Informatique

TP 1

Mise en route

Instructions

L'objet de ce TP est de voir où vous en êtes. Ce n'est en aucun cas une évaluation.

Les questions marquées d'astérisques (∗ ou ∗∗) sont supposées être plus avancées. Nous vous inquiétez pas, nous y reviendrons lentement en cours si c'est nécessaire.

1 Boucles

Question 1. *Écrire et tester une fonction* Python *prennant en argument un entier* `n` *et renvoyant* $\sum_{k=0}^{\texttt{n}} k$.

Question 2. *Écrire et tester une fonction* Python *prennant en argument un entier* `n` *et renvoyant* $\prod_{k=1}^{\texttt{n}} k$.

Question* 3 (Division Euclidienne)**.**

(1) Écrire et tester une fonction Python *qui prend en arguments deux entiers positifs* `a` *et* `b`*, et qui renvoie le* **reste** *de la division euclidienne de* `a` *par* `b`.

(2) Écrire et tester une fonction Python *qui prend en arguments deux entiers positifs* `a` *et* `b`*, et qui renvoie le* **quotient** *de la division euclidienne de* `a` *par* `b`.

2 Suites

Question 4 (Suites arithmétiques)**.** *Étant donnés* u_0 *et* b*, on considère la suite* $(u_n)_{n\in\mathbb{N}}$ *définie par la relation*

$$u_{n+1} \quad = \quad u_n + b$$

(1) Donner deux méthodes pour calculer le terme u_n *en fonction de* u_0*,* b *et* n*.*

(2) Pour chacune de ces deux méthodes, écrire une fonction Python *qui prend en arguments* u_0*,* b *et* n *et qui renvoie le terme* u_n*.*

(3) Tester vos fonctions sur des suites de votre choix.

Question 5 (Suites arithmético-géométriques)**.** *Étant donnés* u_0, a, b*, on considère la suite* $(u_n)_{n\in\mathbb{N}}$ *définie par la relation*

$$u_{n+1} \quad = \quad a \cdot u_n + b$$

On rappelle que lorsque $a \neq 1$*, on a*

$$u_n \quad = \quad a^n(u_0 - r) + r \qquad \textit{avec} \quad r = \frac{b}{1-a} \tag{$\star$}$$

(1) Écrire une fonction Python prennant en arguments u_0*,* a*,* b *et* n *et renvoyant le terme* u_n*, et procédant itérativement,* **sans utiliser la relation** ($\star$).

(2) *On souhaite maintenant utiliser la relation* ($\star$) *(dans le cas* $a \neq 1$*). Quelle(s) opération(s) peut-on utiliser pour calculer* r *?*

(3) *Écrire une fonction* Python *comme en (1), mais cette fois* **en utilisant la relation** ($\star$)*. Comparer avec les résultats de la fonction définie en (1). On pourra considérer les suites* $(u_n)_{n\in\mathbb{N}}$ *et* $(v_n)_{n\in\mathbb{N}}$ *suivantes.*

$$\begin{array}{rcll} u_{n+1} & = & 3 \cdot u_n + 1 & \text{avec} \quad u_0 = 1 \\ v_{n+1} & = & 2^{54} \cdot v_n + 1 & \text{avec} \quad v_0 = 1 \end{array}$$

3 Terminaison d'une boucle `while`

Question 6.** *Considérons la fonction* Python *suivante.*

```
1  def f(x):
2      a = 0
3      n = 1
4      while (x > 0):
5          n = n+1
6          if (a == 0):
7              x = x-2
8              a = 1
9          else:
10             x = x+1
11             a = 0
12     return n
```

Est-ce que l'exécution de `f(x)` *renvoie toujours un résultat ? Justifiez votre réponse.*

4 Listes

Dans ce §4, on pourra utiliser indifféremment des listes Python **ou des tableaux** `numpy` **à une dimension.**

Question 7 (Recherche simple)**.** *Écrire une fonction qui prend en arguments une liste et une valeur, qui renvoie* `True` *si cette valeur apparaît dans la liste, et qui renvoie* `False` *sinon.*

Question 8 (Recherche du minimum d'une liste d'entiers)**.**

(1) *Écrire une fonction qui prend en argument une liste d'entiers et qui renvoie* **l'indice** *de son élément minimum (s'il existe).*

(2) *Écrire une fonction qui prend en argument une liste d'entiers et qui renvoie son* **élément** *minimum (s'il existe).*

Question 9 (Liste triée ?)**.** *Écrire une fonction* Python *qui prend en argument une liste de nombres, qui renvoie* `True` *si cette liste est triée (par ordre croissant) et qui renvoie* `False` *sinon.*

5 Copie et modification de listes

Dans ce §5, on utilisera uniquement les listes Python**, et pas les tableaux** `numpy`**.**

Question 10 (Copie de listes d'éléments non-modifiables)**.** *Écrire une fonction* `copie_liste` *qui prend en argument une liste et qui renvoie une «* **copie profonde** *» de cette liste, c'est-à-dire telle qu'on ait, dans la console*

```
>>> x = [1,2,3]
>>> y = copie_liste(x)
>>> x == y
True
>>> x[0] = 10
>>> x == y
False
```

Question 11 (Renversement d'une liste)**.** *Écrire une fonction* `rev` *qui prend en argument une liste et qui renvoie une liste ayant les mêmes éléments, mais dans l'ordre inverse.*

Question 12. *Pour la Question 10 et la Question 11, donner le nombre d'opérations effectuées* **en fonction de la longueur de la liste passée en argument**.

