INTRODUÇÃO À PROGRAMAÇÃO

UFCD(s) 5091, 5118

GUIA DE LABORATÓRIO 2.3

FICHEIROS, OUTROS EXEMPLOS E EXERCÍCIOS (ßeta)

OBJECTIVOS

- Alguns exemplos comentados que utilizam a biblioteca padrão e exercícios relacionados

INSTRUÇÕES

Controlo da Execução: Ciclo For

1. Vamos fazer um programa que solicita ao utilizador o nome e exibe esse nome na vertical. Como não sabemos a dimensão do nome, vamos ter que utilizar um ciclo.

Crie o ficheiro `nome_vertical1.py` com o seguinte código:

```
nome = input("Como se chama? ")

i = 0
while i < len(nome):
  print(nome[i])
  i += 1
```

Relembrando, uma ***string*** *é uma sequência de caracteres numerada a partir de 0. A variável i é aqui utilizada para armazenar a posição (ie, o número de ordem) dos caracteres. Se o utilizador inserir* `Armando`*, então:*

```
i == 0 => nome[i] == 'A'
i == 1 => nome[i] == 'R'
...
```

A função ***print*** *automaticamente acrescenta uma nova linha e daí o efeito vertical.*

2. Teste o programa com o nome `Alberto`:

```
Como se chama? Alberto
A
L
B
E
R
T
O
```

Em Python, o ciclo ***for*** *permite aceder a todos os elementos de uma colecção (ou de uma estrutura de dados que possa ser "transformada" numa colecção). Noutras linguagens, este ciclo é designado de foreach ("para cada"), nome que é mais apropriado para a semântica da instrução.*
O formato geral do ***for*** *é:*

```
for var in sequência/colecção:
  instruções_que_utilizam var
```

Ao contrário da linguagem C e derivadas, onde o ciclo for é um ciclo geral, semelhante ao ***while****, e mais utilizado para percorrer gamas de valores em progressão, aqui o* ***for*** *percorre os itens da colecção pela ordem pela qual eles estão dispostos nessa colecção.*

3. Vamos agora utilizar o ciclo ***for***. Crie o ficheiro `nome_vertical2.py` e acrescente o código:

```
nome = input("Como se chama? ")
for car in nome:
  print(car)
```

4. Queremos agora desenvolver um programa para calcular a média de valores passados através da linha

de comandos. O programa deve começar por ler os números para uma lista de valores e depois é que calcula a soma (esta lista de números não é necessária, mas vamos assumir que é importante guardar os valores introduzidos para posterior processamento).
Vamos desenvolver duas versões, uma com ***while***, e outra com ***for***. Noutros laboratórios veremos uma forma mais directa de resolver este problema.

Crie um ficheiro com o nome `media1.py` e comece por acrescentar o seguinte código:

```
import sys

if len(sys.argv) < 3:
    print("Utilização: python3", __file__, "num1 num2 [num3 ... numN]", file=sys.stderr)
    sys.exit(2)

nums = []
i = 1
while i < len(sys.argv):
    # Acrescenta o float na i-ésima posição de argv
    nums.append(float(sys.argv[i]))
    i += 1

print("Números lidos com sucesso.")
```

`sys.argv[0]` é uma alternativa a `__file__`. Enquanto `__file__` indica sempre o nome do script (`media1.py` neste caso), `sys.argv[0]` indica como é que o script foi invocado. Se criarmos uma ligação simbólica (`ln -s` em Unixes) para o script, então este será invocado com um nome diferente do nome do ficheiro.

Em caso de invocação indevida do script, a mensagem de utilização é enviada para a saída de erros padrão (STDERR - `sys.stderr`) e não para a saída padrão (STDOUT - `sys.stdout`), que fica assim reservada para o output "regular" do programa. Voltaremos a este assunto mais em baixo no laboratório.

Além de forçar o fim do programa, a função `sys.exit` permite devolver um código de erro para o SO. Um valor diferente de 0 significa que o script não teve sucesso. Fica ao critério de cada programa definir o significado de cada código de erro > 0. Porém, é comum utilizar o valor 2 para indicar erros de síntaxe na invocação do sript na linha de comandos.

5. Agora acrescente o ciclo que *itera* sobre a lista de números e efectua a soma dos elementos (atenção à indentação: este código deve estar alinhado com o ***print*** e o ***while*** anteriores).

