TP 1 : Boucles et listes

Objectifs du TP. Écrire correctement des boucles `for` et `while`, appendre à manipuler des listes. Les élèves qui savent écrire des fonctions peuvent le faire !

1 Rappels du TP précédent

- Sous Pyzo, vous pouvez utiliser les raccourcis suivants : F5 pour exécuter tout le script, CTRL + ENTRÉE pour exécuter seulement une portion délimitée par des lignes d'au moind deux #, CTRL + R pour commenter une portion de code sélectionnée à la souris, et CTRL + T pour décommenter.
- Pour $m < n$ deux entiers, `range(m,n)` produit un *itérable*, fournissant la suite des entiers de m à $n-1$ (on peut écrire `range(n)` à la place de `range(0,n)`). On fait parcourir à une variable `i` ces entiers-là à l'aide d'une boucle `for` ayant la forme suivante :

```
for i in range(m,n):
    [instructions]
```

```
s=0
for i in range(1,10):
    s=s+i #s contiendra la somme des entiers de 1 à 9
```

Les instructions indentées étant répétées pour tout i entre m et $n-1$. On peut de plus spécifier un pas p : pour $p > 0$, `range(m,n,p)` permet d'obtenir les entiers $m, m+p, m+2p...$ strictement inférieurs à n. Pour $p < 0$, `range(m,n,p)` fournit les entiers $m, m-p, m-2p...$ strictement supérieurs à n. Par exemple `range(m,-1,-1)` fournit les entiers de m à 0.

- On rappelle le fonctionnement d'une boucle `while` : tant qu'une certaine condition est vérifiée, on réalise les instructions de la boucle. Plus précisément, à chaque fois que l'expression booléenne qui suit le `while` s'évalue en `True`, on fait un tour de boucle et on réévalue la condition. Dès qu'elle s'évalue en `False`, on passe aux instructions situées après la boucle.

```
while expression:
    [instructions]
```

```
a=123
while a>0:
    print(a) #on affiche à l'écran les quotients dans les
    a=a//2   #divisions successives de 123 par 2
```

Exercice 0. *Une boucle for et une boucle while.* ***1.*** Calculer $\sum_{k=0}^{100} \frac{(-1)^k}{k!}$ à l'aide d'une boucle `for`. Vérifier que cette somme vaut environ $1/\mathrm{e}$. Dans le module `math` se trouvent la fonction factorielle (`factorial`) et la constante `e`.

```
from math import *
print(1/e) #a comparer avec ce que vous trouvez
```

2. Recopier et compléter le code suivant pour calculer le nombre de chiffres de N dans la base b.

```
N= ... #a choisir
b= ... #a choisir
c= ...
while N>0:
    N=N//b
    ...
print("le nombre de chiffres de N dans la base b est", c)
```

Vérifier par exemple que 4614 a 4 chiffres en base 10 (ça se voit !) et 13 en binaire ($4614 = \overline{1001000000110}^2$).

2 Les listes

Une liste est la donnée d'une séquence finie d'éléments, possiblement de types différents. Voici un exemple de liste : `[True, 4, 5, 3.0]`. En pratique, on ne manipulera que des listes homogènes, c'est-à-dire constituées d'éléments de même type. Lisez ou relisez rapidement ce qui suit avant de passer aux exercices.

2.1 Accès aux éléments d'une liste. Modification.

Accès aux éléments. Pour `L` une liste, sa *longueur* (nombre d'éléments, length en anglais) est accessible avec `len(L)`. En notant n cette longueur, les éléments sont indexés par les entiers de 0 à $n-1$. Exemples :

```
>>> L = [8, 4, 14, 2, 5, 13]
>>> L[2]
14
>>> L[len(L)-1]
13
```

Ainsi, `for i in range(len(L)):` est le début d'une boucle permettant à la variable i de parcourir les indices de tous les éléments de la liste.
Si on demande l'accès à un caractère d'indice négatif i compris entre -1 et $-n$, où n est la longueur de la liste, celui-ci est considéré comme étant $n+i$:

```
>>> L[-1] # très pratique pour accéder au dernier élément !
13
>>> L[-5]
4
```

L'accès à tout autre indice produit une erreur :

```
>>> L[len(L)]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Retenez bien cette erreur, vous l'aurez souvent !

Modification d'un élément. Étant donnée une liste `L`, on peut modifier l'élément d'indice i de `L` en le remplaçant par l'élément de son choix. La syntaxe est la même que pour une affectation.

