UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Licence Informatique
Info0603 – 2021/2022

Travaux dirigés et TP n° 2

Itérateurs, Classes603, Compression et Entropie

note : Pour pouvoir faire fonctionner les tests des docString de chaque fonction, ajouter en dernière ligne de votre module :

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Exercice 1 (Éléments Python)
Écrire à la main les sorties du script Python suivant :

```
l=['a','b','c','d','e']
l+=['f']
print(len(l)); print(l+l); print(l[0])
print(l[0:2]); print(l[0:]); print(l[-1])
print(l[-2]); print(l[:-1]); print(l[:-2])
```

Exercice 2 (Listes de Listes)
Écrire les programmes Python générant les sorties suivantes :

```
Itérateur de listes: [0, 0, 0] [0, 0, 1] [0, 0, 2] [0, 0, 3] [0, 1, 0] [0, 1, 1] [0, 1,
2] [0, 1, 3]
[0, 2, 0] [0, 2, 1] [0, 2, 2] [0, 2, 3] [0, 3, 0] [0, 3, 1] [0, 3, 2] [0, 3, 3]
[1, 0, 0] [1, 0, 1] [1, 0, 2] [1, 0, 3] [1, 1, 0] [1, 1, 1] [1, 1, 2] [1, 1, 3]
[1, 2, 0] [1, 2, 1] [1, 2, 2] [1, 2, 3] [1, 3, 0] [1, 3, 1] [1, 3, 2] [1, 3, 3]
[2, 0, 0] [2, 0, 1] [2, 0, 2] [2, 0, 3] [2, 1, 0] [2, 1, 1] [2, 1, 2] [2, 1, 3]
[2, 2, 0] [2, 2, 1] [2, 2, 2] [2, 2, 3] [2, 3, 0] [2, 3, 1] [2, 3, 2] [2, 3, 3]
[3, 0, 0] [3, 0, 1] [3, 0, 2] [3, 0, 3] [3, 1, 0] [3, 1, 1] [3, 1, 2] [3, 1, 3]
[3, 2, 0] [3, 2, 1] [3, 2, 2] [3, 2, 3] [3, 3, 0] [3, 3, 1] [3, 3, 2] [3, 3, 3]
Itérateur d'ensembles :
[0, 1, 2] [0, 1, 3] [0, 2, 3][1, 2, 3]
Itérateur de listes sans redites
[0, 1, 2] [0, 1, 3] [0, 2, 1] [0, 2, 3] [0, 3, 1] [0, 3, 2]
[1, 0, 2] [1, 0, 3] [1, 2, 0] [1, 2, 3] [1, 3, 0] [1, 3, 2]
[2, 0, 1] [2, 0, 3] [2, 1, 0] [2, 1, 3] [2, 3, 0] [2, 3, 1]
[3, 0, 1] [3, 0, 2] [3, 1, 0] [3, 1, 2] [3, 2, 0] [3, 2, 1]
```

Faire la même chose mais généralisé aux cas de p éléments parmi n

*** La même chose mais sans récursivité

Construire les itérateurs correspondant afin de pouvoir avoir les sorties précédentes avec du code tel que :

```
print('Itérateur de listes')
for l in iterListes(3,4):
    print(l)
```

* Reprendre les sorties précédentes pour afficher des liste de lettres ou de mots.

Un exemple de création d'itérateur de coordonnées :

```
def iterCoord(n):
   "Un itérateur renvoyant xy pour tous les points de l'image"
   i=0
   j=0
   for i in range(n):
     for j in range(n):
          yield (i,j) #return mais qui reprendra ici l'exécution au prochain appel
```

Un site à aller voir si on souhaite approfondir : http ://sametmax.com/comment-utiliser-yield-et-les-generateurs-en-python/

Exercice 3 (Entropie)
Calculer les entropies des listes suivantes (à la main !)

[1,2] ; [1,1,2,2] ; [1,2,1,2] ; [1,2,3,4] ; [1,2,3,4,5,6,7,8]

[0,1,2,2] ; [0,1,2,2,3,3,3,3] ; [0,1,2,2,3,3,3,3,4,4,4,4,4,4,4,4]

Classer ces types de fichiers suivants par entropie croissante : txt,odt,pdf,py,odp,zip,jpg,png,pcx,bmp et le vérifier par un calcul en TP.

Exercice 4 (Exercice facultatif : de l'art de distribuer les carte)
Simuler plusieurs façons de mélanger un jeu de carte afin de déterminer la meilleure méthode.

Exercice 5 (La classe Binaire603)
Cette classe hérite de list et est appelée à être utilisée systématiquement dans nos TPs. Il est donc essentiel qu'elle soit complète et testée.

Pour ce faire il vous suffit de charger le fichier Binaire603 et de compléter toutes les fonctions comportant

```
raise NotImplementedError
```

Les méthodes déjà programmées :

* Binaire603 (constructeur à partir d'une liste de byte)
* sauvegardeDansFichier(self,nomFic) : "Enregistre dans un fichier nommé nomFic"
* bin603DepuisFichier(nomFic,verbose=False) : "renvoie un Binaire603 d'après les données d'un fichier"
* exBin603(taille=1000,num=0)
  Exemples de cette classe avec pour num=
  0 : 255 fois 255 puis 254 fois 254 etc...
  1 : un octet de chaque
  2 : 1000 octets aléatoires
  3 : 254 13 puis 255 14
  4 : 254 13 puis 255 14 puis 256 15
  5 : et plus : 1000 fois un octet sur deux à 255
* estOctet(val) : Renvoie true si val est un entier et s'il se trouve dans [0..255]
* ajouteOctet(self,data) : Ajoute un octet ou une liste d'octets tout en vérifiant la validité des données
* lisOctet(self,pos) : Lis un octet et incrémente pos
* __str__ et __repr__ __eq__

Les méthodes à programmer :

+ lFrequencesEtCles(self) (Renvoie la fréquence de chaque octet sous la forme d'une liste ex : lf[13]=0.1 Ainsi que la liste des 256 octets triés dans l'ordre décroissant Voir : https ://linuxhint.com/sort-lambda-python/
+ entropie(self)
+ affFreq(self) : "Affiche le Tableau des fréquences de la liste et son entropie"
+ ajouteLongueValeur(self,val) : Code une valeur entière de taille qulconque de telle façon qu'elle puisse être récupérable par lisLongueValeur
+ lisLongueValeur(self,pos) : Renvoie tout ce qu'il faut pour enregistrer/ajouter une valeur entière de taille quelconque"à la position pos. Renvoie cette valeur et la nouvelle valeur de pos :

```
def lisLongueValeur(self,pos):
    """Ajouter une valeur entière de taille quelconque à la position pos.
    Renvoie cette valeur et la nouvelle valeur de pos
    >> monBin=Binaire603([12,13])
    >> monBin.ajouteLongueValeur(1000);monBin.ajouteLongueValeur(100000)
    >> pos=1
    >> v0,pos=monBin.lisOctet(pos)
    >> v1,pos=monBin.lisLongueValeur(pos);v2,pos=monBin.lisLongueValeur(pos)
    >> v0,v1,v2
    (13, 1000, 100000)
```

+ Ecrire un algorithme de compression applicable à Binaire603