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

6. Finalmente, as instruções que apresentam o resultado:

```
print("Soma:  ", soma)
print("Média: ", soma/len(nums))
```

7. Teste o programa.

8. Vamos agora desenvolver a versão para o ciclo ***for***. Crie o ficheiro `media2.py` e substitua o primeiro ciclo pelo seguinte:

```
for num_txt in sys.argv[1:]:
    nums.append(float(num_txt))
```

Pergunta: porquê `sys.argv[1:]` e não apenas `sys.argv`?

*Este ciclo **for** é, na verdade, desnecessário porque podemos sempre utilizar a built-in **sum** que soma todos os números presentes num objecto iterável, como é o caso de uma lista. Ou seja, em vez do ciclo **for** bastava escrever o seguinte:*

```
soma = sum(nums)
```

9. E agora substitua o segundo ciclo por:

```
for num in nums:
    soma += num
```

Exemplos com Ficheiros

10. Vamos agora fazer um programa que lista o conteúdo de um ficheiro de texto no ecrã. O nome do ficheiro deve ser indicado na linha de comandos. Crie um ficheiro com o nome `mostra.py` e acrescente o seguinte código:

```
import sys

if len(sys.argv) != 2:
    print("Utilização: python3", sys.argv[0],
          "FICH", file=sys.stderr)
    sys.exit(2)
fich = open(sys.argv[1], 'r')
for linha in fich:
    print(linha, end='')
fich.close()
```

*Utilizamos a função built-in **open** para abrir um ficheiro. Esta função possui dois parâmetros: uma indicação do ficheiro a abrir que, normalmente, é uma string com o caminho do ficheiro a abrir, e o parâmetro que indica o modo de abertura. Vários modos são possíveis (consulte a documentação oficial do Python), mas os mais comuns são: **'r'** para abrir em modo de leitura, **'w'** para abrir em modo de escrita, **'+'** para escrita e leitura e **'x'** para escrita exclusiva. Se acrescentarmos **'b'** ao modo (eg, **'rb'**), o ficheiro é aberto em modo binário.*

*O Python distingue dois tipos de ficheiro: ficheiros de texto e ficheiros binários. De ficheiros abertos em modo binário (com **'b'** no modo) obtemos bytes sem qualquer transformação. Em modo texto (por omissão, ou se adicionarmos **'t'** ao modo), o conteúdo do ficheiro é devolvido como objectos do tipo **str**, o que significa, em Python 3, UTF-8. Além disso, pelo facto de ser um ficheiro de texto, alguns caracteres podem ser transformados no processo de leitura, nomeadamente, o(s) caractere(s) de fim de linha. Em Python 3, o fim de linha é sempre indicado pelo caractere **'\n'**, mas em Windows um fim de linha é dado pela sequência **'\r\n'**. Neste último caso, o Python tem que converter a indicação de fim de linha para a sua representação interna.*
Se a operação de abertura for bem sucedida (pode não ser, se o ficheiro em questão não existir, se não tivermos premissão para abrir o ficheiro no modo pretendido, etc.), open devolve um objecto designado por "file object". O tipo concreto do objecto varia (ver documentação), mas devemos contar que ficheiros de texto podem ser lidos caractere-a-caractere ou linha-a-linha, ao passo que ficheiros binário são lidos em blocos de bytes. Do ponto de vista do Python, um "file object" pode ser encarado como uma sequência de linhas e daí podermos utilizar um ciclo for para ler o ficheiro linha-a-linha. Um "file object" é objecto que também é designado por stream. Uma stream é como que um canal de comunicação por onde circula um fluxo de bytes ou caracteres entre origem e destino. Origem e destino podem ser outros dois quaisquer objectos ou o próprio programa.
De notar que podemos sempre abrir um ficheiro de texto em modo binário, mas não devemos abrir um ficheiro binário (eg, uma imagem) em modo texto porque, lá está, o conteúdo pode ser transformado uma vez que o Python "pensa" que está a lidar com caracteres.

Consultar:
https://docs.python.org/3/library/functions.html#open https://docs.python.org/3/library/io.html#module-io
https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

11. Vamos fazer um programa que é uma mistura dos comandos `cat` e `cp` da linha de comandos do Unix. Este programa copia o conteúdo de um ficheiro (`fich1`) para outro (`fich2`), sendo ambos os ficheiros indicados na linha de comandos. Se `fich2` não for indicado, o conteúdo de `fich1` é exibido na saída padrão (`sys.stdout`). Se `fich1` também não for indicado, o conteúdo é lido a partir da entrada padrão e exibido na saída padrão. Nesta versão o programa aceita apenas ficheiros de texto. Além disso, não lidamos com potenciais situações de erro (ficheiros não existentes, leituras/escritas inválida, etc.).

