

Estruturas de Dados para Bioinformática

Hugo Pacheco

DCC/FCUP

24/25

Ficheiros

Dados

- Até agora, os dados utilizados durante a execução de um programa (input ou gerados) são armazenados em memória volátil, apagada quando o programa termina
- Ficheiros externos (imagens, música, texto, ...):
 - Dados armazenados em suportes persistentes (HDD, pen USB, DVD, ...)
 - **Texto**: dados são caracteres (podem ser visualizados com um editor de texto)
 - **Binário**: dados são armazenados mais eficientemente como 0s e 1s (não vamos lidar diretamente com isso)

Organização de ficheiros

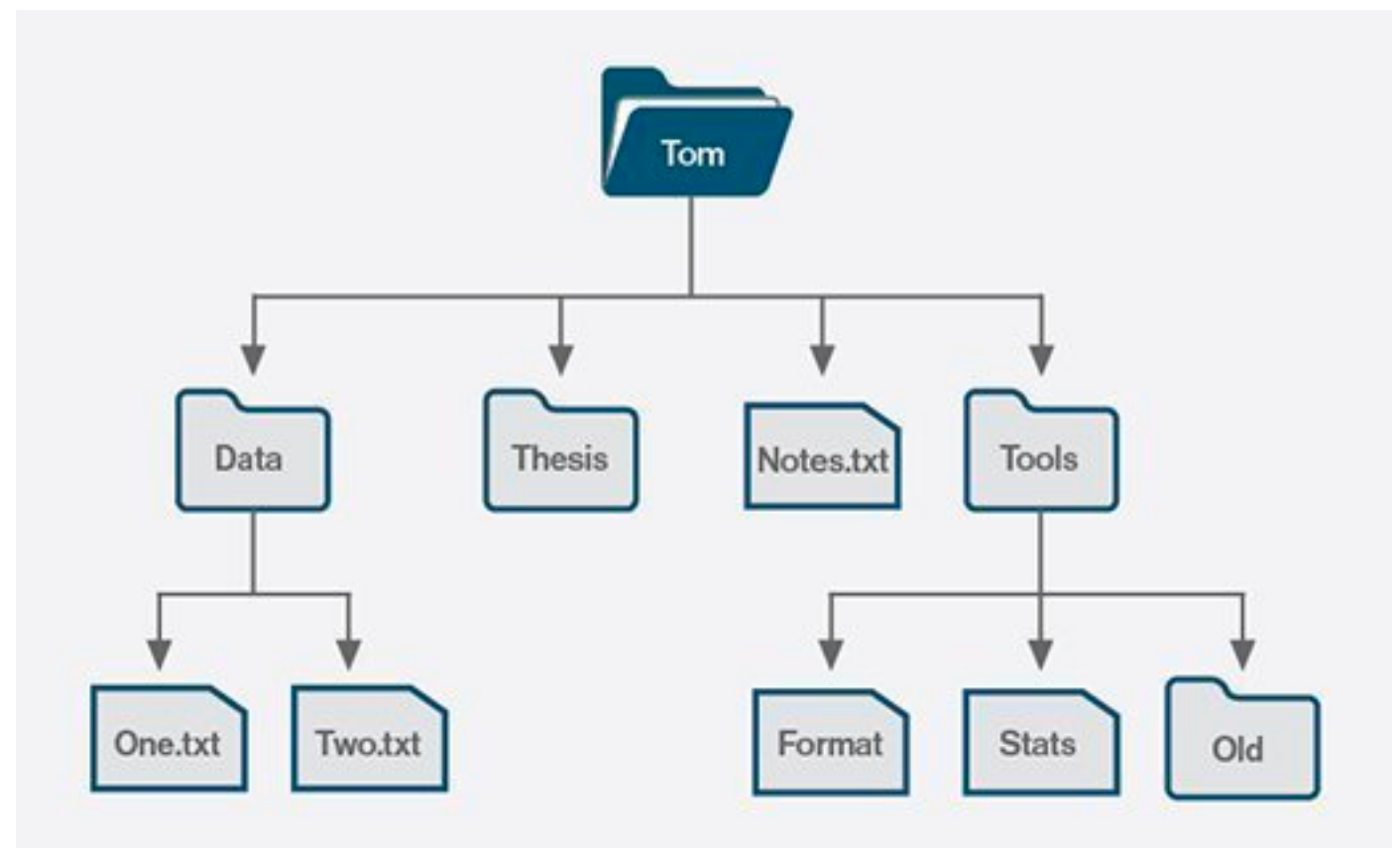
- Identificados por nomes
- Organizados hierarquicamente em pastas

