

# 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 :

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

ou bien ainsi :

```
1 theBeatles = [""]*4
2 theBeatles[0] = "Paul"
3 theBeatles[1] = "John"
4 theBeatles[2] = "Ringo"
5 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 ?

```
1 def funcAB (a) :
2     l = [""] * a
3     s = "ab"
4     for i in range (0, a, 1) :
5         l[i] = s
6         s = s + "ab"
7
8     return (l)
```

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

```
1 ab
2 abab
3 ababab
4 abababab
5 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" :

```
1 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.

```
1 l = [0] * 3
2 l[0] = [0] * 5
3 l[1] = [0] * 5
4 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 :

```
1 li = [0] * 3
2 li[0] = [0] * 1
3 li[1] = [0] * 2
4 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 :

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

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 :

```
1 l = [ [0], [1, 2], [3, 4] ]
2 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]]* » :

```
1 liPascal = [0] * 4
2 liPascal[0] = [0]
3 liPascal[0][0] = 1
4 liPascal[1] = [0] * 2
5 liPascal[1][0] = 1
6 liPascal[1][1] = 1
7 liPascal[2] = [0] * 3
8 liPascal[2][0] = 1
9 liPascal[2][1] = 2
10 liPascal[2][2] = 1
11 liPascal[3] = [0] * 4
12 liPascal[3][0] = 1
13 liPascal[3][1] = 3
14 liPascal[3][2] = 3
15 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, ★)

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, ★★★)

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 (inclusive), 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 prendra 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 :

```

1 def swap (a, i, j) :
2   tmp = a [i]
3   a[i] = a[j]
4   a[j] = tmp

```

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

```

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

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

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.

□

### Exercice 9 (Matrices, \*\*)

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

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

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, \*\*\*)

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, \*\*)**

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.

```
1 while (expression booléenne) :  
2     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 ».

```
1 a = 50  
2 while (a > 0) :  
3     print ("Hello")  
4     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!<sup>1</sup>
- Par exemple, la boucle suivante ne termine jamais et l'instruction « `print ("Bonjour")` » n'est donc jamais exécutée :

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

### Exercice 1 (Première boucle, ☆)

Quelle est la valeur de la variable `r` à la fin de la suite des instructions suivantes ?

```
1 n = 49  
2 r = 0
```



```
3 while (r * r < n) :
4     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”.

```
1 a = 0
2 for i in range (0, 32, 1) :
3     a = a + i
4 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 ?

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

```
1 n = 2
2 while (n > 0) :
3     n = n * 2
4     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 :

```
1 b = False
```

ou bien encore comme cela :

```
1 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 :

```
1 b1 = False #b1 vaut False.
2 b2 = (3 + 2 < 6) # b2 vaut True.
3 b1 = not (b2) # b1 vaut la negation de True, donc False.
4 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 ?

```
1 x = 3 + 5
2 b = (x == 5 + 3)
3 d = (x > x + 1)
4 e = b or d
5 f = b and d
6 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 :

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

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

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

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

```

4 while (i < len (l) and not (found)) :
5     if (l[i] == 3) :
6         found = True
7         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.

```

1 n = 10
2 while (n > 0) :
3     print (n, end=" ")
4     print (" kilometre a pied ca use les souliers")
5     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(l, 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, ★)

Combien y a-t-il d'itérations dans la boucle suivante :

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

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

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

□

### Exercice 17 (Logarithme (itéré), ★★)

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, ★★)

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, ★★★)

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  $a^n \geq x$ . □