# Programação em Python

## Dicionários

2023

Departamento de Ciência de Computadores

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

# Conteúdo

# Dicionários

# Dicionários

- Estrutura de dados para tabelas de associações
- Cada chave é associada a um (e um só) valor
- Analogia: dicionário bilingue (e.g. português-inglês)
- Podemos usar como chave:
  - números
  - cadeias de caracteres
  - tuplos
  - combinações dos anteriores

  (apenas *tipos imutáveis*)
- A ordem dos pares (chave, valor) pode não ser a esperada, já que é estabelecida por algoritmos desenhados com vista ao acesso rápido aos elementos

# 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) ]
```

Problemas:

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

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

  Representa 1, 25 ou 26 bananas?

# Criar um dicionário

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

Podemos consultar valores a partir da chave:

```
>>> inv['bananas']
25
>>> inv['bananas'] = inv['bananas'] + 1
>>> inv
{'laranjas':10, 'peras':12, 'bananas':26}
```

# Criar um dicionário (cont.)

Mais alguns exemplos:

```
>>> emails = { 'Maria João':'mj@mail.pt',
               'João Pedro':'jp@mail.pt' }

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

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

# Algumas operações sobre dicionários

- Obter o valor associado à chave (erro se não existir)

  `dict[chave]`

- Atribuir um valor a uma chave

  `dict[chave] = valor`

- Testar existência de uma chave (resultado: True ou False)

  `chave in dict`

# 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 estão ordenadas!

# 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
```

# Mais operações (cont.)

Percorrer todos os pares de chaves e valores, ordenados por chave:

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

Exemplo:

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

bananas 25
laranjas 10
peras 12
```

# Contar ocorrências

# Contar ocorrências de letras

Problema: contar a ocorrência de cada letra do alfabeto num ficheiro de texto.

Vamos usar um dicionário para associar cada letra à sua contagem.

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

# Um texto de exemplo

O canto I d'*Os Lusíadas*[1].

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

```
def histograma(fich):
    f = open(fich, "r")   # abrir ficheiro para leitura
    count = {}            # inicializa contagem vazia
    while True:           # ler o texto linha-a-linha
        linha = f.readline().lower()
        if linha == '':
            break
        for c in linha:
            if c>='a' and c<='z':
                count[c] = 1+count.get(c,0)
    f.close()
    return count
```

# Contar ocorrências de letras

```
>>> hist = histograma("lusiadas_CantoI.txt")
>>> hist
{'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}
```

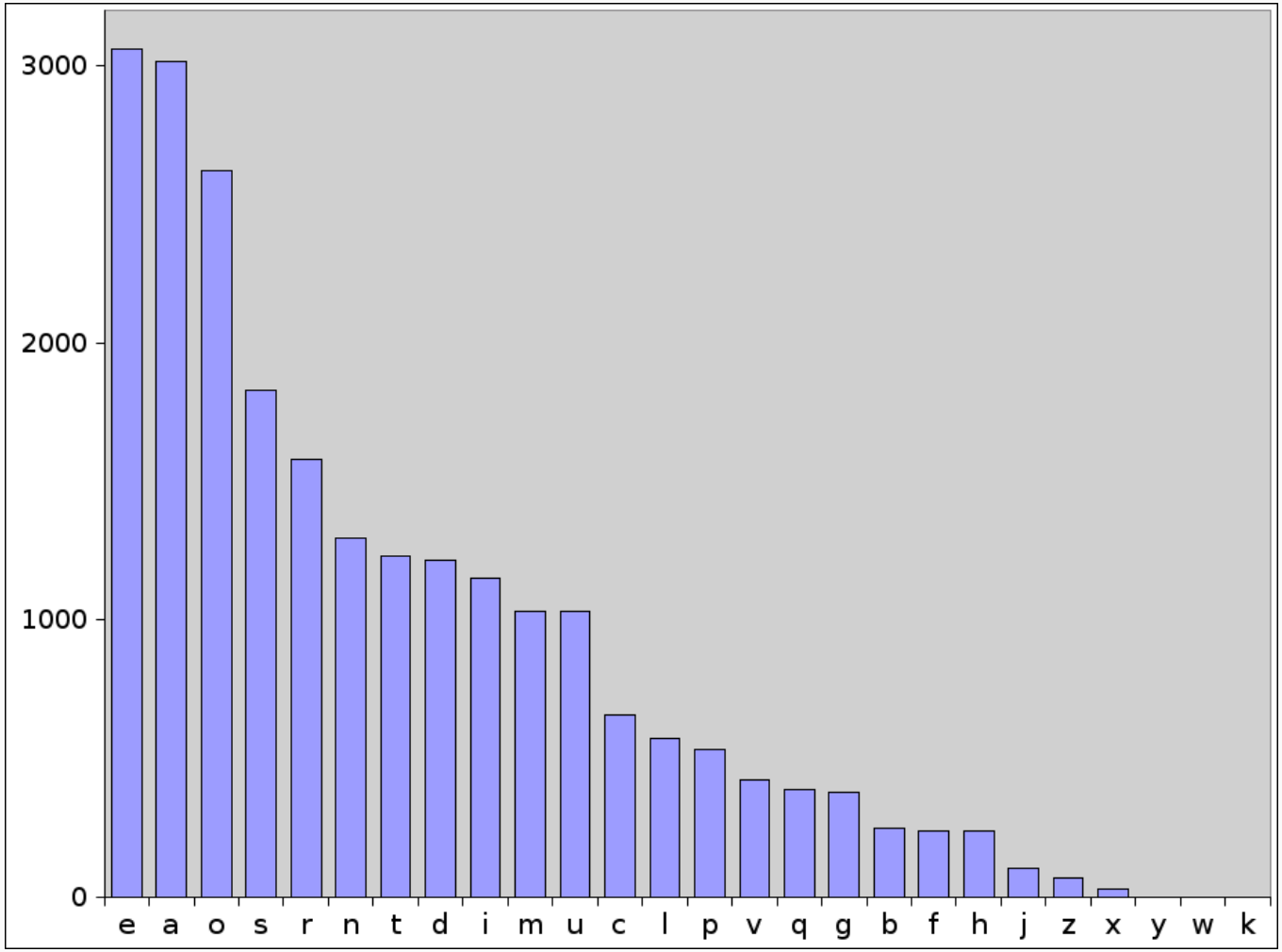
# Visualização

Imprimir uma tabela de ocorrências de letras:

```
hist = histograma("lusiadas_CantoI.txt")
# ciclo sobre as chaves
for c in hist.keys():
    print(c, hist[c])       # letra, contagem
```

Podemos importar para uma folha de cálculo e traçar um gráfico.

# Gráfico do histograma



# Observações

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

# Normalização

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

**NFC** um código numérico composto;
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).

# Normalização (cont.)

O módulo `unicodedata` define uma função `normalize` 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 unicodedata import normalize

def histograma(fich):
    f = open(fich, "r")  # abrir ficheiro para leitura
    count = {}           # inicializa contagem vazia
    while True:          # ler o texto linha-a-linha
        linha = f.readline().lower()
        if linha == '':
            break
        for c in normalize('NFD',linha):
            if c>='a' and c<='z':
                count[c] = 1+count.get(c,0)
    f.close()
    return count
```

# 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}
```

A letra mais frequente é o `'a'` seguido do `'e'`.

# Sumário

- Dicionários permitem representar tabelas de associações
- A ordem entre as entradas não é significativa
- Pesquisa pela chave é mais eficiente do que sobre uma lista (de pares)