Correction du DS 2

Julien Reichert

Partie 1

La réponse aux questions de cours... figure dans le cours !

Partie 2

Exercice 2.1

```
def plus_grand_entier(n):
    chiffres = [0] * 10
    while n != 0:
        chiffres[n % 10] += 1
        n //= 10
    rep = 0
    for i in range(9, -1, -1):
        for _ in range(chiffres[i]):
            rep = 10 * rep + i
    return rep
```

Exercice 2.2

```
def amplitudes(l):
    dico = dict()
    for (s, n) in l:
        if s in dico:
            dico[s].append(n)
        else:
            dico[s] = [n]
    rep = []
    for s in dico:
        rep.append((s, max(dico[s]) - min(dico[s])))
    return rep
```

Exercice 2.3

```
def copieurs(lili):
    dico = dict()
    for ind in range(len(lili)):
        tu = tuple(lili[ind])
        if tu not in dico:
            dico[tu] = [ind]
        else:
            dico[tu].append(ind)
    reprep = []
    for tu in dico:
        if len(dico[tu]) > 1:
            reprep.append(dico[tu])
    return reprep
```

La version ci-avant serait rejetée par l'énoncé, mais on compense en signalant comment organiser le test d'égalité :

```
# [début du code intuitif]
    # ici boucle sur lili
    nouveau = True
    for seq in truc: # truc recensant les listes déjà rencontrées
        if len(seq) == len(lili[ind]):
            rate = False
            for i in range(len(seq)):
                if lili[ind][i] != seq[i]:
                    rate = True # et break éventuel
            if rate:
                # associer lili[ind] aux listes déjà rencontrées
                nouveau = False
    if nouveau:
        # ajouter lili[ind] aux listes déjà rencontrées
# [fin du code intuitive]
```

Exercice 2.4

```
def difference(l1, l2):
    dico1 = dict()
    dico2 = dict()
    for x in l1:
        if x not in dico1:
            dico1[x] = 1
        else:
            dico1[x] += 1
    for x in l2:
        if x not in dico2:
            dico2[x] = 1
        else:
            dico2[x] += 1
    for x in dico1:
        if x not in dico2 or dico1[x] != dico2[x]:
            return x
    raise ValueError("Ceci ne devrait pas se produire")
```

Exercice 2.5

L'énoncé impose tacitement une dichotomie...

```
def difference_croissante(l1, l2):
    deb = 0
    fin = len(l2)-1
    if fin == -1: # l2 vide, un seul élément dans l1
        return l1[0]
    if l1[-2] == l2[-1]: # c'est le dernier élément de l1 qui est absent de l2
        return l1[-1]
    while deb < fin: # On cherche le premier indice où l1 et l2 diffèrent...
        mil = (deb + fin)//2
        if l1[mil] == l2[mil]:
            deb = mil + 1
        else:
            fin = mil
    return l1[fin]
```

Sans croissance stricte, une dichotomie peut atterrir en plein milieu d'une séquence d'éléments identiques dans `l1` et `l2` sans savoir si le décalage est déjà présent ou non, une recherche séquentielle est alors recommandée.

Exercice 2.6

```
def cles_dico(dico, valeur):
    rep = []
    for cle in dico:
        if dico[cle] == valeur:
            rep.append(cle)
    return rep
```

Exercice 2.7

```
def creer_thd(capacite, f):
    rep = dict()
    rep["n"] = 0
    rep["f"] = f
    rep["tab"] = [[] for _ in range(capacite)]

def ajouter_thd(thd, cle, valeur):
    taille = thd["n"]
    capacite = len(thd["tab"])
    if capacite <= 10*(taille+1):
        nv_donnees = [[] for _ in range(capacite*2)]
        for i in range(capacite):
            for (cle, valeur) in thd["tab"][i]:
                nv_donnees[thd["f"](cle) % (2*capacite)].append((cle, valeur))
        thd["tab"] = nv_donnees
    position = thd["f"](cle) % capacite
    present = False
    for (cle2, valeur2) in thd["tab"][position]:
        if cle == cle2:
            raise ValueError("Clé déjà présente") # ou remplacer valeur2 par valeur, au choix
    thd["tab"][position].append((cle, valeur))
    thd["n"] += 1
```

Les autres fonctions sont plus faciles que l'ajout, donc on ne s'embêtera pas à les écrire.

Partie 3

Pour toute cette partie, on suppose `import random as rd` écrit avant les fonctions.

