TD5

February 17, 2020

1 Les listes

Nous avons vu au TD3 que nous pouvions créer facilement des listes en python. Nous allons voir plus en détail ce qu'il est possible de faire avec ces listes.

`list` : *objet qui peut contenir plusieurs objets de différents types*

1.1 Création d'une liste :

1.1.1 Création simple `[element1,element2...]`

Pour fabriquer une liste en python, c'est très simple. Il suffit de mettre les éléments de la liste entre crochets `[]` et de séparer les éléments avec une virgule `,`. Voici un exemple :

```
maliste = [1,"lundi",0.45]
```

- Il est possible de mélanger les types dans une liste, mais attention à ce que vous ferez après. Ici nous avons un `int`, un `str` entre "et un `float`.

1.1.2 Liste vide (`[]`)

On peut aussi créer une liste vide, en n'indiquer aucun élément :

```
maliste = []
```

Ceci peut sembler surprenant et inutile, nous verrons plus loin que si, ça peut être utile.

```
In [2]: maliste=[] # On crée une liste vide
        type(maliste)

Out[2]: list
```

1.2 Accèder à / modifier un élément de la liste

On peut accéder aux éléments d'une liste grâce à son indice, entre crochets [ ] : - le premier élement a l'indice 0 - le second a l'indice 1 - le n^{ime} a l'indice $n-1$

Ex:

```
In [44]: maliste = ['a','b','c','d','e','f','g']
         print(maliste[0])
         print(maliste[3])
```

```
a
d
```

Il est alors possible de modifier un élément de la liste. Pour cela on affecte (=) une nouvelle valeur à l'élement que l'on veut modifier :

```
In [45]: maliste[2]='toto'
         print(maliste)

['a', 'b', 'toto', 'd', 'e', 'f', 'g']
```

Il est également possible d'extraire plusieurs éléments concécutifs. On indique entre `[]` l'index du premier élément et celui auquel on s'arrête (non inclus), en les séparant par `:` (ceci s'appelle techniquement une tranche, ou une slice en anglais)

```
In [47]: maliste[1:3] # on extrait ici les élément d'index 1 et 2. Le 3 est exclu.

Out[47]: ['b', 'toto']
```

1.3 Ajouter un élément dans une liste

1.3.1 Ajout à la fin (`append()`)

Ici nous allons voir quelque chose de nouveau. Nous allons appliquer une fonction qui n'agit que sur un seul type (ici le type `list`). La syntaxe est donc différente de ce que l'on a vu précédemment.

Pour appeler une fonction associée à un type, on indique le nom de la variable suivie d'un `.` puis le nom de la fonction avec ses arguments.

Pour ajouter un élément à la fin d'une liste, on utilise la **fonction `append()` de la classe `list`**. Cette fonction prend en argument la valeur à ajouter :

```
maliste.append("h")
```

Voici un exemple :

```
In [10]: maliste = ['a','b','d','e','f','g']
         maliste.append("h")
         print(maliste)

['a', 'b', 'd', 'e', 'f', 'g', 'h']
```

1.3.2 Insertion d'un élément (`insert()`)

La fonction `insert()` est également **une fonction associée au type `list`**. Elle prend deux arguments : l'indice où l'on va insérer l'élement, et la valeur que l'on souhaite insérer :

```
maliste.insert(2,"c")
```

Regardons ce que cela donne :

```
In [11]: print(maliste)
         maliste.insert(2,"c")
         print(maliste)

['a', 'b', 'd', 'e', 'f', 'g', 'h']
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

Remarque : lorsque l'indice d'insertion vaut n, la méthode va décaler les élements d'indice supérieur à n, pour intercaler la valeur supplémentaire.

1.4 Supprimer un élement (`remove()`)

La fonction `remove()` associée au type `list` permet de supprimer un élément. Cette fonction prend en argument, **non pas l'indice de l'élément à supprimer, mais l'élément lui-même** :

```
maliste.remove("c")
```

Exemple :

```
In [1]: maliste = ['a','b','c','d','c','e']
        print(maliste)
        maliste.remove("c")
        print(maliste)

