1I001 — Éléments de programmation 1

TME Solo – Gr. 11.1 – B
Jeudi 13 novembre 2014 – 17h

Numéro d'étudiant :

Durée : 45 minutes soumission comprise
Aucun document autorisé hormis la carte de référence.
Dans tout ce qui suit, il est expressément demandé de fournir une signature, des hypothèses (quand elles sont nécessaires), une description et un jeu de tests pour les fonctions que vous écrivez.
Pour toutes les fonctions, vous avez le droit de donner une définition utilisant les compréhensions, mais ce n'est pas obligatoire.

Exercice 1. Mesurer la progression d'étudiants

Soit une base de données `BaseEtudiants` comportant :
- le nom de l'étudiant (`str`)
- le prénom de l'étudiant (`str`)
- le numéro d'étudiant (`int`)
- une liste de notes sur 20 (`list[int]`)

Ainsi on définit les alias pour les types `Etudiant` et `BaseEtu` :

```
### Etudiant = tuple[str, str, int, list[int]]
### BaseEtu = list[Etudiant]
```

Et la base `BaseEtudiants` :

```
# Base de donnees des etudiants a utiliser
BaseEtudiants = [('GARGA', 'Amel', 20231343, [12, 8, 11, 17, 9]),
        ('POLO', 'Marcello', 20342241, [3, 9, 11, 19]),
        ('AMANGEAI', 'Hildegard', 20244229, [15, 11, 7, 14, 12]),
        ('DENT', 'Arthur', 42424242, [4, 4, 5, 8, 9, 12]),
        ('ALEZE', 'Blaise', 30012024, [15, 16, 16, 18, 18, 19, 20]),
        ('D2', 'R2', 10100101, [10, 10, 10, 10, 10, 10])]
```

Recopiez cette base en tête du fichier que vous allez soumettre.

Dans cet exercice, nous cherchons à savoir quels sont les étudiants de `BaseEtudiants` qui ont des résultats en progression continue. *Un élève est en progression continue si chacune de ses notes est supérieure ou égale à la précédente.*

1. Progression d'une liste de notes

   Donner la définition de la fonction `est_en_progression_continue` qui à partir d'une liste de notes (entre 0 et 20) `Liste_notes` retourne *True* si elles sont en progression *False* sinon.

   Par exemple :

   ```
   >>> est_en_progression_continue([1,2,3,4,5,6])
   True
   >>> est_en_progression_continue([])
   False
   >>> est_en_progression_continue([1,2,2,4,5,6])
   True
   >>> est_en_progression_continue([1,2,5,4,5,6])
   False
   ```

Numéro d'étudiant :

Correction :

```
def est_en_progression_continue(Liste_notes):
    """ list[int] -> bool
    Retourne true si chaque note est est superieure ou egale a la precedente"""

    if len(Liste_notes) == 0:
        return False

    # note_prec : int
    note_prec = Liste_notes[0] #note_precedente

    # note: int
    for note in Liste_notes[1:]:
        if note_prec > note:
            return False
        note_prec = note
    return True
# jeu de tests
assert est_en_progression_continue([1,2,3,4,5,6]) == True
assert est_en_progression_continue([]) == False
assert est_en_progression_continue([1,2,2,4,5,6]) == True
assert est_en_progression_continue([1,2,5,4,5,6]) == False
```

2. Combien d'étudiants sont en progression continue ?
   Donnez une définition de la fonction `combien_progression_continue` qui étant donnée une base de données d'étudiants `Base` retourne le nombre d'étudiants enregistrés étant en progression continue.
   Par exemple :

```
>>> combien_progression_continue(BaseEtudiants)
4
```

Correction :

```
def combien_progression_continue(Base):
    """ BaseEtu -> int
    Retourne le nombre d'etudiants en progression
    continue dans une base donnees"""

    # compteur: int
    compteur = 0

    # etu: Etudiant
    for etu in Base:
        # notes: list[int]
        nom, prenom, numero, notes = etu # deconstruction du tuple etu

        if est_en_progression_continue(notes):
           compteur = compteur + 1


    return compteur

assert combien_progression_continue(BaseEtudiants) == 4
```

3. Qui sont les étudiants en progression continue ?
   Donnez la définition de la fonction `qui_progression_continue` qui étant donnée une base de données d'étudiants `Base` retourne une liste des tuples de type `Etudiant` dont la progression est continue.

