Informatique - TP6 - Listes

Exercice 6.1

```python
def get_number(p: str)-> float:
    '''
    Demande à l'utilisateur de saisir un nombre.
    Si la valeur saisie n'est pas un nombre, l'utilisateur
    doit entrer une nouvelle valeur.
    '''
    ok = False
    while not ok:
        s = str(input(p))
        try:
            f = float(s)
            ok = True
        except:
            print("Saisissez un nombre !")
    return f

def get_numbers(n: int)-> list[float]:
    '''
    Appelle get_number() n fois et renvoie une liste de n nombres.
    '''
    pad = len(str(n))
    numbers = []
    for i in range(1, n+1):
        numbers.append(get_number(f"Nombre {i: >{pad}}/{n} : "))
    return numbers

n = int(input("Nombre d'éléments dans la liste : "))
numbers = get_numbers(n)
print(numbers)
```

Exercice 6.2

6.2.1 Saisie des notes

Afin de gérer le cas où l'utilisateur saisirait une note non conforme (non convertible en *float* ou non comprise dans l'intervalle $[0, 20]$), on fait appelle au traitement des erreurs.

```python
def get_note(p: str)-> float:
    ok = False
    while not ok:
        s = str(input(p))
        try:
            f = float(s)
            if f < 0 or f > 20:
                raise ValueError("Note invalide.")
```

```
        except:
            print("Saisissez une note entre 0 et 20 !")
        else:
            ok = True
    return f

def get_notes(n: int)-> list[float]:
    pad = len(str(n))
    notes = []
    for i in range(1, n+1):
        notes.append(get_note(f"Note {i: >{pad}}/{n} : "))
    return notes
```

6.2.2 Statistiques

L'énoncé suggère l'unicité de la meilleure note (et donc l'usage de `notes.index(M)`) mais plusieurs élèves peuvent avoir la note maximale ; on est donc obliger de parcourir la liste pour les identifier et on utilisera une liste pour les stocker.

```python
def get_stats(notes: list[float])->dict:
    if len(notes) == 0:
        return {
            "mean" : None,
            "max"  : None,
            "good" : None,
            "best" : None
        }
    else:
        # Moyenne de la classe
        m = sum(notes) / len(notes)
        # Note maximale
        M = max(notes)
        # Nombre d'élèves au-dessus de la moyenne de la classe
        good = 0
        # Indices des élèves ayant la meilleure note
        best:list[int] = []

        # Détermination de `good` et `best``
        for i in range(0, len(notes)):
            if notes[i] >= m :
                good +=1
            if notes[i] == M:
                best.append(i)

        return {
            "mean" : m,
            "max"  : M,
```

```
            "good" : good,
            "best" : best
        }


def display_stats(stats: dict)->None:
    print("Moyenne       :", round(stats["mean"],1))
    print("≥ moyenne     :", stats["good"])
    print("Note maximale :", stats["max"])
    print("Meilleurs     :", ", ".join([str(s) for s in stats["best"]]))
```

NB : on utilise la compréhension de liste pour convertir les nombres en chaînes de caractères afin de pouvoir utiliser `join()` et éviter une boucle.

6.2.3 Programme

```python
n = int(input("Nombre de notes : "))
    notes = get_notes(n)
    stats = get_stats(notes)
    display_stats(stats)
```

Exercice 6.3

Il n'est pas nécessaire de constituer la liste pour l'afficher, il serait plus économique en ressources de l'afficher en même temps que le calcul si on n'en a pas l'usage après.

```python
def get_u_list(n: int) -> list[int]:
    u = [1,2]
    for i in range(2, n+1):
        u.append(5*u[i-1] + 10*u[i-2])
    return u

n = int(input("Nombre de membres (≥ 2) : "))
print(get_u_list(n))
```

Exercice 6.4

2 stratégies possibles :

- On copie la plus longue liste et on ajoute aux premiers éléments la valeur de ceux de la plus courte.
- On crée une nouvelle liste après avoir identifié la plus longue (et donc la plus courte) en ajoutant d'abord la somme des éléments de même rang puis les éléments restants de la liste la plus longue.