```
>>> L=[8, 4, 14, 2, 5, 13]
>>> L[2]=999
>>> L
[8, 4, 999, 2, 5, 13]
```

Les règles régissant l'indice i sont les mêmes que précédemment.

2.2 Construction de listes

Possibilités de construction. On peut construire une liste de plusieurs manières.

- Par la donnée explicite des éléments, entre crochets, séparés par des virgules, comme dans `[1, 2, 3]`.
- Par concaténation de listes (à l'aide de `+`) : `[1, 2, 3]+[4, 5, 6]` s'évalue en `[1, 2, 3, 4, 5, 6]`.
- Avec p un entier positif, et `L` une liste, `L*p` a pour résultat la concaténation de `L` avec elle-même, p fois. On l'utilisera souvent pour créer une liste contenant p fois le même élément, à partir d'une liste de taille 1. Par exemple `[0]*12` est une liste de 12 zéros.
- `list(iterable)` permet de fabriquer une liste à partir d'un itérable. Par exemple, `list(range(4))` s'évalue en `[0, 1, 2, 3]` et `list("truc")` en `['t', 'r', 'u', 'c']`.
- Par compréhension, très pratique avec la structure suivante : `L=[f(x) for x in iterable if P(x)]`, où `f(x)` est une expression dépendant (ou non) de `x`, et `P(x)` est une expression booléenne (facultative). Par exemple :
  - `[x*x for x in range(5)]` s'évalue en la liste `[0, 1, 4, 9, 16]`.
  - `[x*x for x in range(5) if x%2==0]` s'évalue en la liste `[0, 4, 16]`.
- Par *slicing* (tranchage), qu'on ne verra pas dans ce TP.
- Par ajouts d'éléments successifs avec `append`, en partant d'une liste vide `[]`, qu'on explique ici.

Rajout d'un élément à une liste : la méthode `append`. Parmi les nombreuses *méthodes* qui existent sur les listes, l'une d'entre elles nous servira très souvent : `append`. Elle permet de rajouter un élément à la fin d'une liste :

```
>>> L=[1, 2, 3]
>>> L.append(8)
>>> L
[1, 2, 3, 8]
```

Il est très fréquent de créer une liste en partant de la liste vide `[]`, qu'on remplit avec une boucle `for` et `append`.

Exercice 1. *Création de listes.*

1. De deux manières différentes, créer une liste `L0` de taille 10 ne contenant que des zéros (on peut le faire avec une seule instruction, ou en utilisant 10 fois `append` à partir d'une liste vide, via une boucle).

2. De même, créer la liste `L100` contenant tous les entiers entre 0 et 99, dans cet ordre, de plusieurs manières :

— avec `list` ;

— avec une boucle `for` et `append`.

3. Créer la liste `L300` contenant tous les entiers de 0 à 99, trois fois : le début est `[0, 0, 0, 1, 1, 1, 2,...]`. On utilisera par exemple deux boucles `for` imbriquées et `append`.

4. Créer une liste qui contient tous les entiers de 0 à 99, puis ceux de 98 à 0 (elle a donc 199 éléments). On utilisera par exemple deux boucles `for` successives (non imbriquées !). `range(98,-1,-1)` fournit les entiers de 98 à 0, dans l'ordre décroissant.

5. Créer par compréhension la liste des puissances de 2, de 2^0 à 2^{15} inclus :

```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768]
```

6. Créer par compréhension la liste des entiers positifs impairs inférieurs ou égaux à 100 divisibles par 5 ou par 7, mais pas par 35.

```
[5, 7, 15, 21, 25, 45, 49, 55, 63, 65, 75, 77, 85, 91, 95]
```

3 Exercices

Exercice 2. *Les deux algorithmes du cours pour passer d'une base à une autre.* On note $\overline{a_{n-1}a_{n-2}\cdots a_0}^{\,b}$ la représentation d'un entier N dans la base b : cette notation s'interprète comme $N = \sum_{k=0}^{n-1} a_k b^k$.

- Écrire en Python l'algorithme du cours, permettant de calculer les chiffres d'un entier N dans la base b par divisions successives. Le résultat doit être la liste $[a_0, a_1, \ldots, a_{n-1}]$ dans cet ordre.