- Linux/Mac

/Tom/Data/One.txt

- Windows

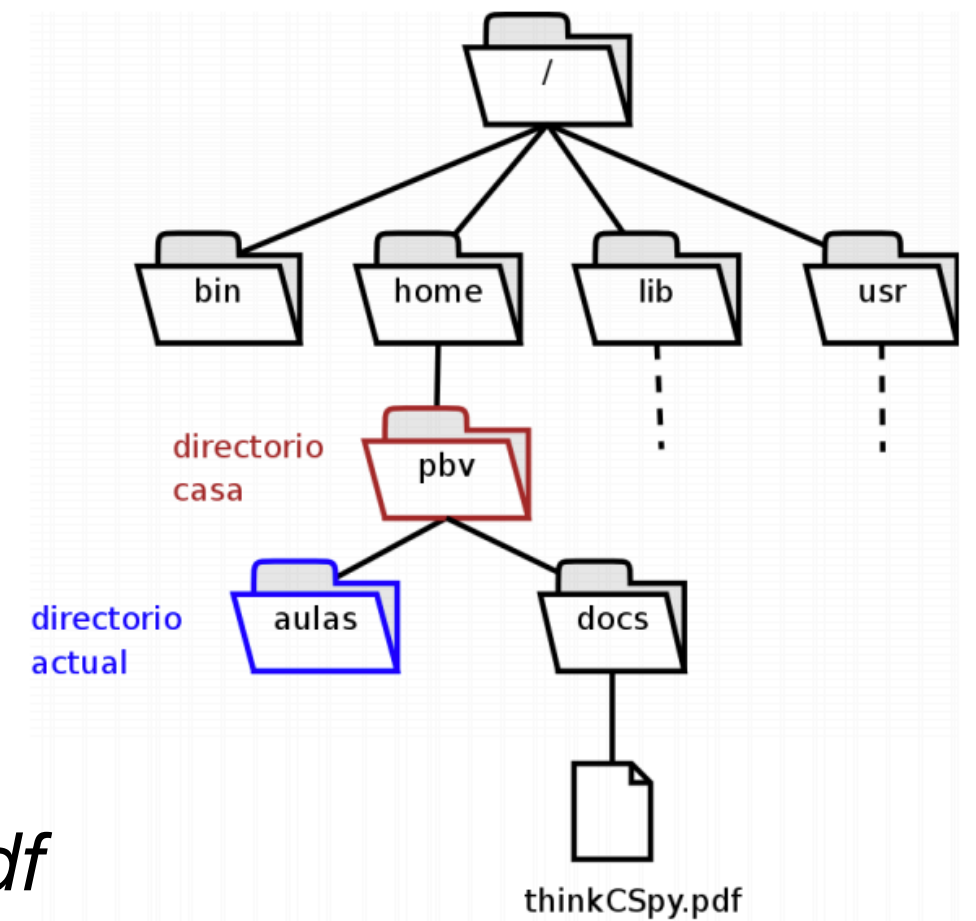
C:\\Tom\\Data\\One.txt



Caminhos no sistema de ficheiros

.	Pasta atual
..	Pasta mãe
~	Pasta home

- Caminhos:
 - */home/pbv/aulas/docs/thinkCSpy.pdf* (absoluto)
 - *../docs/thinkCSpy.pdf* (relativo)
 - *~/docs/thinkCSpy.pdf* (relativo à home)



Navegar no sistema de ficheiros

<i>Alguns comandos UNIX</i>	
<i>ls</i>	Lista ficheiros na pasta atual
<i>pwd</i>	Imprime o caminho atual
<i>cd</i>	Muda a pasta atual
<i>mkdir</i>	Cria uma nova pasta
<i>rmdir</i>	Remove uma pasta vazia
<i>cp</i>	Copia ficheiros
<i>mv</i>	move/renomeia ficheiros
<i>rm</i>	Remove ficheiros
<i>man <comando></i>	Mostra manual de um <comando>

