1ère NSI : Types Construits

Nom : **Prénom :** **Classe :**

Partie 1

On modélise les notes d'une élève de la façon suivante :
notes_de_lea = [12, 14, 3, 16, 17, 2, 13, 19]

Q1 : Quel est le type de *notes_de_lea* ?

- ☐ int
- ☐ float
- ☐ liste
- ☐ tuple
- ☐ dictionnaire

Q2 : Que vaut l'expression *notes_de_lea[2]* ?

- ☐ 3
- ☐ 14
- ☐ 6
- ☐ 5

Q3 : Quelle instruction permet d'ajouter une note de 15 à ce type de structure de données ?

- ☐ *notes_de_lea.append(15)*
- ☐ *notes_de_lea[8] = 15*
- ☐ *notes_de_lea_.append([15])*
- ☐ *notes_de_lea = notes_de_lea + 15*

Q4 : On propose le code suivant :

```
def fonction(liste_de_notes):
"""
'liste_de_notes' est une listes de nombres qui modélise les notes d'un-e élève
Cette fonction renvoie ...?
"""

  compteur1 = 0
  compteur2 = 0
  for note in liste_de_notes:
    if note >= 10:
      compteur1 = compteur1 + 1
    else:
      compteur2 = compteur2 + 1
  return (compteur1, compteur2)
notes_de_lea = [12, 14, 3, 16, 17, 2, 13, 19]
assert fonction(notes_de_lea) == ???
```

1) Quel est le type de retour de cette fonction ?

2) L'instruction *assert* permet de tester une condition. Si celle-ci est vraie, rien ne se passe. Sinon, elle lève une exception.

Recopier et compléter la dernière ligne du code :

3) Expliquer ce que fait cette fonction.

Partie 2

La modélisation précédente n'est pas satisfaisante si l'on souhaite conserver les notes de plusieurs élèves dans une même structure de données.
On propose ici de modéliser les notes des élèves de cette façon :

notes_de_la_classe = [('Enzo', 3), ('Emma', 16), ('Karim', 14), ('Manon', 13)]

Q1 : Quel est le type de *notes_de_la classe* ?

- ☐ int
- ☐ float
- ☐ liste
- ☐ tuple

☐ dictionnaire

Q2 : Que vaut l'expression *notes_de_la_classe[2]* ?
☐ 14
☐ 'Karim'
☐ ('Karim', 14)
☐ 'Emma'

Q3 : Quelle instruction permet d'ajouter une note de 15 obtenue par Farid à ce type de structure de données ?

Q4 : On veut écrire une fonction *nom_du_génie* qui prend une telle structure de données en entrée et renvoie le nom de l'élève qui a eu la meilleure note.

```
      note_max = note
  note_max = None
def nom_du_genie(les_notes):
  return genie
      genie = nom
  genie = None
  for (nom, note) in les_notes:
    if note_max == None or note > note_max :
```

Voici le code de la fonction, à remettre dans le bon ordre !

Q5 : Que vaut l'expression nom_du_genie([ ]) ?

☐ None
☐ ' '
☐ 0
☐ Une erreur

Partie 3

Ici, on souhaite représenter les notes des élèves d'une classe en précisant le nom de la matière concernée par la note. On propose la modélisation suivante :

notes = { 'Enzo ' : ('Math', 3), 'Emma ' : ('Math', 16), 'Karim' : ('NSI', 14), 'Manon' : ('NSI', 13) }

Q1 : Quel est le type de *notes* ?

☐ int
☐ float
☐ liste
☐ tuple
☐ dictionnaire

Q2 : Que vaut l'expression *notes [2]* ?
☐ 14
☐ 'Lucas'
☐ ('NSI', 14)
☐ 3
☐ Rien, c'est une erreur.

Q3 : Quelle instruction permet d'ajouter une note de 15 obtenue par Farid en NSI ?

Q4 : Quel est l'affichage généré par le code suivant ? :

```
for (nom, (matiere, note)) in notes.items():
    if note < 15 :
        print (nom)
```

Q5 : On veut écrire une fonction qui prend une telle structure de données en paramètre, et qui renvoie le nom de l'élève qui a eu la moins bonne note, toute matières confondues. Ecrivez cette fonction.

Q6 (Bonus) : On veut écrire une fonction *tri_matiere* qui renvoie un dictionnaire dont les clés sont les noms des matières, et les valeurs la liste des notes obtenues par les élèves dans chaque matière. Ecrire la fonction.

Exemple :

```
>>> notes = {'Emma' : ('Math', 16), 'Lucas' : ('NSI' 14), 'Manon' : ('NSI', 13) }
>>> tri_par_matiere(notes)
{'Math' : [16], 'NSI' : [14, 13] }
```