6 Matrices

On s'intéresse dans un premier temps à des matrices représentées par des listes Python usuelles. Le cas des matrices NumPy est abordé en §6.3.

6.1 Rappels

On représente les matrices par des listes de listes Python. Par exemple, pour représenter la matrice

$$M = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{pmatrix}$$

on utilisera la liste de listes

```
>>> M = [[0,1,2],[3,4,5]]
```

On voit que la liste `M` contient une liste pour chaque ligne de la matrice M. La matrice M ci-dessus a 2 lignes et 3 colonnes. Nous disons que c'est une matrice de **dimension** 2×3. Plus généralement, une matrice de **dimension** $n \times m$ est une matrice avec n lignes et m colonnes.

Afin de simplifier la description des algorithmes, nous utiliserons, pour indexer les lignes et les colonnes d'une matrice, la même convention que celle des listes Python : les lignes et les colonnes d'une matrice de dimension $n \times m$ sont indexées de 0 à $n-1$ et de 0 à $m-1$ respectivement.

Étant donnée une matrice M de dimension $n \times m$, ainsi que des indices $0 \leq i < n$ et $0 \leq j < m$, on désigne par $M_{i,j}$ l'élément sur la ligne d'indice i et la colonne d'indice j de M. Par exemple, avec la matrice M ci-dessus, on a $M_{1,2} = 5$.

La représentation Pyhton des matrices que nous avons adoptée permet d'accéder simplement aux éléments d'une matrice : pour accéder à l'élément $M_{i,j}$ (c'est-à-dire à l'élément sur la ligne d'indice i et la colonne d'indice j de M), il suffit de faire `M[i][j]`. Par exemple :

```
>>> M[1][2]
5
```

Question 13. *Dans la console* Python*,*

(1) représenter la matrice

$$M = \begin{pmatrix} 5 & 10 & 3 \\ 0 & 7 & 8 \\ 13 & 1 & 4 \end{pmatrix}$$

(2) accéder à l'élément situé sur la ligne d'indice 2 *et la colonne d'indice* 0 *de* M *;*

(3) modifier l'élément situé sur la ligne d'indice 1 *et la colonne d'indice* 2 *de* M *pour lui donner la valeur* 25*.*

6.2 Opérations sur les matrices

6.2.1 Création de matrices

Question 14. *Écrire une fonction* Python `cree_mat` *qui prend en arguments deux entiers positifs* `n` *et* `m`*, et qui renvoie une matrice de dimension* $\texttt{n} \times \texttt{m}$ *(c'est-à-dire avec* `n` *lignes et* `m` *colonnes) et dont tous les éléments sont initialisés à* `None`*.*

Attention ! *On veillera à ce que la fonction* `cree_mat` *se comporte bien vis-à-vis de la modification de matrices. Par exemple, on doit avoir :*

```
>>> M = cree_mat(2,2)
>>> M
[[None, None], [None, None]]
>>> M[0][0] = 1
>>> M
[[1, None], [None, None]]
```

(uniquement la première liste de `M` *a été modifiée).*

Question 15. *Écrire une fonction* Python `cree_init_mat` *qui prend en arguments deux entiers positifs* `n` *et* `m` *ainsi qu'une valeur* `v`*, et qui renvoie une matrice de dimension* $\texttt{n} \times \texttt{m}$ *(c'est-à-dire avec* `n` *lignes et* `m` *colonnes) et dont tous les éléments sont initialisés à* `v`*.*

Cette fonction doit se comporter correctement vis-à-vis de la modification de matrices (voir Question 14).

Question 16. *Écrire une fonction* Python `cree_unit_mat` *qui prend en argument un entier positif* `n`*, et qui renvoie la matrice unité de dimension* $\texttt{n} \times \texttt{n}$*.*

6.2.2 Copie de matrices

La copie profonde de listes telle que vue précédemment ne fonctionne que pour des données **non-modifiables**. Les listes Python étant des données modifiables, pour copier une matrice (c'est-à-dire une liste de listes) il faut une fonction spécifique.