NB : `short = l1` ne crée pas une copie de `l1` mais une référence (comme un alias) vers `l1`; toute modification de `short` est alors une modification de `l1`. Par contre `r = l1.copy()` effectue une copie mais une copie dite *shallow* : cela signifie que les objets référencés dans `l1` (par exemple des listes) ne sont pas copiés et que leur modification est donc effectives dans `l1` et `r`.

```python
def LRES(l1: list[int], l2: list[int]) -> list[int]:
    # Résultat
    r: list[int]
```

```
    # Référence vers la liste la plus courte
    short: list[int]

    if len(l1) >= len(l2):
        r = l1.copy()
        short = l2
    else:
        r = l2.copy()
        short = l1

    for i in range(0, len(short)):
        r[i] += short[i]

    return r


def LRES_2(l1: list[int], l2: list[int]) -> list[int]:
    # Résultat
    r: list[int] = []
    # Référence vers la liste la plus longue
    long : list[int]
    # Référence vers la liste la plus courte
    short: list[int]

    if len(l1) >= len(l2):
        long = l1
        short = l2
    else:
        long = l2
        short = l1

    for i in range(0, len(short)):
        r.append(long[i]+short[i])

    r.extend(long[len(short):])
    return r
```

Exercice 6.6

```python
def moyenne(resultats, eleve: str) -> float:
    s = 0
    n = 0
    for resultat in resultats:
        if resultat[0] == eleve :
            s += resultat[2]
            n += 1
    if n> 0 :
```

```
9           return s / n
10      else:
11          return None
12
13  resultats = [
14      ["Bob", "Python", 11],
15      ["Zoe", "Python", 14],
16      ["Bob", "algebre", 10.5]
17  ]
18
19  print(moyenne(resultats, "Bob")) # 10.75
20  print(moyenne(resultats, "Zoe")) # 14.0
21  print(moyenne(resultats, "Doe")) # None
```

On peut également utiliser une compréhension de liste pour un code plus concis mais pas obligatoirement plus efficace :

```python
def moyenne(resultats, eleve: str) -> float:
    notes = [ r[2] for r in resultats if r[0] == eleve]
    if len(notes) > 0:
        return sum(notes) / len(notes)
    else:
        return None
```

Exercice 6.7

6.7.1

L'erreur de syntaxe nous indique une erreur dans l'opérateur : une espace s'est glissée entre le < et le =.

6.7.2

L'erreur nous indique un retour à la ligne invalide (2 fois la même erreur) : soit on met tout sur la même ligne soit on fait précéder le retour à la ligne d'un anti-slash `'\'` : `'affichage = affichage + ... + \'`

Il manque également un double apostrophe dans la dernière commande `print`: `print("La chaine \"", ch...)`

6.7.3

La variable `nbMots` est lue avant d'être initialisée. Il faut l'initialiser à 0 en tout début d'exécution, avant la boucle `while`.

6.7.4

La concaténation de chaînes ne convertit pas automatiquement les entiers en chaînes, il faut donc le faire expressément avec `str(mot.count(mot[0]))`.

6.7.5

Les deux boucles `while` sur `i` possèdent l'instruction d'incrémentation ; c'est donc vraisemblement la boucle sur `mot` qui ne vérifie jamais la condition de fin. En mettant `print(mot)` dans la boucle on

constate en effet que la variable `mot` n'est pas modifiée car la méthode `replace()` renvoie une nouvelle chaîne sans modifier la chaîne d'origine.

On corrige avec : `mot = mot.replace(mot[0], "")`

6.7.6

Le message d'erreur signifie qu'on essaye de lire un caractère au-delà de la longueur de la chaîne.

Il faut modifier les conditions de fin de `while` par `i < len(ch)` car les indices de caractères varient de 0 à `len(ch)-1`.

6.7.7

Lors de l'évalution d'une condition `a and b`, python n'évalue pas `b` si `a` est faux, ce qui fait qu'on essaie pas de lire `ch[i]` quand on est au-delà de la longueur de `ch`. Si on intervertit les conditions, on essaie de lire un caractère au-delà de la longueur; ce qui provoque une erreur.

6.7.8

Il faut initialiser `affichage` avant la première boucle `while`.

6.7.9 Code corrigé

```python
def compteMotsEtCaracteres(ch):
    i = 0
    nbMots = 0
    affichage = ""

    while i < len(ch):
        mot = ""
        #Récupération du mot courant
        while (i < len(ch) and ch[i] != " "):
            mot += ch[i]
            i += 1

        #Pour ne compter que les mots non vides
        if mot != "":
            nbMots += 1
            affichage += "Le mot \"" + mot + "\" contient : \n"

            #Pour compter les caractères du mot courant
            while (mot != ""):
                affichage += "  " + str(mot.count(mot[0])) + " fois le caractère '" + mot[0] + "'\n"
                mot = mot.replace(mot[0], "")
        i += 1

    if nbMots != 0:
        print("La chaine \"", ch, "\" contient : ", nbMots,
            " mots\n", affichage, sep="", end="")

compteMotsEtCaracteres("ananas poire kiwi")
```