Exercice 3.1

```
def tirage_au_sort(participants):
    partic = participants[:]
    rd.shuffle(partic)
    rep = dict()
    for i in range(len(participants)):
        rep[participants[i]] = partic[i]
    return rep
```

Exercice 3.2

```
def existe_point_fixe(tirage):
    for p in tirage:
        if tirage[p] == p:
            return True
    return False
```

Exercice 3.3

```
def existe_transposition(tirage):
    for p in tirage:
        if tirage[p] != p and tirage[tirage[p]] == p:
            return True
    return False
```

Exercice 3.4

```
def est_cycle(tirage):
    n = len(tirage)
    assert n > 0
    p = list(tirage.keys())[0] # Il en faut bien un, d'autres méthodes existent pour en avoir un...
    p2 = p
    for i in range(n):
        p2 = tirage[p2]
        if p2 == p:
            return i == n-1 # Si non, cycle trop court
    raise ValueError("Ceci ne devrait pas se produire") # Car on a une permutation...
```

Exercice 3.5

```
def tirage_cyclique(participants):
    partic = participants[:]
    rd.shuffle(partic)
    rep = dict()
    for i in range(-1, len(partic)-1): # Astuce !
        rep[partic[i]] = partic[i+1]
    return rep
```

Exercice 3.6

```
def retrait(tirage, p):
    for p2 in tirage:
        if tirage[p2] == p:
            tirage[p2] = tirage[p]
    del tirage[p]
```

Exercice 3.7

La recherche de la clé correspondant à la valeur p peut nécessiter de tester toutes les clés pour enfin trouver la bonne.
En indexant dans les deux sens, cela devient immédiat (ce sera fait en bonus après la correction de la question).

```
def cree_double_dictionnaire(tirage):
    rep = (dict(), dict()) # ne pas mettre tirage en premier élément, ce ne serait pas une copie !
    for p in tirage:
        rep[0][p] = tirage[p]
        rep[1][tirage[p]] = p
    return rep

def retrait2(dd, p):
    p1 = dd[1][p] # qui offre à p
    p2 = dd[0][p] # à qui p offre
    dd[0][p1] = p2 # était p
    dd[1][p2] = p1 # idem
    del dd[0][p]
    del dd[1][p]
```

Exercice 3.8

```
SELECT MAX(prix) FROM OBJETS
```

Exercice 3.9

```
SELECT AVG(prix) FROM OBJETS JOIN CADEAUX ON id_objet = cadeau
```

Exercice 3.10

```
SELECT MIN(prix) FROM OBJETS JOIN CADEAUX ON id_objet = cadeau WHERE satisfaction
```

Exercice 3.11

Version tenant compte des égalités quand même :

```
SELECT categorie FROM
(
  SELECT categorie, SUM(prix) AS prix_total FROM OBJETS JOIN CADEAUX ON id_objet = cadeau
  GROUP BY categorie
)
WHERE prix_total =
(
  SELECT SUM(prix) FROM OBJETS JOIN CADEAUX ON id_objet = cadeau
  GROUP BY categorie
  ORDER BY SUM(prix) DESC LIMIT 1
)
```

Exercice 3.12

```
SELECT COUNT(*) FROM CADEAUX
JOIN PARTICIPANTS AS P1 ON participant = P1.id_participant
JOIN PARTICIPANTS AS P2 ON destinataire = P2.id_participant
WHERE P1.genre='F' AND P2.genre='M'
```

Exercice 3.13

```
SELECT COUNT(*) FROM PARTICIPANTS
JOIN CADEAUX AS C1 ON C1.participant = id_participant
JOIN CADEAUX AS C2 ON C2.destinataire = id_participant
JOIN OBJETS AS O1 ON O1.id_objet = C1.cadeau
JOIN OBJETS AS O2 ON O2.id_objet = C2.cadeau
WHERE C1.categorie = C2.categorie
```

Exercice 3.14

```
SELECT MIN(-o_m_r) AS max_recu_moins_offert, MAX(o_m_r) AS max_offert_moins_recu FROM
(
  SELECT O1.prix - O2.prix AS o_m_r FROM PARTICIPANTS
  JOIN CADEAUX AS C1 ON C1.participant = id_participant
  JOIN OBJETS AS O1 ON O1.id_objet = C1.cadeau
  JOIN CADEAUX AS C2 ON C2.destinataire = id_participant
  JOIN OBJETS AS O2 ON O2.id_objet = C2.cadeau
)
```

Exercice 3.15

```
def sac_a_dos(s, l):
    n = len(l)
    memoire = [[0 for _ in range(s+1)] for _ in range(n+1)]
    for i in range(n): # incorporer l'objet i, réponse en ligne i+1
        (prix, valeur) = l[i]
        for j in range(s+1): # voir pour un budget de j et les objets actuels le maximum de satisfaction
            memoire[i+1][j] = memoire[i][j] # pour le moment, on ne prend pas l'objet i
            if j >= prix and memoire[i+1][j] < memoire[i][j-prix] + valeur:
                memoire[i+1][j] = memoire[i][j-prix] + valeur
    return memoire[n][s]
```

Exercice 3.16

```
def sous_somme(s, l):
    n = len(l)
    possibles = { 0 : True } # Ou une liste de s+1 booléens...
    for i in range(n):
        possiblesbis = possibles.copy() # Sinon on mute un dictionnaire en plein parcours...
        for v in possibles:
            if v + l[i] <= s and v + l[i] not in possiblesbis:
                possiblesbis[v + l[i]] = True
        possibles = possiblesbis
    return s in possibles
```