Manipular ficheiros

- Funcionamento como um bloco de notas:
 - abrir/fechar
 - modo leitura/escrita
 - localização do ficheiro:
 - relativa à pasta onde o ficheiro do programa se encontra
 - absoluta (relativa à raiz do sistema de ficheiros)

Abrir/fechar ficheiros

- Abrir um ficheiro cria um *file handle* que permite ler/escrever no ficheiro
- Tem que se fechar o *file handle* no fim
- Padrão (abre o ficheiro e fecha automaticamente no fim):

```
with open(ficheiro, modo) as f:  
    f.operacao  
    ...
```

- Dica: ficheiro =
 - apenas nome (e.g., '*lusiadas.txt*') se na mesma pasta que o programa
 - caminho absoluto/relativo (e.g., '*/home/progii/lusiadas.txt*' ou '*../lusiadas.txt*') se numa pasta diferente

Ler de ficheiros

- É possível controlar como e quantos caracteres se lêem de um ficheiro

```
with open('test.txt', 'r') as f:
    f.read() # lê até ao fim do ficheiro
with open('test.txt', 'r') as f:
    f.readline() # lê uma linha
with open('test.txt', 'r') as f:
    f.readlines() # lê todo como uma lista de linhas
with open('test.txt', 'r') as f:
    f.read(5) # lê 5 caracteres
with open('test.txt', 'r') as f:
    digit = int(f.read(1)) # lê dígito num linhas
    for _ in range(digit):
        print(f.readline()) # lê várias linhas
```

Escrever em ficheiros

- Modo *write* cria novo ficheiro e sobrepõe se existente
- Modo *append* acrescenta conteúdo no fim do ficheiro

```
# cria/apaga ficheiro e escreve string
with open('test.txt', 'w') as f:
    f.write('Hello\n')
```

```
# acrescenta duas strings no fim do ficheiro
with open('test.txt', 'a') as f:
    f.write('Big\n')
    f.write('World\n')
```

```
# acrescenta lista de strings no fim
with open('test.txt', 'a') as f:
    f.writelines(['Really\n', 'Big\n'])
```

Ler e escrever ficheiros

- E.g., ler um ficheiro de um programa python e copiar todas as linhas não comentário para outro ficheiro

```
# abrir 2 ficheiros de uma vez
with open('test.py', "r") as fin\
    , open('test2.py', "w") as fout:
    for line in fin: # readline() implícito
        if not line.startswith('#'):
            fout.write(line)
```

Download de ficheiros web

- Faz download de *Os Lusíadas* de um endereço web e guarda num ficheiro *'lusiadas.txt'* na pasta atual
- Lê, guarda numa lista de linhas e imprime no ecrã

```
import urllib.request

url = 'http://www.gutenberg.org/cache/epub/3333/pg3333.txt'
urllib.request.urlretrieve(url, 'lusiadas.txt')

with open('lusiadas.txt', 'r') as f:
    lines = f.readlines()
for line in lines:
    print(line)
```

Exemplo Os *Lusíadas*

- Extraí estrofes de Os *Lusíadas*

```
# lê ficheiro como lista de linhas sem \n
with open('lusiadas.txt', 'r') as f:
    lines = f.read().splitlines()
```

```
# cria lista de estrofes
estrofes = []
for i, line in enumerate(lines):
    if line.isnumeric():
        estrofes.append(lines[i+1:i+9])
```

```
# imprime estrofes
for estrofe in estrofes:
    for verso in estrofe: print(verso)
    print()
```

Exemplo *Os Lusíadas*

- Vamos verificar se está tudo bem

```
#numero de estrofes
print(len(estrofes))
```

```
#numero de versos
numversos = 0
for estrofe in estrofes:
    numversos += len(estrofe)
print(numversos)
```

```
#numero de versos (ordem superior)
print(sum(map(len, estrofes)))
```

Os Lusíadas é uma obra de [poesia épica](#) do escritor português [Luís Vaz de Camões](#), a primeira [epopeia](#) portuguesa publicada em versão impressa. Provavelmente iniciada em [1556](#) e concluída em [1571](#), foi publicada em Lisboa em [1572](#) no [período literário do Classicismo, ou Renascimento tardio](#), três anos após o regresso do autor do [Oriente](#), via Moçambique.

