



(Re)prise en main du bestiau

Samedi 17 septembre 2016

Buts du TP

- Reprendre en main l'environnement de travail et la syntaxe de base Python.
- Se (re)mettre au point sur des aspects plus précis (listes, numpy, scipy...).

Exercice 1. Créer (au bon endroit) un dossier associé à ce TP. Y placer une copie du fichier `cadeau_tp1.py` récupéré dans le dossier partagé de travail de la classe.

Lancer Spyder/Pyzo/Idle, sauvegarder immédiatement au bon endroit le fichier `.py` ; écrire une commande absurde, de type `print(5*3)` dans l'éditeur, sauvegarder et exécuter. Sous Spyder, changer (via F6) les options d'exécution, pour avoir à chaque exécution une nouvelle console et garder la main dessus.

1 Boucles, tests

Pour calculer une somme, on peut faire une boucle, en faisant évoluer la valeur d'une variable (initialement mise à zéro) représentant les sommes partielles.

Exercice 2. Calculer $\sum_{k=1}^{100} k^2$ à l'aide d'une boucle, puis directement avec `sum`.

On veillera à bien calculer la somme demandée, et non $\sum_{k=1}^{99} k^2 \dots$

Attention, on se souvient qu'en Python 2.*, les divisions d'entiers peuvent poser problème : veut-on une division euclidienne ou bien une division de flottants ? Ou bien va-t-on laisser le hasard décider ?

Exercice 3. Évaluer $S = \sum_{k=930}^{945} \frac{1}{k}$; comparer S à $T = \ln \frac{946}{930}$ puis $U = T + \frac{1}{2} \left(\frac{1}{930} - \frac{1}{946} \right)$.

Exercice 4. Calculer la somme des entiers $k \in \llbracket 100, 1000 \rrbracket$ tels que $k^2 + 2k + 1$ est divisible par 42. On fera le calcul avec une boucle.

Pour compter plutôt que sommer, on utilise un compteur plutôt qu'un sommeur !

Exercice 5. Compter le nombre de $i \in \llbracket 0, 5000 \rrbracket$ tels que $\sum_{j=0}^i j^4$ est divisible par 17.

Bonus : déterminer ce nombre avec un calcul de moins d'une seconde !

2 Fonctions

Exercice 6. Définir en Python la fonction $f : x \mapsto x^2 + 2$. Valider en évaluant $f(100)$ dans la console.

```
>>> f(100)
10002
```

Exercice 7. Définir en Python la fonction $f : x \mapsto \begin{cases} x & \text{si } x \leq 10 \\ 5x - 40 & \text{si } 10 < x \leq 20 \\ -3x + 120 & \text{sinon} \end{cases}$

Représenter son graphe pour vérifier (le code qui suit est dans le fichier donné en cadeau) :

```

import matplotlib.pyplot as plt

les_x = range(-10, 41)
les_y = [f(x) for x in les_x]
plt.plot(les_x, les_y)

plt.grid()
plt.axhline(color='black')
plt.axvline(color='black')
plt.savefig('graphe_de_f.pdf')

```

On trouvera :

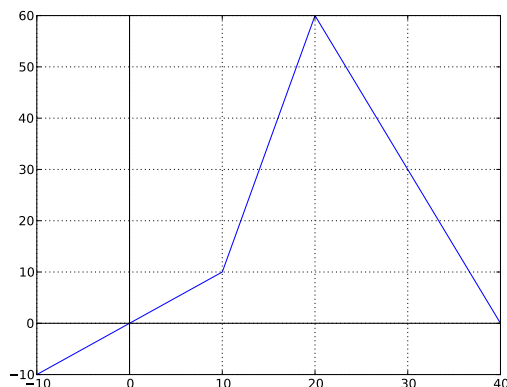


FIGURE 1 – le graphe de f

Pour l'exercice suivant, vous avez le choix entre une version itérative ou récursive

Exercice 8. *Écrire une fonction calculant la factorielle d'un entier donné en paramètre (et qui n'utilise pas la fonction `factorial` de la librairie `math...`).*

```

>>> facto(10)
3628800

```

Maintenant, un classique parmi les classiques, qui sera revisité de nombreuses fois cette année encore.

Exercice 9. *La suite de Fibonacci est définie par ses deux premiers termes et une relation de récurrence d'ordre deux :*

$$f_n = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f_{n-1} + f_{n-2} & \text{sinon} \end{cases}$$

Écrire une fonction prenant en entrée un entier $n \geq 0$ et renvoyant f_n .