```
N = ... #l'entier à décomposer
b = ... #la base
M = N #pour ne pas modifier N
L = [] #la future liste des chiffres de N dans b
while M>0:
    ... #à compléter
print(L) #pour vérifier
```

  On rappelle que `//` permet d'obtenir le quotient dans une division euclidienne, et `%` le reste. Le principe est d'effectuer des divisions euclidiennes sur M tant qu'il est non nul. Un nouveau chiffre est obtenu avec le reste de la division euclidienne de M par b, il faut ensuite remplacer M par son quotient dans la division par b. Avec $N = 158$ et $b = 7$, vous devez obtenir `[4, 1, 3]`, alors qu'en base 2 vous devez avoir `[0, 1, 1, 1, 1, 0, 0, 1]` : les chiffres sont stockés du poids faible au poids fort.

- Inversement, écrire un code permettant de passer d'une liste $\texttt{L} = [a_0, a_1, \ldots, a_{n-1}]$ représentant un entier dans la base b à son interprétation en base 10, $\sum_{k=0}^{n-1} a_k b^k$. On s'autorise l'utilisation de la puissance `**`. Pour parcourir une liste `L`, utiliser `range(len(L))` et une boucle `for`.

Exercice 3. *Parcours de listes.* On suppose que `L` est affectée, choisissez une liste d'entiers et n'hésitez pas à changer !

1. Écrire une suite d'instructions permettant d'affecter à la variable `somme_liste` la somme des éléments de la liste. *Indication : utiliser une variable initialisée à 0 dans laquelle vous ferez la somme en parcourant la liste.*

2. Écrire une suite d'instructions permettant d'affecter à la variable `max_liste` l'élément maximal se trouvant dans la liste (qu'on suppose non vide). *Indication : utiliser une variable initialisée au premier élément de la liste contenant le maximum « temporaire ».*
3. Écrire une suite d'instructions permettant d'affecter à la variable `indice_max_liste` l'indice de l'élément maximal se trouvant dans la liste (c'est-à-dire l'élément i tel que `L[i]` est maximal. Si l'élément maximal se trouve à plusieurs endroits, `indice_max_liste` devra contenir le premier indice où ce maximum est rencontré). *Indication : utiliser une variable initialisée à 0 contenant l'indice du maximum temporaire.*
4. On dit qu'une liste `L` de longueur n est croissante si pour tout i entre 0 et $n-2$, on a `L[i+1]` $\geq$ `L[i]`. Écrire une suite d'instructions permettant d'affecter à la variable `est_croissante` un booléen (`True` ou `False`), ce booléen valant `True` si et seulement si la liste est croissante. *Indication : dès qu'on trouve un indice i tel que `L[i+1]` < `L[i]`, on sait que la liste n'est pas croissante !*

Testez vos codes, y compris avec d'autres listes !

4 Pour aller plus loin

Exercice 4. *Une reprise du TP0.* Écrire un code permettant de tester si un entier $p \geq 2$ stocké dans la variable du même nom est premier : il suffit de tester la divisibilité de p par tous les entiers entre 2 et $p-1$. Affecter le résultat (`True` ou `False` dans une variable booléenne `b`). Tester avec 65521 (premier) et 96709 (non premier).

Exercice 5. La méthode du crible d'Eratosthène permet de calculer efficacement tous les entiers premiers strictement inférieurs à un entier $N > 2$. Pour ce faire :

- on crée une liste `L` de booléens de taille N, initialement tous `True`.
- on « décoche `L[0]` et `L[1]` (c'est-à-dire qu'on affecte `False` à ces positions dans la liste) : ils ne sont pas premiers.
- on effectue ensuite une boucle sur les indices entre 2 et $N-1$:
  - `L[2]` est `True`, donc 2 est premier. On peut « décocher » tous les multiples non triviaux de 2 (4, 6, 8, etc...) : ils ne sont pas premiers.
  - `L[3]` est `True`, donc 3 est premier. On peut « décocher » tous les multiples non triviaux de 3 (6, 9, 12, etc...) : ils ne sont pas premiers.
  - `L[4]` est `False`, donc 4 n'est pas premier.
  - `L[5]` est `True`, donc 5 est premier. On peut « décocher » tous les multiples non triviaux de 5 (10, 15, 20, etc...) : ils ne sont pas premiers.
  - etc...

À l'aide de cette méthode, construire la liste de tous les entiers premiers inférieurs à 10^3 (vous devez en trouver 168). Combien y a-t-il d'entiers premiers inférieurs à 10^6 ?

Exercice 6. Cet exercice est plus délicat. Une sous-suite croissante d'une liste `L` est une portion d'éléments contigus, qui sont dans l'ordre croissant. Par exemple dans la liste `[0, 2, 8, 1, 5, 7, 9, 11, 2, 4, 6, 0, 15]`, la portion `[0, 2, 8]` est une sous-suite croissante. Écrire un morceau de code permettant de calculer la taille d'une sous-suite croissante maximale (dans cet exemple, il s'agit de `[1, 5, 7, 9, 11]`, de taille 5). *Indication : on peut s'en sortir avec une seule boucle `for` : lorsqu'on examiner un nouvel élément, soit il augmente la suite courante, soit il démarre une nouvelle suite croissante.*

Exercice 7. Celui-là est encore plus délicat. Une sous-séquence croissante d'une liste `L` est une portion d'éléments situés dans cet ordre dans `L`, qui sont croissants. Par exemple dans la liste `[0, 2, 8, 1, 5, 7, 9, 11, 2, 4, 6, 0, 15]`, la portion `[0, 2, 8]` est une sous-suite croissante, et donc une sous-séquence croissante. `[0, 1, 5, 7, 9, 11, 15]` est une sous-séquence croissante de taille 7 (c'est le maximum). Écrire un morceau de code permettant de calculer la taille d'une sous-séquence croissante maximale. *Indication : calculer pour chaque indice `i` la longueur maximale d'une sous-séquence croissante terminant par `L[i]`.*