TP Listes

1 Construire des listes

Programme cubons.py :

Ecrivez un script, nommé cubons.py, qui génère, puis affiche dans la console, la liste des carrés et la liste des cubes des nombres de 2 à 16. Le programme sera écrit deux fois, de deux façons différentes :

— avec une boucle while et la concaténation

— avec une boucle for

Programme tournons.py :

En vous inspirant du programme suivant, créer un programme donnant les courbes représentatives des fonctions f et g définies pour tout angle x (en degrés) de [-90 ;270] par $f(x) = sin(2x + 45)$ et $g(x) = cos(x)sin(x)$avec x des angles en degrés pris entre -90 et 270. Les courbes sont à tracer en rouge et vert dans un repère aux axes légendés.

```
import matplotlib.pyplot as plt
import math

x, y =[],[]                  # j'initialise x et y deux listes vides
for i in range(0,360) :
    x.append(i)              # append rajoute la valeur entre
                             # parenthèse à la suite de la liste
    angle = i*2*math.pi/360 #conversion des degrés vers les radians
    y.append(math.cos(angle))
plt.plot(x,y,"bo")           # dans "bo" le b correspond à 'blue' et
                             # le 'o' correspond à un rond
plt.show()                   # une fois que le graphique est prêt on
                             # peut l'afficher.
```

Le listing complet est à la fin du document.

Pour le programme vous pourrez utiliser les fonctions :

legend(" titre de la figure ") axis([xmin, xmax, ymin, ymax])

xlabel("indication sur l'axe des x ") ylabel() savefig("nom du fichier ")

Dans un deuxième temps vous pourrez modifier le calcul de signal par :

— $sinus(angle) + \frac{1}{3}\ sinus(3\ angle)$

— $sinus(angle) + \frac{1}{3}\ sinus(3\ angle) + \frac{1}{5}\ sinus(5\ angle)$

— $sinus(angle) + \frac{1}{3}\ sinus(3\ angle) + \frac{1}{5}\ sinus(5\ angle)) + \frac{1}{7}\ sinus(7angle)$

— etc

2 Parcourir des listes

Programme bonOrdre.py :

Vous disposez d'une liste de nombres entiers quelconques, certains d'entre eux pouvant être présents en plusieurs exemplaires. Par exemple :

liste = [19, 9, 7, 17, 10, 8, 9, 9, 19, 11, 10, 16, 4, 16, 16, 0, 17, 4, 8, 14] .

Dans un script nommé bonOrdre.py, définissez une liste avec des doublons puis écrivez un programme qui supprime ces doublons, trie la liste puis affiche le résultat. Deux approches sont possibles :

— parcourir la liste élément par élément et recopier chacun dans une autre en s'abstenant de le faire dans le cas des doublons.

— parcourir la liste élément par élément et supprimer ceux présents plus d'une fois dans la liste. Attention : comme la liste peut être modifiée à chaque itération de la boucle, la structure de contrôle while est plus adaptée que la boucle for. Ainsi le parcours de la liste s'adapte aux modifications de la longueur de la liste.

Ecrivez votre programme avec les deux approches et testez avec l'exemple donné.

Programme combienDeMots.py :

Le fichier Ciceron.txt accessible sur le site du TP contient un extrait de l'ouvrage philosophique de Cicéron, publié en été 45 av. J.-C, et dédié à Brutus. Ce texte serait à l'origine du faux-texte « Lorem ipsum »

utilisé en typographie. Pour accéder à ce texte dans un script, copiez Ciceron.txt dans le répertoire du script et utilisez l'instruction suivante : texte = open("Ciceron.txt", 'r').read().

Nous verrons bientôt que la variable texte est affectée avec l'ensemble du contenu du fichier Ciceron.txt.