Crie o ficheiro `copia1.py` com:

```
import sys

if not 1 <= len(sys.argv) <= 3:
    print("Utilização: python3", sys.argv[0], "[FICH] | [FICH1 FICH2]")
    sys.exit(2)

if len(sys.argv) >= 2:
    fich1 = open(sys.argv[1], 'r')
else:
    fich1 = sys.stdin

if len(sys.argv) == 3:
    fich2 = open(sys.argv[2], 'w')
else:
    fich2 = sys.stdout

for line in fich1:
    fich2.write(line)

fich1.close()
fich2.close()
```

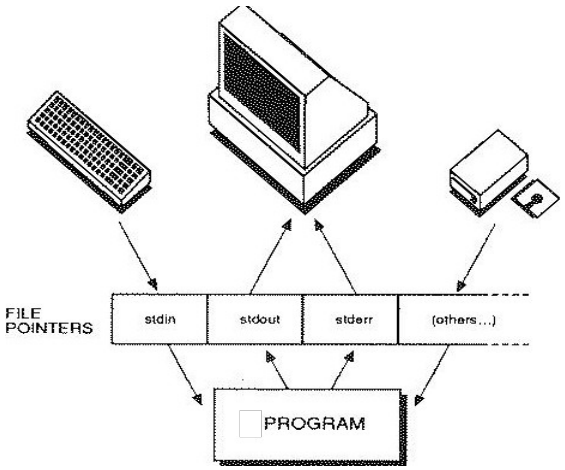
Tal como no exemplo anterior, antes do programa terminar devemos fechar os ficheiros através da função close que é suportada pelos objectos do tipo ficheiro (`fich1` e `fich2`). No laboratório seguinte iremos lidar com outras formas de conseguir isto, mais robustas (eg, neste caso se `fich1` for aberto mas a abertura de `fich2` falhar, o programa aborta com `fich1` por fechar).
Note-se a utilização da expressão de intervalo:

```
1 <= len(sys.argv) <= 3
```

Esta expressão é idêntica a:

```
len(sys.argv) >= 1 and len(sys.argv) <= 3
```

`sys.stdin` e `sys.stdout` representam duas streams muito importantes ao nível do sistema operativo: a entrada padrão e a saída padrão. Normalmente, `sys.stdin` está associada ao teclado e `sys.stdout` ao ecrã.



12. As operações de abertura de ambos os ficheiros podem também ser escritas de forma mais legível utilizando o operador condicional ***if-else*** (que vamos abordar no próximo laboratório):

```
fich1 = open(sys.argv[1], 'r') if len(sys.argv) >= 2 else sys.stdin
fich2 = open(sys.argv[2], 'w') if len(sys.argv) == 3 else sys.stdout
```

13. Vamos agora fazer uma nova versão deste programa que suporta ficheiros binários, como ficheiros áudio e imagens. Começamos por indicar algumas alterações:

- Não podemos processar ficheiros binários linha-a-linha uma vez que o conceito de linha não existe num ficheiro binário.
- Sendo assim, vamos utilizar o método read dos *"file objects*", método que tem um parâmetro para receber quantos bytes pretendemos ler de cada vez. Este método é inserido num ciclo ***while***.
- A entrada e saída padrão são sempre "canais" orientados ao caractere. Sendo assim, sempre que `fich2` não for especificado (o que significa que o conteúdo de *fich1* é para ser passado para `sys.stdout`), então abrimos `fich1` em modo texto - modo `'r'`. Caso contrário, `fich1` e `fich2` são abertos com `'rb'` e `'wb'`.
- Ficheiros de texto continuam a ser suportados pois estes também são ficheiros binários.

Crie o ficheiro `copia2.py` com o seguinte conteúdo:

```
import sys

if not 1 <= len(sys.argv) <= 3:
    print("Utilização: python3", sys.argv[0], "[FICH] | [FICH1 FICH2]")
    sys.exit(2)

if len(sys.argv) == 3:
    # Não utilizamos saída padrão, então abrimos em modo binário
    fich1 = open(sys.argv[1], 'rb')
    fich2 = open(sys.argv[2], 'wb')
else:
    # Saída padrão envolvida, é melhor abrir em modo texto
    fich1 = open(sys.argv[1], 'r') if len(sys.argv) == 2 else sys.stdin
    fich2 = sys.stdout

dados = fich1.read(1)
while dados:
    fich2.write(dados)
    dados = fich1.read(1)

fich1.close()
fich2.close()
```

