Iteração

Programação I

2016.2017

Teresa Gonçalves tcg@uevora.pt

Departamento de Informática, ECT-UÉ

Reminder...

Como programar?



Como aprender?

Estudar, estudar, estudar...

Praticar, praticar, praticar...

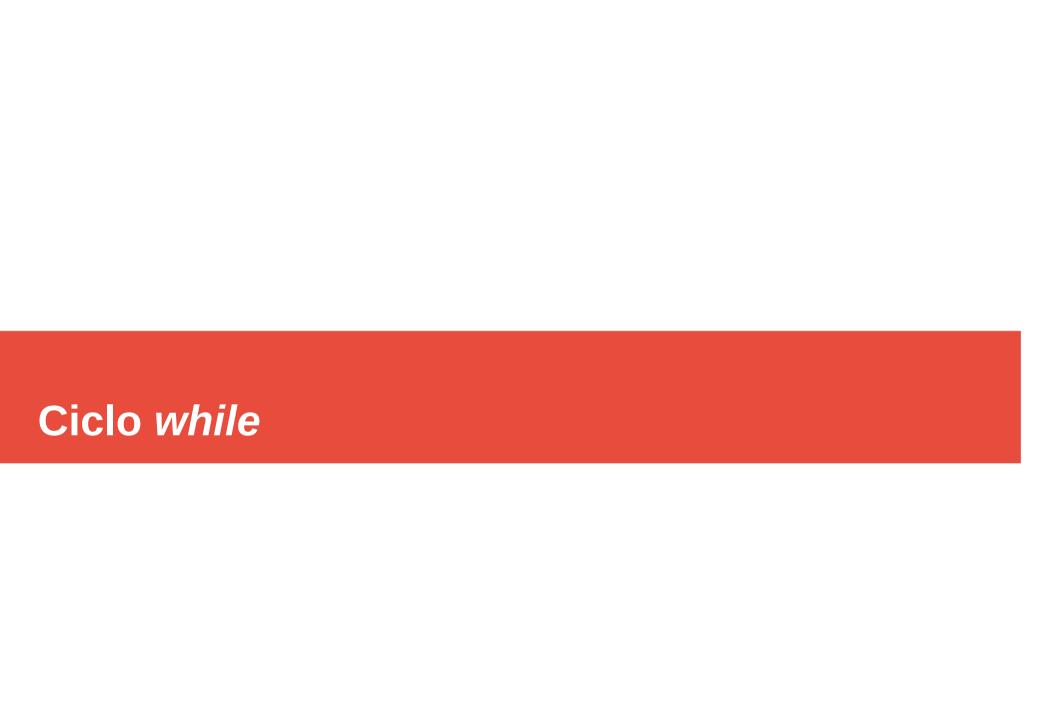
Cometer erros, cometer erros, cometer erros...

Aprender com os erros, aprender com os erros, aprender com os erros ...



Sumário

Ciclo while
Tipos sequenciais
Ciclo for
Algoritmo



Problema - countdown

Mostrar uma contagem decrescente: de 5 a 0

Solução 1

```
print(5)
```

print(4)

print(3)

print(2)

print(1)

print(0)



Problema - countdown

E se a contagem começar em 20?

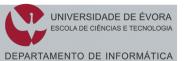
E se a contagem começar em N?

O problema é idêntico!

Varia o **número de vezes** que a função print é invocada e o **valor** que é mostrado

Iteração

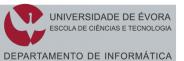
Capacidade de executar um bloco de instruções repetidamente!



Instrução while

Fluxo de execução

- 1. Avaliar a condição, obtendo True ou False
- 2.Se False, sair da instrução while e continuar com próxima instrução
- 3.Se True, executar o corpo do while e voltar ao passo 1



Problema countdown (2)

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

Condição while

Para o ciclo terminar...

... o valor das variáveis da condição devem ser alteradas no corpo do while!

