Introduction à la Programmation 1 PYTHON

51AE011F

Séance 5 de cours/TD

Université Paris-Diderot

Objectifs:

- Faire un bilan après cc, questions/réponses.
- Comprendre les listes de chaînes de caractères.
- Comprendre les listes de listes
- Comprendre le statut particulier des listes en mémoire.

1 Listes d'autres types

Des listes de chaînes de caractères [COURS]

- Une liste peut contenir aussi des chaînes de caractères.
- Par exemple, pour créer une liste de chaînes de caractères `theBeatles` qui vaut initialement `["Paul", "John", "Ringo","Georges"]`, on peut procéder ainsi :

```
theBeatles = ["Paul", "John", "Ringo", "Georges"]
```

ou bien ainsi :

```
theBeatles = [""]*4
theBeatles[0] = "Paul"
theBeatles[1] = "John"
theBeatles[2] = "Ringo"
theBeatles[3] = "Georges"
```

Exercice 1 (Fonctions et liste de chaînes de caractères, ⋆)

1. *Que fait la fonction de signature « `funcAB (a)` » fournie ci-dessous en supposant qu'elle prend en paramètre un entier strictement positif ?*

```
def funcAB (a) :
  l = [""] * a
  s = "ab"
  for i in range (0, a, 1) :
    l[i] = s
    s = s + "ab"

  return (l)
```

2. *Utilisez la fonction précédente dans une suite d'instructions pour afficher 5 lignes de la forme suivante :*

```
ab
abab
ababab
abababab
ababababab
```

(On pourra utiliser la procédure « `print` ».)

□

Exercice 2 (Listes de prénoms, ⋆⋆)
Écrire une fonction prenant en paramètre une liste de prénoms `l` *et qui renvoie l'indice d'un prénom de* `l` *qui y apparaît deux fois. Si une telle valeur n'existe pas, la fonction renvoie* -1. □

Des listes de listes [Cours]

— Les listes sont des valeurs comme les autres. Une liste peut donc aussi contenir une liste dans chacune de ses cases. On parle alors de liste de listes.
— **Remarque :** En fait les éléments d'une liste ne sont pas nécessairement de même type, mais dans ce cours nous nous imposerons de manipuler et de créer des listes dont tous les éléments ont le même type.
— Par exemple, une liste de liste d'entiers peut valoir « `[[1], [11, 22], [111, 222, 333]]` ». La première valeur de cette liste est la liste « `[1]` » , puis à la deuxième valeur de la liste est « `[11, 22]` » et la dernière valeur de la liste est « `[111, 222, 333]` ».
— Pour créer et initialiser une liste de listes, il y a différentes façons de procéder :

1. On peut créer et initialiser toutes les listes au moment de l'affectation de la liste "contenant" :

```
l = [ [1, 2], [11, 22], [111, 222]]
```

2. On peut créer toutes les listes avec des valeurs initiales égales. **Attention :** Pour faire cela, la liste principale contiendra au début des entiers que l'on remplacera par des listes.

```
l = [0] *3
l[0] = [0] * 5
l[1] = [0] * 5
l[2] = [0] * 5
```

L'exemple précédent crée une liste de 3 valeurs, et chaque valeur est une liste de 5 valeurs entières. Dans cet exemples toutes les listes contenues dans la première liste ont la même taille 5, mais on aurait pu leur donner des tailles différentes comme dans l'exemple suivant :

```
li = [0] * 3
li[0] = [0] * 1
li[1] = [0] * 2
li[2] = [0] * 3
```

3. Pour extraire un élément d'une liste de listes il suffit d'enchaîner les indices, d'abord l'indice dans la liste de listes, puis l'indice dans la liste d'entiers sélectionnée. Par exemple :

```
l = [ [0], [1, 2], [3, 4] ]
print(l[2][0])

```

affiche la valeur 3.

4. Pour modifier les valeurs des listes "contenues", on peut utiliser des instructions de modification du contenu de listes en précisant leurs indices :

```
l = [ [0], [1, 2], [3, 4] ]
l[2][0] = 15
```

Cette instruction met la valeur 15 au premier indice de la troisième liste (la liste qui se trouve à la fin de la liste de listes l).
La déclaration suivante crée la liste « `liPascal` » valant « `[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1]]` » :

```
liPascal = [0] * 4
liPascal[0] = [0]
liPascal[0][0] = 1
liPascal[1] = [0] * 2
liPascal[1][0] = 1
liPascal[1][1] = 1
liPascal[2] = [0] * 3
liPascal[2][0] = 1
liPascal[2][1] = 2
liPascal[2][2] = 1
liPascal[3] = [0] * 4
liPascal[3][0] = 1
liPascal[3][1] = 3
liPascal[3][2] = 3
liPascal[3][3] = 1
```

— On parlera de listes à deux dimensions (ou bi-dimensionnelle) pour des listes de listes, les listes à trois dimensions seront des listes de listes de liste, etc.

Exercice 3 (Table de multiplication, $\star$)

1. *Créer une liste à deux dimensions tel que* `l[i][j]` *vaut* `(i + 1) * (j + 1)` *pour* i *et* j *allant de* 0 *à* 9.
2. *Où se trouve le résultat de la multiplication de* 3 *par* 4 *dans cette liste ? Même question pour le résultat de la multiplication de* 7 *par* 6.

□

Exercice 4 (Carré magique, $\star\star\star$)
Un carré magique est une grille carrée dans laquelle des nombres sont placés de telle sorte que la somme des nombres de chaque colonne, chaque ligne et de chacune des deux diagonales soit la même. De plus, le carré doit contenir une fois chaque nombre, de 1 au nombre de cases de la grille. La grille peut être représentée comme une liste bi-dimensionnel d'entiers. Notre objectif est d'écrire une fonction qui vérifie si une grille de nombres reçue comme paramètre est un carré magique.

