Nom : Prénom :

CONCOURS BLANC INFORMATIQUE – TSI1 – 2022

Le seul langage de programmation autorisé dans cette épreuve est Python. Pour répondre à une question, il est possible de faire appel aux fonctions définies dans les questions précédentes.

Ce sujet utilise la syntaxe des annotations pour préciser le type des arguments et du résultat des fonctions à écrire. Ainsi

```
def maFonction(n:int, X:[float], c:str, u) -> (int, dict):
```

signifie que la fonction `maFonction` prend quatre arguments, le premier (`n`) est un entier, le deuxième (`X`) une liste de nombres à virgule flottante, le troisième (`c`) une chaine de caractères et le type du dernier (`u`) n'est pas précisé. Cette fonction renvoie un couple dont le premier élément est un entier et le deuxième un dictionnaire.
Il n'est pas demandé aux candidats de recopier les entêtes avec annotations telles qu'elles sont fournies dans ce sujet, ils peuvent utiliser des entêtes classiques.

Dans ce sujet, le terme « liste » appliqué à un objet Python signifie qu'il s'agit d'une variable de type list. Le terme « séquence » représente une suite itérable et indiçable, indépendamment de son type Python, ainsi un tuple d'entiers, une liste d'entiers et un vecteur d'entiers sont tous trois des « séquences d'entiers ».

Une attention particulière sera portée à la lisibilité, la simplicité et l'efficacité du code proposé. En particulier, l'utilisation d'identifiants significatifs, l'emploi judicieux de commentaires et le marquage des indentations seront appréciés.

I. Fonctions utilitaires

1. Écrire une fonction d'entête

```
def moyenne(X) -> float:
```

qui prend en paramètre une séquence de nombres et qui calcule la moyenne arithmétique de ces nombres. Cette fonction ne doit pas modifier le paramètre `X`.
Par exemple : `moyenne([1, 2, 3, 4]) -> 2.5`

2. Écrire une fonction d'entête

```
def variance(X) -> float:
```

qui calcule la variance d'une séquence de nombres, sans la modifier.
Par exemple : `variance([1, 2, 3, 4]) -> 1.25`

Pour rappel, la variance des n nombres $x_1, \ldots, x_n$ peut se calculer de deux manières :

a. $\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$ b. $\frac{1}{n}\sum_{i=1}^{n}x_i^2 - \bar{x}^2$ avec $\bar{x} = \frac{1}{n}\sum_{i=1}^{n}x_i$

3. Quelle est la complexité de la fonction `variance` si son implémentation reproduit la formule a. ?

4. Et en utilisant la formule b. ?

5. Écrire une fonction d'entête

```
def calculs(X) -> (float, float):
```

qui prend en paramètre une séquence de nombres, sans la modifier, et qui retourne à la fois la moyenne et la variance.
Par exemple : `calculs([1, 2, 3, 4])` -> `(2.5, 1.25)`
Vous ajouterez une assertion permettant de vous assurer que la séquence `X` ne contient que des nombres.
La fonction `calculs` devra utiliser les fonctions `moyenne` et `variance` précédemment définies.

6. L'exécution du code suivant lève une erreur.

```
notes = [5, 15, 20, 12, 3]
a = input("Que voulez-vous calculer ? Taper 0 pour la moyenne et 1
pour la variance.")
res = calculs(notes)[a]
print(res)
```

Pourquoi ? Comment la corriger ?
Rq : vous trouverez à la fin de l'énoncé un extrait de la documentation de la fonction `input`.

II Codage des nombres flottants

Les nombres en virgule flottante (format flottant) peuvent être vus comme l'équivalent informatique de la notation scientifique.
Supposons ici qu'un nombre est codé sur 16 bits qui se décomposent ainsi :
1 bit pour le signe, 6 bits pour l'exposant et 9 bits pour la mantisse.
La mantisse contient 1 bit caché dans le sens où on ne code pas le 1er bit du nombre qui est systématiquement « 1 ».
Pour coder l'exposant, on ajoute à sa valeur $2^5 - 1 = 31$.

Exemple :
dans ce codage, le nombre **2,5** va s'écrire : **1 100000 010000000**
En effet 2,5 > 0 ⇒ **1** pour le signe
$(2,5)_{10} = (10,1)_2 = (1,01)_2 \times 2^1$. L'exposant vaut donc 1+31=32 qui se code **100000** et la mantisse $(101)_2$ se code **010000000** (1er bit retiré).

7. Que vaut le nombre codé par 1 100010 100000000 ?

8. Quel est le nombre de chiffres significatifs décimaux maximum permis par ce codage ?

On définit une fonction `derive` permettant de calculer le nombre dérivé d'une fonction f en x :

```
def derive(f,x) :
    h = 10**-5
    der = (f(x+h)-f(x))/h
    return der
```

9. Que va afficher le code suivant si les flottants sont codés sur 16 bits comme défini précédemment ?

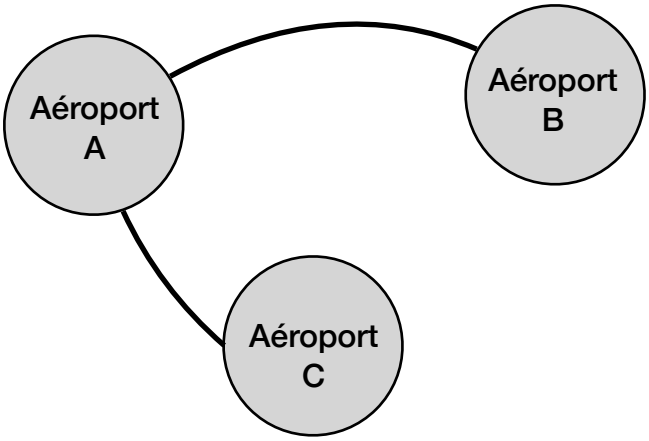
```
def f(x) :
    return x**2

print(derive(f,4))
```

III Graphes

On modélise par un graphe non-orienté un réseau aérien.
Les sommets sont les aéroports et les arêtes sont les liaisons aériennes entre eux.

10. Représenter la matrice d'adjacence correspondant à ce graphe en associant l'indice 1 à l'aéroport A, l'indice 2 à l'aéroport B et l'indice 3 à l'aéroport C.



Les données sources que l'on va utiliser répertorient tous les vols au décollage et à l'atterrissage dans 2500 aéroports. On les représente à l'aide d'une liste de tuples appelée `liste_vols`. Chacun de ces tuples est composée de 3 chaînes de caractères : la date, l'indicatif de l'aéroport de départ et l'indicatif de l'aéroport d'arrivée.
Exemple d'une ligne de `liste_vols` : `('15/7/2022','CDG','LAX')` pour un vol le 15 juillet 2022 entre Paris Charles-de-Gaulle et l'aéroport de Los Angeles.

La liste `Indic_ap` contient chacune des dénominations des aéroports (['CDG','LAX',...])

11. Construire à partir de la liste `Indic_ap` un dictionnaire `Dic_ap_indice` qui associe un entier différent entre 0 et 2499 à chacune des dénominations (les chaînes de caractères sont les clés et les entiers sont les valeurs).

On souhaite maintenant tirer une matrice d'adjacente M de `liste_vols` dont la valeur des éléments devra correspondre au nombre de liaisons directes entre deux aéroports :

- pour chaque ligne de `liste_vol` (décrivant un vol (date, apA, apB)), on ajoute 1 à la valeur située à l'intersection de la ligne correspondant à apA et de la colonne correspondant à apB (les correspondances aéroport ↔ indice étant données par `Dic_ap_indice`).
- Et comme on considère un graphe non orienté, il faut aussi ajouter 1 à la valeur symétrique dans la matrice.

12. Compléter le code ci-dessous visant à construire la matrice M en utilisant le dictionnaire Dic_ap_indice.

```
n = len(Indic_ap)
M = [[0 for i in range(n)] for j in range(n)]
for vol in liste_vols :
     date,A,B = vol
```

13. Écrire une fonction d'entête :

def existence_vol_direct(M,apA:str,apB:str,Dico:i) -> bool:

qui prend en paramètre la matrice d'adjacence précédente, les dénominations d'un aéroport A et d'un aéroport B ainsi qu'un dictionnaire équivalent à Dic_ap_indice (associant un indice à chaque dénomination) et qui renvoie True si une liaison directe existe entre les deux aéroports et False sinon.

On souhaite maintenant obtenir le nombre de lignes directes existantes depuis un aéroport A donné.

14. Comment appelle-t-on cette valeur pour le sommet représentant l'aéroport A dans le graphe ?

15. Compléter le code suivant afin que la fonction retourne bien le nombre cherché (Dico est à nouveau ici équivalent à Dic_ap_indice).

```
def nb_lignes_directes(M,apA,Dico):
    i = Dico[apA]
    nb = 0
```

On souhaite enfin utiliser la matrice d'adjacence pour résoudre un problème bien plus difficile : déterminer le nombre de changements d'avions nécessaires pour aller de l'aéroport A à l'aéroport B.

La matrice d'adjacente M indique le nombre de vols directs entre chaque paire d'aéroports. La matrice $M^2 = M \times M$ indique, elle, le nombre de vols de longueurs 2 (vols indirects avec une escale, soit deux avions) entre ces mêmes paires d'aéroports, et M^n indique le nombre de vols de longueur n.

On définit alors la fonction suivante :

```
def fonction(M,apA,apB,Dico) :
    m = 1
    n = len(M)
    N = M.copy()
    while m <= 10 :
        P = [[0 for i in range(n)] for j in range(n)]
        if N[Dico[apA]][Dico[apB]] > 0 :
            return m
        for i in range(n) :
            for j in range(n) :
                s = 0
                for k in range(n) :
                    s += N[i][k]*M[k][j]
                P[i][j] = s
        N = P
        m += 1
    return float('inf')
```

16. Donner l'expression mathématique de N à la fin de chaque itération de la boucle `while` de l'algorithme (c'est un invariant de l'algorithme) ?

17. Que représente le `m` retourné par la fonction ?

18. À quoi sert la condition `m<=10` ?

19. Quelle est la complexité de la fonction `fonction` ?

20. S'il avait fallu obtenir le même résultat directement à partir de `liste_vols`, en testant toutes les combinaisons d'enchaînements de vols et en choisissant ensuite la plus petite, quelle aurait été la complexité ?

Extrait de la documentation de la fonction input :

> La fonction `input()` accepte un argument qui correspond au texte que l'on souhaite afficher à l'utilisateur.
> La saisie de l'utilisateur est ensuite retournée par la fonction sous la forme d'une chaîne de caractères.
> Exemple :

```
>>> input("Quel âge avez-vous ?")
Quel âge avez-vous ? 25
'25'
```