

Fundamentos da Programação

Recursão

Aula 21

José Monteiro

(slides adaptados do Prof. Alberto Abad)

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.

Elementos da Programação Funcional

- Em 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 dos elementos fundamentais da programação funcional:
 - Funções internas (**ontem**)
 - Recursão (**resto desta semana**)
 - Funções de ordem superior: (**próxima semana**)
 - Funções como parâmetros
 - Funções como valor

Funções Recursivas

- Uma solução recursiva para um problema é obtida pela combinação de soluções de instâncias mais pequenas desse mesmo problema.
- Uma dada entidade é recursiva se ela for definida em termos de si própria.
- O 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.

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:

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

- 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.

Exemplo de Função Recursiva

Exemplo 1: *potencia*

$$potencia(x, n) = \begin{cases} 1 & \text{se } n = 0, \\ x \times potencia(x, n-1) & \text{se } n > 0 \end{cases} \quad (2)$$

In [16]:

```
# solução 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
    else:
        if k > 0:
            return x * potencia_rec(x, k-1)
        else:
            return 1/(x * potencia_rec(x, -k-1))

potencia_rec(2,-3)
```

Out[16]: 0.125

Exemplo de Função Recursiva

Exemplo 1: *potencia*, Python tutor

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

The image shows a screenshot of the Python Tutor website, which visualizes the execution of Python code. The code being executed is a recursive function named `potencia` that calculates the power of a number `x` to the power of `k`. The function has two base cases: `k == 0` (returning 1) and `k < 0` (returning $1/x^{|k|}$). The current step in the execution is line 15, where the function is called with `potencia(2, 4)`. The left pane shows the code with line numbers 1 to 15. The right pane shows the state of the program, including the global frame and several local frames for the recursive calls. The top frame is the 'Global frame' with a variable `potencia` pointing to the function object. Below it are four frames for the recursive calls, each with variables `x` and `k`. The frames show the progression of the recursion: `(x=2, k=4)`, `(x=2, k=3)`, `(x=2, k=2)`, and `(x=2, k=1)`. The bottom frame, which is highlighted, shows `(x=2, k=0)`, representing the base case. The bottom of the interface includes navigation buttons and a progress indicator showing 'Step 31 of 37'.

Mais Exemplos de Funções Recursivas

- Soma de dígitos
- Fatorial
- Progressão aritmética
- Máximo divisor comum
- Alisa
- Maior Subsequência Comum

Exemplo de Função Recursiva

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

Exemplo de Função Recursiva

Exemplo 2, *soma_digitos* de uma *string*

In [4]:

```
def soma_digitos(num):
    soma = 0
    for c in num:
        soma += int(c)
    return soma

def soma_digitos_rec2(num):
    if num == '':
        return 0
    else:
        return int(num[0]) + soma_digitos_rec2(num[1:])

soma_digitos_rec2('567')
```

Out[4]: 18

Exemplo de Função Recursiva

Exemplo 3, *fatorial*

O fatorial $n! = 1 \times 2 \times \dots \times n$ pode também ser definido de forma recursiva:

$$n! = \begin{cases} 1 & \text{se } n = 0, \\ n \times (n-1)! & \text{se } n > 0 \end{cases} \quad (3)$$

In [3]:

```
def fatorial(n):
    res = 1
    for i in range(1, n+1):
        res *= i
    return res

def fatorial_rec(n):
    if n == 0:
        return 1
    else:
        return n * fatorial_rec(n-1)

# fatorial(100000)
fatorial_rec(100)
```

Out[3]: 933262154439441526816992388562667004907159682643816214685929638952175999932
299156089414639761565182862536979208272237582511852109168640000000000000000
00000000

Exemplo de Função Recursiva

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

Exemplo de Função Recursiva

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 [15]:

```
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(987654, 12345678)
```

Out[15]: 6

Exemplo de Função Recursiva

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)

Exemplo de Função Recursiva

Exemplo 7, Maior subsequência comum, *Longest common subsequence* (LCS)

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

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

$$lcs(s, t) = \begin{cases} \emptyset & \text{se } s \text{ ou } t \text{ vazio,} \\ lcs(s_{1..n-1}, t_{1..m-1}) \cup s_n & \text{se } s_n = t_m \\ longest(lcs(s, t_{1..m-1}), lcs(s_{1..n-1}, t)) & \text{se } s_n \neq t_m \end{cases} \quad (4)$$

Exemplo de Função Recursiva

Exemplo 7, Maior subsequência comum, *Longest common subsequence* (LCS)

In [24]:

```
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[24]: 'maa'

Tarefas Próximas Aulas

- Estudar matéria e completar exemplos
- A **ficha 5** da próxima semana é sobre **recursão**
- **ATENÇÃO:** O deadline para entrega do projeto é próxima sexta-feira dia 5 de Novembro até às 17h00!!

