TÉCNICO LISBOA

FUNDAMENTOS DA PROGRAMAÇÃO

2º Teste, 1 de junho de 2013, 1h30m

1. **(3.25)** Escreva um programa que pede o nome (mais de 2 caracteres) e o sobrenome (mais de 4 caracteres) a um utilizador e gera um *username* e uma *password*, que são escritos no ecrã. O nome e sobrenome pedidos devem conter apenas uma palavra e devem ser pedidos ao utilizador para terem dimensão superior a 2 e 4 caracteres, respectivamente. Não necessita de verificar o que é introduzido pelo utilizador. O *username* é composto pelos 2 primeiros caracteres do nome mais os 4 primeiros caracteres do sobrenome. A password deve ser construída usando o primeiro carácter do nome e 6 números inteiros gerados aleatoriamente.

Solução:

```
import random

nome = input('Introduza o seu nome (mais de 2 caracteres): ')
sobrenome = input('Introduza o seu sobrenome (mais de 4 caracteres): ')

username = nome[:2] + sobrenome[:4]

password = nome[0]

for i in range(6):
    password = password + str(int(random.random() * 9))

print('Username:', username, 'Password:', password)
```

2. **(1.75)** Escreva uma função recursiva *digitos_pares* que recebe um número inteiro não negativo *n*, e devolve um inteiro composto apenas pelos dígitos pares de *n*. Se *n* não tiver dígitos pares, a função deve devolver zero.

Solução:

```
def digitos_pares(n):

    if n == 0:
        return 0
    elif n % 2 == 0:
        return digitos_pares(n // 10) * 10 + (n % 10)
    else:
        return digitos_pares(n // 10)
```

3. **(2.0)** A função *somatorio* apresentada no livro:

```
def somatorio(calc_termo, linf, prox, lsup):
      soma = 0
      while linf <= lsup:
            soma = soma + calc_termo(linf)
            linf = prox(linf)
      return soma
```

é apenas a mais simples de um vasto número de abstracções semelhantes que podem ser capturadas por funções de ordem superior. Diga o que fazem as seguintes utilizações desta função, apresentando o valor da variável *soma* ao fim de cada ciclo:

a. `somatorio(lambda x: x+2, 1, lambda x: x*2, 15)`

b. `somatorio(lambda x: x%2, 1, lambda x: x+3, 10)`

Solução:

```
a. Primeira passagem: soma = 0 + 3 = 3
   Segunda passagem: soma = 3 + 4 = 7
   Terceira passagem: soma = 7 + 6 = 13
   Quarta passagem: soma = 13 + 10 = 23

b. Primeira passagem: soma = 0 + 1 = 1
   Segunda passagem: soma = 1 + 0 = 1
   Terceira passagem: soma = 1 + 1 = 2
   Quarta passagem: soma = 2 + 0 = 2
```

4. **(2.5)** Escreva uma função que escreve o conteúdo de um dicionário para um ficheiro com um nome (pré-determinado) à sua escolha. O dicionário contém chaves do tipo *cadeia de caracteres* e cada chave tem associada um valor do tipo *inteiro*. Esta função deve validar que o argumento que recebe é do tipo *dicionário* e se cada valor é do tipo *inteiro*. Todas as situações de erro devem ser assinaladas com de um *`ValueError().`* Cada linha do ficheiro deve conter uma informação deste tipo: *`chave (espaço) valor\n`*

Solução:

```
def escreve_ficheiro(d):
    '''escreve_ficheiro : dict -> {} '''

    if not isinstance(d, dict):
        raise ValueError('escreve_ficheiro: primeiro argumento nao e dicionario')

    conteudo = ''
    for i in d:
        if not isinstance(d[i], int):
            raise ValueError('escreve_ficheiro: valor nao e inteiro')
        else:
            conteudo = conteudo + i + ' ' + str(d[i]) + '\n'

    f = open('fich.txt', 'w')
    f.write(conteudo)
    f.close()
```

5. **(4.0)** Escreva uma função recursiva *conta_caracteres_tuplo* que recebe um tuplo de inteiros e caracteres e devolve o número de caracteres presentes no tuplo. A função deve retornar 0 se o tuplo estiver vazio.

   a. **(1.5)** Escreva a função descrita, gerando um processo recursivo.