*O método read permite ler N bytes ou caracteres consoante a stream a que está associado for binária ou de texto. Com streams binárias, o método devolve um objecto do tipo **bytes**. O tipo de dados **bytes** é uma sequência de bytes, isto é, de octetos (8 bits). É similar a **str**, e até podemos delimitar o seu conteúdo entre plicas ou aspas prefixadas por pela letra **'b'**. Este tipo de dados pode ser utilizado para representar strings em ASCII (e derivados), mas não é o mais adequado para representar texto em Unicode pois aqui pode ser necessário mais do que um byte para representar um caractere. (...)*

14. Se tentar copiar um ficheiro binário com alguns MB irá verificar que a operação é muito demorada. O problema tem a ver com o facto de lermos 1 byte de cada vez. Também não queremos ler o ficheiro todo de uma vez porque isso pode ocupar muita memória. Sendo assim, vamos ler blocos de 8KB de cada vez. Acrescente a seguinte definição logo a seguir ao ***import***:

*(...) Eis três elementos do tipo **bytes** idênticos:*
`bytes((97, 48, 10))` <=> `b'a0\n'` <=> `b'\x61\x30\xA'`
Quando já não há mais bytes a ler, porque chegámos ao fim do ficheiro, o método read devolve `b''` (sequência vazia). Podemos não dar argumentos a `read` e neste caso o método lê o ficheiro todo de uma vez.

```
DIM_BLOCO = 8192   # 8KB (de cada vez)
```

15. Agora modifique o ciclo de leitura/escrita para:

```
dados = fich1.read(DIM_BLOCO)
while dados:
    fich2.write(dados)
    dados = fich1.read(DIM_BLOCO)
```

WITH e Ficheiros

16. Vamos desenvolver um exemplo mais simples do que o anterior, uma vez que este será um clone do `cp` apenas, necessita sempre de dois ficheiros e não contempla a utilização de STDIN nem de STDOUT.

Vamos utilizar a instrução ***with***. Esta instrução é apropriada sempre que pretendemos adquirir um recurso como um ficheiro e queremos ter a certeza que esse recurso é libertado. Crie o script `my_cp.py` e acrescente o seguinte código:

*A instrução **with** é apropriada sempre que pretendemos adquirir um recurso (como um ficheiro) e queremos ter a certeza que esse recurso é libertado, quer as operações de manipulação do recurso tenham sido executadas com sucesso, quer tenha ocorrido um erro grave que impede que esse recurso seja utilizado e que pode levar ao encerramento do script. Considere o seguinte exemplo:*

```
fich = open(caminho, "w")
# ... muitas operações com o ficheiro ...
# operação que leva a um erro irrecuperável!!
# ... mais operações, só que estas não são executadas devido ao erro irrecuperável ...
fich.close()   # ... esta operação tb não vai ser executada; o ficheiro não vai ser fechado ...
```

*Se entre **open** e `file.close()` ocorrer um erro irrecuperável esta última operação não é executada. O recurso principal (o ficheiro) e todos os recursos associados (eg, buffers de memória) não são devolvidos ao sistema. Temos uma situação de "leakage", isto é, de "fuga" de recursos. A solução passa por utilizar a instrução **with** para adquirir acesso ao ficheiro:*

```
with open(caminho, "w") as fich:
     # ... muitas operações com o ficheiro ...
     # operação que leva a um erro irrecuperável!!
    # ... mais operações que não vão ser executadas mas o close não está entre elas ..
```

*A instrução **with** garante `fich.close()` é sempre invocado, tenha ou não tenha ocorrido um erro no bloco de instruções **with**. Abordaremos erros e excepções noutro laboratório mais à frente, e aí voltaremos à instrução **with**.*

Note-se que a instrução ***with*** *é apropriada para qualquer tipo de recurso, e não só para ficheiros. Também é indicada quando queremos temporariamente modificar um parâmetro global do programa ou do ambiente, repondo o seu valor inicial no final do bloco de instruções do* ***with****.*

```
import sys

DIM_BLOCO = 8192

if len(sys.argv) != 3:
    print("Utilização: [python3]", sys.argv[0], "FICH1 FICH2", file=sys.stderr)
    sys.exit(2)

with open(sys.argv[1], 'rb') as src_file:
    with open(sys.argv[2], 'wb') as dest_file:
        data = src_file.read(DIM_BLOCO)
        while data:
            dest_file.write(data)
            data = src_file.read(DIM_BLOCO)
```