A l'aide de méthodes associées aux chaînes (notamment la méthode split[1]) et des méthodes associées aux listes, écrivez un script, nommé combienDeMots.py, qui compte dans le texte de Cicéron :

1. le nombre de fois où la chaîne 'non' apparait (il faudra en trouver 114)
2. le nombre de paragraphes[2] (il faudra en trouver 80)
3. le nombre de paragraphes contenant des caractères numériques (il faudra en trouver 34)

3 Algorithmique avec les listes

Programme premier.py :

Un nombre premier est un nombre qui n'est divisible que par un et par lui-même. On souhaite écrire un programme, nommé premier.py, qui établit la liste de tous les nombres premiers compris entre 1 et 1000, en utilisant la méthode du crible d'Eratosthène[3] vieux de 22 siècles. Pour cela vous pouvez suivre la séquence suivante :

— Créez une liste de 1001 éléments : l'indice d'un élément de cette liste permettra de représenter un nombre de 0 à 1000. La valeur initiale de tous les éléments de cette liste sera 1.

— Parcourez la liste à partir de l'élément d'indice i=2. A chaque itération : si l'élément analysé possède la valeur 1, mettez à zéro tous les éléments de la liste dont les indices sont des multiples entiers de i (sauf l'élément d'indice i).

Lorsque vous aurez parcouru toute la liste, les indices des éléments qui seront restés à 1 seront les nombres premiers recherchés. En effet, à partir de l'indice 2, vous annulez tous les éléments d'indices pairs : 4, 6, 8, 10, etc. Avec l'indice 3, vous annulez les éléments d'indices 6, 9, 12, 15, etc., et ainsi de suite. Seuls resteront à 1 les éléments dont les indices sont effectivement des nombres premiers.

Optionnel : En guise de divertissement, utilisez le crible d'Eratosthène pour réaliser la spirale d'Ulam[4] dans une fenêtre Turtle ou Tkinter[5]

Programme textJumbler.py :

Le paragraphe suivant est une légende de l'internet[6] :

I cdn'uolt blveiee taht I cluod aulaclty uesdnatnrd waht I was rdanieg : the phaonmneel pweor of the hmuan mnid. Aoccdrnig to a rseearch taem at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteer be in the rghit pclae. The rset can be a taotl mses and you can sitll raed it wouthit a porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe. Scuh a cdonition is arppoiatrely cllaed Typoglycemia . Amzanig huh ? Yaeh and you awlyas thguoht slpeling was ipmorantt.

Ce qui, en clair, est :

I couldn't believe that I could actually understand what I was reading: the phenomenal power of the human mind. According to a research team at Cambridge University, it doesn't matter in what order the letters in a word are, the only important thing is that the first and last letter be in the right place. The rest can be a total mess and you can still read it without a problem. This is because the human mind does not read every letter by itself, but the word as a whole. Such a condition is appropriately called Typoglycemia. Amazing, huh ? Yeah and you always thought spelling was important.

On se propose d'écrire un programme Python, nommé textJumbler.py qui permette d'effectuer cette transformation : on mélange les lettres de chaque mot de la phrase. Les mots de trois lettres ou moins restent inchangés. Les mots plus longs sont mélangés, en conservant la première et la dernière lettre.

1. Ecrivez une fonction coupe(phrase) qui découpe une phrase en mots. Toutes les suites de lettres (minuscules ou majuscules) forment un mot et sont groupées ensemble. Les autres caractères sont séparés individuellement. Par exemple :

>>> print coupe('Bonjour , les amis .') donnera ['Bonjour ', ',', ' ', 'les ', ' ', 'amis ', '.']

On pourra s'aider de la constante string.ascii_letters du module string.

---

[1] https://docs.python.org/2.7/library/stdtypes.html?highlight=split#str.split ici pour faire simple chaine.split(" ") va casser la chaine au niveau des espaces créer une liste contenant les morceaux.