Solução:

```
def conta_caracteres_tuplo(t):

    if tuplo == ():
        return 0
    elif isinstance(tuplo[0], str):
        return 1 + conta_caracteres_tuplo(t[1:])
    else:
        return conta_caracteres_tuplo(t[1:])
```

   b. **(1.5)** Escreva a função descrita, gerando um processo iterativo.

Solução:

```
def conta_caracteres_tuplo(t):

    def conta_aux(t, cont):

        if tuplo == ():
            return cont
        elif isinstance(tuplo[0], str):
            return conta_caracteres_tuplo(t[1:], cont + 1)
        else:
            return conta_caracteres_tuplo(t[1:], cont)

    conta_aux(t, 0)
```

   c. **(1)** Ilustre o encadeamento de operações geradas pelo processo recursivo da alínea **a)** para a seguinte chamada da função:

```
conta_caracteres_tuplo((1,'2','a',3,4,'b'))
```

Solução:

```
conta_caracteres_tuplo((1, '2', 'a', 3, 4, 'b'))
conta_caracteres_tuplo(('2', 'a', 3, 4, 'b'))
1 + conta_caracteres_tuplo(('a', 3, 4, 'b'))
1 + (1 + conta_caracteres_tuplo((3, 4, 'b')))
1 + (1 + conta_caracteres_tuplo((4, 'b')))
1 + (1 + conta_caracteres_tuplo(('b')))
1 + (1 + (1 + conta_caracteres_tuplo(())))
1 + (1 + (1 + 0))
1 + (1 + 1)
1 + 2
3
```

6. **(4.5)** Defina uma classe *loja_animais* cujo construtor recebe a informação dos animais que estão disponíveis para venda. A informação sobre os animais existentes está representada por um *dicionário*, onde cada tipo de animal é representado por uma *cadeia de caracteres* a que está associada o número de exemplares disponíveis. Defina um dicionário deste tipo:

   ```
   {'gato':5, 'cao':7, 'peixe':12, 'cobra':1, 'coelho':4}
   ```

   Os outros métodos suportados pela classe são:

   a. *consulta*, imprime a informação sobre os animais que estão para venda. Deve imprimir, por linha, o tipo do animal e a quantidade existente.

   b. *compra*, recebe como parâmetros o tipo de animal a comprar e a quantidade. Deve validar se o animal existe na loja. Caso não exista o animal ou a quantidade pretendida, a compra não tem efeito. A função deve então imprimir para o ecrã uma mensagem dando esta indicação e, como compensação, oferecer um peixe (caso existam na loja), actualizando o número de peixes existentes na loja. Caso não existam peixes na loja, deve simplesmente imprimir para o ecrã uma mensagem indicando que não há na loja animais para oferecer. Caso exista o animal pretendido em quantidade suficiente para efectuar a compra, deve actualizar o número de animais do tipo pretendido restantes após a compra. No final, deverá devolver a informação atualizada sobre o número de animais existentes na loja do tipo do que foi comprado ou, no caso de a compra não ter tido lugar, do número restante de peixes.

Solução:

```
class loja_animais:

    def __init__(self, info):
        self.animais = info

    def consulta(self):
        for a in self.animais:
            print(a, ':', self.animais[a])

    def compra(self, tipo, quantidade):
        if tipo in self.animais and quantidade < self.animais[tipo]:
            self.animais[tipo] = self.animais[tipo] – quantidade
            return self.animais[tipo]
        elif self.animais['peixe'] > 0:
            print('Oferecemos um peixe.')
            self.animais['peixe'] = self.animais['peixe'] – 1
            return self.animais['peixe']
        else:
            print('Nao ha animais para oferecer.')
            return self.animais['peixe']
```

7. **(2.0)** Considere que tem um ficheiro designado *numeros.txt* que contem um número por linha. Defina uma função que calcula e imprime o valor médio dos números contidos no ficheiro.

Solução:

```
def calcula_media():

    f = open('numeros.txt', 'r')
    linhas = f.readlines()
    f.close()

    soma = 0
    for l in linhas:
        soma = soma + eval(l)

    print('O valor medio e:', soma/len(linhas))
```