Comunicação por HTTP: Obter Informação na Web

17. Vamos agora desenvolver um pequeno script para extrairmos alguma informação da Web. Antes de mais, vamos instalar a biblioteca `requests` que nos permite comunicar por http/s. Na linha de comandos do sistema operativo faça:

```
$ pip3 install requests
```

A biblioteca padrão do Python disponibiliza a biblioteca `urllib.request` *(todo o pacote* `urllib` *contém vários utilitários para trabalhar com http) para que possamos desenvolver um cliente http/s. Porém, esta biblioteca é algo complicada e por isso vamos utilizar a biblioteca* `requests`*, que é bem mais simples e popular mas que, no entanto, não faz parte da biblioteca padrão.*
O exemplo que vamos desenvolver vai utilizar a pesquisa instantânea do motor de busca DuckDuckGo para extrair informação acerca de uma determinada expressão. Os resultados vêm no formato JSON (JavaScript Object Notation), um formato que é parecido com o que é utilizado para representar dicionários em Python. Esta primeira versão vai ser algo "primária": envia o pedido para o DuckDuckGo e exibe a resposta sem efectuar qualquer tipo de formatação ou filtragem.
O exemplo que vamos desenvolver deve ser utilizado assim:

```
$ python3 obtem_info1.py European Union
```

O programa pesquisa pela expressão "European Union" e depois exibe a resposta dada pelo motor de busca.

Consultar:
https://docs.python.org/3/library/urllib.request.html http://docs.python-requests.org/en/latest/ https://api.duckduckgo.com/api

18. Agora, crie o ficheiro `obtem_info1.py`:

```
import sys
import requests

if len(sys.argv) == 1:
    print("Utilização: python3", __file__, "expressao")
else:
    expr = ' '.join(sys.argv[1:])
    # A linha seguinte é apenas para testes
    print("Expressão a procurar:", expr)
```

O método join é uma operação que está disponível para strings. Recebe uma sequência de strings e junta-as numa nova string separadas pela string antes do . (ponto). Exemplos:

```
>>> ','.join(["abc", "def", "ghi"])
'abc,def,ghi'
>>> ' -- '.join(('ana', 'zé', 'paulo'))
'ana -- zé -- paulo'
>>> '.'.join("Alberto")
'A.l.b.e.r.t.o'
```

19. Teste o script.

20. Agora remova o ***print*** anterior e acrescente o seguinte código:

```
resp = requests.get(
    'http://api.duckduckgo.com/',
    params={'q': expr, 'format': 'json'}
)
# Linha de teste
print(resp.json())
```

A biblioteca `requests` possui o método `get` que permite enviar um pedido GET, um dos pedidos (ou métodos) do protocolo HTTP. Este método recebe o URL e um dicionário de parâmetros. Quando concatenados ao URL estes parâmetros formam o que se chama a 'query string' do URL. Admitindo que o utilizador pretende procurar por `"European Union"`, o URL de pesquisa enviado para o DuckDuckGo é:
http://api.duckduckgo.com/?q=European+Union&format=json
O método `get` devolve um objecto do tipo `Response`, com várias operações associadas. O método `json` devolve a informação no formato com o mesmo nome.

21. O módulo *pprint* possui alguns utilitários para formatar as estruturas de dados do Python. Acrescente no início o seguinte:

```
import pprint
```

22. E agora substitua o último ***print*** por:

```
pprint.pprint(resp.json())
```

A designação de `pprint` é a abreviatura de "Pretty Print". De notar que o módulo `pprint` tem uma função que também se chama `pprint`. Daí a invocação `pprint.pprint`.
Consultar:
https://docs.python.org/3/library/pprint.html?#module-pprint

EXERCÍCIOS DE REVISÃO

1. O que é uma função?

2. Qual a importância da variável global *__name__*?

3. Obtenha informação sobre os seguintes tópicos relacionados com Python: PyPy, Cython e PyCon.

4. Obtenha informação geral sobre os seguintes módulos da biblioteca padrão: `dict`, `dict.setdefault`, `heapq`, `re`, `calendar`, `random`, `os.path`, `os.path.join`, `os.walk`, `csv`, `zipfile`, `tarfile` e `hashlib`.

