TP noté : corrigé

PTSI Lycée Eiffel

22 décembre 2017

Ce corrigé propose des programmes Python répondant à l'ensemble des questions posées, en essayant de ne pas utiliser de commandes ou fonctions inconnues des élèves ou interdites dans l'énoncé du TP, et en essayant également de conserver des structures lisibles et simples. Autrement dit, il y a souvent des instructions conditionnelles voire des boucles qui pourraient être évitées à coups de solutions un peu techniques mais que j'ai conservées pour tenter d'écrire des programmes dans le style de ce que j'attendais a priori de vous. Bien entendu, l'efficacité des programmes écrits ne sera en général pas commentée puisqu'il ne s'agissait pas d'un objectif pour ce TP. Bien entendu également, pour quasiment toutes les questions, d'autres solutions que celles présentées ici étaient tout à fait envisageables.

I. Quelques algorithmes élémentaires sur les chaînes de caractères.

Question 1.

```
def converliste(c) :
    l=[]
    for i in c :
        l.append(i)
    return l
```

Un algorithme simple et de bon goût : on crée une liste et on insère dedans tous les caractères de la chaîne l'un après l'autre. Notez qu'il existe en fait déjà une commande **list** en Python qui effectue ce travail.

Question 2.

```
def converchaine(l) :
    c=''
    for i in l :
        c=c+i
    return c
```

Exactement la même chose qu'au-dessus, en utlitilisant l'opérateur + pour concaténer les caractères de la chaine. Là aussi, il existe des commandes Python toutes faites mais c'est un poil plus compliqué (les curieux se renseigneront sur la commande **join**).

Question 3.

```
def sommeascii(c) :
    s=0
    for i in c :
        s=s+ord(i)
    return s
```

L'interdiction par l'énoncé d'utiliser la commande **sum** oblige à écrire une petite boucle, mais ça reste très rapide.

Question 4.

```
def miroir(c) :
    z=''
    for i in range(len(c)-1,-1,-1) :
        z=z+c[i]
    return z
```

La méthode la plus rapide consiste bien sûr à transformer la chaîne en liste, lui appliquer la méthode **reverse** puis revenir à une chaîne de caractères. Mais comme l'énoncé n'était pas très clair sur l'autorisation ou non de recourir à reverse, je vous ai présenté une solution plus rustique en utilisant une boucle descendante (ce que vous savez tous faire si vous avez écouté mes remarques quand j'ai rendu le premier DS écrit).

Question 5.

```
def palindrome(c) :
    return (c==miroir(c))
```

On peut difficilement faire plus court que ça, mais ça marche très bien. L'égalité entre c et miroir(c) prend une valeur booléenne (True ou False en Python) qui est exactement ce qu'on veut retourner (bien sûr, on peut ajouter un if).

II. Cryptages par substitution.

Question 6.

```
def decodecesar(c) :
    z=''
    for i in c :
        if ord(i)>67 :
            z=z+chr(ord(i)-3)
        else :
            z=z+chr(ord(i)+23)
    return z
```

L'exemple typique de programme pas optimisé mais relativement simple : on prend chaque caractère de la chaine, et on regarde s'il correspond à une lettre se trouvant après le C (qui a un code ASCII égal à 67). Si c'est le cas on décale la lettre de trois rangs vers le bas dans l'ordre alphabétique (ce qui correspond à diminuer de trois unités son code ASCII puis bien sûr à rechercher le caractère correspondant). Sinon, il s'agit d'un des trois premières lettres de l'alphabet et il faut cette fois la décaler de 23 rangs (dans le sens normal). Le programme ne fonctionnera que sur un texte constitué uniquement de caractères alphabétiques majuscules (sur des lettres minuscules par exemple, il ne ferait pas ce qu'il faut pour les lettres a, b et c), et on pourrait éviter l'instruction conditionnelle en condensant les deux cas en un seul à coups de modulo 26 bien placés.

