Programação I
Aula 9 — Listas e tuplos

Pedro Vasconcelos
DCC/FCUP

Nesta aula

Listas

- Sequências ordenadas de elementos
- Podem conter elementos de quaisquer tipos
- Possivelmente com repetições
- Os elementos são identificados por índices

Exemplo

```
>>> alimentos = ["pão", "pão", "queijo", "queijo"]
>>> alimentos[0]
'pão'
>>> alimentos[1]
'pão'
>>> alimentos[2]
'queijo'
>>> len(alimentos)
4
```

Listas por extensão

- Lista com *n* elementos: `[e1, e2, ..., en]`
- A ordem é significativa
- Podem ocorrer elementos repetidos
- Pode ser a lista vazia: `[]`

Acesso aos elementos

- Operador de indexação: `lista[i]`
- Índices entre 0 e `len(lista)-1`
- Índices negativos: acesso a partir do fim
- Índices inválidos dão um erro de execução

Fatias

`lista[i:j]` elementos entre i e $j-1$ inclusíve

`lista[i:]` elementos entre i até ao final

`lista[:j]` elementos do primeiro até $j-1$ inclusíve

`lista[:]` todos os elementos (cópia da lista)

```
>>> vogais = ['a','e','i','o','u']
>>> vogais[1:4]
['e', 'i', 'o']
>>> vogais[:3]
['a','e','i']
>>> vogais[3:]
['o','u']
>>> vogais[:]
['a', 'e', 'i', 'o', 'u']
```


Fatias (cont.)

Forma geral

`lista[i:j:k]` elementos de i a $j-1$ com incrementos k

Incrementos negativos: percorrer a lista ao contrário.

```
>>> vogais[::2]      # índices pares
['a','i','u']
>>> vogais[1::2]     # índices ímpares
['e','o']
>>> vogais[::-1]     # inverter a lista
['u', 'o', 'i', 'e', 'a']
```

Percorrer os índices e elementos

```
for i in range(len(lista)):
    print(i, lista[i])
```

- Ciclo sobre índices *i* de 0 até *len*(*lista*) − 1
- Elemento *lista*[*i*] associado ao índice *i*

Percorrer todos os elementos

```
for valor in lista:
    print(valor)
```

- Evita manipular explicitamente o índice
- Preferível quando necessitamos dos valores mas não dos índices

Operações com listas

$+$	concatenação
$n*$	repetição (n vezes)

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


Listas são mutáveis

Podemos modificar ou acrescentar elementos:

```
>>> beatles = [1, 2, 3]
>>> beatles[0] = "john"
>>> beatles[2] = "ringo"
>>> beatles
['john', 2, 'ringo']

>>> beatles[1:2] = ['paul', 'george']
>>> beatles
['john', 'paul', 'george', 'ringo']
```

Remover elementos duma lista

```
>>> beatles = ['john', 'paul', 'george', 'ringo']
>>> del beatles[0]
>>> beatles
['paul', 'george', 'ringo']
```

Alternativa:

```
>>> beatles = ['john', 'paul', 'george', 'ringo']
>>> beatles[0:1] = []
>>> beatles
['paul', 'george', 'ringo']
```


Nomes e objectos

É importante distinguir o nome de uma variável da coleção de valores associada a esse nome.

Nomes e objectos (1)

Dois nomes, duas listas separadas:

```
>>> a = [1,2,3]
>>> b = [1,2,3]
>>> a[0] = 'oops'
>>> print(a, b)
['oops', 2, 3] [1, 2, 3]
```


Nomes e objectos (2)

Dois nomes, apenas uma lista:

```
>>> a = [1,2,3]
>>> b = a
>>> a[0] = 'oops'
>>> print(a, b)
['oops', 2, 3] ['oops', 2, 3]
```

Nomes e objectos (3)

Dois nomes, duas listas (fazendo uma cópia):

```
>>> a = [1,2,3]
>>> b = a[:]
>>> a[0] = 'oops'
>>> print(a, b)
['oops', 2, 3] [1, 2, 3]
```


Métodos sobre listas

Alguns métodos pré-definidos:

append	acrescentar um elemento ao final
insert	acrescentar um elemento numa posição
remove	remover um elemento
sort	ordenar os elementos por ordem crescente

- Utilização: `lista.método(argumentos)`
- Modificam a lista

Exemplos

```
>>> beatles = ['john','paul']
>>> beatles.append('george')
>>> beatles.append('ringo')
>>> beatles
['john', 'paul', 'george', 'ringo']
>>> beatles.insert(0, 'paul')
>>> beatles
['paul', 'john', 'paul', 'george', 'ringo']
>>> beatles.sort()
>>> beatles
['george', 'john', 'paul', 'paul', 'ringo']
```

Para obter mais informação:

```
>>> help(list)
```


Listas dentro de listas

- As listas podem conter outras listas
- Podemos assim representar tabelas ou matrizes

```
>>> matriz = [[1,2,-1],
              [3,1,0],
              [0,1,-2]]
>>> matriz[1][0]
3
>>> matriz[1][0] = -3
>>> matriz
[[1, 2, -1], [-3, 1, 0], [0, 1, -2]]
```