5. Considerando que inicialmente `txt = "Bom dia, Alberto!"` , responda às seguintes questões.

5.1 `txt[3]` = ____

5.2 `txt[3:]` = ____

5.3 `txt[-len(txt)]` = ____

5.4 `txt[-len(txt) + 4]` = ____

5.5 `txt[4:-5]` = ____

6. Os seguintes programas ou fragmentos de programas apresentam alguns erros. Corrija-os:

`vals = [1 2 3 4]`
`for val em vals:`
`  print(val)`	

`if __name__ = __main__:` `  print("Bem vindo")`	

`def formula(x, y):`
`  z = 2*x + y/3`
`def mostraResultado(x, y):`
`  print(formula x, y)`	

`import Decimal from decimal` `x = decimal['0.1']`	

7. Considere o seguinte fragmento de código que declara e atribui valores a duas variáveis do tipo ***int*** e ***float***:

```
x = 10.18
y = 41
```

Para cada instrução (`str.`) `format`/ f-string na coluna da esquerda, preencha a grelha correspondente na coluna da direita. Uma quadrícula vazia entre caracteres indica a presença de um espaço. Assuma que todas as instruções estão envolvidas num ***print***.

`f"{y}{x-10}"`																				

`f"{y:5}\n"` `"{1:>6}{0:<6}".format(x, x+2)`																				

8. O que é exibido pelas seguinte instruções (se executadas através de um *script*):

`print('\n'.join("Alberto"))`	
`print('\n\t'.join(("abc", "def", "ghi")))`	

`DIM = 7`
`v = [10, 90, 8, 17, 16, 10, 4]`
`i = 1`
`v[i] = 19; i += 1; v[i] = 10; i+=3; v[i] = 6`
`print(v[v[DIM*2-10] - v[2] - 3] + v[i*2-4])`	

`v = [8, 9, 8, 1, 1, 8]`
`DIM = len(v)`
`i = 1`
`v[DIM-i] = 14; i-=1; v[i] = 10; i+=2; v[DIM-2*i] = 6`
`print(v[i] + v[5]*2 + v[v[0]-v[DIM-1]+5])`	

EXERCÍCIOS DE PROGRAMAÇÃO

9. Faça um programa para ler texto, caractere a caractere, até que o utilizador introduza o caractere . (ponto). No final deverá indicar quantos dígitos e espaços o utilizador inseriu.

10. Faça um programa para indicar quantos nomes masculinos e femininos estão presentes numa base de dados em formato texto passada na invocação do programa através da linha de comandos:

```
$ python3 conta_nomes.py FICH_NOMES
```

`FICH_NOMES` deverá conter um nome completo por linha. Note-se que o nome completo poderá possuir vários nomes pessoais e de família. A determinação do género do nome deverá ser feita pelo primeiro nome.

Para determinar se os nomes em `FICH_NOMES` são masculinos ou femininos, o programa utiliza uma outra base de dados em formato txt. Nesta base de dados - ficheiro `nomes.txt` -, os nomes femininos começam após uma linha que contém apenas o texto `"FEMININOS:"`. De seguida surgem os nomes femininos, um por linha. Igual lógica se aplica aos nomes `"MASCULINOS:"`.

11. Faça um *script* que recebe pela linha de comandos o caminho para um ficheiro de texto e exibe (na saída padrão) o texto com as letras maiúsculas convertidas em minúsculas.

12. Refaça o *script* anterior de modo a que possa ser invocado assim:

```
$ python3 <nome_do_script> [-if input_file] [-of output_file]
```

Para ler os parâmetros, utilize o módulo `argparse` que faz parte da biblioteca padrão. Ambos os parâmetros são opcionais, e se algum não for especificado, deverá utilizar a entrada ou saída padrão.

13. Por vezes é útil mudar o endereço MAC de um adaptador de rede (eg, para contornar um *switch* que bloqueia portas a um determinado MAC). Utilizando as funcionalidades do módulo random tente fazer um programa que gera um MAC address válido de forma aleatória. O formato do MAC gerado deve ser semelhante ao do seguinte exemplo: `97:7C:5E:27:18:9E`.

14. Pretende fazer um programa em Python para gerar aleatoriamente grupos de alunos para a realização de projectos e trabalhos de grupo. O seu *script* deve processar um ficheiro de entrada com os nomes dos alunos, um por linha, e gerar um ficheiro de saída para onde deve enviar a listagem dos grupos, um grupo por linha com os nomes separados por vírgulas. Deve também receber o número de elementos a agrupar.

