Programação I
Aula 14 — Dicionários

Pedro Vasconcelos
DCC/FCUP

Nesta aula

Dicionário

- Estrutura de dados para uma tabela de associações:
  - cada *chave* é associada a um (e um só) *valor*
- Analogia: dicionário bilingue (e.g. português-inglês)
- Podemos como chaves apenas *tipos imutáveis*:
  - números
  - cadeias de caracteres
  - tuplos
  - combinações dos anteriores

Exemplo: um inventário

Um inventário que associa *quantidades disponíveis* a *frutas*.

Item	**Quantidade**
bananas	25
peras	12
laranjas	10

Exemplo: um inventário (cont.)

Poderiamos representar como uma lista de pares:

```
[ ('bananas',25), ('laranjas',10), ('peras',12) ]
```

Desvantagens:

- necessário percorrer a lista para procurar o valor associado a uma chave
- permite repetir chaves; por exemplo

  ```
  [('bananas',1), ('bananas',25),
   ('laranjas',10), ('peras',12),]
  ```

  representa quantas bananas (1, 25 ou 26)?

Usando um dicionário

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
```

Pedir o valor associado a uma chave:

```
>>> inv['bananas']
25
```

Modificar:

```
>>> inv['bananas'] = inv['bananas'] + 1
>>> inv
{'laranjas':10, 'peras':12, 'bananas':26}
```

Usando um dicionário (cont.)

Outros exemplos:

```
>>> emails = { 'Pedro':'pbv@dcc.fc.up.pt',
               'João':'jpp@dcc.fc.up.pt' }

>>> dirs = { (0,1):'N', (0,-1):'S',
             (-1,0):'W', (1,0):'E' }

>>> vazio = {}      # um dicionário vazio
```


Algumas operações sobre dicionários

- Obter o valor associado à chave:

  dict`[`*chave*`]`

  (erro se a chave não existir)
- Atribuir um valor a uma chave:

  dict`[`*chave*`] =` *valor*
- Testar se existe uma chave:

  chave `in` *dict*

  (resultado: True ou False)

Exemplos

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
>>> inv['bananas']
25

>>> inv['kiwis']
KeyError: 'kiwis'

>>> 'bananas' in inv
True
>>> 'kiwis' in inv
False
```


Mais operações

Obter o valor ou *default* se a *chave* não existir:

dict.get(*chave*,*default*)

Exemplos:

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
>>> inv.get('bananas',0)
25
>>> inv.get('kiwis',0)
0
```

Mais operações (cont.)

Percorrer todas as chaves:

```
for chave in dict.keys():
    ...
```

Exemplo:

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
>>> for fruit in inv.keys():
...     print(fruit)

bananas
peras
laranjas
```

Nota: as chaves não são listadas por ordem!

Mais operações (cont.)

Obter uma lista com todas as chaves:

```
list(dict.keys())
```

Exemplo:

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
>>> list(inv.keys())
['bananas', 'peras', 'laranjas']
```

Como anteriormente as chaves não estão ordenadas.

Mais operações (cont.)

Percorrer todos os pares de chaves e valores:

```
for chave,valor in dict.items():
    ...
```

Exemplo:

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
>>> for fruit,quant in inv.items():
...     print(fruit,quant)

bananas 25
peras 12
laranjas 10
```


Anotações de tipos

`Dict[`$T1$`,`$T2$`]` tipo dos dicionários cujas chaves são de tipo $T1$ e os valores são de tipo $T2$

Necessitamos da seguinte linha no ínício do ficheiro:

```
from typing import *
```

Anotações de tipos (cont.)

Exemplos:

```
>>> inv = {'bananas':25,'laranjas':10,'peras':12}
>>> emails = { 'Pedro':'pbv@dcc.fc.up.pt',
               'João':'jpp@dcc.fc.up.pt' }
>>> dirs = { (0,1):'N', (0,-1):'S',
             (-1,0):'W', (1,0):'E' }
```

`inv`	tem tipo	`Dict[str,int]`
`emails`	tem tipo	`Dict[str,str]`
`dirs`	tem tipo	`Dict[Tuple[int,int],str]`

Contar ocorrências de letras

Problema: contar o número de ocorrências de cada letra num ficheiro de texto.

(Uma tabela deste género chama-se um *histograma*.)

Vamos usar um dicionário para associar cada letra ao seu número de ocorrências.

Um texto de exemplo

Usamos o Canto I d'*Os Lusíadas*[1] como texto de exemplo.

lusiadas_cantoI.txt

```
1

As armas e os barões assinalados,
Que da ocidental praia Lusitana,
Por mares nunca de antes navegados,
Passaram ainda além da Taprobana,
Em perigos e guerras esforçados,
...
```

[1] Fonte: Instituto Camões `http://www.instituto-camoes.pt`.

Histograma de ocorrências

```
from typing import *

def histograma(fich: str) -> Dict[str,int]:
    '''Construir o dicionário de ocorrências
       das letras de um ficheiro.'''
    f = open(fich, "r", encoding="utf-8")
    hist: Dict[str,int] = {}  # inicializar
    while True:   # ler o texto linha-à-linha
        linha = f.readline().lower()
        if linha == "":
            break
        for c in linha: # para cada carater
            if c>='a' and c<='z':
                hist[c] = 1+hist.get(c,0)
    f.close()
    return hist
```

