Contrôle de connaissances

Nom :..... **Prénom :**.....

Toutes les fonctions devront être écrites sous une forme récursive.

Exercice 1 : (4 points)

On considère la fonction `f`, prenant en argument deux entiers, suivante :

```
def f (n,p) :
  # n et p sont des entiers
    if (p <= 0 or n <= 0) :
        return 1
    elif (p>n) :
        return f( n , p-n)
    else :
        return n + f( n-1 , p+1 )
```

1. Expliquer les différents appels pour le calcul de $\texttt{f}(3,2)$ (on indique que $\texttt{f}(3,2)=7$) :

Correction

$$\begin{aligned} f((3,2) &= 3+f(2,3)\\ &= 3+f(2,1)\\ &= 3+2+f(1,2)\\ &= 3+2+f(1,1)\\ &= 3+2+1+f(0,2)\\ &= 3+2+1+1\\ &= 7 \end{aligned}$$

2. Expliquer les différents appels pour le calcul de $\texttt{f}(5,9)$ (on indique que $\texttt{f}(5,9)=16$) :

Correction

$$\begin{aligned} f((5,9) &= f(5,4)\\ &= 5+f(4,5)\\ &= 5+f(4,1)\\ &= 5+4+f(3,2)\\ &= 9+7\\ &= 16 \end{aligned}$$

Exercice 2 : (4 points)

On appelle **palindrome** un mot qui se lit dans les deux sens comme "kayak" ou "radar".
L'algorithme récursif pour vérifier si un mot est un palindrome se décrit ainsi :

- Le mot vide, un mot d'une lettre est forcement un palindrome.
- Si la première lettre est identique à la dernière, la récursion se fait sans ces lettres (exemple : pour "kayak" , "k=k" et "aya" est un palindrome...)
  Sinon, le mot n'est pas un palindrome.

On rappelle que la première lettre et la dernière lettre d'une chaine de caractère `mot`, non vide, sont respectivement `mot`$[0]$ et `mot`$[-1]$; Le mot sans ses deux extrémités est obtenu par `mot`$[1:-1]$.
Exemple :

```
>>> m = "abcde"
>>> m[0]
'a'
>>> m[-1]
'e'
>>> m[1:-1]
'bcd'
```

1. Écrire la fonction `est_un_palindrome`, qui prend en argument une chaine de caractère, et renvoie le booléen correspondant à la propriété d'un palindrome. Exemple :

```
>>> est_un_palindrome("aabaa")
True
>>> est_un_palindrome("aaabaa")
False
```

Correction

```
def est_un_palindrome(mot):
    if len(mot) ==0 or len(mot) ==1:
        return True
    else :
        if mot[0] == mot[-1] :
            return est_un_palindrome(mot[1:-1])
        else :
            return False
```

2. Modifier le programme précédent pour tester une phrase palindrome : Exemple :

```
>>> phrase ="elu␣par␣cette␣crapule"
>>> phrase_palindrome(phrase)
True
```

Correction

```
def phrase_palindrome(phrase):
    if len(phrase) ==0 or len(phrase) ==1:
        return True
    else :
        if phrase[0] == phrase[-1] :
            return phrase_palindrome(phrase[1:-1])
        elif phrase[0]== "␣" :
            return phrase_palindrome(phrase[1:])
        elif phrase[-1]== "␣" :
            return phrase_palindrome(phrase[:-1])
        else :
            return False
```

Exercice 3 : (12 points)

Nous disposons d'une base de données sous forme de tableau de couples, où chaque nom est associé à sa note.(chaque nom est supposé unique dans la liste)

Exemple :

```
liste_exemple = [ ("A",10) , ("B" , 12) , ("C", 13),("D",8) , ("E" , 7),\
                  ("F" , 12) , ("G" , 8) , ("H" , 12)]
```

Dans cet exemple, l'élève "A" a donc pour note 10 ; "B" a pour note 12, et "C" a pour note 13, etc...

1. Compléter la fonction `recherche_note` qui prend une base de données `liste` et une chaine de caractères `nom`, et renvoie la note correspondante au `nom` dans la liste.

   Exemple :

   ```
   >>> recherche_note ( liste_exemple , "D" )
   8
   ```

   ```
   def recherche_note ( liste, nom) :

       if len(liste) == ... : #cas teminal

           return "eleve inconnu"
       else :
           ( nom_0 , note_0 ) = ....
           if nom_0 == nom :
               # si le premier nom correspond
               return ....
           else :
             # sinon on cherche dans le reste de la liste

               return recherche_note ( .....    , ......)
   ```

   Correction

   ```
   def recherche_note ( liste, nom) :
       if  len(liste) == 0 : #cas teminal
           return "eleve inconnu"
       else :
           ( mon_0 , note_0 ) = liste[0]
           if mon_0 == nom :
               return note_0
           else :
               return recherche_note ( liste[1:], nom)
   ```

2. Compléter la fonction `note_max` qui prend une base de données `liste` et renvoie le couple correspondant à la première note maximale dans la liste.

   Exemple :

   ```
   >>> note_max (liste_exemple)
   ('C', 13)
   ```

```
def note_max ( liste ) :
    if liste == [] : #cas a eviter
        return "␣liste_vide"
    elif len(liste) == 1 : # cas terminal
        return ....
    else :
      # on recupere le couple solution du reste de la liste
        ( nom_max_reste, note_max_reste) = note_max(.............)

    # on identifie le couple de tete
        (nom_0 , note_0) = .......
        # on les compare
        if ........ > ...........:

            return ..................
        else :
            return ..................
```

Correction

```
def note_max ( liste ) :
    if liste == [] :
        return "␣liste_vide"
    elif len(liste) == 1 :
        return (liste[0])
    else :
        ( nom_max_reste, note_max_reste) = note_max(liste[1:])
        (nom_0 , note_0) = liste[0]
        if note_0 > note_max_reste :
            return liste[0]
        else :
            return (  nom_max_reste, note_max_reste)
```

3. Compléter la fonction `ont_la_moyenne` qui prend une base de données `liste` et renvoie la liste des élèves ayant une note supérieur à 10.

   Exemple :

```
>>> ont_la_moyenne(liste_exemple)
['A', 'B', 'C', 'F', 'H']
```

```
def ont_la_moyenne(liste) :
    if liste == [] :  #cas terminal
        return ............
    else :
        ( nom_0 , note_0 ) = ...........

        if note_0 >= ....... :

            return [.......] + ont_la_moyenne(.......)
        else :
```

```
            return .......
```

Correction

```
def ont_la_moyenne(liste) :
    if liste == [] :
        return []
    else :
        ( nom_0 , note_0 ) = liste[0]
        if note_0 >= 10 :
            return [nom_0] + ont_la_moyenne(liste[1:])
        else :
            return ont_la_moyenne(liste[1:])
```

4. Écrire la fonction `somme_note` qui prend en argument une base de données et renvoie la somme des notes.

   Exemple :

```
>>> somme_note(liste_exemple)
82
```

Correction

```
def somme_note(liste ) :
    if liste == [] :
        return 0
    else :
        (nom,note ) = liste[0]
        return note + somme_note(liste[1:])
```