1. *Écrire une fonction* `carre` *qui prend une liste de listes d'entiers* `m` *en paramètre et qui vérifie que* `m` *représente bien une grille carrée.*
2. *Écrire une fonction* `aplatir` *qui prend en paramètre une liste de listes d'entiers* `m` *qu'on peut supposer d'être une grille carrée, et qui envoie une liste d'entiers qui contient tous les entiers présents dans* `m`. *Par exemple, appliquée à* `[[1, 2, 3], [4, 5, 6], [6, 8, 9]]`, *la fonction doit envoyer le résultat* `[1, 2, 3, 4, 5, 6, 6, 8, 9]`.
3. *Écrire une fonction* `domaine` *qui prend une liste d'entiers* `l` *en paramètre et qui envoie le booléen* `True` *si tous les éléments de* `l` *sont des valeurs entre* 0 *(non-inclus) et la longueur de la liste (incluse), et*

`False` *sinon.*

4. *Écrire une fonction* `différents` *qui prend une liste d'entiers* `l` *en paramètre et qui envoie le booléen* `True` *si tous les éléments de* l *sont des valeurs différentes et* `False` *sinon.*
5. *Écrire une fonction* `lignes` *qui prend une liste de liste d'entiers* `m` *représentant une grille carrée et un entier* `a` *en paramètre et qui envoie le booléen* `True` *si la somme des nombres de chaque ligne est égale à* `a` *et* `False` *sinon.*
6. *Écrire une fonction* `colonnes` *qui prend une liste de liste d'entiers* `m` *représentant une grille carrée et un entier* `a` *en paramètre et qui envoie le booléen* `True` *si la somme des nombres de chaque colonne est égale à* `a` *et* `False` *sinon.*
7. *Écrire une fonction* `diagonales` *qui prend une liste de liste d'entiers* `m` *représentant une grille carrée et un entier* `a` *en paramètre et qui envoie le booléen* `True` *si la somme des nombres de chaque diagonale est égale à* `a` *et* `False` *sinon.*
8. *Enfin, écrire une fonction* `magique` *qui prend une liste de liste d'entiers* `m` *en paramètre et qui envoie* `True` *si* `m` *représente un carré magique, et* `False` *sinon. Utilisez les fonctions demandées aux questions précédentes.*

□

Exercice 5 (Léger Gomoku , ⋆⋆⋆)
Le Gomoku est un jeu de plateau à deux joueurs, dans lequel pour gagner, chaque joueur doit réussir à aligner 5 pions sur des cases consécutives d'un plateau, horizontalement, verticalement ou en diagonale. On va faire une version légère où **on ne prendera pas en compte les diagonales**. *Le plateau est une grille carrée de taille quelconque supérieure ou égale à 5, et il peut être représenté comme une liste de liste d'entiers. L'entier vaut 0 si la case est vide, 1 si elle contient un pion du joueur 1, et 2 pour un pion du joueur 2. On souhaite écrire une fonction* `winGomoku` *qui reçoit comme paramètre le contenu d'une partie de Gomoku et affiche* `"Personne ne gagne"`, *ou* `"Joueur 1 gagne"` *ou* `"Joueur 2 gagne"` *ou* `"Mauvaise grille"`, *selon les cas.*

1. *Écrire une fonction* `isGomokuSquare` *qui teste si la liste de liste d'entiers donnée en paramètre correspond à une grille carrée de dimension supérieure ou égale à 5, c'est-à-dire chaque liste de la liste principale doit avoir la même longueur que la liste principale et cette longueur doit être supérieure ou égale à 5.*
2. *Écrire une fonction* `fiveInARow` *qui prend en paramètre une liste de liste d'entiers correspondant à une grille carrée et un numéro de joueur (qui vaudra 1 ou 2) et qui teste si une liste de la grille carrée contient cinq fois le numéro du joueur à la suite.*
3. *Écrire une fonction* `fiveInAColumn` *qui prend en paramètre une liste de liste d'entiers correspondant à une grille carrée et un numéro de joueur (qui vaudra 1 ou 2) et qui teste si une colonne de la grille carrée contient cinq fois le numéro du joueur à la suite.*
4. *En vous servant des fonctions précédentes, donner le code de la fonction* `winGomoku` *(si les deux joueurs vérifient la condition pour gagner, on suppose que c'est le joueur 1 qui gagne).*

□

2 Le statut particulier des listes en mémoire

Mémoire et liste [Cours]

— Dans la mémoire, on stocke également les listes et leur contenu.
— Si `a` est une variable de type liste référant une liste créée, alors dans la mémoire, la variable `a` sera associée à la valeur $\$i$ où i est un entier positif, appelée *son adresse dans le tas*. Le tas est une mémoire auxiliaire *qui est préservée par les appels de fonction et de procédure*.

— Dans notre modèle d'exécution des programmes, le contenu d'une liste est représenté *à côté de la mémoire des variables* dans une autre mémoire. Cette seconde mémoire, le tas, associe des listes à des valeurs de la forme $\$i$.
— Par exemple :

$l1$	$l2$
$1	$2

$\$1 = [2, 3, 4, 5] \quad \$2 = [-9, -3, 0]$

indique une mémoire où la variable `l1` réfère la liste `[2, 3, 4, 5]` et la variable `l2` la liste `[-9, -3, 0]`
— Si une expression accède aux valeurs de la liste (par exemple dans « `a[2]` ») alors il faut aller regarder dans la mémoire quelle est la liste référée par la variable `a`, par exemple $\$3$ et ensuite prendre la valeur qui se trouve dans la troisième case de la liste $\$3$. Le même procédé s'applique pour les modifications du contenu de la liste.
— Si une liste $\$i$ n'est référencée par aucune variable dans la mémoire, alors on peut la supprimer.
— Comme le contenu du tas est préservé par les appels de fonctions et de procédures, on peut écrire des fonctions et des procédures qui modifient les listes passées en paramètres. C'est très pratique de ne pas avoir à recopier le contenu des listes à chaque appel de procédure car la taille des listes peut être importante.