Question 7.

```
def decodedecalage(c,n) :
    z=''
    for i in c :
        if ord(i)>64+n :
            z=z+chr(ord(i)-n)
        else :
            z=z+chr(ord(i)+26-n)
```

```
    return z
```

C'est exactement le même programme que ci-dessus, adapté bien sûr à la valeur de n.

Question 8.

```
def codemiroir(c) :
    z=''
    for i in c :
            z=z+chr(155-ord(i))
    return z
```

Là, une seule commande doit suffire mais il faut réfléchir un peu pour trouver quelque chose qui marche. Si on remplace un entier k par l'entier $27 - k$, celà revient à inverser les rangs de l'ordre alphabétique (1 devient 26, 2 devient 25 etc). Ici, il faut en plus tenir compte qu'on est décalés (on part de 65 au lieu de partir de 1), et remplacer k par $64 + (27 - (k - 64)) = 155 - k$.

Question 9.

```
def nombreE(c) :
    n=0
    for i in c :
        if i=='E' :
            n=n+1
    return n
```

Ça c'est un exemple vu en cours, interdiction de se tromper sur cette question !

Question 10.

```
def plusfrequent(c) :
    m=0
    for i in range(65,91) :
        z=chr(i)
        n=0
        for j in c :
            if j==z :
                n=n+1
        if n>m
            m=n
            a=z
    return a
```

Encore un programme assez rustique, mais normalement assez simple à comprendre. La variable m représente le maximum d'occurrences obtenues pour une lettre, et a représente la lettre correspondante. On initialise m à 0 (inutile d'initialiser a, sauf si le texte est vide, au moins une lettre apparaitra au moins une fois). On parcourt ensuite l'ensemble les lettres majuscules (la variable z prend successivement pour valeur 'A', 'B' etc.) et pour chaune on compte le nombre d'apparitions exactement comme dans le programme précédent. Si on en obtient plus que m, on change les valeurs des variables de stockage m et a et on passe à la lettre suivante.

Question 11.

```
def dedodedecalageinconnu(c) :
    a=plusfrequent(c)
    n=(ord(a)-69)%26
    return decodedecalage(c,n)
```

Inutile de réecrire un programme, on obtient l'image de la lettre 'E' grâce au programme de la question 10 puis on décode en utilisant celui de la question 7 avec le bon décalage (qui correspond à l'écart entre le code ASCII de la lettre 'E', donc 69, et celui de son image, avec un modulo 26 pour tenir compte du cas où cette image se trouve avant le 'E' dans l'alphabet). Le décodage du texte donné dans l'énoncé donne : 'PETITPAPANOELQUANDTUDESCENDRASDUCIELAVECDESJOUETSPARMILLIERSNOUBLIESPASMONPETITSOULIER'. À la faute do'rthographe près (un S en trop à OUBLIE), ça semble cohérent.

Question 12.

```
def decodedecalageprogressif(c) :
    z=''
    n=0
    for i in c :
        n=(n+1)%26
        a=ord(i)-n
        if a>64 :
            z=z+chr(a)
        else :
            z=z+chr(a+26)
    return z
```

Le principe est globalement le même que pour un décalage constant, avec une variable supplémentaire qui indique le décalage (ici l'entier n) et qui évolue donc à chaque nouveau passage dans la boucle (avec un modulo 26 pour revenir à un décalage nul après 25 lettres). Dans le cas où le décalage a ramené notre code ASCII à une valeur inférieure à 64, on rajoute 26 pour retrouver la bonne lettre de l'alphabet.

III. Cryptage de Vigenere.

Question 13.

```
def vigenere(c,cle) :
    n=len(cle)
    a=0
    z=''
    for i in c :
        b=ord(i)+ord(cle[a])-64
        if b<91 :
            z=z+chr(b)
        else :
            z=z+chr(b-26)
        a=(a+1)%n
    return z
```

En fait ça fonctionne à peu près comme les programmes précédents : la variable a joue essentiellement le rôle que jouait n dans la question 12, elle sert à savoir quelle lettre de la clé doit être utilisée pour le décalage, et change donc après chaque passage dans la boucle (avec à nouveau un modulo pour revenir au début de la clé après avoir atteint la fin). Notons que l'exemple donné dans l'énoncé était incorrect puisque le décalage était obtenu en soutrayant un au rang alphabétique de chaque lettre de la clé. En gros, on considérait qu'un 'A' dans la clé ne devrait pas changer la lettre du code, alors que dans la description que j'avais donnée, c'est le 'Z' qui joue ce rôle. Si on applique le programme écrit ci-dessus, on aura donc un message décalé d'un caractère par rapport à celui de l'énoncé.