[2] https://fr.wikipedia.org/wiki/Paragraphe

[3] https://fr.wikipedia.org/wiki/Crible_d%27%C3%89ratosth%C3%A8ne

[4] https://fr.wikipedia.org/wiki/Spirale_d%27Ulam

[5] http://www.jchr.be/python/tkinter.htm

[6] https://en.wikipedia.org/wiki/Typoglycemia

Par exemple : **if chaine[i] not in string.ascii_letters** : déclanchera une action si l'élément de rang i de la chaine n'est pas une lettre.

2. Ecrivez une fonction melange(mot) qui mélange les lettres du milieu du mot, en laissant intactes la première et la dernière lettre. Si le mot fait plus de 3 lettres, la fonction mélange doit obligatoirement retourner un mot différent du paramètre mot. Vous êtes libres de choisir la méthode que vous souhaitez pour mélanger les lettres. La fonction suivante peut vous être utile :

```
def echange(mot , i, j):
        """     ( i doit être strictement plus petit que j )
        Cette fonction échange deux lettres .
        Par exemple : >>> echange('amis', 1, 2) 'aims'          """
        return mot[:i] + mot[j] + mot[i+1:j] + mot[i] + mot[j+1:]
```

3. Utilisez les fonctions précédentes dans le programme principal pour transformer une phrase saisie par l'utilisateur en mélangeant les lettres du milieu des mots. Affichez le résultat.

Programme tri.py :

La fonction sort permet de faire un tri sur une liste. Plusieurs algorithmes existent, le tri par sélection est parmi les plus simples. Cet algorithme peut être effectué in-situ en considérant la liste comme un tableau et en permutant le plus petit élément avec l'élément à sa position[7]. La version proposée ici construit une nouvelle liste à partir d'une liste initiale dans une fonction comme suit :

```
1 # définition de la fonction de tri par sélection avec une liste à trier en argument
2       # initialisation de la liste résultat
3       # boucle répétitive tant que la liste à trier est non vide
4               # détermination de l'élément de plus petite valeur
5               # suppression de l'élément de plus petite valeur de la liste
6               # ajout de l'élément de plus petite valeur à liste résultat
```

Dans un premier temps, écrivez la fonction pluspetit qui cherche la position du plus petit élément d'une liste. Ecrivez ensuite la fonction triSelection qui réalise le tri par sélection. Pour cela utilisez les méthodes index, append et pop[8] . Testez votre tri sur une liste simple dans votre script. Testez aussi votre tri avec une liste de grande dimension (10000) remplie de valeurs aléatoires (avec randint ou random du module random) et mesurez le temps qu'il faut pour effectuer ce tri comme suit :

```
1       # tri avec affichage du temps écoulé
2       import time
3       tps=time.clock()
4       triSelection(L)
5       print time.clock()-tps
```

Il serait intéressant de comparer les performances du tri par sélection par rapport à la méthode sort. Pour cela il faut évaluer le temps moyen de tri pour une liste de dimension donnée M (100, 1000 ou 10000 à tester). Il faut donc réaliser N tris de N listes de valeurs aléatoires et diviser le temps total par N. Réalisez un script qui compare les performances du tri par sélection par rapport à la méthode sort pour une liste de 10000 éléments. A l'image du TD, écrivez une fonction triBulle[9] et comparez les performances des trois algorithmes.

Listing : programme

```
import matplotlib.pyplot as plt
import math
x, y =[],[]                     # j'initialise x et y deux listes vides
for i in range(0,360) :
   x.append(i)                  # append rajoute la valeur entre  parenthèse à la suite de la liste
   angle = i*2*math.pi/360      #conversion des degrés vers les radians
   y.append(math.cos(angle))
plt.plot(x,y,"bo")              # dans "bo" le b correspond à 'blue' et le 'o' correspond à un rond
plt.show()                      # une fois que le graphique est prêt on peut l'afficher.
```

[7] https://fr.wikipedia.org/wiki/Tri_par_s%C3%A9lection
[8] http://docs.python.org/2/tutorial/datastructures.html
[9] https://fr.wikipedia.org/wiki/Tri_%C3%A0_bulles