Question 17. *Écrire une fonction* Python `copie_mat` *qui prend en arguments une matrice* `mat`*, son nombre de lignes* `n` *et son nombre de colonnes* `m`*, et qui renvoie une copie profonde de* `mat`*, au sens où on doit avoir*

```
>>> mat = cree_init_mat(3,3,5)
>>> x = mat
>>> mat[0][0] = 0
>>> mat
[[0, 5, 5], [5, 5, 5], [5, 5, 5]]
>>> x
```

```
[[0, 5, 5], [5, 5, 5], [5, 5, 5]]
>>> x = copie_mat(mat,3,3)
>>> mat[0][0] = 9
>>> mat
[[9, 5, 5], [5, 5, 5], [5, 5, 5]]
>>> x
[[0, 5, 5], [5, 5, 5], [5, 5, 5]]
```

6.2.3 Produits de matrices

Question 18. *Écrire une fonction* `prod_mat(a,b,n,m,p)`*, où*
- `a` *est une matrice* $\mathtt{n} \times \mathtt{m}$*,*
- `b` *est une matrice* $\mathtt{m} \times \mathtt{p}$*,*

et qui renvoie la matrice produit $\mathtt{a} \cdot \mathtt{b}$ *(de dimension* $\mathtt{n} \times \mathtt{p}$*).*

6.3 Matrices NumPy

On utilise la bibliothèque NumPy avec la convention de nommage suivante.

```
import numpy as np
```

La bibliothèque NumPy propose (entre autres) des implémentations efficaces d'opérations matricielles. Cette efficacité repose sur l'utilisation de nombres de taille fixée.

Question 19. *Pour chaque valeur de* `n` *ci-dessous, comparer les exécutions des commandes* `np.array(n)` *et* `np.array(n,dtype=int)` *dans la console* Python*.*

(1) Pour $\mathtt{n} = 2^{64}$*.*

(2) Pour $\mathtt{n} = 2^{63}$*.*

(3) Pour $\mathtt{n} = -2^{63}$*.*

(4) Pour $\mathtt{n} = 2^{63} - 1$*.*

Le *dtype* d'un tableau NumPy est le type des éléments du tableau (on parle parfois de « scalaires »). Les *dtypes* NumPy sont en général différents des types Python correspondants. Le *dtype* `int` correspond à des entiers « signés » de taille fixe (64 bits). C'est sur les *dtypes* de taille fixe que les primitives NumPy sont particulièrement efficaces. Le *dtype* NumPy `object` permet de représenter en NumPy les entiers Python usuels (de taille variable).

Produit de matrices NumPy**.** On rappelle que les matrices NumPy peuvent être multipliées en utilisant l'opérateur `@`. Par exemple, dans la console :

```
>>> import numpy as np
>>> x = [[1,0,0],[0,1,0],[0,0,1]]
>>> a = np.array(x)
>>> y = [[0,0,0],[0,0,0],[0,0,0]]
>>> b = np.array(y)
>>> a@b
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

7 Compléments

Question 20 (Opérateurs non-stricts)**.** *Comparer l'exécution des commandes suivantes dans la console* Python *:*

- `1/0`
- `False and 1/0`
- `True and 1/0`
- `1/0 and False`

De même avec :

- `1/0`
- `True or 1/0`
- `False or 1/0`
- `1/0 or True`

Question* 21 (Opérations bit à bit)**.** *Les opérateurs « & » et « | » implémentent des opérations bit-à-bit sur la représentation binaire des entiers. L'opérateur « & » calcule le « **et** » bit-à-bit alors que « | » calcule le « **ou** » bit-à-bit. Ils renvoient tous deux un résultat entier. Par exemple, avec* $3 = \overline{11}_2$ *et* $2 = \overline{10}_2$*, on a*

$$\begin{array}{rcc} & 1 & 1 \\ \textbf{ou} & 1 & 0 \\ \hline & 1 & 1 \end{array} \qquad \begin{array}{rcc} & 1 & 1 \\ \textbf{et} & 1 & 0 \\ \hline & 1 & 0 \end{array}$$

Ainsi,

$$\begin{array}{rcccc} 3|2 & = & \overline{11}_2 & = & 3 \\ 3\&2 & = & \overline{10}_2 & = & 2 \end{array}$$

Calculer sur papier le résultat de $2\&1$*, de* $4|1$ *et de* $4\&4$*, et comparer avec le résultat de la console* Python*.*

Question* 22 (Décalages de bits)**.** *Les opérations* `x >> y` *et* `x << y` *modifient la représentation binaire de* `x` *:*

- *l'opération* `x << y` *renvoie l'entier obtenu en décalant la représentation binaire de* `x` *de* `y` *bits vers la gauche (les nouveaux bits à droite sont à* 0*) ;*
- *l'opération* `x >> y` *renvoie l'entier obtenu en décalant la représentation binaire de* `x` *de* `y` *bits vers la droite.*

Utiliser les opérations `x >> y` *et* `x << y` *pour implémenter les fonctions* $(n, k) \mapsto n//2^k$ *et* $(n, k) \mapsto n \cdot 2^k$*, où* n *et* k *sont des entiers positifs.*