Tuplos

- Sequências ordenadas de elementos:

```
(e1, e2, ..., en)
```

- Acesso aos elementos por índices
- Ao contrário das listas, os tuplos são imutáveis

Exemplo

```
>>> nota = ('Pedro', 12)
>>> nota[0]
'Pedro'
>>> nota[1]
12
>>> nota[0] = 'Joao'
TypeError: 'tuple' object does not support
item assignment
```

Operações sobre tuplos

Operadores + e * análogos aos de listas:

```
>>> nt1 = ('Pedro', 12)
>>> nt2 = ('Joao', 14)
>>> nt1 + nt2
('Pedro', 12, 'Joao', 14)
>>> 3*nt1
('Pedro', 12, 'Pedro', 12, 'Pedro', 12)
```


Atribuição a tuplos de variáveis

```
>>> (x,y) = (5,-7)
>>> x
5
>>> y
-7
```

Ou simplesmente:

```
>>> x,y = 5,-7
>>> x
5
>>> y
-7
```

Listas e tuplos combinados

Vamos representar uma agenda como uma lista de pares *nome*/*email*:

```
[('Pedro Vasconcelos', 'pbv@dcc.fc.up.pt'),
 ('Pedro Brandão', 'pbrandao@dcc.fc.up.pt'),
 ('João Pedro Pedroso', 'jpp@dcc.fc.up.pt')]
```

Operações:

- acrescentar uma entrada (nome e email)
- procurar email pelo nome

Acrescentar uma entrada

```
def acrescentar(agenda, nome, email):
    "Acrescentar um nome e email à agenda."
    agenda.append((nome,email))
```

Procurar um nome (1)

```
def procurar(agenda, txt):
    "Procurar emails por parte do nome."
    emails = []
    for par in agenda:
        if txt in par[0]:          # txt ocorre no nome?
            emails.append(par[1]) # acrescenta email
    return emails
```


Procurar um nome (2)

```
def procurar(agenda, txt):
    "Procurar emails por parte do nome."
    emails = []
    for (nome,email) in agenda:
        if txt in nome:            # txt ocorre no nome?
            emails.append(email) # acrescenta email
    return emails
```

Exemplos

```
>>> agenda = []
>>> acrescentar(agenda, "Pedro Vasconcelos",
                "pbv@dcc.fc.up.pt")
>>> acrescentar(agenda,"João Pedro",
               "jpp@dcc.fc.up.pt")

>>> procurar(agenda, "João")
['jpp@dcc.fc.up.pt']

>>> procurar(agenda, "Pedro")
['pbv@dcc.fc.up.pt', 'jpp@dcc.fc.up.pt']
```


Anotações de tipos

- Podemos usar *anotações de tipos* para documentar os argumentos de funções
- O sistema de testes automáticos usa essas anotações para detetar potenciais erros
- Nos exercícios anteriores usamos apenas *tipos básicos*: `int`, `float`, `str`, `bool`
- Vamos agora ver como anotar *listas* e *tuplos*

Anotações de tipos

`List[`*T*`]`	é o tipo das *listas* de *T*
`Tuple[`*T*1, *T*2`]`	é o tipo dos *pares* de *T*1 e *T*2
`Tuple[`*T*1, *T*2, *T*3`]`	é o tipo dos *trios* de *T*1, *T*2 e *T*3

- O tipo de listas diz quais o *tipo dos elementos* dentro da lista mas não o *comprimento*
- O tipo de tuplos diz o *tipo e também o número* de elementos
- Se não quiseremos "fixar" o tipo podemos usar `Any`

Alguns exemplos

`[1, 2, 3]`	tem tipo	`List[int]`
`[-12, 1]`	tem tipo	`List[int]`
`['hello', 'world']`	tem tipo	`List[str]`
`['hello', 42]`	tem tipo	`List[Any]`
`(-12, 1)`	tem tipo	`Tuple[int,int]`
`('alfa', 12)`	tem tipo	`Tuple[str,int]`
`('beta', 1, -0.5)`	tem tipo	`Tuple[str,int,float]`
`[[0], [2,-1]]`	tem tipo	`List[List[int]]`
`[(0,0), (2,-1)]`	tem tipo	`List[Tuple[int,int]]`

Anotando o exemplo

- Vamos agora anotar as funções da agenda de *email*
- Necessitamos da seguinte linha no ínício do ficheiro:

```
from typing import *
```


Anotando o exemplo (cont.)

```
from typing import *

def acrescentar(agenda: List[Tuple[str,str]],
                nome: str,
                email: str):
    agenda.append((nome,email))

def procurar(agenda: List[Tuple[str,str]],
             txt: str) -> List[str]:
    emails = []
    for (nome,email) in agenda:
        if txt in nome:
            emails.append(email)
    return emails
```