Exercice 6.8

```python
def somme_des_entiers(n: int) -> int :
    s = 0
    for i in range(1, n+1):
        s += i
    return s

def get_integer() -> int:
    r = 0
    while r < 1:
        r = int(input("Entrez un entier strictement positif : "))
    return r

n = get_integer()
print(f"Somme des entiers de {1} à {n} :", somme_des_entiers(n))
print(f"Somme des entiers de {1} à {n} :", sum(range(1, n+1)))
```

Exercice 6.9

On propose 3 méthodes de calcul du produit :

1. la plus évidente qui multiplie tous les nombres de la suite ;
2. une évolution de la précédente qui s'arrête si le produit est nul ;
3. l'utilisation de `reduce()` avec une fonction lambda.

```python
from random import randint
from functools import reduce

def TabAlea(n, a, b) -> list[int]:
    return [ randint(a,b) for i in range(0,n) ]

def TabProduit_1(T) -> int:
    p = 1
    for i in range(0, len(T)):
        p *= T[i]
    return p

def TabProduit_2(T) -> int:
    p = 1
    i = 0
    while i<len(T) and p != 0 :
        p *= T[i]
        i += 1
    return p

def TabProduit_3(T) -> int:
    return reduce(lambda x,y: x*y, T)
```

Exercice 6.10

On peaufine l'affichage en calculant la taille maximale des entiers avec `ceil(log10(n * 7))`

```python
from math import ceil, log10

def display(n: int, p: int):
    l = [ 7 * i for i in range(1,n+1)]
    m = 0
    pad = ceil(log10(n * 7))
    for i in l:
        print(f"{i:{pad}d}", end="")
        if i % p == 0:
            m +=1
            print("")
        else:
            print(" ; ", end="")
    if i % p != 0:
        print("")
    print(f"Vous avez affiché {m} multiples de {p}")

n = int(input("Combien de multiples de 7 voulez-vous afficher ? "))
print("Vous irez à la ligne après chaque multiple de...")
p = int(input("(nombre strictement positif svp) : "))

display(n,p)
```

Plutôt que de parcourir la liste, on peut envisager de calculer le nombre d'éléments de chaque ligne et d'extraire les éléments correspondants et de les afficher après les avoir joints.

Le nombre d'éléments de chaque ligne est :

- p si p n'est pas un multiple de 7 (p et 7 premiers entre eux);
- $\frac{p}{7}$ si p est un multiple de 7.

```python
from math import ceil, log10

def display(n: int, p:int):
    l = [ 7 * i for i in range(1,n+1)]
    k: int  = p
    if p % 7 == 0:
        k = p // 7

    pad = ceil(log10(n * 7))
    for i in range(0, n, k):
        print( " ; ".join([f"{str(m):>{pad}}" for m in l[i : i+k] ]) )
    print(f"Vous avez affiché {n // k} multiples de {p}")

n = int(input("Combien de multiples de 7 voulez-vous afficher ? "))
print("Vous irez à la ligne après chaque multiple de...")
```

```python
p = int(input("(nombre strictement positif svp) : "))

display(n,p)
```

Exercice 6.12

On génère une liste de listes aléatoire qui prend comme paramètres :

- `n`: nombre de sous-listes,
- `l_max` : longueur maximale des sous-listes,
- `v_max` : valeur entière maximale des éléments des sous-listes.

Le fait de renvoyer la somme de la première liste de longueur maximale simplifie l'algorithme et de ne considérer que les sous-suites de longueur strictement supérieure à la dernière plus grande.

```python
from random import randint

def get_list_of_lists(n=5, l_max=6, v_max=9):
    r = []
    for i in range(0,n):
        l = [ randint(0, v_max) for j in range(0, randint(0, l_max)) ]
        r.append(l)
    return r

def get_sum_of_first_longest_sublist(lol):
    l = 0
    s = 0
    for sl in lol:
        if len(sl) > l:
            l = len(sl)
            s = sum(sl)
    return s

l = get_list_of_lists(5,5,9)
print(l)
print(get_sum_of_first_longest_sublist(l))
```

Exercice 6.13

6.13.1

C'est une très mauvaise façon de stocker les données…

On essaie d'aligner verticalement les prénoms et les notes pour faciliter la maintenance du code

```python
students = ["Florian","Antoine","Charles1","Robert","Charles2","Erwan","Jean","Xavier","Didie
notes    = [ 2       , 12      , 0        , 0       , 8        , 7     , 20    ,
11     , 20     , 0      , 12    ]
```