A obra é composta por dez [cantos](#), 1 102 [estrofes](#) e 8 816 versos em oitavas

[decassilábicas](#), sujeitas ao esquema [rímico](#) fixo AB AB AB CC – oitava rima real, ou camoniana. A ação central é a [descoberta do caminho marítimo para a Índia](#) por [Vasco da Gama](#), à volta da qual se vão evocando outros episódios da [história de Portugal](#), glorificado o [povo português](#).

Exemplo *Os Lusíadas*

- Remover caracteres especiais (não alfabeto nem espaço)

```
# strings
def rem_verso(verso):
    for c in verso:
        if not c.isspace() and not (c.isalpha() or c.isalnum()):
            verso = verso.replace(c, '')
    return verso
```

```
# expressões regulares
import re
def rem_verso2(verso):
    return re.sub(r"(\S|\W)", "", verso)
```

Exemplo *Os Lusíadas*

- Remover caracteres especiais para todos os versos de estrofes
 - \S = non-space, \W = non-word

modificar listas in-place, cópia de strings

```
for estrofe in estrofes:
```

```
    for i, verso in enumerate(estrofe):
```

```
        estrofe[i] = rem_verso(verso)
```

ordem superior

```
rem_versos = lambda versos : list(map(rem_verso, versos))
```

```
estrofes = list(map(rem_versos, estrofes))
```


Exemplo Os *Lusíadas*

- Guarda estrofes num novo ficheiro 'estrofes.txt'

```
# escreve estrofes para um ficheiro
with open('estrofes.txt', 'w') as f:
    for estrofe in estrofes:
        for verso in estrofe:
            # um verso por linha
            f.write(verso+"\n")
        # linha de espaço entre estrofes
        f.write("\n")
```

Exemplo Os *Lusíadas*

- Contar palavras

```
# versão imperativa
npalavras = 0
for estrofe in estrofes:
    for verso in estrofe:
        npalavras += len(verso.split())
```

```
# versão funcional
def sum_verso(verso):
    return len(verso.split())
def sum_estrofe(estrofe):
    return sum(map(sum_verso, estrofe))
npalavras = sum(map(sum_estrofe, estrofes))
```

Exemplo Os *Lusíadas*

- Média de palavras por estrofe

```
import statistics
```

```
# versão imperativa
```

```
npalavras = []
```

```
for estrofe in estrofes:
```

```
    npalavras.append(sum_estrofe(estrofe))
```

```
avg = statistics.mean(npalavras)
```

```
# versão funcional
```

```
avg = statistics.mean(map(sum_estrofe, estrofes))
```

Exemplo *Os Lusíadas*

- Expressões regulares para palavras:
 - começadas por "a", comprimento entre 4 e 6:
"a.{3,5}"
 - acabadas em "ão" ou em "ões":
".*(ão|ões)\$"
 - começadas em "ra" e acabadas em "os":
"^ra.*os\$"
 - que usam uma vogal (sem considerar acentos):
"[^aeiou]*[aeiou][^aeiou]*"
 - que usam no máximo duas vogais (sem considerar acentos):
"[^aeiou]*([aeiou][^aeiou]*){0,2}"

Dicionários e sets

Dicionários

- Dicionários são outro tipo composto para **tabelas de associação**, analogia com dicionário Português->Inglês
- Cada **chave** mapeia num (e só num) **valor**
 - **chaves** têm que ser valores de tipos imutáveis (strings, ints, floats, tuplos ou combinações dos mesmos)
 - **valores** podem ser tipos compostos, e.g., uma palavra em Português associada a uma lista de possíveis traduções em Inglês
- Só estamos interessados na associação, ordem não interessa e pode variar!
- São objetos **mutáveis**, tal como listas
- Algoritmos eficientes para pesquisa, inserção, remoção, etc

Dicionários

- Associar frutas a quantidades
- Porque não apenas listas de pares?

Fruta	Quantidade
bananas	25
peras	10
laranjas	5

```
[ ("bananas", 25), ("peras", 10), ("laranjas", 5) ]
```

- Problemas:
 - algoritmos mais complicados e ineficientes porque ordem conta
 - é necessário percorrer a lista toda
 - lista não garante unicidade de chaves

```
[ ("bananas", 10), ("peras", 10), ("laranjas", 5), ("bananas", 15) ]
```

