+1/1/60+

Programmation impérative en Python — SPUF21

Année 2022-2023 — Seconde session

Nom : .....

Prénom : .....

Numéro d'étudiant : .....

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

Durée : 2 heures.

Aucun document n'est autorisé. L'usage de la calculatrice ou de tout autre appareil électronique est interdit.

Les exercices sont indépendants. Au sein d'un même exercice, vous pouvez utiliser les variables et fonctions des questions précédentes, même si vous n'avez pas su les faire ; chaque question est donc indépendante.

À part les méthodes et fonctions de base, vous n'avez pas le droit d'utiliser les fonctions et les méthodes « avancées », sauf si l'énoncé vous conseille l'utilisation de certaines d'entre elles.

```
# Fonctions autorisées
range(…) len(…)
print(…) int(…)

# Méthodes et mots-clés autorisés
L.append(x)
x in L
```

```
# Par exemple les méthodes et fonctions suivantes sont entre autres interdites
max(…) min(…) sum(…) abs(…) eval(…)
s.split(…) s.index(…) L.extend(…)

# Vous n'avez pas le droit d'utiliser des compréhensions ou des slices
# À la place vous devez utiliser des boucles.
[ x for x in range(L) ]
chaine[début:fin:pas]
```

Exercice 1 Questions rapides . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

1. Construire avec une boucle une liste `L = [0.0, 0.5, … , 19.5, 20.0]` contenant toutes les valeurs de 0 à 20 avec un pas de $\frac{1}{2}$.

```
L=[]
for i in range(0,41): # Les pas de 0.5 sont interdits !
    L.append(i/2)
```

2. On se donne une chaîne *s*. En utilisant une boucle `for`, définir un ensemble `E` contenant tous les caractères de `s`. On n'aura évidemment pas le droit de faire `E = set(s)`.

```
E = set()  # Attention E = {} définit un dictionnaire !
for c in s:
    E.add(c)
```

3. On définit ci-dessous une fonction `construire_matrice(n)` qui renvoie une matrice nulle de `n` lignes et `n` colonnes.

```
def construire_matrice(n):
    ligne = [0] * n
    M = []
    for i in range(n):
        M.append(ligne)
    return M
```

En utilisant cette fonction `construire_matrice(n)`, on initialise une matrice `M` et on la modifie. Que vaut `M` après la modification ? Justifier.

```
>>> M = construire_matrice(3)
>>> M
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> M[1][1] = 5
```

```
M vaut [ [0,5,0] , [0,5,0] , [0,5,0] ].
En effet, M contient 3 fois la même liste (même adresse en mémoire),
donc si on en modifie une, on modifie automatiquement les autres.
```

Exercice 2 Arithmétique . . . . . 5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5

1. Écrire une fonction `liste_diviseur(n)` qui renvoie la liste des diviseurs de `n` (`1` et `n` compris).

```
def liste_diviseur(n):
    L=[]
    for i in range(1,n+1):
        if n%i==0:
            L.append(i)
    return L
```

2. Écrire une fonction `moyenne_nombre_diviseurs(N)` qui renvoie la moyenne des nombres de diviseurs de chaque entier de `1` à `N`. Par exemple « `1` » a **1** diviseur, « `2` » et « `3` » ont **2** diviseurs chacun et « `4` » en a **3**. Ainsi, pour `N = 4`, `moyenne_nombre_diviseurs(N)` renverra la moyenne de **1**, **2**, **2** et **3** c'est-à-dire `2.0`.

```
def moyenne_nombre_diviseur(N):
    s=0
    for n in range(1,N+1):
        s = s+len(liste_diviseur(n))
    return s/N
```

3. En mathématiques, un entier n est dit parfait s'il est égal à la somme de ses diviseurs **stricts** (tous ses diviseurs, sauf lui-même). Par exemple 6 est parfait car c'est la somme de ses trois diviseurs : $1+2+3=6$. Écrire une fonction `est_parfait(n)` qui renvoie `True` si `n` est parfait et `False` sinon. On rappelle que la fonction `sum` est interdite.

```
def est_parfait(n):
    s=0
    L = liste_diviseur(n)
    for i in range(len(L)-1):
        s = s+L[i]
    return s==n
```

4. Écrire un programme `nombres_parfaits(k)` qui affiche les `k` premiers nombres parfaits.

```
def nombres_parfaits(k):
    i=0
    n=0
    while n<k:
        if est_parfait(i):
            print(i)
            n = n+1
        i=i+1
```

5. On définit par récurrence la suite u_n par

$$\begin{cases} u_0 = u_1 & = 1 \\ u_{n+1} & = k \times u_{k-1} \text{ où } k \text{ est le nombre de diviseurs de } n+1 \end{cases}$$

Écrire la fonction `u(n)` correspondante, calculant u_n de manière récursive.

```
def u(n):
    if n==0 or n==1:
        return 1
    else:
        k = len(liste_diviseur(n))
        return k * u(k-1)
```

Exercice 3 Génération de liste aléatoire . . . . . 3,5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5

1. Écrire une fonction `liste(n)` qui prend en paramètre un entier `n` > 0 et renvoie une liste contenant les nombres de 1 à `n` compris, dans l'ordre.

```
>>> liste(7)
[1, 2, 3, 4, 5, 6, 7]
```

```
def liste(n):
    L=[]
    for i in range(1,n+1):
        L.append(i)
    return L
```

2. Écrire une fonction `tirage_aléatoire(L)` qui tire un indice aléatoire et renvoie l'élément de la liste correspondant. La fonction devra en outre échanger la dernière case du tableau avec celle choisie puis supprimer le dernier élément avec la méthode `L.pop()`. On utilisera la fonction `randint(a,b)` qui renvoie un nombre aléatoire entre `a` et `b` compris.

```
>>> L = [1,2,3,4,5]
>>> tirage_aléatoire(L)
3
>>> L
[1, 2, 5, 4]
```

Dans l'exemple ci-dessus, la fonction `randint` nous a renvoyé l'indice 2 (correspondant à la valeur 3 dans la liste).

```
L = [1, 2, 3, 4, 5] # avant
L = [1, 2, 5, 4, 3] # randint renvoie l'indice 2, on échange donc L[2] et L[4]
L.pop() # On supprime le dernier élément de L
L = [1, 2, 5, 4] # la liste finale obtenue
```

```
def tirage_aléatoire(L):
    n = len(L)-1
    i = randint(0,n)
    tmp = L[i]
    L[i] = L[n]
    L[n] = tmp
    L.pop()
    return tmp
```

3. En utilisant les deux fonctions précédentes, écrire une fonction `aléatoire(n)` qui renvoie les nombres de 1 à `n` (inclus) dans une liste et dans un ordre aléatoire.

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

```
def aléatoire(n):
    L = liste(n)
    M = []
    while  L  != []:
        M.append(tirage_aléatoire(L))
    return M
```

Exercice 4 CSV — première partie . . . . . 6,5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5 ☐5,5 ☐6 ☐6,5

Un fichier CSV est un fichier permettant de réprésenter un tableau à deux dimensions sous format texte. Les colonnes sont séparées par des symboles, typiquement `","` ou `";"`. Ci-contre, un exemple de fichier au format CSV de quatre colonnes.

```
Alice;14;5;AJ
Bob;20;17;ACQ
Charlie;15;9;ACQ
Davidou;2;3;AJ
```

1. On vous donne la fonction suivante avec un exemple d'utilisation ci-dessous. Contient-elle des erreurs ? Donnez la raison de la ligne 5.

```
1 def lire_fichier(nom_fichier):
2     f = open(nom_fichier, 'r', encoding='utf-8')
3     résultat = []
4     for ligne in f:
5         chaîne = ligne[:-1]
6         résultat.append(chaîne)
7     return résultat
```

```
1 >>> lire_fichier("notes.csv")
2 ['Alice;14;5;AJ', 'Bob;20;17;ACQ', 'Charlie;15;9;ACQ', 'Davidou;2;3;AJ']
```

La ligne 5 sert à éliminer les symboles "\n" de fin de ligne

Il manque le f.close()

2. Écrire une fonction `nombre_colonnes(ligne, symbole)` qui donne le nombre de colonnes dans la chaîne `ligne` avec `symbole` comme séparateur.

```
1 >>> nombre_colonnes("Alice;14;5;AJ" , ";")
2 4
```

```
def nombre_colonnes(chaîne, symbole):
    n = 1
    for caractère in chaîne:
        if caractère == symbole:
            n = n+1
    return n
```

3. On cherche à savoir si un fichier CSV est valide. Pour cela il faut que chaque ligne contienne le même nombre de colonnes. Écrire une fonction `fichier_valide(liste_ligne, symbole)` qui renvoie `True` si le fichier est valide et `False` sinon. Si la liste en paramètre est vide, la fonction lèvera une erreur comme dans l'exemple ci-dessous.

```
>>> L = ["Alice;14;5;AJ","Bob;20;17;ACQ","Charlie;15;9;ACQ","Davidou;2;3;AJ"]
>>> fichier_valide(L, ";") # Toutes les lignes ont 4 colonnes
True
>>> fichier_valide(["Alice;14;5;AJ","Bob;20;17;33;ACQ"], ";") # 4 et 5 colonnes
False
>>> fichier_valide([],";")
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "<console>", line 3, in fichier_valide
ValueError:  Fichier vide
```

```
def fichier_valide(liste_ligne, symbole):
    if liste_ligne == []:
        raise ValueError("Fichier vide")
    n = nombre_colonnes(liste_ligne[0], symbole)
    for ligne in liste_ligne:
        if nombre_colonnes(ligne, symbole) != n:
            return False
    return True
```

4. Écrire une fonction `séparer(ligne,symbole)` qui découpe une chaîne de caractères selon le caractère `symbole` et renvoie la liste obtenue. On ne pourra pas utiliser la méthode `ligne.split(symbole)` qui fait sensiblement la même chose.

```
>>> séparer("Alice,13.5,alice.miroir@paysdesmerveilles.com", ",")
['Alice', '13.5', 'alice.miroir@paysdesmerveilles.com']
>>> séparer("Bob;11;11;15;13;bob.y@lapointe.fr", ";")
['Bob', '11', '11', '15', '13', 'bob.y@lapointe.fr']
```

```
def séparer(chaîne,symbole):
    res = []
    courant = ""
    for c in chaîne:
        if c==symbole:
            res.append(courant)
            courant =""
        else:
            courant = courant + c
    if courant != "":
        res.append(courant)
    return res
```

5. Écrire une fonction `joindre(liste, symbole)` qui fait l'inverse de la fonction précédente : la fonction prend une liste de chaînes et renvoie une unique chaîne reliant les précédentes avec le caractère `symbole`. On n'aura pas le droit d'utiliser la méthode `symbole.join(liste)` qui fait sensiblement la même chose.

```
>>> joindre(['Alice', '13.5', 'alice.miroir@paysdesmerveilles.com'], ";")
'Alice;13.5;alice.miroir@paysdesmerveilles.com'
>>> joindre(['Bob', '11', '11', '15', '13', 'bob.y@lapointe.fr'], ",")
'Bob,11,11,15,13,bob.y@lapointe.fr'
```

```
def joindre(liste, symbole):
    if liste == []:
        return ""
    else:
        res = liste[0]
        for i in range(1,len(liste)):
            res = res + symbole + liste[i]
        return res
```

Exercice 5 CSV — seconde partie . . . . . 2 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2

Cet exercice est la suite du précédent.

1. Écrire une fonction `inverser(liste)` qui renvoie la liste donnée en paramètre mais en miroir.

```
>>> inverser([1,2,3])
[3, 2, 1]
>>> inverser(['Bob', '11', '11', '15', '13', 'bob.y@lapointe.fr'])
['bob.y@lapointe.fr', '13', '15', '11', '11', 'Bob']
```

```
def inverser(liste):
    res = []
    for i in range(len(liste)-1, -1, -1):
        res.append(liste[i])
    return res
```

2. Il s'agit maintenant de combiner les fonctions vues dans cet exercice et le précédent. Écrire une fonction `inverser_colonnes(nom_fichier)` qui lit un fichier CSV et l'**affiche** ligne par ligne en inversant l'ordre des colonnes.

```
Alice;14;5;AJ
Bob;20;17;ACQ
Charlie;15;9;ACQ
Davidou;2;3;AJ
```

```
>>> inverser_colonnes("notes.csv")
AJ;5;14;Alice
ACQ;17;20;Bob
ACQ;9;15;Charlie
AJ;3;2;Davidou
```

```
def inverser_colonnes(nom_fichier):
    liste = lire_fichier(nom_fichier)
    for ligne in liste:
        print(joindre(inverser(séparer(ligne, ";")),";"))
```