Fundamentos de Programação @ LEIC/LETI

Aulas 07 e 08

Tuplos e Ciclos contados

Tuplos. Exemplos. Ciclos contados. Cadeias de caracteres revisitadas. Exemplos.

Alberto Abad, Tagus Park, IST, 2018

Tuplos

- Um tuplo é uma sequência de elementos.
- Elementos são referenciados através do seu índice ou posição.
- Representação externa de um tuplo em Python (BNF):

```
<tuplo> ::= () | (<elemento>, <elementos>)
<elementos> ::= <nada> | <elemento> | <elemento>, <elementos>
<elemento> ::= <expressão> | <tuplo> | <lista> | <dicionário>
<nada> ::=
```

Exemplos de tuplos

```
>>> type(())
<class 'tuple'>
>>> type((2))
<class 'int'>
>>> type((2,))
<class 'tuple'>
>>> type((2,4,5))
<class 'tuple'>

>>> type((2,4,5,))
<class 'tuple'>
>>> type((2,4,5,'ola'))
<class 'tuple'>
>>> type((2,4,5,'ola',(8,9,)))
<class 'tuple'>
>>> type((2,4,(False,5),True,(8,9,)))
<class 'tuple'>
```

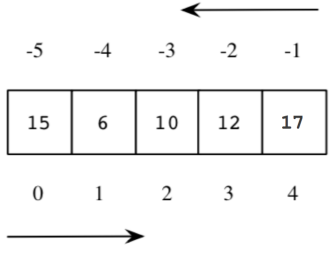
In [ ]:
```
type((2, 3*3, (2,3), 'str'))
```

Referir elmentos de um tuplo

- Sintaxe BNF:

```
<nome indexado> ::= <nome>[<expressão>]
```

- Índices (inteiros):



Exemplos indexação de tuplos

```
>>> notas = (15, 6, 10, 12, 17)

>>> notas[0]
15
>>> notas[2]
10
>>> notas[-1]
17
>>> notas[-2]
12

>>> notas[3+1]
17
>>> i = 5
>>> notas[i-4]
6
```

In [ ]:
```
notas = (15, 6, 10, 12, 17)
```

Exemplos indexação de tuplos

```
>>> notas[9]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range

>>> v = (12, 10, (15, 11, 14), 18, 17)
>>> v[2]
(15, 11, 14)
>>> v[2][1]
11

>>> v[2] = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

In [ ]:
```
notas[9]
```

Operações sobre tuplos

Operação	*Tipo dos argumentos*	*Valor*
t_1 `+` t_2	Tuplos	A concatenação dos tuplos t_1 e t_2.
t `*` i	Tuplo e inteiro	A repetição i vezes do tuplo t.
t`[`i_1`:`i_2`]`	Tuplo e inteiros	O sub-tuplo de t entre os índices i_1 e $i_2 - 1$.
e `in` t	Universal e tuplo	`True` se o elemento e pertence ao tuplo t; `False` em caso contrário.
e `not in` t	Universal e tuplo	A negação do resultado da operação e `in` t.
`tuple(`a`)`	Lista ou dicionário ou cadeia de carateres	Transforma o seu argumento num tuplo. Se não forem fornecidos argumentos, devolve o tuplo vazio.
`len(`t`)`	Tuplo	O número de elementos do tuplo t.

Exemplos de operações sobre tuplos: +/*

```
>>> a = (2, 1, 3, 7, 5)
>>> b = (8, 2, 4, 7)
>>> a + b
(2, 1, 3, 7, 5, 8, 2, 4, 7)
>>> c = a + b

>>> a * b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'tuple'
>>> a * 2
(2, 1, 3, 7, 5, 2, 1, 3, 7, 5)
```

- Note-se a sobrecarga dos operadores + e *
- Que acontece com `a + (2)`?

In [ ]:
```
a = (2, 1, 3, 7, 5)
b = (8, 2, 4, 7)
a + b
```

Exemplos de operações sobre tuplos: *slicing*

- Seleção dos elementos dum tuplo (sub-tuplo) des de possição inicial (*inclusive*) até possição final (*exclusive*) com passos ou incrementos fixos:

```
vector[inicio:fim:incremento] -->  (vector[inicio], vector[inicio+1*incremento],
 ..., vector[inicio+i*incremento])
```