Contar ocorrências de letras

Canto I de *Os Lusíadas*

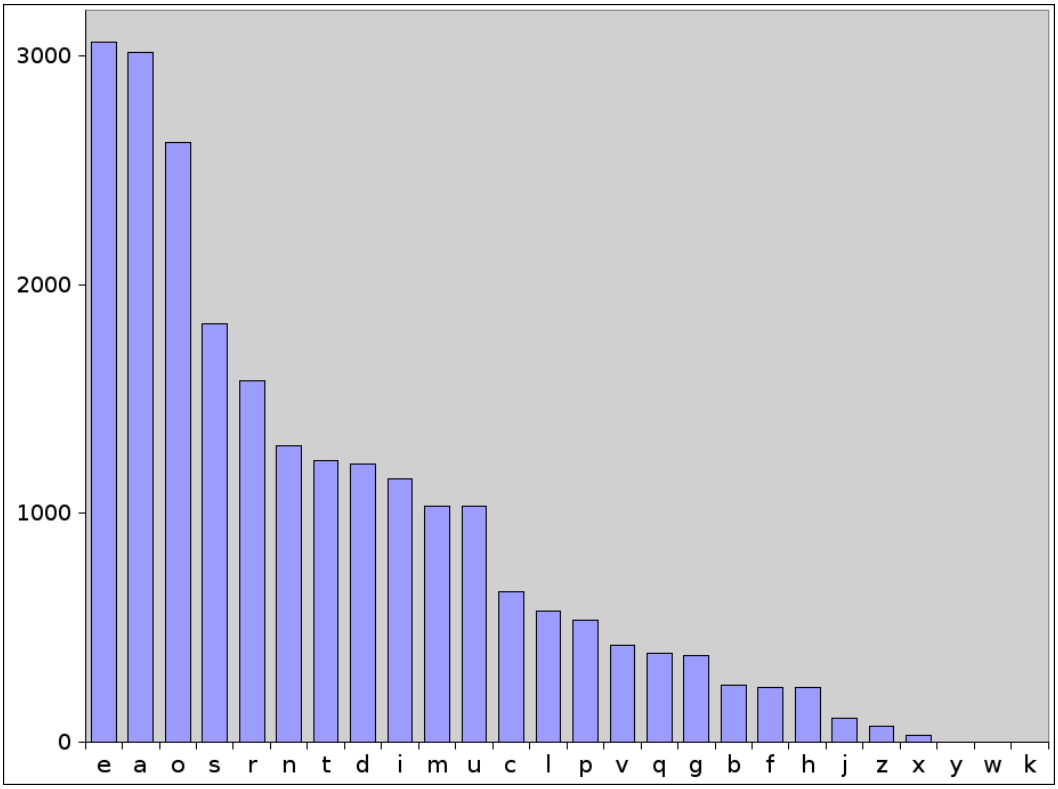
```
>>> histograma("lusiadas_CantoI.txt")
{'a': 3015, 'c': 658, 'b': 249, 'e': 3064,
 'd': 1217, 'g': 378, 'f': 240, 'i': 1154,
 'h': 239, 'j': 103, 'm': 1033, 'l': 572,
 'o': 2622, 'n': 1294, 'q': 390, 'p': 531,
 's': 1828, 'r': 1578, 'u': 1031, 't': 1229,
 'v': 425, 'y': 1, 'x': 29, 'z': 68}
```


Imprimir a tabela de contagems

```
hist = histograma("lusiadas_CantoI.txt")
for letra,cont in hist.items():
    print(letra, cont)  # letra, contagem
```

Podemos importar para uma folha de cálculo e visualizar gráficamente.

Gráfico do histograma



Observações

- A letra `'e'` é a que ocorre mais vezes (3064) seguida o `'a'` (3015)
- A letra `'y'` ocorre apenas 1 vez
- Contudo: os resultados não são exatos porque não contabilizamos letras acentuadas!
- Para corrigir esses casos, vamos converter letras acentuadas em não acentuadas

Digressão: normalização do texto

Em Unicode, uma letra acentuada pode ser representada de duas formas distintas:

NFC um só código numérico;
por exemplo, Ç é U+00C7 (LATIN CAPITAL LETTER C WITH CEDILLA);

NFD um *par de códigos* da letra e do acento;
por exemplo, Ç é U+0043 (LATIN CAPITAL LETTER C) U+0327 (COMBINING CEDILLA).

Digressão: normalização do texto (cont.)

A função `normalize` do módulo `unicodedata` permite converter entre estas formas.

`normalize('NFC', str)` converte *str* para a forma composta.

`normalize('NFD', str)` converte *str* para a forma decomposta.

Para ignorar os acentos vamos usar a normalização NFD:

‘ã’ ⟶ ‘a’ + ‘~’
‘á’ ⟶ ‘a’ + ‘´’
‘Ç’ ⟶ ‘C’ + ‘¸’
⋮

Histograma de ocorrências (2)

```
from typing import *
from unicodedata import normalize

def histograma(fich: str) -> Dict[str,int]:
    ...
    f = open(fich, "r", encoding="utf-8")
    hist = {}
    while True:
        linha = f.readline().lower()
        if linha == "":
            break
        for c in normalize('NFD',linha):
            if c>='a' and c<='z':
                hist[c] = 1+hist.get(c,0)
    f.close()
    return hist
```


Contagem revista

```
{'a': 3296, 'c': 739, 'b': 249, 'e': 3180,
 'd': 1217, 'g': 378, 'f': 240, 'i': 1210,
 'h': 239, 'j': 103, 'm': 1033, 'l': 572,
 'o': 2707, 'n': 1294, 'q': 390, 'p': 531,
 's': 1828, 'r': 1578, 'u': 1042, 't': 1229,
 'v': 425, 'y': 1, 'x': 29, 'z': 68}
```

- Correção das contagens de `'a'`,`'c'`,`'e'`,`'i'` e `'o'`
- A letra mais frequente é o `'a'` e não o `'e'`!

Sumário

- Dicionários permite representar tabelas de associações
- A ordem entre as entradas não é significativa
- Procurar por uma chave é mais fácil e eficiente do que numa lista