Nesta aula. . .

Conteúdo

1 Valores boleanos e condicionais

Operadores de comparação

==	igual
!=	diferente
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual

```
>>> 3 == 1+2
True
>>> 2.5 < 2
False
>>> 'Python' == 'python'
False
>>> 'a' < 'b'
True
```

Operadores lógicos

P `and` Q	conjunção
P `or` Q	disjunção
`not` P	negação

```
>>> import math
>>> math.pi>3
True
>>> math.pi>3 and math.pi<4
True
>>> math.pi>3.5
False
>>> not (math.pi>3.5)
True
```

Valores lógicos

`True` ou qualquer valor diferente de zero

`False` ou zero

```
>>> True and False
False
>>> True and 0
0
>>> 1 and True
True
>>> not 100
False
```

Execução condicional

```
if condição:
  instruções 1
else:
  instruções 2
```

- a condição é uma expressão boleana
- se a condição for `True` é executado bloco após `if`
- se a condição for `False` é executado bloco após `else`
- cláusula e bloco `else` podem ser omitidos

Condições embricadas

```
if x == y:
  print x, "e", y, "são iguais"
else:
  if x < y:
    print x, "é menor que", y
  else:
    print x, "é maior que", y
```

- a indentação indica a estrutura das condições
- mais do que dois níveis: difícil de ler

Condições embricadas (2)

```
if x == y:
  print x, "e", y, "são iguais"
elif x < y:
  print x, "é menor que", y
else:
  print x, "é maior que", y
```

- `elif` substitui o `else`...`if`
- um nível de indentação: leitura mais fácil
- caso geral:
  1. um `if`
  2. um ou mais `elif`
  3. um `else`

Exemplo: equação do 2º grau

Escrever uma função para calcular as raizes duma equação do 2º grau

$$ax^2 + bx + c = 0$$

a partir dos parâmetros a, b, c.

Ideia: empregar a *fórmula resolvente*

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Análise da fórmula resolvente

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Designando por $\Delta = b^2 - 4ac$, temos três casos notáveis:

se $\Delta > 0$:	tem duas raizes reais
se $\Delta = 0$:	tem uma raiz dupla
se $\Delta < 0$:	não tem raizes reais

Função

```
from math import *      # para usar sqrt

def resolve2ograu(a,b,c):
    "Resolve a equação ax**2+bx+c=0."
    delta = b**2 - 4*a*c
    if delta>0:    # duas raizes
       x1 = (-b+sqrt(delta))/(2*a)
       x2 = (-b-sqrt(delta))/(2*a)
       print 'Duas raizes:', x1, x2
    elif delta==0: # raiz dupla
       x1 = -b/(2*a)
       print 'Raiz dupla:', x1
    else:          # não tem raizes
       print 'Não tem raizes reais'
```

2 Recursão e iteração

Recursão e iteração

- resolver um problema à custa da solução de um caso menor
- repetir tarefas com dados diferentes

Factorial

O factorial de um número natural n é

$$\begin{aligned} 0! &= 1 \\ n! &= n \times (n-1)! \end{aligned}$$

Esta definição permite *calcular* o factorial para qualquer $n \geq 0 \ldots$

Exemplo

$4!$	$= ?$	$3! = ?$	recursivamente…
	$= 4 \times (3 \times 2!)$	$2! = ?$	
	$= 4 \times (3 \times (2 \times 1!))$	$1! = ?$	
	$= 4 \times (3 \times (2 \times (1 \times 0!)))$	$0! = ?$	
	$= 4 \times (3 \times (2 \times (1 \times 1)))$	$0! = 1$	caso base
	$= 4 \times (3 \times (2 \times 1))$	$1! = 1$	
	$= 4 \times (3 \times 2)$	$2! = 2$	
	$= 4 \times 6$	$3! = 6$	
	$= 24$		

Factorial recursivo

```
def factorial(n):
    "Calcula factorial de n."
    if n==0:
       return 1              # caso base
    else
       r = factorial(n-1)    # caso recursivo
       return n*r
```

Definições recursivas

caso recursivo: define a solução do problema à custa de soluções de casos menores

caso base: definido directamente (e.g. factorial de zero)

Deve sempre haver pelo menos um caso base (senão a recursão não termina!)

Iteração

```
while condição:
     instruções do ciclo
resto do programa
```

- se a condição for verdadeira, executa o bloco e repete;
- se a condição for falsa, continua o resto do programa;
- a condição é re-avaliada após cada iteração.

Exemplo do livro

```
def countdown(n):
    while n>0:
       print n
       n = n - 1
    print 'Blastoff!'
```

Exemplo do livro

```
>>> countdown(10)
10
9
8
7
6
5
4
3
2
1
Blastoff!
```

Factorial iterativo

$$n! = 1 \times 2 \times \cdots \times (n-1) \times n$$

Idea:

- repetir para $i = 1, 2, \ldots n$
- inicialmente: $i = 1$ e $r = 1$
- invariante: $r = 1 \times 2 \times \cdots \times i$
- no fim: $i = n$ e $r = n!$

Factorial iterativo

```
def factorial(n):
    "Calcula o factorial de n."
    # i vai percorrer os valores de 1 até n-1
    # r vai acumular o producto 1*2*...*i
    i = 1
    r = 1
    while i<n:
      # neste ponto: r==1*2*...*i
      i = i+1
      r = r*i
    # fim de ciclo
    # neste ponto: r==1*2*...*n
    return r
```

Definições iterativas

- pensar qual deve ser o *invariante* do ciclo
- antes do ciclo: *inicializar* as variáveis
- corpo de ciclo: *actualizar* as variáveis
- condição: teste de paragem

Cuidado com a *ordem* nas actualizações de variáveis…