```
>>> a = (2, 1, 3, 7, 5)
```

```
>>> a[2:4]
(3, 7)
>>> c[2:5]
(3, 7, 5)
>>> a[:3]
(2, 1, 3)
>>> a[4:]
(5,)
>>> a[:]
(2, 1, 3, 7, 5)
```

```
>>> a[::2]
(2, 3, 5)
>>> a[-1::-1]
(5, 7, 3, 1, 2)
```

In [ ]:
```
a = (2, 1, 3, 7, 5)
```

Exemplos de operações sobre tuplos: *in, not in, len, tuple*

```
>>> a = (2, 1, 3, 7, 5)
>>> b = (8, 2, 4, 7)

>>> 1 in a
True
>>> 1 in b
False
>>> 'b' not in b
True

>>> len(a)
5

>>> tuple('hello world')
('h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd')
>>> tuple(8)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
>>>
```

Sobre a imutabilidade dos tuplos

```
>>> a = (3, 4, 5, 6)
>>> b = (7, 8)
>>> a = a + b
>>> a
(3, 4, 5, 6, 7, 8)
```

- O que esta a acontecer? Os tuplos não são imutáveis!?
- Num tuplo ser *imutável*:
  - Não podemos alterar um valor de um elemento de um tuplo.
  - Podemos criar tuplos (com mesmo nome) a partir de outros tuplos.
  - Para efectuarmos transformações sobre tuplos temos de aplicar as operações acima e construir novos tuplos.

In [ ]:
```
a = (3, 4, 5, 6)
b = (7, 8)
a = a + b
a
```

Sobre a imutabilidade dos tuplos

Python 3.6

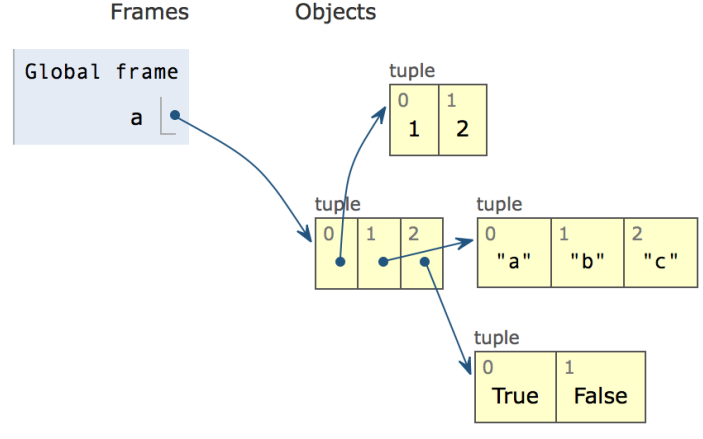
```
1  a = ((1,2),('a','b','c'),(True, False))
2  a0 = a[0]
3  a1 = a[1]
4  a2 = a[2]
5  a = a[:1] + a[2:]
6  a1 = ('d', 'e')
```

Edit this code

line that has just executed
next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First | < Back | Step 2 of 6 | Forward > | Last >>



Exemplo 1: Substituir um elemento de um tuplo

```
def substitui(t, p, e):
    if not (0 <= p < len(t)):
        raise IndexError('substitui: no tuplo dado como primeiro argumento')
    return t[:p] + (e,) + t[p+1:]
```

Exemplos:

```
>>> a = (2, 1, 3, 3, 5)
>>> substitui(a, 2, 'a')
(2, 1, 'a', 3, 5)
>>> substitui(a, 4, 'a')
(2, 1, 3, 3, 'a')
>>> a = substitui(a, 0, 'a')

>>> a = substitui(a, 5, 'a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in substitui
IndexError: substitui: no tuplo dado como primeiro argumento
>>> a
('a', 1, 3, 3, 5)
>>>
```

In [ ]:

```
def substitui(t, p, e):
    if not (0 <= p < len(t)):
        raise IndexError('substitui: no tuplo dado como primeiro argumento')
    return t[:p] + (e,) + t[p+1:]
```

Exemplo 2: Percorrer elementos tuplo, *soma_elementos*

```
def soma_elementos(t):
    soma = 0
    i = 0
    while i < len(t):
        soma = soma + t[i]
        i = i +1
    return soma
```

In [ ]:

```
def soma_elementos(t):

    soma = 0
    i = 0
    while i < len(t):
        soma = soma + t[i]
        i = i +1

    return soma

print(soma_elementos((1,2,3)))

#question1: como optimizar a condição?
#question2: alterações para obter tuplo de quadrados
#question3: como verificar tipos?
```

Exemplo 3: Percorrer elementos tuplo, *soma_vectores*

In [ ]:
```
def soma_vectores(v1, v2):

    if len(v1) != len(v2):
        raise ValueError("Dimensão vectores diferente")

    tamanho = len(v1)
    soma = ()
    i = 0
    while i < tamanho:
        soma = soma + (v1[i] + v2[i],)
        i = i +1

    return soma

print(soma_vectores((1,2,3),(0,7,2)))
```

Exemplo 4: Função *alisa*

```
>>> alisa((2, 4, (8, (9, (7, ), 3, 4), 7), 6, (5, (7, (8, )))))
(2, 4, 8, 9, 7, 3, 4, 7, 6, 5, 7, 8)
```

t	i	t[:i]	t[i]	t[i+1:]
((1, 2), 3, (4, (5)))	0	()	(1, 2)	(3, (4, 5))
(1, 2, 3, (4, 5))	1			
(1, 2, 3, (4, 5))	2			
(1, 2, 3, (4, 5))	3	(1, 2, 3)	(4, 5)	()
(1, 2, 3, 4, 5)	4			

Exemplo 4: Função *alisa*, *isinstance*

- Para escrever a função `alisa`, iremos utilizar a função embedida `isinstance`, em BNF:

```
<isinstance> ::= isinstance(<expressão>, <designação de tipo>)
<designação de tipo> ::= <expressão> | <tuplo>
```

- Alternativa a `type` que retorrna `True` ou `False` is isinstance (https://docs.python.org/3/library/functions.html#isinstance):

```
>>> isinstance(3, int)
True
>>> isinstance(3, (int, bool))
True
>>> isinstance(True, (int, bool))
True
>>> isinstance(5.6, (int, bool))
False

>>> isinstance('a', (int, bool))
False
>>> isinstance('a', (int, bool, str))
True
>>> isinstance((8,), tuple)
True
>>>
```

Exemplo 4: Função *alisa*, proposta de solução

In [ ]:
```
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

alisa((2, 4, (8, (9, (7, ), 3, 4), 7), 6, (5, (7, (8, )))))
```

Ciclos contados (com `while`)

- Para percorrer tuplos (Exemplos 2 e 3), 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 [ ]:
```
vector = (1, 2, 3)
i = 0
tamanho = len(vector)
while i < tamanho:
    print(vector[i])
    i = i + 1
```

Ciclos contados (com `for`)

- 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 instruçõe
s
```

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

In [ ]:
```
vector = (1, 2, 3)

for e in vector:
    print(e)

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

Exemplo 5: *soma_elementos* com `for`

In [ ]:
```
def soma_elementos2(t):

    soma = 0
    for e in t: # e é um elemento do tuplo
        soma = soma + e

    return soma

print(soma_elementos2((1,2,3)))
```

Sequências de inteiros com `range`

- A função *built-in* `range` retorna um objeto iterável corresnpondente 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`, Exemplos:

```
>>> 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 [ ]:
```
tuple(range(-5, 10))
```

Exemplo 6: *soma_elementos* com `for` e `range`

In [ ]:
```
def soma_elementos3(t):

    soma = 0
    for i in range(len(t)): # i é um índice
        soma = soma + t[i]

    return soma

print(soma_elementos3((1,2,3)))
```

Exemplo 7: Ciclos annihados (*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` nos permete iterar sobre os valores duma sequeência e obter um índice ao mesmo tempo:

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

Cadeias de carateres revisitadas

- Em Python, as cadeias de carateres (`str`) são um tipo estruturado **imutável** correspondente a uma **sequência** de carateres individuais.
- São definidas de acordo com a sintaxe BNF:

```
<cadeia de carateres> ::=  '<caráter>*' | "<caráter>*" | """<caráter>*"""
```

- A sequência de caracteres com 0 caracteres, ou vazia, é representada por `''` ou `""`.