['a', 'b', 'c', 'd', 'c', 'e']
['a', 'b', 'd', 'c', 'e']
```

On note que s'il y a plusieurs fois le même élément, c'est le premier qui est supprimé.

1.5 Concaténer des listes (+)

Nous l'avons déjà beaucoup pratiqué sur les chaînes de caractères (il se trouve qu'il s'agit d'un type de `list`).

Il est possible de concaténer (de mettre bout à bout) des listes grâce à l'opérateur +:

```
maliste1+maliste2
```

exemple :

```
In [24]: maliste1 = ["a","b","c"]
         maliste2 = ["d","e","f"]
         maliste1 + maliste2

Out[24]: ['a', 'b', 'c', 'd', 'e', 'f']
```

1.6 Nombre d'éléments dans une liste (`len()`)

La fonction `len()` permet de connaître le nombre d'éléments d'une liste :

```
len(maliste)
```

Exemple :

```
In [16]: maliste = ["a","b","c"]
         len(maliste)

Out[16]: 3
```

1.7 Parcourir une liste :

1.7.1 Avec un boucle `while` :

Grâce à la fonction`len()`, nous connaissons le nombre d'éléments de la liste. Il est alors facile d'écrire une boucle `while` pour parcourir notre liste :

```
In [18]: liste = ['a', 'b', 'c', 'd', 'e', 'f']
         i = 0 #initialisation du compteur

         while i < len(liste) :
             print(liste[i])
             i += 1    # ne pas oublier de faire avancer le compteur

a
b
c
d
e
f
```

Cette méthode n'est pas la plus élégante ni la plus naturelle en python : elle nécessite entre autre de créer à la main un compteur. On évitera de s'en servir en python. Cette méthode reste cependant la plus utilisée dans les autres langages.

1.8 Avec une boucle `for` :

Nous avons déjà vu cela dans le TD3 sur les boucles. Nous allons parcourir la liste en prenant chaque élément, l'un après l'autre :

```
In [19]: liste = ['a', 'b', 'c', 'd', 'e', 'f']

         for element in liste :
             print(element)
```

```
a
b
c
d
e
f
```

Ici pas besoin de connaître le nombre d'élément, ni de créer un compteur. C'est bien plus joli. Mais sans compteur, nous ne savons pas à quel élément nous en sommes, ce qui sera gênant dans certaines situations. Il y a donc une dernière solution.

1.9 Avec une boucle `for` et la fonction `enumerate()` :

Il est plus simple de vous présenter un exemple et de le commenter :

```
In [22]: liste = ['a', 'b', 'c', 'd', 'e', 'f']

         for i, element in enumerate(liste) :
             print("L'élément",i,"vaut :",element)
```

```
L'élément 0 vaut : a
L'élément 1 vaut : b
L'élément 2 vaut : c
L'élément 3 vaut : d
L'élément 4 vaut : e
L'élément 5 vaut : f
```

Durant cette boucle `for`, la variable `element` va parcourir tous les élements de la liste pendant que la variable `i` va servir de compteur et prendre ainsi le numéro de l'indice courant.

Il reste encore plusieurs choses à voir sur les listes, mais faisons dès maintenant quelques applications, pour s'exercer.

1.10 Exercice 1 : Echange de valeurs

Ecrire un programme qui échange les valeurs du premier et du dernier élement d'une liste. Ce programme doit fonctionner quelle que soit la liste initiale.