O *script* deve ser invocado de acordo a com a seguinte sintaxe:

```
$ python3 agrupa.py FICHEIRO_ENTRADA FICHEIRO_SAÍDA NUM
```

Exemplo:

Ficheiro de entrada: "nomes.txt"	**$ python3 agrupa.py nomes.txt grupos.txt 3**
Alberto Antunes Arnaldo Alves António Almeida Catarina Costa Carlos Catarino Constança Cunha Diogo Diniz Duarte Damásio Eduardo Esteves Ernesto Estrela	**(possível conteúdo de "grupos.txt")** Alberto Antunes, Eduardo Esteves, Catarina Costa Constança Cunha, António Almeida, Carlos Catarino Ernesto Estrela, Diogo Diniz, Arnaldo Alves Duarte Damásio

15. Utilizando o módulo `argparse`, adapte o programa anterior à seguinte invocação:

```
$ python3 agrupa.py [-in FICH_ENTRADA] [-out FICH_SAÍDA] [-n NUM]
```

Os parâmetros de entrada são todos opcionais. Os valores por omissão são a `sys.stdin`, `sys.stdout` e `2`, respectivamente.

16. Instale a biblioteca `docopt` (utilize o comando `pip3`) e repita o exercício anterior mas utilizando esta biblioteca. Tente obter a sintaxe mais aproximada que conseguir.

17. Vamos agora fazer um programa para detectar ficheiros idênticos, isto é, ficheiros cujo *conteúdo* é igual. Através da linha de comandos, o seu programa recebe um caminho para uma directoria e no final deve exibir uma listagem com todos os ficheiros duplicados dentro dessa directoria, inclusive dentro das sub-directorias.

Sugestões:

1. Utilize `os.walk` para percorrer árvore de directorias dentro do caminho fornecido

2. Para evitar ter que comparar ficheiros, processo dispendioso em termos de CPU e de utilização de memória, utilize `hashlib.md5` para obter de forma eficiente um resumo de cada ficheiro. Dois ficheiros diferentes originam, com muito elevada probabilidade, um resumo diferente.

3. Depois deve pensar na estrutura de dados adequada para organizar estes resumos e, através dela, obter todos os ficheiros que produzem o mesmo resumo, isto é, os duplicados.

18. Pretende fazer um clone do utilitário `wc` presente na maioria dos sistemas baseados em Unix, e que, dados um ou mais ficheiros, permite obter uma contagem de bytes, de caracteres, de palavras e de linhas para cada ficheiro. A sintaxe de invocação do `wc.py` deverá ser semelhante à do próprio `wc`:

```
$ python3 wc.py [-clmw] [FILE]...
```

O saída do seu programa deverá ser idêntica à do `wc`. Consulte `man wc`. Note que o `wc` pode ser invocado sem indicação de ficheiros, caso em que calcula os contadores pedidos na linha de comandos a partir dos dados introduzidos através da entrada padrão. Quando não indicamos nenhuma opção, o `wc` exibe três contadores: bytes, palavras e linhas. Neste mesmo cenário, o script que vamos desenvolver deve exibir quatro contadores.

19. Utilize a biblioteca requests e a API Web do DuckDuckGo (DDG) para pesquisar por definições de palavras em inglês. O seu script deve receber um conjunto de palavras a partir da entrada padrão e exibir o texto obtido por cada tópico relacionado que é devolvido pela API do DDG. Esta informação deve ser exibida na saída padrão com o seguinte formato:

```
$ python3 find_defs.py car computer
Car      : Car A wheeled motor vehicle used for
         : transportation.
         : Central African Republic A landlocked country in
         : Central Africa.
         : Cars (film) A 2006 American computer-animated
         : comedy-adventure film produced by Pixar Animation
         : Studios and...

Computer : Computer A device that can be instructed to carry
         : out arbitrary sequences of arithmetic or
         : logical...
```

```
: Personal computer A multi-purpose computer whose
: size, capabilities, and price make it feasible for
: individual use.
: OK Computer The third studio album by English
: alternative rock band Radiohead, released in 1997
: on EMI...
: Computer (magazine) An IEEE Computer Society
: practitioner-oriented magazine issued to all
: members of the society.
: Human computer The term "computer", in use from
: the early 17th century, meant "one who computes":
: a person...
```

Cada linha (sem contar com a indentação do lado esquerdo) deverá ocupar, no máximo, 50 caracteres. O seu script deverá suportar a opção `-f FILE` que permite obter as palavras a partir de um ficheiro de texto contendo uma palavra por linha. Neste caso, as palavras passadas na linha de comandos são ignoradas. Também deve suportar as opções `-H`, para gerar uma tabela HTML, e `-W`, semelhante a `-H`, mas para gerar um documento HTML completo. O documento HTML gerado com a opção `-W` deve incluir uma folha de estilos CSS interna.

A análise das opções deve ser feita por inspecção directa de `sys.argv`. Dito de outra forma, não deve utilizar bibliotecas como `argparse` ou `docopt`. As opções devem preceder as palavras na linha de comandos e devem poder ser inseridas como é habitual em sistemas Unix; exemplos: `-W -H -f xpto`, `-WH -f xpto`, `-f xpto -HW`, `-HWf xpto`, etc.

20. Pretende fazer uma variação do comando `tr` (consultar `man tr`). Na sua forma mais geral, este clone, `tr.py`, recebe uma lista de padrões - `SET1` - e uma lista de transformações - `SET2` -, delimitadas em ambos os casos por `:` , e aplica-as por ordem. Um padrão é uma expressão regular aceite pelas funções do módulo `re`. Em alternativa pode utilizar o mais completo módulo `regex`, que não faz parte da biblioteca padrão. Uma transformação é uma sequência de uma ou mais palavras delimitada por `:` . Vejamos os seguinte exemplos:

```
$ python3 tr.py alberto:armando:augusto:casa "Alberto:Armando:Augusto:à casota"
o augusto e o alberto foram a casa do armando
o Augusto e o Alberto foram à casota do Armando

$ python3 tr.py promenor:pograma:verdeira:assegorar pormenor:programa:verdadeira:assegurar
Um promenor que escapou ao pograma foi a verdeira tentativa de assegorar a...
Um pormenor que escapou ao programa foi a verdadeira tentativa de assegurar a...

$ python3 tr.py ana:maria Ana:Maria
A ana é amiga da mariana e da maria
A Ana é amiga da mariAna e da Maria

$ python3 tr.py '\bana\b:maria' Ana:Maria
A ana é amiga da mariana e da maria
A Ana é amiga da Mariana e da Maria

$ python3 tr.py '[0-9]{2}:Alberta Silva' 'DD:Alberta Torres'
```

```
Em 99 encontrei 12 vezes a Alberta Silva
Em DD encontrei DD vezes a Alberta Torres
```

```
$ python3 tr.py '[a-c]{2}[0-9]{2}:Albert[oa].+va' '-:!'
ab ab20 20 Alberto Antunes Silva
ab - 20 !
```

```
$ python3 tr.py '([a-c]{2})([0-9]{2}):Albert([oa])(\s+)Victor' '\2\1:\1 Albert\1\2Vitor'
ab Alberto    Victor ab20 20 AlbertoVictor
ab o Alberto    Victor 20ab 20 AlbertoVictor
```

À semelhança do `tr`, o seu programa deve suportar a opção `-d` para remover instâncias dos padrões indicados em `SET1`. A opção `-d` invalida a utilização de `SET2`.

21. Pretende-se que implemente um codificador e descodificador do tipo *Run-Length Encoding* (RLE), que podemos traduzir para algo como *"Codificador de Comprimento de Séries"*. O RLE é muito utilizado para comprimir blocos repetidos de informação definidos consecutivamente. Designamos estes blocos por "séries". O seu codificador deverá codificar séries de bytes, porém vamos ilustrar a sua aplicação na codificação de caracteres. Considere a seguinte sequência de texto (tirada da Wikipedia):

```
WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW
```

Existem várias codificações RLE possíveis para esta sequência. A pretendida neste exercício é a seguinte:

```
WW12BWW12BB3WW24BWW14
```

Apenas uma ocorrência de um caractere leva a que este seja passado para a saída inalterado. Uma série de duas ou mais ocorrências do mesmo caractere é codificada com a dimensão precedida de dupla ocorrência do caractere. Ou seja, o caractere é colocado duas vezes na saída, seguindo-se um número que indica o comprimento da série. Deste modo, quando o descodificador detecta um mesmo caractere duas vezes seguidas, sabe que o caractere foi codificado com RLE e que deve consultar o número que segue esta dupla ocorrência do caractere.

O algoritmo a implementar deve ser orientado ao byte. Para distinguir um byte da sequência original, da própria dimensão da série, esta dimensão deve ser codificada com um byte também. Isto leva a que sequências de mais de 255 ocorrências de um mesmo byte tenham que ser divididas em duas. Por exemplo (utilizando uma notação semelhante à do Python para representar bytes), a sequência

```
\x82\x7F\x7F\x7F\x7F\x7a\x7a... série de 300 ocorrências deste último byte...\x7a
```

é codificada da seguinte forma:

```
\x82\x7F\x7F\x04\x8a\x8a\xFF\x7a\x7a\x2D
```

A negrito estão destacadas as dimensões das séries. Em decimal essas dimensões são 4, 255 e 45, respectivamente.

Consultar: https://en.wikipedia.org/wiki/Run-length_encoding