Les listes et les dates en Python

Romain Tavenard

Le but de cette séance est de commencer à manipuler d'une part les listes et d'autre part les dates en Python. Il est fortement conseillé de tester les éléments de syntaxe présentés dans le cours avant de vous lancer dans ce TD. Comme d'habitude, un rappel : la documentation Python est de très bonne qualité, utilisez-la :

— pour les listes : https://docs.python.org/3/tutorial/datastructures.html ;
— pour les dates : https://docs.python.org/3.5/library/datetime.html.

1 Travail à préparer chez vous avant la séance

1. Écrivez, en pseudo-code, un algorithme calculant la différence minimale (en valeur absolue) entre éléments consécutifs d'une liste fournie en entrée. Par exemple, pour la liste `[1, 3, 5, 4]`, la différence retournée devra être `1` (la différence entre les deux derniers éléments de la liste).

2 Les listes

2.1 Création de liste

La première chose à faire pour pouvoir manipuler des listes est d'apprendre comment les définir en Python. Il existe pour cela plusieurs syntaxes possibles (nous ne les verrons pas toutes aujourd'hui et reviendrons sur ce point lors d'un prochain TD). Le cas le plus simple est celui pour lequel la liste que l'on souhaite créer est de taille suffisamment réduite pour pouvoir l'écrire de manière explicite :

```
ma_liste = [1, 7, 9, 12]
```

Toutefois, dans certains cas, on souhaitera générer des listes très longues, par exemple, la liste des entiers de 0 à $n-1$ (avec n possiblement grand):

```
ma_liste = range(n)  # Entiers de 0 à n-1
ma_liste = range(n0, n)  # Entiers de n0 à n-1
```

2. Affichez la liste obtenue : que remarquez-vous ?

En fait, je vous ai menti, la fonction `range` ne retourne pas une liste mais un itérateur (`iterator`). Un itérateur est un flux de valeurs pour lequel la principale opération disponible est de demander la valeur suivante. Ainsi, si l'on souhaite juste parcourir de manière linéaire l'ensemble des entiers entre 0 et $n-1$, il n'y aura pas de différence entre l'utilisation d'une liste et celle d'un itérateur : la syntaxe Python sera la même et vous aurez même tendance à oublier que vous ne manipulez pas exactement une liste. Si toutefois vous deviez à tout prix manipuler une liste (mais cela est peu probable), il est possible de transformer un itérateur en liste :

```
ma_liste = list(range(n))
```

2.2 Recherche d'élément dans une liste

3. Écrivez une fonction qui prend en entrée une liste et un élément (par exemple un entier) et retourne l'indice de cet élément dans la liste (ou `None` si l'élément n'est pas présent dans la liste) sans faire appel à la méthode `index`.
4. Écrivez une fonction qui prend en entrée une liste et un élément (par exemple un entier) et retourne une version de la liste passée en argument dans laquelle toutes les occurrences de cet élément ont été supprimées.

2.3 Copie de liste

La copie de liste en Python est un exercice délicat. En effet, si l'on crée une première liste `liste1` puis que l'on écrit

```
liste2 = liste1
```

le contenu de `liste1` ne sera pas recopié dans `liste2`, les deux objets ne feront qu'un. Cela signifie notamment que si l'on modifie l'une des deux listes, la modification sera répercutée sur l'autre liste.

5. Pour vous en convaincre, mettez en œuvre ce scénario et affichez le contenu des deux listes après modification de l'une des deux.

Ainsi, si l'on veut copier le contenu d'une liste dans une autre, on devra utiliser une astuce syntaxique. Deux approches sont possibles. La première utilise une liste en compréhension, la deuxième la fonction `list` qui effectue une copie de la liste passée en argument.

6. Mettez en œuvre ces deux approches et vérifiez que le problème observé lors de la manipulation précédente ne se pose plus.

3 Les dates

Le type date (plus précisément le type `datetime` qui permet de représenter conjointement une date et une heure) est défini dans le module `datetime`. Commencez donc par ajouter l'instruction d'importation de ce module **en début de votre script Python**.

7. Supposons que soit stocké, dans une chaîne de caractères `s`, le contenu suivant : *"24-08, 2017, 16:53"*. Chargez cette date dans une variable `d1` de type `datetime` et affichez le contenu de cette variable.
8. Écrivez une fonction qui prenne en entrée une date et retourne le nombre d'heures écoulées depuis cette date. Combien d'heures se sont écoulées depuis `d1` ?
9. Écrivez une fonction qui prenne en entrée une date `d0` et retourne la date située une semaine après `d0`. À quelle date sera-t-on rendus dans une semaine ?

4 Exercice de synthèse

10. Écrivez une fonction qui prenne en entrée (i) une liste de chaînes de caractères représentant des dates et (ii) une chaîne de caractère définissant le format de date utilisé dans la liste. Votre fonction devra retourner la durée la plus grande (en valeur absolue) entre deux dates consécutives de la liste. Vous pourrez par exemple tester votre fonction avec la liste suivante :

```
l_dates = [
    "16h12, 22/09/2017",
    "9h23, 31/12/1993",
    "12h56, 08/02/2010",
    "0h00, 01/01/1900"
]
```

N.B. : Pour calculer la valeur absolue d'une quantité numérique, on pourra utiliser la fonction `abs` du module `math`.