TP 4 Python – Listes et boucles bornées

1. Bases sur les listes en Python

Définition

En Python, une **liste** (type `list`) est une collection finie et ordonnée d'objets.
Une liste est définie par une séquence de valeurs listées entre crochet et séparées par des virgules.
Les éléments d'une liste peuvent être de types différents !

Exemple

- `[5, -7, "serpent", 2.5, True]` est une liste composée de 5 objets de types variés.
- `[0,1,2,3,4,5,6,7,8]` est la liste des 9 premiers entiers naturels
- `[]` est la liste vide.
- L'instruction `liste_saisons = ["printemps", "été", "automne", "hiver"]` permet de créer une variable `liste_saisons` qui contient la liste des noms des 4 saisons.

On peut accéder aux éléments d'une liste par leur position dans la liste (l'**indice**). Ainsi, l'instruction `ma_liste[indice]` renvoie la valeur située à l'indice `indice` dans la liste appelée `ma_liste`.
⚠ La numérotation **commence à 0**.

Exemple

```
>>> liste_saisons[0]
'printemps'
>>> liste_saisons[2]
'automne'
>>> ma_liste = [5, -7, "serpent", 2.5, True]
```

1. Quel est le type de `ma_liste[3]` ?
2. Que renvoie l'instruction `ma_liste[5]` ?

Exercice 1.
1. Créer une liste `jours_par_mois` qui contient le nombre de jour de chaque mois de l'année, en supposant que l'année est non bissextile. La liste doit contenir 12 éléments !
2. Donner tous les indices correspondant à la valeur `31` dans la liste `jours_par_mois`.
3. Écrire une fonction `nombre_de_jours(m)` qui prend en paramètre le numéro d'un mois (janvier ↔ 1, février ↔ 2, ..., décembre ↔ 12) et qui renvoie le nombre de jours de ce mois.

Remarque. La syntaxe précédente permet également de **modifier** la valeur d'indice donné dans une liste. Par exemple :

```
>>> ma_liste[2] = "changement"
>>> ma_liste
[5, -7, "changement", 2.5, True]
>>> ma_liste[1] = ma_liste[0]
>>> ma_liste
[5, 5, "changement", 2.5, True]
```

2. Boucles bornées

Une boucle bornée est une structure algorithmique qui permet de répéter une instruction un certain nombre de fois, ce nombre étant connu à l'avance.
En Python, c'est le mot-clé `for` qui permet d'écrire des boucles bornées.

En Python :

```
for élément in séquence:
    instruction1
instruction2
```

En pseudo-code :

```
Pour chaque élément dans séquence :
      Faire instruction_1
Fin Pour
instruction_2
```

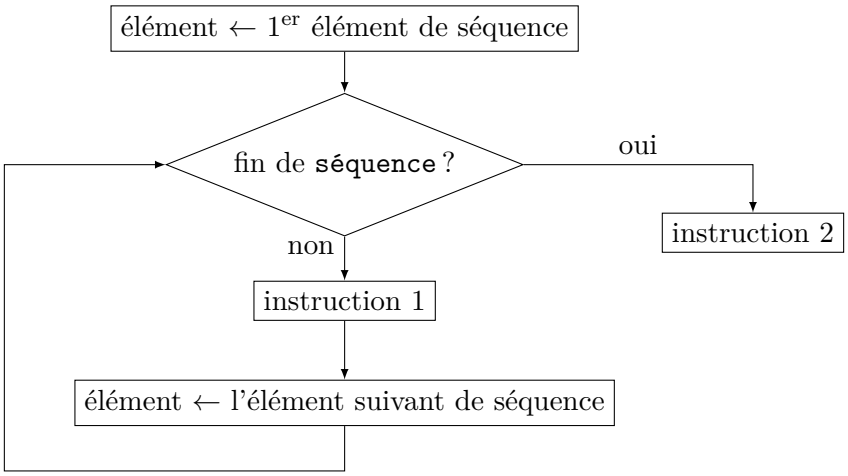
Exemple

```
>>> for i in [0,1,2,3,4]:
...     print(i**2)
...
0
1
4
9
16
```

```
>>> for _ in liste_saisons:
...     print("La vie est belle")
...print("Fin")
...
La vie est belle
La vie est belle
La vie est belle
La vie est belle
Fin
```

Remarque.
- À chaque itération de la boucle, la variable `élément` prend la valeur suivante dans la séquence.
- La boucle se répète donc autant de fois qu'il y a d'éléments dans la séquence.
- La "séquence" peut être **une liste**, mais aussi beaucoup d'autres choses (un ensemble, un n-uplet, une chaîne de caratères). En Python, ces objets s'appellent des *itérables*.

Diagramme de flux :



Exercice 2.
1. Écrire un programme Python qui affiche 7 fois le mot "Bonjour".
2. Écrire un programme qui affiche les 10 premiers entiers naturels impairs, en utilisant seulement la liste `[0,1,2,3,4,5,6,7,8,9]`.
3. Compléter le programme suivant qui détermine le nombre de jours dans l'année à partir de la liste `jours_par_mois` créée à l'exercice 1.

```
total_jours = 0
for n in jours_par_mois:
    total_jours =
print(total_jours)
```

Boucle `for` ou boucle `while` ?

