Fundamentos da Programação

Ciclos contados

Aula 8

José Monteiro

(slides adaptados do Prof. Alberto Abad)

Ciclos Contados (com `while`)

- Para percorrer tuplos (exemplos 2 e 3 da última aula), temos utilizado a instruição `while` com um *contador*:
  - O *contador* é inicializado antes do iníco do *while* (`i = 0`)
  - O *contador* é atualizado no corpo do ciclo (`i = i + 1`)
  - É definida uma condição de paragem (`i < tamanho`)

```
vector = (1, 2, 3)
i = 0
tamanho = len(vector)
while i < tamanho:
    print(vector[i])
    i = i + 1
```

In [ ]:

Ciclos Contados (com `for`)

- O Python fornece um mecanismo para *iterar* sobre uma sequência de valores chamado instrução `for`.
- Sintaxe BNF:

```
<instrução for> ::= for <nome simples> in <iterável>: NEWLINE
                        <bloco de instruções>
```

- `<iterável>` em Python corresponde a várias entidades, como por exemplo as sequências e os tuplos.
- A instrução `break` permite interromper ciclos (tal como no `while`)

In [ ]:
```
t = (1, 'a', 3, 4, 6)

for elemento in t:
    print(elemento)

# Exemplo break: sair se um elemento é um inteiro maior que 2
```

Ciclos Contados: Exercício 1

Soma Elementos com `for`

In [ ]:
```
def soma_elementos_for(t):
    soma = 0
    for elemento in t:
        soma += elemento
    return soma

print(soma_elementos_for((1,2,3,7)))
```

Sequências de Inteiros com `range`

- A função *built-in* `range` retorna um objeto iterável correspondente a uma sequência de inteiros:
  - útil para indexar sequências
- Sintaxe BNF:

```
<range> ::= range(<argumentos>)
<argumentos> ::= <expressão> | <expressão>, <expressão> |
<expressão, <expressão>, <expressão>
```

- Os valores de `<expressão>` são do tipo inteiro:
  - O primeiro argumento define o início da sequência (inclusive)
  - O segundo argumento define o fim da sequência (exclusive)
  - O terceiro argumento define o passo ou incremento

Sequências de Inteiros com `range`

```
>>> tuple(range(10))
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> tuple(range(5,10))
(5, 6, 7, 8, 9)
>>> tuple(range(-5,10))
(-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> tuple(range(-5,10,2))
(-5, -3, -1, 1, 3, 5, 7, 9)
>>> tuple(range(-5,10,-2))
()
>>> tuple(range(10,-5,-2))
(10, 8, 6, 4, 2, 0, -2, -4)
>>> tuple(range(10,-5,-1))
(10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4)
>>> vector = ('a', 'b', 'c', 'd', 'e')
>>> tuple(range(len(vector)))
(0, 1, 2, 3, 4)
```

In [ ]:

Ciclos Contados: Exercício 2

Soma Elementos com `for` e `range`

In [ ]:

```
def soma_elementos_for_range(t):
    soma = 0
    for i in t:
        if i % 2 == 0:
            soma = soma + i
    return soma

def soma_vetores(v1, v2):
    res = ()
    for i in range(len(v1)):
        res = res + (v1[i] + v2[i],)
    return res

print(soma_elementos_for_range((2,4,1,5,9,7,3,6)))

# SÓ ÍNDICES / VALORES PARES
# SEM IF
```

Ciclos Contados: Exercício 3

Ciclos aninhados (*Nested loops*)

In [ ]:

```
def tabelas():
    for x in range(1, 11):
        for y in range(1, 11):
            print(x, "x", y, "=", x * y)
        print("")
    return

tabelas()
```

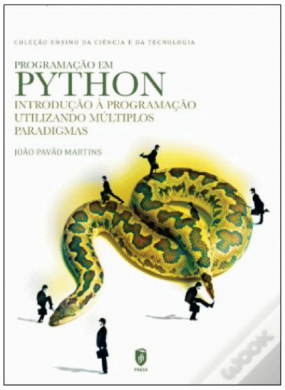
Ciclos Contados - Notas Finais

- Tudo o que faz o `for` , pode ser feito com `while`
- O `for` é mais eficiente e normalmente preferível
- Nem sempre o `for` é adequado, em particular, quando iteramos sobre um objeto que pode ser alterado em cada ciclo. Exemplo: a função `alisa`
- **Exercícios** com `for` e `range` :
  - Ex.1: Defina uma função que calcula a soma dos primeiros n números naturais (progressão aritmética)
  - Ex.2: Defina uma função que permite verificar se um tuplo com valores numéricos está ordenado
- **(Opcional avançado)** A função *built-in* `enumerate` permite iterar sobre os valores duma sequência e obter um índice ao mesmo tempo:

In [ ]:
```
vector = ('a','b','c')
for i, v in enumerate(vector):
    print (i, v)
```

Ciclos Contados - Tarefas próxima aula

- Praticar matéria apresentada hoje
- Nas aulas de problemas: tuplos e ciclos contados



In [ ]:
```

```