Dicionários (operações)

```
d = {"bananas":25, "peras":10, "laranjas":5}
```

- Pesquisa: obter o valor associado a uma chave
- Atualização/Inserção: altera o valor associado a uma chave
- Pertença: testa se uma chave existe no dicionário

```
>>> d['bananas']  
25  
>>> d['kiwis']  
KeyError: 'kiwis'
```

```
>>> d['bananas'] = d['bananas'] + 10  
>>> d['kiwis'] = 5  
>>> d  
{ 'bananas': 35, ..., 'kiwis': 5 }
```

```
>>> 'limao' in d  
False  
>>> 'bananas' in d  
True
```


Dicionários (operações)

```
d = {"bananas":25, "peras":10, "laranjas":5}
```

- Valores por omissão

```
>>> d.get("bananas",0)
25
>>> d.get("kiwis",0)
0
```

- Apagar chave

```
>>> del d["peras"]
>>> d
{"bananas":25, "laranjas":5}
```

- Lista de chaves/valores

```
>>> list(d.keys())
['bananas', 'laranjas']
>>> list(d.values())
[25, 5]
>>> list(d.items())
[('bananas', 25), ('laranjas', 5)]
```

- Limpar dicionário

```
>>> d = {}
>>> d.clear()
```

Dicionários (iteração)

```
d = {"bananas":25, "peras":10, "laranjas":5}
```

- Há diferentes formas de percorrer um dicionário

```
# percorrer as chaves (por defeito)
```

```
for k in d:  
    print(k, d[k])
```

```
# percorrer as chaves ordenadas
```

```
for k in sorted(d):  
    print(k, d[k])
```

```
# percorrer pares (chave, valor)
```

```
for k, v in d.items():  
    print(k, v)
```

Exemplo *Os Lusíadas*

- Contar o número de ocorrências de cada letra

```
# lê estrofes de ficheiro
with open('estrofes.txt', 'r') as f:
    texto = f.read()

# número de ocorrências por character
count = {}
for c in texto:
    if c.isalpha():
        count[c] = 1 + count.get(c, 0)
print(count)
```

- Maiúsculas e acentos são caracteres diferentes

```
{ 'A': 1222, 'a': 29815, 'o': 26409, 'õ': 113, ... }
```

Exemplo *Os Lusíadas*

- Normalizar caracteres convertendo em minúsculas e manipulando a representação unicode (package *unicodecode*)
- Contar o número de ocorrências de cada letra outra vez

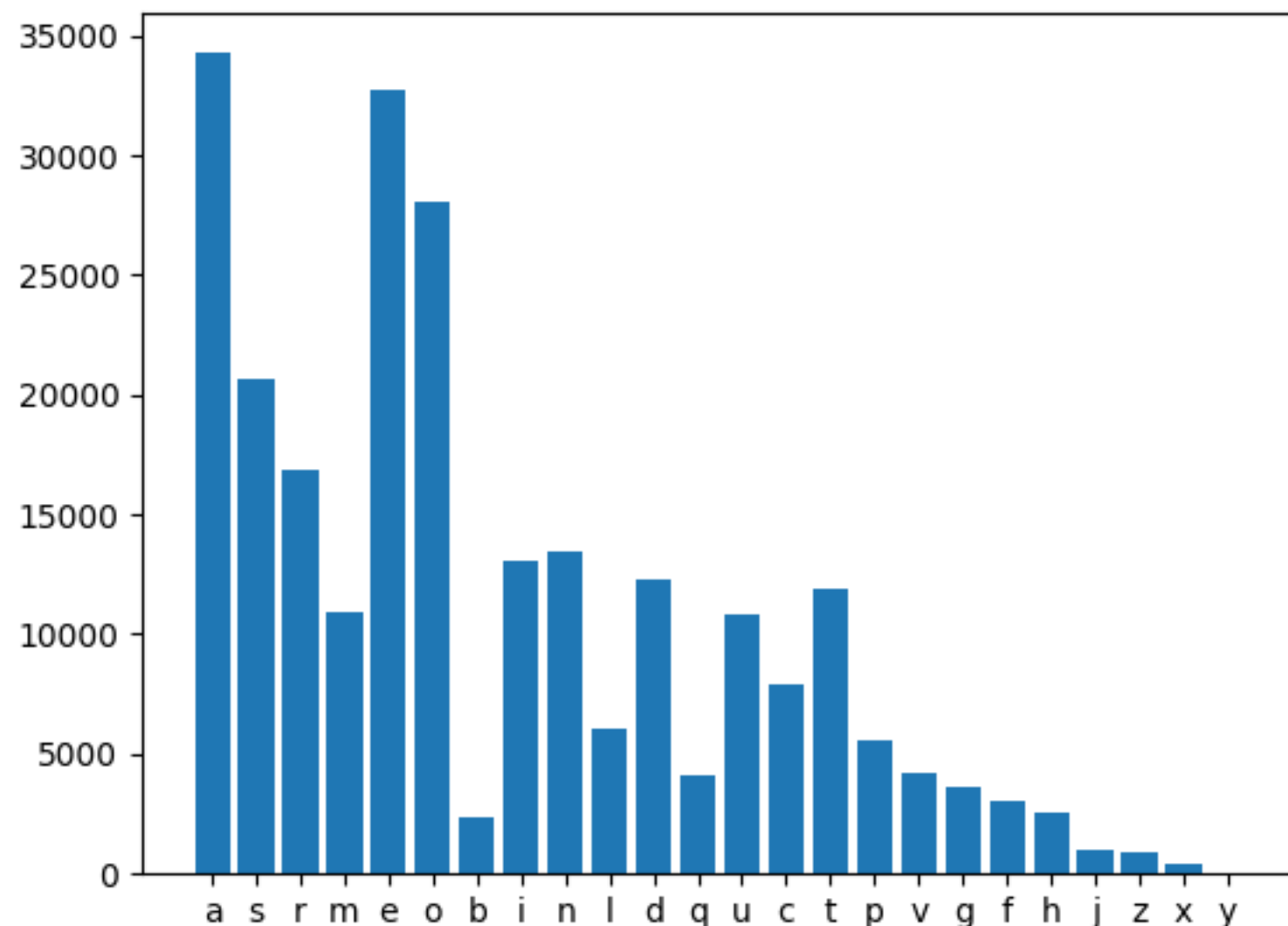
```
import unicodecode as uni
def normaliza(c):
    return (uni.unicodecode(c.lower()))

count = {}
for c in texto:
    if c.isalpha():
        c = normaliza(c)
        count[c] = 1 + count.get(c, 0)
print(count)
```