6.13.2 Calcul et affichage de la moyenne

```python
def moyenne(notes: list[float]) -> float:
```

```
    if len(notes) > 0:
        return sum(notes)/len(notes)
    else:
        return None

print("Moyenne : ", round(moyenne(notes), 1))
```

6.13.3 Affichage des résultats

```python
def print_results(students: list[str], notes: list[float])->None:
    w = max([len(s) for s in students])
    for i in range(0, len(students)):
        print(f"{students[i]:<{w}} : {notes[i]:4.1f} => {'recalé' if notes[i]<10 else 'admis'}")
```

Noter l'usage de l'opérateur ternaire dans `'recalé' if notes[i]<10 else 'admis'`.

6.13.4

```python
ok_students = []
ok_notes = []
ko_students = []
ko_notes = []

def split_ok_ko(students, notes):
    for i in range(0, len(students)):
        if notes[i]>= 10 :
            ok_students.append(students[i])
            ok_notes.append(notes[i])
        else:
            ko_students.append(students[i])
            ko_notes.append(notes[i])

```

6.13.5 Suppression des zéros

PIÈGE

Le code ci-dessous **NE FONCTIONNE PAS** pour plusieurs raison :

- lors de la suppression d'une note nulle à la position `p`, les éléments suivants sont décalés : l'élément `p+1` devient l'élément `p` et n'est pas examiné car `i` passe à `p+1` et teste alors l'élément qui était `p+2`.
- si on supprime des éléments, on va essayer d'accéder à des indices au-delà de la nouvelle longueur de la liste.

```python
# NE FONCTIONNE PAS
def bad_remove_zeroes(students, notes):
    for i in range(0, len(students)):
        if notes[i] == 0:
            notes.pop(i)
            students.pop(i)
```

L'usage d'une boucle `while` incrémentale avec des tests sur la longueur supprime les erreurs d'exécution liées à la longueur de liste mais ne résoud pas le premier problème :

```python
# NE FONCTIONNE PAS
def bad_remove_zeroes(students, notes):
    while i< len(students):
        if i < len(students) and notes[i] == 0:
            notes.pop(i)
            students.pop(i)
        i += 1
```

Solution 1

On parcourt la liste des notes dans l'ordre des indices décroissants, les suppressions n'influençant alors pas les indices qu'il nous reste à explorer.

```python
def remove_zeroes_backwards(students, notes):
    for i in range(len(students)-1, -1, -1):
        if notes[i] == 0:
            notes.pop(i)
            students.pop(i)
```

Solution 2

On peut également utiliser une boucle `while` en traitant l'élément qui remplace celui supprimé (éventuellemeent plusieurs fois) avant d'incrémenter l'indice.

```python
def remove_zeroes_forwards(students, notes):
    i = 0
    while i < len(students):
        while i < len(notes) and notes[i] == 0:
            notes.pop(i)
            students.pop(i)
        i += 1
```

Solution 3

Une autre solution consiste à stocker dans un premier temps les indices des éléments nuls puis à les supprimer en les décalant de 1 pour chaque suppression antérieure.

```python
def remove_zeroes_afterwards(students, notes):
    zeroes = []
    for i in range(0, len(students)):
        if notes[i] == 0:
            zeroes.append(i)
    for j in range(0, len(zeroes)):
        notes.pop(zeroes[j]-j)
        students.pop(zeroes[j]-j)
```

Solution 4

Une autre solution consiste à chercher les zéros et supprimer l'élément correspondant jusqu'à ce qu'il y en ait plus. `index()` produisant une erreur si l'élément est introuvable, on peut utiliser les

mécanismes de gestion d'erreur (`try...except...`) ou faire précéder l'appel d'un `count()` (solution 5, qui multiplie les recherches)

```python
def remove_zeroes_search(students: list[str], notes: list[float]):
    pos = 0
    while pos >=0:
        try:
            pos = notes.index(0, pos)
        except:
            pos = -1
        else:
            students.pop(pos)
            notes.pop(pos)
```

Solution 5

```python
def remove_zeroes_count_index(students: list[str], notes: list[float]):
    pos = 0
    while notes.count(0) > 0:
        pos = notes.index(0, pos)
        students.pop(pos)
        notes.pop(pos)
```