Nem sempre é fácil verificar a convergência

```
def sequence(n):
    while n>1:
        print(n)
        if n%2 == 0:
            n = n//2
        else:
            n = n*3+1
```



Instrução break

Permite terminar o ciclo antecipadamente

```
while TRUE:
    line = input('> ')
    if line == 'done':
        break
    print(line)
print('Done!')
```

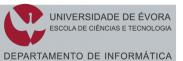


Instrução continue

Permite passar de imediato para nova iteração (fazendo o teste)

Não executa as instruções restantes do bloco

```
while TRUE:
    line = input('> ')
    if line == 'jump':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')
```



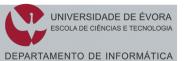
Saída do ciclo

Verificar a condição de paragem

É comum existir uma iteração a mais ou a menos

Verificar efeitos da interrupção antecipada

As instruções break e continue podem produzir efeitos não desejados no comportamento do ciclo



```
n = 0
n = 0
while TRUE:
                                while TRUE:
                                     line = input('> ')
    line = input('> ')
    if line == 'jump':
                                     n = n+1
        continue
                                     if line == 'jump':
    if line == 'done':
                                         continue
                                     if line == 'done':
        break
    print(line)
                                         break
    n = n+1
                                     print(line)
print('Done!')
                                print('Done!')
```

Exemplo – cálculo da raíz quadrada

Método de Newton

Começa com uma estimativa raiz quadrada, x

Calcula uma nova estimativa, y

Termina quando a diferença entre 2 estimativas consecutivas é desprezível

Estimativa

$$y = \frac{x + a/x}{2}$$

Exemplo – cálculo da raíz quadrada

Parâmetros

Número: a

Estimativa inicial: x

Valor de paragem: epsilon

$$y = \frac{x + a/x}{2}$$

```
def raiz2(a, x, epsilon)
  while TRUE:
    y = (x+a/x)/2
    print(y)
    if abs(y-x)<epsilon:
        break
    x = y</pre>
```

Tipos sequenciais

String

List

Range

Sequência

Coleção ordenada de valores

String

```
Sequência de caracteres
>>> fruit = 'banana'
```

Lista

```
Sequência de elementos (ou items)
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
```



Lista

Criar lista

A forma mais simples é colocar os elementos dentro de []

Os elementos não precisam ser todos do mesmo tipo

>>> ['spam', 2.0, 5]

Lista encaixada (nested)

Uma lista dentro de outra lista

>>> nested=['spam', 2.0, 5, [10,20]]

Operador []

Permite aceder aos elementos de uma sequência através de um índice

O índice do primeiro elemento é 0 (zero)

Índice

Pode ser uma expressão que contém variáveis e operadores

... mas tem de ser um inteiro!

```
>>> letter=fruit[1]
>>> letter
'a'
>>> nested[3]
[10,20]
```



Segmento

Operador [m:n]

Devolve os elementos da sequência da posição m até à posição n

Inclui m mas exclui n

Omissão

```
1º índice – segmento começa no início da sequência
```

2º índice – segmento termina no fim da sequência

```
>>> s = 'Monty Python'
>>> s[1:5]
'onthy'
>>> s[6:]
'Python'
```



Tamanho da sequência

Função len()

```
>>> len(fruit)
6
>>> len(nested)
4
```

Exercício

Qual o resultado de

```
fruit[:]
nested[2:2]
Como obter o último carácter da string?
```



Sequência vazia

Não contém nenhum elemento Tem tamanho 0 (zero)

```
>>> stringVazia =''
>>> len(stringVazia)
0
>>> listaVazia=[]
>>> len(listaVazia)
0
```

Índices negativos

Contam do fim para o início

```
>>> lenght = len(fruit)
>>> last=[lenght-1]
'a'
>>> fruit[-1]
'a'
>>> nested[-2]
5
```

Strings são imutáveis

Não é possível alterar uma string existente

```
>>> greeting='hello world!'
>>> greeting[0] = 'j'
TypeError: 'str' object does not support item assignment
```

Como fazer?

```
Criar uma nova string, variante da original
```

```
>>> greeting='hello world!'
>>> new_greeting = 'j' + greeting[1:]
>>> new_greeting
'jello world!'
```



