

Séance 6 : LISTES, GESTION MÉMOIRE ET EXCEPTIONS

L1 – Université Côte d'Azur

Exercice 1 — Occurrence

Pour chacune des fonctions, écrivez au moins trois tests avec `assert`.

1. En utilisant une boucle `for`, définissez une fonction `apparaît(x, L)` qui renvoie `True` si le nombre `x` apparaît dans la liste `L` et `False` sinon.
2. Comment auriez-vous fait sans boucle explicite mais avec le mot-clé `in` ?
3. Définissez la fonction `contient(L1, L2)` qui renvoie `True` si tous les nombres de la liste de nombres `L1` apparaissent dans `L2` et `False` sinon.
4. Définissez la fonction `commun(L1, L2)` qui renvoie le premier nombre de `L1` qui apparaît aussi dans `L2`. Si un tel nombre n'existe pas, `commun(L1, L2)` renvoie `False`.

Exercice 2 — Simulateur d'achat

On se donne une liste `stock` correspondant à un inventaire des produits d'un magasin.

```
1 >>> stock
2 [('Pommes', 10), ('Carottes', 5), ('Radis', 23), ('Lentilles', 534), ('Poivrons', 12)]
3 >>> maj_inventaire(stock, 'Carottes', 3)
4 >>> stock # il n'y a maintenant plus que 2 carottes !
5 [('Pommes', 10), ('Carottes', 2), ('Radis', 23), ('Lentilles', 534), ('Poivrons', 12)]
```

1. Écrivez une procédure `maj_inventaire(stock, produit, n)` qui **modifie** la liste `stock` en supprimant `n` occurrences du produit `produit`.
2. Modifiez la fonction pour qu'elle lance `IndexError` si le produit n'existe pas et lance `ValueError` s'il n'y a pas assez de produits disponibles.
3. Écrivez un programme `achat(stock, produit, n)` qui fait appel à la fonction `maj_inventaire` et qui rattrape les éventuelles exceptions pour afficher un des trois messages suivant « Merci pour votre achat. », « Produit inexistant. » et « Produit en quantité insuffisante. »

```
1 >>> achat(stock, 'Pommes', 1)
2 Merci pour votre achat.
3 >>> achat(stock, 'Poivrons', 2048)
4 Produit en quantité insuffisante.
5 >>> achat(stock, 'Aubergines', 10)
6 Produit inexistant.
```

Exercice 3 — Crible d'Ératosthène

Écrivez une fonction `ératosthène(n)` qui renvoie la liste des nombres premiers inférieurs ou égaux `n` en appliquant l'algorithme du crible d'Ératosthène. On commence par créer un tableau dont les indices vont de 0 à `n` et dont toutes les cases sont à `True`. Lorsque l'on affecte `False` à la case d'indice `i`, on dit que l'on barre le nombre `i`. L'idée est de barrer tous les nombres inférieurs à `n` qui sont multiples d'un autre nombre ($\neq 1$) strictement plus petit ; à la fin de l'algorithme, les seuls nombres restants non barrés sont alors les nombres premiers.

Algorithme : On barre 0 et 1 qui ne sont pas premiers, puis pour chaque entier de 2 à \sqrt{n} , si `k` n'est pas barré, on barre tous les multiples de `k` (autre que `k`).

Exemple : Pour $n=100$, on barre les multiples de 2 (4, 6, 8, etc.), puis ceux de 3 (6, 9, 12, etc.), on ignore 4 qui est déjà barré, puis on recommence avec les multiples de 5, etc. On s'arrête après avoir fait les multiples de 10, car $10 \times 10 \geq 100$.

```

1 def ératosthène(n):
2     # On initialise L à [0..n]
3     L=[True] * (n+1)
4
5     # Par définition 0 et 1 ne sont pas premiers
6     L[0] = False
7     L[1] = False
8     # On parcours les entiers de i à racine de n.
9     i=2
10    while i*i <= n:
11        # Si i est premier
12        # On barre tous les multiples de i : 2*i, 3*i, etc.
13        if L[i] != False:
14            for k in range(2*i, n+1, i): # ou range(i*i, n+1, i) car les m*i avec m<i
15                L[k] = False           # on déjà été barrés dans les tours précédents
16        i = i+1
17    # Pour chaque indice i :
18    # si L[k] = True -> k est premier
19    # Si L[k] = False -> k n'est pas premier
20    return [ k for k in range(len(L)) if L[k] == True]

```

Exercice 4 — Reconnaître une permutation

Définissez la fonction `est_permutation(L)` qui prend en argument une liste L de n entiers et qui renvoie `True` si L contient une et une seule fois tous les nombres de 0 à $n - 1$. Par exemple, `est_permutation([0,2,1])` renvoie `True` mais `est_permutation([0,1,0])` et `est_permutation([1,3,2])` renvoient `False`. Cherchez d'abord une solution quelconque, puis ajoutez la contrainte de parcourir une seule fois L (complexité linéaire).