Programme résultant

```python
g_students = ["Florian","Antoine","Charles1","Robert","Charles2","Erwan","Jean","Xavier","Didi"]
g_notes    = [ 2       , 12      , 0        , 0      , 8        , 7     , 20    , 11     , 20     , 0     , 12    ]

ok_students = []
ok_notes = []
ko_students = []
ko_notes = []

def print_results()->None:
    for i in range(0, len(ok_students)):
        print(f"{ok_students[i]:<10} : {ok_notes[i]:4.1f} => admis")
    for i in range(0, len(ko_students)):
        print(f"{ko_students[i]:<10} : {ko_notes[i]:4.1f} => recalé")

def moyenne()-> float:
    n = len(ok_notes) + len(ko_notes)
    if n > 0 :
        return (sum(ok_notes) + sum(ko_notes))/ n
    else:
        return None

def split_ok_ko(students, notes):
```

```
    for i in range(0, len(students)):
        if notes[i]>= 10 :
            ok_students.append(students[i])
            ok_notes.append(notes[i])
        else:
            ko_students.append(students[i])
            ko_notes.append(notes[i])

def remove_zeroes(students, notes):
    for i in range(len(students)-1, -1, -1):
        if notes[i] == 0:
            notes.pop(i)
            students.pop(i)


split_ok_ko(g_students, g_notes)
remove_zeroes(ko_students, ko_notes)
print("Moyenne   : ", round(moyenne(), 1))
print_results()
```

6.13.6 Ajout de note

Python

```
...

def get_new_notes():
    resp = 1
    while resp == 1:
        resp = int(input("Voulez-vous saisir une nouvelle note (1: oui ; 0 : non): "))
        if resp != 0 :
            student = str(input("Etudiant : "))
            note = float(input("Note : "))
            if note < 10  and note > 0:
                ko_students.append(student)
                ko_notes.append(note)
            elif note >= 10 :
                ok_students.append(student)
                ok_notes.append(note)

split_ok_ko(g_students, g_notes)
get_new_notes()
remove_zeroes(ko_students, ko_notes)
print("Moyenne   : ", round(moyenne(), 1))
print_results()
```

Exercice 6.14

6.14.1

```python
def lireDistance():
    distance = [[0 for j in villes] for i in villes]
    print("Entrez les distances entre les villes")
    for i in range(0, len(villes)-1):
        print("")
        for j in range(i+1, len(villes)):
            distance[i][j] = int(input(f"    {villes[i]:>8} <-> {villes[j]:<8} : "))
            distance[j][i] = distance[i][j]
    return distance
```

6.14.2

Si on tiens pour acquis que le tableau est symétrique avec une diagonale nulle (ce qu'il est par construction), il suffit de tester les couples (i, j) avec $i < j$; l'inégalité devenant une égalité si k est égal à i ou j, on pourrait économiser 2 tests par couple (i, j).

```python
def check_consistency(d):
    pb = []
    for i in range (0, len(villes)):
        for j in range(i+1, len(villes)):
            for k in range(0, len(villes)):
                if d[i][j] > d[i][k] + d[k][j]:
                    pb.append([i,j,k])
    return pb
```

6.14.3

```python
villes = ["Auxerre", "Avallon", "Clamecy", "Joigny", "Migennes"]

def lireDistance():
    distance = [[0 for j in villes] for i in villes]
    print("Entrez les distances entre les villes")
    for i in range(0, len(villes)-1):
        print("")
        for j in range(i+1, len(villes)):
            distance[i][j] = int(input(f"    {villes[i]:>8} <-> {villes[j]:<8} : "))
            distance[j][i] = distance[i][j]
    return distance

def check_consistency(d):
    pb = []
    for i in range (0, len(villes)):
        for j in range(i+1, len(villes)):
            for k in range(0, len(villes)):
```

```
                if d[i][j] > d[i][k] + d[k][j]:
                    pb.append([i,j,k])
    return pb


def print_inconsistencies(pb):
    if len(pb) > 0:
        print(f"Il y a {len(pb)} incohérences : ")
        for p in pb:
            print(f"{villes[p[0]]} - {villes[p[1]]} - {villes[p[2]]}")
    else:
        print("Toutes les inégalités triangulaires sont respectées.")

distance = lireDistance()
pb = check_consistency(distance)
print_inconsistencies(pb)
```

On constate 3 triplets qui pose problème :

- Avallon - Migennes - Auxerre
- Avallon - Migennes - Clamecy
- Avallon - Migennes - Joigny

C'est donc la Avallon - Migennes qui est surévaluée, toute valeur entre 61 et 65 permet de respecter les inégalités triangulaires