Que vaut f_{30} ? Et f_{100} ?

```

>>> fibo(30), fibo(100)
832040, 354224848179261915075

```

L'exercice suivant n'est à traiter que si ce qui précède a été évacué rapidement... Et pourra être repris à la fin.

Exercice 10. *Si $n \in \mathbb{N}^*$, la suite de Syracuse issue de n est définie par son premier terme $u_0 = n$, et la relation de récurrence :*

$$u_{k+1} = \begin{cases} \frac{u_k}{2} & \text{si } u_k \text{ est pair} \\ 3u_k + 1 & \text{sinon} \end{cases}$$

Une conjecture célèbre dit que quel que soit $n \geq 1$, la suite va atteindre 1, et ensuite cycler :

$$1 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow \dots$$

1. Écrire une fonction, disons *syracuse*, prenant en entrée $n \geq 1$ et renvoyant le premier k tel que $u_k = 1$.
Ici encore, on a le choix entre le point de vue récursif et celui itératif.
2. Que vaut *syracuse*(127) ?
3. Quel est le maximum des *syracuse*(n), pour n décrivant $\llbracket 1, 1000 \rrbracket$?

3 Listes

Tout d'abord, créons rapidement des listes simples. Outre `range` (attention, `list(range(...))` sous Python 3), on dispose des compréhensions de listes, d'utilisation typique :

```
[foo(x) for x in whatever if bar(x)]
```

On construit ici la liste constituée des $foo(x)$, pour x décrivant tous les éléments x de l'ensemble *whatever* vérifiant la condition $bar(x)$.

Exercice 11. Créer (en une seule ligne) :

- la liste des entiers positifs impairs majorés par 30 ;
 - la liste constituée des carrés des entiers précédents ;
 - celle constituée des éléments de la liste précédente (la deuxième!) égaux à 1 modulo 12
 - celle constituée de 5 listes elles-mêmes constituées de 4 entiers (il s'agit donc d'un tableau 5×4) telle que $t_4[i][j] = 2^i(2j + 1)$:
- ```
t4 = [[1, 3, 5, 7], [2, 6, 10, 14], ..., [16, 48, 80, 112]]
```

Il faut savoir parcourir une liste pour faire des choses simples...

**Exercice 12.** Écrire des fonctions prenant en entrée une liste d'entiers (ou flottants), et retournant respectivement :

- la somme des éléments de la liste ;
- leur produit ;
- leur maximum ;
- le premier indice correspondant à ce maximum.

Pour les tests, en notant  $t_1$  la première liste de l'exercice précédent :

```
>>> somme(t1), produit(t1), maximum([10, 42, 5, 42, 30]), indice_maximum([10, 42, 5, 42, 30])
(225, 6190283353629375L, 42, 1)
```

Maintenant, un parcours double.

**Exercice 13.** Écrire une fonction calculant le nombre d'inversions d'une liste (disons  $t$ ), c'est-à-dire le nombre de couples  $(i, j)$  tels que  $0 \leq i < j < |t|$  et  $t[j] < t[i]$ .

```
>>> inversions([1, 2, 3, 0])
3
>>> inversions([4, 3, 2, 1])
6
```

Les deux exercices suivants sont à nouveau optionnels (à reporter à la fin ou après le TP) :

**Exercice 14.** Avec des *append* successifs, construire la liste  $t$  de longueur 11 telle que  $t_0 = 1$ , et pour tout  $j \in \llbracket 1, 10 \rrbracket$ ,  $t_j = \sum_{i=0}^{j-1} (i+1)t_{j-1-i}$ .

```
>>> t
[1, 1, 3, 8, 21, 55, 144, 377, 987, 2584, 6765]
```

**Exercice 15.** Écrire une fonction calculant le triangle de Pascal à un ordre donné, en ne faisant que des additions :

```
>>> pascal(4)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

## 4 Calcul numérique

On va travailler ici avec la bibliothèque `numpy.linalg` pour du calcul matriciel. La fonction `array` de `numpy` permet de définir une matrice (sous la forme d'un tableau bidimensionnel). Dans la bibliothèque `numpy.linalg`, on trouvera les fonction `inv` pour inverser, `solve` pour résoudre un système linéaire de la forme  $AX = Y$ , et `det` qui calcule le déterminant d'une matrice.

Dans les deux premiers exercices,  $A$  désigne la matrice de  $\mathcal{M}_5(\mathbb{R})$  de terme général  $M_{i,j} = (i - j)^4$ .

$$M = \begin{pmatrix} 0 & 1 & \cdots & 256 \\ 1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 256 & \cdots & 1 & 0 \end{pmatrix}$$

De telles matrices possèdent une méthode `dot` pour être multipliées : `A.dot(B)` est un `array` correspondant au produit des matrices  $A$  et  $B$  (ici, `dot` est une *méthode* de l'objet  $A$ , c'est-à-dire une fonction « embarquée dans les valises » de  $A$ , un peu comme chaque liste qui se promène avec sa méthode `append` dans ses bagages).

**Exercice 16.** Définir  $A$  sous la forme