Une première idée consiste à vérifier que tous les nombres de L sont uniques et bien compris entre 0 et $\text{len}(L)-1$.

```

1 def unique(x,L):
2     déjà_trouvé = False
3     for e in L:
4         if e==x and déjà_trouvé:
5             return False
6         elif e==x:
7             déjà_trouvé = True
8     return déjà_trouvé
9
10 def est_permutation(L):
11     n=len(L)
12     for e in L:
13         if not( 0<=e and e<n and unique(e,L)):
14             return False
15     return True

```

Une méthode plus rapide consiste à vérifier que l'on trouve tous les nombres de 0 à $\text{len}(L)-1$ au moins une fois. Forcément, il ne peut pas y avoir d'autres valeurs.

```

1 def est_permutation(L):
2     for i in range(len(L)):
3         if i not in L: # if not(i in L):
4             return False
5     return True

```

La méthode la plus efficace consiste à faire un seul parcours de la liste et d'utiliser une liste pour se rappeler si on ait déjà tombé sur un nombre. Si on parcourt les $\text{len}(L)$ de la liste en constatant qu'elles sont toutes uniques et comprise entre 0 et

Len(L)-1 c'est forcément que l'on a une permutation des éléments entre 0 et Len(L)-1.

```

1 def est_permutation(L):
2     n = len(L)
3     # Pour l'instant on en a trouvé aucun
4     trouvés = [False]*n
5     for e in L:
6         if not(0<=e and e<n):
7             return False
8         elif trouvés[e]: # si on l'a déjà rencontré
9             return False
10        else:
11            trouvés[e]=True
12    return True

```

Exercice 5 — Les neurones de la lecture

- Définissez une fonction `mélange(m)` qui renvoie un mot obtenu à partir de `m` en changeant aléatoirement l'ordre de ces lettres à l'exception de la première et de la dernière lettre. Par exemple, `mélange('cerise')` pourra renvoyer '`creise`' ou '`cierse`'. *Indication*: Vous pouvez utiliser `list` pour convertir une chaîne de caractères en liste et écrire vous-même la fonction inverse `liste_vers_chaine(L)`. La fonction `random.shuffle` permet de mélanger les éléments d'une liste.

```

1 from random import shuffle
2
3 def liste_vers_chaine(L):
4     s=""
5     for e in L:
6         s = s+e
7     return s
8
9 def mélange(mot):
10    n=len(mot)
11    if n<=2: # si le mot est très court, rien à faire !
12        return mot
13    else:
14        centre=list(mot[1:n-1]) # On aurait pu créer la liste avec une boucle
15        shuffle(centre) # on mélange
16        return mot[0] + liste_vers_chaine(centre) + mot[n-1]

```

- Définissez une fonction `liste_des_mots(m)` qui prend en argument un texte composé de mots et de ponctuation ('.,.; !?') et qui renvoie la liste des mots et ponctuations qui constituent ce texte. Par exemple,

```

1 >>> liste_des_mots('Un: deux? Trois!')
2 ['Un', ':', ' ', 'deux', '?', ' ', 'Trois', '!']

```

```

1 def liste_des_mots(m):
2     ponctuation= " :,.;!?'"
3     liste_mots = []
4     mot_en_cours = ''
5     for c in m:
6         if c in ponctuation:
7             # Si deux ponctuations se suivent on n'ajoute pas de mot vide.
8             if mot_en_cours != '':
9                 liste_mots.append(mot_en_cours)
10            # Dans tous les cas
11            liste_mots.append(c)
12            mot_en_cours = ''
13        else:
14            mot_en_cours = mot_en_cours + c
15        # On est arrivé à la fin.
16        # On ajoute le dernier mot à la liste
17        if mot_en_cours != '':
18            liste_mots.append(mot_en_cours)
19    return liste_mots