Question 14.

```
def indicecoincidences(c1,c2) :
    n1=len(c1)
    n2=len(c2)
    if n1<n2 :
        n,p=n1,n2
    else :
        n,p=n2,n1
    s=0
    for i in range(n) :
        if c1[i]==c2[i] :
            s=s+1
    return s/p
```

La laideur du programme découle surtout des premières lignes permettant de gérer le cas où les deux chaînes sont de longueur différentes : n est la plus petite des deux longueurs, celle qui permet de savoir combien de caractères il faudra comparer, et p est la plus grande, celle par laquelle on va diviser à la fin pour obtenir l'indice. Pour le reste, c'est assez facile.

Question 15.

```
def longueurcle(c) :
    im=0
    n=0
    for i in range(1,11) :
        c1=c
        c2=c[i :]+c[ :i]
        p=indicecoincidences(c1,c2)
        if p>im :
            im=p
            n=i
    return n
```

C'est un calcul de maximum classique : la variable im représente l'incide de coïncidence maximal obtenu, la variable n le décalage correspondant à cet indice maximal, et on se contente de calculer les indices les uns après les autres. La ligne c2=c[i :]+c[:i] permet d'obtenir une copie de c décalée de i unités (en recopiant les i premiers caractères au bout de la chaine).

Question 16.

```
def decodedecalageprogressif(c) :
    z=''
    n=0
    for i in c :
        n=(n+1)%26
        a=ord(i)-n
        if a>64 :
            z=z+chr(a)
        else :
            z=z+chr(a+26)
    return z
```

Bon, c'est un peu moche mais c'est surtout de la mise en forme : les premières lignes permettent de découper le texte codé et de le caser dans une liste de chaînes de caractères, chaque élément de la liste ne contenant que des caractères qui ont subi le même décalage (la liste a donc une longueur correspondant à la longueur de la clé). Ensuite, on décode indépendamment chaque élément de la liste

à l'aide du programme de la question 11, puis on reconstitue le texte en piochant successivement un caractère dans chaque chaîne (ce qui est un peu pénible car bien entendu le nombre de caractères du texte total n'a aucune raison d'être un multiple de la longueur de la clé). Bon, le programme marche, mais quand on décode le texte proposé, on trouve un étrange 'ILESTFVMPSPOGIMOIDEOCO-RECEFGBIENTDFPLONGQKDIFFIOZLEENVAL' (je vous ai juste mis le début) manifestement incorrect. En fait, le texte donné n'était pas assez long (après découpages en tranches du moins) pour que le programme de la question 11 fonctionne sur chaque chaîne ! Bon, la longueur de clé obtenue est égale à 7, ça c'est correct, et si on se débrouille pour demande au programme la clé avec lequel le texte est censé avoir été codé, on voit que ça commence par 'ROUPO' et on se doute que la clé était 'ROUPOIL' même si les deux derniers décalages ne sont pas trouvés correctement par notre programme. Après décodage avec cette bonne clé, on obtient :

'ILESTTEMPSPOURMOIDECLORECETPBIENTROPLONGETDIFFICILEENVOUSSOUHAITENT
DETRESBONNESVACANCESETDEJOYEUSESFETESDEFINDANNEEAVECDEBEAUXCADEAUX
AUPIEDDUSPINILPARAITQUELESLIVRESDEPYTHONSONTTRESALAMODECETTEANNEENH
ESITEZPASAENDEMANDERUNAUPERENOEL'

Une ou deux fautes de frappe encore une fois !