- Boucle `for` si on connaît à l'avance le nombre de répétitions à effectuer, ou simplement si on doit parcourir un « itérable » (liste, ...)
- Boucle `while` si la décision d'arrêter la boucle ne peut s'exprimer que par un test.

3. L'instruction « range »

Soit n un entier. À la place d'écrire à la main la liste `[0,1,2,…,n-1]` (par exemple pour répéter n fois une instruction), on peut utiliser l'instruction `range(n)`, qui crée la séquence des nombres entre 0 et $n-1$.

Ainsi, au lieu de `for i in [0,1,2,3,4,5,6,7]`, on écrira `for i in range(8)`.

Exercice 3. Refaire les deux premières questions de l'exercice 2, en utilisant des `range` à la place des listes.

Plus précisément :

- `range(n)` contient les entiers de 0 à $n-1$.
- `range(n, m)` contient les entiers de n à $m-1$.
- `range(n, m, p)` contient les entiers, séparés de p, allant de n inclus à m exclu. p est appelé le *pas*.

Exemple

Dans cet exemples, on utilise l'instruction `list()` pour convertir les "range" en véritables listes.

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

```
>>> list(range(3, 10))
[3, 4, 5, 6, 7, 8, 9]
```

```
>>> list(range(0,12,3))
[0,3,6,9]
```

Remarque. Le mot *range* signifie *intervalle* en anglais.

4. Exercices

4.A) Des boucles

Exercice 4. Pour chacun des deux programmes ci-dessous :

- Prédire ce qui s'affichera lors de l'exécution du code.
- Vérifier votre prédiction en tapant le code dans une console.

```
for k in range(6):
    if k%2 == 0:
        print(k)
```

```
somme = 0
for k in range(1, 5):
    somme = somme + k
```

Exercice 5. En vous inspirant du second programme de l'exercice précédent, écrire un script Python qui calcule la somme suivante : $1^2 + 2^2 + \ldots + 100^2$.

Exercice 6. Écrire une fonction `factorielle(n)` qui renvoie le nombre $n! = 1 \times 2 \times \ldots \times n$.

Exercice 7. À l'aide d'un programme Python, calculer 3^{42} sans utiliser l'exponentiation `**`.

Exercice 8. Calculer la somme de tous les entiers naturels inférieurs à 1000 qui ne sont divisibles ni par 3 ni par 5.

Exercice 9. On définit une suite $(u_n)_{n\in\mathbb{N}}$ par $u_0 = 150$ et, pour tout $n \in \mathbb{N}$, $u_{n+1} = \dfrac{u_n}{3} + 70$.

1. Calculer à la main u_1 et u_2.
2. Écrire une fonction de paramètre n renvoyant le terme u_n.
3. Que vaut u_{20} ? Et u_{500} ?
4. À l'aide d'un script Python, déterminer le plus petit rang $n \in \mathbb{N}$ tel que $|u_n - 105| < 10^{-5}$.

Exercice 10. On définit la *suite de Fibonacci* $(F_n)_{n\in\mathbb{N}}$ par $F_0 = 0$, $F_1 = 1$ et, pour tout $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$.

1. Calculer à la main les 10 premiers termes de la suite de Fibonacci.
2. Écrire une fonction de paramètre n renvoyant le terme F_n (et la tester sur les petites valeurs de n).
3. Écrire une fonction de paramètre A, renvoyant le premier rang $n \in \mathbb{N}$ à partir duquel $F_n \geq A$.

4.B) Des listes

On verra qu'il existe des fonctions et méthodes prédéfinies sur les listes, qui permettent d'effectuer les opérations de base. Dans la suite, on demande de ne pas les utiliser, et d'écrire les fonctions à l'aide de boucles `for`.

Exercice 11. Écrire une fonction de paramètre un entier positif n, qui renvoie une liste de taille n remplie de 0. On utilisera l'instruction `list(range(n))` pour initialiser la liste.

Exercice 12. Écrire des fonctions qui prennent en argument une liste de nombres, et qui renvoient, respectivement :

- La longueur de la liste.
- La somme des valeurs de la liste.
- La dernière valeur de la liste.
- Le maximum de la liste.

Exercice 13. Écrire une fonction qui prend en paramètre une liste L et une valeur a, et qui renvoie `True` si la valeur a est présente dans la liste L, et `False` sinon.

Exercice 14. Écrire une fonction qui prend en paramètre une liste L et un indice i, et qui renvoie une liste contenant les valeurs de L d'indices i, $i+1$ et $i+2$.

Exercice 15. Écrire une fonction qui prend en paramètre deux listes et qui les **concatène**, c'est-à-dire qu'elle renvoie une liste contenant les valeurs de la première suivies des valeurs de la seconde.