Exercice 6 (Échange du contenu de deux indices, ⋆⋆)
Soit la procédure suivante :

```
def swap (a, i, j) :
  tmp = a [i]
  a[i] = a[j]
  a[j] = tmp
```

Donner l'évolution de la mémoire et du tas au cours de l'évaluation des instructions suivantes :

```
li = [ 3, 2, 1, 0 ]
swap (li, 1, 2)
swap (li, 0, 3)
```

□

3 DIY

Exercice 7 (Comptage, ⋆⋆)
Écrire une fonction qui, étant donnée une liste `l` *de nombres entiers et un nombre entier* `n`*, renvoie :*
— *la liste vide si la liste donnée contient (au moins) un nombre* `x` *tel que* $x < 0$ *ou* $x > n$*,*
— *une liste* `li` *de* $n + 1$ *entiers tel que* `li[i]` *soit égal au nombre d'éléments de* `l` *égaux à* `i` *si la liste ne contient que des nombres compris, au sens large, entre* `0` *et* `n`*.*

Contrat:
Si la liste vaut `[0, 1, 2, 2, 0, 0]` *alors*
si le deuxième argument vaut 1, la fonction renverra `[]`
si le deuxième argument vaut 2, la fonction renverra `[ 2, 1, 3]`

□

Exercice 8 (Suites d'entiers, ⋆⋆)
Toute suite finie d'entiers peut être décomposée de manière unique en une suite de séquences strictement croissantes maximales. En représentant une suite dans une liste `l`*, la liste*

$$[1\ 2\ 5\ 7\ \underline{2}\ 6\ \underline{0}\ 5\ \underline{2}\ 4\ 6\ 7\ 8\ 9\ \underline{3}\ 4\ 6\ \underline{1}\ 2\ 7\ 8\ 9\ \underline{4}\ \underline{2}\ 3\ \underline{1}\ 5\ 9\ \underline{7}\ \underline{1}\ 6\ \underline{6}\ \underline{3}]$$

se décompose ainsi en la liste de 13 listes d'entiers suivante :

$$[\ [1\ 2\ 5\ 7]\ [2\ 6]\ [0\ 5]\ [2\ 4\ 6\ 7\ 8\ 9]\ [3\ 4\ 6]\ [1\ 2\ 7\ 8\ 9]\ [4]\ [2\ 3]\ [1\ 5\ 9]\ [7]\ [1\ 6]\ [6]\ [3]\ ].$$

Les premiers éléments de ces séquences sont, en plus du premier élément de la liste, les éléments (soulignés sur l'exemple) qui sont inférieurs ou égaux à leur prédécesseur dans la liste (`l[i]`$\leq$`l[i-1]`*).*

1. *Écrire une fonction* `numRuptures` *qui une liste* `l` *d'entiers donnée en paramètre renvoie le nombre d'indices* `i` *tels que* `l[i]`$\leq$`l[i-1]`*. Pour la liste donnée en exemple, la fonction renvoie 12 (le nombre de positions soulignées).*
2. *Écrire une fonction* `rupture` *qui, étant donnée une liste* `l` *d'entiers, renvoie une liste contenant 0 en premier élément et les indices des éléments de* `l` *inférieurs ou égaux à leur prédécesseur en ordre croissant. Pour la liste donnée en exemple, la fonction renvoie la liste :*

$$[0\ 4\ 6\ 8\ 14\ 17\ 22\ 23\ 25\ 28\ 29\ 31\ 32].$$

3. *Écrire une fonction* `factorisation` *qui, étant donné une liste* `l` *d'entiers, renvoie une liste de listes d'entiers, dont la* `i`*-ème liste contient la* `i`*-ème plus longue séquence croissante de nombres adjacents dans la liste* `l` *(résultat tel que celui donné dans l'exemple). Indication : on pourra utiliser la fonction* `rupture` *pour déterminer le nombre de listes et la taille de chaque liste de cette liste de liste.*

□

Exercice 9 (Matrices, $\star\star$)
Étant donnée une matrice (liste à deux dimensions) `A` *avec* `n` *lignes et* `m` *colonnes, un couple d'indices (*`i`,`j`*) représente un* min-max *de cette matrice si la valeur* `A[i][j]` *est un minimum de la ligne* `i` *et un maximum de la colonne* `j`*, c'est-à-dire*

$$\texttt{A[i][j]} = min\{\texttt{A[i][0]}, \ldots, \texttt{A[i][m]}\} \quad \texttt{A[i][j]} = max\{\texttt{A[0][j]}, \ldots, \texttt{A[n][j]}\}.$$

Ecrivez une fonction prenant en paramètre une liste de liste d'entiers encodant une matrice et qui affiche l'ensemble de tels couples (`i`,`j`*). La méthode proposée consiste à effectuer le travail suivant pour chaque ligne* `i` *: (1) trouver les minima de la ligne* `i` *et en mémoriser les numéros de colonne et (2) pour chacun de ces rangs* `j`*, déterminer si* `a[i][j]` *est un maximum pour sa colonne.* □