```
In [ ]:
```

1.11 Exercice 1 bis : Recherche de petits nombres

Ecrire un programme qui, dans une liste de nombres, par exemple (1, 13, 33, 2, 4, 40), supprime tous ceux qui sont supérieurs à 10.

1.12 Exercice 2 : Liste symétrique

Ecrire un programme qui vérifie si une liste est symétrique (liste identique à la liste à l'envers)

In [ ]:

1.13 Exercice 3 : Encadrement d'angles [-180,180]

Ecrire un code qui remplace les angles de la liste suivante par leurs équivalents entre $[-180, 180]$.
liste_angle = [1234,17345,-31243,23,245,456,3600]

In [ ]:

1.14 Exercice 4 : Trier une liste

Ecrire une programme qui trie une liste composée de nombres quelconques du plus petit au plus grand nombre *(attention aux petits malins, on vous demande de créer votre propre progamme).*
liste = [435,324,456,56,567,-45,546,0,345,2,-5]

In [ ]:

Reprenons le cours ici.

1.15 Les listes de listes

Si l'on relit la définition d'une liste, *objet qui peut contenir plusieurs objets*, rien ne nous empêche de placer des listes dans des listes. Par exemple :

```
In [18]: maliste = [["Fer",26],["Ag",47],["Ca",20],["Al",13]]
```

Afin de récupérer une sous-liste, on utilise son index :

```
In [20]: print(maliste[0])
         print(maliste[3])
```

```
['Fer', 26]
['Al', 13]
```

Mais il est possible de récupérer directement l'élement d'une sous-liste, en utilisant un premier index pour sélectionner la sous-liste, puis un second pour sélectionner l'élément de la sous-liste :

```
In [23]: print(maliste[0][1])
         print("L'élément",maliste[0][0],"a",maliste[0][1],"protons.")
```

```
26
L'élément Fer a 26 protons.
```

1.16 Les compréhensions de liste :

The list comprehension permet de modifier ou de filtrer une liste très facilement.

1.16.1 Opérations simples :

Imaginons que nous voulons créer une liste en mettant au carré chaque élément d'une liste d'origine. L'idée qui vient tout de suite à l'esprit est de faire une boucle comme ceci (on remarque au passage l'intérêt de créer une liste vide, ici appelée `carre` : ceci permet d'utiliser la fonction `append()` sur l'objet `carre`, qui existe bien)

```
In [4]: liste = [0,1,3,-1,5, 6] # initialisation
        carre = [] #initialisation de la liste contenant les carrés

        for i,elt in enumerate(liste) :
            carre.append(liste[i]**2)
        print(carre)
```

```
[0, 1, 9, 1, 25, 36]
```

```
In [2]: liste = [0,1,3,-1,5, 6] # initialisation
        carre = [] #initialisation de la liste contenant les carrés

        for elt in liste :
            carre.append(elt**2)
        print(carre)
```

```
[0, 1, 9, 1, 25, 36]
```

Avec *python*, il est possible de faire cette même boucle en une ligne de commande, que nous commenterons après :

```
In [1]: liste = [0,1,3,-1,5, 6] # initialisation

        carre = [elt**2 for elt in liste]
        print(carre)
```

```
[0, 1, 9, 1, 25, 36]
```

Parcourons la seconde ligne de droite à gauche :

- la variable `elt` parcourt les élements de la liste, grâce à la commande `for elt in liste`.
- pour chaque élément, on calcule `elt**2`.
- les `[]` indiquent que les résultats précédents vont créer une liste
- liste que l'on affecte à la variable `carre` grâce à l'opérateur =

1.17 Filtres sur une liste

Il est également possible d'ajouter une condition pour ne sélectionner qu'une partie des élements d'une liste.

```
In [62]: nombres = [-11,10,9,-8,12,-4,20]

         nombres_positifs = [elt for elt in nombres if elt > 0]
         print(nombres_positifs)
```

```
[10, 9, 12, 20]
```

- ici `elt` parcourt les élements de la liste `nombres` en ne concidérant que les nombres positifs.
- pour chaque élement retenu, on retourne sa valeur pour former une nouvelle liste `nombres_positifs`.

1.18 Fonctions sur les listes :

1.18.1 Fonctions usuelles

Il existe plusieurs fonctions usuelles bien pratiques à connaître. Vous allez les découvrir au fur et à mesure. En voici quelques unes. Leur nom est assez explicite :

```
In [12]: nombres = [-11,10,9,-8,12,-4,20]

         print(sum(nombres)) # fait la somme des élements d'une liste
         print(max(nombres)) # retourne le maximum d'une liste
         print(min(nombres)) # retourne le minimum d'une liste
         sorted(nombres) # ordonne du plus petit au plus grand les éléments d'une liste
```

