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: chaîne de caractères
11      @param car:    caractère à rechercher dans la chaîne
12      @param ind:    indice de démarrage de la recherche
13      @return:       nombre d'occurrences successives du caractère car dans chaine
14                     à partir de l'indice ind jusqu'à l'apparition d'un autre caractère
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) --> 2
enigme("aabb", 'b', 2) --> 2
enigme("aabcb", 'b', 2) --> 1
enigme("caabb", 'a', 0) --> 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 ?

len est une fonction et pas une liste.

Donc pour appeler la fonction il faut mettre des parenthèses et non des [ ]

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

oui car l'éditeur de code surligne l'erreur en jaune.

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)
```

```
ltests = ["aabbb", "abbb", "aabb", "bba", "a", "bb", "aabba", "aybb"]
fonction_principale(ltests, 'a', 'b')
print(ltests)
```

```
['aabbb', 'abbb', 'aabb', 'a', 'bb']
```