Exercice 10 (Championnat, $\star\star\star$)
On considère une liste à 3 dimensions stockant les scores d'un championnat de handball. Pour `n` *équipes, la liste aura* `n` *lignes et* `n` *colonnes et à chacun de ses indices on trouvera une liste de longueur 2 contenant le score d'un match (on ne tient pas compte de ce qui est stocké dans la diagonale). Ainsi, pour la liste championnat* `ch`*, on trouvera dans* `ch[i][j]` *le score du match de l'équipe* `i+1` *contre l'équipe* `j+1` *et dans* `ch[j][i]` *le score du match de l'équipe* `j+1` *contre l'équipe* `i+1`*. De même, pour un score stocké, le premier entier de* `ch[i][j]` *sera le nombre de but(s) marqué(s) par l'équipe* `i+1` *dans le match l'opposant à l'équipe* `j+1`*. Par conséquent, dans* `ch[i][j][0]` *on trouve le nombre de bus obtenu par* `i+1` *dans le match l'opposant à* `j+1` *et dans* `ch[i][j][1]` *on trouve le nombre de bus obtenu par* `j+1` *dans le match l'opposant à* `i+1`*. On remarque que le match* `ch[i][j]` *(match aller) n'est pas le même que* `ch[j][i]` *(match retour). Finalement, on suppose que lorsqu'une équipe gagne un match, elle obtient 3 points, 1 seul point pour un match nul et 0 point dans le cas où elle perd le match.*

1. *Écrire une méthode* `nombrePoints` *qui prend en arguments une liste championnat* `ch` *de longueur* `n` *et le numéro d'une équipe (entre 1 à* `n`*) et qui renvoie le nombre de point(s) obtenu(s) par cette équipe pendant le championnat.*
2. *Écrire une méthode* `stockerScore` *qui prend en arguments une liste championnat* `ch` *de longueur* `n`*, le numéro* `i` *d'une équipe, le numéro* `j` *d'une autre équipe et le score du match de* `i` *contre* `j` *et qui met à jour la liste* `ch`*.*
3. *Écrire une méthode champion qui prend en argument une liste championnat* `ch` *et qui renvoie le numéro de l'équipe championne. Une équipe est championne si elle a strictement plus de points que toutes les autres. Si plusieurs équipes ont le même nombre de points, alors une équipe est meilleure si elle a marqué strictement plus de buts. Dans le cas d'égalité parfaite (même nombre maximum de points et même nombre maximum de buts marqués), la méthode renverra 0 pour signaler l'impossibilité de désigner un champion.*

□

Exercice 11 (Maxima locaux, $\star\star$)
Écrire une fonction qui prend en paramètre une liste d'entiers à deux dimensions (rectangulaire) et calcule le **nombre** *d'entrées* **intérieures** *de la matrice dont tous les voisins sont strictement plus petits. Chaque entrée intérieure de la matrice a quatre voisins (à gauche, à droite, vers le haut, vers le bas). Par exemple, pour la matrice*

```
1 4 9 1 4
4 8 1 2 5
4 1 3 4 6
5 0 4 7 6
2 4 9 1 5
```

la méthode devrait renvoyer 2 *car il y a deux éléments de la matrice (*8 *et* 7*) qui ont uniquement des voisins plus petits.* □

Introduction à la Programmation 1 PYTHON

51AE011F

Séance 6 de cours/TD

Université Paris-Diderot

Objectifs:

- Boucles `while`.
- Variables booléennes.

1 La boucle “while”

Boucle non bornée [COURS]

La boucle non bornée permet de répéter des instructions tant qu'une expression booléenne est vraie. Elle est utile lorsqu'on ne connaît pas *a priori* le nombre d'itérations nécessaires.

```
while (expression booléenne) :
    instructions à répéter
```

- L'expression booléenne est appelée condition ou encore test d'arrêt de la boucle.
- Par exemple, la boucle suivante s'arrêtera quand la valeur de la variable entière a sera 0 après avoir affiché 50 lignes contenant la chaîne « `Hello` ».

```
a = 50
while (a > 0) :
    print ("Hello")
    a = a - 1
```

- Une boucle non bornée peut ne jamais terminer si sa condition est toujours vraie. La plupart du temps, la non terminaison du programme n'est pas le comportement escompté car on souhaite obtenir un résultat ![1]
- Par exemple, la boucle suivante ne termine jamais et l'instruction « `print ("Bonjour")` » n'est donc jamais exécutée :

```
b = 0
while (b == 0) :
    b = b // 2
print ("Bonjour")
```

Exercice 1 (Première boucle, ⋆)
Quelle est la valeur de la variable `r` *à la fin de la suite des instructions suivantes ?*

```
n = 49
r = 0
```

```
while (r * r < n) :
    r = r + 1
```

□

Exercice 2 (Les "for"s vus comme des "while"s, ⋆)

Réécrivez la suite d'instructions suivante pour obtenir le même comportement en utilisant un "while" à la place du "for".

```
a = 0
for i in range (0, 32, 1) :
    a = a + i
print (a)
```

□

Exercice 3 (Affichage maîtrisé, ⋆)

Écrire, à l'aide d'une boucle, un programme qui affiche la chaîne `"bla"` *plusieurs fois de suite autant que possible, sans dépasser les* 80 *caractères (longueur standard d'une ligne dans un terminal).*

□

Exercice 4 (Terminaison, ⋆)

Qu'affichent les deux séquences d'instructions suivantes ? Est-ce que leur exécution se termine ?

```
n = 2
while (n > 0) :
    print (n)
```

```
n = 2
while (n > 0) :
    n = n * 2
    print (n)
```

□

2 Variables de type `bool`

Présentation du type `bool` [Cours]

— Comme toute valeur, une valeur de type `bool` (qui correspond à un booléen) peut être stockée dans une variable.
— Rappel : Il y a deux valeurs pour le type `bool` qui sont `True` et `False`.
— Par exemple, on peut initialiser une variable booléenne ainsi :

```
b = False
```

ou bien encore comme cela :

```
b = x >5
```

Dans ce dernier cas, si la valeur contenue dans `x` est strictement plus grande que 5 alors l'expression booléenne « `x > 5` » s'évaluera en la valeur `True`, qui sera la valeur initiale de la variable `b`.