```
28
20
-11
```

```
Out[12]: [-11, -8, -4, 9, 10, 12, 20]
```

```
In [10]: mots = ['bonjour','girafe','schtroumpf','tagada', 'ananas', 'zoo']

         print(max(mots)) # retourne le maximum d'une liste
         print(min(mots)) # retourne le minimum d'une liste
         sorted(mots) # ordonne du plus petit au plus grand les éléments d'une liste
```

```
zoo
ananas
```

```
Out[10]: ['ananas', 'bonjour', 'girafe', 'schtroumpf', 'tagada', 'zoo']
```

1.18.2 Fonctions provenant de librairies

Il existe beaucoup de librairies contenant des fonctions s'appliquant aux listes. A vous de les chercher en fonction de vos besoins. En voici une :

```
In [65]: from statistics import mean, median
         median(nombres) # retourne la valeur médiane
         mean(nombres) # retourne la valeur moyenne
```

```
Out[65]: 4
```

1.18.3 Visualiser des données numériques

Nous arrivons ici à un point ultra important pour un scientifique, la visualisation. Ici nous allons voir une première façon d'afficher un graphe représentant des données.

Nous allons utiliser la librairie **matplotlib** pour afficher notre première courbe à partir de deux listes. Importons cette librairie :

```
In [34]: import matplotlib.pyplot as plt # on importe la librairie
```

Si sous linux, cette librairie n'est pas installée, lancer un Terminal puis taper : python3 -m pip install -U matplotlib –user

Imaginons que nous voulons afficher la fonction x^2 entre 0 et 10. Nous allons créer deux listes, une contenant les abscisses, l'autre les ordonnées :

```
In [78]: x = range(0,10)
         y = [i*i for i in x]
```

Remarque : vous auriez peut-être voulu écrire quelque chose du type suivant, et non ça ne marche pas, python ne sait pas ce que signifie le carré d'une liste.

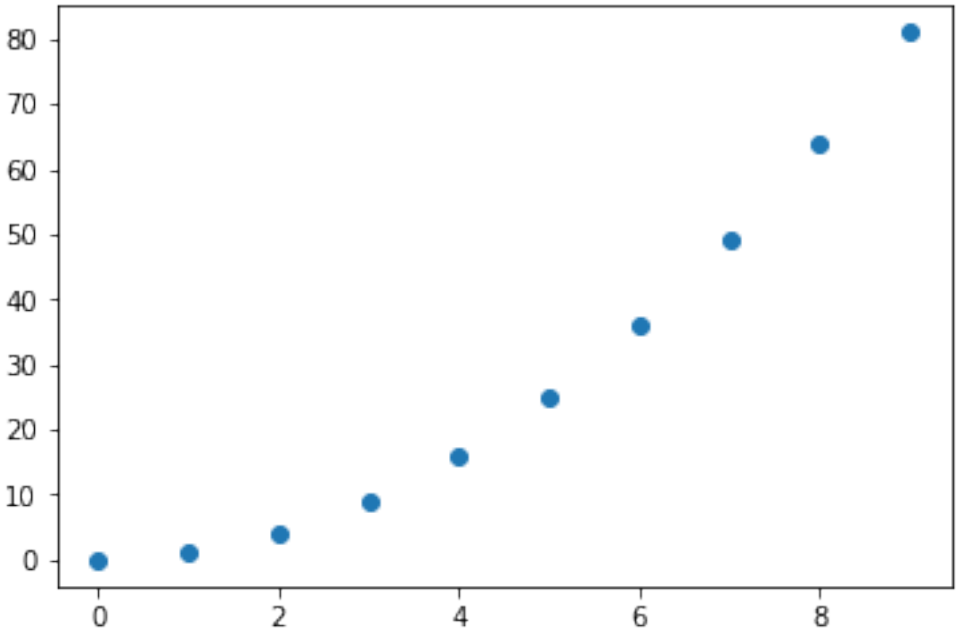
```
In [13]: x = range(0,10)
         y = x**2


         ---------------------------------------------------------------------------

         TypeError                                 Traceback (most recent call last)

         <ipython-input-13-3c3ad855defc> in <module>
           1 x = range(0,10)
     ----> 2 y = x**2


         TypeError: unsupported operand type(s) for ** or pow(): 'range' and 'int'
```

Reprenons nos listes correctement construites.