- Nota-se que a triple `"` em Python é também utilizada para documentação:

Cadeias de carateres revisitadas - Operações e funções *built-in*

Operação	*Tipo dos argumentos*	*Valor*
s_1 `+` s_2	Cadeias de carateres	A concatenação das cadeias de carateres s_1 e s_2.
s `*` i	Cadeia de carateres e inteiro	A repetição i vezes da cadeia de carateres s.
s`[`i_1`:`i_2`]`	Cadeia de carateres e inteiros	A sub-cadeia de carateres de s entre os índices i_1 e $i_2 - 1$.
e `in` s	Cadeias de carateres	`True` se e pertence à cadeia de carateres s; `False` em caso contrário.
e `not in` s	Cadeias de carateres	A negação do resultado da operação e `in` s.
`len(`s`)`	Cadeia de carateres	O número de elementos da cadeia de carateres s.
`str(`a`)`	Universal	Transforma o seu argumento numa cadeia de carateres.

Cadeias de carateres revisitadas - Indexação e *slicing*

- Tal como os tuplos, as *strings* são sequências e podemos aceder aos seus elementos de forma idêntica:

```
>>> fp='Fundamentos da Programacao'

>>> fp[0]
'F'
>>> fp[15:]
'Programacao'

>>> fp[:11]
'Fundamentos'
>>> fp[-3:]
'cao'
>>> fp[::2]
'Fnaetsd rgaaa'
```

In [ ]:
```
fp='Fundamentos da Programacao'
```

Cadeias de carateres revisitadas - Operações e funções *built-in*

```
>>> f = 'Fundamentos'
>>> p = 'Programacao'

>>> f + ' de ' + p
'Fundamentos de Programacao'
>>> f*3
'FundamentosFundamentosFundamentos`

>>> 'c' in p
True
>>> 'c' in f
False

>>> len(p)
11
>>> str(9+8)
'17'
>>> str((9,8,20))
'(9, 8, 20)'
>>> eval('f + p')
'FundamentosProgramacao'
>>>
```

In [ ]:
```
f = 'Fundamentos'
p = 'Programacao'
```

Cadeias de carateres revisitadas - Exemplo *simbolos_comum*

In [ ]:
```
def simbolos_comum(s1, s2):
    res = ''
    for e in s1:
        if e in s2:
            res += e
    return res

f1 = 'Fundamentos da programação'
f2 = 'Algebra linear'
simbolos_comum(f1, f2)

#question: alterar para não mostrar repetidos
```

Cadeias de carateres revisitadas - Representação interna caracteres

- Os carateres são representados dentro do computador associados a um código numérico.
- O Python utiliza o **Unicode** (o código ASCII está contido no Unicode)
- Funções built-in relacionas:
  - `ord`: devolve o código numérico (unicode) de um caráter
  - `chr`: devolve o string correspondente a um código numérico (unicode)

```
>>> ord('A')
65
>>> ord('a')
97
>>> chr(97)
'a'
```

Cadeias de carateres revisitadas - Exemplo *toupper*

In [ ]:

```
def toupper(s):
    i = 0
    while i < len(s):
        if ord('a') <= ord(s[i]) <= ord('z'):
            s = s[:i] + chr(ord(s[i]) - ord('a')
                            + ord('A')) + s[i+1:]
        i = i + 1

    return s

def toupper2(s):
    ns = ''
    for c in s:
        if ord('a') <= ord(c) <= ord('z'):
            ns = ns + chr(ord(c) + ord('A') - ord('a'))
        else:
            ns = ns + c
    return ns

print(toupper('aBceF'))
print(toupper2('aBceF'))
```

Cadeias de carateres revisitadas - Mais operações

- Como as strings correspondem a sequências de códigos numéricos (Unicode), as seguintes operações são possíveis:

```
>>> 'a' < 'z'
True
>>> 'a' < 'Z'
False
>>> 'a' > 'Z'
True

>>> 'Fundamentos' > 'Programacao'
False
>>> 'fundamentos' > 'Programacao'
True
>>> 'fundamentos' > 'fundao'
False
>>> 'fundamentos' < 'fundao'
True
>>>
```

In [ ]:
```
'a' < 'z'
```

Cadeias de carateres revisitadas - Exemplo verifica ISBN

- O International Standard Book Number (ISBN) é um sistema de identificação de livros e softwares que utiliza números únicos para classificá-los por título, autor, país, editora e edição:
  - **ISBN-10** (antes de 2007): O último dígito (x_9) é de controlo e varia de 0 a 10 (o símbolo 'X' é usado em vez de 10) e deve ser tal que:

$$(1 * x_0 + 2 * x_1 + 3 * x_2 + 4 * x_3 + 5 * x_4 + 6 * x_5 + 7 * x_6 + 8 * x_7 + 9 * x_8 + 10 * x_9) \, mod 11 = 0$$