— Rappels : les opérateurs booléens sont `or` (**ou**), `and` (**et**) et `not` (**non**).

— Par exemple, les affectations suivantes sont valides :

```
b1 = False #b1 vaut False.
b2 = (3 + 2 < 6) # b2 vaut True.
b1 = not (b2)  # b1 vaut la negation de True, donc False.
b2 = b1 or b2 # b2 vaut False or True, donc True.
```

— **Attention :** le symbole = est utilisé dans les instructions d'affectation tandis que le symbole == est un opérateur de test d'égalité entre entiers ou entre chaînes de caractères.

— Comme toute valeur, une valeur de type `bool` peut être passée en paramètre à une fonction ou une procédure et renvoyée par une fonction.

Exercice 5 (Évaluer, ⋆)

Quelles sont les valeurs des variables `x`, `b`, `d`, `e` *et* `f` *après avoir exécuté les instructions suivantes ?*

```
x = 3 + 5
b = (x == 5 + 3)
d = (x > x + 1)
e = b or d
f = b and d
f = (e != b)
```

□

Exercice 6 (Parité, ⋆)

Écrire une fonction « `isEven (a)` *» qui prend en paramètre un entier et renvoie* `True` *si cet entier est pair,* `False` *sinon.* □

Exemples d'utilisation de variables booléennes [Cours]

— Les variables booléennes peuvent être utilisées pour signaler qu'une condition a été vérifiée, ce qui peut être utile pour arrêter une boucle.

— On peut aussi utiliser les variables booléennes comme paramètres de fonction pour faire varier le comportement selon des conditions.

— Les variables booléennes sont parfois appelées drapeaux (*flag* en anglais).

Exercice 7 (Drapeau et recherche, ⋆)

1. *On considère la suite d'instructions suivante qui parcourt une liste* `l` *d'entiers et affecte* `True` *à la variable* `found` *si une des cases de* `l` *vaut* 3 *:*

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
i = 0
found = False  # flag
while (i < len (l)) :
  if (l[i] == 3) :
    found = True
  i = i + 1
```

   Quelle est la valeur contenue dans la variable `i` *après ces instructions ?*

2. *Même question sur la suite d'instructions suivante :*

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
i = 0
found = False  # flag
```

```
while (i < len (l) and not (found)) :
  if (l[i] == 3) :
    found = True
  i = i + 1
```

3. *Qu'en déduisez-vous ?*
4. *Écrire une fonction « `findValueList (l, v)` » qui teste si la valeur contenue dans `v` est présente dans la liste d'entiers `l`.*

□

Exercice 8 (Paramètre booléen, ⋆)
Écrire une procédure qui énumère les entiers entre 1 et n. La procédure prend deux paramètres : l'entier n, et un booléen `up`. Si le paramètre booléen est vrai, on affiche les entiers de 1 à n, sinon on affiche les entiers de n à 1. □

3 DIY

Exercice 9 (Comptine, ⋆)
Modifier les instructions suivantes pour que l'accord de kilomètre(s) soit correct.

```
n = 10
while (n > 0) :
    print (n, end="")
    print (" kilometre a pied ca use les souliers")
    n = n - 1
```

□

Exercice 10 (Palindrome, ⋆⋆)
Un mot est un palindrome si on obtient le même mot en le lisant à l'envers. Par exemple "kayak", "ressasser", ou "laval" sont des palindromes. Écrire une fonction qui prend en paramètre une chaîne de caractères `s` et qui renvoie le booléen `True` si le mot `s` est un palindrome et `False` sinon. □

Exercice 11 (Logarithme, ⋆)
Écrire une fonction qui prend en paramètre un entier n, et renvoie le nombre de fois qu'il faut diviser celui-ci par deux avant que le résultat soit inférieur ou égal à 1. □

Exercice 12 (n-ème chiffre, ⋆⋆)
Écrire une fonction qui prend en paramètres deux entiers k et n et renvoie le n-ème chiffre de k, ou 0 si k n'est pas aussi long (par exemple le 2ème chiffre de 1789 est 8). □

Exercice 13 (Binaire, ⋆⋆⋆)
Écrire une fonction qui prend en paramètre un entier `n` et qui renvoie la représentation binaire de `n` sous la forme d'une chaîne de caractères formée de caractères 0 et 1. □

Exercice 14 (lastOcc, ⋆⋆)
Écrire une fonction « `lastOcc(li, x)` » qui prend en paramètre une liste d'entiers et une valeur entière. Si la liste contient cette valeur, la fonction renvoie l'indice de la dernière occurrence de la valeur. Si la liste ne contient pas cette valeur, la fonction renvoie -1.