```
In [14]: x = range(0,10)
         y = [i*i for i in x]
```

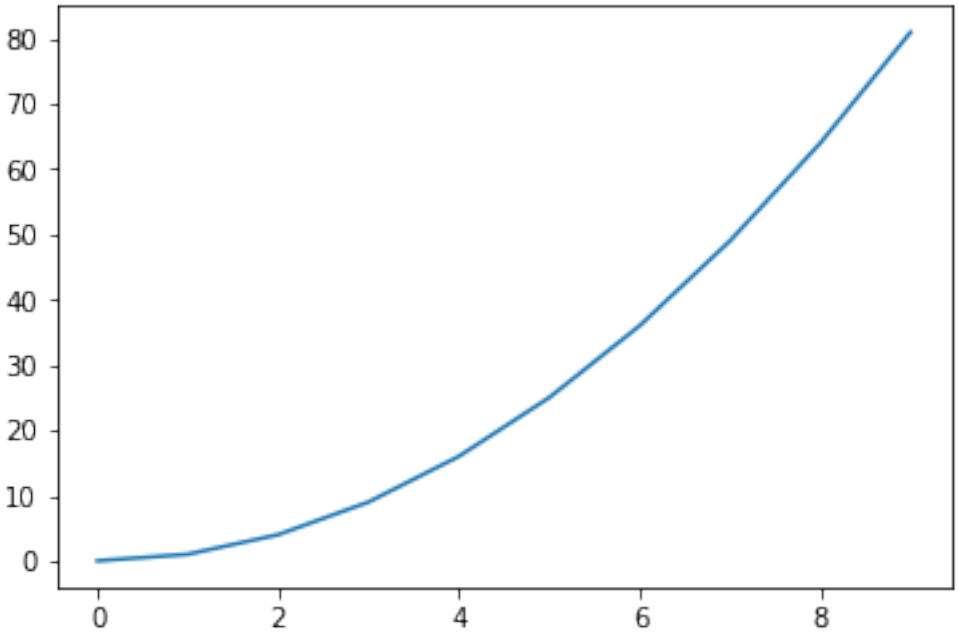
On peut alors afficher y en fonction de x à l'aide d'un nuage de points :

```
In [79]: plt.scatter(x,y) # on génère un graphique point par point
         plt.show() # on affiche le graphique
```



On peut également afficher une courbe en remplaçant `scatter` par `plot`. Les points sont alors reliés par des segments.

```
In [80]: plt.plot(x,y) # on génère une courbe
         plt.show() # on affiche le graphique
```



1.19 Exercice 5 :

Produire la courbe représentative de la fonction $f(x) = \sin(x)$ entre 0 et 10π.

In [ ]:

1.20 Exercice 6 :

Ecrire une fonction qui calcule le produit vectoriel de deux vecteurs. Les paramètres d'entré seront deux `list` (`vec1`,`vec2`) et le résultat sera également une liste. On rappelle que le produit vectoriel est donné par :

In [ ]:

1.21 Exercice 7 : Tableau périodique

Atomes = [["Fer",26],["Ag",47],["Ca",20],["Al",13],["Ne",10],["O",8],["Au",79]]

Ecrire un programme qui trie les élements chimiques précédents par ordre croissant de numéro atomique.

In [ ]:

1.22 Exercice 8 : fonction créneaux

Tracer une fonction $f(x)$ entre 0 et 10 avec des points tous les 0.1, telle que :

- $f(x) = 1$ si la partie entière de x est paire
- $f(x) = 0$ si la partie entière de x est impaire

In [ ]: