Folha prática 6 Introdução à Programação, DCC/FCUP, 2009/2010

Notas.

- Saber se um caracter `c` é uma letra (após "`from string import *`"): "`c in letters`"
- Saber se um caracter `c` é uma letra minúscula: "`c in lowercase`"
- Abrir o ficheiro chamado `"nome.txt"` para leitura: `f = open("nome.txt","r")` Ver Cap. 11 de "How to think...". O ficheiro aberto passa a ser designado por `f`.
- Ler todo o ficheiro `f` para a "string" `s`: `s = f.read()`
- Fechar o ficheiro designado por `f` (normalmente no fim da função ou programa): `f.close()`

Exercícios

---

1. Escreva uma função `palavras(txt)` que retorna uma lista das palavras numa "string" `txt`. Considere que uma *palavra* é uma sequência de letras maiúsculas ou minúsculas (i.e. caracteres pertencentes a `letters`). Exemplo:

   ```
   >>> palavras("---A Maria tinha um cordeirinho?")
   ["A", "Maria", "tinha", "um", "cordeirinho"]
   ```

2. Escreva uma função `contarChars(fich)` que conte o número de caracteres no ficheiro com o nome `fich`.
   Sugestão. Leia as notas sobre ficheiros no início desta folha.

3. Escreva uma função `contarPalavras(fich)` que conte o número de palavras no ficheiro com o nome `fich`.
   Sugestões: use a função `palavras` do exercício 1; leia as notas sobre ficheiros no início desta folha.

4. Pretende-se construir uma tabela com o número de palavras com comprimento 1, 2,... num ficheiro de texto (o comprimento de uma palavra é o número de caracteres que a constituem). Formato pretendido:

   ```
   comprimento 1: 2 palavras
   comprimento 3: 4 palavras
   comprimento 6: 1 palavras
   ...            ...
   ```

   Baseando-se no exercício 1, escreva uma "função" `comprimentos(fich)` para imprimir a tabela; apenas devem ser considerados para impressão os comprimentos para os quais existem palavras. O parâmetro `fich` é o nome do ficheiro de texto.
   Experimente!

5. Um polinómio é representado pela lista dos seus coeficientes, do menor para o maior grau; por exemplo

$$x^5 + 4x^3 + x + 2 \text{ é representado por } [2, 1, 0, 4, 0, 1]$$

A função que se apresenta em baixo efectua a conversão inversa, dando como resultado uma "string" que representa o polinõmio, *começando nos termos de maior ordem*. Por exemplo,

```
>>> polin([4,0,-5,8,1])
    x^4+8x^3-5x^2+4
>>> polin([1])
    1
>>> polin([1,1])
    x+1
>>> polin([1,1,1])
    x^2+x+1
>>> polin([1,1,0,8,1])
    x^4+8x^3+x+1
>>> polin([])
    0
```

Definição das funções:

```
def polin(li):              #  1
    n=len(li)               #  2
    sinal=False             #  3 Para que serve ´sinal´?
    r=""                    #  4 O que é ´r´?
    for i in range(n-1,-1,-1): #  5  Que valores toma ´i´?
        if li[i]!=0:        #  6 Nada se faz quando li[i]==0?
            r = r + termo(sinal,li[i],i)#  7
            sinal=True      #  8 Para que serve ´sinal´?
    if r=="":               #  9 Para quê este caso especial?
        r="0"               # 10
    return r                # 11

def termo(sinal,coef,grau): # A Para que serve esta função?
    s=""                    # B
    if coef==0:             # C
        return s            # D Que caso é este?
    if sinal and coef>0:    # E
        s=s+"+"             # F Quando é que se coloca sinal ´+´?
    if coef!=1 or grau==0:  # G
        s=s+str(coef)       # H Quando é que não se coloca...
    if grau>0:              # I     ...coeficiente?
        s=s+"x"             # J Quando é que se coloca ´x´?
    if grau>=2:             # K
        s=s+"^"+str(grau)   # L Quando é que se coloca a potência?
    return s                # M
```

(a) Função `polin`: (i) Responda às perguntas nos comentários.

(ii) Exprima de forma sucinta e clara – por forma a obter uma documentação apropriada – o efeito da função.
Experimente com as chamadas

```
polin([])  polin([1,0,0,0,1])  polin([-4,0,1,3])
polin([0,0,1])
```

(b) Função `termo`: (i) Responda às perguntas nos comentários.
(ii) Exprima de forma sucinta e clara – por forma a obter uma documentação apropriada – o efeito da função.
Experimente com as chamadas

```
termo(False,5,1)   termo(True,5,1)  termo(True,-1,0)
termo(True,-1,2)   termo(False,1,2)
```

6. O código Morse associa cada letra do alfabeto a uma sequência de "pontos" e "traços", conforme a tabela seguinte:

A	.-	B	-...	C	-.-.	D	-..	E	.	F	..-.
G	--.	H	....	I	..	J	.---	K	-.-	L	.-..
M	--	N	-.	O	---	P	.--.	Q	--.-	R	.-.
S	...	T	-	U	..-	V	...-	W	.--	X	-..-
Y	-.--	Z	--..								

Escreva uma função `morse(txt)` que converte as letras numa sequência de caracteres para Morse. O resultado deve ser uma "string" com pontos, traços e espaços. Caracteres da "string" original que não forem letras devem ser ignorados. Suponha que `txt` não contém letras maiúsculas. Exemplo de utilização:

```
>>> morse("ataque hoje")
    .- - .- --.- ..- . .... --- .--- .
```

Sugestão: começe por definir a lista `codigo = [".-",-...",-.-.", ...]`

7. Pretende-se representar conjuntos finitos em Python. Uma possibilidade é representar um conjunto de objectos $\{a_1, a_2, \ldots, a_n\}$ (números, "strings", etc.) como uma lista $[a_1, a_2, \ldots, a_n]$ (sem elementos repetidos).

   (a) Escreva uma função `pertence(x,A)` para testar se um objecto x pertence ao conjunto A.

   (b) Escreva funções `uniao(A,B)` e `interseccao(A,B)` para calcular, respectivamente, a união e intersecção de dois conjuntos.

   (c) Se assumir que as listas estão ordenadas, poderá implementar as operações anteriores de forma mais eficiente. Modifique as funções `pertence`, `uniao` e `interseccao` de forma a tomar partido desta hipótese.