Cet exercice a déjà été traité au TD 4, mais on demande de remplacer la boucle `for` par une boucle `while`. □

Exercice 15 (Syracuse, ⋆⋆⋆)
La suite de Syracuse de premier terme p (entier strictement positif) est définie par $a_0 = p$ et

$$a_{n+1} = \begin{cases} \frac{a_n}{2} & \text{si } a_n \text{ est pair} \\ 3 \cdot a_n + 1 & \text{si } a_n \text{ est impair} \end{cases} \quad \text{pour } n \geq 0$$

Une conjecture (jamais prouvée à ce jour !) est que toute suite de Syracuse contient un terme $a_m = 1$. Trouver le premier m pour lequel cela arrive, étant donné p. □

Exercice 16 (Itérations, $\star$)
Combien y a-t-il d'itérations dans la boucle suivante :

```
n = 15
while (n >= 0) :
    n = n - 1
```

Même question pour la boucle suivante, en fonction du paramètre `n` *:*

```
def nbIterations (n) :
    while (n >= 0) :
        n = n - 1
```

□

Exercice 17 (Logarithme (itéré), $\star\star$)

1. *Le logarithme en base 2 d'un entier $n \geq 1$ (noté $\log_2 n$) est le réel x tel que $2^x = n$. On se propose de calculer la partie entière de $\log_2 n$, c'est-à-dire le plus grand entier m tel que $2^m \leq n$. On notera l la partie entière du logarithme en base 2, c'est-à-dire que $m = l(n)$. Par exemple, $l(8) = l(9) = 3$ car $2^3 = 8 \leq 9 < 2^4$.*
   Écrire une fonction `lo` *qui prend en paramètre un entier n et renvoie la partie entière* `lo (n)` *de son logarithme en base 2. On calculera* `lo (n)` *en effectuant des divisions successives par 2.*
2. *À partir de la fonction* `lo` *précédente, on peut définir une fonction* `loStar` *ainsi :* `loStar (n)` *est le plus petit entier i tel que la i-ème itération de* `lo` *sur l'entrée n vaille 0. Par exemple,* `loStar (1)` *vaut 1 car dès la première itération,* `lo (1)` *vaut 0 ; ou encore* `loStar (2)` *vaut 2 car* `lo (2)` *vaut 1 et* `lo (1)` *vaut 0. Pour prendre un exemple plus grand, on a* `loStar (1500)` *vaut 4 car* `lo (1500)` *vaut 10,* `lo (10)` *vaut 3,* `lo (3)` *vaut 1 et* `lo (1)` *vaut 0 ; la quatrième itération de* `lo` *sur 1500 vaut donc 0.*
   Écrire la fonction `loStar` *qui prend en paramètre un entier n.*

□

Exercice 18 (Racine cubique, $\star\star$)
Écrire une fonction qui renvoie la racine cubique d'un entier x, c'est-à-dire le plus petit entier a tel que $a^3 \geq x$. On peut s'inspirer de l'exercice 1. □

Exercice 19 (Racine $n$ième, $\star\star\star$)
Maintenant, écrire une fonction qui renvoie la racine $n$ième d'un entier x, c'est-à-dire le plus petit entier a tel que $a^n \geq x$. □

Introduction à la Programmation 1 PYTHON

51AE011F

Séance 7 de cours/TD

Université Paris-Diderot

Objectifs:

— Connaître ses classiques sur les listes.

— Apprendre à voir quand un programme est faux.

1 Devenir incollable sur les listes

Opérations à savoir faire sur les listes [COURS]

- **Rechercher**
  - Dire si un élément est présent dans une liste
  - Renvoyer la position d'un élément dans la liste (la première ou la dernière)
  - Compter le nombre de fois qu'un élément est présent dans une liste.
  - Savoir construire une liste des positions d'un élément dans une liste
- **Modifier**
  - Savoir échanger deux éléments de place dans une liste
  - Savoir renverser une liste
  - Savoir décaler tous les éléments d'une liste
  - Savoir appliquer une fonction à tous les éléments d'une liste

Exercice 1 (Présence dans une liste, ⋆)
On considère la fonction `isPresent (el, li)` *qui renvoie* `True` *si* `el` *est dans la liste* `li`.

```
def isPresent (el, li) :
    for i in range (0, len(li), 1) :
        if (li[i] == el) :
            return (True)
        else :
            return (False)
    return (False)

def isPresent (el, li) :
    b = True
    for i in range (0, len(li), 1) :
        if (li[i] == el) :
            b = True
        else :
            b = False
    return (b)

```

```python
def isPresent (el, li) :
    b = True
    for i in range (0, len(li), 1) :
        return (li[i] == el)

def isPresent (el, li) :
    b = False
    i = 0
    while ( i < len (li) and b) :
        if (li[i] == el) :
            b = True
    return (b)

def isPresent (el, li) :
    b = False
    i = 0
    while ( i < len (li) and not(b)) :
        b = b and (li[i] == el)
    return (b)
```

code/exoPresent.py

1. *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de listes d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*
2. *Proposez une correction pour chacun des cas.*

□

Exercice 2 (Occurences dans une liste, ⋆)
On considère la fonction `occ (el, li)` *qui renvoie le premier indice de* `el` *dans la liste* `li` *et* `-1` *si* `el` *n'est pas dans la liste.*

```python
def occ (el, li) :
    for i in range (0, len(li), 1) :
        if (el == li[i]) :
            return (i)
        else :
            return (-1)
    return (-1)

def occ (el, li) :
    for i in range (0, len(li), 1) :
        if (el == li[i]) :
            return (i)
    return (i)

def occ (el, li) :
    p = 0
    for i in range (0, len(li), 1) :
        if (el == li[i]) :
            p = i
        else :
            p = -1
    return (p)

def occ (el, li) :
    p = 0
    for i in range (0, len(li), 1) :
```

```
27          if (el == li[i]) :
28              p = i
29      return (p)
30
31  def occ (el, li) :
32      p = -1
33      i = 0
34      while ( i < len(li)) :
35          if (el == li [i]) :
36              p = i
37              i = i + 1
38      return (p)
39
40  def occ (el, li) :
41      p = -1
42      i = 0
43      while ( i < len(li) and p != -1 ) :
44          if (el == li [i]) :
45              p = i
46          i = i + 1
47      return (p)
```

code/exoOccurence.py

.