Listas são mutáveis

```
>>> numbers=[42,153]
>>> numbers[1]= 'a'
>>> numbers
[42, 'a']
```

Pesquisa

O que faz a função?

```
def find(sequence, element):
    index=0
    while index < len(sequence):
        if sequence[index] == element:
            return index
        index = index+1
    return -1</pre>
```

Operadores

Concatenação

```
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> a+b
[1,2,3,4,5,6]
```

Repetição

```
>>> [0]*4
>>>[0,0,0,0]
>>> [1,2,['a']]*3
[1,2,['a'],1,2,['a'],1,2,['a']]
```

Operador in

Operador binário booleano

String

Operação entre 2 strings

Retorna True se a primeira string é substring da segunda

Lista

Operação entre um elemento e uma lista

Retorna True se o elemento existe na lista

Operadores relacionais

Operadores

Ordem lexicográfica

```
>>> 'banana' < 'laranja'
True
>>> 'banana' < 'Laranja'
>>> ord('b')
98
>>> [1,1,3,1] < [1,3,1,1]
True
>>> [1,3,1,1] < [1,1,3]
False</pre>
```

Ciclo for

Instrução for

Iteração sobre sequências

Para cada elemento da sequência, executa um bloco de instruções

Em cada iteração <variavel> assume um dos elementos da <sequencia>, por ordem

Exemplos

Mostrar as letras de uma string

```
>>> fruit='banana'
>>> for letter in fruit:
    print(letter)
```

Imprimir os elementos de uma lista

```
>>> cheeses=['Cheddar', 'Edam', 'Gouda']
>>> for cheese in cheeses:
    print(cheese)
```

Para modificar os elementos é necessário um índice!



Função range()

Gera uma sequência de inteiros, normalmente utilizada para iterar sobre ciclos for

range(stop)

stop: número de inteiros a gerar, começando em zero

range(3) == [0, 1, 2]

range([start], stop[, step])

start: número inicial da sequência

stop: gera números até, mas não inclui este número

step: Diferença entre 2 valores consecutivos da sequência



```
>>> for i in range(3):
                                >>> for i in range(4,10,2):
                                         print(i)
         print(i)
                                4
0
                                6
                                8
>>> for i in range(3,6):
                                >>> for i in range(0, -8, -2):
                                        print(i)
         print(i)
3
                                0
                                -2
4
5
                                -4
                                -6
```

Exemplo – listas

```
>>> cheeses=['Cheddar', 'Edam', 'Gouda']
>>> for i in range(len(cheeses)):
        print(cheeses[i])
Cheddar
Edam
Gouda
>>> numbers=[1,2,3,4,5]
>>> for i in range(len(numbers)):
        numbers[i] = numbers[i]*2
```

break e continue

Comportamento idêntico ao do ciclo while

break

Sai do ciclo

Se existirem ciclos encaixados (ciclos dentro de ciclos), sai do ciclo mais recente

continue

Passa à iteração seguinte ignorando o resto do bloco nesta iteração



Exemplo – números primos entre 2 e 100

```
for n in range(2,101):
    divisores=False
    for x in range(2,n//2):
        if n%x==0:
            divisores=True
            break
    if not divisores:
        print(n, 'é primo!')
```

while vs. for

São instruções para ciclos

while: mais genérico

Pode usar-se para percorrer listas

for: para iterar sobre sequências

Simplifica a sintaxe

Facilita a leitura

Não requer a manipulação explícita da variavel envolvida na condição



```
numbers=[10,20,30]
i=0
while i<len(numbers):
    print(numbers[i])
    i=i+1</pre>
numbers=[10,20,30]
for n in numbers:
print(n)
```



Algoritmo

É um processo mecânico para resolver problemas

A tabuada não é um exemplo de algoritmo

Mas o "truque" para multiplicar um inteiro com 1 algarismo por 9 pode ser considerado um algoritmo!

Os algoritmos

não precisam de "inteligência" para serem executados...

... mas é necessário "inteligência" para concebê-los!

São um componente fulcral da Programação!