Exemplo *Os Lusíadas*

- Desenhar um histograma (vamos estudar gráficos mais tarde)

```
import matplotlib.pyplot as plt
plt.bar(count.keys(),
count.values())
plt.show()
```



Conjuntos (sets)

- Sets são outro tipo composto para **conjuntos** tal como na matemática
- **Não** têm ordem **nem** repetidos
- São objetos **mutáveis**, tal como listas e dicionários
- São essencialmente dicionários sem valores
- Algoritmos eficientes para pertença, inserção, remoção, etc

Sets

- Criar sets

```
set() # {} está reservado para dicts  
{1, 2, 3} # set não vazio  
set([1, 2, 4.5]) # conversão de sequência  
set((1, 2, True)) # conversão de sequência  
set("abcd") # conversão de sequência
```

- Elimina repetidos por defeito

```
>>> set([1, 2, 4, 2, 4])  
{1, 2, 4}
```

Sets (operações)

```
s = {'bananas', 'peras', 'laranjas'}
```

- Como não têm ordem, indexação **não** tem significado \Rightarrow erro

```
>>> s['peras']  
TypeError: 'set' object is not subscriptable
```

- Inserção

```
>>> s.add('kiwis')  
{'kiwis', 'bananas', 'peras'}  
>>> s.update(['abacate', 'diospiro'])  
{'diospiro', 'peras', 'bananas', 'abacate', 'kiwis'}
```

- Remoção

```
>>> s.discard('peras')  
{'abacate', 'bananas', 'diospiro', 'kiwis'}  
>>> s.discard('laranja') #ignora se não existir  
{'abacate', 'bananas', 'diospiro', 'kiwis'}  
>>> s.remove('laranja') #erro se não existir  
KeyError: 'laranja'
```


Sets (operações)

- Operações matemáticas de conjuntos
 - União ($S_1 \cup S_2$) $s1 \mid s2$
 - Interseção ($S_1 \cap S_2$) $s1 \& s2$
 - Diferença (S_1 / S_2) $s1 - s2$
 - Pertença ($x \in S$) $x \text{ **in** } s$
 - Subset ($S_1 \subseteq S_2$) $s1.\text{issubset}(s2)$