1. *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de listes d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*
2. *Proposez une correction pour chacun des cas.*

□

Exercice 3 (Compter dans une liste, ⋆)

On considère la fonction `count (el, li)` *qui renvoie le nombre de fois que l'élément* `el` *apparaît dans la liste* `li`.

```
1   def count (el, li) :
2       c = 0
3       for i in range (0, len(li), 1) :
4           if (li[i] == el) :
5               c = c + 1
6               return (c)
7       return (c)
8
9   def count (el, li) :
10      c = 0
11      for i in range (0, len(li), 1) :
12          if (li[i] == el) :
13              c = c + 1
14          else :
15              c = c - 1
16      return (c)
```

code/exoCounting.py

.

1. *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de listes d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*
2. *Proposez une correction pour chacun des cas.*

□

Exercice 4 (Construire la liste des positions, ⋆)
On considère la fonction `pos (el, li)` *qui renvoie une liste contenant les positions où* `el` *apparaît dans la liste* `li`.

1. *Que valent les variables* `li1`, `li2` *et* `li3` *après exécution du programme suivant (en supposant que la fonction* `pos` *est correctement définie).*

```
li1 = pos (2, [3, 4, 5, 8])
li2 = pos (3, [3, 3, 4, 3, 4, 5, 6, 4, 3])
li3 = pos (4, [3, 3, 4, 3, 4, 5, 6, 4, 3])
```

2. *On considère les propositions suivantes pour la fonction* `pos (el, li)` *(on suppose que la fonction* `count` *est celle de l'exercice précédent et qu'elle est correcte).*

```
def pos (el, li) :
    lret = []
    for i in range (0, len(li), 1) :
        if (li[i] == el) :
            lret = [i]
    return lret

def pos (el, li) :
    n = count (el, li)
    lret = [0] * n
    for i in range (0, len(li), 1) :
        if (li[i] == el) :
            lret[i] = i
    return lret

def pos (el, li) :
    n = count (el, li)
    lret = [0] * n
    for i in range (0, len(li), 1) :
        if (li[i] == el) :
            lret[el] = i
    return lret

def pos (el, li) :
    n = count (el, li)
    lret = [0] * n
    j = 0
    for i in range (0, len(li), 1) :
        if (li[i] == el) :
            lret[j] = i
    return lret
```

code/exoPos.py

   a *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de listes d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*

   b *Proposez une correction.*

□

Exercice 5 (Échanger deux éléments dans une liste, ⋆)
Dans cet exercice, on considère la fonction `swap (i , j, li)` *qui échange les valeurs des indices* `i` *et* `j` *de la liste* `li`.

1. *Pour comprendre ce qui se passe tout en s'échauffant dîtes ce qu'affiche le programme suivant :*

```python
def modif (l) :
    l[0] = 5

li = [1, 2, 4, 8, 16, 31]
modif (li)
print (li)
```

code/exoSwap1.py

2. *En déduire ce que doit retourner la fonction* `swap`.
3. *Pourquoi la définition suivante de* `swap` *n'est pas correcte ? Proposez une correction.*

```python
def swap (i , j, li) :
    if (i < len (li) and j < len(li)) :
        li[i] = li[j]
        li[j] = li[i]
```

code/exoSwap2.py

□

Exercice 6 (Inverser une liste d'éléments, ⋆)

Dans cet exercice, on considère la fonction `reverse (li)` *qui prend en paramètre une liste* `li` *et renvoie la liste où l'ordre des valeurs est inversée.*

1. *Que renvoie donc* `reverse ([0, 1, 2, 3, 4, 5])`.

```python
def reverse (li) :
    for i in range (0, len (li), 1) :
        li[i] = li [len(li) - 1 - i]
    return li

def reverse (li) :
    for i in range (0, len (li), 1) :
        temp = li[i]
        li[i] = li [len(li) - 1 - i]
        li [len - 1 - i] = temp
    return li

def reverse (li) :
    l = []
    for i in range (0, len (li), 1) :
        l [i] = li [len(li) - 1 - i]
    return li
```

code/exoReverse.py

2. a *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de listes d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*

   b *Proposez une correction.*

   c *Parmi les corrections proposées, quelles sont celles qui changent la liste* `li` *passée en paramètre et quelles sont celles qui la laissent inchangée ?*

□

Exercice 7 (Décaler les éléments d'une liste, ⋆⋆)

Dans cet exercice, on considère la fonction `shift (li, p)` *qui prend en paramètre une liste* `li` *et modifie cette liste en décalant (de façon cyclique vers la droite) ses éléments de* `p` *positions (en supposant que* `p` *est positif).*

1. *Que valent les variables* `li1`, `li2` *et* `li3` *après exécution du programme suivant (en supposant que la fonction* `shift` *est correctement définie).*

```
li1 = [3, 4, 5, 8]
shift (li1, 1)
li2 = [3, 3, 4, 3, 4, 5]
shift (li2, 2 )
li3 = [1, 2, 3, 4, 5, 6]
shift (li3, 7)
```

```
def shift (li, p) :
    for i in range (0, len (li), 1) :
        li [i] = li [i - p]

def shift (li, p) :
    for i in range (0, len (li), 1) :
        temp = li [i]
        li [i - p] = temp

def shift (li, p) :
    for i in range (0, len (li), 1) :
        li [(i + p) % len(li)] = li [i]
```

code/exoShift.py

2. a *Pourquoi les propositions précédentes sont-elles fausses ? À chaque cas, donnez un exemple de listes d'entiers et d'éléments pour laquelle la fonction renvoie un mauvais résultat.*

   b *Proposez une solution. On pourra pour cela commencer par écrire une fonction* `shift1` *qui décale la liste d'un élément vers la droite et utiliser ensuite cette fonction pour faire* `shift`*.*

□

2 DIY

Pour certains exercices, un "contrat" est proposé : ce sont des tests que nous vous fournissons pour vérifier votre réponse. Si votre réponse ne passe pas ces tests, il y a une erreur ; si votre réponse passe les tests, rien n'est garanti, mais c'est vraisemblablement proche de la réponse. Quand aucun contrat n'est spécifié, à vous de trouver des tests à faire.

