

Fundamentos da Programação LEIC/LETI

Recursão

Recursividade. Recursão de cauda

Aula 21

Alberto Abad, Tagus Park, IST, 2021-22

Funções revisitadas

Programação Funcional

- *Programação imperativa*: programa como conjunto de instruções em que a instrução de atribuição tem um papel preponderante.
- A **Programação funcional** é um paradigma de programação exclusivamente baseado na utilização de funções:
 - Funções calculam ou avaliam outras funções e retornam um valor/resultado, evitando alterações de estado e entidades mutáveis.
 - Não existe o conceito de atribuição e não existem ciclos.
 - O conceito de iteração é conseguido através de recursividade.

Funções revisitadas

Elementos da Programação Funcional

- Na informática, diz-se que uma linguagem de programação tem funções de primeira classe (*first-class functions*) se a linguagem suporta utilizar funções como argumentos para outras funções, retornar funções como valor de outras funções, atribuir funções a variáveis, ou armazenar funções em estruturas de dados.
- O Python, tem funções de primeira classe o que nos fornece alguns dois elementos fundamentais dada programação funcional:
 - Funções internas (**ontem**)
 - Recursão (**hoje e o resto da semana**)
 - Funções de ordem superior: (**a próxima semana**)
 - Funções como parâmetros
 - Funções como valor

Funções revisitadas

Funções recursivas

- Uma solução recursiva para um problema depende da combinação de soluções para instâncias mais pequenas desse mesmo problema.
- Uma dada entidade é recursiva se ela for definida em termos de si própria.
- Python, tal como a maioria das linguagens de programação, suporta explicitamente soluções recursivas permitindo que as funções possam invocar-se a si mesmas.
- Em *programação funcional* e em linguagens puramente funcionais, estamos limitados ao uso de funções recursivas, não sendo possível o uso de ciclos iterativos.

Funções revisitadas

Funções recursivas. Exemplos de entidades recursivas

- BNFs:

$\langle \text{nomes} \rangle ::= \langle \text{nome} \rangle \mid$
 $\langle \text{nome} \rangle, \langle \text{nomes} \rangle$

- Na matemática, por exemplo a Série de Fibonacci:

$$fib(n) = \begin{cases} 0 & \text{se } n = 1, \\ 1 & \text{se } n = 2, \\ fib(n-1) + fib(n-2) & \text{se } n > 2 \end{cases}$$

- O que têm em comum estas definições...
 - Um caso base ou **caso terminal**, que corresponde à versão mais simples do problema;
 - Um passo recursivo ou **caso geral**, que corresponde à definição recursiva de uma solução para o problema em termos de soluções para sub-problemas deste mas mais simples.

Funções revisitadas

Funções recursivas. Exemplo 1, *potencia*

$$pow(x, n) = \begin{cases} 1 & \text{se } n = 0, \\ x * pow(x, n-1) & \text{se } n > 0 \end{cases}$$

```
In [24]: # solucao iterativa
def potencia_it(x, k):
    pot = 1
    while k > 0:
        pot = pot * x
        k -= 1
    return pot

def potencia_rec(x, k):
    # Caso terminal
    if k == 0:
        return 1
    # Caso geral
    elif k > 0:
        return x*potencia_rec(x, k-1)
    else:
        return potencia_rec(x, k+1)/x

potencia_rec(2,50)
```

Out[24]: 1125899906842624

Funções revisitadas

Funções recursivas. Exemplo 1, *potencia*. Python Tutor

<http://pythontutor.com/visualize.html> (<http://pythontutor.com/visualize.html>)

The image shows a screenshot of the Python Tutor website, which visualizes the execution of Python code. The main window displays the code for a recursive function named `potencia`. The code is as follows:

```
1 def potencia(x, k):
2     if k < 0:
3         raise ValueError('potencia: expoente k negativo')
4     elif type(k) != int:
5         raise ValueError('potencia: expoente k nao inteiro')
6     elif type(x) != int and type(x) != float:
7         raise ValueError('potencia: base x nao e\' um numero')
8
9     if k == 0:
10        return 1
11    else:
12        return x * potencia(x, k-1)
13
14
15 potencia(2, 4)
```

The execution is shown at Step 31 of 37. The current line of execution is line 9, which is the start of the `if k == 0:` block. The next line to execute is line 10, which is `return 1`.

On the right side, the 'Frames' and 'Objects' panels are visible. The 'Frames' panel shows four frames of the `potencia` function, each with its own local variables `x` and `k`. The 'Objects' panel shows the function object `potencia(x, k)` and the local variables `x` and `k` for each frame.

At the bottom, there are navigation buttons: `<< First`, `< Back`, `Step 31 of 37`, `Forward >`, and `Last >>`. There is also a legend indicating that a green arrow points to the line that has just executed, and a red arrow points to the next line to execute.