```

3. Définissez une fonction `mélange_texte(s)` qui renvoie ce même texte où chaque mot a été mélangé comme à la question 1.

```

1 def mélange_mots(m):
2     mots_normaux = liste_des_mots(m)
3     mots_mélangés = []
4     for mot in mots_normaux:
5         mots_mélangés.append(mélange(mot))
6     return liste_vers_chaine(mots_mélangés)

```

4. Mélangez l'article 1 de la déclaration universelle des droits de l'homme. Confirmez-vous ce que rapportent certains sites webs sur la capacité du cerveau humain à remettre dans le bon ordre les lettres en lecture rapide ?

Exercice 6 — Statistiques

Vous êtes libre pour gérer vous-même l'affichage (taille des marges, couleurs utilisés, etc) tant que le résultat est satisfaisant à vos yeux exigeants. Seule condition : le code doit fonctionner quelque soit la taille de la fenêtre.

1. Créer une fonction `effectif(L)` qui renvoie une liste de la forme `[(xmin,ymin), ... , (xmax,ymax)]`. Chaque couple `(x,y)` obtenu correspond à un entier `x` compris entre `xmin` et `xmax` et à son nombre d'appartitions `y` dans la liste `L`. Les valeurs `xmin` et `xmax` étant respectivement le minimum et le maximum de `L`.

```

1 >>> effectif([10,5,5,10,7,10,10,5])
2 [(5, 3), (6, 0), (7, 1), (8, 0), (9, 0), (10, 4)]

```

```

1 def effectifs(L):
2     Xmin=min(L)
3     Xmax=max(L)
4     E = [ (i,0) for i in range(Xmin,Xmax+1) ]
5     for e in L:
6         (ee,xx) = E[e-Xmin] # ee=e
7         E[e-Xmin] = (ee,xx+1)
8     return E

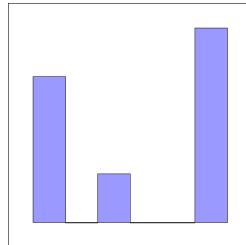
```

2. Avec Tk, faites un programme qui affiche l'histogramme correspondant à l'effectif. On prendra soin de n'afficher que les valeurs comprises entre le minimum et le maximum de `L` comme dans le graphisme ci-dessous correspondant à l'exemple précédent : `[10,5,5,10,7,10,10,5]`.

```

1 Titre = "Afficheur statistique"
2 Hauteur = 500
3 Largeur = Hauteur
4 marge=Hauteur/10
5 root = tk.Tk()
6 root.title(Titre)
7 Dessin = tk.Canvas(root,height=Hauteur,width=Largeur,bg="white")
8 Dessin.pack()
9 # Quelques variables globales
10 eff=effectifs(L)
11 Xmin=eff[0][0]
12 Xmax=eff[-1][0]
13 Ymax=max([y for (x,y) in eff])
14 Δx = (Largeur-2*marge)/(Xmax-Xmin+1)
15 Δy = (Hauteur-2*marge)/Ymax
16
17 def convertir_coodonnée(x,y):
18     X = marge+x*Δx
19     Y = Hauteur - (marge+y*Δy)
20     return (X,Y)
21
22 def afficher_rectangle(i,j):
23     p=convertir_coodonnée(i,0)
24     q=convertir_coodonnée(i+1,j)
25     Dessin.create_rectangle(p,q,fill="#9999FF")
26
27 for x in range(Xmin,Xmax+1):
28     (i,j)=eff[x-Xmin]
29     afficher_rectangle(i-Xmin,j)

```



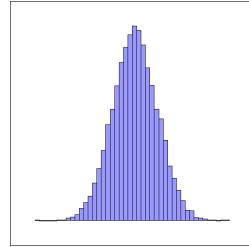
3. Écrivez une fonction `hasard(n)` qui simule n lancers de pièce et qui compte le nombre de piles.

```

1 # À mettre au début du fichier
2 from random import randint
3
4 def hasard(n):
5     i=0
6     for j in range(n):
7         i = i + randint(0,1)
8     return i

```

4. Utilisez cette fonction pour générer une liste de 1000 valeurs aléatoires correspondantes chacune à 500 lancers de pièces, puis, affichez son histogramme. Vous devez obtenir une courbe en cloche.



```
1 L = [ hasard(1000) for i in range(10000) ]
```