Exercice 8 (Primalité, ⋆⋆)
Un entier p est premier si $p \geq 2$ et s'il n'a pas d'autres diviseurs que 1 et lui-même.

1. *Écrire une fonction* `prime` *qui prend en paramètre un entier n et qui renvoie* `True` *si n est premier, ou* `False` *sinon.*
2. *Écrire une fonction* `next` *qui prend en entrée un entier x et qui renvoie le plus petit nombre premier $p \geq x$. On pourra bien sûr se servir de la fonction* `prime` *précédente.*
3. *Écrire une fonction* `number` *qui prend en entrée un entier y et qui renvoie le nombre de nombres premiers $p \leq y$. On pourra bien sûr se servir de la fonction* `prime`*.*

Contrat:
Pour la fonction `next` *:*

x = 2 → 2
x = 10 → 11
x = 20 → 23

Pour la fonction `number` *:*

x = 10 → 4
x = 20 → 8

□

Exercice 9 (Multiplication de matrices, ⋆⋆)
Dans cet exercice, on suppose qu'une matrice A à m lignes et n colonnes est encodée par une liste de listes contenant m listes de taille n. Par exemple, la matrice

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

est encodée en Python par `[[1, 2, 3], [4, 5, 6]]`.

Étant données deux matrices A (à m lignes et n colonnes) et B (à n lignes et p colonnes), le produit matriciel de A par B est une matrice C à m lignes et p colonnes telle que pour tout $1 \leq i \leq m$ et pour tout $1 \leq j \leq p$, le coefficient $C_{i,j}$ de C se trouvant à la i-ème ligne et la j-ième colonne est égal à :

$$C_{i,j} = \Sigma_{1 \leq k \leq n} A_{i,k} B_{k,j}$$

1. *Écrire une fonction* `isMatrix (A)` *qui prend en paramètre une liste de listes d'entiers et vérifie qu'il s'agit de l'encodage d'une matrice (c'est à dire que chaque sous-liste à la même longueur) ; elle renvoie* `True` *dans ce cas et* `False` *sinon.*
2. *Écrire une fonction* `nbLines (A)` *qui prend en paramètre une liste de listes d'entiers qui encode une matrice et renvoie son nombre de lignes.*
3. *Écrire une fonction* `nbColumns (A)` *qui prend en paramètre une liste de listes d'entiers qui encode une matrice et renvoie son nombre de colonnes.*
4. *Écrire une fonction* `matriProx (A, B)` *qui prend en paramètre deux listes de listes d'entiers, vérifie qu'il s'agit de deux matrices, et vérifie que le nombre de colonnes de* `A` *est égal au nombre de lignes de* `B`. *Si ces conditions ne sont pas satisfaites, la fonction renvoie* `[]` *et sinon elle renvoie une liste de listes contenant la matrice correspondant au produit matriciel de* `A` *par* `B`.

□

Exercice 10 (Addition, ⋆⋆⋆)
Le but de cet exercice est de programmer l'addition décimale. Les deux nombres à additionner sont donnés sous forme de listes.

Par exemple, `x = [7, 4, 3]` *et* `y = [1, 9]`, *dont la somme doit être retournée sous forme de liste, dans cet exemple,* `[7, 6, 2]`.

Écrire une fonction `add` *qui prend en paramètre deux listes et fait l'addition des deux nombres représentés par les listes.*

Par exemple, si les entrées sont les listes `t1 = [3, 4, 7], t2 =[9, 1]`, *cela représente la somme $743+19$. On calcule d'abord $9+3$ ce qui fait 2 avec une retenue de 1. Puis on calcule $4+1$, plus la retenue, ce qui donne 6, avec une retenue de 0. Enfin, le dernier chiffre est 7. À la fin, on doit renvoyer la liste* `[0, 7, 6, 2]`.
Remarque : *Le premier chiffre dans la liste (celui le plus à gauche) peut-être 0.*

Contrat:

`t1 = [2, 4, 3, 5]`	`t2 = [4, 3, 6]`	→	*renvoie :* `[0, 2, 8, 7, 1]`
`t1 = [1, 2]`	`t2 = [1, 3]`	→	*renvoie :* `[0, 2, 5]`
`t1 = [6, 1, 2]`	`t2 = [0, 6, 3, 0]`	→	*renvoie :* `[1, 2, 4, 2]`

□

Exercice 11 (Pascal, ⋆⋆⋆)
Écrire un programme qui affiche le triangle de Pascal jusqu'au rang n où n est un entier demandé à l'utilisateur. Si l'utilisateur rentre 4, le programme affichera :

```
1
1  1
1  2  1
```

```
1  3  3  1
1  4  6  4  1
```

Le programme doit faire appel à deux fonctions auxiliaires la première qui, étant donné n, renvoie le triangle de Pascal sous forme d'une liste de listes d'entiers de n + 1 lignes, la i-ème ligne contenant les coefficients de la i-ème puissance du binôme $(a + b)$ et la deuxième qui sert à afficher une liste de liste en affichant sur chaque ligne le contenu de chacune des listes (en séparant les différentes valeurs par des espaces). □