  - **ISBN-13** (desde 2007): O último dígito (x_{12}) é de controlo e varia de 0 a 9 e deve ser tal que:

$$(x_0 + 3 * x_1 + x_2 + 3 * x_3 + x_4 + 3 * x_5 + x_6 + 3 * x_7 + x_8 + 3 * x_9 + x_{10} + 3 * x_{11} + x_{12}) \, mod 10 = 0$$

Cadeias de carateres revisitadas - Exemplo verifica ISBN-10

In [1]:
```
def verifica_isbn10(isbn):
    def codigo_control(isbn):
        if isbn[-1] == 'X':
            return 10
        else:
            return int(isbn[-1])
        ## advanced (ternary operator)
        ## return 10 if (isbn[-1] == 'X') else int(isbn[-1])

    soma = 0
    for i in range(len(isbn)-1):
        soma += (i+1)*int(isbn[i])
    soma += 10*codigo_control(isbn)

    return soma%11 == 0

verifica_isbn10('054792822X') # https://isbnsearch.org/isbn/054792822X
```

Out[1]:
```
True
```

Tuplos e ciclos contados

Cadeias de carateres revisitadas - Exemplo verifica ISBN-13

In [ ]:
```
def verifica_isbn13(isbn):
    soma = 0
    for i in range(len(isbn)-1):
        soma += ((2*i +1)%4)*int(isbn[i])
    soma += int(isbn[-1])

    return soma%10 == 0

verifica_isbn13('9789898481474') # https://isbnsearch.org/isbn/9789898481474
```

Cadeias de carateres revisitadas - Exemplo verifica ISBN, valida argumentos

In [ ]:

```
def allValidChars(isbn):
    if len(isbn) == 10:
        for i in range(len(isbn) -1):
            if not '0'<= isbn[i] <= '9':
                return False
        if not ('0' <= isbn[-1] <= '9' or  isbn[-1] == 'X'):
            return False
    elif len(isbn) == 13:
        for i in range(len(isbn)):
            if not '0'<= isbn[i] <= '9':
                return False
    else:
        return False
    return True

#NOTA: Quando aprendamos funcionais e/ou PO, podemos fazer esta função numa linha
 de codigo
```

Cadeias de carateres revisitadas - Exemplo verifica ISBN

In [ ]:
```
def verifica_isbn(isbn):
    if not (type(isbn) == str and allValidChars(isbn)):
        raise ValueError("Invalid ISBN string")
    elif len(isbn) == 10:
        return verifica_isbn10(isbn)
    else:
        return verifica_isbn13(isbn)

verifica_isbn('054792822X') # https://isbnsearch.org/isbn/054792822X
verifica_isbn('9789898481474') # https://isbnsearch.org/isbn/9789898481474
```

Cadeias de carateres revisitadas - Formatação

- Como formatar *strings* com `''.format()` (novo estilo)
- Mais informação neste link (https://pyformat.info)

In [ ]:
```
print('Inteiros: {} e {:d}'.format(1, 2))
print('Floats: {} e {:f}'.format(1.456, 2.3007))
print('Strings: {} e {:s}'.format('um','dois'))
print('Primeiro é {} e Segundo é {}'.format(1, 2))
print('Segundo é {1} e Primeiro é {0}'.format(1, 2))
print('Primeiro é {first} e Segundo é {second}'.format(first=1, second='dois'))
print('Primeiro é {first} e Segundo é {second}'.format(second='dois', first=1))
print('{:<20}'.format('FP'))
print('{:>20}'.format('FP'))
print('{:^20}'.format('FP'))
from math import pi
print('{:.2f}'.format(pi))
print('{:.6f}'.format(pi))
print('{:0>20.6f}'.format(pi))
```

Cadeias de carateres revisitadas - Exemplo de formatação, horário

In [ ]:
```
slot = 0
for h in range (0,24):
    for m in range(0,60,30):
        print("Time slot {:03d} --> {:02d}:{:02d}".format(slot, h, m))
        slot += 1
```