Funções revisitadas

Mais exemplos de funções recursivas

- Soma de dígitos
- Factorial
- Progressão aritmética
- Máximo divisor comum
- Alisa

Funções revisitadas

Funções recursivas. Exemplo 2, *soma_digitos* de um inteiro

```
In [25]: def soma_digitos(num):  
        soma = 0  
        while num!=0:  
            soma += num % 10  
            num = num // 10  
        return soma  
  
        def soma_digitos_rec(n):  
            # return n if n < 10 else n%10 + soma_digitos_rec(n//10)  
            if n < 10:  
                return n  
            else:  
                return n%10 + soma_digitos_rec(n//10)  
  
        soma_digitos_rec(56071)
```

Out[25]: 19

Funções revisitadas

Funções recursivas. Exemplo 2, *soma_digitos* de um string

Out[26]: 18

```
Out[27]: 933262154439441526816992388562667004907159682643816214685929638952  
175999932299156089414639761565182862536979208272237582511852109168  
64000000000000000000000000
```

Funções revisitadas

Funções recursivas. Exemplo 4, soma progressão aritmética

```
In [28]: def soma(n):
          res = 0
          for i in range(1,n+1):
              res += i
          return res

          def soma_rec(n):
              if n == 0:
                  return 0
              else:
                  return n+soma_rec(n-1)

          soma_rec(10)
```

Out[28]: 55

Funções

Funções recursivas. Exemplo 5, Máximo divisor comum

1. O máximo divisor comum entre um número e zero é o próprio número: $\text{mdc}(m, 0) = m$
2. Quando dividimos um número m por um menor n , o máximo divisor comum entre o resto da divisão e o divisor é o mesmo que o máximo divisor comum entre o dividendo e o divisor: $\text{mdc}(m, n) = \text{mdc}(n, m \% n)$

```
def mdc(m,n):
    while n != 0:
        m, n = n, m % n
    return m
```

```
In [29]: def mdc(m,n):  
    while n != 0:  
        m, n = n, m % n  
    return m  
  
def mdc_rec(m, n):  
    if n == 0:  
        return m  
    else:  
        return mdc_rec(n, m%n)  
  
mdc_rec(12, 8)
```

Out[29]: 4

Funções

Funções recursivas. Exemplo 6, Função *alisa*

```
In [30]: def alisa(t):  
    i = 0  
    while i < len(t):  
        if isinstance(t[i], tuple):  
            t = t[:i] + t[i] + t[i+1:]  
        else:  
            i = i + 1  
  
    return t  
  
def alisa_rec(t):  
    if t == ():  
        return ()  
    else:  
        if isinstance(t[0], tuple):  
            return alisa_rec(t[0]) + alisa_rec(t[1:])  
        else:  
            return (t[0],) + alisa_rec(t[1:])  
  
a = (2, 4, (8, (9, (7, ), 3, 4), 7), 6, (5, (7, (8, ))))  
alisa_rec(a)
```

Out[30]: (2, 4, 8, 9, 7, 3, 4, 7, 6, 5, 7, 8)

Funções

Funções recursivas. Exemplo 7, Longest common subsequence (LCS)

https://en.wikipedia.org/wiki/Longest_common_subsequence_problem

(https://en.wikipedia.org/wiki/Longest_common_subsequence_problem)

Sejam duas sequencias s e t tal que $|s| = n$ e $|t| = m$, a LCS é:

$$lcs(s, t) = \begin{cases} \emptyset & \text{se } s \text{ ou } t \text{ vazio,} \\ lcs(s_{1..n-1}, t_{1..m-1}) \cup s_n & \text{se } s_n = t_m \\ longest(lcs(s, t_{1..m-1}), lcs(s_{1..n-1}, t)) & \text{se } s_n \neq t_m \end{cases}$$

Funções

Funções recursivas. Exemplo 7, Longest common subsequence (LCS)

```
In [31]: def lcs(a, b):
          def longest(a, b):
              if len(a) > len(b):
                  return a
              else:
                  return b

          if len(a) == 0 or len(b) == 0:
              return type(a)()
          elif a[-1] == b[-1]:
              return lcs(a[:-1], b[:-1]) + a[-1:]
          else:
              return longest(lcs(a[:-1], b), lcs(a, b[:-1]))

lcs('mara', 'matias')
```

Out[31]: 'maa'

Tarefas próxima aula

- Estudar matéria e completar exemplos
- Ler seções 7.3 do livro (recursão de cauda)
- **WARNING:** O deadline para entrega do projeto 1 é esta sexta-feira dia 5 de Novembro até às 17h00!!



In []: