E.N .S.G. 1ère année *NOM :*

Epreuve d'Informatique – 11 janvier 2022 - 1h

Ecrire directement sur les feuilles de l'énoncé.
Rendre toutes les feuilles (même vierges) avec votre nom dessus.
Documents autorisés : notes de cours et T.D., polys de cours.

Vous veillerez à soigner la présentation des programmes et à choisir des noms de variables « significatifs ». On ne demande cependant pas d'écrire les docstrings des fonctions.

Le nombre de lignes ou le temps précisé permet d'avoir une idée de la complexité du programme demandé, mais il n'est qu'approximatif et peut varier d'une solution à l'autre.

Exercice I - Pythonesquerie

Compétences évaluées :
AF2 - Compétences en programmation dans le langage de programmation Python

(*environ 10 minutes*)
Soit la fonction mystérieuse suivante :

```
7   def enigme(chaine, car, ind):
8       """
9
10      @param chaine:
11      @param car:
12      @param ind:
13      @return:
14
15
16
17
18      """
19      n = 0
20      for i in range(ind, len(chaine)):
21          if chaine[i] != car:
22              return n
23          n += 1
24
25      return n
```

1) Que retourne la fonction *enigme* ?
Pour comprendre ce qu'elle fait, on peut la tester avec les paramètres :
enigme("aabb", 'a', 0)
enigme("aabb", 'b', 2)
enigme("aabcb", 'b', 2)
enigme("caabb", 'a', 0)
Pour expliquer ce qu'elle retourne, on vous demande de compléter la *docstring* de cette fonction, vous pouvez le faire directement au niveau du code de la fonction dans l'énoncé.

2) Soit la fonction *mystere* ci-dessous, elle vérifie qu'une chaîne donnée vérifie la syntaxe attendue (on ne vous demande pas d'expliquer cette syntaxe) :

```
26
27  def mystere(ch, car1, car2):
28
29      n1 = enigme(ch, car1, 0)
30      n2 = enigme(ch, car2, n1)
31
32      if (n1 + n2) != len[ch]:
33          return False
34
35      return True
36
```

Quand je lance cette fonction, j'obtiens un message d'erreur :

```
ch = "aabbb"
print(mystere(ch, 'a', 'b'))
```

```
Traceback (most recent call last):
  File "C:/Users/fay7/Nextcloud/1 - ENSG1A/ENSG1A-2021-2022/Semestre 5/Evaluations/Examen info 2022-01-11/Corrections/mystere.py", line 48, in <module>
    print(mystere(ch, 'a', 'b'))
  File "C:/Users/fay7/Nextcloud/1 - ENSG1A/ENSG1A-2021-2022/Semestre 5/Evaluations/Examen info 2022-01-11/Corrections/mystere.py", line 32, in mystere
    if (n1 + n2) != len[ch]:
TypeError: 'builtin_function_or_method' object is not subscriptable
```

Comment corriger l'erreur ?

Est-ce que j'aurais pu détecter l'erreur avant l'exécution du programme ?

3) La fonction *mystere* précédente est appelée dans la fonction principale ci-dessous :

```
36
37  def fonction_principale(lchaines, car1, car2):
38      i = 0
39      while i<len(lchaines):
40          if mystere(lchaines[i], car1, car2):
41              i+=1
42          else:
43              lchaines.remove(lchaines[i])
```

Cette fonction supprime de la liste *lchaines* les chaines non conformes à la syntaxe attendue. Entourez parmi les propositions suivantes pour lancer l'exécution de cette fonction et afficher la liste mise à jour, la(les) proposition(s) correcte(s).
Certaines de ces propositions ne provoquent pas d'erreur mais ne sont pas "cohérentes" avec la définition de la fonction.

On dispose de la liste : *ltests = ["aabbb", "abbb", "aabb", "bba", "a", "bb", "aabba", "aybb"]*

1.
```
print(fonction_principale(ltests, 'a', 'b'))
```
2.
```
fonction_principale(ltests, 'a', 'b')
print(ltests)
```
3.
```
c1 = 'a'
c2 = 'b'
fonction_principale (ltests, c1, c2)
print(tab)
```
4.
```
lchaines = fonction_principale(ltests, 'a', 'b')
print(lchaines)
```

Exercice II - Calcul d'une suite

Compétences évaluées :

AF 1 - Compétences méthodologiques et de conceptualisation (algorithmique)

Soit la suite u suivante qui converge vers 1 :

$$u_i = \sqrt{\frac{1+ u_{i-1}{}^2}{2}}$$

avec u_0 compris entre -1 et 1

1) Ecrire une fonction ***suiteU*** qui, pour un u_0 et un *epsilon* donnés en paramètres, retourne le premier *n* et la valeur de *u* correspondante qui permette d'atteindre une valeur de la suite *u* proche de 1 à *epsilon* près. (*7 lignes environ – 10 minutes*).

2) Ecrire le programme permettant de récupérer et d'afficher les résultats pour $u_0 = 0.5$ et *epsilon* = 0.001

Exercice III - Minéraux

Compétences évaluées :
AF1 - Compétences méthodologiques et de conceptualisation (algorithmique)
AF2 - Compétences en programmation dans le langage de programmation Python

Dans l'objectif de manipuler des minéraux, on suppose que ces derniers sont décrits sous la forme d'une liste de couples "*symbole chimique-répétition"* séparés par un "-".

Exemple : ["*Na-2", "Al-1", "Si-3", "H-2", "O-10"*]

1) Ecrire une fonction *compose* (*environ 6 lignes – 8 minutes*) :

- qui reçoit en données :
  - un symbole d'élément chimique (*ex : "Ca"*)
  - un minéral sous forme d'une liste de composants sous forme de couples ("*symbole-répétition"*)
- et retourne *True* si l'élément chimique apparaît la liste des composants du minéral

Exemple : compose("Ca", ["Ca-1", "S-1", "O-4", "H-2", "O-1"]) retourne True

compose("C", ["Ca-1", "S-1", "O-4", "H-2", "O-1"]) retourne False

- *Indication : on peut utiliser la fonction split :*

texte.split(sep) découpe texte en utilisant le séparateur sep
elle retourne une liste de sous-chaînes de caractères

exemple : texte = "toto:12:321:blabla:313"

texte.split(":") → ["toto", "12", "321", "blabla", "313]

2) Soit un dictionnaire *mineraux* tel que : mineraux[nom_mineral] = composition chimique

Exemple :

*mineraux = {"**Gypse**" : ["Ca-1", "S-1", "O-4", "H-2", "O-1"], "**Orthose**" : ["K-1", "Al-1", "Si-3", "O-8"],*
*"**Calcite**" : ["Ca-1", "C-1", "O-3"], "**Diamant**" : ["C-1"], "**Beryl**" : ["Be-3", "Al-2", "Si-6", "O-18"]}*

En utilisant la fonction précédente, écrire une fonction *inventaire (environ 6 lignes – 8 minutes)* :

- qui reçoit en données :
  - un dictionnaire de minéraux
  - un symbole chimique sous forme de chaînes de caractères
- et retourne la liste des minéraux dont la composition chimique intègre ce symbole

  Exemple : pour dico_mineraux et le symbole "Ca"
  la fonction inventaire retourne la liste ["Gypse", "Calcite"]

3) Ecrire une fonction *ajoute_mineral* (*environ 5 lignes – 5 minutes*) :

- qui reçoit en entrée :
  - un dictionnaire des minéraux
  - un nom de minéral, exemple "*Sylvanite*"
  - une liste de composants, exemple *[("Ag", "1"), ("Au", "1"), ("Te", "4")]*
- et qui ajoute ce minéral dans le dictionnaire : exemple *"Sylvanite" : ["Ag-1", "Au-1", "Te-4"*)

Indications : Rappelez-vous les règles de modification d'un dictionnaire dans une fonction

Exercice IV - Carré magique

(environ *15 minutes*)

Compétences évaluées :

AF1 - Compétences méthodologiques et de conceptualisation (algorithmique)

AF2 - Compétences en programmation dans le langage de programmation Python

Si n est un entier impair plus grand que 1, un carré magique d'ordre n est un carré contenant tous les entiers de *1* à n^2 disposés de façon à obtenir la même somme sur n'importe quelle ligne et n'importe quelle colonne.

Par exemple, si n=5, on a le carré magique suivant :

0	17	24	1	8	15	=65
1	23	5	7	14	16	=65
2	4	6	13	20	22	=65
3	10	12	19	21	3	=65
4	11	18	25	2	9	=65
	=65	=65	=65	=65	=65	

On suppose qu'un carré est représenté par une liste de listes.

1) Écrire une fonction *somColonne* qui calcule la somme des éléments d'une colonne de numéro *numcol* dans un carre de taille *n* x *n*. Cette fonction (*environ 5 lignes*) :

- reçoit en paramètre *carre* et *numcol*
- retourne la somme des éléments de la colonne de numéro *numcol* dans *carre*

2) On suppose qu'on dispose d'une fonction *somLigne* similaire à la fonction précédente qui fait la somme des éléments d'une ligne. En utilisant ces deux fonctions, écrire une fonction *estUnCarreMagique* qui vérifie si un carre est magique ou non. Cette fonction (*entre 8 et 9 lignes*) :

- reçoit en paramètre le *carre*
- retourne *True* si c'est un carré magique, *False* sinon.

Précision : on ne demande pas d'écrire la fonction *somLigne*