Numéro d'étudiant :

Par exemple :

```
>>> qui_progression_continue(BaseEtudiants)
[('POLO', 'Marcello', 20342241, [3, 9, 11, 19]),
('DENT', 'Arthur', 42424242, [4, 4, 5, 8, 9, 12]),
('ALEZE', 'Blaise', 30012024, [15, 16, 16, 18, 18, 19, 20]),
('D2', 'R2', 10100101, [10, 10, 10, 10, 10, 10])]
```

Correction :

```
def qui_progression_continue(Base):
    """ BaseEtu -> BaseEtu
    Retourne une Base d'etudiants filtree ne contenant que
    les etudiants etant en progression continue"""

    # Base_resultats : BaseEtu
    Base_resultats = []

    # etu : Etudiant
    for etu in Base:
        # notes: list[int]
        nom, prenom, numero, notes = etu # deconstruction du tuple etu

        if est_en_progression_continue(notes):
            Base_resultats.append(etu)


    return Base_resultats


assert qui_progression_continue(BaseEtudiants) == [('POLO', 'Marcello', 20342241, [3, 9, 11
                                                    ('DENT', 'Arthur', 42424242, [4, 4, 5, 8
                                                    ('ALEZE', 'Blaise', 30012024, [15, 16, 1
                                                    ('D2', 'R2', 10100101, [10, 10, 10, 10,
```

4. Sous une moyenne, mais en progression ?

   Donnez la définition de la fonction `progression_continue_sous_moyenne` qui étant donnée une base de données d'étudiants `Base` et un entier `moy` compris entre 0 et 20 retourne une liste des tuples de type `Etudiant` dont la progression est continue, mais dont la moyenne est inférieure ou égale à `moy`.

   Par exemple :

```
>>> progression_continue_sous_moyenne(BaseEtudiants,10)
[('DENT', 'Arthur', 42424242, [4, 4, 5, 8, 9, 12]),
('D2', 'R2', 10100101, [10, 10, 10, 10, 10, 10])]
>>> progression_continue_sous_moyenne(BaseEtudiants,20)
[('POLO', 'Marcello', 20342241, [3, 9, 11, 19]),
('DENT', 'Arthur', 42424242, [4, 4, 5, 8, 9, 12]),
('ALEZE', 'Blaise', 30012024, [15, 16, 16, 18, 18, 19, 20]),
('D2', 'R2', 10100101, [10, 10, 10, 10, 10, 10])]
```

Correction :

```
def progression_continue_sous_moyenne(Base,moy):
    """ BaseEtu * moy -> BaseEtu
    Retourne une Base d'etudiants filtree ne contenant que
    les etudiants etant en progression continue et
    dont la moyenne est inferieure ou egale a moy"""
```

Numéro d'étudiant :

```
    # moyenne_eleve: float
    moyenne_eleve = 0.0

    # Base_resultats : BaseEtu
    Base_resultats = []

    # etu : Etudiant
    for etu in Base:
        # notes: list[int]
        nom, prenom, numero, notes = etu # deconstruction du tuple etu


        moyenne_eleve = 0.0 # reinitialisation de la moyenne

        # note : int
        for note in notes: # Il est aussi possible de faire une fonction a part
            moyenne_eleve = moyenne_eleve + note

        moyenne_eleve = moyenne_eleve / len(notes)

        if est_en_progression_continue(notes) and moyenne_eleve <= moy:
            Base_resultats.append(etu)


    return Base_resultats

assert progression_continue_sous_moyenne(BaseEtudiants,10) == [('DENT', 'Arthur', 42424242,
                                                               ('D2', 'R2', 10100101, [10,

assert progression_continue_sous_moyenne(BaseEtudiants,20) == [('POLO', 'Marcello', 2034224
                                                               ('DENT', 'Arthur', 42424242,
                                                               ('ALEZE', 'Blaise', 30012024
                                                               ('D2', 'R2', 10100101, [10,
```

Exercice 2. Suite de Syracuse

On appelle « suite de Syracuse » la suite $U_n(p)$, définie par, pour tout $n \geq 0$:

$$U_n(p) = \begin{cases} p & \text{si } n = 0 \\ \dfrac{u_{n-1}}{2} & \text{si } u_{n-1} \text{ est pair} \\ 3u_{n-1} + 1 & \text{si } u_{n-1} \text{ est impair} \end{cases}$$

Donner une définition de la fonction `syracuse` qui étant donné un entier `p` et un entier `n`, renvoie la liste des n premières valeurs de la suite $U_n(p)$.
Par exemple :

```
>>> syracuse(15,0)
[15]
>>> syracuse(15,4)
[15, 46, 23, 70, 35]
>>> syracuse(0,6)
[0, 0, 0, 0, 0, 0, 0]
>>> syracuse(42,9)
[42, 21, 64, 32, 16, 8, 4, 2, 1, 4]
```

Numéro d'étudiant :

Correction :

```
def syracuse(p,n):
    """ int * int -> list[int]
    Retourne la liste des n premieres valeurs
    de la suite de syracuse pour un entier p"""


    #prec :  int
    prec = p # valeur precedente

    #LR: list[int]
    LR = [prec]

    # i: int
    for i in range(1,n+1):
        if p != 0:
            if prec % 2 == 0:
                prec = prec//2
            else:
                prec = 3*prec+1
        LR.append(prec)
    return LR
# jeux de test
assert syracuse(15,0) == [15]
assert syracuse(15,4) == [15, 46, 23, 70, 35]
assert syracuse(0,6) == [0, 0, 0, 0, 0, 0, 0]
assert syracuse(42,9) == [42, 21, 64, 32, 16, 8, 4, 2, 1, 4]
```