```
A = array([[... for j in range(...)] for i in range(...)])
```

Calculer le déterminant de  $A$ , ainsi que  $A^3$  et  $A^{-1}$ .

**Exercice 17.** Résoudre  $AX = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ 5 \end{pmatrix}$ .

On comparera le résultat obtenu avec la fonction `solve` et celui obtenue par inversion de  $A$ .

La bibliothèque `scipy.integrate` fournit `quad` et `odeint` permettant respectivement d'approximer des intégrales et des solutions d'équations différentielles (RTFM!). Pour approcher une solution d'équation de la forme  $f(x) = 0$ , on pourra utiliser la fonction `fsolve` de la librairie `scipy`.

**Exercice 18.** Donner une approximation de  $\int_0^1 \sin(t + \sqrt{t} + 1) dt$ .

**Exercice 19.** Donner une valeur approchée du plus petit réel  $t$  vérifiant

$$\cos(t^2) + \sin^2(1 - t) = 0$$

Donner une valeur approchée de l'unique réel  $x$  vérifiant

$$\int_0^x \sin^2(t + \sqrt{t} + 1) dt = 1$$

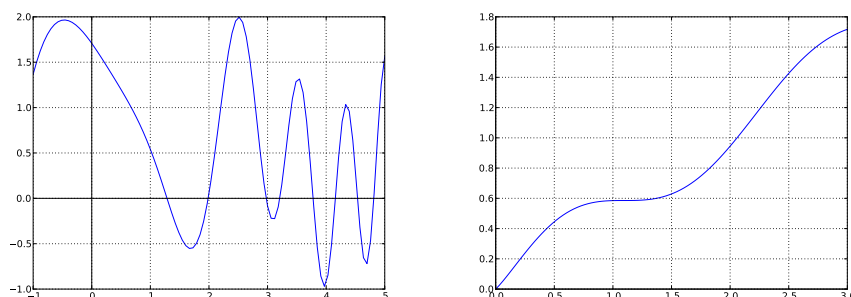


FIGURE 2 – Les graphes de  $f : t \mapsto \cos(t^2) + \sin^2(1 - t)$  et  $h : x \mapsto \int_0^x \sin^2(t + \sqrt{t} + 1) dt$

**Exercice 20.** Donner une approximation de  $Y(2)$ , avec  $Y$  l'unique solution au problème de Cauchy :

$$\begin{cases} y'(t) = \cos(t + y(t)^2) \\ y(0) = 0 \end{cases} .$$

L'exercice suivant est à nouveau optionnel :

**Exercice 21.** Donner une approximation de  $Y(1)$  et  $Y'(1)$ , avec  $Y$  l'unique solution au problème de Cauchy :

$$\begin{cases} y''(t) = \cos(y(t)) - 2t \\ y(0) = 0 \\ y'(0) = 0 \end{cases} .$$

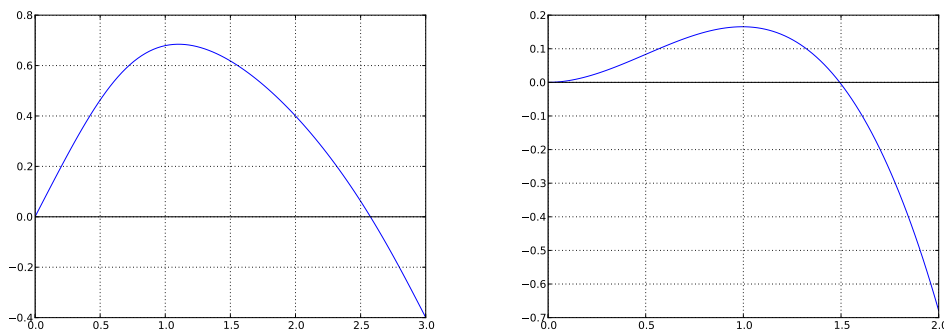


FIGURE 3 – Graphe des solutions approchées des deux derniers exercices