realizam uma computação.

Exemplos de funções

- Já fizemos chamadas a funções. Aonde?
- Exemplo

```
>>> type(32)
<class 'int'>
```

  O nome da função é `type` e o argumento é 32
- É usual dizer que uma função *recebe* um argumento e *devolve* um resultado.

Funções para a conversão de tipos I

int

```
>>> int('32')
32
>>> int('Hello')
ValueError: invalid literal for int(): Hello
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

float

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
```

Funções para a conversão de tipos II

str

```
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

Definição

Um módulo é um "ficheiro"que contém um conjunto de funções relacionadas

Módulo datetime

```
>>> import datetime
>>> print(datetime)
<module 'datetime' from '/usr/lib/python3.4/datetime.py'>
```

Dot notation

```
>>> import math
>>> degrees = 45
>>> radians = degrees / 360 * 2 * math.pi
>>> math.sin(radians)
0.707106781187
```

Composição

Exemplos

```
x = math.sin(degrees / 360 * 2 * math.pi)

x = math.exp(math.log(x+1))
```

Não esquecer que o lado esquerdo de uma instrução de atribuição tem de ser o nome de uma variável.

Acrescentando funções I

Definição

A definição de uma função especifica o nome da nova função assim como a sequência de instruções que deve ser executadas quando a função é invocada

Exemplo: `print_lyrics`

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print("I sleep all night and I work all day.")
```

- Header e body?
- Identação e aspas?

```
 >>> print(print_lyrics)
<function print_lyrics at 0xb7e99e9c>
>>> type(print_lyrics)
<type 'function'>
```

Acrescentando funções II

Exemplo: `repeat_lyrics`

```
def repeat_lyrics():
    print_lyrics()
    print_lyrics()
```

O que acontece se fizermos

```
>>> repeat_lyrics()
```

Definições e usos

- As definições de funções são semelhantes às que já conhecemos, somente o efeito é que é diferente: criamos funções!
- As instruções dentro de uma função só são executadas quando a função é invocada.
- Uma função tem de ser definida antes de ser invocada.

Fluxo de execução

- A execução começa na primeira instrução do programa
- As instruções são executadas uma de cada vez, por ordem, de cima para baixo.
- A invocação de uma função é como que um *desvio* no fluxo de execução. Em vez de seguir para a próxima instrução, o fluxo
  1. salta para a "body" da função
  2. executa as instruções existentes no "body"
  3. regressa, retomando o ponto onde tinha ficado
- Não esquecer que no "body" de uma função podemos invocar outra ...
- Quando lemos um programa, em vez de ler de "cima para baixo" podemos (e devemos) fazer estes "saltos"

Parâmetros e Argumentos I

Definição (Argumentos)

Valor fornecidos a uma função aquando da sua invocação

Definição (Parâmetros)

O nome utilizado dentro de uma função para se referir ao valor passado como argumentos

Função `print_twice`

```
def print_twice(bruce):
    print(bruce)
    print(bruce)
```

Parâmetros e Argumentos II

Exemplos

```
>>> print_twice('Spam')
Spam
Spam
>>> print_twice(17)
...
>>> print_twice('Spam '*4)
...
>>> michael = 'Eric, the half a bee.'
>>> print_twice(michael)
...
```

Variáveis e parâmetros são locais às funções

Função `cat_twice`

```
def cat_twice(part1, part2):
    cat = part1 + part2
    print_twice(cat)
```

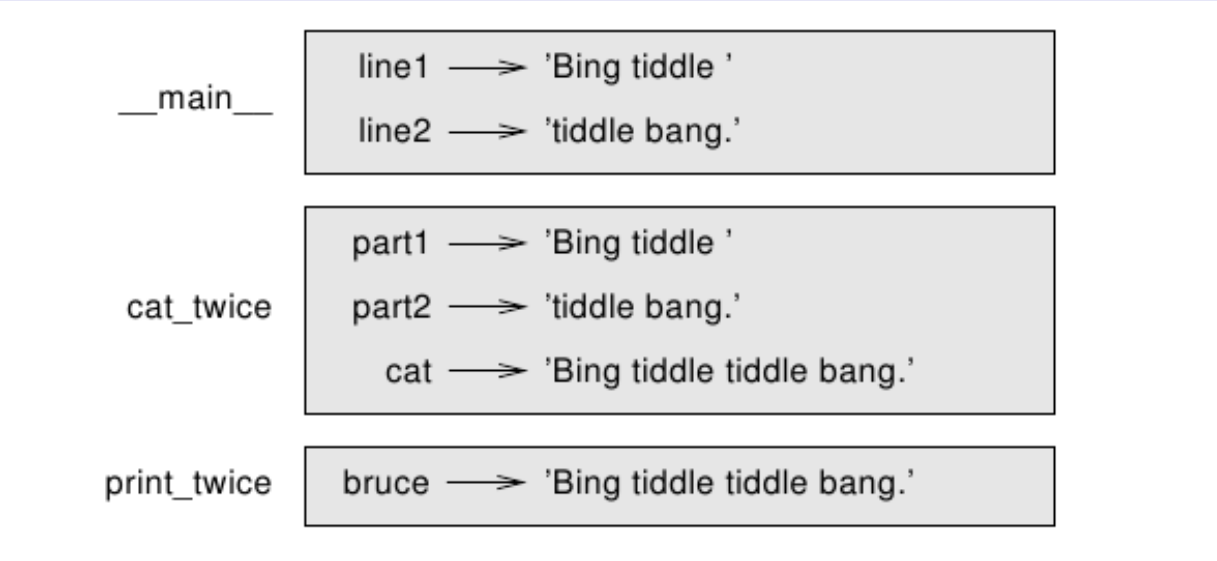
Exemplos

```
>>> line1 = 'Bing tiddle '
>>> line2 = 'tiddle bang.'
>>> cat_twice(line1, line2)
Bing tiddle tiddle bang.
Bing tiddle tiddle bang.

>>> print(cat)
NameError: name 'cat' is not defined
```

Stack Diagrams I

Stack diagrams in frames



O que acontece se tentarmos aceder à variável `cat` dentro da função `print_twice`?

Stack Diagrams II

Traceback

```
Traceback (innermost last):
File "test.py", line 13, in __main__
cat_twice(line1, line2)
File "test.py", line 5, in cat_twice
print_twice(cat)
File "test.py", line 9, in print_twice
print cat
NameError: name 'cat' is not defined
```

Funções void e não void I

Definição (Função void)

Uma função que não retorna nada

Definição (Função não void)

Uma função não é *void* se devolver algo.

Exemplos

```
x = math.cos(radianos)
golden = (math.sqrt(5) + 1) / 2
```

- Qual é a diferença de invocarmos `math.cos(radianos)` num script ou em modo interactivo?

Funções void e não void II

Mais exemplos

```
>>> result = print_twice('Bing')
Bing
Bing
>>> print(result)
None
>>> print(type(None))
<class 'NoneType'>
```

Porquê utilizar funções

- Ao darmos nome a um grupo de instruções, tornamos o programa mais legível e mais fácil de fazer debugging.
- As funções, ao eliminar código repetitivo, podem tornar um programa mais pequeno.
- Dividir um programa extenso em funções permite fazer o debugging de cada uma das partes e somente depois juntá-las.
- Funções "bem concebidas" são por vezes muito úteis para muitos programas.