Dicionários como sets

- É possível definir operações análogas de conjuntos sobre dicionários
- **União:** temos que decidir o que fazer com valores repetidos

```
# função f une 2 valores repetidos  
def uniao(f, d1, d2):  
    d = d1.copy()  
    for k in d2:  
        if k in d: d[k] = f(d[k], d2[k])  
        else: d[k] = d2[k]  
    return d
```

```
# união de dois dicionários somando valores  
>>> d1 = {'a':1, 'b':2, 'c':3}  
>>> d2 = {'a':2, 'c':1, 'd':1}  
>>> uniao(lambda x, y:x+y, d1, d2)  
{ 'a': 3, 'b': 2, 'c': 4, 'd': 1 }
```

Dicionários como sets

- É possível definir operações análogas de conjuntos sobre dicionários
- **Interseção:** temos que decidir o que fazer com valores repetidos

```
# função f une 2 valores repetidos
```

```
def intersecao(f,d1,d2):  
    d = {}  
    for k in d1:  
        if k in d2: d[k] = f(d1[k],d2[k])  
    return d
```

```
# interseção de dois dicionários somando valores
```

```
>>> d1 = {'a':1, 'b':2, 'c':3}  
>>> d2 = {'a':2, 'c':1, 'd':1}  
>>> intersecao(lambda x,y:x+y, d1, d2)  
{ 'a': 3, 'c': 4 }
```

Dicionários como sets

- É possível definir operações análogas de conjuntos sobre dicionários
- **Diferença:**

```
def diferenca (d1, d2) :  
    d = {}  
    for k in d1:  
        if k not in d2: d[k]=d1[k]  
    return d
```

```
>>> d1 = {'a':1, 'b':2, 'c':3}  
>>> d2 = {'a':2, 'c':1, 'd':1}  
>>> diferenca (d1, d2)  
{ 'b': 2 }
```

Exemplo (pangrama)

- Um pangrama é uma palavra que utiliza todas as letras do alfabeto
- E.g., em Português: *“Fidel exporta whiskey, vinho, queijo azul, caju, manga e nabo.”*

```
def pangrama(frase):  
    alfabeto = set("abcdefghijklmnopqrstuvwxyz")  
    chars = set(normaliza(frase))  
    return alfabeto.issubset(chars)
```

Exemplo (heterograma)

- Um heterograma é uma palavra que utiliza cada letra do alfabeto no máximo uma vez
- E.g., em Inglês: *“The big dwarf only jumps.”*

```
def alpha(c): return c.isalpha()
def heterograma(frase):
    chars = list(filter(alpha, normaliza(frase)))
    return len(set(chars)) == len(chars)
```

Exemplo *Os Lusíadas*

- Estudar o vocabulário utilizado em *Os Lusíadas*

```
# lê estrofes de ficheiro
with open('estrofes.txt', 'r') as f:
    texto = f.read()

# set de palavras
vocabulario = set(normaliza(texto).split())
```

Exemplo (DNA → RNA)

- Converter uma sequência de DNA em RNA

```
bases = { 'A': 'U', 'T': 'A', 'C': 'G', 'G': 'C' }  
def base2rna(b): return bases.get(b)  
def dna2rna(dna):  
    return ' '.join(map(base2rna, dna))  
  
print(dna2rna('ATCG'))  
# UAGC
```


Exemplo (Gene Ontology)

- Considerem uma Gene Ontology de exemplo, retirada deste tutorial

- Uma linha por gene

- PZid GO:id nome

PZ7180000020811_DVU	GO:0003824	GJ12748 [Drosophila virilis]
PZ7180000020752_DVU	GO:0003824	GI16375 [Drosophila mojavensis]
PZ7180000034678_DWY	GO:0003824	hypothetical protein YpF1991016_1335 [Ye
PZ7180000024883_EZN	GO:0006548	sjchgc01974 protein
PZ7180000024883_EZN	GO:0004252	sjchgc01974 protein
PZ7180000024883_EZN	GO:0004500	sjchgc01974 protein

- Encontrar gene com maior número de identificadores

```
with open('PZ.annot.txt', 'r') as f:
    linhas = f.read().splitlines()
```

```
genes = {}
for linha in linhas:
    id, go, name = linha.split('\t')
    genes[name] = genes.get(name, set()) | {id, go}
```

```
maxg = max(genes, key=lambda g: len(genes[g]))
print(maxg, genes[maxg])
```